Genetic Programming for Automatically Synthesising Robust Image Descriptors with A Small Number of Instances

by

Harith Al-Sahaf

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2017

Abstract

Image classification is a core task in many applications of computer vision, including object detection and recognition. It aims at analysing the visual content and automatically categorising a set of images into different groups. Performing image classification can largely be affected by the features used to perform this task. Extracting features from images is a challenging task due to the large search space size and practical requirements such as domain knowledge and human intervention. Human intervention is usually needed to identify a good set of keypoints (regions of interest), design a set of features to be extracted from those keypoints such as lines and corners, and develop a way to extract those features. Automating these tasks has great potential to dramatically decrease the time and cost, and may potentially improve the performance of the classification task.

There are two well-recognised approaches in the literature to automate the processes of identifying keypoints and extracting image features. Designing a set of domain-independent features is the first approach, where the focus is on dividing the image into a number of predefined regions and extracting features from those regions. The second approach is synthesising a function or a set of functions to form an image descriptor that aims at automatically detecting a set of keypoints such as lines and corners, and performing feature extraction. Although employing image descriptors is more effective and very popular in the literature, designing those descriptors is a difficult task that in most cases requires domain-expert intervention.

The overall goal of this thesis is to develop a new domain independent Genetic Programming (GP) approach to image classification by utilising GP to evolve programs that are capable of automatically detecting diverse and informative keypoints, designing a set of features, and performing feature extraction using only a small number of training instances to facilitate image classification, and are robust to different image changes such as illumination and rotation. This thesis focuses on incorporating a variety of simple arithmetic operators and first-order

statistics (mid-level features) into the evolutionary process and on representation of GP to evolve programs that are robust to image changes for image classification.

This thesis proposes methods for domain-independent binary classification in images using GP to automatically identify regions within an image that have the potential to improve classification while considering the limitation of having a small training set. Experimental results show that in over 67% of cases the new methods significantly outperform the use of existing hand-crafted features and features automatically detected by other methods.

This thesis proposes the first GP approach for automatically evolving an illumination-invariant dense image descriptor that detects automatically designed keypoints, and performs feature extraction using only a few instances of each class. The experimental results show improvement of 86% on average compared to two GP-based methods, and can significantly outperform domain-expert hand-crafted descriptors in more than 89% of the cases.

This thesis also considers rotation variation of images and proposes a method for automatically evolving rotation-invariant image descriptors through integrating a set of first-order statistics as terminals. Compared to hand-crafted descriptors, the experimental results reveal that the proposed method has significantly better performance in more than 83% of the cases.

This thesis proposes a new GP representation that allows the system to automatically choose the length of the feature vector side-by-side with evolving an image descriptor. Automatically determining the length of the feature vector helps to reduce the number of the parameters to be set. The results show that this method has evolved descriptors with a very small feature vector which yet still significantly outperform the competitive methods in more than 91% of the cases.

This thesis proposes a method for transfer learning by model in GP, where an image descriptor evolved on instances of a related problem (source domain) is applied directly to solve a problem being tackled (target domain). The results show that the new method evolves image descriptors that have better generalisability compared to hand-crafted image descriptors. Those automatically evolved descriptors show positive influence on classifying the target domain datasets in more than 56% of the cases.

Dedication

To my parents, brother and sister, and my beloved wife.

Acknowledgments

I would like to take immense pleasure in thanking those whom without their help this thesis would not have been possible. Above all, I thank God for the blessings, ability, health and wellness He gave me to accomplish this research and to learn a little more about His universe.

First and foremost, I would like to express my deep sense of gratitude to my supervisors **Prof Zhang** and **Dr Johnston** for the support, entrenchment and guidance. Without guidance and constant feedback of Prof Zhang this PhD would not have been achievable. My great appreciations go in particular to him for helping me in various ways to shape my academic and leadership skills. I am indebted to him more than he knows. Needless to mention Dr Johnston who had been a source of inspiration and he provided me with unflinching encouragement which helped me to complete the work of this project. I am indebted to him for the many dedicated hours he has spent reading my articles and this thesis. I offer my utmost gratitude to him for his valuable advice and crucial contribution to this thesis.

Special thanks to all staff members of the School of Engineering and Computer Science. I am indebted to the members of the Evolutionary Computation Research Group for their valuable feedback and suggestions. I gratefully acknowledge Dr Bing Xue for all the help, support, and advice on multiple occasions.

Words are inadequate in expressing my gratitude and thanks for blessings, unending support and encouragement of my beloved parents (Prof Fadhil Al-Sahaf and Hyfaa Abdulabas).

I owe my deepest gratitude to my brother (Ausama) and his beloved family (Rawaa, Yousif and Layan), sister (Esraa) and her family (Omar, Aya, Sali, and Ahmed), and friends for endless love, inspiration, and encouragement.

List of Publications

- Muhammad Iqbal, **Harith Al-Sahaf**, Bing Xue, and Mengjie Zhang, "Genetic programming with transfer learning for texture image classification," *Engineering Applications of Artificial Intelligence.* (Under review).

- **Harith Al-Sahaf**, Mengjie Zhang, Ausama Al-Sahaf, and Mark Johnston, "Keypoints detection and feature extraction: a dynamic genetic programming approach for evolving rotation-invariant texture image descriptors," *IEEE Transactions on Evolutionary Computation.* (Conditionally Accepted).

- Muhammad Iqbal, Bing Xue, **Harith Al-Sahaf**, and Mengjie Zhang, "Cross-domain reuse of extracted knowledge in genetic programming for image classification," *IEEE Transactions on Evolutionary Computation*, (2017). doi:10.1109/TEVC.2017.2657556.

- **Harith Al-Sahaf**, Ausama Al-Sahaf, Bing Xue, Mark Johnston, and Mengjie Zhang, "Automatically evolving rotation-invariant texture image descriptors by genetic programming," *IEEE Transactions on Evolutionary Computation 21*, 1 (2017), 83–101. doi:10.1109/TEVC.2016.2577548.

- **Harith Al-Sahaf**, Mengjie Zhang, and Mark Johnston, "Binary image classification: a genetic programming approach to the problem of limited training instances," *Evolutionary Computation (Journal, MIT Press) 24*, 1 (2016), 143–182.

- **Harith Al-Sahaf**, Mengjie Zhang, and Mark Johnston, "Evolutionary image descriptor: a dynamic genetic programming representation for feature extraction," in *Proceedings of the 2015 Genetic and Evolutionary Computation*

Conference (2015), ACM, pp. 975–982.

- Andrew Lensen, **Harith Al-Sahaf**, Mengie Zhang, and Bing Xue, "Genetic programming for region detection, feature extraction, feature construction and classification in image data," in *Proceedings of the 19th European Conference on Genetic Programming*, vol. 9594 of *Lecture Notes in Computer Science*. Springer, 2016, pp. 51–67.

- Andrew Lensen, **Harith Al-Sahaf**, Mengjie Zhang, and Bing Xue, "A hybrid genetic programming approach to feature detection and image classification," in *Proceedings of the 30th International Conference on Image and Vision Computing New Zealand* (2015), IEEE, pp 1–8.

- **Harith Al-Sahaf**, Mengjie Zhang Mark Johnston, and Brijesh Verma, "Image descriptor: a genetic programming approach to multiclass texture classification," in *Proceedings of the IEEE Congress on Evolutionary Computation* (2015), IEEE, pp. 2460–2467. This work was awarded the overall best paper award.

- Andrew Lensen, **Harith Al-Sahaf**, Mengjie Zhang, Brijesh Verma, "Genetic programming for algae detection in river images," in *Proceedings of the IEEE Congress on Evolutionary Computation* (2015), IEEE, pp. 2468–2475.

- **Harith Al-Sahaf**, Mengjie Zhang, and Mark Johnston, "Genetic programming evolved f from a small number of instances for multiclass texture classification," in *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand* (2014), ACM, pp. 84–89.

- **Harith Al-Sahaf**, Mengjie Zhang, and Mark Johnston, "Genetic programming for multiclass texture classification using a small number of instances," in *Proceedings of the 10th International Conference on Simulated Evolution and Learning*, vol. 8886 of *Lecture Notes in Computer Science*. Springer, 2014, pp. 335–346.

- **Harith Al-Sahaf**, Mengjie Zhang, and Mark Johnston, "Binary image classification using genetic programming based on local binary patterns,"

in *Proceedings of the 28th International Conference on Image and Vision Computing New Zealand* (2013), IEEE, pp. 220–225.

- **Harith Al-Sahaf**, Mengjie Zhang, and Mark Johnston, "A one-shot learning approach to image classification using genetic programming," inn *Proceedings of the 26th Australasian Joint Conference on Artificial Intelligence*, vol. 8272 of *Lecture Notes in Computer Science*. Springer, 2013, pp 110–122.

Contents

List of Tables

List of Figures

1 Introduction

This chapter provides an introduction to this thesis and its motivations, goals, contributions, and organisation. The problem statement is provided first, followed by a discussion on the motivations, challenges, and the main limitations of existing approaches. The research goals and major contributions of this thesis are then discussed. This chapter concludes with a brief discussion on the thesis organisation.

1.1 Problem Statement

Image classification aims at categorising a set of images into different groups based on visual content, and is an essential task in a wide variety of applications in computer vision and pattern recognition, e.g., in medicine (cancer and fracture classification), military (battle field analyses), environment (vegetation classification), and robotics (robot navigation) [33]. Image classification is a challenging task that has attracted increasing interest over the past 50 years [3, 304, 117, 275, 84, 268, 248]. Traditionally, image classification includes different stages, including keypoints (regions of interest) identification and feature extraction operations. These stages aim at transforming the high dimensional low-level feature representation (raw

pixel values) into a reduced dimensional mid- or high-level feature representation [335, 337]. Identifying keypoints and extracting a good set of features are crucial steps that typically require domain-expert intervention. In fact, often a domain-expert in the application being tackled is required to identify the regions of interest and yet another expert in image processing is needed to outline the required steps to perform feature extraction. Those experts are very expensive to employ and, in many cases, are difficult to find. Therefore, two approaches have emerged to address this problem: (1) proposing a set of domain-independent feature extraction methods [337]; and (2) using image descriptors [117, 218, 185]. An image descriptor is a carefully designed function, or set of functions, that aims at automatically performing keypoints detection and/or feature extraction [219, 303]. Although both of these approaches have the potential to achieve a satisfactory level of performance, each approach has its own limitations. The former suffers from being too abstract, which makes it appropriate for some applications but inappropriate for others. For example, a set of features that are designed to tackle face detection tasks may not help perform texture classification tasks. The latter (image descriptors) still requires human intervention to design an image descriptor or to extend an image descriptor to handle different image deformations, e.g., variation in illumination, rotation, or scale.

The process of designing image descriptors is a very challenging task that requires expertise in both image processing and mathematics. The former is needed in order to identify some important keypoints, whilst the latter is needed to designing a mathematical formula to detect those keypoints. Local Binary Patterns (LBP) [219] and its variants, and Scale-invariant Transform Features (SIFT) [185] and its variants, are some typical examples of widely used image descriptors. Recently, with improvement of computing power, some machine learning techniques have received much attention and have been used to improve existing image descriptors [199, 301, 233, 299, 222, 2, 302, 179, 17, 271, 182, 124]. Genetic Programming (GP) [153] has been successfully used as an approach to tackle a variety of image descriptor-related issues [300, 231, 232, 221, 234]. However, these GP-based methods are either designed to improve the performance of an existing image descriptor or to perform keypoints detection of a predefined type, e.g., corners and lines.

The human visual system is capable of learning to recognise complex object categories relying on a single view/snapshot, whereas using only a few training instances to build or train a machine learning model is still a challenging task [256]. The vast majority of machine learning algorithms are designed based on the assumption that an abundant set of training examples is available [134, 252, 77]. However, acquiring a large number of instances is often difficult, expensive, or infeasible. Fingerprint recognition/identification [12, 19], ID-card identification [269], and Biometric passport (e-passport) for identity authentication [325] are some typical examples. Furthermore, most learning algorithms demand human intervention to align each training instance, e.g., each instance contains a single face for face recognition where the eyes, nose and mouth regions appear roughly at the same coordinates, as those algorithms were not designed to handle, or to be robust to, image deformations [86]. As a consequence, this increases the difficulty of the problem (e.g. classification which is concerned with assigning a class label to an image, or recognition which is concerned with localising and assigning a class label for an object in an image) and makes it impractical to achieve satisfactory results/performance.

The overall goal of this thesis is to develop a new domain independent GP approach to image classification by utilising GP to evolve programs that are capable of automatically detecting diverse and informative keypoints, designing a set of features, and performing feature extraction using only a small number of training instances to facilitate image classification, and is robust to different image changes such as illumination and rotation.

1.2 Motivation

Image classification is a cornerstone of computer vision and pattern recognition [38, 9]. An image is made up of a number of pixels, where each pixel has an intensity value. Due to the large number of pixels, even for small images such as 200×200 pixels, machine learning algorithms seldom operate directly on raw pixels. Therefore, numerous methods for keypoints detection and feature extraction are proposed in the literature that aim at transforming the low-level raw pixel representation to a reduced high-level representation. However, there are three

main limitations of this approach [132, 133, 171, 189]. First, it requires human intervention to design those keypoints, how they can be detected, and what features are to be extracted from each keypoint. Second, manual alignment of the images is needed as a large number of domain-independent feature extraction methods assume that each object appears approximately at the same coordinates within an image; those methods are not designed to handle the shift, rotation, and scale variances of objects. Third, the performance of the final results of classification is largely dependent on the goodness (or descriptiveness) of the extracted features. Regardless of how powerful a classification algorithm is, if the extracted features in prior stages are "weak", then the built classifier is more likely to be "weak" as well. Those methods that are designed to operate directly on raw pixel values generally require a large number of training instances in order to build a good model that is capable of performing image classification effectively. However, a sufficiently large number of training instances are not always available or cannot always be acquired. Moreover, the overall complexity (both memory and CPU time) of the learning algorithms will increase substantially when dealing with large images.

The remarkable ability of the human visual system has motivated the proposal of a large number of methods in the literature that aim at replicating the human visual system functionality in machines [264, 265, 44]. However, despite the great progress and achievements in this field, the dream of developing a computer that has the capability of a two-year child to interpret an image remains elusive [293]. One way to mitigate the inability to deal with acquiring a large number of training instances is to adopt a transfer learning approach. **What**, **when**, and **how** to transfer are the main questions that need to be addressed in the transfer learning approach [227]. Moreover, the problem of dealing with a large number of instances still persists.

Genetic Programming (GP) is an evolutionary computation (EC) algorithm that simulates biological evolution and natural selection to automatically evolve a computer program (solution) for a user-defined problem [153, 243]. GP has been used to tackle image-related problems such as object detection [336], classification [275], edge detection [99], segmentation [170], and image descriptors [234].

Similar to non-GP methods, GP methods proposed in the literature to tackle image classification either require human intervention to perform feature extraction

[334, 274], or require a large number of training instances [283, 281]. Moreover, GP has seldom been used to evolve image descriptors [234, 179].

1.2.1 Challenges of Image Classification

Some of the main challenges of image classification can be summarised as follows.

1. **Raw pixels.** Using the raw pixel intensity values to perform classification suffers from two main difficulties: large search space, and pixel values by themselves are meaningless if they are treated in isolation of each other as a pixel value at specific coordinates is likely to be related to its neighbouring pixels (spatial structure). Therefore, human intervention is typically required to specify and design some prominent regions of interest that can be used to identify the instances of the different categories. The challenge is how to design a model that can operate directly on the raw pixel values, detect relations between the pixels of the different regions of an image, and transfer the high-dimension low-level feature space to a reduced-dimension higher-level feature space.

2. **Image variations.** Images from the same object can appear differently when they are captured in an uncontrolled environment. The main variations are *illumination*, *rotation*, *translation*, and *scale*. Other sorts of image variations are *deformation*, and *clutter* (also known as noise). Designing a model that is capable of generating features that are robust to some or all these variations is a very challenging task.

3. **Human intervention.** The intervention of a domain-expert represents an essential key component in performing different operations in image classification. Detecting a good set of keypoints and designing an algorithm to extract features from those keypoints are typical examples where human intervention is needed. Designing a model that can automatically perform such operations (keypoints detection and feature extraction) is not an easy task. Moreover, a model designed to operate on one domain may not generalise well to other domains; which means, different models are needed for different domains. For example, designing a model for texture image

classification may not achieve desirable performance if it is used for face or object image classification.

4. **Flexibility.** Some of the existing image descriptors such as Local Binary Patterns (LBP) [219] and Scale-invariant Feature Transform (SIFT) [185] comprise a number of parameters. Altering those parameters, e.g., the radius (r) and the number of neighbouring pixels (p) in LBP, or extending those existing methods to handle different image variations in many cases requires substantial changes. Developing a parameter-free or self-tuning model has the potential to increase the flexibility of such a model to be used in different situations; however, the task of developing such a model is challenging as the algorithm needs to handle multiple tasks simultaneously.

5. **Number of Instances.** In classification, the performance of a learnt classifier can be significantly affected by the number of instances used during the training phase [134, 252, 77, 315, 255, 256]. Most of the methods developed for image classification will perform poorly when the number of training instances is small [266, 313, 310, 85]. Using a large number of images during the training phase imposes a heavy load on many learning algorithms as typically images comprise a large number of pixels (even for a small image with size 20×20 pixels, there are 400 values). The number of images will be more problematic when the training phase requires iterating over the pixel values of each image over and over such as in the case of using a genetic beam search method. Reducing the number of training instances can significantly speed-up the learning task and reduce the memory required to store those instances. In other words, it will improve the overall system efficiency. However, developing a model using only a few instances during the training phase that can generalise well to unseen data is a very difficult task due to having only a limited source of data to capture informative features and relations between these features.

1.2.2 Why GP?

GP is a widely used global search technique that has been applied to tackle numerous problems in a wide range of applications [154, 243, 146]. Unlike other

techniques under the EC umbrella, GP typically evolves programs using a tree-based representation that gives it some characteristics that make this technique preferred over other EC techniques for a variety of problems such as symbolic regression and classification. The following directly motivate employing GP for image-related tasks.

- **Automatically evolving models.** GP and a few other techniques such as learning classifier systems [129, 322] and artificial immune systems [62, 47], have the ability to automatically build or evolve a model to tackle the problem at hand. For example, in a curve fitting or symbolic regression problem if the data points are known to follow a linear model, then the task is to find the slope and intercept of the line that minimises the error between those data points and the line. However, it will be more difficult to tackle the problem when the model is not known; and here GP has the potential to automatically evolve a model that fits those data points without prior knowledge or human intervention.

- **Dynamic representation.** Unlike many other techniques such as Genetic Algorithms (GAs), Particle Swarm Optimisation (PSO) and Artificial Neural Networks (ANNs), GP representation is not fixed and can be dynamically evolved during the evolutionary process. A tree-based GP evolves solutions that have a tree structure, and different solutions, i.e., individuals, can have different sizes and internal structure (different node combinations); whilst techniques such as GAs and ANNs typically have a fixed model representation that has to be determined prior to the training phase. This is an important attribute as predicting the size and structure of the solution in many problems is not easy. Symbolic regression is a typical example where the aim is to build a model that can best fit the data, which is unlike statistical regression where the assumption is that the data follow a known model and the aim is only to tune the parameters of that model [153, 243].

- **Multiple tasks.** In GP, it is possible to perform multiple tasks simultaneously, where other techniques require a multi-stage approach [331]. A typical example is evolving a classifier and selecting only a subset of the available features [173]. This attribute allows the system to select different terminals

each of which performs a feature extraction operation, and an appropriate set of non-terminal nodes to combine those terminals in a useful manner to evolve a program that performs region detection, feature extraction, feature selection/construction, and classification in images.

- **Flexibility.** GP does not require a specific type of input to operate on and can be designed to operate on diverse types of data. For example, many machine learning algorithms require only numerical data such as ANNs, whilst GP can operate on numerical and categorical data simultaneously. Moreover, GP can evolve linear and non-linear models, and uses domain-independent and domain-specific functions. This property reduces the work required to preprocess the data to fit the learning algorithm requirements.
- **Interpretability.** Unlike many learning algorithms, GP does not evolve a black-box model that can be very difficult to interpret/understand. An individual program evolved by GP can be converted into a mathematical formula or a set of rules depending on the function and terminal sets used. This property facilitates addressing how an evolved solution can solve the problem, which can be accomplished through traversing the solution tree and examining the changes that occur on the instance (or its feature values) being evaluated at each node.
- **Constraints.** Using Strongly-typed GP (STGP) [207], it is possible to introduce a variety of constraints on the different nodes of an evolved program; and therefore, each node can be designed to take a different number and types of inputs and generates different outputs. For example, it is possible to have a combination of mathematical ($+$, $-$, and $\times$) and conditional ($\geq$, $\leq$, and `if-then-else`) operators in a single solution. This property allows some nodes to operate on numerical values and perform typical mathematical operators, whereas other nodes convert the inputs into a binary code.

1.3 Research Goals

The overall goal of this thesis is to develop a new domain independent GP approach to image classification by utilising GP to evolve programs that are capable of

automatically detecting diverse and informative keypoints, designing a set of features, and performing feature extraction using only a small number of training instances to facilitate image classification, and is robust to different image changes such as illumination and rotation. Although only a small number of labelled data is used during the training stage, it is expected that the evolved programs can be effective on unseen data and have the potential to compete with hand-crafted methods proposed in the literature.

1.3.1 Research Questions

The research in this thesis will help to answer the research questions below.

(i) *How can GP be used to evolve models that operate on the raw pixel values and are capable of handling domain independent image classification using only a small number of training instances?*
Image classification is an essential, yet challenging, task in a variety of domains. The difficulty of this task is due to the large size of the search space, diversity of instances of objects from the same class in terms of shape and appearance, clutter or noisy instances, and the difficulty of acquiring a large number of labelled instances in some cases such as in the medical domain. The use of GP techniques for image classification is motivated by the capability to automatically explore a large search space in order to evolve a solution. GP will be used to evolve programs that perform multiple tasks simultaneously such as feature extraction and classification in images.

(ii) *How can GP be used to perform multi-class image classification using only a small number of training instances?*
Multi-class image classification is typically a difficult and more challenging problem than binary classification; it aims at building a model that discriminates between instances of more than two classes. Some methods, such as k-Nearest Neighbour (k-NN), naturally permit handling the multi-class classification task [63]. However, extending tree-based GP to perform multi-class image classification needs to be handled carefully [184, 281, 274, 168]. This becomes even more challenging in the presence of having only a small number of instances. GP will be used to evolve an image descriptor for automatically

designed keypoints, synthesise mathematical formulae to detect those keypoints, and perform feature extraction. Simultaneously, the GP system will be designed to consider the separability of the extracted features to tackle the multi-class classification problem.

(iii) *How can GP be designed to evolve image descriptors that are robust to image deformations?*
Image deformations such as illumination and rotation can largely affect the performance of a model if such deformations were not considered during the feature extraction phase. In images, it is very likely that an object can have different feature values if the image were instead captured in an uncontrolled lighting or rotation environment. Applying image processing techniques to adjust the pixel values can be used to tackle the problem of having different feature values due to illumination; whilst rotation is a more challenging task to handle as altering the rotation of an object within an image requires more advanced techniques. Designing a method that can extract illumination-invariant or rotation-invariant features is a very difficult task. GP will be used to evolve illumination- and rotation-invariant image descriptors that have the potential to generate nearly consistent feature vectors for instances belonging to the same class regardless their illuminations and rotations.

(iv) *How can GP be designed to automatically specify the length of the feature vector simultaneously while evolving an image descriptor?*
The length of a feature vector generated from an image is highly dependent on the specific properties to be measured, and therefore enforced by the image descriptor, e.g., Local Binary Patterns (LBP) [217]. One way to allow the system to automatically specify the length of the feature vector is via adopting the concept of "*Bag-of-Words*" (BoW) [120], which in computer vision is also known as "*Bag-of-Visual-Words*" [272]. A new GP representation will be used in this thesis that allows the system to use nodes of varying number of children directly affects the length of the feature vector in order to evolve an image descriptor and select an appropriate set of features.

(v) *Can one model evolved by GP to tackle an image classification problem be used to tackle another image classification problem in the same or a related*

domain?

Typically, it is assumed that the training and test instances of a problem have the same underlying distributions and the same feature space [227]. However, this assumption may not hold in many real-life applications. Therefore, a built or trained model to tackle one problem is not expected to perform well on a different problem if the underlying distributions of the two problems' instances are different. For example, a model may perform well to classify texture images, but it is not the case when the same images are provided with the presence of noise. Moreover, a model built to tackle one type of problem is not expected to perform well on problems from a different domain (cross-domain). For example, a multilayer preceptron trained to perform hand-written digit classification is not expected to perform hand-written character classification or face recognition as they require different network structures (number of inputs, outputs, and/or hidden nodes) as well as different extracted features for each class of the two problems. *Inductive transfer* [197], also known as *transfer learning*, is an approach that allows the knowledge gained while tackling one problem to be (re)used to tackle a related problem [329]. Investigating how an image descriptor evolved by GP can be directly used to solve problems of related domains is very interesting.

1.3.2 Research Objectives

The following set of research objectives have been defined in order to fulfil the overall goal and research questions.

1. *Develop a new GP approach that combines both sparse and dense image descriptors by detecting sparse regions of an image and uses a dense image descriptor on those detected regions to extract the feature vector.*

   There are some existing works [295, 336, 283, 18, 7] using GP to evolve programs for image classification. A large number of the existing GP and non-GP methods in the literature either require human intervention to perform keypoints detection and feature extraction as preprocessing steps prior to the classifier evolving/training phase, or require a large number of training instances to automatically perform those tasks simultaneously.

This thesis aims to propose new GP representations to tackle the problem of classification in images. Using only a few training instances, the proposed methods are expected to automatically detect some informative regions of the image, extract an appropriate set of features, and perform classification. In order to point out the advantages and disadvantages of the programs evolved by the proposed representations, different aspects such as training and evaluation times, number of instances for training, size, and interpretability of the evolved programs will be investigated. This thesis will also compare the performance of the evolved programs with well-known GP and non-GP methods proposed in the literature.

2. *Develop a new GP approach to evolve an illumination-invariant image descriptor to the multi-class image classification task.*
Generally, increasing the number of classes can impose more difficulties on many classification methods [296]. It is well-known that the larger the number of classes to be handled, the larger the number of instances required in order to estimate the model parameter values [324]. Typically, each GP evolved program produces a single value for each instance; this suits the requirements of binary classification tasks well. Some existing research [336, 283, 337, 338] has investigated different strategies to utilise GP for multi-class classification tasks. However, the majority of those methods do not perform well when there are only a few training instances. Multi-class classification is handled differently in this thesis; instead of focusing on evolving a sophisticated classifier, this thesis will concentrate on using GP to evolve image descriptors by using only a small number of instances. An evolved descriptor simultaneously performs keypoints detection and feature extraction as well as considering the separability of the instances belonging to different classes. The evolved image descriptor is expected to generalise well to the unseen data. This thesis will compare the performance of image descriptors evolved by the proposed method with the use of hand-crafted features proposed in the literature as well as existing well-known and widely used domain-expert designed image descriptors. To carry out this research effectively, the program size, convergence, and interpretability of those evolved image descriptors will be investigated. The proposed method is also aimed to be illumination-invariant;

therefore, the robustness of an evolved image descriptor to the global change of the image pixel values will also be investigated in this thesis.

3. *Extend the GP method for evolving illumination-invariant image descriptors to handle rotation and evolve rotation-invariant image descriptors.*
   The term "*rotation*" will be used in this thesis to refer to the rotating of the **entire frame**, i.e., image, in Euclidean space around the central pixel of the image instead of an object within an image [23]. Handling rotation is a very challenging task that, in most cases, requires domain-expert intervention, and therefore is the main limitation of many existing methods. Furthermore, the majority of existing methods in the literature to tackle the rotation problem are computationally expensive as the procedure of each involves a large number of expensive steps that mostly require frequent rescanning the entire image. The methods proposed in this thesis are expected to evolve illumination-invariant and rotation-invariant image descriptors by combining GP search and a set of first-order statistics. The proposed method does not require human intervention to specify the keypoints to be detected or the features to be extracted from those keypoints. The robustness to rotation of the evolved image descriptors will be investigated and the performance will be compared with the existing rotation-invariant domain-expert designed image descriptors. Different aspects such as convergence, the impact of the sliding window size and the length of the feature vector on the performance, and interpretability will be investigated in order to point out the advantages and disadvantages of the proposed method.

4. *Improve the illumination-invariant and rotation-invariant GP method to automatically specify the length of the feature vector side-by-side with the process of evolving an image descriptor.*
   The number of measured attributes, i.e., extracted features, specifies the length of the feature vector. Some of the features are redundant or irrelevant, which may demand employing feature selection techniques to eliminate/reduce such features. However, applying feature selection techniques can potentially impose a heavy overhead on computations as such techniques mostly require extra calculations to measure the relevance or goodness of the individual

features or subsets of features [307, 181, 327]. Empirically setting a parameter value in a model can be a very expensive task to perform. This task (i.e. finding a good value setting) will become more problematic when the method under consideration has some stochasticity/randomness as the same experiment has to be repeated multiple times. The method proposed in this thesis is expected to reduce the number of parameters needed to evolve an image descriptor by making the length of the feature vector part of the search. The new representations will allow GP to find an appropriate feature vector length during the evolutionary process. This thesis will evaluate the proposed method by conducting a set of experiments and extensively investigate different aspects, e.g., window size, code length, and interpretability, of this method in order to point out its advantages and limitations. The performance of the image descriptors evolved by the method proposed in this thesis will also be compared to a number of domain-expert designed image descriptors.

5. *Utilise and investigate GP to perform transfer learning for image classification tasks.*

   The majority of the currently existing image descriptors were not designed to tackle a specific type of problem, e.g., texture classification, and have been used for a wide variety of applications such as texture image classification [180, 65, 312] and face recognition [5, 328]. This is a very important property as it reduces the need for designing an image descriptor for each problem in every domain. However, such methods detect a specific set of keypoints, such as corners and lines, that domain-experts consider as very reliable attributes to discriminate between different images or objects within an image. In this thesis, an image descriptor evolved by GP on the source domain will be directly used to perform image classification in the target domain. The proposed method is expected to use only a few samples from the source domain, e.g., textures, to evolve an image descriptor that can be reused to detect a set of automatically designed keypoints and extract features to facilitate classifying the instance of the target domains, e.g., textures and non-textures. The performance of those image descriptors will be assessed using a large number of benchmark image classification datasets and compared with

the performance of well-known domain-expert designed image descriptors. The impact of using different window sizes between the source and target domains on the performance will also be investigated.

Completing the first objective will address research question (i), while completing the second, third and fourth objectives will address research questions (ii), (iii) and (iv). Meanwhile, the fifth research objective is investigated to help tackling the last research question.

1.4 Major Contributions

A summary of the major contributions of this thesis is presented in this section, whereas each of the Chapters 3 to 7 is dedicated to discuss each of the contributions presented below.

1. This thesis proposes a method for domain-independent binary classification in images using GP to automatically identify regions within an image and extract features from those regions that have the potential to improve the classification performance while considering the limitation of having a small training set. Experimental results show that the methods proposed in this thesis have achieved significantly better performance than using both hand-crafted features and features automatically detected by other methods. Furthermore, the features generated by the new methods have positive influence on the performance of classification algorithms of different types.

   Parts of this contribution have been published in:

   Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston, "A one-shot learning approach to image classification using genetic programming," in *Proceedings of the 26th Australasian Joint Conference on Artificial Intelligence*, vol. 8272 of *Lecture Notes in Computer Science*. Springer, 2013, pp 110–122.

   Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston, "Binary image classification using genetic programming based on local binary patterns," in *Proceedings of the 28th International Conference on Image and Vision Computing New Zealand* (2013), IEEE, pp. 220–225.

Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston, "Binary image classification: a genetic programming approach to the problem of limited training instances," *Evolutionary Computation (Journal, MIT Press) 24*, 1 (2016), 143–182.

2. This thesis proposes the first GP approach for automatically evolving an illumination-invariant dense image descriptor using only a few instances of each class and is capable of designing keypoints, detecting those keypoints, and performing feature extraction. The method operates directly on the raw pixel values and synthesises a set of mathematical formulae using simple arithmetic operators to form a descriptor. As the proposed method does not require any human intervention to design what keypoints are better to identify the instances of each class, how to detect those keypoints, and what and how features are extracted from those keypoints, this method clearly shows potential promise in computer vision to address different problems. The results of the experiments reveal the potential of this method to significantly outperform domain-expert hand-crafted descriptors and improve the performance of the different classification algorithms.

   Parts of this contribution have been published in:

   Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston, "Genetic programming for multiclass texture classification using a small number of instances," in *Proceedings of the 10th International Conference on Simulated Evolution and Learning*, vol. 8886 of *Lecture Notes in Computer Science*. Springer, 2014, pp. 335–346.

   Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston, "Genetic programming evolved f from a small number of instances for multiclass texture classification," in *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand* (2014), ACM, pp. 84–89.

   Harith Al-Sahaf, Mengjie Zhang Mark Johnston, and Brijesh Verma, "Image descriptor: a genetic programming approach to multiclass texture classification," in *Proceedings of the IEEE Congress on Evolutionary Computation* (2015), IEEE, pp. 2460–2467.

3. This thesis considers rotation variation of images and automatically evolves

rotation-invariant image descriptors in addition to being invariant to illumination through integrating a set of first-order statistics as terminals. Handling rotations represents a more challenging task than illumination and requires major changes to be applied to some of the existing methods in order to achieve a satisfactory level of performance, whereas the proposed method handles everything automatically using only a small set of instances of each class. Experimental results reveal the advantages of the proposed method compared to hand-crafted image descriptors. Furthermore, the descriptors evolved by the proposed method can, to some extent, be interpreted.

Part of this contribution has been published in:

Harith Al-Sahaf, Ausama Al-Sahaf, Bing Xue, Mark Johnston, and Mengjie Zhang, "Automatically evolving rotation-invariant texture image descriptors by genetic programming," *IEEE Transactions on Evolutionary Computation 21*, 1 (2017), 83–101. doi:10.1109/TEVC.2016.2577548.

4. This thesis proposes a dynamic GP representation that allows the system to automatically identify the length of the feature vector side-by-side with evolving an image descriptor. The method reduces the number of parameters that the user needs to set, which can be considered as an advantage over hand-crafted methods. Using different images datasets, the results of the experiments show that this method has achieved significantly better or comparable performance compared to hand-crafted domain-expert designed image descriptors. Moreover, some of the evolved descriptors have a very small feature vector and yet still have significantly outperformed the competitive methods. This is due to the ability of the method to automatically identify a set of features that better suit each problem instead of using generic hand-crafted features.

   Parts of this contribution have been published in:

   Harith Al-Sahaf, Mengjie Zhang, and Mark Johnston, "Evolutionary image descriptor: a dynamic genetic programming representation for feature extraction," in *Proceedings of the 2015 Genetic and Evolutionary Computation Conference* (2015), ACM, pp. 975–982.

   Harith Al-Sahaf, Mengjie Zhang, Ausama Al-Sahaf, and Mark Johnston,

"Keypoints detection and feature extraction: a dynamic genetic programming approach for evolving rotation-invariant texture image descriptors," *IEEE Transactions on Evolutionary Computation*. (Conditionally Accepted).

5. This thesis proposes a methodology for transfer learning in GP, where an image descriptor evolved on instances of a related problem (source domain) is applied directly to a problem being tackled (target domain). The proposed method investigates the generalisability of an evolved image descriptor. This has the potential to not only demonstrate the possibility of automatically evolving image descriptors to directly tackle problems of different types, e.g., texture classification and object classification, but to also show the ability of those descriptors to handle a number of classes and sizes of instances in the target domains different from those used in the source domain. The results show that an image descriptor automatically evolved by the proposed method has significantly outperformed domain-independent hand-crafted image descriptors.

1.5 Organisation of the Thesis

The remainder of this thesis is organised as follows. The literature of related work is reviewed in Chapter 2. Each of the five subsequent chapters, i.e., Chapters 3 to 7, addresses one of the research goals. Chapter 8 concludes this thesis.

Chapter 2 presents a detailed description of the image classification problem and provides an overview of the methodologies previously proposed to tackle this problem. The main concepts of GP are also explained in this chapter. This chapter then gives a review of transfer learning and the different methodologies that are used to accomplish this type of learning.

Chapter 3 proposes new representations for binary classification in images where GP is used to detect some informative regions. Different representations and fitness measures are experimentally assessed and some evolved programs are analysed to show how they solve the problem. This chapter also provides further analysis of the proposed representations.

Chapter 4 presents a new representation to evolve illumination-invariant image

descriptors, and a new fitness measure that considers the between-class and within-class distances to measure the separability of the instances. A deeper analysis of some evolved image descriptors is also provided to shed light on how they work.

Chapter 5 extends the illumination-invariant image descriptor presented in Chapter 4 to tackle the rotation issue/challenge by employing simple first-order statistics. The chapter then investigates an evolved descriptor and the generated feature vectors to show how such a descriptor helps to tackle the problem.

Chapter 6 develops a new representation to allow GP to automatically evolve an image descriptor and simultaneously specify the length of the feature vector. The method extends the method presented in Chapter 5 by introducing a node into the function set that allows having a different number of children. Using the new representation reduces the number of parameters that need to be set by the user. An extensive comparison between the proposed method and a range of widely used image descriptors from the literature is performed.

Chapter 7 investigates a methodology to adopt transfer learning in GP. The ability to use an evolved image descriptor to perform image classification in datasets that were not used for training from the same and different domain is investigated. The evolved descriptors are investigated to show how knowledge can be transferred from one task to another.

Chapter 8 concludes this thesis and summarises the key findings. The research contributions and key points of this thesis are highlighted and discussed. The chapter also suggests and discusses different opportunities for future work.

2

Literature Survey

This chapter serves as a foundation stone for this thesis. This chapter covers essential background and provides definitions of the basic concepts and terminology in computer vision such as image types and representations, image features, and image descriptors. The chapter also provides a brief introduction to the different paradigms of machine learning. Details about evolutionary computation (EC) techniques in general, and particularly genetic programming (GP) are also provided in this chapter. The discussion includes the key concepts of GP such as the program representation, selection methods, and genetic operators. This chapter then reviews typical works and summarises the research topics of using traditional machine learning methods and evolutionary methods for image-related tasks such as keypoints identification, feature extraction, and image classification. The benchmark datasets used throughout this thesis to assess the performance of the different methods are also discussed in this chapter.

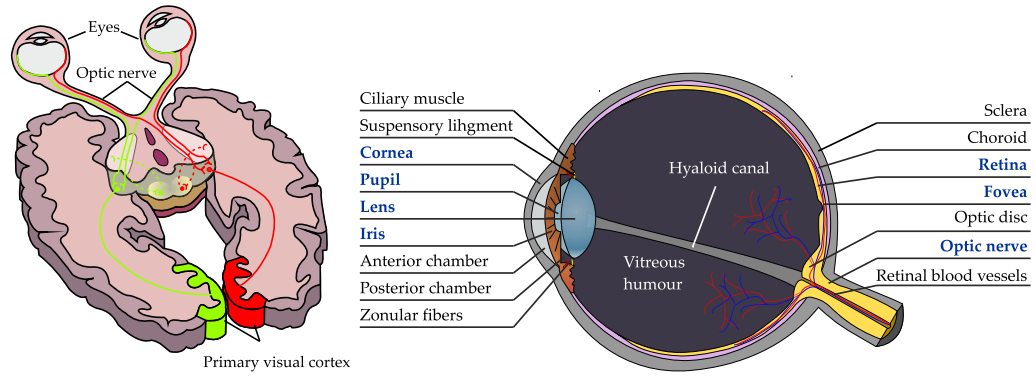
2.1 Basic Concepts

Computer imaging is concerned with acquiring and processing visual information by computers [306]. It can also be defined as the field that is concerned with acquiring, understanding, processing and analysing images by computers [288, 209]. Computer imaging can be categorised into two categories: *image processing* (IP), and *computer vision* (CV). The former aims at generating a modified image that will be interpreted by a human; whereas the latter aims at manipulating the image content and the results will be interpreted by computers [104, 228]. Image enhancement, image restoration and image noise cancellation are some typical examples of IP; whilst image classification and object detection are typical examples of CV. Clearly, images play an essential role and represent the key component of both fields. A *digital image* can be defined as a two dimensional matrix of values where each value reflects the intensity level of the corresponding pixel. Image in this thesis refers to digital images unless otherwise mentioned. The algorithms of computer vision aim at replicating the human vision system via electronically perceiving and analysing the content of an image [285]. As presented in Figure 2.1(a), the human visual system is made of three components: (1) eyes; (2) optic nerve; and (3) primary visual cortex. Generally, human sight can be divided into three operations [309]:

- **Image capturing:** the task of capturing the light emitted from, or reflected on, an object. The adjusted amount of light that falls on the cornea by the iris enters the eye through the pupil. The lens then focuses this light on a specific part of the retina known as fovea. Figure 2.1(b) presents a human eye labelling its main parts.

- **Feature extraction:** is concerned with transferring the captured light/images into a different form that can be processed by the brain. The *cones* and *rods* photoreceptor cells, that are located on the retina, convert the light into electric impulses. The generated impulses are then sent through the optic nerve to the brain.

- **Recognition:** the primary visual cortex processes the received impulses from the optic nerve and communicate with different visual responsive areas



(a) (b)

Figure 2.1: A demonstration of (a) the human visual system components (reproduced from [105]); and (b) the main parts of a human eye (reproduced from [319]).

via multiple pathways in order to interpret the signals (impulses). Relying on previously seen and learnt objects, the brain can categorise the current object or orders the body to take an action. The mystery of understanding the human brain and how it performs the different tasks is still elusive.

In machines, the corresponding components of the human visual system are: (1) cameras; (2) wires; and (3) processing units. Like human eyes, digital cameras capture the light emitted from a light source or reflected on surface, which fall on the lens that focuses it on the sensor. Similar to the retina in a human eye, the sensor consists of a large number of *photosites* —also called Bayer filters— which are sensitive to light. These photosites convert the captured light into electrical impulses each of which represents red, green, or blue colour. The generated impulses are then transmitted through bands of tiny wires to a processing unit. The processing unit interprets the received impulses and projects the image on a screen or passes it to a program to perform an action. Figure 2.2 demonstrates the process of capturing an image in digital cameras.

The introduction of digital cameras and other sophisticated devices (e.g. cellphones and other smart devices) has greatly facilitated the task of gathering data in different domains. In order to gain some useful information, the obtained data require analysis and processing first. In image processing and computer vision, a

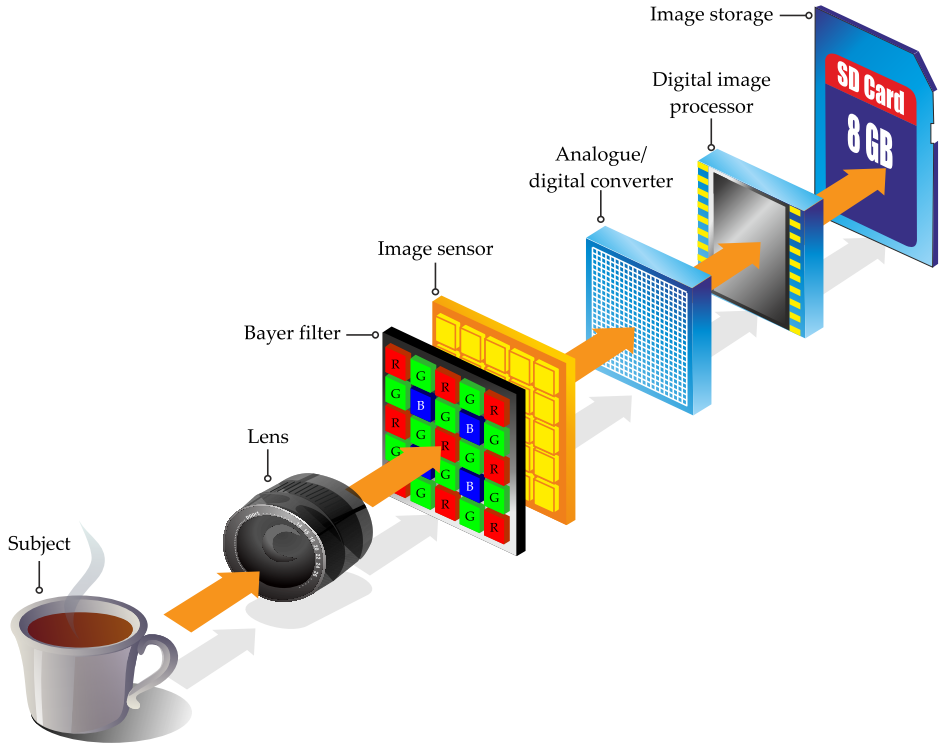


Figure 2.2: Diagram shows the process of image capturing in cameras (reproduced from [69]).

large number of algorithms and methods have been proposed that aim at performing a variety of tasks such as image enhancement [250, 112], recognition [15], image restoration [20, 43], motion analysis [83, 107], and scene reconstruction [278, 151].

Some basic tasks and their definitions are briefly discussed below as they are closely related to computer vision.

2.1.0.1 Object Detection

Object detection is concerned with finding instances of semantic objects in an image against the background [334, 305, 284]. For example, if we have an image that consists of a uniform background with two types of objects, e.g., circles and rectangles, then the task of detection can be locating the coordinates of the central pixel of each object irrespective of the type, i.e., class label, as presented in

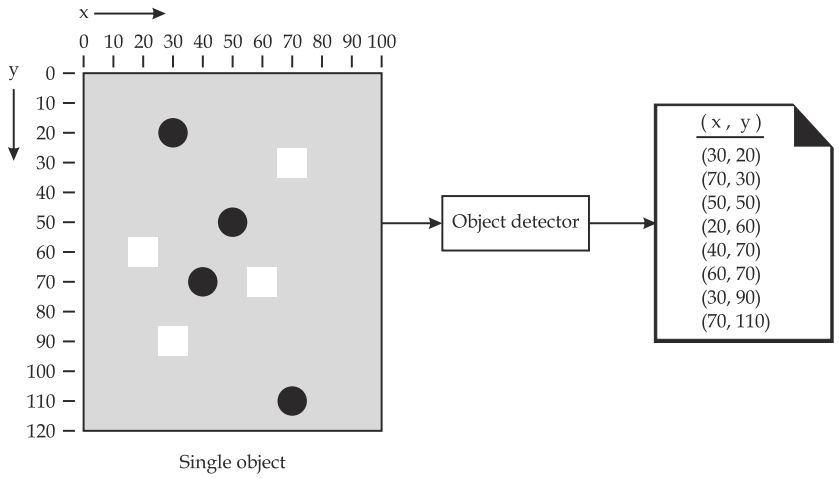


Figure 2.3: Example demonstrating the role of an object detector.

Figure 2.3.

2.1.0.2 Object Classification

Unlike object detection, object classification is the task of assigning a class label to each instance [305, 284]. In other words, each instance is assumed to represent a single object and the model/classifier tries to group similar (based on shared characteristics) instances into one group as demonstrated in Figure 2.4.

2.1.0.3 Object Recognition

In the literature, object recognition and object classification have often been used interchangeably [98]. However, here the term object recognition will be used to refer to the task of finding and identifying objects in an image. In other words, it refers to performing both object detection and classification operations at the same time [15]. Figure 2.5 shows an example of this task.

2.1.1 Keypoints and Image Features

There is still no uniform definition of the term *feature* and it can interpreted differently for different problems, applications, or domains. A feature can be

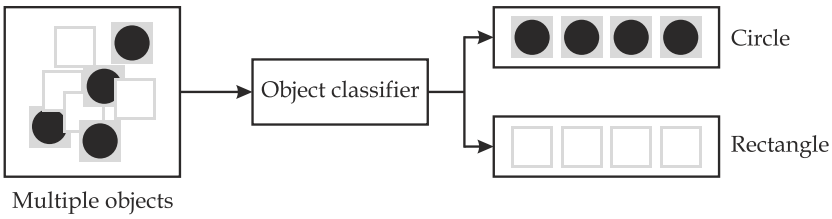


Figure 2.4: Example demonstrating the role of an object classifier.

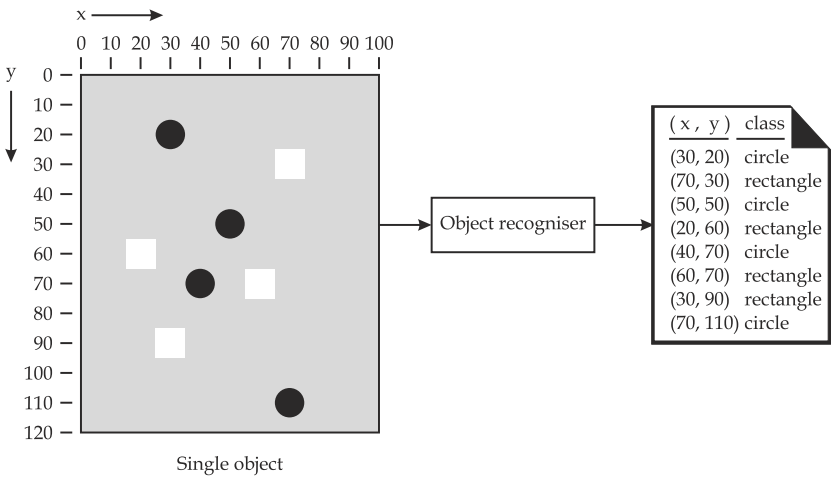


Figure 2.5: Example demonstrating the role of an object recogniser.

defined as a value that describes a measurable attribute [35]. For example, 5.2 centimetres represents the feature of the *length* attribute of an object. In computer vision, features can be calculated from different regions of an image that known as *region-of-interest* (ROI) or keypoints. Lines, corners, homogeneous, and spots are some typical keypoints examples. In machine learning, each instance is represented by a set of features that are calculated manually or automatically, which can be used as input "object" or "example" to perform a task such as classification or recognition. The features in computer vision can be at least categorised into low-level, mid-level, and high-level features. Although there is no standard definition for each of these three levels, the following definitions will be used throughout this thesis to refer to the different types of features. Low-level features are represented by the intensity value of image pixels, where each pixel corresponds to a single feature [215] (a colour pixel might need 3 features). Therefore, the dimensionality

of low-level features is very high. Clearly, low-level features are domain independent. Mid-level features, also called pixel statistics, are generated by constructing a value from a group of pixels such as calculating the average intensity of the eye region in an image of a face [37, 340, 215]. Similar to low-level features, mid-level features are domain independent; however, mid-level features have the potential to reduce the dimensionality of the feature vector. High-level features in computer vision refer to a more descriptive and domain specific features than simple pixel statistics [340, 215]. The magnitude, direction, and scale of the feature, patches for template matching, and shape matching to mention but a few examples. Following are some of the most common operations that can be applied on features.

2.1.1.1 Keypoint Detection

Keypoint detection is the task of exploring the low-level feature space to identify some regions of interest. In other words, a keypoint detector makes a decision at each pixel of the image on whether this pixel is an image region of interest of a predefined type, e.g, corner, or not [293, 95]. Typical image keypoints include edges, corners, blobs or connected regions, and ridges. The problem of detecting image keypoints has been a topic of research for many years and a large number of methods have been proposed. Examples include: the Sobel operator [279], Canny [46], and Harris [119] for edge detection; Laplacian of Gaussian (LoG) [294], Difference of Gaussian (DoG), and features from accelerated segment test (FAST) [258] for corner detection; and maximally stable extremal regions (MSER) [192], Grey-level blobs [174], and principal curvature-based region detector (PCBR) [67] for blob detection.

2.1.1.2 Feature Extraction

Feature extraction is the process of transferring the input data into a reduced representation set. For example, aggregating a group of pixel values and calculating a single value out of it. In images, feature extraction can be applied on the keypoints of an image to construct an informative feature. An example of feature extraction is calculating the mean, the variance, or the histogram of pixel values for a specific region of the image. This task plays an important role in computer vision in which

it can significantly reduce the search space. Feature extraction can also be used to reduce the amount of irrelevant information in the data [89].

2.1.1.3 Feature Construction

Similar to feature extraction, feature construction aim at transferring the input data into a reduced domain; however, the main difference between the two is the input domain. While feature extraction operates on the raw input data, feature construction operates on the feature space, i.e., a set of pre-extracted features, in order to generate a higher or more powerful features to improve the performance of a model [213].

2.1.1.4 Feature Selection

Feature selection methods aim at selecting a subset of features from the set of available features. Mainly, feature selection is concerned with selecting relevant features and neglecting redundant features or less important features [177]. Feature selection must not be confused with feature extraction. The former is only selecting a subset of the original features and does not perform any transformation on the original values, whereas the latter transfers feature values into a reduced representation by creating/generating new features through the use of a function. Feature selection methods can be categorise into three groups [206, 115, 327]: (1) filter; (2) wrapper; and (3) embedded.

- **Filter Approach**
  In the filter approach, the feature selection task is performed as a preprocessing step that aims at filtering nonuseful or redundant data prior to the learning process [150]. The methods of this approach rely on the use of a search algorithm and relevance measure to explore the feature space in order to select a good subset of features. The relevance measure is calculated based on the correlation between the selected features and the desired outputs (i.e. the class label) [148, 206]. Filter-based feature selection methods are typically fast to execute and less computationally intensive compared to methods of other approaches. However, insufficient reliability for classification represents the major drawback of the filter methods [230]. Moreover, many of the good

features may get neglected in some methods due to the relevance measure that ranks each feature based on the correlation with the class label in isolation of other features where those features are not independent [211, 249].

- **Wrapper Approach**
  Unlike the filter approach, the wrapper methods use a searching algorithm in conjunction with a classifier that evaluates the goodness of the selected features. This approach is computationally intensive due to the frequent evaluation of each combination of selected feature [136]. The trained model is often used to measure the goodness of the selected features via adopting a cross-validation scheme in order to define the best combination of the features [200].

- **Embedded Approach**
  The learning method has its own feature selection algorithm that implicitly or explicitly takes place through the process of building the model. In other words, there is no distinction between the feature selection and learning process phases, and they cooperate with each other in an interactive fashion. Decision trees (DT) [246] and random forest (RF) [40] are two examples. The idea of such methods is that at each node the method selects a feature that has a higher split amongst the list of features. GP represents another example of this approach that gradually tries to evolve a good solution and implicitly selects only a subset of features simultaneously [211, 249, 230].

2.1.2 Image Descriptors

In order to construct a feature vector for an image, a set of keypoints, i.e., regions of interest, need to be identified first. An example of keypoint identification is the legs and wheels to discriminate between images where each is either a horse or a vehicle. In texture images, some typical keypoints are lines, corners, circles, and spots. Usually a domain-expert is required to define those informative keypoints.

The second step is to detect those keypoints in an image, i.e., find their corresponding pixel coordinates. Conventionally, this task is performed manually where a domain-expert has to label the coordinates of each keypoint in every image, which is an expensive and time-consuming task. This has motivated researchers

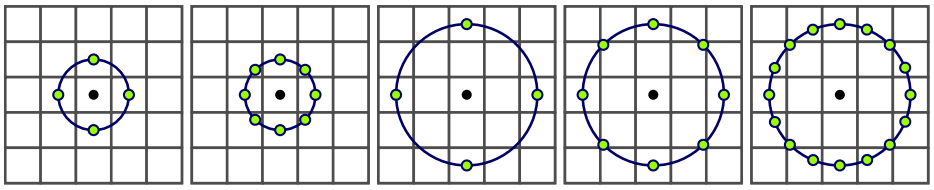
to automate this task and numerous algorithms have been proposed that aim at detecting a specific keypoint, e.g., corners [208, 119, 321], edges [46], and ridges [175], or a set of keypoints [217, 50].

The third step is to extract a feature or a set of features from a detected keypoint such as calculating the average value of pixel intensities, contrast, homogeneity, or correlation.

Automating the second and third steps, and combining them into a single model is known as *image descriptor* in computer vision and pattern recognition [106, 155]. Developing image descriptors has attracted many researchers and received increasing attention over the past few decades [49]. Examples of commonly used image descriptors are grey-level co-occurrence matrix (GLCM) [117], local binary patterns (LBP) [217], scale-invariant feature transform (SIFT) [185, 186], principal components analysis SIFT (PCA-SIFT) [143], speeded-up robust features (SURF) [26], weber local descriptor (WLD) [50], KAZE features [11], and fast retina keypoint (FREAK) [226]. Different descriptors have different properties in terms of being invariant to one or more of the following: scale, rotation, illumination, and affine transformation.

Based on how they operate, image descriptors can be at least categorised into *dense* and *sparse* [51, 155]. Dense descriptors operate in a pixel-by-pixel fashion such as LBP; whereas sparse descriptors consider only some pixels/parts of an image such as SIFT. Although the process of detecting those keypoints and extracting features have been automated, the intervention of a domain-expert is still required to perform the first step, i.e., keypoint identification. Furthermore, how to automatically detect those keypoints and what features can be extracted from the detected keypoints are also typically determined by a domain-expert (e.g. GLCM and SURF).

As has been mentioned earlier, another important issue is that some of these descriptors are not robust to image variants such as illumination, rotation, and scale. Conventional LBP is a typical example that has been extended in order to handle illumination and rotation. Extending a descriptor to be invariant to rotation, for example, may require major changes of the descriptors, and different components or formulae, which increases the complexity, e.g., in terms of the amount of computation required, of the overall system.



(a) (b) (c) (d) (e)

Figure 2.6: Illustration of the LBP parameters (a) $\text{LBP}_{4,1}$, (b) $\text{LBP}_{8,1}$, (c) $\text{LBP}_{4,2}$, (d) $\text{LBP}_{8,2}$, and (e) $\text{LBP}_{16,2}$.

2.1.2.1 Local Binary Pattern (LBP)

In computer vision and pattern recognition, *local binary pattern* (LBP) [217] is one of the most widely used dense descriptors for detecting and extracting image features. LBP aims at detecting image keypoints, and generates a histogram (feature vector) that corresponds to the distribution of those keypoints [212]. Conventional LBP operates by scanning the pixels of the image using a sliding window and generates a binary code based on the differences between the central pixel of the window and its circular equidistant neighbours. The distance is specified by the radius parameter (r), whereas the number of considered neighbours within the window is controlled by the pixel parameter (p) as depicted in Figure 2.6. The formal representation of the LBP operator is defined as follows:

$$\text{LBP}_{p,r} = \sum_{i=0}^{p-1} s(g_i - g_c)2^i, \quad s\left(x\right) = \begin{cases} 0, & x < 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.1}$$

$$g_i = I\left(\mathbf{x}_i, \mathbf{y}_i\right) \tag{2.2}$$

$$\mathbf{x}_i = \mathbf{x}_c + r\cos\left(2\pi i/p\right) \tag{2.3}$$

$$\mathbf{y}_i = \mathbf{y}_c - r\sin\left(2\pi i/p\right) \tag{2.4}$$

where $I\left(\mathbf{x}_i, \mathbf{y}_i\right)$ is the i^{th} pixel at the $(\mathbf{x}_i, \mathbf{y}_i)$ coordinate of image I, the coordinate of the central pixel of the current window is denoted by $I\left(\mathbf{x}_c, \mathbf{y}_c\right)$, and g_c and g_i are, respectively, the value/intensity of the central and i^{th} neighbouring pixels.

The process comprises four steps at each position of the sliding window as presented in Figure 2.7. First, the value of each neighbouring pixel is subtracted from that of the central pixel. Second, each negative value is substituted with a

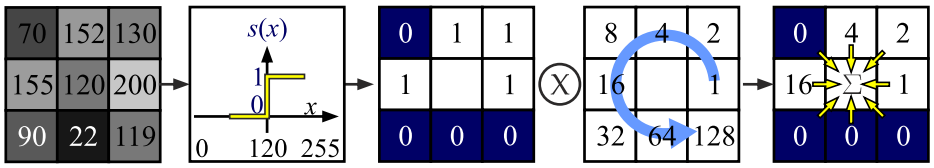


Figure 2.7: Illustration of the LBP main steps.

0, otherwise a 1 is substituted. The combination of these two steps represents a thresholding operator where the value of the central pixel is used as the threshold. Third, the values (0s and 1s) are used to form a binary code. Fourth, the binary code is converted into a decimal value, and the count of the corresponding bin in the histogram is incremented by 1.

Ojala *et al.* [219, 220] have classified LBP codes into *uniform* and *non-uniform*. A code is designated as uniform ($\text{LBP}^{u2}_{p,r}$) if circularly it has no more than two bitwise transitions from 1 to 0 and vice versa as shown in Figure 2.8. The following formula is used to calculate the number of bitwise transitions in a code:

$$\begin{aligned} U\left(\text{LBP}_{p,r}\right) = &\left|s\left(g_{p-1}-g_c\right)-s\left(g_0-g_c\right)\right| + \\ &\sum_{i=1}^{p-1}\left|s\left(g_i-g_c\right)-s\left(g_{i-1}-g_c\right)\right| \end{aligned} \tag{2.5}$$

where a code is said to be uniform if $U(\cdot) \leq 2$. Considering only uniform codes reduces the length of the feature vector from 2^p to $p(p-1)+3$ bins. Formally, the value of the b^{th} bin in a histogram (feature vector) is calculated as:

$$H\left(b\right) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\delta\left(\text{LBP}^{u2}_{p,r}\left(i,j\right),b\right), \quad b \in [0,B] \tag{2.6}$$

$$\delta\left(\alpha,\beta\right) = \begin{cases} 1, & \alpha = \beta \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

where M and N are, respectively, the width and height of the image, B is the maximum number of bins in the histogram, and $\text{LBP}^{u2}_{p,r}\left(i,j\right)$ is the LBP code generated from positioning the sliding window at pixel coordinates (i,j).

To tackle variation due to rotation, a *rotation-invariant* LBP ($\text{LBP}^{ri}_{p,r}$) is also introduced by Ojala *et al.* [220]. The idea is to circularly rotate the code until the

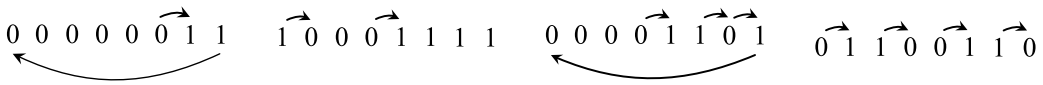


Figure 2.8: Examples of (a) and (b) uniform codes, and (c) and (d) non-uniform codes. The values of the uniformity measure are, correspondingly, $U\,(00000011) = 2$, $U\,(10001111) = 2$, $U\,(00001101) = 4$, and $U\,(11100110) = 4$.

smallest value is found as presented in Equation (2.8).

$$\mathrm{LBP}^{ri}_{p,r} = \min\left(ROR\left(\mathrm{LBP}_{p,r}, x\right)\right), \quad x = 0, \ldots, p-1 \tag{2.8}$$

Here, the $ROR\,(\cdot,\cdot)$ function performs a bitwise right shift operation on the first argument (binary code) equal to the number specified by the second argument.

Then $\mathrm{LBP}^{ri}_{p,r}$ is combined with $\mathrm{LBP}^{u2}_{p,r}$ to generate a potentially more powerful feature vector than that generated by conventional $\mathrm{LBP}_{p,r}$, which is indicated as $\mathrm{LBP}^{riu2}_{p,r}$ and the formal definition is as presented in Equation (2.9).

$$\mathrm{LBP}^{riu2}_{p,r} = \begin{cases} \sum_{i=0}^{p-1} s\left(g_i - g_c\right), & U\left(\mathrm{LBP}_{p,r}\right) \leq 2 \\ p+1, & \text{otherwise} \end{cases} \tag{2.9}$$

As discussed above, the two parameters r (radius) and p (number of considered neighbouring pixels) can significantly affect the design of LBP. Altering the number of considered neighbouring pixels, or the formula to generate the code, requires human intervention that in many cases can be a very difficult task. Moreover, making the descriptor robust to rotation makes the task even more difficult. Hence, the method proposed in this study is designed to address these difficulties by *automatically synthesising* a set of suitable formulae, without any human intervention or background knowledge, to form a rotation-invariant image descriptor.

2.1.2.2 Completed Local Binary Pattern (CLBP)

Typically, only the sign is considered to generate the histogram in LBP as discussed above. Guo *et al.* [110] showed that additional discriminant power is achieved by utilising the *magnitude* (CLBP_M) and the value of the central pixel (CLBP_C)

along with the sign (CLBP_S), and hence, they proposed three operators:

$$\text{CLBP\_S}_{p,r} = \sum_{i=0}^{p-1} s(g_i - g_c)2^i, \quad s(x) = \begin{cases} 0, & x < 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.10}$$

$$\text{CLBP\_M}_{p,r} = \sum_{i=0}^{p-1} s(m_i - g_c)2^i, \quad m_i = |g_i - g_c| \tag{2.11}$$

$$\text{CLBP\_C}_{p,r} = s(g_c - c_I) \tag{2.12}$$

where m_i is the magnitude of the i^{th} pixel, which represents the absolute difference between the intensity of that i^{th} pixel (g_i) and the central pixel intensity (g_c), and c_I is the average intensity, i.e., grey level, of the entire image. Clearly, $\text{CLBP\_S}_{p,r}$ is equivalent to conventional $\text{LBP}_{p,r}$.

2.1.2.3 Local Binary Count (LBC)

Inspired by LBP, Zhao *et al.* [339] proposed the *local binary count* (LBC) descriptor. The main difference between LBP and LBC is that the code generated at each pixel (i.e. window position) is encoded into a decimal value in LBP, whereas merely the number of 1's are counted in LBC. Hence, formally LBC is defined as:

$$\text{LBC}_{p,r} = \sum_{i=0}^{p-1} s(g_i - g_c), \quad s(x) = \begin{cases} 0, & x < 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.13}$$

where, p, r, g_i, g_c, and $s(\cdot)$ have their corresponding meanings to those symbols in $\text{LBP}_{p,r}$.

Another core difference between LBP and LBC is that the local structure information of the pattern is maintained in LBP which is not the case in LBC due to the fact that only the number of counted bits is considered while the position information is discarded.

2.1.2.4 Completed Local Binary Count (CLBC)

A *completed local binary count* (CLBC) is proposed in [339] to mimic CLBP. Hence, the magnitude ($\text{CLBC\_M}_{p,r}$) and centre pixel ($\text{CLBC\_C}_{p,r}$) are also considered in

addition to the sign (CLBC_$S_{p,r}$). Formally, these three operators are defined as:

$$\text{CLBC\_S}_{p,r} = \sum_{i=0}^{p-1} s\left(g_i - g_c\right), \quad s\left(x\right) = \begin{cases} 0, & x < 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.14}$$

$$\text{CLBC\_M}_{p,r} = \sum_{i=0}^{p-1} s\left(m_i - g_c\right), \quad m_i = \left|g_i - g_c\right| \tag{2.15}$$

$$\text{CLBC\_C}_{p,r} = s\left(g_c - c_I\right) \tag{2.16}$$

2.1.2.5 Haralick Texture Features

Haralick *et al.* [117] proposed a set of operators based on the *grey-level co-occurrence matrix* (GLCM) that have been widely adopted by researchers in pattern recognition and computer vision. Each matrix in GLCM has size $L \times L$, where L is the number of grey levels, and is generated by considering the occurrences of the adjacent pixels in predefined offset (τ) and angle (θ). Then a set of features are calculated from those matrices that were designed to detect the structure characterised by the keypoints. Following are some of those broadly used features.

$$\text{Contrast} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \left(i-j\right)^2 f\left(i,j\right) \tag{2.17}$$

$$\text{Homogeneity} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \frac{f\left(i,j\right)}{1+\left|i-j\right|} \tag{2.18}$$

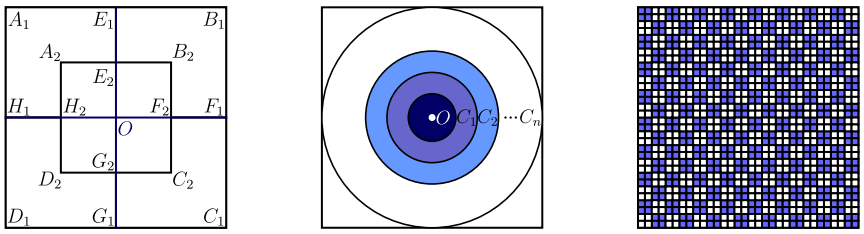
$$\text{Energy} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} f\left(i,j\right)^2 \tag{2.19}$$

$$\text{Dissimilarity} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \left|i-j\right| f\left(i,j\right) \tag{2.20}$$

$$\text{Entropy} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} f\left(i,j\right)\left[-\log_2 f\left(i,j\right)\right] \tag{2.21}$$

$$\text{Correlation} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \frac{\left(i-\mu_i\right)\left(j-\mu_j\right) f\left(i,j\right)}{\sigma_i \sigma_j} \tag{2.22}$$

Here, the function $f\left(\cdot,\cdot\right)$ returns the value of the specified cell from the matrix, and the mean and standard deviation of the i^{th} row are, respectively, denoted as



(a) (b) (c)

Figure 2.9: Three schemes to divide an image in DIF (a) rectilinear, (b) circular, and (c) pixel features [337].

Table 2.1: Pixel statistics of the rectilinear method.

Regions of interest	Features		Regions of interest	Features	
	μ	σ		μ	σ
Square $A_1B_1C_1D_1$	F_1	F_2	Square $A_2B_2C_2D_2$	F_{11}	F_{12}
Quadrant $A_1E_1OH_1$	F_3	F_4	Horizontal line H_1F_1	F_{13}	F_{14}
Quadrant $E_1B_1F_1O$	F_5	F_6	Horizontal line H_2F_2	F_{15}	F_{16}
Quadrant $H_1OG_1D_1$	F_7	F_8	Vertical line E_1G_1	F_{17}	F_{18}
Quadrant $OF_1C_1G_1$	F_9	F_{10}	Vertical line E_2G_2	F_{19}	F_{20}

μ_i and σ_i. Similarly, μ_j and σ_j denote, correspondingly, the mean and standard deviation of the j^{th} column.

2.1.2.6 Domain-Independent Features

In 2003, Zhang *et al.* [337] proposed *domain-independent features* (DIF). The core idea of DIF is to extract a set of first order statistics, e.g., mean and standard deviation, from predefined image regions. Although only three schemes (rectilinear, circular, and pixel) of dividing the image are presented in their work as depicted in Figure 2.9, this method is not limited and numerous schemes can be used. The features of the *rectilinear* method are shown in Table 2.1.

2.1.3 Machine Learning

Machine learning can be considered one of the most rapidly evolved fields in computer science [14]. It aims at designing computer programs that are capable

of automatically extracting useful patterns or knowledge from data without being explicitly programmed in order to solve a problem [13]. Machine learning can also be defined as a subbranch or subfield of the wider artificial intelligence field (AI) [267, 14], which is concerned with designing algorithms that allows computers to learn using example data or past experience [260, 13].

Machine learning methods can be at least categorised into five main categories: (1) supervised learning, (2) unsupervised learning, (3) semi-supervised learning, (4) reinforcement learning, and (5) transfer learning or learning by knowledge transfer [260, 227, 329, 316].

2.1.3.1 Supervised Learning

Methods that use example inputs and are guided/supervised by their corresponding or desired outputs to learn is known as supervised learning. Therefore, a supervised learning method aims at defining a generalised mapping from the inputs to the desired outputs [35]. Classification and regression are typical examples of supervised learning algorithms. Some of the well-recognised supervised learning methods are artificial neural networks, decision trees, naïve bayes, and genetic programming.

2.1.3.2 Unsupervised Learning

Unsupervised learning, on the other hand, is concerned with example inputs where their corresponding actual class labels or desired outputs are not available. Hence, an unsupervised learning method aims at grouping the examples into different groups based on predetermined criteria. Clustering algorithms are probably the most widely recognised examples of unsupervised learning. The k-means clustering [121], and hierarchical clustering [270] methods are typical examples of this category.

2.1.3.3 Semi-supervised Learning

In semi-supervised learning, the aim is to decompose the problem and solve its different components using a mixture of supervised and unsupervised learning algorithms due to having both labelled (the desired output is available) and unlabelled data. Methods of semi-supervised learning can be seen as either transductive or inductive learning. The objective of the former is to correctly infer the class label

of unlabelled data (unsupervised learning), whilst the objective of the latter is to define a mapping function from the inputs to the outputs (supervised learning). A typical example of semi-supervised learning methods is transductive support vector machine [135].

2.1.3.4 Reinforcement Learning

Methods of reinforcement learning are concerned with developing an agent (e.g. computer program) that aims at exploring an environment and takes an appropriate action in which some cumulative reward is to be maximised. Reinforcement learning algorithms are designed based on the principle of reward and punishment, such that an agent will gain more rewards by taking more correct actions, and penalised whenever an incorrect action is taken. A well-known technique of reinforcement learning that a large number of other techniques are based on is the Temporal Difference (TD) learning method [291, 292, 35].

2.1.3.5 Learning by Knowledge Transfer

In psychology, it has been observed that humans have an outstanding ability in categorising a variety of objects without much attention and at very high speed [244, 297, 87]. Humans are relying on previously obtained knowledge to learn a new category or categorise a new object and only simply store incremental new informative knowledge of this object or category [85]. In machine learning, this process is known as knowledge transfer [85]. In contrast to the human ability of learning to classify objects, the majority of existing classification algorithms demand abundant examples in order to learn every single category or group of objects [85]. The number of training instances required to estimate the model's parameters can vary between hundreds and thousands depending on the search space and the number of parameters to be adjusted [266, 313, 310]. Typically, the search space in images is considered to be large even for small images. For example, if we have a grey-scale (with 256 grey levels) image of size 20×20 pixels that means we will have 400 low-level features that give 256^{400} possible combinations. In traditional machine learning, the assumption is that both of the source and target domains or tasks are the same (e.g. drawn from the same distribution).

However in transfer learning, this assumption almost always does not hold as the key point of this approach is to define common knowledge that may help improve the performance of the model.

There are three main questions that need to be addressed [227]:

(i) **What to transfer?** This is concerned with identifying the part of knowledge that can have an influence across domains. The knowledge can be either specific for some domains, or common between a variety of domains, and can be used to improve the performance of the model in the target domain.

(ii) **When to transfer?** In order to avoid the problem of *negative transfer* (i.e. knowledge obtained from the source domain negatively affect the learning on the target domain), it is important to understand the nature of the source and the target domains. In other words, in situations where the source and target domains are unrelated to or independent of each other, then transferring knowledge may have opposite impact on the performance of the model in the target domain.

(iii) **How to transfer?** This is concerned with what methods can be used to transfer the knowledge from the source to the target domain under the assumption that both of the domains are related to each other.

Pan *et al.* [227] have suggested to categorise transfer learning into three groups based on the availability of the labelled data. In the first group, if labelled data are available in the target domain then it is inductive transfer learning. Moreover, if labelled data are not available in the source domain then it is known as self-taught learning; otherwise, it is known as multi-task learning. In the second group, if labelled data are available only in the source domain then it is called transductive transfer learning. The third group is known as unsupervised transfer learning and it handles the situation where labelled data is neither available in the source nor in the target domain.

By limiting the scope of knowledge transfer to the computer vision field, and based on "how to transfer", the different forms of knowledge transfer can be categorised into four approaches [85, 227]:

Table 2.2: Groups of knowledge transfer learning [227]

	Labelled data in source	Labelled data in target	
Inductive transfer learning	Available	Available	Multi-task learning
	Unavailable	Available	Self-taught learning
Transductive transfer learning	Available	Unavailable	Domain adaptation
Unsupervised transfer learning	Unavailable	Unavailable	

- **Knowledge transfer by model parameters.**
  A model trained on a set of instances of one domain can be reused to classify the instances of another domain under the assumption that both of the domains are related. In other words, a large number of instances of the source domain are used to adjust the model's parameters, which then are used on the instances of the target domain to exploit the similarity between instances of the two domains. An example is to train a model to recognise instances of the letter "A" and then use this model to recognise instances of digit "4" [201].

- **Knowledge transfer by sharing features.**
  Another group of algorithms that perform learning by knowledge transfer is through transferring shared parts between different domains or classes. For example, instances of a cow are assumed to have shared features with those of a horse. Therefore, capturing the shared features will help in identifying the instances of the new class. Bart *et al.* [25] proposed an algorithm that aims at extracting patches that maximise the mutual information from instances of the source domain, and then use those extracted patches to learn a new class of objects using a few instances of the target domain.

- **Knowledge transfer by contextual information.**
  Methods of this approach work via obtaining or capturing relations amongst the data of the source domain, and transfer these relations to the target domain. In other words, instead of transferring the parameters (transfer by model parameters) or the features (transfer by sharing features), the learning is achieved though transferring the relationship among the instances or the different regions of those instances. For example, identifying the relations

between strokes to represent a character in order to predict the characters of a different language [160]. Two statistical based learning methods that are dominating this context have been proposed recently: Short-Range to Long-Range (SR2LR) [198] and Deep Transfer via Markov Logic (DTM) [64].

- **Knowledge transfer by instances.**
  Methods of this context use complete instances of the source domain to classify or recognise instances in the target domain [59, 332, 58, 247]. For example, the source domain consists of a set of clear cuts of football pictures, and the task is to locate balls of different types in an image that contains a variety of objects.

2.1.4 Evolutionary Computation

Evolutionary computation (EC) is another fast growing subfield of AI, soft computing more specifically, which is concerned with biologically inspired algorithms [21]. There are two well-recognised main streams in this research area [21]: Evolutionary Algorithms (EAs) [139]; and Swarm Intelligence (SI) [79, 145]. While algorithms of the former group mimic Darwinian principles of natural selection and survival of the fittest, the algorithms of the latter group are inspired by the collective intelligence of a group of agents and mimic the social behaviour of animals such as birds flocking [28, 36]. EC algorithms use a population of candidate solutions to perform a global search rather than considering a single point in the search space. The history of EC techniques is believed to be as recent as the late 1950s [22]. In the following subsection, some of the main algorithms of EC methods are briefly discussed; whereas genetic programming (as it is the main focus of this thesis) will be discussed in detail in the next section.

2.1.4.1 Evolutionary Algorithms

The algorithms under this part of the EC field simulate the biological evolution (Darwinian principles) via performing selection, crossover, mutation and reproduction in order to evolve a population of individuals. The individuals are competing for survival, where good individuals have better chances to survive [61]. The

goodness or fitness of an individual is reflected by its ability to tackle a specific problem. Following are some of the prominent evolutionary algorithms.

- **Genetic Algorithms (GAs) [129]**
  GAs were introduced by John Holland [22, 128] and became popular after his work in the early 1970's [127, 130], and hence, GAs represent one of the earliest techniques that have adopted biological evolution in its process [103]. GAs use a fixed-length bit string representation, called a chromosome, to encode each individual. The chromosome can have other type of values such as real numbers, and integers. The individuals are decoded in order to get the solution for the problem being tackled. Crossover, mutation, and reproduction are the genetic operators in GAs that are used to improve/evolve the individuals. GAs have been widely to tackle optimisation problem [314].

- **Evolutionary Strategies (ES) [253, 30]**
  ES is an evolutionary algorithm that rely on mutation and discrete or intermediate recombination to evolve an individual. The main aim of ES algorithms is to tackle problems in the real-value domain. Unlike GAs, ES uses a deterministic selection mechanism.

- **Evolutionary Programming (EP) [92, 94]**
  EP was introduced by Fogel in 1962 [93, 22] and is aimed at evolving finite-state machines for predicting events based on previous observations. Similar to GAs, the individual representation in EP is fixed and only the parameters are allowed to be adjusted via using self-adaptation. The main operation in EP is mutation. To populate the next generation, the newly generated offspring via mutation are mixed with the individuals of the current generation (parents) and then some of them will be selected to be put into the next generation.

- **Genetic Programming (GP) [153]**
  GP is an evolutionary algorithm that uses a dynamic individual representation to evolve computer programs for a user-defined problem. Similar to GAs, GP also populates the next generation from the individuals of the current generation by exchanging the genetic materials through mutation

and crossover operations. The flexibility of GP individual representation has attracted researchers and facilitated tackling a variety of problems in a wide range of domains.

2.1.4.2 Swarm Intelligence

The algorithms in SI are mainly inspired by the social behaviour of simple agents such as bird flocks, fish schools, ant or bee colonies, and bacterial growth [145, 36, 28]. The main idea is that each agent performs a simple local search in order to find a better solution, and by interacting with other agents the whole group can reach global behaviour. Particle Swarm Optimisation and Ant Colony Optimisation are two typical examples of SI that have been widely employed to solve different optimisation problems.

- **Particle Swarm Optimisation (PSO) [79]**
  In 1995, Eberhart and Kennedy [79] introduced PSO to simulate social behaviours. Like other EC techniques, PSO uses a population (swarm) of candidate solutions (particles) that explore the search space. The algorithm allows each particle to locally explore the nearby areas as well as globally via interacting with other particles. The process starts by randomly generating a set of particles. Each particle is then evaluated based on a predefined criterion, and therefore, assigned a fitness value which reflects its performance. Those particles then adjust their position by considering its own fitness value as well as the global best fitness. In [242, 240], Poli *et al.* provided a comprehensive survey on the theoretical and practical work of using PSO to solve problems.

- **Ant Colony Optimisation (ACO) [71, 73]**
  Inspired by the foraging behaviour of ant colonies, a number of algorithms have been proposed in the early 1990s such as the ant system (AS) algorithm [72, 70]. Dorigo and Caro [71] put those algorithms into a single framework and called it ACO [73]. The main procedure consists of three components: ants activity, pheromone evaporation, and daemon actions. Ant activity concerns with the movement of the ant colony concurrently and asynchronously in the search space of the problem. Each ant in the colony makes a stochastic decision in order to explore the neighbouring states (positions). The

pheromone evaporation operation is concerned with automatically reducing the pheromone trail intensity on the paths, which is used as a mechanism to prevent the algorithm form rapidly converge towards sub-optimal regions. The daemon actions are an optional component that can be used to integrate centralised actions that single ants cannot perform by themselves.

Besides PSO and ACO, there are a number of other algorithms that can be categorised under the SI umbrella, such as artificial bee colony (ABC) [142], grey wolf optimiser (GWO) [204], and bat algorithm and cuckoo search or bat swarm optimisation (BSO) [330].

Apart from the aforementioned EC methods, there are other well-known and widely used method such as differential evolution (DE) [289, 245], learning classifier systems (LCS) [129, 322], evolutionary multi-objective optimization algorithms (EMO) [55], and artificial immune systems (AIS) [47, 62].

2.2 Genetic Programming

Genetic Programming (GP) is an Evolutionary Computation (EC) algorithm that simulates biological evolution and natural selection to automatically explore the search space and evolve a computer program (solution) for a user-defined problem [153]. GP has been utilised to solve complex problems in numerous domains where the search space is extremely large [153, 154, 163]. The ability of GP to effectively explore the search space has attracted researchers to perform different tasks, such as classification [82], regression [113], scheduling [214], optimisation [287], and modelling [78, 114].

2.2.1 GP Overview and Basic Algorithm

Probably the birth of what is known as "genetic programming" goes back to 1964 and Lawrence Fogel [92] who is considered as one of the earliest practitioners of this methodology [21]. Classifying heart disease via evolving tree rules was carried out by Forsyth in 1981 [96]. Then in 1985, Cramer [57] introduced the first statement of modern *tree-based* GP. This work was largely developed and expanded later by Koza [153] who has pioneered GP to tackle diverse optimisation and search

problems. Since then, GP has been widely used to tackle diverse problems in a wide range of real-world applications [21, 163, 243]. GP has received increasing attention due to its properties (more details are discussed in this section) and the promising results of the evolved solutions. The key concepts and the algorithm main steps of GP are explained in the rest of this section. The discussion includes the representation, initialisation, evaluation, selection, and genetic operators.

The concept of "Survival of the Fittest" represents the main scheme of the GP evolutionary process in which a number of individuals (computer programs) are gradually evolved to gain better performance. As presented in Algorithm 1, the process starts by randomly creating a predefined number (δ) of candidate solutions via using different combinations of the elements in the function ($\mathcal{F}$) and terminal ($\mathcal{T}$) sets. Relying on the use of a carefully designed fitness measurement, the fitness (Δ_ξ) of each individual (ξ) is calculated. The fitness value reflects the ability of the individual to tackle the problem. GP uses reproduction, crossover, and mutation operators to produce the individuals of the next generation (Ξ_{i+1}) from those of the current generation (Ξ_i). Individuals with better fitness values are more likely to be selected to participate in populating the next generation. Unlike other EC techniques, e.g., GAs and PSO, GP typically uses a tree representation [153] to evolve programs instead of fixed-length string chromosomes (integers, real numbers, bits, symbols). This allows GP to evolve programs (e.g. trees) of varying sizes and shapes, which means the user does not need to specify the size of the final solution that generally is unknown or hard to approximate.

2.2.2 Program Representation

The tree-based representation is one of the most commonly used individual representations in GP [243]. An individual tree is made up of a root node, a number of internal nodes, and leaf nodes. An example of a GP individual is depicted in Figure 2.10 and shows the tree representation of the mathematical formula $(y - x) + (max(z, 0.8))$. The arithmetic operators $\{-, +, max\}$, which represent the root and nonterminal parts of the tree, are called functions. Functions can be as simple as arithmetic operators such as addition, subtraction, and multiplication, or more complex such as loop structures, and domain specific. Meanwhile, the

Algorithm 1 The GP Evolutionary Process

Input: $\mathcal{T}, \mathcal{F}, \delta, \beta, \gamma$ ▷ Terminal ($\mathcal{T}$) and function ($\mathcal{F}$) sets, population size (δ), number of generations (β), and ideal fitness value (γ)

Output: ϑ ▷ Best evolved program

1: $i \leftarrow 0$ ▷ The generations counter
2: $\lambda \leftarrow +\infty$ ▷ Best fitness so far
3: $\vartheta \leftarrow null$ ▷ Best solution so far
4: $\Xi_0 \leftarrow$ GENERATE$(\mathcal{T}, \mathcal{F}, \delta)$ ▷ Randomly generate the initial population
5: **repeat**
6: **for all** $\xi \in \Xi_i$ **do**
7: $\Delta_\xi \leftarrow$ FITNESS(ξ) ▷ Fitness of the current individual
8: **if** $\Delta_\xi < \lambda$ **then** ▷ If the fitness is better than the best so far
9: $\lambda \leftarrow \Delta_\xi$ ▷ Report the fitness as the best so far
10: $\vartheta \leftarrow \xi$ ▷ Make the individual as the best so far
11: **end if**
12: **end for**
13: $\Xi_{i+1} \leftarrow$ POPULATE(Ξ_i) ▷ Populating subsequent generation
14: $i \leftarrow i + 1$ ▷ Increment the generations counter
15: **until** $(i = \beta$ **or** $\lambda = \gamma)$ ▷ Check if a termination criterion is met
16: **return** ϑ

variables $\{x, y, z\}$ and constant $\{0.8\}$ are the inputs of the individual and called terminals, which represent the leaves part of the individual tree. The root and all nonterminal nodes are drawn from the function set, whilst the leaf nodes are taken from the terminal set. Each function node represents an operation that needs to be performed on a list of inputs that can be the values of terminals or the outputs of other functions. In order to ease observing the relations between the different parts of a program tree, the Lisp expression or *prefix* notation is widely used in the GP literature. For example, the formula $(y - x) + (max(z, 0.8))$ can be written as $(+\,(-\ y\ x)\,(max\,(z\ 0.8)))$.

Typically, an evolved GP program produces/returns a single numerical value as it represents the type of all terminals and the returned value of each nonterminal node. The program presented in Figure 2.10 is an example. However, different

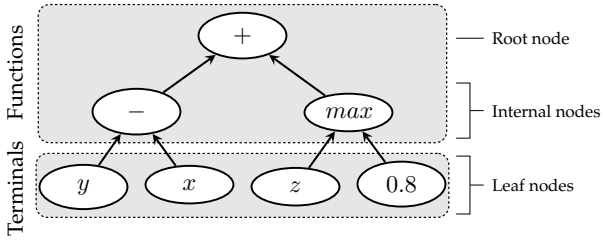


Figure 2.10: The GP tree-based representation for $(y - x) + (max(z, 0.8))$.

applications require the use of other types such as boolean and string, or more than a single value such as a vector of elements. For example, to allow an individual to make different decisions based on the inputs, conditional functions such as `if-then-else` can be used. Another example is the use of logical operators such as greater than and less than, to compare two numerical values and return a boolean value (either `true` or `false`). Therefore, a variety of approaches have been proposed in the literature that aim at extending conventional GP to consider different types in a single program tree and ensure the *closure* property of those programs. The grammar-based GP and strongly-typed GP are some typical examples [317, 207, 243]. The use of strongly-typed GP (STGP) [207] allowed the incorporation of data types and their constraints, in which terminal nodes can be of different types and each function outputs a specific type and operates on arguments of predetermined types. Another GP approach based on the concepts of grammars (GBGP) and context-free grammar (CFG-GP) has been proposed in the mid 1990s [317, 108, 126, 224] to tackle this problem (allowing the nodes to have different input and output types). McKay *et al.* [194] provide a comprehensive survey on grammar-based methods.

Although the tree-based representation of GP programs is widely used, it is not the only one. A number of researchers have investigated different types, such as linear GP (LGP) [223, 243], parallel distributed GP (PDGP) [238, 8], and cartesian GP (CGP) [203, 202].

2.2.3 Initialisation Methods

Like other EC techniques, GP starts the process by randomly generating a number of initial/candidate solutions. Although there is a rich literature of the approaches

that can be adopted to perform this task, the *full* and *grow* methods are two of the simplest and oldest methods that have been extensively used in the literature [153]. The full method generates an individual by randomly selecting elements of the function set until the maximum allowed depth of the tree is reached; then only elements of the terminal set are drawn at random to populate the leaf nodes. The maximum depth of a tree is defined as the longest path that starts at the root node and the farthest leaf node. The grow method, on the other hand, is similar to the full method and the only difference is that selecting from both sets (function and terminal) is permitted as long as the maximum depth has not been reached; and therefore, the growth of a branch will be stopped in the case that a terminal node is selected. Similar to the full method, the grow method forces to select from the terminal nodes once the maximum depth is reached.

To ensure having individuals vary in shape and size, the *ramped half-and-half* method has been devised based on the combination of the full and grow methods [152]. In this method, half of the individuals to populate the initial population are generated using the full method; whilst the other half is generated using the grow method. The ramped half-and-half method is the most widespread method [243].

2.2.4 Evaluation

A key component of the GP evolutionary process is the evaluation measure, which measures the goodness of, or how fit (fitness function) is, an evolved program to tackle the problem at hand [153, 162, 243]. The importance of this component comes from its role to guide the evolutionary process towards finding better solutions. The process of evaluating an individual starts by iterating over the content of the training set, and for each instance the system recursively traverses the program tree to calculate the results of the different nodes. Starting from the root node, the system postpones applying the function until the values of the input arguments (children) became available. Therefore, the terminal nodes are the first to get their values from the feature vector of the instances being evaluated, and pass their values to the parent nodes. The parent nodes then apply the corresponding operations and each passes the result to its parent node. This process continues until the root node is evaluated and returns a value. The design of this component,

i.e., fitness function, is heavily dependent on the problem. For example, if the task is to perform classification, then the fitness function can be the accuracy (the ratio of correctly classified instances to the total number of instances); or if the task is to solve a regression problem, then a good fitness measure can be based on how far is the value returned by the root node from the desired or the expected value.

2.2.5 Selection Methods

Selection methods are concerned with the task of which individuals will participate in creating individuals of the subsequent generation. The fitness value is used to assess the chances of an individual to be selected. In other words, better individuals, i.e., individuals having good fitness value, are more likely to be selected than inferior ones to produce offspring. The fitness proportionate selection (also known as Roulette wheel selection) represents one of the earliest selection methods that was originally used in GAs [103]. This method works by randomly selecting an individual based on the distribution of all fitness values of the current generation. A drawback of the Roulette wheel method is that individuals with bad fitness values may have good building blocks (sub trees) but will have very low probability to be selected. To tackle this problem, tournament selection has been proposed. In this method, a predefined number of individuals are randomly drawn from the population. Then, the winner amongst them, which has the best fitness value, is selected. Regardless of the fitness value, this method gives all individuals an equal chance to be drawn in the first step [190]. However, this method introduces an extra parameter that needs to be set which is the size of the tournament. The use of a large tournament will reduce the chances of inferior individuals to be selected; however, the use of a small tournament will increase the probability of inferior individuals to be selected and reduces the probability of good individuals to take place in the mating pool. Setting the tournament size to be one (only a single individual) will result in making the selection task completely random.

2.2.6 Genetic Operators

In GP, the individuals of the current generation are used to populate the subsequent generation through applying a number of genetic operators. Those operators aim

at generating new individuals (children) by utilising the genetic materials of the current population individuals (parents). Basically, there are three operators in GP: (1) reproduction; (2) crossover; and (3) mutation. Each of these operators is applied based on a user-defined probability in which the summation of the three is 1. In GP, these three operators are mutually exclusive, which is unlike other EC methods where these operators can be applied sequentially [243].

2.2.6.1 Reproduction

Reproduction is simply performing a direct copy operation of the selected individual to the next generation. Another similar operator to reproduction is *elitism*. Instead of applying the copy operation based on a predefined probability, a predefined number of the top or elite individuals are copied unchanged from the current generation to the next one. Hence, this operator aims at maintaining the achieved level of performance and prevent it from degrading. In other words, this operation ensures that the next generation is at least as good as the current generation and the system does not spent time rediscovering some good materials that have been previously found. In both reproduction and elitism, the selected individuals remain in the current generation and are allowed to take place in breeding the next generation. The content of the final generation will be identical to the initial population when the elitism is set to be equivalent to the population size. Hence, the probability of this operator is set to a very low value to give the system the flexibility to explore the solution space.

2.2.6.2 Crossover

The crossover operator allows the system to generate new individuals (children) by exchanging the genetic materials from two existing individuals (parents). The parent individuals are first selected using one of the selection methods discussed earlier, then a crossover-point from the tree of each of them is randomly chosen, and the sub trees are swapped at the crossover points. An example of this operation is demonstrated in Figure 2.11. The crossover operator has been extensively studied and various methods to improve or modify this operator have been proposed [326].

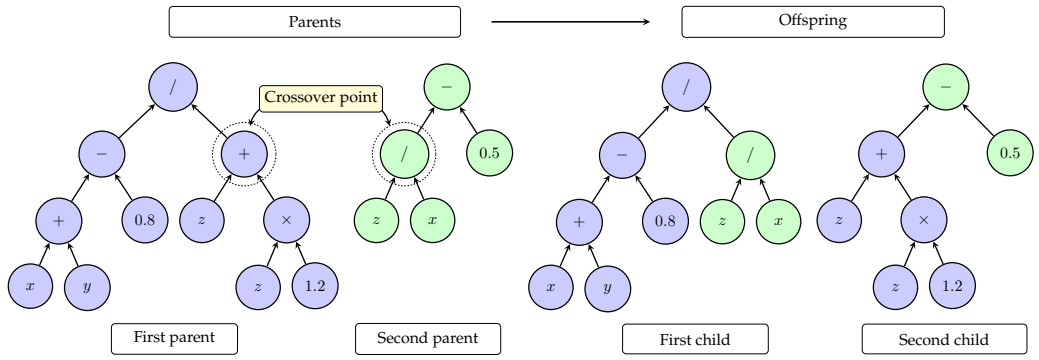


Figure 2.11: Example demonstrating the crossover operator.

2.2.6.3 Mutation

Similar to the crossover operator, the mutation operator uses existing individuals to generate new ones. However, in mutation, only one individual (parent) is selected from the existing population and the other parent is randomly generated using one of the initialisation methods explained earlier. Moreover, a mutation-point on the parent is randomly selected, and the randomly generated parent is simply replacing the sub-tree at that point in order to generate the new individual (child). An example of this operation is presented in Figure 2.12. There is a clear, yet important, difference between crossover and mutation in which the latter allows the system to introduce new building blocks (sub trees); while the former only allows the system to try different combinations of the existing building blocks. Moreover, the number of individuals generated after applying each of these two operators is different where only one individual is generated from mutation and crossover produces two individuals. The mutation operator has been extensively studied and various methods are proposed [236]

2.2.7 Open Issues in GP

Although GP has many good properties compared to other EC and non-EC methods (see Section 1.2.2 on page 6), it suffers from some issues that have been indicated by a number of studies in the literature [225]. Providing a comprehensive investigation and review of the open issues in GP is beyond the scope of this dissertation, and

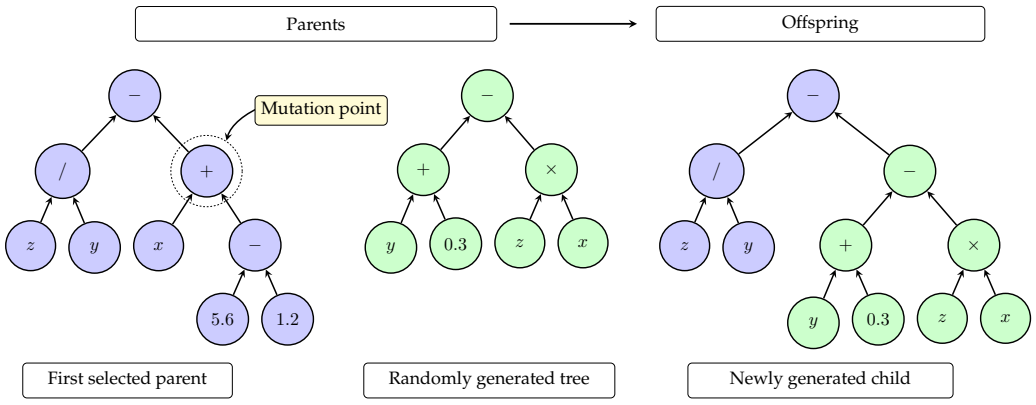


Figure 2.12: Example demonstrating the mutation operator.

only some of those well-known issues are briefly discussed below.

2.2.7.1 Code Bloating

Generally, GP tends to gradually increase the size of the evolved programs during the evolutionary process, which is a well-known phenomenon in GP denoted as "bloat" [243, 318]. Bloat can hamper the efficiency and largely affect the ability of the system to find good solutions [16]. This phenomenon has been widely studied and numerous methods have been proposed to control the size of the evolved programs [239, 290, 273, 147, 280].

2.2.7.2 Program Representation

The ability to have a flexible program representation is a very important and powerful property of GP; however, identifying appropriate representations is highly influenced by the problem being tackled and as it can be very difficult to find an optimal representation [225].

2.2.7.3 Terminal and Function Sets

The programs in GP are generated using the content of the function set (nonterminal nodes) and the terminal set. Typically, the content of the terminal set is the inputs of the problem at hand. However, the content of the function set is not bounded

or limited to a specific set. The function set is a critical component and its content must be related to the problem at hand; otherwise, GP will not be able to find a good solution [308].

2.3 Related Work

This section presents and discusses the closely related work in the literature to the different methods proposed in this thesis. Providing a full survey on the literature regarding image-related methods is beyond the scope of this thesis. The use of GP for binary, i.e., two classes, image classification is discussed first. The use of GP for multi-class image classification is then surveyed, followed by a very brief discussion on non-GP methods for image classification. Automatically evolving image descriptors represents a core part of this thesis, and therefore, evolutionary-based image descriptors are also discussed in this section. Another key part of this thesis is to use a limited number of training examples, and adopting transfer learning; hence, the related work on these two schemes is reviewed.

2.3.1 Genetic Programming for Image Classification

Adopting GP techniques in different domains to solve a variety of problems has increased in the last few decades, such as feature extraction [10, 6], classification [261, 275, 283], object detection [29, 31, 32], image segmentation [311, 169, 170], image regeneration [52, 161], and image processing [118]. The GP methods proposed in the field of computer vision can be categorised differently based on different criteria, e.g., in terms of the task, the domain, and the program structure. In the subsequent sections, some of the recently proposed methods for the problem of image classification are discussed. Based on the number of classes, the methods are categorised into GP for binary image classification, and GP for multi-class image classification.

2.3.1.1 Genetic programming for binary image classification

Typically, a tree-based GP individual program produces a single value from the root node for each instance. In the case of binary classification tasks, a predefined

threshold value can be used to split the result space into two intervals such that each is used to represent the set of mapping values (i.e. the program's output) of one class [275, 276]. For example, selecting zero as a threshold value such that all negative values are assigned to one class, and all positive and zero values are assigned to the other class.

Bhowan *et al.* [34] proposed two GP-based methods to address the problem of class unbalance for binary image classification. The traditional single objective GP system is used in the first method, which aims at adapting the fitness function to evolve an individual that is capable of handling the class unbalance problem. In the second method, a multi-objective GP approach was adopted to perform two tasks simultaneously: (1) evolving a set of classifiers; and (2) tackle the problem of minority and majority class trade-off. The performance of these two methods was evaluated using two datasets with a highly unbalanced number of instances in the two classes. The methods have successfully addressed the problem of unbalanced data; however, human intervention to detect and extract features is required prior to the use of the GP system. Moreover, a large number of instances is required to evolve an individual as those methods were not designed to tackle the problem of having a limited number of instances.

In [167], Li *et al.* proposed a GP method for binary image classification through the use of a loop structure. The method was applied on a dataset that consists of instances fall into two groups: circle, and square. Compared to programs without loops, their experiment results show that a much better performance has been achieved by the evolved programs with loops. However, performing feature extraction prior to the task of evolving a classifier is required. Furthermore, domain knowledge is needed to provide some restrictions on the used loop statements.

Abdulhamid *et al.* [1] investigated the use of four different loop structures in a GP system for image classification. Three different datasets were used to evaluate the evolved program and compare it to conventional GP. The results suggested that the use of loop structures can significantly improve the performance of the evolved classifier.

A multi-tier domain-independent GP method for binary image classification was proposed by Atkins *et al.* [18]. The main objective was to automatically evolve a classifier that is capable of performing image filtering, feature extraction,

and classification. Their experiments on two datasets revealed that a comparable performance to the use of domain-specific features has been achieved.

Motivated by the work of Atkins *et al.* [18], Al-Sahaf *et al.* [7] have proposed and investigated a variety of GP methods for binary image classification. Evaluating their methods on four datasets of increasing difficulty showed that a significantly better performance has been achieved compared to different GP-based and non-GP methods.

2.3.1.2 Genetic programming for multi-class image classification

Different approaches have been adopted to utilise GP for multi-class classification problems such as binary decomposition, static range selection, dynamic range selection, class enumeration, and evidence accumulation [336, 184].

A multi-class image classification method has been proposed by Li *et al.* [168] via introducing two improvements in the GP system. The first was the use of different sets of power values during the evolving (training) process in order to find a better range of threshold values. Introducing the program size in the fitness function represents the second improvement which aims at controlling the size of the evolved classifier. The Chinese character dataset and grass leaves dataset were used to evaluate their method. The results show that a superior performance has been achieved over the use of static and dynamic range selection methods.

In [275], Smart *et al.* developed two dynamic-based range selection methods for the problem of multi-class image classification in GP. The first method is centred dynamic range selection, and the second is slotted dynamic range selection. The results of evaluating those two methods using five datasets of varying difficulty show that both of those methods have outperformed the use of the static range selection method.

A GP-based method for the tasks of texture classification and texture segmentation was proposed by Song *et al.* [281]. In their work, a bitmap textures dataset that consists of 48 different textures were used to evaluate the proposed method, and the results show that GP is able to evolve accurate classifiers. Moreover, their method does not require feature extraction as a preprocessing step.

A domain-independent approach for the problem of multi-class object detection using GP is proposed by Zhang *et al.* [337]. The aim of their method is to locate

a number of objects of different classes that are contained in a large image, and predict the class label of each of the detected objects. In other words, the evolved program is capable of performing object detection and multi-class classification tasks. The method is tested using three datasets of increasing difficulty.

Downey *et al.* [74] proposed a linear GP (LGP) approach for the problem of multi-class image classification. Moreover, an extended implementation of the proposed LGP modified the mutation operation to selective mutation. Both of those methods have outperformed the tree-based GP on the three experimented datasets.

2.3.1.3 Genetic programming for other image-related tasks

Ryan *et al.* [261] used GP techniques for detecting stage-1 cancer in digital mammograms. Their method performs a series of preprocessing operations, e.g., background suppression, image segmentation, feature detection, and feature selection, in order to reduce the volume of data to process, and then the preprocessed data are fed into GP to evolve a classifier. The results of their experiments show that this work-flow (preprocessing steps and GP classifier) can successfully detect a stage-1 cancer in digital mammograms. The work-flow requires task-specific features which requires domain-expert intervention; and as a multi-stage system, the success of any subsequent stage is subject to the goodness of performing previous stages.

Detecting edges in images is an important task in a wide variety applications in computer vision such as image segmentation. Fu *et al.* [99, 100] studied edge detection and used GP to evolve edge detectors that outperformed some well-known detectors that are designed by domain-experts such as Sobel [279, 76] and Canny [46]. However, their methods are designed to detect only one type of keypoint, i.e., edges, while neglecting other types of keypoints, and requires a large number of training examples.

By adopting a multi-objective approach, Albukhanajer *et al.* [10] utilised GP for extraction of image features that are robust to noise and invariant to geometric deformations, e.g., illumination, rotation, and scale. Their system automatically combines different functionals and aims to maximise the between-class variance and minimise the within-class variance. The results of their experiments on two datasets showed good robustness of the method to noise and geometric deformations.

2.3.2 Other Paradigms for Image Classification

In [45], Camps-Valls *et al.* presented a kernel-based method framework for hyperspectral image classification. The aim of their study was to examine and analyse the properties of different kernel-based methods in the domain of hyperspectral images. Moreover, the authors have developed two new methods that are regularised radial basis function neural networks, and regularised AdaBoost. The study also compares the performance of the two newly developed methods to that of SVM and kernel Fisher discriminant methods.

An in-depth experimental investigation of the problem of pedestrian classification is presented in [210] by Munder *et al.* via examining a variety of combinations of multi-feature classifiers. Moreover, the local versus global and nonadaptive versus adaptive features were investigated in their study. Another objective of the paper was to study the impact of using different numbers of training instances on the performance. In other words, they investigate the correlation between the size of the training set and the performance of the trained model on the test set. The results of their experiments revealed that combining SVM with local receptive fields (LRFs) achieved the best performance amongst all other combinations.

Larochelle and Bengio have developed a restricted Boltzmann machine (RBM) [277] based classifier in [164]. Usually, RBMs are used as a preprocessing step to perform feature extraction for another learning algorithm. However, the main argument of Larochelle and Bengio is that RBMs can be used to develop a standalone classifier as well. Furthermore, in their paper the authors emphasised the possibility of employing RBMs in a semi-supervised setting.

A recent work by Loukas *et al.* [183] proposed a pattern classification model for the problem of breast cancer image classification. Using texture analysis, 30 patches are extracted from each instance (image) to form a set of textural features. The extracted features are then fed into one of three commonly used methods k-Nearest Neighbours (k-NN), support vector machines (SVM), and probabilistic neural networks (PNN). The results of their experiments show that the method has achieved a high level of performance on the researched problem.

Motivated by the success of using backpropagation [257, 259] to train multilayer perceptrons, and inspired by the organisation of the animal visual cortex [193], convolutional neural networks (CNNs) have emerged and have been widely

investigated in computer vision and pattern recognition [166, 165]. Over the past 30 years, CNNs have received a lot of attention and have been used in a wide range of real-world applications [81, 286, 75]. Although CNNs have been shown to achieve the-state-of-the-art performance compared to other methods [75], they suffers from a number of issues. Two of the most well-known issues are the number of training examples, and network structure [88]. It is well-known that a large number of examples are needed in order to train a CNN, which is also affected by the size of the network, i.e., number of free parameters to be adjusted [122]. The structure of the network, on the other hand, needs to be predetermined beforehand including the number of layers, weights, nodes, and how the nodes in a layer are connected to those in the adjacent layers [88].

2.3.3 Evolutionary-based Image Descriptors

Since the late 1990s, GP has been used to automatically evolve/construct image keypoints and features [241, 74, 1, 6, 10, 337, 282, 169, 170, 52], showing good potential in this direction. The method proposed by Ebner and Zell [80] is one of the earliest works employing GP to automatically evolve an interest point detector. Similarly, Trujillo and Olague [300] have used GP to synthesise an interest point detector. They extended this work in [301] to improve the performance of the evolved points detector by taking into consideration the global separability and geometric stability of the detected points. Olague and Trujillo [221] used GP to evolve image operators for detecting interest points in an image. Motivated by the success of [300, 221], Perez *et al.* [233] proposed several methods, where GP is used as a strategy to evolve image descriptors for object detection tasks. The main focus of Liu *et al.* [178] is on evolving a spatio-temporal descriptor for human action recognition by employing GP techniques. Shao *et al.* [268] proposed a multi-objective GP methodology for the task of feature learning in image classification.

Combining GP and SIFT features to improve the performance for object recognition was proposed by Hindmarsh *et al.* [125]. The idea is to use GP as a post-processing step to construct better features from the detected SIFT features. The results in [125] are comparable to the use of SIFT features alone. The

system operates in two stages where keypoints detection and feature extraction are performed by SIFT in the first stage, and feature construction and classification are performed in the second stage by GP. However, abundant labelled data is needed.

Trujillo *et al.* [300] proposed a GP-based low-level feature detector and extractor via formulating the task of interest point detection in terms of an optimisation problem. The main focus of the authors was balancing between the domain knowledge expertise and genetic programming in order to achieve comparable or better results than human crafted features. In their paper the Harris detector [119] has been used as a competitor descriptor on two datasets. Their results suggest that superior performance has been achieved by their method over the Harris detector. The authors have extended their work in [301] and [221] via presenting 15 new GP evolved descriptors.

2.3.4 Transfer Learning for Image Classification with Small Number of Instances

In [86, 85], Li *et al.* proposed a Bayesian-based model for the problem of object recognition. In their system, *general knowledge* is extracted from previously learnt groups of objects using abundant instances, which is then used to form a prior probability density function. A posterior density is then produced via updating this knowledge given a small set of training instances in the target domain. The system was tested using a dataset consisting of instances that fall in 101 different categories, and compared against two commonly used methods: maximum likelihood (ML), and maximum a posteriori (MAP). The results of their experiments show that their system has significantly outperformed the competitive methods when the number of training instances is relatively small. Moreover, they investigated the effectiveness of using the knowledge extracted by their method on the two competitor methods. The results of the investigation suggested that a better performance has been achieved when both of the ML and MAP methods used the extracted knowledge.

Miller *et al.* [201] used deformation matrices for the problem of object classification. The system is trained using a large number of instances of a character (e.g. letter “A”) and then try to make the system learn a different object (e.g. digit “4”) using a single (or few) instance. To achieve this goal, two learning scenarios are

combined under a single framework: adopting the transfer learning using a model parameters approach in order to use a reduced number of instances in the target domain; and frequently updating the system as more training images trickle in. The authors have identified situations where their method for a set of characters will fail to converge to the global optimum. The method is tested on binary image classification using two different training set sizes: 1000 instances; and only one instance. The results of their experiment show that, unlike other comparative methods, the performance did not drop significantly after reducing the training instances of the target domain from 1000 to 1.

An online algorithm for the problem of object identification was proposed by Ferencz *et al.* [90]. Unlike object categorisation where the task is to predict the class label of an instance (e.g. is the image a face or non face), object identification aims at identifying the content of an instance (e.g. Bob's face or Jen's car). The algorithm takes a single labelled instance and builds an effective "same" vs. "different" classification cascade via detecting the most informative (i.e. discriminative) features and patches for the object of that instance. Moreover, in their paper they have tried to address two challenges: (1) the problem of small inter-class variation; and (2) the limitation of positive instance availability. In order to address these challenges, the proposed algorithm is split into three stages: (i) learning hyper-features; (ii) modelling pairwise relationships between patches; and (iii) building the classifier cascade. Each stage involves a number of complicated sub-stages in which each has its own assumptions. The method was tested on two datasets and the results show a good level of performance has been achieved over other competing methods. Furthermore, the authors compared their method to one of the leading methods (SIFT [186]) on one dataset and also the proposed method has outperformed that method.

Motivated by the ability of a human brain for learning by knowledge transfer, Rodner *et al.* [256] have modified the Randomised Decision Forest (RDF) [102] learning algorithm in order to build a classifier using very few instances. Similar object categories are used to learn a prior distribution, which is then reused (transferred) to maximise a posteriori estimation of the model parameters. Evaluating the method on three datasets shows that a significant performance improvement have been achieved over Geurts *et al.* 's RDF classifier [102].

A one-shot learning approach (a sub-approach of transfer learning in which there are only a few instances of the target domain are available for training) has been adopted in [159] for the problem of handwritten character recognition via knowledge transfer by contextual information. The system infers the latent strokes from previously learnt different characters compositions. The relationships between different strokes that are used to form different characters are transferred to a different domain to help in recognising previously unseen data. Using two datasets and compared to human perceptual judgements, the authors tested their method. However, the results of their experiments revealed that no significant improvement has been achieved by the proposed method over the competing methods.

A hierarchical non-parametric Bayesian model was developed in [263] for object categorisation via adopting the one-shot learning approach. The model relies on previously learnt categories to transfer acquired knowledge to a novel category. The model groups different categories into informative super-categories. Given a single example, the model infers under which super-category this instance can be categorised. Moreover, the model is capable of identifying new categories in an unsupervised learning fashion from a single or a few instances. The model has been tested on two different datasets of varying number of groups and instances per group. The results of their experiment show that a significantly better performance has been achieved by their method over the use of simple hierarchical Bayesian approaches. Moreover, their method is capable of discovering new categories given a few instances in an unsupervised fashion.

A recent work by Lu *et al.* [188] proposed a method based on the discriminative multimanifold analysis for the problem of face recognition. The main aim of the paper is addressing the problem of building a classifier using a single sample per person (SSPP). To achieve this task, (1) the system partitions each instance into a number of non overlapping patches; (2) the single sample per person recognition is then formulated as *manifold-manifold* matching problem; and (3) in order to identify an unlabelled subject, a reconstruction-based manifold-manifold distance is used. The authors show that their method has achieved good performance compared to the performance of the stat-of-the-art method on three broadly used faces datasets.

Table 2.3: A summary of the benchmark image classification datasets.

			Changes			Dimensions		
Dataset	Classes	Instances	Light	Rotation	Scale	width	height	Chapters
Coins	2	384	✓	1	1	55	55	{5, 7}
Faces	2	31022	✗	1	1	19	19	{3, 5, 7}
KTH-TIPS	10	810	✗	1	9	200	200	{3}
BrNoRo	20	1680	✗	1	1	64	64	{4, 5, 6, 7}
BrWiRo	20	20160	✗	12	1	64	64	{5, 6, 7}
OutexTC00	24	480	✗	1	1	128	128	{4, 5, 6, 7}
OutexTC10	24	4320	✗	9	1	128	128	{5, 6, 7}
KySinHw	25	22500	✓	9	1	122	122	{6, 7}
KyNoRo	28	4480	✗	1	1	115	15	{3, 4, 5, 6, 7}
KyWiRo	28	53760	✗	12	1	115	115	{3, 5, 6, 7}
Dslr	31	498	✗	✗	✗	1000	1000	{7}
Webcam	31	795	✗	✗	✗	152-752	152-752	{7}
Amazon	31	2817	✗	✗	✗	300	300	{7}
CUReT	61	5612	✗	1	1	200	200	{7}

✓ The change is controlled.
✗ The change is uncontrolled.

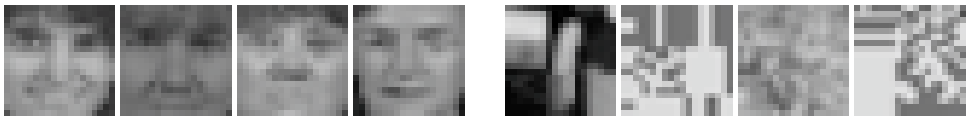
2.4 Benchmark Datasets

The methods developed in this thesis are evaluated using a set of carefully chosen benchmarks for image classification of varying difficulty. However, those methods can also be used to perform other tasks such as content-based image retrieval and image segmentation. As summarised in Table 2.3, these datasets varying in domain (texture and object classification), rotation, illumination, scale, size of instances, number of classes, and number of instances per class. However, the instances of all those image datasets are grey-scale images, i.e., each pixel carries only brightness/intensity information that can be white at the strongest intensity or black at the weakest intensity [138]. Therefore, the pixel values are ranging between 0 (black) and 255 (white). The datasets presented in Table 2.3 are primarily sorted in ascending order based on the total number of classes, and secondly sorted based on the total number of instances.

The *New Zealand Coins* dataset [274] comprises a set of 5 cent New Zealand coins that fall into two classes: *head* and *tail*. This dataset consists of 384 grey-scale instances in total, each of which is of size 55×55 pixels, where 192 of them

(a) (b)

Figure 2.13: Samples of the Coins dataset (a) head, and (b) tail instances.



(a) (b)

Figure 2.14: Samples of the Faces dataset (a) face, and (b) non face instances.

showing the head side of the coin and 192 showing the tail side. Those instances appear in different (uncontrolled) rotation angles around the centre as presented in Figure 2.13.

The *CBCL Face*[1] dataset [123] consists of two classes: face, and non face. The task is to classify/discriminate between face and non face images as shown in Figure 2.14. Each instance of the CBCL Face dataset is of size 19×19 pixels, where the face instances were hand-aligned to be relatively in the centre of the example. Originally, there are 2429 face and 4548 non face instances in the training set, and 472 face and 23573 non face instances in the test set. Clearly, the number of instances of the two classes is highly unbalanced; hence, we did not use the original division of the data and instead all instances of the face class are used and a nearly equal number of instances from the non face class has been randomly selected. In other words, 6000 instances in total have been selected from the two classes that are 2901 faces and 3099 non faces to form the *Faces* dataset in this thesis.

The *Kylberg Texture*[2] dataset [157] is widely used in computer vision for texture classification. It contains images of different materials, e.g., cushion, rug, rice, grass, and stone (Figure 2.15). In total, this dataset comprises 28 classes each of which comes in two flavours: *without-* and *with-rotation*. Originally, the instances of this

[1]Available at: `http://poggio-lab.mit.edu/codedatasets`

[2]Available at: `http://www.cb.uu.se/~gustaf/texture/`

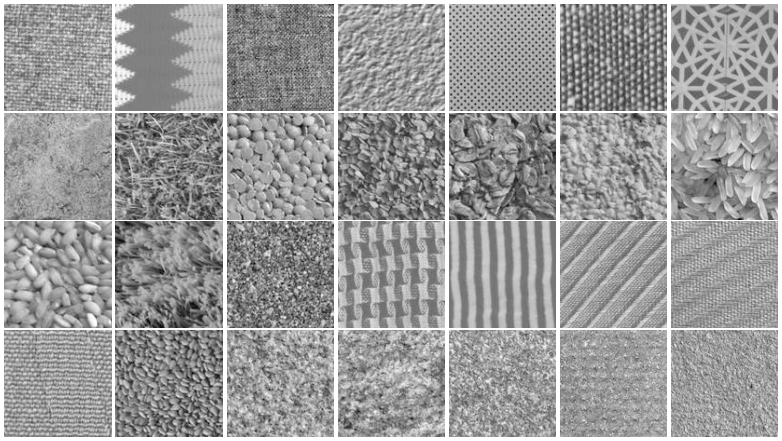


Figure 2.15: Samples of the Kylberg dataset.

dataset are of size 576×576 pixels. Each of the without-rotation classes consists of 160 instances whilst there are 1920 instances in each class of the with-rotation classes that fall into 12 rotation angles $\{0°, 30°, \ldots, 330°\}$. Dealing with large images is computationally intensive and a time-consuming task [53], where different methods can be used to reduce the computation time and required memory [24]. In this thesis, the size of the original instances have been reduced to 115×115 pixels via subsampling, i.e., average pooling, in order to reduce the computation time.

A recently proposed dataset for texture classification that represents a variant of the Kylberg dataset is *Kylberg Sintorn Rotation*[3] dataset [158]. This dataset consists of 25 texture classes, each made up of grey-scale instances of size 122×122 pixels as presented in Figure 2.16. Each class comprises of 900 instances that fall into 9 different angles: 0°, 40°, 80°, 120°, 160°, 200°, 240°, 280°, and 320°. Moreover, the instances of this dataset are normalised with a mean value of 127 and a standard deviation of 40, and rotated using 6 different methods: *hardware*, *nearest neighbour*, *linear interpolation*, 3^{rd} *order cubic interpolation*, *B-spline interpolation*, and *Lanczos 3 interpolation*. In this thesis, only the instances that were rotated by the hardware method (KySinHw) are considered as it better represents the real-world situations compared to the other methods, where an sample is placed on a rotatable desk and the camera is positioned vertically on the top as depicted in Figure 2.17.

[3]Available at: `http://www.cb.uu.se/~gustaf/KylbergSintornRotation/`

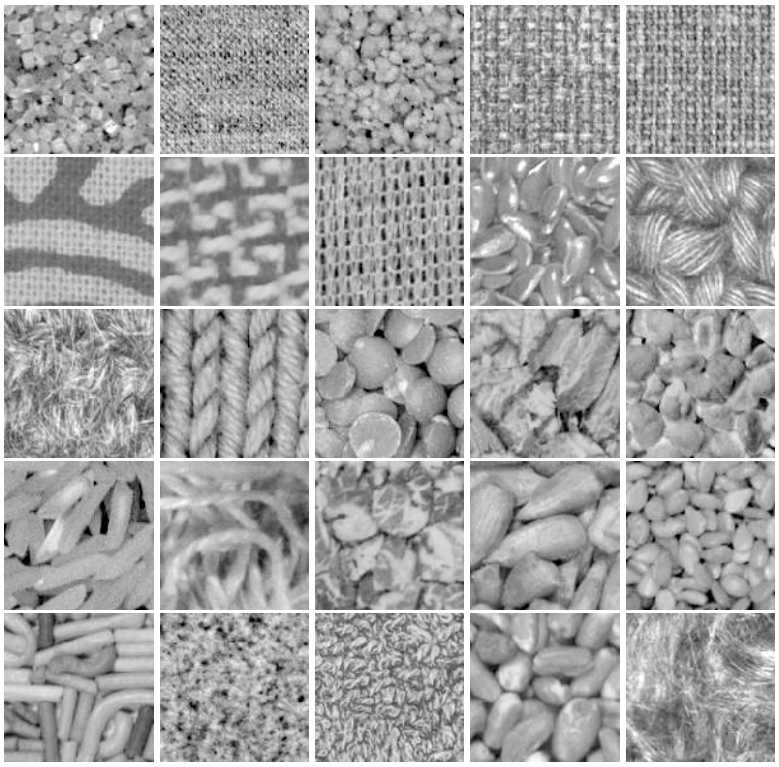


Figure 2.16: Samples of the KySinHw dataset.

Another popular and widely used dataset for texture classification in computer vision is the *Brodatz Texture*[4] dataset [42]. The Brodatz dataset contains images for different materials, e.g., grass, bark, wood grain, and brick wall (Figure 2.18). In total, there are 112 classes in the Brodatz dataset, each of which consists of a single grey-scale instance of size 640×640 pixels. Only 20 classes have been randomly selected out of the 112 classes, mainly due to the associated preprocessing effort needed to prepare the instances of each class, and kept consistent in this thesis. In order to generate the Brodatz without rotation (BrNoRo) dataset in this thesis, the single instances of those randomly selected 20 classes is re-sampled into non-overlapping sub images each of size 84×84 pixels. Meanwhile, the same examples of those 20 classes are then rotated around the centre by 12

[4]Available at: `http://multibandtexture.recherche.usherbrooke.ca/original_brodatz.html`

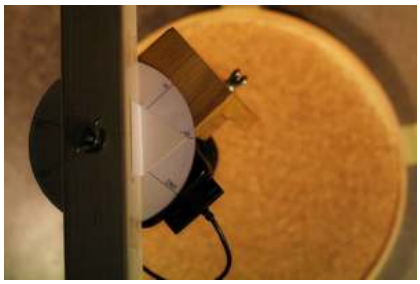


Figure 2.17: The setup during acquisition of the hardware rotation method (reproduced from [156]).

angles with a step of size 30° to generate the Brodatz with rotation (BrWiRo) dataset in this thesis. Therefore, the total number of instances in BrNoRo and BrWiRo is, respectively, 1680 (20 (classes)×84(instances)) and 20160 (20 (classes)× 12 (rotations) × 84 (instances)) instances. The single original image (640 × 640 pixels) can be divided into a grid of 10 × 10 non-overlapping tiles as presented in Figure 2.19(a). However, if it is rotated to any angle around the centre as presented in Figure 2.19(b), some of those tiles will be outside the boundaries of the image, i.e., will have part of the image and the rest has to be filled with some default values such as white colour. Rotating the image 45° gives only 85 complete tiles as shown in Figure 2.19(c). When the original image is rotated 30°, only 84 complete tiles enclosed within the boundaries of the image can be extracted as depicted in Figure 2.19(d).

The *Outex Texture Classification*[5] dataset [216] consists of 17 texture classification test suites {Outex_TC_00000, Outex_TC_00001,. . . , Outex_TC_00016}. Those 17 datasets comprise different numbers of classes, materials, rotations, and instances. The Outex_TC_00000 dataset comprises 24 classes each of which consists of 20 instances (Figure 2.20). The instances of this dataset are rotation-free and it has been used in this thesis as OutexTC00. Meanwhile, the Outex_TC_00010 dataset is also comprises of 24 classes each of which has 180 instances. The instances of this dataset fall into 9 different angles: 0°, 5°, 10°, 15°, 30°, 45°, 60°, 75°, and 90°. In this thesis, the OutexTC10 is formed using the instances of the

[5]Available at: `http://www.outex.oulu.fi/index.php?page=classification`

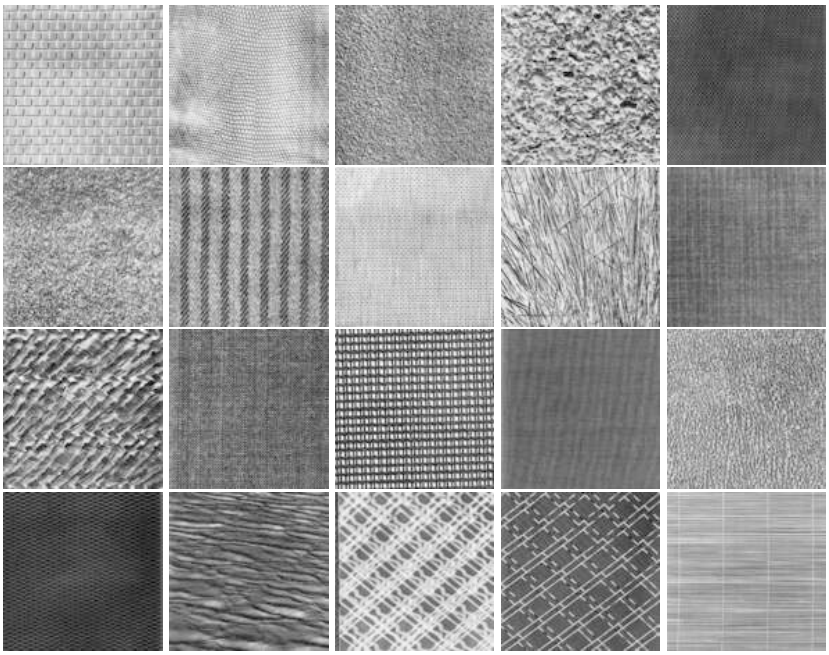


Figure 2.18: Samples from the 20 randomly selected classes of the Brodatz dataset.

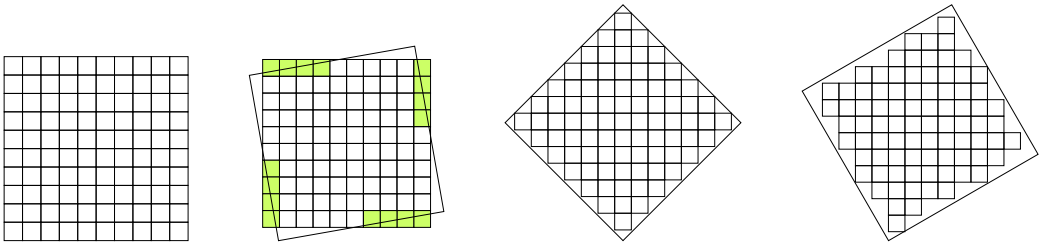
Outex_TC_00010. The instances of both OutexTC00 and OutexTC10 datasets are of size 128×128 pixels.

The *Columbia-Utrecht Reflectance and Texture*[6] (CUReT) dataset [60] consists of 61 real-world surfaces textures as shown in Figure 2.21. The classes were carefully chosen to span a variety of geometric and photometric properties such as specular, diffuse, isotropic, anisotropic, coloured, natural, and man-made surfaces. The size of each instance is 200×200 pixels and there are 92 instances in each class. Originally, this dataset comprised coloured images that have been been converted in this thesis into grey-scale using the Decolorize method [109] as suggested in [140].

The *KTH-Textures under varying Illumination, Pose, and Scale*[7] (KTH-TIPS) dataset [39] consists of ten classes as depicted in Figure 2.22. This dataset extends the CUReT dataset in two ways: it provides instances under different scales, and

[6]Available at: `http://www1.cs.columbia.edu/CAVE//exclude/curet/.index.html`

[7]Available at: `http://www.nada.kth.se/cvap/databases/kth-tips/`



(a) (b) (c) (d)

Figure 2.19: Illustration of different sampling of an image rotated (a) 0°; (b) 10°; (c) 45°; and (d) 30° around the centre.

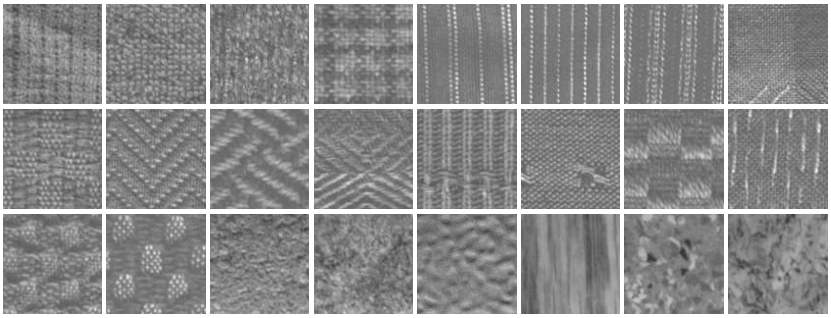


Figure 2.20: Samples of the Outex TC dataset.

it introduces samples of more real-world materials. Each class of the KTH-TIPS dataset comprises 81 instances of size 200 × 200 pixels. In this thesis, only two visually-close classes are chosen: sponge, and brown bread.

The *Office*[8] dataset [262] was originally designed to investigate the ability of a method for domain adaptation learning [41, 27] (an approach of the transfer learning or learning by knowledge transfer). This dataset comprises 3 domains: amazon, dslr, and webcam, each of which consists of 31 classes. The 31 classes are identical in the three domains, however, the instances of those classes in each domains are different (the object, number of instances, and size of instances). Samples of these domains are depicted in Figures 2.23 to 2.25. The dslr domain (Dslr) comprises 498 high-resolution images of size 1000×1000 pixels that are captured using digital single-lens

[8]Available at: `https://people.eecs.berkeley.edu/~jhoffman/domainadapt/#datasets_code`

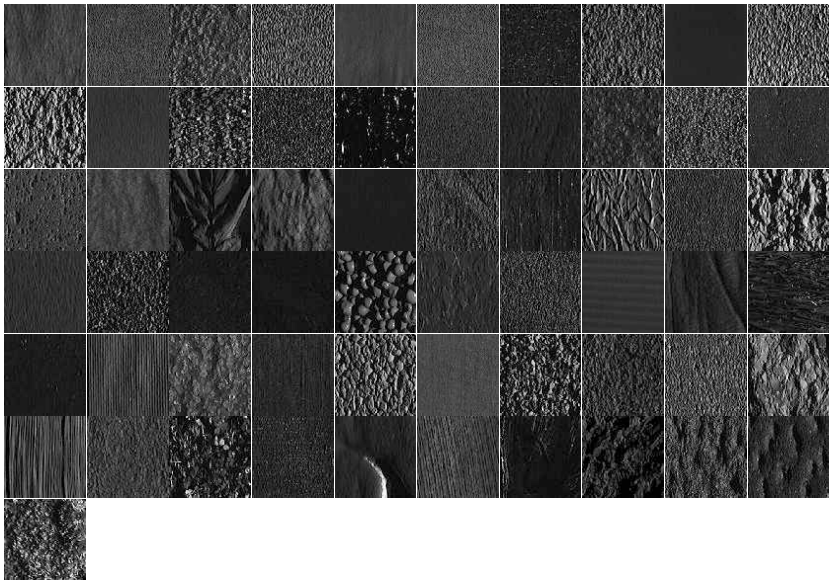


Figure 2.21: Samples of the CUReT dataset.

reflex (DSLR) cameras in uncontrolled (realistic) environment. The amazon domain (Amazon) comprises 2817 mid-resolution images of size 300×300 pixels, which are images of products listed on the `www.amazon.com` online shopping web page. The webcam domain (Webcam) comprises 795 low-resolution images that have been captured using webcam cameras. Unlike the other two domains, the instances of the webcam domain are varying in size that ranges from 152×152 pixels to 752×752 pixels.

2.5 Chapter Summary

The basic concepts and terminology of computer vision have been reviewed in this chapter. This chapter also presented a brief overview of the concepts of keypoints and image features, feature extraction and selection, image descriptors, machine learning, transfer learning, evolutionary computation, and genetic programming. The related work of using GP and other techniques for image classification, and evolving image descriptors were also reviewed in this chapter. The benchmark datasets used to assess the performance of the different methods proposed in this

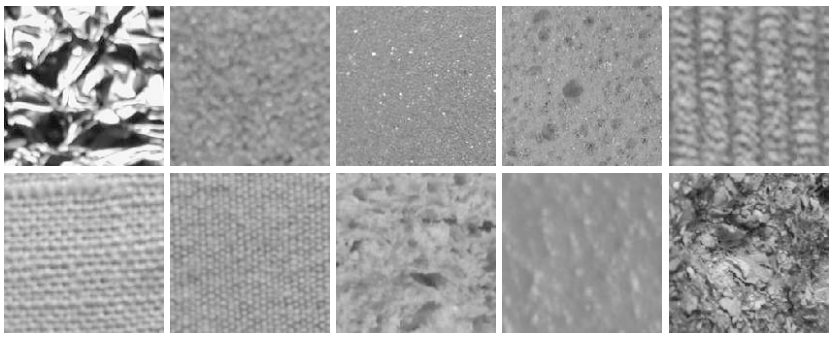


Figure 2.22: Samples of the KTH-TIPS dataset.

Figure 2.23: Samples from the Dslr domain of the office dataset.

thesis were presented and discussed in this chapter.

This chapter highlighted the limitations of the existing work in order to form the motivations of this research. Performing image classification is not an easy task and requires human intervention to design the regions of interest, and extract features in order to build a classifier. The main aim of this research is to use GP to automatically evolve a program using only a few instances of the problem, which operates on the raw pixel values and extracts features to facilitate performing image classification. The work in the literature shows that the problems of automatically detecting image keypoints and performing feature extraction have been dealt with extensively. However, the vast majority of those methods are manually designed by domain-experts which is a very time consuming and expensive task to perform. GP

Figure 2.24: Samples from the Amazon domain of the office dataset.

Figure 2.25: Samples from the Webcam domain of the office dataset.

has been successfully used to handle and tackle different problems, and can cope with problems of different nature due to GP's flexibility. However, automatically evolving image descriptors has not yet been investigated in depth and there are still a number of unanswered questions and open issues that require more thorough investigation.

- **GP-based descriptors**
  Using GP to evolve image descriptors or improve existing ones has been studied recently. However, this research direction is relatively new and it is mainly focused on sparse image descriptors. Automatically evolving dense

image descriptors using GP or other EC techniques has not been investigated yet. Using a sample of the problem instances to evolve a descriptor allows the system to discover more domain specific features. Moreover, a domain-expert may miss to consider some important features during the design phase which will become impossible to be detected in the subsequent phases.

- **GP with limited number of examples**
  The vast majority of existing machine learning algorithms were designed base on the assumption that a sufficient number of examples are available. Such algorithms may not cope with a small number of training instances. This problem can be tackled if the features extracted from the training set are informative and rich, i.e., avoid the use only a few features that can be good to discriminant between the training instances but not reliable to generalise to the unseen data. The evolutionary nature of GP, and with the use of a carefully designed fitness function, the system can be guided toward discovering more informative features. Using a small number of examples can increase the system efficiency and speed-up the training procedure.

- **Robust image descriptors**
  Designing an image descriptor is a challenging task, and this task will be even more challenging to make the descriptor invariant to image changes such as illumination, rotation, and scale. Existing descriptors are hand-crafted which means extending such descriptors to handle some of these changes requires manual adjustments and in many situations significant changes to the underlying formulae are needed. Using GP, or other EC techniques, to automatically evolve rotation-invariant image descriptors has not been widely investigated.

- **Self-tuning GP**
  With more parameters to set, it becomes impractical to reliably and efficiently use a system as more experiments are required to find a good set of values for these parameters. The majority of existing methods have different numbers of parameters such as LBP and SIFT. Designing a system to automatically define some of the parameters required during training can potentially affect the efficiency of the system. The flexibility of GP program representation

makes this task possible, where some parameters can be incorporated into the program representation that will be optimised during the evolutionary process.

- **Transfer learning GP**
  Although existing hand-crafted image descriptors were designed to detect a specific set of keypoints that are believed to be reliable, such as corner and edges, they have been used in different domains and applications. The different domains and applications may require different sets of keypoints and features, but some of these features may also be shared amongst a variety of domains. The impact of using a program evolved to tackle a problem directly to tackle another related or not related problem has not yet been investigated deeply in GP. Adopting transfer learning in GP to transfer knowledge from one domain to another is a promising approach that requires more investigation.

Using GP to address these issues is discussed in the following five chapters of this thesis where different new algorithms are proposed.

3

GP and One-shot Learning for Binary Classification

3.1 Introduction

Image classification represents a cornerstone in a broad range of domains such as *Computer Vision* and *Pattern Recognition*. Mainly, image classification aims at categorising images into different groups based on their contents. This task has received a lot of attention over the last few decades due to its importance and difficulty. Hence, a large number of methods have been proposed in the literature that aim at tackling different aspects of the image classification task. Some of those methods aim at addressing the accuracy problem [187], whilst others try to speed-up the training process [84]. Moreover, some methods have been proposed to tackle different variations that can occur on instances of the same group such as illumination, rotation, and noise [219, 111].

The majority of those methods were not designed to build a model using a limited or small number of labelled instances. In other words, those methods

were designed based on the assumption of having an abundant (to some extent) number of instances in order to build a model that can sufficiently generalise to unseen data. However, it is not always easy, feasible, or even possible to acquire a large number of labelled instances [342]. Therefore, the limitation of having a few labelled instances to build or estimate the set of parameters of a given model needs to be addressed.

3.1.1 Chapter Goals

Motivated by the remarkable ability of the human's brain to learn a new object from only one or a few examples, two GP-based methods, namely One-shot GP and Compound-GP, are developed for the task of binary classification in images. Precisely, this chapter aims at addressing the following objectives.

- Comparing the performances of those two methods against both GP and non-GP methods using domain-specific hand-crafted features;
- Investigating the capability of the evolved programs by those two methods to handle the rotation variation;
- Studying the goodness of the detected and extracted features by the evolved programs from those two methods via testing the impact of these features on the performance of different types of classifiers;
- Investigating the efficiency of the two methods by analysing the average time required to evolve a program, average time to evaluate an instance, and the program size; and
- Investigating the interpretability of the evolved programs by those two methods.

3.1.2 Chapter Organisation

The remainder of this chapter is organised as follows. Section 3.2 describes the One-shot GP and Compound-GP methods. The experiment settings, datasets, and baseline methods are discussed in Section 3.3. The results of the experiments are

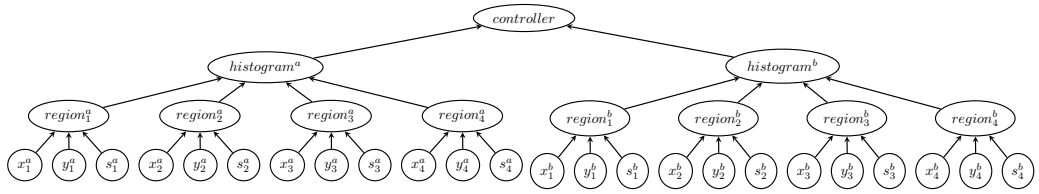


Figure 3.1: The general structure of a program evolved by the One-shot GP method.

presented in Section 3.4. Section 3.5 provides an interpretation of some typical programs evolved by the One-shot GP and Compound-GP methods. Section 3.6 concludes this chapter.

3.2 One-shot Learning GP Methods

The structures of the One-shot GP and Compound-GP methods, including the function and terminal sets, and the main components are explained in this section. This section also highlights the major similarities and differences between those two methods. In each method, the function and terminal sets are explained, followed by the fitness measure, and training and testing/evaluation procedures.

3.2.1 One-shot GP

The program evolved by this method has a static structure (e.g. type and number of nodes); however, it is dynamic in terms of the position and size of the detected regions. Figure 3.1 shows a general structure of a program evolved by the One-shot GP method.

3.2.1.1 Terminal Set

The terminal set consists of three nodes: x, y, and s as shown in Figure 3.1. The values of these nodes are randomly generated and represent a *square-shaped* window of size equals to s and centred at a pixel with the coordinates (x, y). Therefore, this part of the evolved program tree is dynamic as the values of those nodes are randomly generated. The value of the x and y coordinates cannot be negative or greater than the image width and height respectively, i.e., $0 \leq x < W - 1$ and

$0 \leq y < H - 1$, where W and H are, respectively, the width and height of the image. Moreover, the s node has been limited to be between 3 (i.e. 3×3 window) and $\min(W, H)/2$, where $\min(\cdot, \cdot)$ returns the minimum value of the arguments. The sampling windows are truncated if they exceed the boundaries of the image.

3.2.1.2 Function Set

The function set is made up of three node types each of which has its own restrictions and performs a distinct task. The first node is *controller*, which only occurs at the root of an evolved program. Thus, each program has only one node of this type. The *controller* node is responsible for predicting the class label of the instance being evaluated based on the results of its children. The second node is *histogram*, which represents the type of children for the *controller* node. The *histogram* nodes are responsible for accumulating the results of its children to form a single $\text{LBP}_{P,R}$ histogram. Each *histogram* node corresponds to a single class, i.e., the number of this type of node depends on the total number of classes. The third node is *region*. As the name suggests, each *region* node corresponds to a region of the instance being evaluated (the image), that is specified by the values of its children. The *region* nodes represent the children of the *histogram* nodes, and are responsible for performing the feature extraction task. Similar to the *histogram* nodes, the number of the *region* nodes is predefined; however, this number is not restricted by the number of the classes. In other words, this number is set empirically and in our experiments, this number has been set to four.

3.2.1.3 Fitness Function

The fitness function of the One-shot GP method aims at maximising the *between-class* distance, minimising the *within-class* distance, maximising the accuracy, and minimising the overlapping ratio of the detected regions to ensure the distinction of those regions. The fitness function for One-shot GP ($Fitness_1$) is defined as

$$Fitness_1 = \frac{D_{\text{W}} + OVR}{D_{\text{B}} + ACC_1} \tag{3.1}$$

where D_{W} and D_{B} are the within-class and between-class distances that are, respectively, defined as

$$D_{\mathrm{W}} = \sum_{\alpha \in \mathbf{N}} \sum_{\beta \in \mathbf{R}} \mathit{Dist}\left(\alpha, \beta\right), \qquad \{\alpha, \beta \mid \mathit{class}\left(\alpha\right) = \mathit{class}\left(\beta\right)\} \tag{3.2}$$

$$D_{\mathrm{B}} = \sum_{\alpha \in \mathbf{N}} \sum_{\beta \in \mathbf{R}} \mathit{Dist}\left(\alpha, \beta\right), \qquad \{\alpha, \beta \mid \mathit{class}\left(\alpha\right) \neq \mathit{class}\left(\beta\right)\} \tag{3.3}$$

where $\mathbf{R}$ and $\mathbf{N}$ are the sets of representative and non-representative instances respectively (discussed below). The function $\mathit{Dist}\left(\cdot, \cdot\right)$ measures the distance, also known as the overlapping coefficient [205], between two feature vectors of the same length as

$$\mathit{Dist}\left(\vec{u}, \vec{v}\right) = \frac{\left(\mathit{mean}\left(\vec{u}\right) - \mathit{mean}\left(\vec{v}\right)\right)^2}{\mathit{stdev}\left(\vec{u}\right) + \mathit{stdev}\left(\vec{v}\right)} \tag{3.4}$$

where $\mathit{mean}\left(\cdot\right)$ and $\mathit{stdev}\left(\cdot\right)$, respectively, calculate the mean (Equation (3.5)) and standard deviation (Equation (3.6)) of a vector of values.

$$\mathit{mean}\left(\vec{x}\right) = \frac{1}{|\vec{x}|} \sum_{e \in \vec{x}} e \tag{3.5}$$

$$\mathit{stdev}\left(\vec{x}\right) = \sqrt{\frac{1}{|\vec{x}| - 1} \sum_{e \in \vec{x}} \left(v - \mathit{mean}\left(\vec{x}\right)\right)^2} \tag{3.6}$$

The accuracy component (ACC_1) of the fitness function measures the performance of a k-NN classifier, which is defined as:

$$ACC_1 = \frac{\mathit{correct}}{|\mathbf{N}|} \tag{3.7}$$

where *correct* is the number of correctly classified instances, and $|\mathbf{N}|$ is the number of non-representative instances.

The overlapping ratio of the detected regions is measured by the OVR component of the fitness function, which is calculated as:

$$OVR = \frac{1}{\sum\limits_{g \in \mathbf{G}} \mathit{area}\left(g\right)} \sum_{i=1}^{|\mathbf{G}|-1} \sum_{j=i+1}^{|\mathbf{G}|} \mathit{Intersect}\left(g_i, g_j\right), \quad \{g_i, g_j \in \mathbf{G}\} \tag{3.8}$$

where $\mathbf{G}$ is a set of image regions each of which is a 2D array of pixel values, $\mathit{area}\left(\cdot\right)$ returns the area (number of pixels) of its argument, and the function $\mathit{Intersect}\left(\cdot, \cdot\right)$ returns the intersection (the number of shared pixels) between the arguments.

In order to prevent division by zero, the denominator of the fitness function is set to a very small value (0.0001) when both of the between-class distance and the accuracy of the k-NN classifier are zero.

It is worth noting that the fitness function presented in Equation (3.1) can be designed differently to achieve the same goal, i.e., maximising between-class distance and minimising within-class distance, such as using the subtraction operator instead of division.

The One-shot GP method uses some of the training set instances to be the basis for comparison and decision making, denoted as *representative instances* (**R**). Each representative instance is randomly selected from the training set; however, only **one** instance of each class is selected. The number of the representative instances is equal to the total number of classes. Each of the representative instances is assigned to one of the *controller* node children (i.e. the *histogram* nodes). The aim behind assigning an instance to a *histogram* node is to make this node responsible for identifying instances of only one class (i.e. identifying instances having a class label similar to that of the representative instance), which can be seen as a *one-versus-all* approach [254]. The rest of the training set instances form the *non-representative instances* (**N**) that is used to measure the performance of the evolved program during the training phase.

The training process consists of four steps. In the first step, the system iterates over the list of the content of **R** to generate the representative histograms, i.e., each *histogram* node generates a single LBP histogram relying on the detected regions by the *region* nodes and the assigned representative instance. The distance between the representative instances (D_{B}) is calculated using the generated representative histograms. In the second step, the overlapping ratio (OVR) of the detected regions is calculated using Equation (3.8). Third, the system uses the content of **N** to measure the performance of the k-NN classifier (ACC_1), and calculates the within-class distance (D_{W}). To accomplish this third step, the system generates a set of histograms for each instance (one from each of the *histogram* nodes), calculates the distance between each of the generated histograms and the corresponding representative histograms, and predicts a class label similar to that of the closest representative instance, i.e, *the-Nearest Neighbour* (1-NN) algorithm [91]. The fourth step, is to measure the goodness of the evolved program, which is achieved

via passing the calculated distances (D_{B} and D_{W}), overlapping ratio (OVR), and accuracy (ACC_1) to the fitness function. It is important to notice that at least two instances of each class are required to evolve the model, where one of them is used as a representative instance and one (or more) is used to populate the non-representative set.

3.2.1.4 The Test/Evaluation Procedure

The testing phase is handled differently from the training phase. The main concern of the evaluation phase is to test the generalisation ability of the best evolved program on unseen (i.e. test set) data. Therefore, the proportion of the correctly classified instances to the total number of instances represents the final result of this phase.

In order to classify an instance, the system generates one histogram from each of the *histogram* nodes based on the specified region by this node's *region* nodes. Then the distances between the generated histograms and the corresponding representative histograms are calculated (Equation (3.4)). The class label is predicted based on 1-NN. In other words, the class label of the closest representative histogram is assigned to the instance being evaluated.

3.2.2 Compound-GP

Clearly, the structure of the One-shot GP method is static and the number of regions is predetermined. This represents the main limitation of One-shot GP. Therefore, the Compound-GP method is introduced in order to overcome this limitation. Figure 3.2 shows a general structure of an evolved program by Compound-GP.

3.2.2.1 Terminal Set

The terminal set consists of four nodes: x, y, w, and h. Those four nodes represent a *rectangular* window of size $w \times h$ and centred at a pixel with coordinates (x, y). The values of those four nodes are positive (including zero) and randomly selected from an associated predefined interval for each of them. The intervals of the x and y coordinates are $\{0, 1, \ldots, W-1\}$ and $\{0, 1, \ldots, H-1\}$ respectively; where W and H are the image width and height respectively. On the other hand, the values

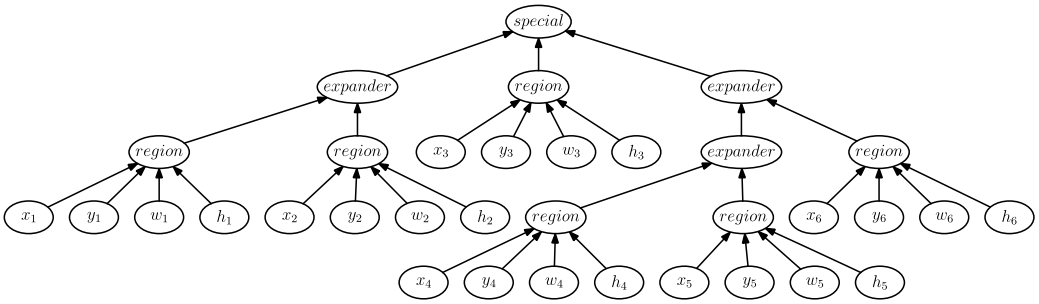


Figure 3.2: The general structure of a program evolved by the Compound-GP method.

of w and h are selected from $\{3, 4, \ldots, W\}$ and $\{3, 4, \ldots, H\}$ respectively. Like One-shot GP, the sampling windows are truncated if they exceed the boundaries of the image.

3.2.2.2 Function Set

Similar to One-shot GP, the tree evolved by the Compound-GP method consists of three types of non-terminal nodes as presented in Figure 3.2. The first node is *special*, which represents the root node similar to the *controller* node of the One-shot GP method. The main roles of this node are to generate and save a number of *patch* objects based on the results of its children, and use the generated patch objects to train a number of wrapped classifiers. The patch object is made up of the mean and standard deviation values of a histogram generated from the instances being evaluated, along with the actual class label of the instance being evaluated. Unlike the *controller* node of the One-shot GP method, the number of children of the *special* node has no relation with the number of the classes. Moreover, the children can be of different types such as *expander* and *region*. The evolved program by the One-shot GP method consists of only one wrapped classifier, whilst the evolved program by Compound-GP consists of four classifiers of two types for each child node (branch).

The second node is *expander*, which is responsible for allowing the system to evolve programs of different sizes by having chains of this node. An example is presented in Figure 3.2, where the *special* node has three children that each has

different tree size. The *expander* node does not alter the results of its children, and only passes these results to its parent node. Hence, the appearance of this node in the program tree is optional.

The third node is *region*, which resides near the leaves of the program tree. Each *region* node represents a detected region of the image. However, the number of children of this node is different in One-shot GP and Compound-GP. In the former, this node has three children (x, y, and s); while it has four children (rectangle) in the latter.

3.2.2.3 Fitness Measure

The fitness function of Compound-GP is composed of three main components as presented in Equation (3.9).

$$Fitness_2 = \frac{p\text{-}value + OVR}{ACC_2} \tag{3.9}$$

$$ACC_2 = \sum_{i=1}^{B} \left(SVM_i^{\mathbf{S}} + SVM_i^{\mathbf{L}} \right) \tag{3.10}$$

Here *p-value* is the between groups difference that is calculated using the one-way *analysis of variance* (ANOVA), OVR is the overlapping ratio between the detected regions of the image using Equation (3.8), and ACC_2 is the total performance (accuracy) of the wrapped *support vector machines* (SVM) classifiers [56] on the training set (more details below). B is the total number of children of the *special* node, which is a fixed predefined value that was empirically set to 3 in our experiments. The $SVM_i^{\mathbf{S}}$ and $SVM_i^{\mathbf{L}}$ are the i^{th} SVM classifier that is trained using the list of single and multi-patch objects respectively.

The training process of the Compound-GP method is more complicated than that of the One-shot GP method. The process consists of eight steps as depicted in Figure 3.3. In the first step, the system iterates over the set of the detected regions (*region* nodes) and calculates the ratio of overlapping (OVR) between those regions. Iterating over the instances of the training set and generating a number of LBP histograms for each instance (one from each *region* node), represents the second step. Each of those histograms is generated from a *region* node based on the specified region by the values of the four children (x, y, w and h) of that node.

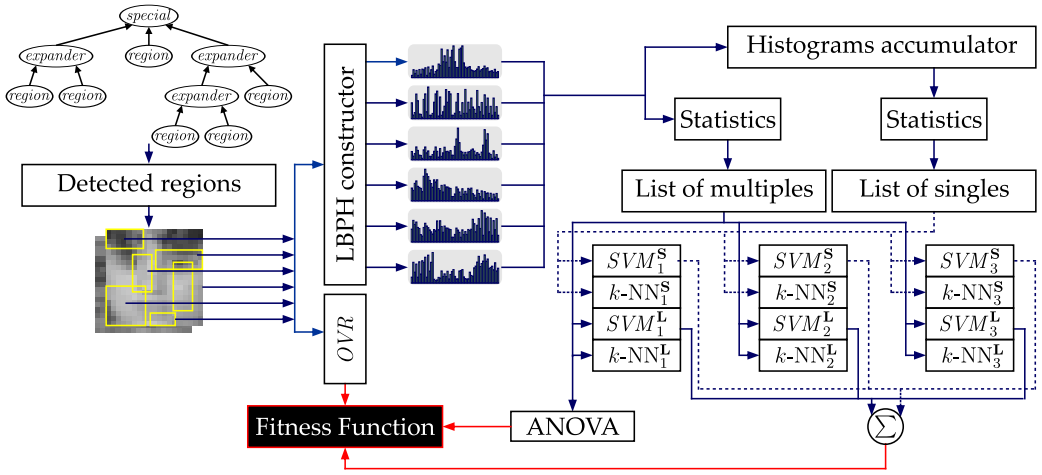


Figure 3.3: The process of calculating the fitness value of an evolved program by the Compound-GP method during the training phase.

In the third step, the statistics, i.e., mean and standard deviation, of each of those histograms are calculated, along with the actual class label of the instance being evaluated to construct a patch object, and store this patch object in the *list of multiples* that is denoted as **L** as demonstrated in Figure 3.4. In the fourth step, those histograms (generated from different *region* nodes) are concatenated with each other, the statistics of the resulted joined histogram are calculated, and a patch object is extracted using these statistics along with the actual class label of the instance being evaluated as shown in Figure 3.4. The patch object constructed in this fourth step is stored in the *list of singles* that is denoted as **S**. Therefore, the result of the third and fourth steps is $|\mathbf{G}|+1$ patch objects for each instance, where **G** is the set of detected regions. Thus, the total number of patch objects in **L** is the total number of training instances $\times$ the number of *region* nodes. Meanwhile, the **L** list consists of an equal number of patch objects to the number of instances in the training set (one object per instance). The use of both local and global features has been shown to have potential on improving the classifier performance [176, 172]. Therefore, in this study features generated from each of the detected regions (i.e. the **L** list) as well as those resulted from the combination of multiple regions (i.e. the **S** list) are used.

It is important to notice that the patch objects that were generated from each

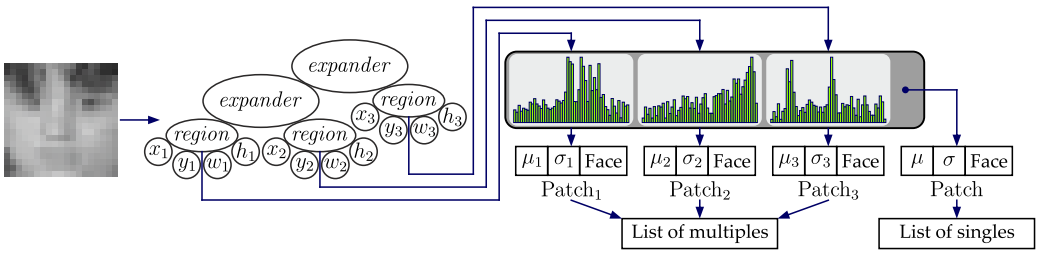


Figure 3.4: How the detected regions of an image are used to extract three *patch* objects and add each of them to the *list of multiples*, and to extract only one *patch* object from the concatenated histograms and add it to the *list of singles*.

of the *special* node children are grouped together to form the **L** and **S** lists. In other word, each of the *special* node children has one of each of those two lists (B (children) $\times$ 2 (lists)). Moreover, each of the *special* node children has four classifiers of two types: (1) two SVM classifiers; and (2) two k-NN (k=1) classifiers. Only the SVM classifiers are used during the training process, because the system is designed to evolve a program even when there is only one instance per class. The first SVM classifier is trained using the patch objects of the **S** list and denoted as $SVM^{\mathbf{S}}$. The second SVM classifier is trained using the objects of the **L** list and denoted as $SVM^{\mathbf{L}}$. Training those SVM classifiers represents the fifth step of the training procedure. In the sixth step, the performance of those SVM classifiers (ACC_2) that were trained using the lists of patches of the corresponding branch is measured using the patch objects resulted from other branches. In the seventh step, the system measures the distinction of the detected regions (*p-value*) via using the ANOVA test on the content (i.e. the mean and standard deviation values) of all **L** lists. The last step of the training process is to calculate the fitness function value using the results of the first, sixth, and seventh steps.

In summary, the results of the training phase are two trained SVM classifiers and two lists of patches for each sub-tree (branch) of the *special* node. The lists of patches will be used as the knowledge-base for the two k-NN classifiers of each sub-tree. Figure 3.3 demonstrates the process of calculating the fitness function components of an evolved program during the training phase.

3.2.2.4 The Test/Evaluation Procedure

The test phase is quite different and less complicated than the training phase. In order to test an instance, the system feeds this instance to each sub-tree of the *special* node and performs the following steps. First, the system generates an LBP histogram from each of the detect regions and uses it to construct a patch object. Second, the constructed patch objects are then fed to SVM and k-NN classifiers that were trained using the **L** list and the predicted class label of each of them is reported. Third, the generated histograms in the first step are then concatenated and used to produce a single patch object, which is then fed to SVM and k-NN classifiers that were trained using the **S** list and the predicted class labels are also reported.

After repeating the above three steps for each sub-tree of the *special* node, the system will report $B \times 4$ class labels, where B is the total number of children of the *special* node, and 4 as there are four classifiers (two SVM and two k-NN) associated with each child node. Via adopting the *voting* approach, the system will predict the class label that has the majority of the votes. However, having an even number of classifiers may result in a situation where the votes are equally divided between the two classes. Hence, such a situation has been handled by relying on the closest instances (e.g. the smallest distance measured by all k-NN classifiers).

3.3 Experiment Design

In order to test the performance of the One-shot GP and Compound-GP methods, a series of experiments have been conducted that aim at investigating different aspects. Generally, those experiments can be divided into four groups: (1) comparing the performance of the One-shot GP and Compound-GP methods with the performance of the baseline methods; (2) checking the impact of the features extracted by each of the two methods (One-shot GP and Compound-GP) on the performance of a number of classifiers compared to the use of hand-crafted and Two-tier GP [7] (see Section 3.3.3) extracted features; (3) investigating the ability of the two methods to handle the rotation variation; and (4) investigating the ability of the two methods to handle the scale variation.

This section provides more in-depth explanation of the above experiments. Moreover, the properties of the datasets that were used, parameter settings, baseline methods, and software characteristics are also discussed in this section.

3.3.1 Datasets

The performance of the One-shot GP and Compound-GP methods has been evaluated using different types of datasets. The datasets can be categorised into four groups where three of them are textures and the fourth is object classification. Each dataset in each group consists of only two classes (binary classification) of grey-scale images. The following subsections provide detailed discussion of each of those four groups.

3.3.1.1 Group A

The instances of the first group were taken from the *Kylberg Texture* dataset [157]. As discussed in Section 2.4 (page 62), the *Kylberg Texture* dataset consists of 28 classes and comes in two flavours: (1) without rotation; and (2) with rotation.

Each class of the without rotation group consists of 160 unique instances. Only eight visually-close classes of this group (textures without rotation) have been selected in this chapter to form four datasets for binary classification. *Textures-1* is the first set, made up of the *stoneslab1* and *wall1* classes. *Textures-2* is the second set, made up of the *rice2* and *sesameseeds1* classes. The *blanket1* and *canvas1* classes are selected to form the third set *Textures-3*. The fourth set is *Textures-4*, consists of the *linseds1* and *pearlsugar1* classes.

3.3.1.2 Group B

The dataset of this group was taken from *KTH-Textures under varying Illumination, Pose, and Scale* (KTH-TIPS) [39] (see Section 2.4 on page 62).

In this chapter, only two classes (sponge, and brown bread) that are visually-close of KTH-TIPS are selected to form the *Textures-5* dataset in the experiments. This dataset is more challenging than other texture datasets that were used in this study due to the scale variation of its instances which impose more difficulties on the model to handle.

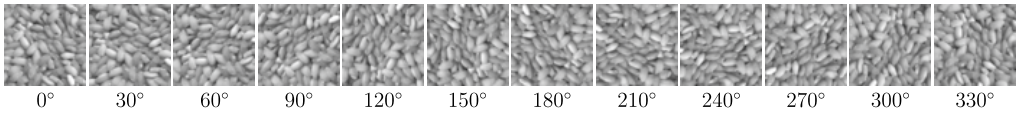


Figure 3.5: One sample from the *rice2* class of the *Kylberg Texture* dataset rotated about the center in 12 different angles taken from the *textures with rotation* group.

3.3.1.3 Group C

The instances of the third group were also taken from the Kylberg Texture dataset. However, the aim of this group's datasets is to test whether the One-shot GP and Compound-GP methods are invariant to rotation or not. Therefore, the classes of this group datasets were drawn from the textures with rotation classes of Kylberg Texture. The instances are rotated between 0° and 330° increments by 30°. An example of an instance of the *rice2* class rotated at different angles is presented in Figure 3.5. For comparison purposes, the same images that were selected from the without rotation group of the Kylberg Texture dataset to form Textures-1, Textures-2, Textures-3, and Textures-4, have been selected from the with rotation group to form *Textures-6*, *Textures-7*, *Textures-8*, and *Textures-9* datasets, respectively.

3.3.1.4 Group D

Similar to Group B, this group consists of only one dataset which is the *CBCL Faces* dataset [123]. Unlike the datasets of the previous groups, Faces is not texture-based and the task is to discriminate between face and non-face instances (see Section 2.4 on page 62). Therefore, the use of this dataset will allow us to test the ability of the One-shot GP and Compound-GP methods to handle a different task other than texture classification.

3.3.2 Datasets Preparation

Applying different image processing techniques as a preprocessing step can significantly affect the performance of the used model. Some of the well-known operations are the histogram equalisation, quantisation, and convolution operators such as

Gaussian blurring. However, different datasets require different processing schemes, and can be even harder if the instances of the same dataset were captured in an uncontrolled environment.

The total number of instances of each class has been divided equally between the training and test sets. Moreover, the original instances were used without applying any image preprocessing in order to investigate the ability of the One-shot GP and Compound-GP methods to handle the shifting of pixel values. However, each instance of the Kylberg Texture database was re-sampled (resized) to 57×57 pixels in our experiments in order to reduce the computational costs.

Apart from the Conventional-GP (see Section 3.3.3) method, all other GP methods operate on raw pixel values, and automatically detect and extract features. However, Conventional-GP and all the non-GP methods require a prior step to detect and extract feature vectors, which needs to be handled by a domain-expert in order to design highly discriminative features. Thus, the features of all texture-based datasets were extracted from ten regions [337] as depicted in Figure 3.6(a). The mean and standard deviation statistics of each of the four quadrants (AEQH, EBFQ, HQGD, and QFCG), the central quarter (IJKL), the horizontal lines (HF, and PN), the vertical lines (EG, and MO), and the entire image (ABCD) have been calculated to form a feature vector that consists of 20 values. Similarly, the mean and standard deviation of the eyes (LMFD), nose (JKON), mouth (PQSR), and the four quadrants (ABED, BCFE, DEHG, and EFIH) regions have been calculated for each of the Faces dataset instances to construct a feature vector that consists of 14 values. The regions of the faces dataset were designed based on the work of [34] as presented in Figure 3.6(b).

3.3.3 Methods for Comparison

In order to check the effectiveness of the proposed methods, a number of GP and non-GP methods have been evaluated on the used datasets.

3.3.3.1 GP-based methods

The One-shot GP and Compound-GP methods have been compared with two GP-based methods. The *Conventional-GP* is the first method that has a function

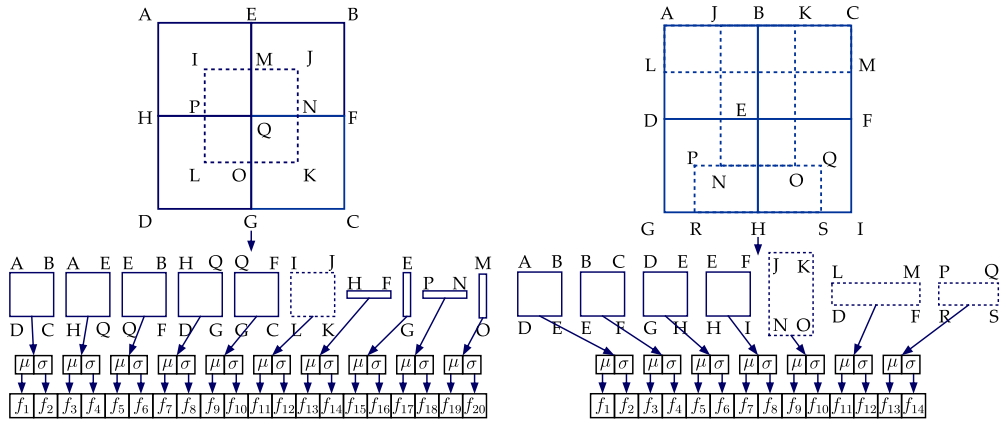


Figure 3.6: The regions of extracted features of the (a) texture, and (b) faces datasets.

set made up of the four arithmetic operators $+, -, \times$, and $/$. Those operators have their regular meaning apart from the / operator, which is protected so that returns zero if the dominator is zero. The terminal set, on the other hand, consists of *rand* which is a randomly generated double-precision float value between -1 and $+1$ (inclusive), and f_i which indicates the value of the i^{th} feature. As mentioned earlier, this method relies on domain-specific hand-crafted features. The fitness measure of Conventional-GP in both of the training and test phases is the accuracy that is formally defined as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.11}$$

where TP, TN, FP, and FN are the number of true positives, true negatives, false positives, and false negatives, respectively.

The *Two-tier GP* [7] method is made up of two tiers each of which has been designed to perform a specific task. The upper part of the program's tree (first tier) represents the classification part that consists of the four arithmetic operators (similar to Conventional-GP) and `if-then-else`. Unlike other operators, the `if-then-else` operator has three children, which returns the value of the second child if the value of the first child is negative, and the value of the third child otherwise. The second tier, which occupies the lower part of the evolved program's

tree, represents the aggregation part that consists of special nodes to perform the feature extraction task. Hence, the aggregation part comprises of *AggMin*, *AggMax*, *AggMed*, *AggMean* and *AggStdev*, which return the minimum, maximum, median, mean and standard deviation value, respectively. Each of these five aggregation functions takes five arguments that are the image being evaluated (*image*), x and y coordinates ($(xCord, yCord)$) of the upper-left corner of a region, the shape of the region (*shape*), and the size of the region (*size*). Similar to Conventional-GP, the accuracy is used as a fitness function in Two-tier GP as presented in Equation (3.11). Unlike the Conventional-GP method, the Two-tier GP method operates directly on the image raw pixel values and no preprocessing is required.

In both of the Conventional-GP and Two-tier GP methods, the program output space is divided into two parts each of which corresponds to a group of instances of the same class label: negative; and positive including the zero value. An instance is classified as belonging to a group (e.g. foreground) if the value of the root node is negative; otherwise, it is classified as belonging to another group (e.g. background).

3.3.3.2 Non-GP methods

The developed methods are compared to six non-GP methods.

- Support Vector Machines (SVM) [56]: A non-probabilistic linear classifier that is also known as *support-vector networks*. SVMs are very popular in machine learning and have been adopted in diverse applications in computer vision. Mostly, SVM uses an associated learning algorithm to analyse data and extract or recognise patterns. Therefore, the *Sequential Minimal Optimisation* (SOM) [237] algorithm is used in this study to train SVMs.

- Naïve Bayes (NB) [137]: A broadly used and easy to implement probabilistic method which builds a model by adopting Bayes' theorem. Since the 1950s, NB has been studied and used extensively as a baseline method [260].

- Adaptive Boosting (AdaBoost) [97]: A meta-algorithm that was introduced by Freund *et al.* in 1996 [97] designed to be used in conjunction with other machine learning algorithms to enhance their performance. The overall idea of AdaBoost is to adaptively build a model via tweaking those instances misclassified by previous models to improve the subsequent ones.

- KStar (K*) [54]: An instance-based method that predicts a class label for an instance by considering the class label of the similar instances in the training set, i.e., similar to k-NN [91]. The key difference between K* and k-NN is that the former measures the distance between instances using an entropy-based distance measure.

- Non-Nested Generalized (NNge) [191]: Also an instance-based classifier that considers the class label of the closest instances in the training set to the one being evaluated, but using a non-nested exemplar [333].

- Naïve Bayes/Decision Tree (NBTree) [149]: A hybrid method that combines both decision trees (DT) and Naïve Bayes to inherit the features of the two methods. The leaf nodes of the DT are populated using NB classifiers.

For more details regarding these methods, see [323].

3.3.4 Experiments

To evaluate each of the four GP-based methods (Conventional-GP, Two-tier GP, One-shot GP and Compound-GP), a specified number of instances of each class are randomly selected from the total number of instances available in the training set. The best evolved program using the selected instances at the end of the run is tested against the unseen data (test set). Due to the stochastic nature of GP, the same process has been independently executed 50 times using a different *starting point* (seed value) each time, and only the average performance is reported. The non-GP methods on the other hand, have been trained using the exact same instances, but without repeating the execution multiple times (deterministic methods).

Moreover, due to the impact of the selected instances on the performance of the evolved classifier, the 50 runs of each GP-based method and the single execution of the non-GP methods, have been repeated 20 times using different *instances* each time. Therefore, the total number of independent runs on a single dataset is $\big((4\text{ (GP methods)} \times 50\text{ (runs)} \times 20\text{ (repetitions)}) + (6\text{ (non-GP methods)} \times 1\text{ (run)} \times 20\text{ (repetitions)})\big) = 4,120$. The standard deviation over the 20 repetitions (i.e. 20 average performances) is reported.

The above procedure has further been repeated ten times using training sets of different *sizes* that starts from the minimum number (only one instance per class) and increases by one instance every time (the largest is ten instances per class). However, the smallest number of instances per class that can be used to evolve a program by One-shot GP is two (one representative and one or more non-representative); therefore, in the case of this method the above process has been repeated nine times instead of ten. This ten repetitions makes the total number of executions on each dataset is $\big((1,000 \text{ (runs)} \times 3 \text{ (methods)} \times 10 \text{ (sets)}) + (1,000 \text{ (runs)} \times 1 \text{ (method)} \times 9 \text{ (sets)})$ (GP methods) $+ (120 \text{ (run)} \times 10 \text{ (sets)})$ (non-GP methods)$\big) = 40,200$.

3.3.5 Feature Extraction

The two new GP methods have their own mechanisms to perform feature extraction; hence, these methods can also be used for automatic feature extraction. The impact of the detected and extracted features by each of the One-shot GP and Compound-GP methods on the performance of the six non-GP baseline classifiers is also investigated. To measure the goodness of the extracted features by each of the One-shot GP and Compound-GP methods, the hand-crafted features (as discussed in Section 3.3.2) and those extracted by Two-tier GP (as discussed in [6]) are used.

In the case of the One-shot GP method, the mean and standard deviation values are calculated for each LBP histogram resulted from each *histogram* node. As discussed in Section 3.2.1, each evolved program has two *histogram* nodes; therefore, the feature vector of each instance consists of four values.

Similarly, the features extracted by the Compound-GP method represents the calculated statistics (mean and standard deviation) of the resulted LBP histograms. However, the program evolved by Compound-GP generates a number of histograms for each instance. Moreover, some histograms are generated from the *region* nodes; whilst others result from the concatenation of histograms as described in Section 3.2.2. In other words, the patch objects of the **S** (list of singles) and **L** (list of multiples) are used as the extracted features. As it is hard to guess which of the two lists is better than the other, the two lists are used individually, as well as the combination of the two.

Table 3.1: The GP Parameters of all experiments

Parameter	Value	Conventional-GP	Two-tier GP	One-shot GP	Compound-GP
Crossover Rate	0.80	✓	✓	✓	✓
Mutation Rate	0.20	✓	✓	✓	✓
Elitism	Keep the best	✓	✓	✓	✓
Population size	200	✓	✓	✓	✓
Generations	20	✓	✓	✓	✓
Tree depth	2-10	✓	✓	✗	✓
Selection Type	Tournament	✓	✓	✓	✓
Tournament Size	7	✓	✓	✓	✓
Initial Population	Ramped half-and-half	✓	✓	✗	✓

3.3.6 Parameter Settings

In this study, four GP methods have been used. For comparison purposes, the settings of those methods' parameters are kept identical in all of the experiments as listed in Table 3.1. It is very important to notice that some of the parameters are not applicable in the case of One-shot GP due to the restrictions of the evolved program by this method. Moreover, the conventional GP crossover and mutation operators are used and the closure property is maintained by strongly-typed GP.

3.3.7 Implementation

The GP-based methods have been implemented using the platform provided by the *Evolutionary Computation Java-based* (ECJ) package version 23 [190]. The implementation of the non-GP methods have been taken from the *Waikato Environment for Knowledge Analysis* (WEKA) package version 3.8 [116].

3.4 Results and Discussions

The results of the experiments are reported and discussed in this section. This section is divided into four subsections that describe different aspects of the obtained results. The performances of the One-shot GP, Compound-GP, and all the baseline methods in terms of accuracy are presented in the first subsection. The goodness of the features extracted by each of the One-shot GP and Compound-GP methods

is compared to the hand-crafted areas and those extracted by the Two-tier GP in the second subsection. The third subsection, provides discussions on both of the training and test time of the four GP-based methods. Finally, the fourth subsection shows the complexity of the evolved program in terms of the average size per generation of the four GP-based methods.

3.4.1 Accuracy

In order to check if the average performances of One-shot GP and Compound-GP are statistically significant compared to the performance of each of the baseline methods, a *Wilcoxon signed-rank* test [320, 66] is used. The methods are compared in pairs, and the significance level of the test was set to 5%. The $\downarrow$ and $\uparrow$ symbols indicate that the performance of the One-shot GP method compared to that of the other method is significantly worse and better, respectively. Meanwhile, the $\Downarrow$ and $\Uparrow$ symbols appear if the performance of the Compound-GP method compared to that of the other method is significantly worse and better respectively. The method with the highest performance amongst all other comparative methods has its result made **bold**. However, in the case of having more than one method that have achieved 100% accuracy, none of those methods are made bold.

The results of this experiment are presented in tables in this section. Each table aggregates the results of datasets of one group. The first column of each table shows the name of the dataset, and the total number of instances per class that were used in the training set are listed in the second column. Horizontally, each table is divided into two parts: the first lists the results of the non-GP methods; whilst the results of the GP-based methods are listed under the second part. The result of One-shot GP is missing from the first row of all tables (indicated by N/A), as the minimum number of instances required by this method is two of each class.

3.4.1.1 Group A Datasets

The results of the datasets of this group are presented in Table 3.2, which consists of the Textures-1, Textures-2, Textures-3, and Textures-4 datasets.

The first block of Table 3.2 shows the results of the methods on the Textures-1 dataset. Compound-GP has scored second best performance after the AdaBoost

method. The One-shot GP method is also showing good performance and scores third when the number of instances is less than six.

The results of Textures-2 dataset are presented in the second block of Table 3.2. This dataset represents one of the easiest; that most of the methods, apart from Conventional-GP and Two-tier GP, have scored above 99% when there were four or more instances of each class in the training set. However, the use of only one instance was enough to achieve 100% accuracy by the Compound-GP method, and 99.9% using two instances in the case of One-shot GP.

For Textures-3, all of the methods, apart from Two-tier GP, have achieved reasonably good accuracy above 80% when there were three or more instances of each class in the training set as shown in the third block of Table 3.2. Moreover, the Compound-GP method shows either the highest or in the top three ranked performance amongst other methods. Although One-shot GP shows the second lowest performance amongst the comparative methods on this dataset, the result shows that this method has achieved on average 79.3% accuracy using only two instances per class.

The results on Textures-4 dataset are presented in the last block of Table 3.2. Apart from K*, Conventional-GP, and Two-tier GP, all other methods have achieved on average over 80% accuracy. Moreover, Compound-GP has significantly outperformed all other methods on this dataset with 95.1% accuracy using only one instance per class, which is significantly better than the highest achieved results by other methods even when there were 10 instances per class in the training set (NNge with 90.8% accuracy).

3.4.1.2 Group B Dataset

The results of the Textures-5 dataset are presented in Table 3.3. This dataset represents a more challenging task compared to all other texture-based datasets due to the variation in illumination, scale, and pose of its instances. The results show that One-shot GP and Compound-GP have significantly outperformed all other methods on this dataset. Moreover, these two methods have achieved on average over 90% accuracy even when there are two instances in the training set. This shows that the programs evolved by the two new methods are invariant (to some extent) to those variations.

Table 3.2: Results of the Group A datasets.

		Non-GP Methods						GP-based Methods			
	Size	AdaBoost	K*	NB	NBTree	NNge	SVM	Conventional	Two-tier	One-shot	Compound
Textures-1	1	**99.4 ± 0.5** ⇓	61.1 ± 7.1 ⇑	57.0 ± 5.8 ⇑	63.7 ± 8.3 ⇑	52.5 ± 6.9 ⇑	52.5 ± 6.9 ⇑	54.1 ± 3.5 ⇑	51.0 ± 0.8 ⇑	N/A	96.5 ± 1.7
	2	**99.4 ± 0.3**↓⇓	67.3 ± 12.6↑⇑	68.7 ± 15.1↑⇑	61.3 ± 12.2↑⇑	63.4 ± 11.5↑⇑	62.6 ± 11.4↑⇑	55.4 ± 3.8↑⇑	51.9 ± 1.1↑⇑	90.7 ± 1.3	97.1 ± 1.0
	3	**99.4 ± 0.3**↓⇓	71.1 ± 12.3↑⇑	75.8 ± 17.7↑⇑	68.0 ± 18.2↑⇑	74.0 ± 15.8↑⇑	73.0 ± 14.4↑⇑	57.1 ± 3.7↑⇑	52.4 ± 1.1↑⇑	90.6 ± 1.1	97.5 ± 0.6
	4	**99.4 ± 0.3**↓⇓	70.0 ± 11.0↑⇑	82.2 ± 17.6↑⇑	79.5 ± 19.6↑⇑	83.3 ± 16.3↑⇑	80.0 ± 12.7↑⇑	58.3 ± 3.9↑⇑	53.2 ± 1.4↑⇑	91.0 ± 0.8	97.6 ± 0.4
	5	**99.4 ± 0.3**↓⇓	70.0 ± 8.2 ↑⇑	86.5 ± 15.8↑⇑	77.7 ± 17.4↑⇑	89.3 ± 14.9↑⇑	86.4 ± 11.8↑⇑	59.1 ± 3.8↑⇑	53.4 ± 1.2↑⇑	92.1 ± 0.5	97.6 ± 0.4
	6	**99.4 ± 0.3**↓⇓	69.0 ± 6.4 ↑⇑	90.8 ± 12.8↑⇑	90.1 ± 12.8↑⇑	94.5 ± 9.7 ↓⇑	89.8 ± 8.2 ↑⇑	60.2 ± 3.6↑⇑	53.2 ± 1.2↑⇑	92.9 ± 0.4	97.6 ± 0.3
	7	**99.4 ± 0.2**↓⇓	68.9 ± 5.1 ↑⇑	95.0 ± 7.9 ↓⇑	89.8 ± 9.0 ↑⇑	97.7 ± 2.6 ↓	94.0 ± 4.6 ↓⇑	60.7 ± 3.3↑⇑	53.7 ± 1.2↑⇑	93.2 ± 0.6	97.7 ± 0.2
	8	**99.4 ± 0.1**↓⇓	69.0 ± 4.4 ↑⇑	97.5 ± 1.6 ↓	88.0 ± 8.1 ↑⇑	98.9 ± 1.3 ↓⇓	94.7 ± 3.7 ↓⇑	61.2 ± 2.5↑⇑	53.9 ± 1.2↑⇑	93.4 ± 0.5	97.7 ± 0.2
	9	**99.4 ± 0.1**↓⇓	69.3 ± 4.2 ↑⇑	97.2 ± 1.8 ↓⇑	88.0 ± 9.6 ↑⇑	99.1 ± 0.9 ↓⇓	93.9 ± 4.4 ⇑	61.1 ± 2.3↑⇑	54.5 ± 0.9↑⇑	93.7 ± 0.5	97.7 ± 0.1
	10	**99.4 ± 0.1**↓⇓	70.8 ± 4.6 ↑⇑	96.3 ± 3.3 ↓⇓	88.4 ± 8.8 ↑⇑	99.1 ± 0.8 ↓⇓	94.4 ± 3.9 ↓⇑	61.1 ± 2.9↑⇑	54.5 ± 1.1↑⇑	94.0 ± 0.4	97.7 ± 0.1
Textures-2	1	52.3 ± 11.3 ⇑	97.4 ± 1.9 ⇑	74.7 ± 7.8 ⇑	96.3 ± 2.9 ⇑	69.7 ± 9.8 ⇑	69.7 ± 9.8 ⇑	53.8 ± 2.2 ⇑	50.3 ± 0.6 ⇑	N/A	**100 ± 0.0**
	2	79.6 ± 25.6↑⇑	98.2 ± 2.2↑⇑	92.1 ± 9.6↑⇑	99.8 ± 0.6↑⇑	95.0 ± 9.5↑⇑	95.6 ± 6.9↑⇑	59.7 ± 4.0↑⇑	51.1 ± 0.8↑⇑	100 ± 0.0	**100 ± 0.0**
	3	92.3 ± 18.9↑⇑	99.8 ± 0.3↑⇑	95.9 ± 6.6↑⇑	100 ± 0.0↓	99.1 ± 2.2↑⇑	98.8 ± 1.3↑⇑	65.6 ± 5.7↑⇑	51.7 ± 0.9↑⇑	100 ± 0.0	100 ± 0.0
	4	100 ± 0.0 ↓	100 ± 0.1 ⇑	99.1 ± 2.0↑⇑	100 ± 0.1 ⇑	99.9 ± 0.2↑⇑	99.5 ± 1.1↑⇑	70.1 ± 5.7↑⇑	52.2 ± 0.7↑⇑	100 ± 0.0	100 ± 0.0
	5	100 ± 0.0 ↓	100 ± 0.1 ⇑	99.3 ± 2.5↑⇑	100 ± 0.0↓	100 ± 0.0↓	99.8 ± 0.4↑⇑	73.0 ± 5.0↑⇑	52.3 ± 0.7↑⇑	100 ± 0.0	100 ± 0.0
	6	100 ± 0.0 ↓	100 ± 0.1 ⇑	99.8 ± 0.6↑⇑	100 ± 0.0↓	100 ± 0.0↓	99.9 ± 0.3↑⇑	74.7 ± 4.3↑⇑	52.4 ± 0.9↑⇑	100 ± 0.0	100 ± 0.0
	7	100 ± 0.0 ↓	100 ± 0.0↓	99.4 ± 2.0↑⇑	100 ± 0.0↓	100 ± 0.0↓	99.9 ± 0.3↑⇑	77.5 ± 3.3↑⇑	52.6 ± 1.1↑⇑	100 ± 0.0	100 ± 0.0
	8	100 ± 0.0 ↓	100 ± 0.0↓	99.5 ± 1.7↑⇑	100 ± 0.0↓	100 ± 0.0↓	99.9 ± 0.2↑⇑	78.0 ± 3.6↑⇑	53.0 ± 1.0↑⇑	100 ± 0.0	100 ± 0.0
	9	100 ± 0.0 ↓	100 ± 0.0↓	99.7 ± 0.7↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.1 ⇑	78.1 ± 3.2↑⇑	53.0 ± 0.9↑⇑	100 ± 0.0	100 ± 0.0
	10	100 ± 0.0 ↓	100 ± 0.0↓	99.9 ± 0.2↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓	78.7 ± 3.8↑⇑	53.6 ± 0.8↑⇑	100 ± 0.0	100 ± 0.0
Textures-3	1	57.7 ± 24.9 ⇑	**90.8 ± 4.2**	74.8 ± 9.8 ⇑	90.8 ± 4.5	66.3 ± 14.2 ⇑	66.3 ± 14.2 ⇑	67.4 ± 4.5 ⇑	51.0 ± 1.1 ⇑	N/A	89.4 ± 8.7
	2	83.6 ± 16.7↓⇑	90.8 ± 4.8↓⇑	85.3 ± 9.7↓⇑	91.9 ± 3.5↓	90.3 ± 8.1 ↓⇑	87.7 ± 10.3↓⇑	80.4 ± 4.9 ⇑	53.0 ± 1.2↑⇑	79.3 ± 3.1	**92.0 ± 2.3**
	3	89.0 ± 4.6 ↓⇑	93.0 ± 3.4↓	87.2 ± 7.5↓⇑	91.5 ± 3.1↓	**93.3 ± 3.5** ↓⇓	92.8 ± 4.9 ↓	87.0 ± 2.7↓⇑	54.7 ± 1.6↑⇑	79.3 ± 2.4	92.2 ± 2.3
	4	90.3 ± 2.1 ↓⇑	93.0 ± 2.9↓	89.1 ± 6.3↓⇑	92.1 ± 2.1↓	93.0 ± 3.3 ↓	**94.3 ± 3.3** ↓⇓	89.2 ± 1.6↓⇑	55.8 ± 1.7↑⇑	80.4 ± 2.1	92.2 ± 2.1
	5	90.3 ± 2.2 ↓⇑	93.0 ± 3.0↓	89.4 ± 4.6↓⇑	92.3 ± 2.4↓	92.3 ± 2.7 ↓	**94.8 ± 2.1** ↓⇓	90.5 ± 0.9↓⇑	56.7 ± 1.8↑⇑	81.9 ± 1.7	92.4 ± 1.8
	6	90.4 ± 2.2 ↓⇑	93.6 ± 2.2↓⇓	89.2 ± 4.1↓⇑	91.6 ± 2.6↓	92.2 ± 2.8 ↓	**94.5 ± 2.3** ↓⇓	90.7 ± 0.9↓⇑	58.1 ± 1.4↑⇑	82.8 ± 1.2	92.3 ± 1.8
	7	90.3 ± 2.3 ↓⇑	93.8 ± 1.8↓⇓	88.0 ± 4.4↓⇑	91.7 ± 2.5↓	92.0 ± 2.6 ↓	**94.1 ± 2.6** ↓⇓	91.0 ± 0.9↓⇑	58.7 ± 1.5↑⇑	84.1 ± 1.4	92.4 ± 1.7
	8	90.5 ± 2.5 ↓⇑	93.8 ± 1.9↓⇓	88.9 ± 3.5↓⇑	92.4 ± 2.3↓	91.9 ± 2.5 ↓	**93.8 ± 2.3** ↓⇓	91.2 ± 1.1↓⇑	59.1 ± 1.4↑⇑	84.1 ± 1.3	92.4 ± 1.6
	9	90.7 ± 2.6 ↓⇑	**93.8 ± 2.0**↓⇓	89.1 ± 3.6↓⇑	92.3 ± 2.2↓	91.9 ± 2.1 ↓	93.6 ± 2.2 ↓⇓	91.4 ± 1.1↓⇑	60.1 ± 1.1↑⇑	84.6 ± 1.5	92.5 ± 1.6
	10	90.6 ± 2.6 ↓⇑	**93.6 ± 2.0**↓⇓	89.6 ± 3.2↓⇑	92.4 ± 2.4↓	92.0 ± 2.0 ↓⇑	93.0 ± 2.6 ↓	91.4 ± 1.1↓⇑	60.7 ± 1.0↑⇑	85.0 ± 1.6	92.8 ± 1.4
Textures-4	1	53.7 ± 7.0 ⇑	72.2 ± 7.5 ⇑	67.3 ± 7.0 ⇑	73.3 ± 7.2 ⇑	57.7 ± 16.6 ⇑	57.7 ± 16.6 ⇑	50.4 ± 1.3 ⇑	50.0 ± 0.5 ⇑	N/A	**95.1 ± 3.2**
	2	79.1 ± 13.6↑⇑	74.3 ± 7.4↑⇑	66.8 ± 8.4↑⇑	80.5 ± 6.6 ⇑	78.8 ± 8.0 ↑⇑	76.9 ± 8.2 ↑⇑	51.5 ± 1.6↑⇑	50.3 ± 0.5↑⇑	81.9 ± 4.1	**95.7 ± 2.3**
	3	87.6 ± 8.5 ↓⇑	76.0 ± 5.9↑⇑	74.8 ± 7.1↑⇑	84.0 ± 6.8 ↓⇑	81.2 ± 7.5 ⇑	81.2 ± 7.6 ⇑	52.0 ± 1.7↑⇑	50.5 ± 0.5↑⇑	79.6 ± 5.4	**96.0 ± 2.4**
	4	87.5 ± 8.1 ↓⇑	77.2 ± 6.1↑⇑	79.6 ± 7.9 ⇑	76.4 ± 13.3↑⇑	84.0 ± 7.2 ↓⇑	82.6 ± 6.8 ↓⇑	52.5 ± 1.6↑⇑	50.4 ± 0.6↑⇑	79.6 ± 5.3	**96.1 ± 2.1**
	5	87.5 ± 8.7 ↓⇑	76.8 ± 6.1↑⇑	83.8 ± 6.5↓⇑	84.2 ± 5.1 ↓⇑	85.5 ± 6.1 ↓⇑	85.5 ± 5.7 ↓⇑	52.7 ± 1.6↑⇑	50.3 ± 0.4↑⇑	81.7 ± 3.6	**97.1 ± 1.2**
	6	87.8 ± 6.5 ↓⇑	77.5 ± 6.4↑⇑	83.2 ± 6.3 ⇑	80.8 ± 12.0↑⇑	87.0 ± 5.8 ↓⇑	86.6 ± 4.8 ↓⇑	53.5 ± 1.7↑⇑	50.6 ± 0.5↑⇑	82.9 ± 3.7	**97.3 ± 0.9**
	7	87.5 ± 5.5 ↓⇑	77.6 ± 6.3↑⇑	83.4 ± 4.9 ⇑	86.3 ± 5.5 ↓⇑	89.5 ± 4.6 ↓⇑	87.7 ± 4.4 ↓⇑	54.4 ± 2.1↑⇑	50.7 ± 0.5↑⇑	84.4 ± 3.0	**97.6 ± 0.6**
	8	88.4 ± 4.1 ↓⇑	77.6 ± 6.9↑⇑	83.1 ± 4.6↑⇑	87.7 ± 4.6 ↓⇑	89.9 ± 4.2 ↓⇑	87.2 ± 3.5 ↓⇑	54.8 ± 1.2↑⇑	50.6 ± 0.5↑⇑	85.5 ± 2.7	**97.7 ± 0.6**
	9	88.3 ± 3.7 ↓⇑	77.4 ± 6.5↑⇑	84.1 ± 4.4↑⇑	86.6 ± 4.4 ⇑	90.6 ± 4.2 ↓⇑	87.9 ± 4.4 ↓⇑	55.1 ± 1.7↑⇑	50.6 ± 0.5↑⇑	86.4 ± 2.3	**97.7 ± 0.5**
	10	87.5 ± 4.0 ⇑	77.6 ± 6.1↑⇑	84.7 ± 3.7↑⇑	87.4 ± 4.7 ⇑	90.8 ± 4.0 ↓⇑	87.5 ± 5.0 ⇑	55.0 ± 2.0↑⇑	50.7 ± 0.6↑⇑	87.1 ± 1.7	**97.9 ± 0.4**

Table 3.3: Results of the group B dataset.

		Non-GP Methods						GP-based Methods			
	Size	AdaBoost	K*	NB	NBTree	NNge	SVM	Conventional	Two-tier	One-shot	Compound
Textures-5	1	70.3 ± 24.6 ⇑	66.2 ± 10.7 ⇑	67.9 ± 11.8 ⇑	63.7 ± 12.5 ⇑	64.9 ± 20.1 ⇑	64.9 ± 20.1 ⇑	58.5 ± 5.0 ⇑	49.8 ± 1.0 ⇑	N/A	**91.4 ± 14.3**
	2	73.4 ± 17.3↑⇑	71.5 ± 12.4↑⇑	71.5 ± 13.5↑⇑	69.8 ± 14.6↑⇑	73.4 ± 14.8↑⇑	69.0 ± 16.1↑⇑	55.9 ± 6.8↑⇑	51.1 ± 1.6↑⇑	91.3 ± 8.8	**91.6 ± 13.1**
	3	73.5 ± 15.1↑⇑	72.0 ± 14.2↑⇑	81.5 ± 10.3↑⇑	62.8 ± 15.7↑⇑	77.0 ± 13.1↑⇑	69.1 ± 18.1↑⇑	54.5 ± 6.4↑⇑	51.9 ± 1.7↑⇑	**91.4 ± 8.2**	91.2 ± 12.2
	4	74.9 ± 15.4↑⇑	72.3 ± 14.3↑⇑	83.9 ± 9.20↑⇑	62.0 ± 16.0↑⇑	76.0 ± 13.2↑⇑	69.7 ± 17.8↑⇑	54.9 ± 6.8↑⇑	52.2 ± 1.6↑⇑	**91.3 ± 7.6**	90.7 ± 11.7
	5	76.7 ± 16.2↑⇑	72.4 ± 14.0↑⇑	85.1 ± 7.8 ↑⇑	69.2 ± 17.2↑⇑	76.2 ± 12.5↑⇑	70.6 ± 17.4↑⇑	55.3 ± 6.6↑⇑	52.0 ± 2.2↑⇑	91.4 ± 7.1	**91.8 ± 10.6**
	6	76.8 ± 15.4↑⇑	72.8 ± 13.1↑⇑	85.2 ± 8.2 ↑⇑	69.2 ± 17.9↑⇑	77.6 ± 13.2↑⇑	71.5 ± 17.7↑⇑	54.8 ± 6.2↑⇑	52.5 ± 2.5↑⇑	**92.3 ± 6.7**	91.9 ± 10.6
	7	76.6 ± 15.4↑⇑	74.3 ± 13.1↑⇑	87.0 ± 6.5 ↑⇑	73.7 ± 17.0↑⇑	82.9 ± 7.9 ↑⇑	73.7 ± 16.9↑⇑	54.0 ± 6.8↑⇑	53.1 ± 2.2↑⇑	**92.7 ± 6.2**	92.5 ± 9.9
	8	77.2 ± 16.0↑⇑	75.4 ± 12.1↑⇑	87.2 ± 6.2 ↑⇑	76.0 ± 16.7↑⇑	84.3 ± 8.0 ↑⇑	74.6 ± 17.2↑⇑	52.9 ± 6.8↑⇑	53.2 ± 2.1↑⇑	**93.2 ± 6.3**	93.1 ± 9.2
	9	77.7 ± 16.3↑⇑	77.3 ± 11.6↑⇑	87.9 ± 4.3 ↑⇑	75.4 ± 17.3↑⇑	85.2 ± 7.9 ↑⇑	75.9 ± 16.3↑⇑	53.0 ± 6.8↑⇑	53.2 ± 2.1↑⇑	**93.6 ± 5.3**	93.5 ± 8.5
	10	77.7 ± 15.3↑⇑	77.6 ± 11.2↑⇑	88.5 ± 4.5 ↑⇑	78.2 ± 15.2↑⇑	85.4 ± 8.0 ↑⇑	77.6 ± 14.9↑⇑	52.0 ± 6.7↑⇑	53.3 ± 2.3↑⇑	94.0 ± 4.7	**94.1 ± 7.9**

3.4.1.3 Group C Datasets

Table 3.4 presents the results on datasets of the third group, which consists of Textures-6, Textures-7, Textures-8, and Textures-9 datasets.

The results of Textures-6 dataset (which represents the rotated version of Textures-1) show that Compound-GP and One-shot GP have, respectively, scored the first and third best performance as shown in the first block of Table 3.4. The two new methods have significantly outperformed all other methods, apart from AdaBoost compared to One-shot GP, on this dataset. Moreover, apart from K*, all comparative methods show a significant drop in their performances compared to Textures-1; whilst the two new methods show nearly consistent performance on the two datasets.

Apart from AdaBoost, most of the methods have achieved similar accuracy on Textures-7 to Textures-2 as presented in the second block of Table 3.4. One-shot GP and Compound-GP have achieved 100% accuracy even when the number of available instances is relatively small (less than three instances per class).

Similarly, the comparative methods show nearly consistent or slightly dropped performance on the rotated version Textures-8 of the Textures-3 dataset as presented in the third block of Table 3.4.

The last block of Table 3.4 shows the results of Textures-9, which represents the rotated version of the Textures-4 dataset. While Compound-GP has maintained its performance, the performance of the One-shot GP method has greatly dropped compared to Textures-4. Similar to Compound-GP, other methods have also shown a nearly consistent performance on this dataset. Noticeably, AdaBoost shows a considerably better performance on this dataset compared to the performance of this method on Textures-4.

3.4.1.4 Group D Dataset

Table 3.5 shows the results of the experiment on the Faces dataset. Both of One-shot GP and Compound-GP have achieved better accuracy than all other methods on this dataset when the number of instances is smaller than four per class. Meanwhile, the Two-tier GP and NB start to compete when the number of training instances increases. Apart from NB, Compound-GP has significantly

Table 3.4: Results of the group C datasets.

		Non-GP Methods						GP-based Methods			
	Size	AdaBoost	K*	NB	NBTree	NNge	SVM	Conventional	Two-tier	One-shot	Compound
Textures-6	1	93.6 ± 5.4 ⇑	61.9 ± 5.6 ⇑	53.1 ± 5.0 ⇑	61.5 ± 6.9 ⇑	49.5 ± 4.0 ⇑	49.5 ± 4.0 ⇑	50.4 ± 0.8 ⇑	50.2 ± 0.2 ⇑	N/A	**98.2 ± 1.0**
	2	94.2 ± 5.4 ⇑	67.7 ± 7.9↑⇑	65.2 ± 7.2↑⇑	63.4 ± 6.4 ↑⇑	55.4 ± 4.3↑⇑	55.1 ± 4.3↑⇑	50.7 ± 0.8↑⇑	50.3 ± 0.3↑⇑	93.7 ± 1.0	**98.7 ± 0.2**
	3	94.7 ± 5.3↓⇑	70.5 ± 4.8↑⇑	63.4 ± 7.6↑⇑	62.4 ± 7.2 ↑⇑	57.9 ± 3.7↑⇑	59.2 ± 5.2↑⇑	50.9 ± 0.8↑⇑	50.4 ± 0.2↑⇑	93.3 ± 0.7	**98.8 ± 0.2**
	4	95.2 ± 5.2↓⇑	72.4 ± 7.5↑⇑	62.2 ± 7.9↑⇑	65.0 ± 11.2↑⇑	59.1 ± 3.6↑⇑	60.2 ± 5.8↑⇑	51.5 ± 0.9↑⇑	50.5 ± 0.3↑⇑	93.5 ± 0.8	**98.8 ± 0.2**
	5	95.7 ± 5.1↓⇑	72.3 ± 6.0↑⇑	59.3 ± 6.4↑⇑	65.0 ± 11.1↑⇑	61.7 ± 4.0↑⇑	61.0 ± 5.8↑⇑	51.5 ± 1.2↑⇑	50.5 ± 0.4↑⇑	94.4 ± 0.4	**98.8 ± 0.2**
	6	96.3 ± 4.9↓⇑	73.4 ± 7.7↑⇑	56.8 ± 2.3↑⇑	65.8 ± 13.7↑⇑	64.7 ± 6.4↑⇑	64.0 ± 7.3↑⇑	51.5 ± 1.2↑⇑	50.6 ± 0.4↑⇑	95.0 ± 0.2	**98.8 ± 0.1**
	7	96.8 ± 4.6↓⇑	75.6 ± 7.1↑⇑	58.1 ± 2.7↑⇑	67.0 ± 14.3↑⇑	67.0 ± 6.5↑⇑	64.3 ± 6.8↑⇑	51.4 ± 1.5↑⇑	50.7 ± 0.3↑⇑	95.3 ± 0.3	**98.8 ± 0.1**
	8	97.3 ± 4.3↓⇑	78.7 ± 8.9↑⇑	58.5 ± 3.1↑⇑	67.5 ± 14.6↑⇑	69.5 ± 8.9↑⇑	66.0 ± 8.2↑⇑	51.5 ± 1.8↑⇑	50.6 ± 0.4↑⇑	95.5 ± 0.2	**98.8 ± 0.1**
	9	97.8 ± 3.8↓⇑	79.0 ± 9.1↑⇑	58.9 ± 3.2↑⇑	63.7 ± 16.8↑⇑	70.7 ± 10.0↑⇑	66.0 ± 7.6↑⇑	51.5 ± 1.8↑⇑	50.8 ± 0.4↑⇑	95.8 ± 0.3	**98.8 ± 0.1**
	10	98.3 ± 3.2↓⇑	81.3 ± 7.4↑⇑	60.2 ± 3.7↑⇑	65.6 ± 18.8↑⇑	70.6 ± 9.0↑⇑	66.4 ± 7.1↑⇑	51.5 ± 1.9↑⇑	51.0 ± 0.3↑⇑	96.0 ± 0.3	**98.8 ± 0.1**
Textures-7	1	49.6 ± 0.8 ⇑	97.6 ± 2.4 ⇑	73.3 ± 7.5 ⇑	97.8 ± 4.0 ⇑	66.5 ± 9.4 ⇑	66.5 ± 9.4 ⇑	56.8 ± 2.2 ⇑	51.1 ± 0.4 ⇑	N/A	**100 ± 0.0**
	2	67.6 ± 24.4↑⇑	99.2 ± 0.7↑⇑	99.1 ± 2.4↑⇑	99.9 ± 0.3↑⇑	97.6 ± 3.4↑⇑	93.2 ± 9.3↑⇑	66.7 ± 6.5↑⇑	51.4 ± 0.4↑⇑	100 ± 0.0	**100 ± 0.0**
	3	77.8 ± 25.2↑⇑	99.9 ± 0.2↑⇑	99.5 ± 0.9↑⇑	100 ± 0.0↓	100 ± 0.1 ⇑	99.9 ± 0.3↑⇑	76.1 ± 4.3↑⇑	52.1 ± 0.7↑⇑	100 ± 0.0	100 ± 0.0
	4	80.3 ± 24.8↑⇑	100 ± 0.0↓⇑	99.8 ± 0.3↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓⇑	79.6 ± 2.0↑⇑	52.5 ± 0.7↑⇑	100 ± 0.0	100 ± 0.0
	5	82.7 ± 24.2↑⇑	100 ± 0.0↓	99.9 ± 0.2↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓	82.0 ± 2.0↑⇑	53.1 ± 0.8↑⇑	100 ± 0.0	100 ± 0.0
	6	85.2 ± 23.2↑⇑	100 ± 0.0↓	99.1 ± 4.1↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓	82.7 ± 2.3↑⇑	53.2 ± 0.8↑⇑	100 ± 0.0	100 ± 0.0
	7	87.7 ± 21.9↑⇑	100 ± 0.0↓	99.8 ± 0.6↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓	83.8 ± 2.6↑⇑	53.6 ± 0.7↑⇑	100 ± 0.0	100 ± 0.0
	8	90.1 ± 20.3↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↓	85.1 ± 2.4↑⇑	54.1 ± 0.7↑⇑	100 ± 0.0	100 ± 0.0
	9	92.6 ± 18.1↑⇑	100 ± 0.0↓	100 ± 0.0↓	100 ± 0.0↑⇑	100 ± 0.0↓	100 ± 0.0↓	86.3 ± 2.3↑⇑	54.4 ± 0.6↑⇑	100 ± 0.0	100 ± 0.0
	10	95.1 ± 15.2↑⇑	100 ± 0.0	100 ± 0.0	100 ± 0.0↑⇑	100 ± 0.0	100 ± 0.0	86.7 ± 2.6↑⇑	54.8 ± 0.8↑⇑	100 ± 0.0	100 ± 0.0
Textures-8	1	57.7 ± 30.3 ⇑	**89.0 ± 4.7** ⇓	67.5 ± 8.8 ⇑	87.2 ± 6.2 ⇓	69.6 ± 12.5 ⇑	69.6 ± 12.5 ⇑	70.0 ± 2.1 ⇑	50.9 ± 0.5 ⇑	N/A	80.7 ± 8.1
	2	77.1 ± 26.4↓⇑	90.0 ± 3.7↓⇓	**91.0 ± 5.2**↓⇓	90.8 ± 4.6↓⇓	78.2 ± 9.9 ↓⇑	79.9 ± 11.7↓⇑	78.5 ± 2.8↓⇑	52.2 ± 0.7↑⇑	74.4 ± 4.7	84.4 ± 5.4
	3	**93.6 ± 1.4** ↓⇓	92.2 ± 3.4↓⇓	93.0 ± 2.5↓⇓	92.5 ± 1.6↓⇓	85.7 ± 6.7 ↓	85.4 ± 11.0↓	84.9 ± 2.5↓	53.4 ± 0.6↑⇑	75.4 ± 2.9	86.5 ± 4.6
	4	**93.7 ± 1.4** ↓⇓	92.7 ± 3.3↓⇓	93.3 ± 0.9↓⇓	93.5 ± 1.4↓⇓	89.3 ± 6.4 ↓⇓	88.0 ± 10.0 ↓	87.4 ± 1.3↓	54.1 ± 0.8↑⇑	75.9 ± 2.7	86.3 ± 4.3
	5	**93.8 ± 1.3** ↓⇓	93.0 ± 2.9↓⇓	92.4 ± 2.4↓⇓	93.7 ± 1.5↓⇓	91.1 ± 5.8 ↓⇓	89.7 ± 8.6 ↓⇓	88.7 ± 1.4↓⇓	55.0 ± 1.1↑⇑	77.2 ± 2.7	86.8 ± 2.4
	6	93.8 ± 1.3 ↓⇓	93.6 ± 2.6↓⇓	91.1 ± 3.5↓⇓	**94.0 ± 1.3**↓⇓	93.1 ± 4.9 ↓⇓	91.4 ± 7.4 ↓⇓	89.0 ± 1.4↓⇓	55.9 ± 1.1↑⇑	79.0 ± 2.3	87.8 ± 1.8
	7	93.9 ± 1.2 ↓⇓	93.7 ± 2.5↓⇓	93.0 ± 1.7↓⇓	94.1 ± 1.0↓⇓	**94.3 ± 3.9** ↓⇓	92.8 ± 6.4 ↓⇓	89.7 ± 1.1↓⇓	56.7 ± 1.1↑⇑	80.2 ± 2.6	87.1 ± 1.8
	8	93.9 ± 1.2 ↓⇓	93.9 ± 2.3↓⇓	92.6 ± 1.7↓⇓	94.0 ± 1.2↓⇓	**94.9 ± 3.7** ↓⇓	94.2 ± 5.1 ↓⇓	90.0 ± 1.0↓⇓	57.3 ± 0.9↑⇑	80.8 ± 2.3	87.4 ± 1.5
	9	93.9 ± 1.1 ↓⇓	94.1 ± 2.0↓⇓	89.1 ± 4.0↓⇓	94.3 ± 1.0↓⇓	95.4 ± 3.4 ↓⇓	**95.4 ± 3.8** ↓⇓	90.6 ± 1.0↓⇓	57.7 ± 1.1↑⇑	81.8 ± 2.2	87.5 ± 1.2
	10	94.0 ± 1.1 ↓⇓	94.1 ± 1.9↓⇓	91.1 ± 2.3↓⇓	94.4 ± 0.8↓⇓	95.9 ± 3.0 ↓⇓	**96.0 ± 3.3** ↓⇓	90.7 ± 0.9↓⇓	58.2 ± 0.8↑⇑	82.3 ± 1.9	87.5 ± 1.4
Textures-9	1	48.5 ± 9.4 ⇑	83.1 ± 5.7 ⇑	76.1 ± 6.9 ⇑	80.5 ± 6.1 ⇑	70.6 ± 15.2 ⇑	70.6 ± 15.2 ⇑	50.4 ± 1.0 ⇑	50.0 ± 0.3 ⇑	N/A	**90.5 ± 5.0**
	2	91.4 ± 14.0↓	83.8 ± 6.9↓⇑	71.5 ± 8.6 ⇑	86.1 ± 4.3↓⇑	90.3 ± 3.2 ↓⇑	90.1 ± 2.1 ↓⇑	51.3 ± 1.1↑⇑	50.1 ± 0.3↑⇑	71.4 ± 6.4	**92.0 ± 2.5**
	3	**97.0 ± 0.8**↓⇓	89.7 ± 2.9↓⇑	78.6 ± 10.1↓⇑	85.1 ± 8.0↓⇑	92.3 ± 3.3 ↓	92.1 ± 1.9 ↓	52.8 ± 2.1↑⇑	50.4 ± 0.2↑⇑	69.6 ± 5.5	92.4 ± 2.0
	4	**97.0 ± 0.8**↓⇓	91.3 ± 3.4↓⇑	71.6 ± 8.5 ↓⇑	89.7 ± 7.5↓⇑	94.0 ± 3.2 ↓⇓	92.9 ± 1.9 ↓	54.4 ± 2.7↑⇑	50.5 ± 0.4↑⇑	69.0 ± 4.6	92.3 ± 1.5
	5	**97.1 ± 0.9**↓⇓	92.2 ± 2.0↓⇑	74.4 ± 11.1↓⇑	91.1 ± 6.1↓⇑	94.9 ± 2.3 ↓⇓	93.5 ± 1.7 ↓	56.3 ± 3.5↑⇑	50.5 ± 0.4↑⇑	71.1 ± 3.9	93.0 ± 1.6
	6	**97.2 ± 0.9**↓⇓	93.2 ± 2.3↓	78.7 ± 9.7 ↓⇑	93.4 ± 3.2↓	95.5 ± 1.5 ↓⇓	94.1 ± 1.4 ↓⇓	57.6 ± 3.8↑⇑	50.7 ± 0.4↑⇑	73.3 ± 2.9	93.1 ± 1.5
	7	**97.2 ± 1.0**↓⇓	93.0 ± 2.6↓	81.4 ± 9.7 ↓⇑	92.9 ± 4.2↓	95.8 ± 1.4 ↓⇓	94.6 ± 1.4 ↓⇓	58.2 ± 3.1↑⇑	50.7 ± 0.5↑⇑	75.8 ± 2.8	93.3 ± 1.5
	8	**97.3 ± 1.0**↓⇓	93.1 ± 1.9↓	84.1 ± 9.5 ↓⇑	93.3 ± 4.1↓	96.2 ± 1.1 ↓⇓	94.7 ± 1.5 ↓⇓	59.6 ± 3.3↑⇑	50.7 ± 0.6↑⇑	77.4 ± 2.7	93.6 ± 1.5
	9	**97.4 ± 1.0**↓⇓	93.2 ± 1.8↓	84.8 ± 11.9↓⇑	93.8 ± 3.0↓	96.4 ± 0.9 ↓⇓	95.2 ± 1.2 ↓⇓	59.4 ± 3.3↑⇑	50.8 ± 0.5↑⇑	79.4 ± 2.8	93.5 ± 1.2
	10	**97.4 ± 1.0**↓⇓	93.2 ± 2.0↓	86.6 ± 9.8 ↓⇑	94.4 ± 2.3↓⇓	96.4 ± 0.9 ↓⇓	95.6 ± 1.0 ↓⇓	60.4 ± 3.3↓⇑	50.9 ± 0.4↓⇑	80.4 ± 2.8	93.6 ± 1.1

outperformed all other methods on this dataset. The One-shot GP method, on the other hand, has significantly outperformed NBTree, SVM, and Conventional-GP in all the nine different sizes, and some cases compared to AdaBoost, K*, NNge, and the Two-tier GP methods.

3.4.1.5 Summary

The results show that both One-shot GP and Compound-GP have successfully evolved programs using only a few instances that can generalised well to the unseen data. Compared to non-GP and GP-based methods, One-shot GP and Compound-GP have achieved significantly better or comparable results in most cases. However, in some situations the two new methods, especially One-shot GP, show significantly worse performance, e.g., on Textures-1 compared to the performance of AdaBoost.

3.4.2 Feature Extraction

As highlighted in Section 3.3, the goodness of the detected and extracted features by One-shot GP and Compound-GP has been investigated via feeding those features to six different classifiers (the non-GP methods that were used in the first experiment). The performance of those classifiers on the hand-crafted features and those extracted by the Two-tier GP method have been compared to the use of One-shot GP and Compound-GP extracted features.

The result of each of the six classifiers on each dataset using four different sets of features is represented by a single line chart. Moreover, the results of the all six classifiers are aligned on a single row for each dataset. The vertical and horizontal axes of each chart represent the average accuracy and number of instances per class in the training set, respectively.

3.4.2.1 Group A Datasets

The results on the test set of this experiment on the datasets of the first group (Textures-1, Textures-2, Textures-3 and Textures-4) are presented in Figure 3.7.

Apart from AdaBoost, the use of One-shot GP and Compound-GP extracted features show significant improvement in the performance of the classifiers when there are fewer than six instances per class in the training set.

Table 3.5: Results of the group D dataset.

		Non-GP Methods						GP-based Methods			
	Size	AdaBoost	K*	NB	NBTree	NNge	SVM	Conventional	Two-tier	One-shot	Compound
Faces	1	56.1 ± 8.2 ⇑	61.2 ± 9.0 ⇑	57.3 ± 7.6 ⇑	61.7 ± 9.3 ⇑	57.6 ± 9.0 ⇑	57.7 ± 9.2 ⇑	55.2 ± 2.7 ⇑	55.9 ± 1.9 ⇑	N/A	**67.8 ± 10.0**
	2	55.4 ± 10.0↑⇑	60.8 ± 9.2↑⇑	64.9 ± 8.8	59.1 ± 11.6↑⇑	61.4 ± 11.8↑⇑	62.8 ± 12.3↑⇑	57.6 ± 3.4↑⇑	60.9 ± 2.9↑⇑	67.6 ± 6.3	**68.6 ± 9.7**
	3	57.6 ± 9.6 ↑⇑	63.6 ± 9.0↑⇑	67.4 ± 11.0 ⇑	58.3 ± 11.4↑⇑	64.2 ± 12.9↑⇑	64.9 ± 12.3↑⇑	58.6 ± 3.3↑⇑	63.2 ± 2.7↑⇑	68.1 ± 4.6	**70.4 ± 5.2**
	4	63.3 ± 10.8↑⇑	65.3 ± 8.5↑⇑	**71.3 ± 10.2**↓	57.6 ± 10.7↑⇑	64.8 ± 13.1↑⇑	63.4 ± 16.0↑⇑	59.6 ± 3.4↑⇑	65.1 ± 2.6↑⇑	68.1 ± 4.9	70.6 ± 5.4
	5	64.5 ± 11.4↑⇑	67.0 ± 8.2 ⇑	**73.2 ± 10.2**↓	65.0 ± 10.8↑⇑	66.2 ± 12.3↑⇑	63.9 ± 12.1↑⇑	60.7 ± 4.7↑⇑	66.7 ± 2.5↑⇑	68.6 ± 4.1	72.8 ± 2.5
	6	66.9 ± 11.1↑⇑	68.8 ± 7.6 ⇑	**73.6 ± 9.2** ↓	60.0 ± 11.2↑⇑	68.2 ± 11.6 ⇑	63.6 ± 11.5↑⇑	61.5 ± 4.5↑⇑	68.1 ± 2.0 ⇑	69.3 ± 3.7	73.5 ± 2.3
	7	66.3 ± 11.2↑⇑	69.6 ± 7.5 ⇑	**75.4 ± 7.6** ↓	63.8 ± 9.2 ↑⇑	69.0 ± 12.3 ⇑	61.8 ± 12.5↑⇑	61.7 ± 4.8↑⇑	69.2 ± 1.3 ⇑	69.4 ± 3.7	74.9 ± 2.6
	8	68.6 ± 10.8 ⇑	70.1 ± 7.3 ⇑	**76.9 ± 6.9** ↓⇓	66.1 ± 9.3 ↑⇑	70.4 ± 11.2 ⇑	62.6 ± 12.8↑⇑	61.9 ± 4.6↑⇑	70.2 ± 1.7 ⇑	70.0 ± 3.1	75.4 ± 2.1
	9	69.8 ± 8.7 ⇑	70.3 ± 7.1 ⇑	**77.2 ± 6.7** ↓⇓	61.4 ± 8.9 ↑⇑	71.8 ± 9.9 ↓⇑	62.3 ± 10.8↑⇑	62.0 ± 4.2↑⇑	71.1 ± 1.6 ⇑	70.4 ± 2.5	76.0 ± 1.7
	10	72.2 ± 7.7 ↓⇑	70.3 ± 6.9 ⇑	**77.5 ± 6.9** ↓⇓	66.0 ± 9.5 ↑⇑	73.2 ± 9.3 ↓⇑	62.7 ± 9.7 ↑⇑	62.6 ± 3.6↑⇑	71.1 ± 1.7 ⇑	70.8 ± 2.4	76.6 ± 1.3

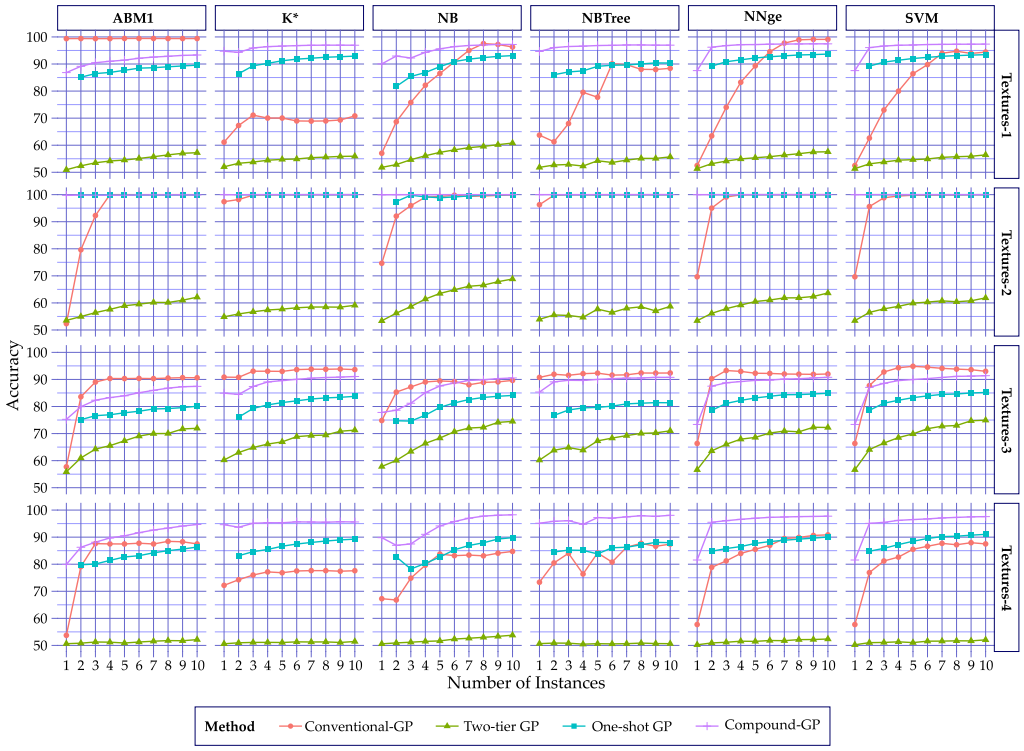


Figure 3.7: The average performance of the experimented methods on the datasets of Group A using four different sets of features.

On Textures-2, on the other hand, all of the six classifiers have achieved perfect performance using the features extracted by the One-shot GP and Compound-GP methods. Apart from K* and NBTree, it is clear that the new features have large positive impact on the performance of the other four classifiers when there are less than four instances per class in the training set.

The results of the experiment on Textures-3 show that the features of Compound-GP has slightly dropped the performance of all the six methods, whilst the features of One-shot GP show large drop in the performance of those classifiers. However, all the six classifiers have achieved significantly better performance using the features of the two new methods compared to those extracted by the Two-tier GP.

The features extracted by One-shot GP show improvement in the performance of all the classifiers apart from AdaBoost with hand-crafted features. The features extracted by the Compound-GP method, on the other hand, show positive impact

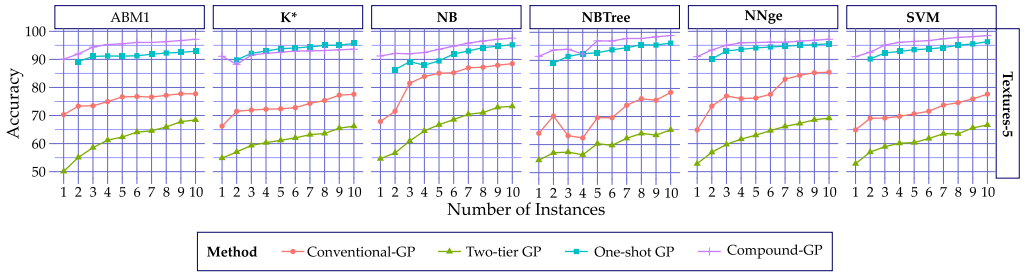


Figure 3.8: The average performance of the experimented methods on the Texture-5 dataset using four different sets of features.

and have significantly improved the performance of all the six classifiers over the use of both the hand-crafted (conventional GP) and Two-tier features.

3.4.2.2 Group B Dataset

On the Textures-5 dataset, the features extracted by One-shot GP and Compound-GP have significantly improved the performance of all the six classifiers over both hand-crafted and Two-tier GP features as presented in Figure 3.8. As highlighted in Section 3.3, this dataset are more challenging than other texture-based datasets due to illumination, scale, and pose variations of its instances.

3.4.2.3 Group C Datasets

As depicted in Figure 3.9, The results show that the six classifiers have maintained their performances when the features of One-shot GP and Compound-GP are used on Textures-6 compared to Textures-1. Meanwhile, apart from AdaBoost, the use of hand-crafted features have significant negative impact on the performance of those classifiers.

The features extracted by both One-shot GP and Compound-GP show insensitivity to rotation on Textures-7. A nearly consistent performance has been achieved by all those six classifiers when the hand-crafted and Two-tier features are used.

Comparing the performance achieved of the six classifiers on Textures-8 and Textures-3, the classifiers show consistent or slightly improved performance when the hand-crafted features are used; while the Compound-GP features have dropped

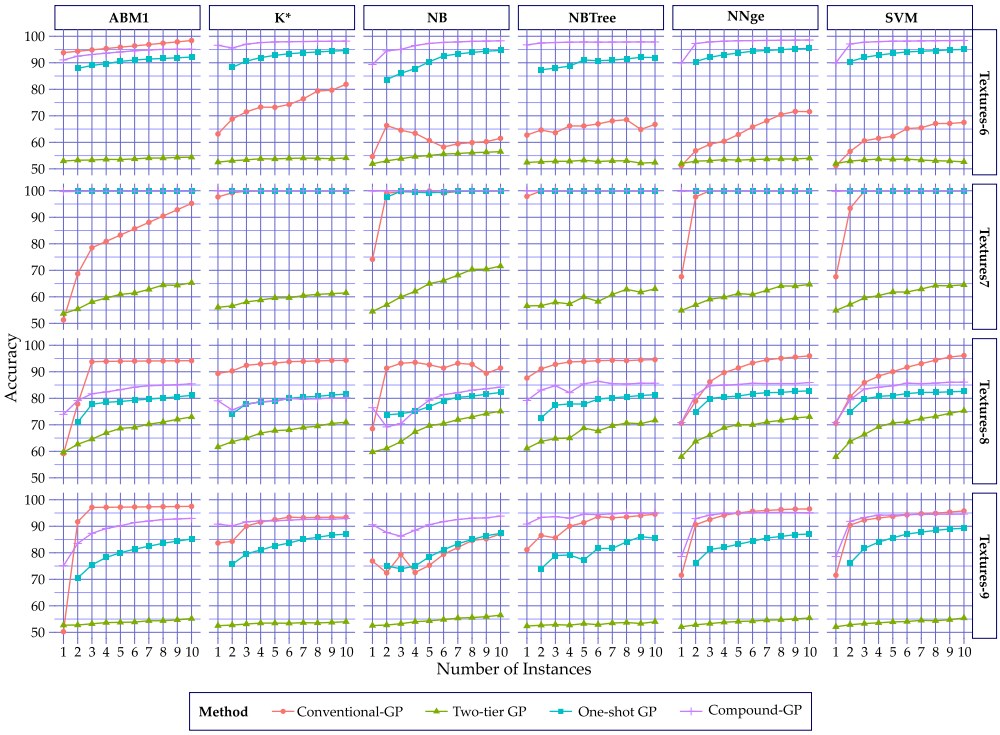


Figure 3.9: The average performance of the experimented methods on the datasets of Group C using four different sets of features.

the performance of almost all the six classifiers. The features extracted by One-shot GP, on the other hand, show similar behaviour to the hand-crafted features; however, the gap between the two is significant. The inconsistency in the performance that the Compound-GP features show was expected due to having the same behaviour in the results of the first experiment on this dataset.

On Textures-9, compared to Textures-4, the six classifiers have maintained their performances when the One-shot GP and Compound-GP extracted features were used. The hand-crafted features show positive improvement in the performance of all the six classifiers compared to that achieved on Textures-4.

3.4.2.4 Group D Dataset

Apart from NB, the features from the One-shot GP and Compound-GP methods have improved the performance of the other five classifiers on this dataset as shown

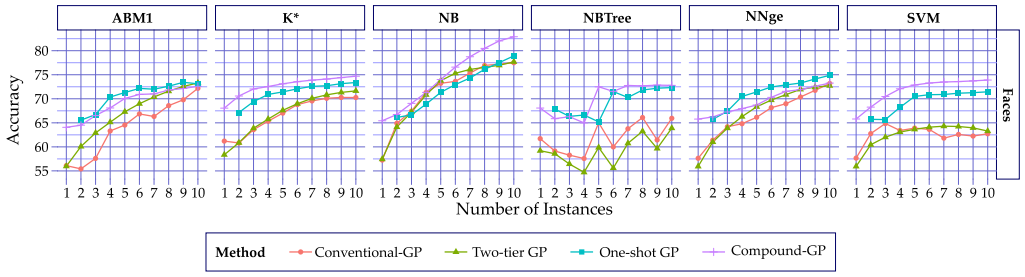


Figure 3.10: The average performance of the experimented methods on the Faces dataset using four different sets of features.

in Figure 3.10. Although the Two-tier GP extracted features show improvement in the performance of those classifiers, this improvement is still not as good as those introduced by the two new methods (One-shot GP and Compound-GP). The hand-crafted features show, in most cases, the worst performance on this dataset.

3.4.2.5 Summary

The results of this experiment show that the features extracted by the two new methods have improved the performance of the six classifiers in most cases. However, those features are also showing an inconsistency in other cases especially Textures-3 and its rotated version Textures-8. Most importantly, the features detected by the One-shot-GP and Compound-GP methods are not biased to a specific classifier, and are invariant to rotation, illumination, and scale variations to some extent.

3.4.3 Training and Testing Time

The average training and testing times are also measured in order to highlight the cost of evolving a program by each of the One-shot GP and Compound-GP methods. The results are presented in blocks of line charts, each of which shows the average time required to evolve or evaluate a program by each of the four GP methods. Each row of blocks represents either the training or testing phase, while each column presents the results of a single dataset as presented in Figure 3.11(a), Figure 3.11(b), Figure 3.11(c), and Figure 3.11(d), for the datasets of Group A, Group B, Group C, and Group D, respectively. Similar to the plots of the previous

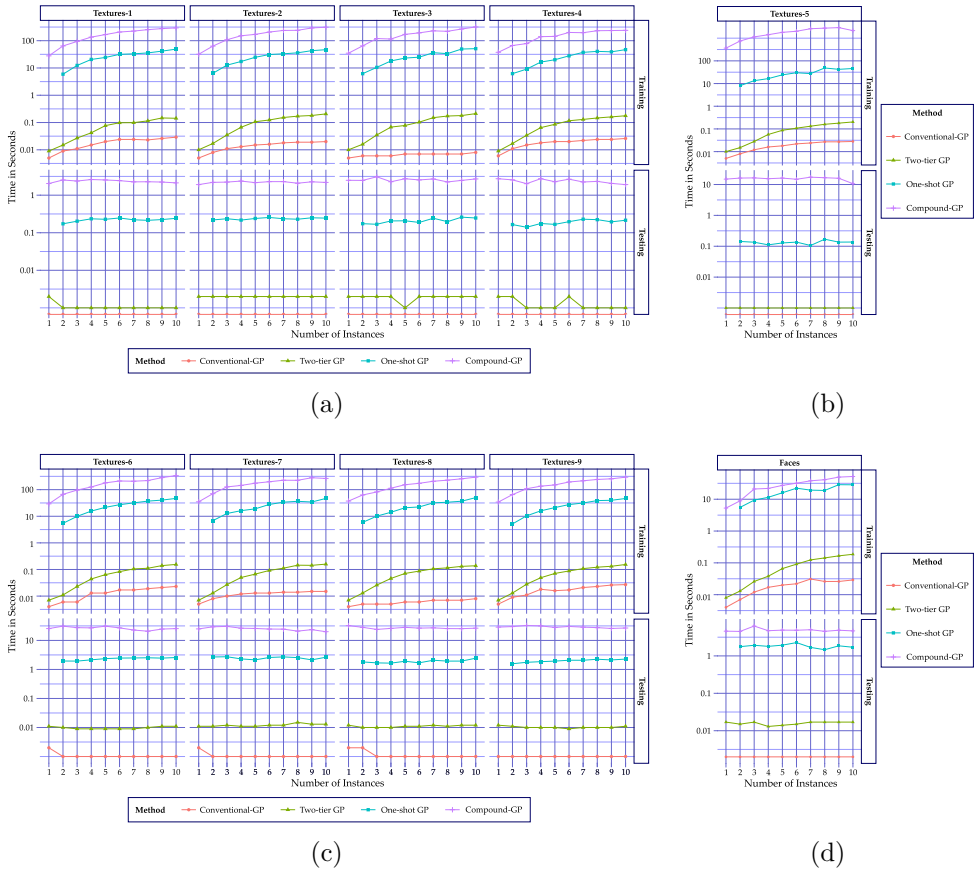


Figure 3.11: The average training and test time of the four GP-based methods on the (a) Group A; (b) Group B; (c) Group C; and (d) Group D datasets.

section, the horizontal axes represents the number of instances per class in the training set; and the vertical axes represents the time in seconds.

The results show that both of the One-shot GP and Compound-GP methods take significantly longer time to evolve and evaluate/test a program than Conventional-GP and Two-tier GP. Moreover, Conventional-GP is the fastest method amongst the four GP methods, followed by the Two-tier GP with a slightly slower speed. This large gap between the two new and baseline methods was expected due to the following reasons:

- **Feature detection and extraction:** Apart from Conventional-GP, the other three GP-based methods perform feature detection and extraction

at the lower part (near the leaves) of the evolved program. Performing those operations increases the complexity of the evolved program in terms of memory and computation costs. Moreover, in the case of the Two-tier GP, there are only four simple operations (minimum, maximum, mean, and standard deviation) that can be used to perform feature extraction. However, this operation is more complicated in the case of both One-shot GP and Compound-GP methods. The complication results from the calculation of the LBP code of each pixel in each of the detected regions, which require applying a threshold operation, checking if the calculated code is uniform or not, and accumulate it with other codes to form a histogram.

- **Wrapped classifiers:** In the case of both Conventional-GP and Two-tier GP, the aim is to evolve a GP-based classifier. Thus, both of these methods do not have any wrapped classifier that needs to be trained. Meanwhile, One-shot GP and Compound-GP have different numbers and types of wrapped classifiers. The One-shot GP method uses a simple k-NN classifier; whereas Compound-GP consists of two SVM and two k-NN classifiers for each of the *special* node (the root of the evolved program's tree) children. Training and evaluating those wrapped classifiers introduces extra complexities that need to be handled.

- **Fitness function:** The fitness functions of both Conventional-GP and Two-tier GP are simple and does not require extra calculations. However, the fitness functions of both One-shot GP and Compound-GP are complex and require calculating more parameters such as the overlapping ratio of the detected regions (OVR).

- **Termination criteria:** In the case of the One-shot GP and Compound-GP, the system is forced to proceed until the maximum number of generations is reached. Meanwhile, the Conventional-GP and Two-tier GP can terminate once an *ideal* program has been found.

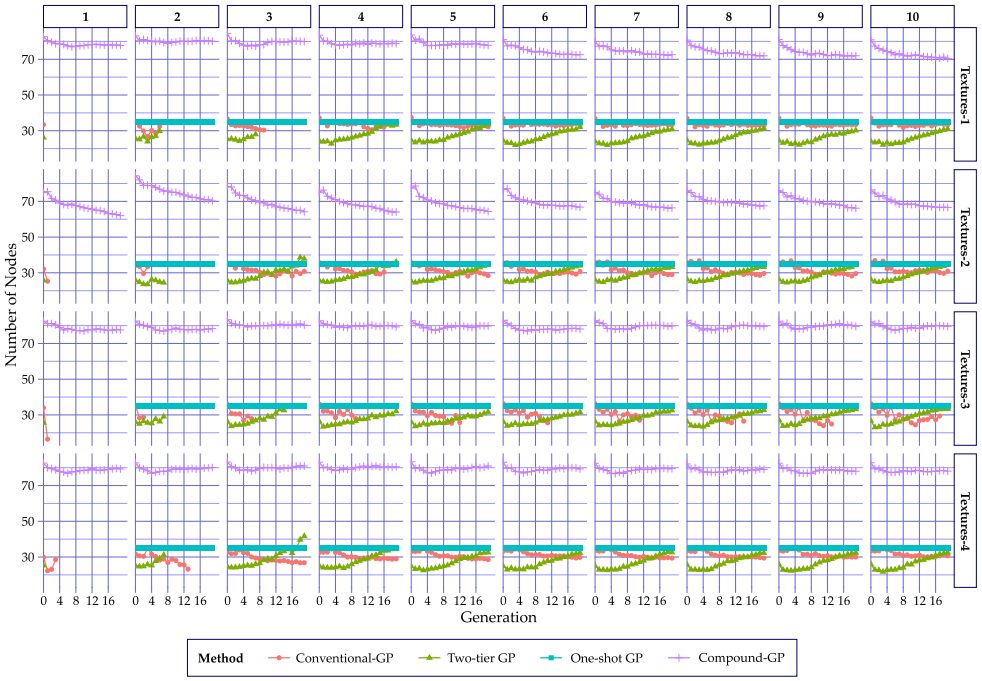


Figure 3.12: The average program size per generation of the four GP-based methods on the datasets of Group A.

3.4.4 Program Size

Here, the complexity of the program evolved by each of the four GP-based methods in terms of average program size is investigated. The results are presented in line plots, where each row corresponds to one of the datasets, and each column represents the number of instances per class in the training set as presented in Figures 3.12 to 3.15. The vertical axes and horizontal axes represent the number of nodes and generations, respectively. Each block contains four lines one for each of the four GP methods.

The results of Conventional-GP and Two-tier GP show that when there are fewer than four instances per class, these two methods terminate early and before the maximum number of generations is reached. This was expected as both of these methods rely on the accuracy alone as a goodness measure. However, this does not occur in One-shot GP and Compound-GP due to the other components of the fitness measures of these two methods that force the system to proceed.

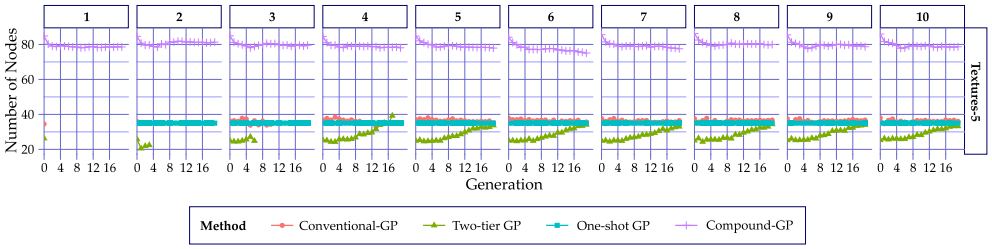


Figure 3.13: The average program size per generation of the four GP-based methods on the dataset of Group B.

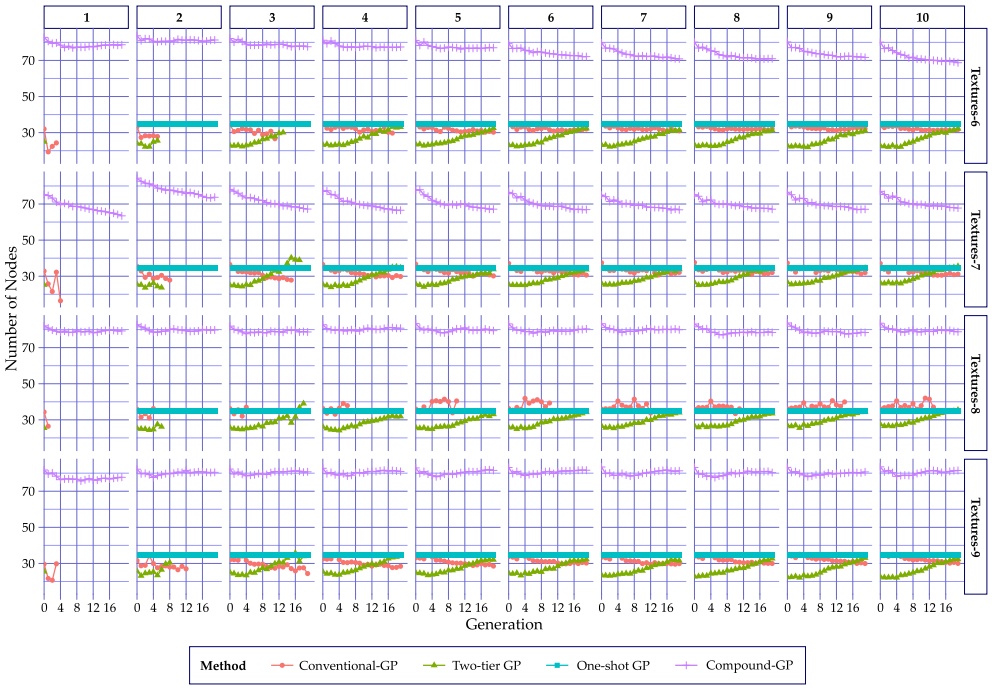


Figure 3.14: The average program size per generation of the four GP-based methods on the datasets of Group C.

The results show that One-shot GP has a constant program size that is neither affected by the size of the training set nor by the generation (progress of the run). This was expected due to the restriction of the program size of the One-shot GP method. The size is 35 nodes that are: one *controller*; two *histogram*; eight *region*; and twenty four terminal nodes.

The Compound-GP method, on the other hand, evolves programs of different

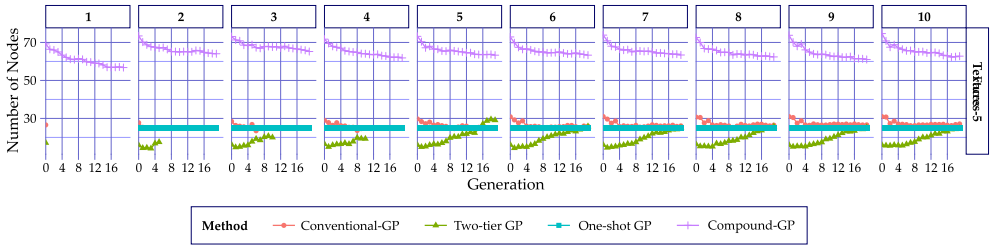


Figure 3.15: The average program size per generation of the four GP-based methods on the dataset of Group D.

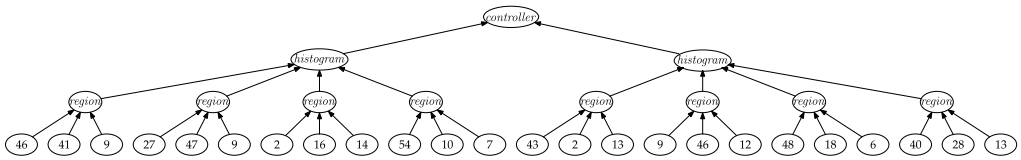
sizes, only restricted by the maximum-depth of the tree parameter. The results show that on average the method starts with a large program that gets reduced in later stages (subsequent generations). The main reason of this behaviour is the overlapping ratio (OVR) component of the fitness function, which forces the system to detect distinctive regions with minimal overlapping.

3.5 Further Analysis

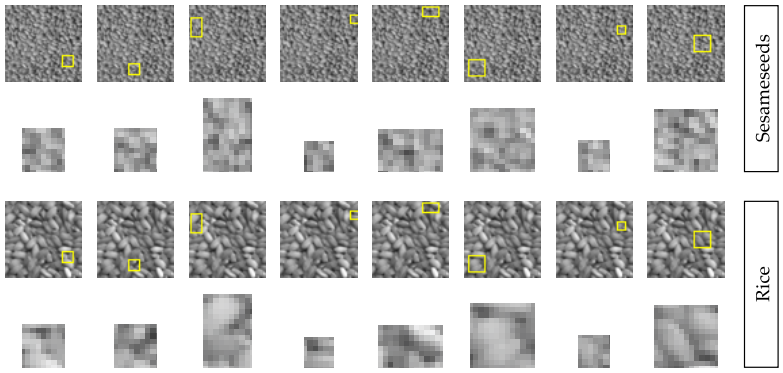
Two evolved programs from each of the One-shot GP and Compound-GP methods (four in total), are investigated in this section. The first example is taken from those programs that were evolved on one of the texture-based datasets, whereas the second example is a program that was evolved to discriminate between face and non-face instances.

3.5.1 One-shot GP Examples

The first example of an evolved program by One-shot GP on Textures-2 is presented in Figure 3.16. The tree representation of this program is shown in Figure 3.16(a) that is made up of one *controller*, two *histogram*, and eight *region* nodes. The position of each of those eight regions is highlighted in Figure 3.16(b) on one example of each class and the enlarged cut-outs are presented below each image. This program was evolved using the minimum allowed number of instances (two per class) by One-shot GP, and has achieved 100% accuracy on the unseen data of this dataset. A closer look on the enlarged cut-outs reveals that the regions of the



(a)



(b)

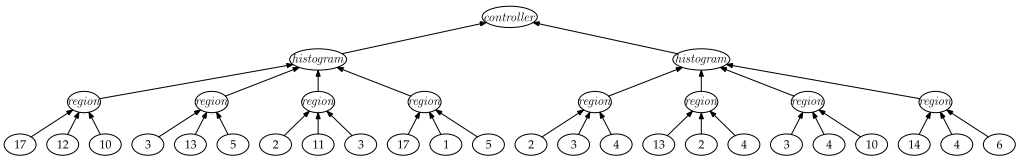
Figure 3.16: An evolved program by the One-shot GP method on the Textures-2 dataset (a) the tree representation, and (b) the detected regions.

rice instance has less texture than that of the corresponding *sesameseeds* instance.

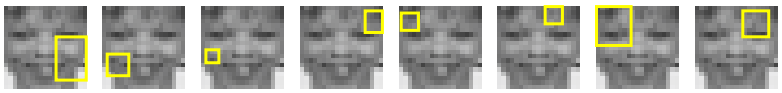
The example presented in Figure 3.17 shows an evolved program by the One-shot GP method on the Faces dataset. This program has achieved 78.3% accuracy using only two instances of each class in the training set. The program detects the regions around both eyes, eyebrows, and cheeks. Those regions (especially around the eyes and cheeks) are similar to those designed by a domain-expert, which shows the ability of the system to automatically detect such important regions.

3.5.2 Compound-GP Examples

The tree representation of an evolved program by Compound-GP on the Textures-2 dataset is presented in Figure 3.18(a). This program was evolved using only one instance of each class, and has scored 100% accuracy on the unseen data of this dataset. The detected regions by this program are depicted in Figure 3.18(b). The evolved program shows that the regions of different sizes and shapes have been detected, and the cut-outs of the two instances show clear differences in the texture



(a)



(b)

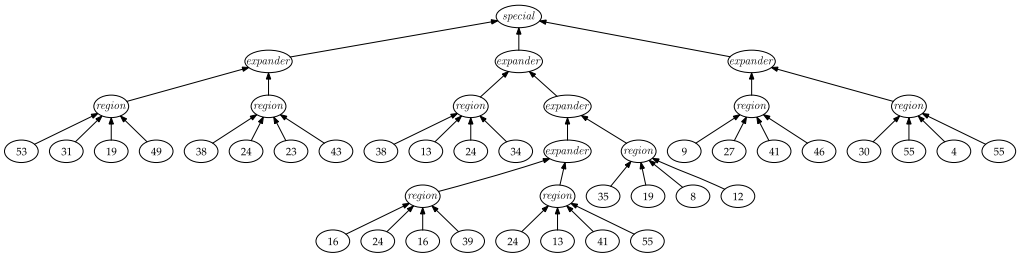
Figure 3.17: An evolved program by the One-shot GP method on the Faces dataset (a) the tree representation, and (b) the detected regions.

especially the small ones.

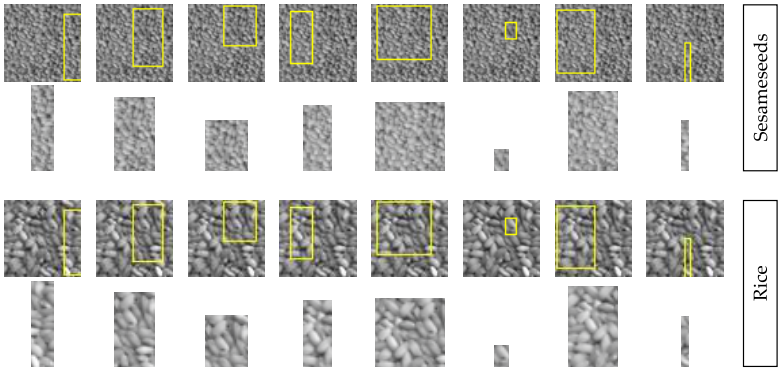
Figure 3.19 shows an evolved program by the Compound-GP method on the Faces dataset, which has achieved 78.0% accuracy using only one instance per class. The tree representation shows that nine regions have been detected by this program as depicted in Figure 3.19(a). These regions are highlighted in Figure 3.19(b), which shows that some interesting regions such as the mouth, three regions around the left eye and forehead, two regions detecting the left cheek, and three detecting the right cheek. This example shows that the detected regions have common features with those that were designed by a domain-expert.

3.5.3 Comments on the Number of Examples

An important question that is applicable to both methods is *why using a few instances can be sufficient to evolve a model with reasonably good performance*. The answer to this question can be the similarity between instances belonging to the same class. That is, instances belonging to one class must have distinctive features to those of other classes; otherwise, they must not be grouped together. For example, the instances of each texture-based class have a special repetitive pattern or structure, where detecting this structure represents the main task of the *Texture synthesis* field [235, 101]. Similarly, all instances of the Faces dataset have similar features in relatively fixed positions (eyes, nose, cheeks and so on). Therefore, allowing the system to detect distinctive regions from one or a few instances can be

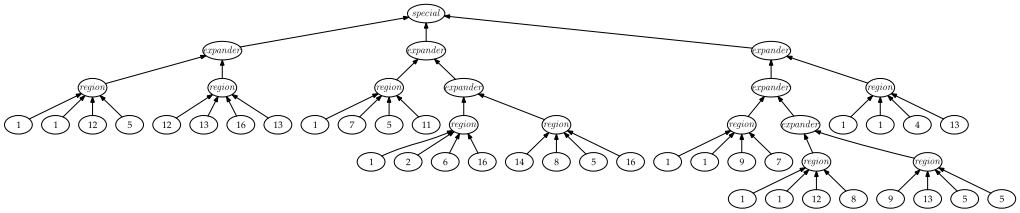


(a)

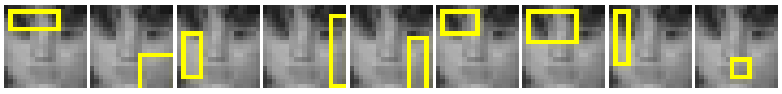


(b)

Figure 3.18: An evolved program by the Compound-GP method on the Textures-2 dataset (a) the tree representation, and (b) the detected regions.



(a)



(b)

Figure 3.19: An evolved program by the Compound-GP method on the Faces dataset (a) the tree representation, and (b) the detected regions.

sufficient to generalise to unseen data. Therefore, the assumption is that the same pattern or structure is shared by all instances of one class. Moreover, the search (evolutionary) process is not terminated in One-shot GP and Compound-GP once the instances of the training set are correctly classified as in the other methods. Instead, the system tries to find better regions that allow the evolved programs to generalise well to the unseen data.

3.6 Chapter Summary

The overall aim of this chapter was to investigate the capability of GP to evolve a model for binary classification tasks in images using only one or a few instances per class. This goal was fulfilled by developing two GP methods namely One-shot GP and Compound GP. Neither of the two methods relies on the concept of *learning by knowledge transfer* approach where some instances from related domains can be used to assist the process of evolving a model. Using ten datasets of different flavours, and compared against two other GP and six non-GP methods, the results revealed that both of these new methods (One-shot GP and Compound-GP) were able to evolve good programs that have the potential to perform image classification. Furthermore, the evolved programs have either outperformed all other competitor methods, or achieved comparable performance to the best of these other methods in most of the studied cases.

Some of the datasets that were used are aimed at testing the ability of the evolved programs by One-shot GP and Compound-GP to handle illumination, rotation, and scale variations. The results show that the evolved programs by those two methods have maintained their performances or slightly dropped when one or more of those variations occurred. Hence, both of these new methods are capable to evolve (to some extent) illumination-, rotation-, and scale-invariant programs.

The impact of the detected and extracted features by One-shot GP and Compound-GP on the performance of a variety of classifiers such as AdaBoost, NB, NBTree, SVM, K*, and NNeg has also been investigated in this chapter. The results were compared against the use of hand-crafted features (in conventional GP) and those were extracted by the Two-tier GP [6] method. The results show that the features of One-shot GP and Compound-GP have positive influence on

the performance of most of the experimented classifiers in most cases. However, in other cases those two methods (One-shot GP and Compound-GP) features have slightly degraded the performance of some of those classifiers.

In order to address the interpretability aspect, two examples of the evolved programs by each of the One-shot GP and Compound-GP methods have been analysed and discussed in detail. The discussions revealed that the evolved programs are easy to interpret, unlike other methods that build a black-box like model.

Despite the good ability of the One-shot GP and Compound-GP methods to evolve good programs using a small number of examples, a closer inspection on the complexity of the evolved programs by these two methods reveals their drawbacks. The methods take a considerably longer time (especially Compound-GP) to evolve a good program than other GP-based methods specifically Conventional-GP and Two-tier GP. Moreover, the size of the program evolved by Compound-GP is larger than those evolved by Conventional-GP, Two-tier GP, and One-shot GP. The size of the program evolved by One-shot GP, is fixed and introduces an extra parameter that needs to be set. Although the features extracted by either of these two methods, i.e., One-shot GP and Compound-GP, are not biased to a specific classifier (as the results suggest), it may vary when the wrapped classifiers are replaced by other classifiers.

Tackling the problem of multi-class classification problem is addressed in the next chapter. Instead of evolving a classifier, the aim is to evolve a model that can generate relatively similar features for instances of belonging to the same class and different than those features of the instances in other classes.

4

GP for Illumination-invariant Image Descriptors

4.1 Introduction

The performance of a model to perform image classification is highly dependent on the extracted features. Identifying potential keypoints and specifying a set of features to be extracted from those keypoints are typically performed by a domain-expert who, in many cases, is expensive to employ and hard to find. Therefore, image descriptors have emerged to automate the task of feature detection and/or feature extraction. However, developing those descriptors is not a trivial task. It requires domain-expert intervention to design the needed components and methodologies to accomplish the intended tasks.

The limitations of the existing image descriptors can be summarised in three main points. Firstly, they are developed to detect a specific type or a set of keypoints that were previously designed by a domain-expert. Although the domain-expert can identify crucial keypoints, e.g., corners and lines, some of the less

noticeable or irregular keypoints may have a significant effect on the final results. Furthermore, the difficulty of identifying informative keypoints increases when the number of classes increases. Secondly, these descriptors are abstract and assumed to be domain-independent. It is very difficult to find a set of keypoints that are useful over several different problems. For example, detecting lines is perfectly good when performing texture classification, e.g., to differentiate instances that have more vertical lines than those with more horizontal lines, but it is less likely to be effective for face detection. Thirdly, the vast majority of these descriptors are not flexible and substantial changes may required to alter a parameter or make it robust to image variants such as LBP (see Section 2.1.2 on page 29).

4.1.1 Chapter Goals

Motivated by the success employing image descriptors to tackle a variety of problems in computer vision, we wish to use GP to automatically evolve an illumination-invariant image descriptor by combining a set of arithmetic operators and raw pixel values. This chapter aims at addressing the following objectives:

- Designing an appropriate GP representation that can be used as an image descriptor;
- Developing a fitness function that is capable of providing adequate feedback regarding the separability level of the detected keypoints and extracted features when there are only two instances per class in the training set;
- Assessing the performance of the evolved descriptors over a series of experiments in order to investigate whether they can outperform hand-crafted features and domain-expert designed descriptors;
- Examining the robustness of an evolved descriptor to the change of image illumination; and
- Investigating an evolved image descriptor to gain insight into how it can solve the problem, and whether patterns about the data can be drawn from those automatically evolved descriptors.

4.1.2 Chapter Organisation

The remainder of the chapter is organised as follows. Section 4.2 presents the proposed GP illumination-invariant image descriptor method and explains the main steps to construct the feature vector for an image. Section 4.3.4 discusses the datasets used to assess the performance of the proposed method, the parameter settings, and the methods for comparison. The results of the experiments are presented and discussed in Section 4.4. Section 4.5 provides a deep analysis of an evolved image descriptor. The chapter summary is presented in Section 4.6.

4.2 GP Image Descriptor

The proposed illumination-invariant GP image descriptor method is discussed in this section. The newly proposed method in this chapter is named GP-criptor throughout this thesis. The overall algorithm is discussed first, followed by descriptions of the program representation, feature vector, and fitness function.

4.2.1 The Overall Algorithm

The procedure adopted to evaluate the newly introduced method is made up of the typical machine learning *training* and *testing* stages. Figure Figure 4.1 illustrates the overall algorithm. The content of the dataset is equally divided into training and test sets. The system then randomly picks two instances from each class and feeds them to the GP process. The GP evolutionary process runs and the best evolved program at the end, i.e., final generation or a program with ideal fitness, is selected. The training instances, i.e., the two randomly selected instances per class, are fed to the best evolved program in order to generate the *knowledge base* vectors ($\mathbf{K}$). Hence, $\mathbf{K}$ comprises the feature vectors and the corresponding class labels of the instances that were used during the evolutionary process. The system then uses the best evolved program to generate the feature vector for each instance in the test set. The resulting knowledge base is used along with a classifier (e.g. 1-NN) to predict the class label for the instances of the test set. The performance is assessed by the ability of the model to correctly classify the unseen data. More

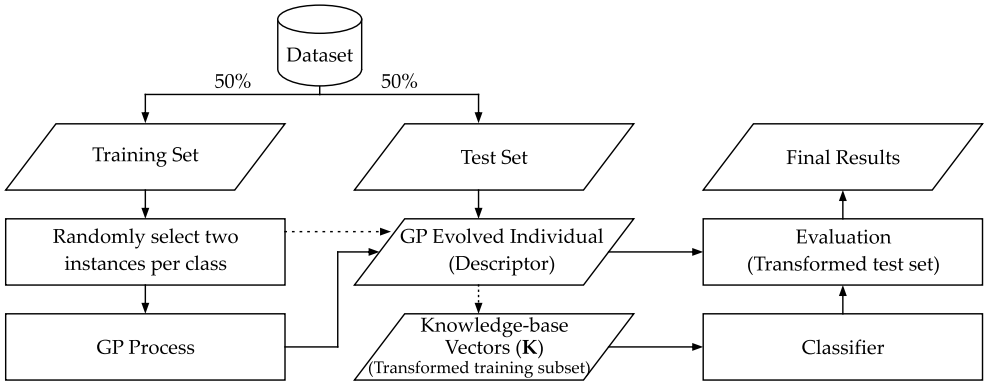


Figure 4.1: An overview of the overall algorithm.

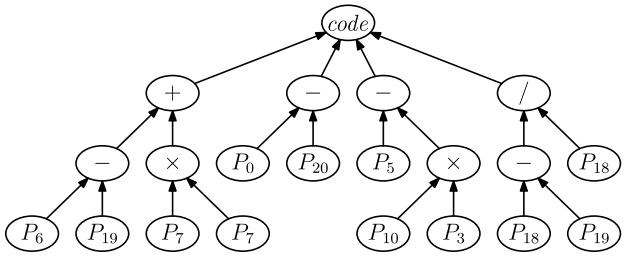


Figure 4.2: An example of a program evolved by GP-criptor.

details regarding each of these steps are presented in the following subsections.

4.2.2 Program Representation

The tree-based GP representation [153] is adopted to represent a program evolved by GP-criptor as depicted in Figure 4.2. In order to define constraints on the nodes and to satisfy the "closure" property [153, 243] (to preserve the constraints between the nodes of an evolved program and ensure those constraints are not violated when crossover and mutation are performed), strongly-typed GP [207] is used.

4.2.2.1 Terminal Set

GP-criptor is designed to operate directly on image raw pixel values. Therefore, the pixel values are the content of the terminal set. The system uses a sliding window of a specific size, and at each position, the pixel values that fall within the

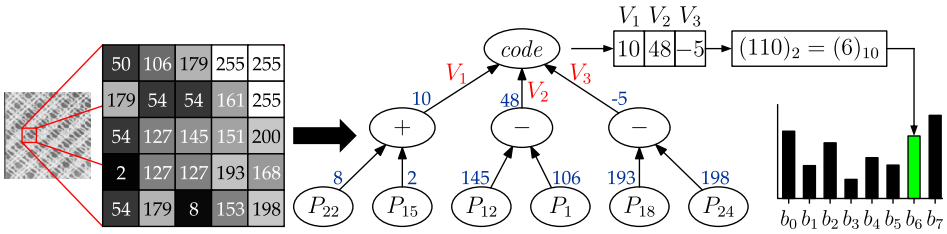
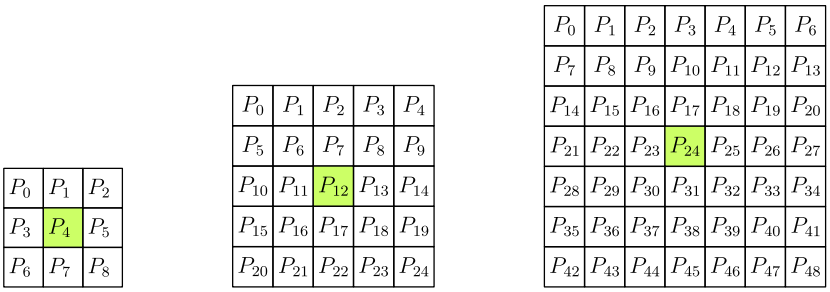


Figure 4.3: An example demonstrating the process of generating a feature vector for an image by the GP-criptor method.



(a) (b) (c)

Figure 4.4: Examples shows the pixel indices of sliding windows with size (a) 3×3, (b) 5×5, and (c) 7×7 pixels.

window are used as inputs as shown in Figure 4.3. The index of the i^{th} pixel of the sliding window is represented as P_i. Figure 4.4(a)–(c) shows three different window sizes and the pixel indices of each of them. In our experiment, the size of the sliding window is set to 5×5 pixels (see Section 4.4.1 on page 130). Thus, the terminal set for the GP is the indices of 25 pixels $\{P_0, P_1, \ldots, P_{24}\}$. An evolved program is expected to be illumination-invariant as the terminal set comprises only the indices of the pixels and not the actual values. This allows the system to capture the relations between those indices, which will be the same if the intensity value of all pixels has been increased or decreased uniformly.

4.2.2.2 Function Set

The function set, on the other hand, consists of the $+$, $-$, $\times$, protected $/$, and *code* operators. Each of the first four operators has its regular mathematical meaning, takes two parameters, and returns the resulting value after applying the

corresponding operator on the inputs. The / operator is protected such that it returns zero if the denominator is zero. These four operators have identical input and output types. Thus, they can be used to form long chains of operators. The *code* node, on the other hand, is a special node that represents the root of the program tree. This node uses the input parameters, i.e., the values of its children, to generate a binary code at each position of the sliding window (more details are provided in the following subsection). The number of children of the *code* node specifies the length of the generated code, which in turn specifies the range of the values that can be represented (i.e. 2^n where n is the number of children). For example, if the number of children is 3, then the program can produce $2^3 = 8$ different values as shown in Figure 4.3 (where the resulting feature vector comprises the bins values $\{b_0, b_1, \ldots, b_7\}$). As the input and output types of the *code* node are different, and it is also different than the other four elements of the function set, this node only appears at the root of an evolved program.

4.2.3 Generating the Feature Vector

The aim of the program evolved by the GP-criptor method is to synthesise a set of equations that are used to convert an image to a feature vector (histogram) via generating binary codes using the pixel values. The process is demonstrated in Figure 4.3. The system scans the pixels of the instance being evaluated from left to right and from top to bottom using a sliding window. At each pixel, the system performs four steps. Firstly, the values of the current window, i.e., centred at the current pixel, are fed into the leaf nodes of the program tree. Secondly, the GP program is evaluated starting from the lower part (leaves) to the top part (root). In the third step, the negative values of the *code* node children are set to 0, whilst positive and zero values are set to 1. Fourthly, the generated code in the previous step is converted to decimal and the corresponding cell (bin) of the feature vector is incremented by 1.

4.2.4 Generating the Knowledge base

The *knowledge base* is a special set of vectors that is denoted as $\mathbf{K}$. This set is generated using the evolved program and the content of the training set (i.e. the

two randomly selected instances per class). Thus, the size of $\mathbf{K}$ is $2 \times C$ where C is the total number of classes. The system follows the steps outlined in the previous subsection to generate the feature vector for each instance in the training set and store it in $\mathbf{K}$. The set $\mathbf{K}$ plays two essential roles. Firstly, it is used to measure the fitness value. Secondly, it serves as the knowledge-base to help in classifying the unseen data.

4.2.5 Fitness Function

Due to having a small number of instances in the training set (only two instances per class) and a large feature space, the fitness function has been designed to extract as much information as possible from those instances. Unlike the typical fitness function where the accuracy is considered alone, the fitness function of GP-criptor takes in to account the distances between the training instances. The fitness function is defined as:

$$\textit{fitness} = 1 - \left(\frac{1}{1 + e^{-5(D_\mathrm{b} - D_\mathrm{w})}} \right) \tag{4.1}$$

where D_b is the average distance of *between-class* instances calculated using Equation (4.2), and D_w is the average distance of *within-class* instances calculated using Equation (4.3).

$$D_\mathrm{b} = \frac{1}{z\,(z-m)} \sum_{\substack{\mathbf{u}^\alpha, \mathbf{v}^\beta \in \mathbf{S}_\mathrm{tr} \\ \forall \vec{u} \in \mathbf{u}^\alpha \\ \forall \vec{v} \in \mathbf{v}^\beta}} \chi^2\left(\vec{u}, \vec{v}\right), \qquad \alpha, \beta \in \{1, 2, \ldots, C\}, \alpha \neq \beta \tag{4.2}$$

$$D_\mathrm{w} = \frac{1}{z\,(m-1)} \sum_{\substack{\mathbf{u}^\alpha, \mathbf{v}^\alpha \in \mathbf{S}_\mathrm{tr} \\ \forall \vec{u} \in \mathbf{u}^\alpha \\ \forall \vec{v} \in \mathbf{v}^\alpha}} \chi^2\left(\vec{u}, \vec{v}\right), \qquad \alpha \in \{1, 2, \ldots, C\} \tag{4.3}$$

Here, $\mathbf{S}_\mathrm{tr} = \{(\vec{x}_i, y_i)\}$ is the training set, where $\vec{x}_i \in \mathbb{R}_{\geq 0} = \{l \in \mathbb{R} \mid l \geq 0\}$ is the feature vector, y_i is the class label, and $i \in \{1, 2, \ldots, z\}$; C and m are, respectively, the total number of classes and the number of instances per class; z is the total number of instances in the training set (i.e. $C \times m$), and $\mathbf{x}^\alpha$ is the set of all instances of the α^{th} class in $\mathbf{S}_\mathrm{tr}$. The widely used $\chi^2(\cdot, \cdot)$ function measures the distance between two normalised vectors of the same length as:

$$\chi^2\left(\vec{u}, \vec{v}\right) = \frac{1}{2} \sum_i \frac{(u_i - v_i)^2}{u_i + v_i} \tag{4.4}$$

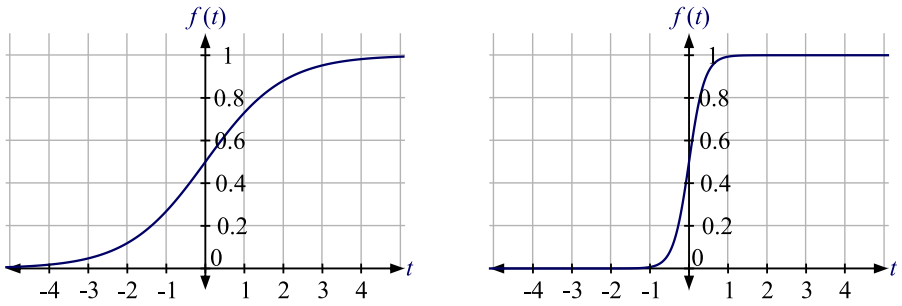


Figure 4.5: Logistic function (a) $f(t) = \frac{1}{1+e^{-t}}$, and (b) $f(t) = \frac{1}{1+e^{-5t}}$.

where u_i and v_i are the i^{th} element in the $\vec{u}$ and $\vec{v}$ vectors, respectively. When the denominator of Equation (4.4) for an index i, i.e., $u_i + v_i$, is zero, the function returns zero in order to prevent the *division by zero* issue [48].

This fitness measure (Equation (4.1)) returns 1 and 0, respectively, in the worst and best case scenarios. Moreover, the second part of Equation (4.1) is a modified version of the conventional *sigmoid* function as presented in Figure 4.5(a). The value 5 is included in the exponent to scale down the input range from $(-5, +5)$ to $(-1, +1)$ as depicted in Figure 4.5(b). The motivation behind making the effective input range narrow is mainly because the $\chi^2(\cdot, \cdot)$ function returns values in the interval $[0, 1]$. Therefore, using the conventional sigmoid function the results of $f(t)$ will be in the interval $[0.27, 0.73]$. Therefore, including 5 is to scale the output interval to be $[0, 1]$.

4.3 Experiment Design

The experiments are highlighted and discussed in this section. The discussion includes the datasets, parameter settings, and methods for comparison.

4.3.1 Datasets

The performance of the proposed method is assessed by using three datasets that are drawn from three widely used texture benchmarks in computer vision.

The instances of the first dataset (BrNoRo) are taken from the popular Brodatz Textures dataset [42] (see Section 2.4 on page 62). The BrNoRo comprises of 20 classes each of which consists of 84 rotation-free instances.

The second dataset (KyNoRo) in this chapter is formed from the Kylberg Texture [157] (see Section 2.4 on page 62). Only the instances of the without rotation dataset are used. This dataset comprises 28 classes each of which consists of 160 instances.

The third dataset (OutexTC00) is formed using the instances of the Outex Texture Classification (see Section 2.4 on page 62). This dataset comprises 24 classes each of which consists of 20 instances.

The total number of instances of each of the selected classes has been equally divided between the training and test sets in the experiments of this chapter. Hence, each of the training and test sets of BrNoRo has 840 instances (= 84 (instances) × 20 (classes)). Meanwhile, KyNoRo has 2240 instances (= 80 (instances) × 28 (classes)) in each of the training and test sets. Finally, the OutexTC00 dataset has 240 instances (= 10 (instances) × 24 (classes)) in each of the training and test sets.

4.3.2 Benchmark Methods for Comparison

In order to investigate the effectiveness of the proposed method, its performance is compared to the performance of the common state-of-the-art descriptors DIF, GLCM, and $\text{LBP}^{u2}_{p,r}$. Image descriptors are used as pre-processing methods to convert an instance from raw pixel values into a feature vector after detecting some keypoints. Hence, researchers broadly rely on classification or recognition to assess the goodness of a descriptor [217, 185, 4, 26]. Here, the classification performance is also used to assess whether the proposed method is capable of evolving good descriptors. GP-criptor uses a simple instance-based classification algorithm namely *the-nearest-neighbour*, i.e., k-Nearest Neighbour (k-NN) with k set to 1. Classifiers of different types have been intentionally selected in the conducted experiments to ensure that the new method is not biased towards a specific classifier or type of classifier. Those classifiers are (see Section 3.3.3.1 on page 89) k-NN, Support Vector Machines (SVM), Naïve Bayes (NB), Naïve Bayes/Decision Tree (NBTree),

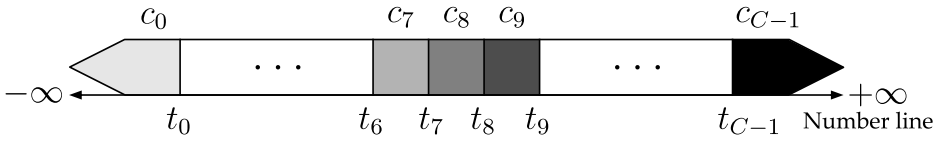


Figure 4.6: The real number line division in SRS, where c_i is the i^{th} class label, t_i is the i^{th} threshold value, and C is the total number of classes.

Multilayer Perceptron (MLP), Adaptive Boosting (AdaBoost), Decision Trees (J48), and Random Forest (RF). The implementations of these classifiers are taken from the commonly used Waikato Environment for Knowledge Analysis (WEKA) [116] software. For more details regarding these methods, see [323].

In [283], GP is utilised to perform multi-class texture classification by employing the *Static Range Selection* (SRS) [295, 336] and *Dynamic Range Selection* (DRS) [184, 283] methods to evolve a classifier that operates directly on the raw pixel values. These two methods are also considered in the conducted experiments.

SRS is designed to extend GP for multi-class classification tasks. The main idea is to keep the typical tree representation of GP that produces a single value from the root node, and change the mapping process to predict the class label. Thus, the real number line is divided into C intervals instead of only two (i.e. negative and positive), where C is the total number of classes as shown in Figure 4.6. Each of those intervals is static and assigned to a specific class. The main drawback of this method is that it requires to manually specify $C - 1$ threshold values.

DRS also aims at extending GP for multi-class classification by changing the program output mapping mechanism. Similarly, in DRS the real number line is split into a number of intervals. However, the intervals are not fixed as in SRS and they are determined in a dynamic way. The DRS method divides the training set into *evaluation* and *segmentation* sets. The former is used to measure the goodness of the evolved program (e.g. accuracy), whereas the latter is used to determine the intervals.

The program representation, terminal and function sets, and fitness function of SRS and DRS are similar. More details are in [283].

Table 4.1: The parameter settings of the GP methods

Parameter	Value	Parameter	Value
Crossover Rate	0.80	Generations	50
Mutation Rate	0.20	Population Size	300
Elitism	Keep the best	Initial Population	Ramped half-and-half
Tree minimum-depth	2	Selection Type	Tournament
Tree maximum-depth	10	Tournament Size	5

4.3.3 Parameter Settings

4.3.3.1 Evolutionary parameters of the proposed method

The parameter settings of the three GP methods, i.e., GP-criptor, SRS, and DRS, are summarised in Table 4.1. The ramped half-and-half method is used to generate the initial GP population. Dealing with images is an expensive task in general, hence, the population size is set to 300 individuals. The tournament selection strategy with a tournament of size 5 is used to maintain the population diversity. The probabilities of applying crossover, and mutation operators are respectively 0.80, and 0.20; whereas *keep the best* mechanism is used to prevent the evolutionary process from degrading. The tree depth of an evolved program is restricted to be between 2 and 10 levels in order to avoid code bloating [318]. Finally, the evolving process stops when an *ideal* individual, i.e., fitness value is 0 (or very close to ideal, e.g., $< 10^{-6}$), is found, or the maximum number of generations, i.e., 50, is reached.

4.3.3.2 Parameters of the classifiers

In this chapter, a number of well-known classification methods are used to assess the goodness of the generated features by an image descriptor evolved by GP-criptor$^{\text{ri}}$ or one of the baseline methods. Those classifiers (see Section 3.3.3 on page 89), as discussed in [116], have several parameters and tuning all of them is beyond the scope of this study. However, some of the important ones are considered in our experiments. Due to having only two instances of each class in the training set, the number of the neighbours for all instance-based methods, e.g., NNge, and k-NN, is set to 1 (the closest neighbour).

For SVM, a study by Keerthi and Lin [144] investigated the impact of linear and non-linear kernels on the performance. In their study, it has been observed that using non-linear kernels in SVM are more likely to give better performance than using linear kernels. Following their suggestions, the radial basis function kernel is employed in our experiments.

Similarly, the network structure of MLP is specified based on the guidelines of Trenn [298]. Typically, MLP has a single input layer that comprises of one node for each feature in the feature vector; whereas the number of classes specifies the number of nodes in the output layer. Only one hidden layer is used and, based on [298], the following formula is considered to calculate the number of nodes in this layer:

$$N_{\text{hidden}} = \left\lceil \frac{N_{\text{in}} + N_{\text{out}}}{2} \right\rceil \tag{4.5}$$

where the values of N_{in} and N_{out} correspond to the number of nodes in the input and output layers.

AdaBoost is a meta-algorithm that was introduced by Freund *et al.* in 1996 [97] designed to be used in conjunction with other machine learning algorithms in order to enhance their performance. The overall idea of AdaBoost is to adaptively build a model by considering those previously misclassified instances by the current models to improve the subsequent ones. In our experiments, the multi-class alternating decision trees classifier [131] is used as it gave better performance than DecisionStump.

4.3.4 Experiments

In this chapter, three sets of experiments have been conducted each of which aims as investigating a specific aspect. GP-criptor comprises two parameters: *window size*, and *code length*. The impact of using different settings for these two parameters on the performance is investigated in the first set of experiments. Three window sizes are tested in this experiment: 3×3, 5×5, and 7×7 pixels. Similarly, three code lengths are examined: 7-bits, 8-bits, and 9-bits. The second set of experiments investigates and compares the performance of GP-criptor to the GP-based baseline methods, i.e., SRS and DRS. Investigating and comparing the performance of the GP-criptor method to eight state-of-the-art hand-crafted image descriptors using

nine different classification algorithms (non-GP methods) represents the aim of the third set of experiments.

In each experiment, each of the stochastic methods has been run independently 30 times using a different **seed** value each time, and the average performance is reported; whereas deterministic methods have been executed only one time. As mentioned earlier, only two instances per class of the training pool (the 50% of the whole dataset instances) are **randomly** selected to form the training set. Thus, the same process of the 30 independent runs for non-deterministic and the single run of the deterministic methods has been further repeated 10 times using different instances in the training set. Moreover, the *mean* ($\bar{x}$) along with *standard deviation* (s) statistics of these methods on the test set have been calculated and reported.

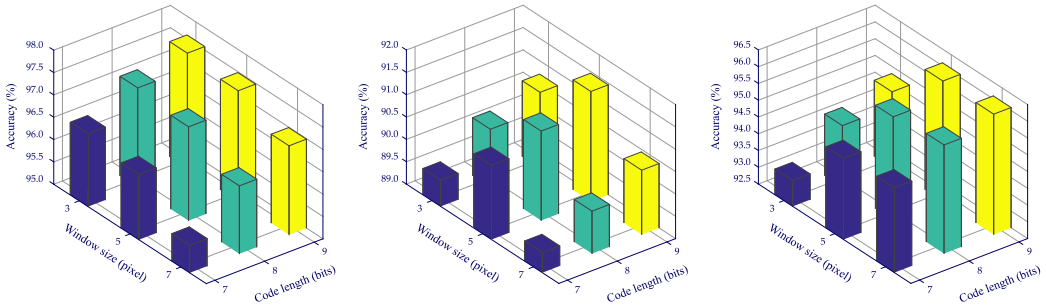
4.3.5 Implementation

Similar to One-shot GP and Compound-GP (Chapter 3 on page 75), the STGP platform provided by ECJ version 23 [190] is used to implement the three GP methods, i.e., SRS, DRS, and GP-criptor. Meanwhile, the WEKA package version 3.8 [116] is used to run all the non-GP methods (see Section 3.3.7 on page 94).

4.4 Results and Discussions

This section presents and discusses the results obtained from the conducted experiments on the three datasets. GP-criptor comprises two parameters: *window size*, and *code length*. The impact of using different combinations of these two parameter settings on the performance is discussed first. The performance of the GP-based methods (GP-criptor, SRS, and DRS) is then presented and discussed. Comparing the performance of GP-criptor image descriptors to eight hand-crafted descriptors using nine classification algorithms is then discussed in this section.

A significant difference between the performance of the newly introduced method and that of each of the baseline methods has been determined using the *Wilcoxon signed-rank* test [66, 320] with a significant level of 5%.



(a) BrNoRo (b) KyNoRo (c) OutexTC00

Figure 4.7: The impact of the window size and code length on the performance on the (a) BrNoRo, (b) KyNoRo, and (c) OutexTC00 datasets.

4.4.1 Window Size and Code Length

The results of the first set of experiments are presented in Figure 4.7 and Table 4.2. Generally, using more children under the *code* node has a positive influence on the performance of GP-criptor. Meanwhile, the impact of the window size on the performance is varying between the datasets. However, the performance gap between using different sizes is not large. On the BrNoRo dataset, a 3×3 pixels window shows better performance than the other two sizes; where the best average performance (97.58%) was achieved when the code length is 9-bits. The results on the KyNoRo dataset show that the a 5×5 pixels window and 9-bits code combination has achieved the best average performance (91.46%) on this dataset. On OutexTC00, GP-criptor shows very similar performances between using 5×5 pixels and 7×7 pixels windows. However, the latter, i.e., 7×7 pixels, has a slightly higher performance on average as presented in Table 4.2; where the best achieved performance on this dataset was 96.90% on average.

Based on the results obtained in this set of experiments, the code length is set to 9-bits and the window size is set to 5×5 pixels in the subsequent experiments.

4.4.2 GP-based Methods

The results of the three GP-based methods (GP-criptor, SRS, and DRS) on the three texture datasets are presented in Table 4.3. A "$*$" sign is placed beside the

Table 4.2: Accuracy (%) for using different combinations of code lengths and window sizes on the three datasets ($\bar{x} \pm s$).

		Window size		
Dataset	Code length	3×3 pixels	5×5 pixels	7×7 pixels
BrNoRo	7-bits	96.63 ± 0.93	96.46 ± 0.72	95.62 ± 0.88
	8-bits	97.23 ± 0.85	97.10 ± 0.63	96.52 ± 0.66
	9-bits	97.58 ± 0.79	97.48 ± 0.62	96.99 ± 0.86
KyNoRo	7-bits	89.59 ± 1.02	90.66 ± 1.21	89.45 ± 1.86
	8-bits	90.30 ± 1.08	91.00 ± 1.28	89.95 ± 1.91
	9-bits	90.70 ± 1.09	91.46 ± 1.30	90.44 ± 1.73
OutexTC00	7-bits	93.27 ± 2.38	94.92 ± 1.95	95.03 ± 1.76
	8-bits	94.34 ± 2.28	95.59 ± 1.92	95.74 ± 1.73
	9-bits	94.78 ± 2.52	96.09 ± 1.98	96.09 ± 1.83

value of each method in Table 4.3 that has significantly worse performance than that of the GP-criptor method. The SRS and DRS methods have achieved the worst performance in the experiments on all the three datasets. GP-criptor has significantly outperformed both of these methods as presented in Table 4.3. At this stage, it is difficult to claim whether SRS and DRS performed poorly since there are a large number of classes, they require a large number of instances in order to achieve better performance, or the instances are very large (dimensions). The impact of the three factors is not examined in this thesis and will be investigated in the future. It is important to notice that both SRS and DRS are designed to evolve a classifier rather than an image descriptor, whilst GP-criptor evolves an image descriptor and uses k-NN to perform the classification task. Furthermore, those methods are not designed to perform classification using only a few training instances. This clarifies the poor performance of those methods in the conducted experiments. Meanwhile, the proposed method, i.e., GP-criptor, has been specifically designed to operate using a small number of training instances and uses a more powerful mechanism to evolve an image descriptor that is capable of handling this problem.

4.4.3 Baseline Image Descriptors

The aim of this set of experiments is to compare the performance of GP-criptor to eight hand-crafted image descriptors. The results are statistically tested using the

Table 4.3: The accuracies (%) on the test set for GP-based methods on the three datasets ($\bar{x} \pm s$).

	BrNoRo	KyNoRo	OutexTC00
SRS	5.70 ± 0.32 *	3.55 ± 0.15 *	4.73 ± 0.49 *
DRS	11.5 ± 0.34 *	5.39 ± 0.23 *	9.91 ± 1.50 *
GP-criptor	97.5 ± 0.62	91.5 ± 1.30	96.1 ± 1.98

Wilcoxon signed-rank test with a significant level of 5%. The statistical test has been applied twice, first, to check whether GP-criptor with a simple classifier (1-NN) can compete with the baseline descriptors with more powerful classifiers; and second, to test whether the GP-criptor method can compete with the baseline methods using the same classifier. The symbols "$*$" and "$-$" are used to, respectively, represent significantly better and significantly worse in the first test, whereas significantly better and significantly worse in the second test are indicated by "↑" and "↓", respectively. For each dataset, the corresponding method with best average performance for each classifier is made **bold**; whilst the best performance amongst all methods and classifiers is underlined. If two or more methods have the same best average performance, the one with the smallest standard deviation is underlined.

Table 4.4 show the results of applying nine classification algorithms on the BrNoRo dataset using the features of GP-criptor and the baseline image descriptors. Using k-NN, the GP-criptor method has achieved 97.6% accuracy on average and significantly outperformed all other hand-crafted image descriptors with k-NN and all other classification algorithms. Using the same classification algorithm, GP-criptor has also achieved comparable or significantly better performance in most of the cases as shown in Table 4.4. Apart from J48, all other classifiers have show the best performance using GP-criptor features compared to the other state-of-the-art image descriptors.

Table 4.5 presents the results of the experiment on the KyNoRo dataset. Similar to BrNoRo, a good performance has been achieved by GP-criptor over all the baseline methods on the KyNoRo dataset. Apart from $\text{CLBC}_{8,1}$ with k-NN, MLP, and NNge, GP-criptor with k-NN has significantly outperformed all other

Table 4.4: The average accuracy (%) of nine classifiers using nine image descriptors on the **BrNoRo** texture images dataset ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	MLP	NB	NBTree	NNge	RF	SVM
BrNoRo	DIF	36.8 ± 2.75∗	18.0 ± 3.60∗↑	28.5 ± 4.94∗↑	35.3 ± 2.50∗↑	21.0 ± 6.40∗↑	33.4 ± 4.74∗↑	34.3 ± 3.46∗↑	31.7 ± 3.46∗↑	33.6 ± 2.82∗↑
	GLCM	51.5 ± 7.31∗	17.0 ± 8.59∗↑	33.6 ± 6.08∗	53.6 ± 7.03∗↑	38.7 ± 6.45∗↑	44.6 ± 6.32∗↑	48.0 ± 5.07∗↑	43.9 ± 6.24∗↑	47.3 ± 6.33∗↑
	$LBP^{u2}_{8,1}$	84.0 ± 1.95∗	40.4 ± 4.68∗↑	33.6 ± 5.11∗	83.4 ± 1.59∗↑	62.8 ± 8.31∗↑	71.4 ± 5.84∗↑	82.1 ± 2.74∗↑	56.5 ± 1.62∗↑	75.3 ± 5.28∗↑
	$LBP^{riu2}_{8,1}$	68.5 ± 3.18∗	25.1 ± 4.34∗↑	**39.0 ± 4.95**∗↓	68.8 ± 2.87∗↑	34.7 ± 7.17∗↑	46.7 ± 2.57∗↑	63.3 ± 3.94∗↑	48.4 ± 2.13∗↑	62.8 ± 3.61∗↑
	$CLBP_{8,1}$	82.4 ± 4.88∗	37.3 ± 7.99∗↑	35.3 ± 4.43∗	85.1 ± 3.50∗↑	71.5 ± 5.04∗↑	77.4 ± 5.32∗↑	83.7 ± 4.65∗↑	61.3 ± 1.82∗↑	70.9 ± 5.75∗↑
	$LBC_{8,1}$	66.3 ± 2.80∗	23.5 ± 7.98∗↑	36.0 ± 6.77∗	67.5 ± 3.05∗↑	30.2 ± 7.25∗↑	45.0 ± 3.12∗↑	61.7 ± 2.77∗↑	45.3 ± 2.26∗↑	62.9 ± 2.79∗↑
	$CLBC_{8,1}$	63.6 ± 2.52∗	32.5 ± 3.90∗↑	32.7 ± 3.87∗	68.0 ± 2.25∗↑	61.9 ± 4.02∗↑	67.8 ± 5.87∗↑	65.9 ± 1.14∗↑	54.5 ± 1.37∗↑	50.7 ± 4.33∗↑
	$DRLBP_{8,1}$	83.2 ± 2.49∗	39.0 ± 4.67∗↑	35.7 ± 4.28∗↓	83.6 ± 2.69∗↑	58.0 ± 10.1∗↑	71.0 ± 4.90∗↑	82.9 ± 3.77∗↑	59.7 ± 1.47∗↑	73.3 ± 6.44∗↑
	GP-criptor	**97.6 ± 0.79**	**44.0 ± 1.45**	32.8 ± 0.94	**93.9 ± 1.09**	**87.2 ± 0.86**	**85.5 ± 1.26**	**95.8 ± 0.74**	**66.4 ± 0.82**	**86.7 ± 1.84**

Table 4.5: The average accuracy (%) of nine classifiers using nine image descriptors on the **KyNoRo** texture images dataset ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	MLP	NB	NBTree	NNge	RF	SVM
KyNoRo	DIF	22.2 ± 1.70∗	10.5 ± 2.73∗↑	28.4 ± 1.82∗↓	23.1 ± 1.95∗↑	19.4 ± 2.77∗↑	20.1 ± 2.60∗↑	27.1 ± 2.49∗↑	22.4 ± 0.87∗↑	21.1 ± 1.82∗↑
	GLCM	68.0 ± 3.05∗	18.2 ± 3.88∗↑	**44.5 ± 5.04**∗↓	64.5 ± 3.69∗↑	48.1 ± 5.68∗↑	55.1 ± 3.15∗↑	65.7 ± 3.79∗↑	56.0 ± 2.09∗	67.0 ± 2.80∗↑
	$LBP^{u2}_{8,1}$	75.5 ± 1.96∗	31.8 ± 3.06∗↑	36.7 ± 4.29∗↓	74.6 ± 2.06∗↑	53.0 ± 7.43∗↑	62.4 ± 4.62∗↑	73.6 ± 3.07∗↑	56.9 ± 1.28∗	66.5 ± 3.22∗↑
	$LBP^{riu2}_{8,1}$	67.4 ± 2.94∗	20.2 ± 6.54∗↑	38.2 ± 3.35∗↓	66.1 ± 2.57∗↑	40.2 ± 3.67∗↑	50.6 ± 3.67∗↑	64.4 ± 2.80∗↑	53.5 ± 2.73∗↑	66.1 ± 2.90∗↑
	$CLBP_{8,1}$	90.6 ± 1.16	**41.8 ± 2.43**∗	33.3 ± 3.54∗↓	**91.0 ± 1.55** ↓	67.5 ± 9.38∗	77.1 ± 4.05∗	**90.6 ± 1.58** ↓	**63.5 ± 0.99**∗	78.7 ± 4.50∗
	$LBC_{8,1}$	66.1 ± 2.76∗	18.4 ± 4.01∗↑	39.8 ± 4.61∗↓	66.6 ± 2.72∗↑	39.7 ± 4.97∗↑	52.0 ± 4.03∗↑	64.4 ± 3.37∗↑	52.4 ± 3.31∗↑	65.3 ± 2.66∗↑
	$CLBC_{8,1}$	76.7 ± 4.05∗	38.3 ± 3.95∗	32.8 ± 3.91∗↓	78.1 ± 3.76∗↑	64.2 ± 7.38∗↑	69.4 ± 4.45∗↑	77.8 ± 2.48∗↑	59.7 ± 1.57∗↓	51.3 ± 2.45∗↑
	$DRLBP_{8,1}$	86.3 ± 1.14∗	32.3 ± 6.84∗↑	37.1 ± 3.58∗↓	86.9 ± 1.29∗	64.3 ± 5.08∗↑	73.0 ± 3.82∗↑	85.9 ± 1.77∗↑	62.5 ± 1.78∗↓	76.1 ± 3.40∗↑
	GP-criptor	**90.7 ± 1.09**	39.5 ± 0.57	26.6 ± 1.02	87.3 ± 1.23	**69.8 ± 3.23**	**77.3 ± 1.68**	88.5 ± 1.90	58.1 ± 1.17	**80.1 ± 1.04**

descriptors and the gap is ranging between 3.8% and 80.2% accuracy on average. In the case of using other classifier than k-NN, GP-criptor has significantly better performance in more than 67% (= 43/64) of the cases. However, GP-criptor shows the worst average performance amongst the experimented image descriptors in the case of J48.

Finally, the results on the OutexTC00 dataset are presented in Table 4.6. This dataset is relatively smaller than the other two (BrNoRo and KyNoRo), where each class comprises of 20 instances in total that are divided equally between the training set and test set. The results show that GP-criptor with k-NN has significantly outperformed all other hand-crafted descriptors and achieved 94.8% performance on average. Moreover, the best average performance achieved on this dataset was by GP-criptor with MLP (94.9%). Using the same classification algorithm, GP-criptor has significantly outperformed the other eight image descriptors in more than 90% (58 out of 64) of the cases.Furthermore, the proposed method has the best performance amongst the experimented descriptors when the same classification algorithm is used, apart from J48 where $\text{DRLBP}_{8,1}$ has achieved significantly better performance than GP-criptor.

4.5 Further Analysis

This section aims at analysing different components of the GP-criptor method. The general analyses of the algorithm including the program size, fitness value, and evolutionary time are discussed in the first subsection. Then sample programs evolved on each of the two datasets are deeply analysed in the second subsection.

4.5.1 General Analyses

4.5.1.1 Convergence

The fitness value for the best program at each generation was reported in the previous two sets of experiments. Here, only those programs evolved using the combination of 9-bits code length and 5×5 pixels window size are discussed.

Figure 4.8(a) shows the average fitness value per generation of the best program on the BrNoRo dataset. Clearly, the system has made large jumps in the first

Table 4.6: The average accuracy (%) of nine classifiers using nine image descriptors on the **OutexTC00** texture images dataset ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	MLP	NB	NBTree	NNge	RF	SVM
OutexTC00	DIF	17.8 ± 1.27*	8.29 ± 2.54*↑	12.8 ± 4.51*↑	15.7 ± 1.81*↑	11.5 ± 2.46*↑	12.2 ± 4.14*↑	16.3 ± 1.20*↑	13.2 ± 1.34*↑	15.1 ± 2.60*↑
	GLCM	70.9 ± 2.10*	20.0 ± 6.89*↑	36.0 ± 7.40*	66.1 ± 4.62*↑	41.3 ± 11.8*↑	52.0 ± 6.65*↑	69.4 ± 2.82*↑	49.7 ± 5.13*↑	70.2 ± 2.61*↑
	$LBP^{u2}_{8,1}$	87.9 ± 1.23*	40.8 ± 12.0*	36.9 ± 3.84*	87.5 ± 1.21*↑	67.2 ± 10.3*↑	73.5 ± 3.09*↑	86.8 ± 1.36*↑	65.6 ± 1.48*↑	80.4 ± 3.04*↑
	$LBP^{riu2}_{8,1}$	69.5 ± 2.88*	26.8 ± 7.93*↑	39.2 ± 6.06*	66.5 ± 3.28*↑	44.3 ± 6.85*↑	51.6 ± 4.20*↑	68.8 ± 2.22*↑	53.0 ± 2.05*↑	69.7 ± 2.78*↑
	$CLBP_{8,1}$	81.0 ± 2.95*	37.4 ± 7.61*↑	23.4 ± 3.64*↑	82.1 ± 1.30*↑	61.3 ± 4.19*↑	68.2 ± 4.29*↑	80.7 ± 2.49*↑	56.8 ± 1.29*↑	70.2 ± 6.53*↑
	$LBC_{8,1}$	68.2 ± 3.81*	20.2 ± 4.60*↑	36.2 ± 3.75*	63.9 ± 3.69*↑	39.6 ± 5.78*↑	48.5 ± 3.05*↑	66.6 ± 3.87*↑	50.5 ± 1.67*↑	66.8 ± 2.98*↑
	$CLBC_{8,1}$	72.8 ± 2.87*	37.1 ± 5.85*↑	30.4 ± 3.55*↑	73.8 ± 2.92*↑	64.3 ± 4.86*↑	66.1 ± 4.91*↑	71.1 ± 3.53*↑	57.3 ± 1.46*↑	55.0 ± 4.70*↑
	$DRLBP_{8,1}$	85.0 ± 1.79*	34.1 ± 10.7*↑	**40.9 ± 4.32***↓	83.7 ± 1.47*↑	64.1 ± 5.49*↑	71.6 ± 3.83*↑	83.3 ± 1.68*↑	66.9 ± 1.48*↑	79.5 ± 4.79*↑
	GP-criptor	**94.8 ± 2.52**	**48.7 ± 2.02**	36.5 ± 3.45	**94.9 ± 1.81**	**81.3 ± 2.21**	**83.3 ± 2.07**	**93.4 ± 3.12**	**72.4 ± 1.35**	**88.6 ± 2.14**

20 generations where the fitness value has decreased from 0.287 to 0.169. The improvement was slower over the subsequent 30 generations and the fitness value has decreased to 0.141.

The average fitness value of the best programs per generation on the KyNoRo dataset is depicted in Figure 4.8(b). Similar to BrNoRo, on KyNoRo the system noticeably has reduced the fitness value over the first 20 generations from 0.361 to 0.261. The reduction of the fitness value was smaller that is approximately 0.028 (from 0.261 to 0.233) over the subsequent 30 generations.

Figure 4.8(c) shows the average fitness value per generation for the best programs on the OutexTC00 dataset. Following the same pattern of the other two datasets, the system has reduced the fitness value on average from 0.367 to 0.266 over the first 20 generations. Meanwhile, the system has reduced the fitness by approximately 0.026 (from 0.266 to 0.240) over the subsequent 30 generations.

This demonstrates the ability of the system to evolve good solutions over a small number of generations.

4.5.1.2 Program size

The system shows a similar pattern of the program size, which starts from small programs. The average program size (over 300 runs) of the best individual at each generation in terms of number of nodes on the BrNoRo dataset is presented in Figure 4.9(a). On average, the system starts with a program that comprises 120 nodes and starts to grow from the 6^{th} generation onward. Similarly, the results of the KyNoRo dataset are depicted in Figure 4.9(b) and show that the system starts with individuals of size 114 nodes on average in the early generations; where they get larger in the later generation and reaches around 179 nodes on average. Figure 4.9(c) shows the average program size per generation of those best evolved programs on the OutexTC00 dataset. In the early generations, the system uses a program that comprises 104 nodes on average and starts to grow from the second generation (114 nodes) until it reaches 177 nodes on average in the final generation.

Noticeably that the size of these best programs varies a lot as reflected by the standard deviation bars presented in Figure 4.9(a)–(c). That means the system has the potential to evolve diverse image descriptors that can be very small programs (easy to interpret and fast to execute), or very big and complex programs.

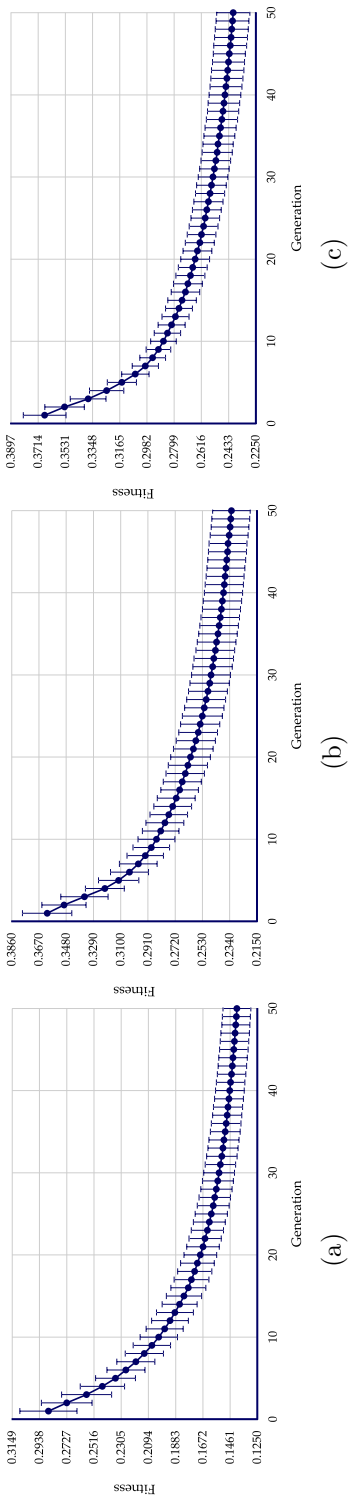


Figure 4.8: The average fitness value per generation over 300 (10 repeitation $\times$ 30 runs) best programs on the (a) BrNoRo, (b) KyNoRo, and (c) OutexTC00 datasets using a code of length 9-bits and a window of size 5×5 pixels (the whiskers represent the standard deviation).

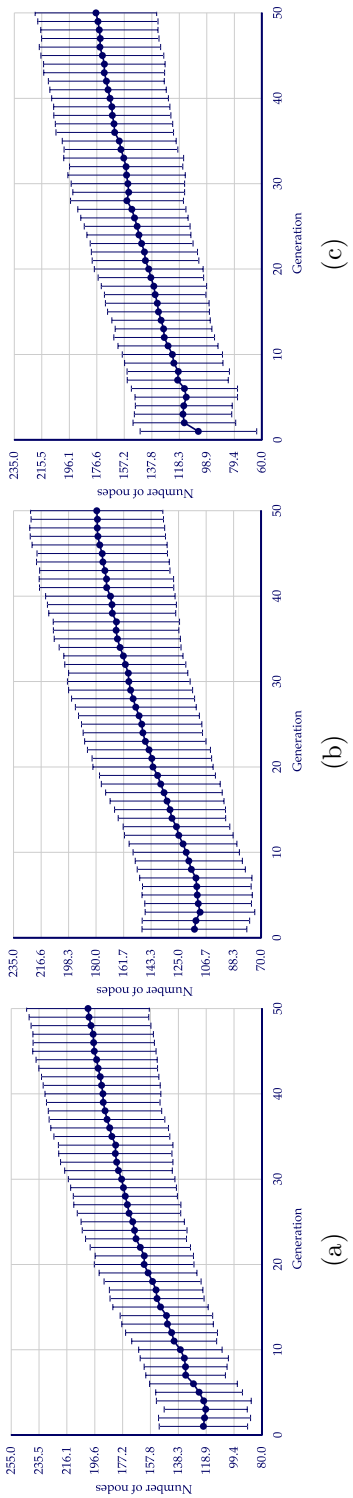


Figure 4.9: The average program size (number of nodes) per generation over 300 (10 repeitation $\times$ 30 runs) best programs on the (a) BrNoRo, (b) KyNoRo, and (c) OutexTC00 datasets using a code of length 9-bits and a window of size 5×5 pixels (the whiskers represent the standard deviation).

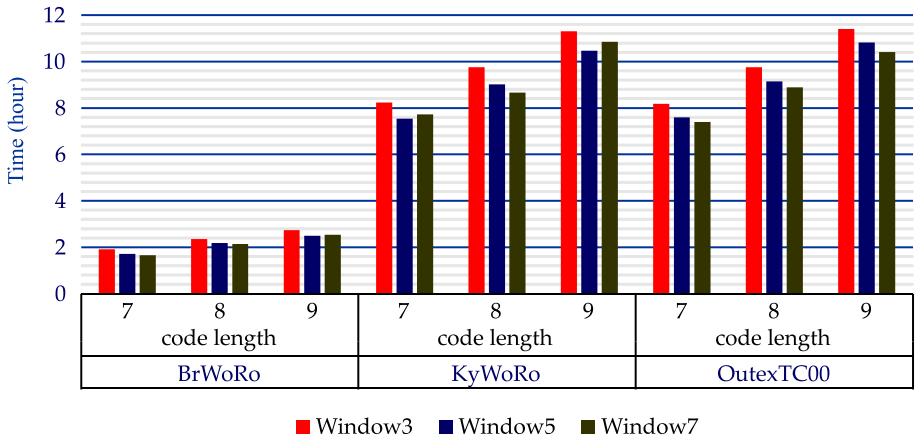


Figure 4.10: The average time in hours required to evolve a descriptor by GP-criptor using different code lengths and window sizes.

4.5.2 Evolutionary time

The average time required to evolve a program on each of the three experimented datasets is also measured in order to give a figure about this important characteristic.

Figure 4.10 shows the average time in hours required to evolve an image descriptor on the BrNoRo, KyNoRo, and OutexTC00 datasets, and presents the impact of using different window sizes and code lengths on the time. Clearly, GP-criptor needs shorter time to evolve a descriptor on the BrNoRo dataset compared to time needed for the other two datasets. This was expected, mainly because BrNoRo has fewer number of classes and the size of its instances is relatively smaller than the instances of KyNoRo and OutexTC00. Having more children under the *code* node (the length of the code) increases the time required to obtain a good descriptor; whereas the larger the window size, the shorter time that is needed to evolve a good descriptor. Using a larger window means the number of ignored pixel on the boundaries of each instance is more, which reduces the number of pixels that need to be considered.

4.5.3 Sample Programs

One of the best programs evolved by the GP-criptor method on each dataset is analysed and discussed in this subsection.

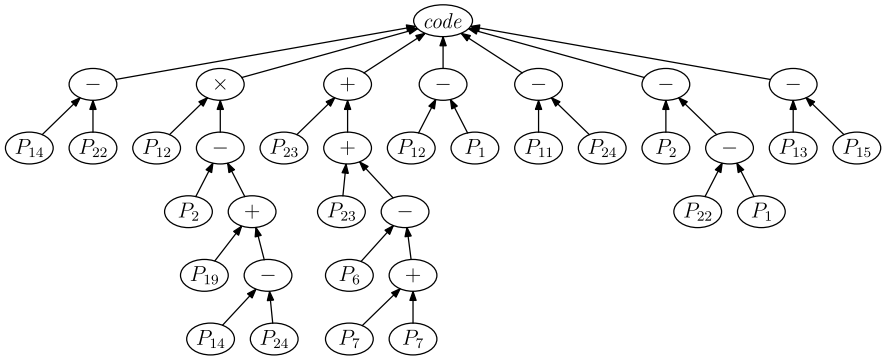


Figure 4.11: A program evolved by GP-criptor on the BrNoRo dataset.

4.5.3.1 BrNoRo

The tree representation of an evolved program on the BrNoRo dataset is shown in Figure 4.11. This program has achieved 97.7% accuracy on the unseen data. The tree shows that the system not only has successfully selected the pixels of the window that differentiate between instances of different classes, but also synthesises the arithmetic operators to be used to generate codes. The first, fourth, fifth, and seventh bits of the code are generated by subtracting the value of two pixels; whereas generating the other bits requires applying more operators. On average, the program performs two operations to calculate the value of each bit which is computationally efficient for online applications.

4.5.3.2 KyNoRo

Figure 4.12 presents the tree representation of a program evolved on the KyNoRo dataset that has scored 99.1% accuracy on the unseen data. The program applies only a single operator to generate the value of the first, third, fifth, sixth, and seventh bits of the code. Similarly, the program performs on average 2.3 operations to generate the value of each bit in the code.

4.5.3.3 KyNoRo

A program evolved on the OutexTC00 dataset is presented in Figure 4.13. Unlike the previous two examples, the program presented here is larger and comprises 78

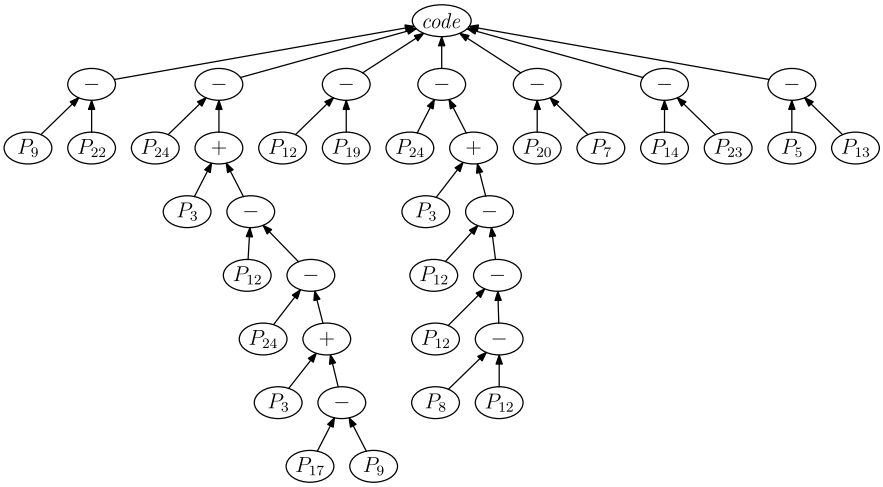


Figure 4.12: A program evolved by GP-criptor on the KyNoRo dataset.

nodes in total. The program is evolved using a code of length 7-bits and a 3×3 window size. This program achieved 96.67% accuracy on the unseen data and the confusion matrix (where the row indices represent the actual classes, while column indices the predicted classes) for this program is shown in Table 4.7.

A closer inspection of the previously presented examples shows that the "$-$" operator appears more frequently than the other operators. This was expected as this operator has the potential to switch the values from positive to negative, i.e., flips the bit value from 1 to 0, and vice versa.

4.6 Chapter Summary

In this chapter, GP has successfully been utilised to evolve illumination-invariant image descriptors for multi-class texture classification using only two instances per class. In the proposed method, only the raw pixel values have been used as inputs along with simple mathematical operators. The proposed method synthesises a set of mathematical formulae to generate a binary code from the pixels of a sliding window. The codes generated are used to populate the feature vector that is then fed into a simple instance-based classifier (1-NN) to predict the class label. The performance of the proposed method is assessed using three publicly available datasets, and

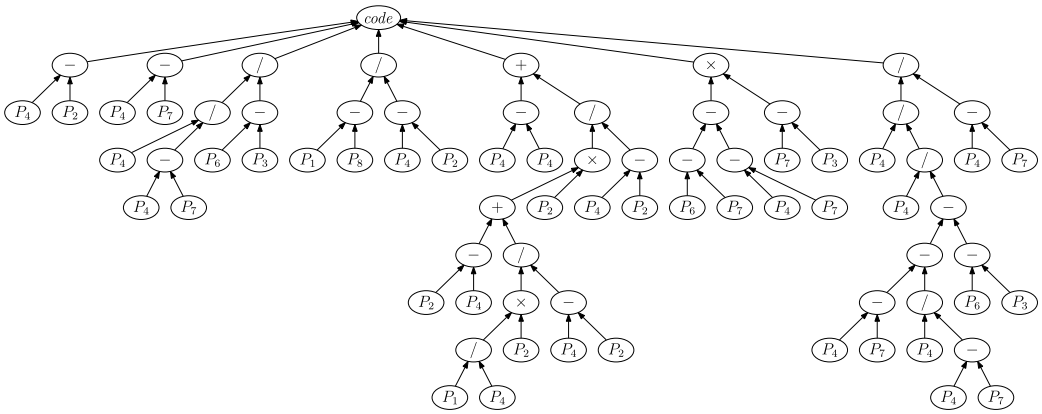


Figure 4.13: A program evolved by GP-criptor on the OutexTC00 dataset.

Table 4.7: The confusion matrix on OutexTC00 for the program presented in Figure 4.13

	C00	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	Total%
C00	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C01	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C02	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C03	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C04	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C05	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C06	0	0	0	0	0	0	3	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
C07	0	0	0	0	0	0	0	9	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	90
C08	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C09	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C10	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	100
C11	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	100
C12	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	100
C13	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	100
C14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	100
C15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	100
C16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0	0	0	50
C17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	100
C18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	100
C19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	100
C20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	100
C21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	100
C22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	100
C23	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	80

compared against the performances of two GP-based methods. Moreover, the performance of the new method is compared to that of using eight state-of-the-art hand-crafted descriptors with nine widely-used classification algorithms. The comparative results clearly show that a good performance has been achieved by the new method over the other methods. Furthermore, the results make it evident that the proposed method has the ability to evolve a descriptor that better fits the instances under consideration than the expert-designed methods. Analysing three programs (one one each dataset) evolved by the proposed method shows only a few arithmetic operators are required to generate the code at each position of the sliding window, which suggests the possibility of using those descriptors for online applications.

Extending GP-criptor to evolve rotation-invariant descriptors will be investigated in the next chapter.

5

GP for Rotation-invariant Image Descriptors

5.1 Introduction

Handling rotation in images and building a rotation-invariant descriptor is a challenging task. In the previous chapter, we have shown how GP is utilised to automatically evolve illumination-invariant image descriptors. However, further experiments (Section 5.5.1) shows the inability of GP-criptor to handle images with rotation. In this chapter, a novel GP based method (GP-criptor$^{\text{ri}}$) is developed that automatically synthesises a rotation- and illumination-invariant descriptor using only two training instances per class. To evolve a rotation-invariant descriptor, GP-criptor$^{\text{ri}}$ combines arithmetic operators and first-order statistics, e.g., mean and standard deviation, to form a model that takes an image and generates a feature vector.

5.1.1 Chapter Goals

The aim of this chapter is to develop a new GP approach to automatically constructing rotation-invariant image descriptors that can detect good keypoints and extract informative features simultaneously for image classification. Instead of using a large number of instances to train/learn a classifier as in most existing supervised approaches, the proposed approach will use only two instances per class in the training set. To achieve automatic construction of rotation-invariant image descriptors, the terminal sets of GP-criptor (Chapter 4) are changed in order to cope with rotation, whereas all other components are kept identical. Unlike in GP-criptor were only illumination-invariant descriptors used, the image descriptors automatically constructed by GP-criptor$^{\text{ri}}$ will be examined and compared with seven state-of-the-art domain-expert illumination- and rotation-invariant designed image descriptors on ten commonly used learning/classification methods on six texture and two non-texture image datasets of varying difficulty with different rotations. Specifically, we will investigate the following objectives:

- Develop a new terminal set to allow the proposed GP system to handle the rotation variation in images for image classification and automatically evolve/construct image descriptors from a small set of training instances;
- Investigate whether the GP-evolved image descriptors are robust to rotation and can achieve similar or even better performance than the seven state-of-the-art domain-expert designed image descriptors;
- Investigate whether the image features evolved/generated by the GP-criptor$^{\text{ri}}$ method can improve the performance of different types of learning/classification methods applied to images with different rotations; and
- Investigate to what extent the evolved image descriptors/genetic programs can be interpreted by humans.

5.1.2 Chapter Organisation

The remainder of the chapter is organised as follows. The proposed method is described in Section 5.2. The experiment design is explained in Section 5.3. The

results are presented and discussed in Section 5.4. To provide some in-depth analysis, an example program evolved by the proposed method is extensively examined in Section 5.5. The summary of this chapter is presented in Section 5.6.

5.2 GP Image Descriptor

This section provides a detailed description of the proposed *rotation-invariant* GP-criptor (GP-criptor$^{\text{ri}}$) method. The section starts by presenting an overview of the algorithm to evolve a program in order to highlight the key components of GP-criptor$^{\text{ri}}$, and how the evolved program is evaluated. Then the program structure, i.e., terminal and function sets, fitness measure, and feature vector extraction process, are discussed.

The proposed method operates directly on the raw pixel values, and therefore it does not require human intervention to provide a set of predefined/extracted features. Unlike methods designed by domain-experts, the proposed method does not use domain knowledge to detect a specific set of keypoints such as lines, corners, spots, or homogeneous regions. Instead, GP-criptor$^{\text{ri}}$ automatically discovers good keypoints that vary in their frequency of appearance between the instances of the different classes. Moreover, GP-criptor$^{\text{ri}}$ does not require human intervention to manually combine the detected keypoints (like in LBP and GLCM) and design them to be rotation-invariant. Instead, this method automatically synthesises a set of mathematical formulae to accomplish this task. Another key feature of the proposed method is that it does not need a large number of training instances to evolve a descriptor, which makes it suitable for applications where the labelled data is limited. In fact, being capable of operating on just a few instances has a large impact on reducing the training costs, i.e., memory and CPU time.

5.2.1 The Overall Algorithm

As GP-criptor$^{\text{ri}}$ is extended from GP-criptor, the two methods share some components. The overall algorithm of the GP-criptor$^{\text{ri}}$ is identical to that of GP-criptor (see Section 4.2.1 on page 119). The main difference is in the program representation, and hence, the other components are kept unchanged.

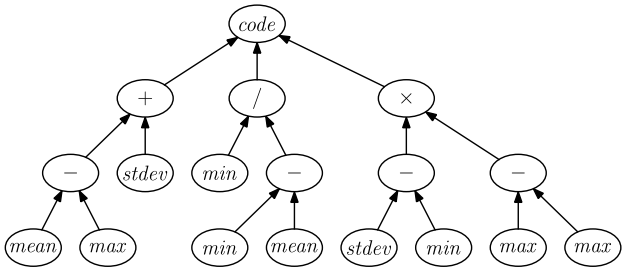


Figure 5.1: The program representation of an individual evolved by GP-criptor$^{\text{ri}}$.

5.2.2 Program Representation

In GP, the individual programs are constructed from elements of the terminal and function sets. Here, tree based GP [153] is used to represent an individual (i.e. image descriptor) evolved by GP-criptor$^{\text{ri}}$, where the terminal nodes, i.e., leaves, are taken from the terminal set, and all non-terminal nodes are drawn from the function set. Figure 5.1 depicts an example of a GP-criptor$^{\text{ri}}$ evolved individual. Moreover, strongly-typed GP [207] is used to introduce restrictions on the nodes. Each individual is a set of synthesised formulae that are used to extract the feature vector (more details in Section 5.2.3).

5.2.2.1 Terminal Set

In GP-criptor$^{\text{ri}}$, the terminal set consists of four node types: $min\,(\vec{x})$, $max\,(\vec{x})$, $mean\,(\vec{x})$, and $stdev\,(\vec{x})$, which are functions that respectively return the minimum, maximum, mean, and standard deviation values of the elements of a vector. The intuition behind choosing these functions is their order-independent property when extracting features. In other words, shuffling the values of the vector will not affect the results returned by those functions. This is very important to handle the rotation variants of the pixels. The terminal nodes take a vector of integer values, and return a single floating point value. It is worth noting that the design of GP-criptor$^{\text{ri}}$ is not limited to only these four functions and other functions that have the same property, i.e., order-independence, can be used.

The terminal set is a key difference between GP-criptor (Chapter 4) and GP-criptor$^{\text{ri}}$. In GP-criptor, the leaf nodes of an evolved program are the original pixel

values in a chosen index of the sliding window as presented in Figure 4.2; whereas the leaf nodes in GP-criptor^{ri} are calculated statistics of the pixel values of the sliding window as shown in Figure 5.1.

5.2.2.2 Function Set

The function set of GP-criptor^{ri} is keep identical to that of GP-criptor (see Section 4.2.2 on page 120). Therefore, the function set is made up of five nodes: *code* and the four arithmetic operators $+$, $-$, $/$, and $\times$.

5.2.3 Feature Vector Extraction

A core task of an individual evolved by GP-criptor^{ri} is to automatically detect keypoints and extract a feature vector from an instance, i.e., an image, being evaluated using a sliding window of predetermined size. This process is also very similar to the way GP-criptor generates a feature vector for an image (see Section 4.2.3 on page 122). The length of the feature vector depends on the number of children of the *code* node (the root of the individual tree). If there are n nodes in the children list of *code*, then the resulting vector for each instance is of length 2^n. As demonstrated in Figure 5.2, the instance undergoes five steps at each position of the sliding window traversing the instance pixel-by-pixel row-wise starting from the top-left corner and ending at the bottom-right corner.

Step 1: The minimum, maximum, mean, and standard deviation values of the current window pixels are calculated.

Step 2: Those calculated values are fed to the terminal nodes of the individual.

Step 3: The internal (non-terminal) nodes, apart from the root node, are evaluated starting from those near the leaves by applying the corresponding operator to the list of arguments, i.e., children.

Step 4: The root node (i.e. *code*) returns a binary code by converting each of its arguments to 0 if it is negative and 1 otherwise.

Step 5: The generated binary code is converted to decimal, and the corresponding bin of the feature vector (histogram) is incremented by 1.

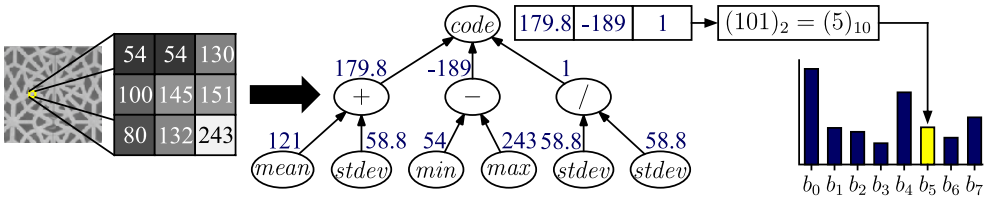


Figure 5.2: An example demonstrating the process of generating a feature vector for an image by the GP-criptor$^{\text{ri}}$ method.

Clearly the *code* node in this context mimics the thresholding step of the conventional LBP descriptor. However, the latter uses the central pixel's value as a threshold; whilst 0 is used as a threshold value in the former.

5.2.4 Fitness Function

Typically, the classification accuracy is used as the fitness measure to gauge the performance of the individual to discriminate between instances of different classes. Accuracy is defined as the ratio between the number of correctly classified instances and the total number of instances.

The use of accuracy to measure the fitness may not be a good option when there are only a few examples in the training set as the algorithm will simply memorise those examples, which can increase the possibility of "over-fitting" occurring and affect the generalisability of the evolved program on the unseen data. Therefore, a different measure is needed to cope with the problem of having only a limited number of training instances. Clearly, we need a fitness function that can detect as many representative keypoints as possible that reliably separate the instances of different classes farther apart, and keeps the distances between instances of a particular class as close to each other as possible [195]. Hence, the fitness measure that is used in this chapter is the same as Section 4.2.5 (see page 123).

5.3 Experiment Design

The aim and design of the experiments are discussed in this section. The discussion also includes the datasets, methods for comparison, and parameter settings.

5.3.1 Datasets

GP-criptor$^{\text{ri}}$ is evaluated using six image datasets for texture classification and two non-texture datasets. The instances of all those image datasets are grey-scale images, i.e., each pixel carries only brightness/intensity information that can be white at the strongest intensity or black at the weakest intensity [138]. Therefore, the pixel values are ranging between 0 (black) and 255 (white). Those eight image datasets vary in number of classes (binary and multi-class), size of instances (as small as 19×19 pixels and as large as 128×128 pixels), and applications (texture classification, face classification, and coin head and tail classification).

The first (BrNoRo) and second (BrWiRo) datasets in this chapter are taken from the Brodatz Texture [42] (see Section 2.4 on page 62). Similarly, the third (KyNoRo) and fourth (KyWiRo) datasets are formed using the instances of Kylberg Texture [157]. OutexTC00 and OutextTC10 are, respectively, the fifth and sixth datasets in this chapter that are formed from Outex Texture Classification [216]. In addition to the aforementioned datasets, two datasets have been used to assess the performance of GP-criptor$^{\text{ri}}$ on different types of application other than texture classification. Both of these datasets are for binary classification (two classes). Therefore, the seventh and eighth datasets are Faces, and Coins, respectively. The details of these datasets are provided in Section 2.4 (see page 62).

5.3.2 Benchmark Methods for Comparison

In order to investigate the effectiveness of the GP-criptor$^{\text{ri}}$ method, its performance is compared to the performance of the common state-of-the-art descriptors DIF, GLCM, $\text{LBP}^{u2}_{p,r}$, $\text{LBP}^{riu2}_{p,r}$, $\text{CLBP}_{p,r}$, $\text{LBC}_{p,r}$, and $\text{CLBC}_{p,r}$ (see Section 2.1.2 on page 29). Image descriptors are used as pre-processing methods to convert an instance from raw pixel values into a feature vector after detecting some keypoints. Hence, researchers broadly rely on classification or recognition to assess the goodness of a descriptor [217, 185, 4, 26]. Here, we also used the classification performance to assess whether the proposed method is capable of evolving good descriptors. We have intentionally selected classifiers of different types in this study to ensure that the GP-criptor$^{\text{ri}}$ method is not biased towards a specific classifier or type of classifier. Those classifiers are (see Section 3.3.3.1 on page 89) Support Vector

Table 5.1: The GP parameters

Parameter	Value	Parameter	Value
Generations	50	Crossover Rate	0.80
Population Size	200	Mutation Rate	0.20
Minimum Depth	2	Maximum Depth	10
Selection Type	Tournament	Reproduction	Keep the best
Tournament size	5	Initial Population	Half-and-half

Machines (SVM), Naïve Bayes (NB), Adaptive Boosting (AdaBoost), Decision Trees (J48), Random Forest (RF), Naïve Bayes/Decision Tree (NBTree), KStar (K*), Non-Nested generalised (NNge), k-Nearest Neighbour (k-NN), and Multilayer Perceptron (MLP).

5.3.3 Parameter Settings

The methods used for comparison in this chapter as well as the proposed method contain a number of parameters that need to be set. The parameter settings of the proposed method are discussed first, followed by a discussion on setting those of the other methods.

5.3.3.1 Evolutionary parameters of the proposed method

Apart from the population size, the GP evolutionary parameters are kept identical to that of GP-criptor (see Section 4.3.3 on page 127) as presented in Table 5.1.

5.3.3.2 Parameters of the baseline methods

Based on the observations of [339] and [251], the radius (r) and the number of neighbouring pixels (p) for LBP, CLBP, LBC and CLBC have been, respectively, set to 3 and 24. This combination has shown very good performance in most cases in both studies. Hence, we have also used the same settings in our experiments. For the $\text{LBP}^{u2}_{p,r}$ method, on the other hand, our experiments show that this method has achieved good performance when r is set to 1 and p to 8.

5.3.3.3 Parameters of the classifiers

The parameters of all the classifiers used in this chapter are kept identical with Section 4.3.3 (see page 127) for comparison purposes.

5.3.4 Experiments

The aim of the proposed method is to automatically evolve an image descriptor that generates distinctive feature vectors for instances belonging to different classes. Therefore, two sets of experiments are designed each of which aims at investigating a specific aspect, e.g., window size and code length. On each dataset, the GP-criptor$^{\text{ri}}$ method is executed 30 times using different random seeds, and the performance of the best evolved program at each run is recorded. Then the average performance ($\bar{x} \pm s$) of those 30 best programs is calculated. The training instances (2 instances per class) are *randomly* selected, hence using different instances could give different results. Therefore, the process of 30 runs is repeated 10 times using different instances for training each time. The experiments have been executed on the grid-computing facility provided by Victoria University of Wellington. This grid runs under the *Sun Grid Engine* (SGE) control, and consists of a large number of machines that are running Linux version 3.7.5-1-ARCH operating system with an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz and a 8GByte of memory each. It is important to notice that the measured time in all our experiments was the CPU time and not the wall-clock time. The first experiment can be considered as a *parameter-tuning* phase, where in total we have 9 combinations of different window sizes and code lengths (details are below). Hence, we have $21,600$ (experiments) in total, i.e., 8 (datasets) $\times$ 9 (combinations) $\times$ 10 (repetitions) $\times$ 30 (runs). Using one more window size or code length will require 900 runs to be added, which is a very time consuming process.

5.3.4.1 Window size and code length

In the first set of experiments, the impacts of changing the window size and the code length on the performance of GP-criptor$^{\text{ri}}$ are studied. Changing the sliding window size allows a different number of pixels to contribute towards calculating the code at each position. Therefore, three window sizes are tested: 3×3, 5×5, and

7×7. The use of other window sizes is possible and the system can automatically handle it. Experimenting with more window sizes is very expensive and require 300 (= 30 (seeds) $\times$ 10 (repetitions)) independent runs. Moreover, typically LBP methods are tested with those sizes in the literature.

The number of bits of the binary code, on the other hand, is the only factor that specifies the feature vector length (2^n where n is the number of bits/children of *code*), therefore, the length is doubled for each extra bit added to the code. We have experimented with three code lengths: 7, 8, and 9. Unlike LBP methods, the use of different code lengths is possible without the need for changing the implementation of the method. Experimenting with more code lengths is also expensive similar to the window size factor.

5.3.4.2 Image classification

The second set of experiments is designed to test whether using a simple instance-based classifier, i.e., the 1-NN (k-NN with k set to 1), with only a few learning examples can achieve comparable or even better performance than using LBP^{u2}, LBP^{riu2}, CLBP, LBC, CLPC, DIF, and GLCM with more powerful classifiers such as SVM, K*, RF, AdaBoost, NNge, NB, NBTree, J48, and MLP. The proposed method heavily relies on the between-class and within-class distances as the main criteria to evolve a good descriptor. Hence, the impact of the features generated by the proposed method on the performance of each of the aforementioned nine classifiers is also studied. This will help in identifying whether those features are biased toward a specific type (i.e. instance-based or k-NN-like) of classifier or not.

5.4 Results and Discussions

The results of the experiments are presented and discussed in this section.

5.4.1 Window Size and Code Length

The effect of the size of the sliding window and the code length on the performance are not independent. The results of each dataset are presented in a single 3D bar chart, where the x-axis is the code length (number of bits), y-axis is the window

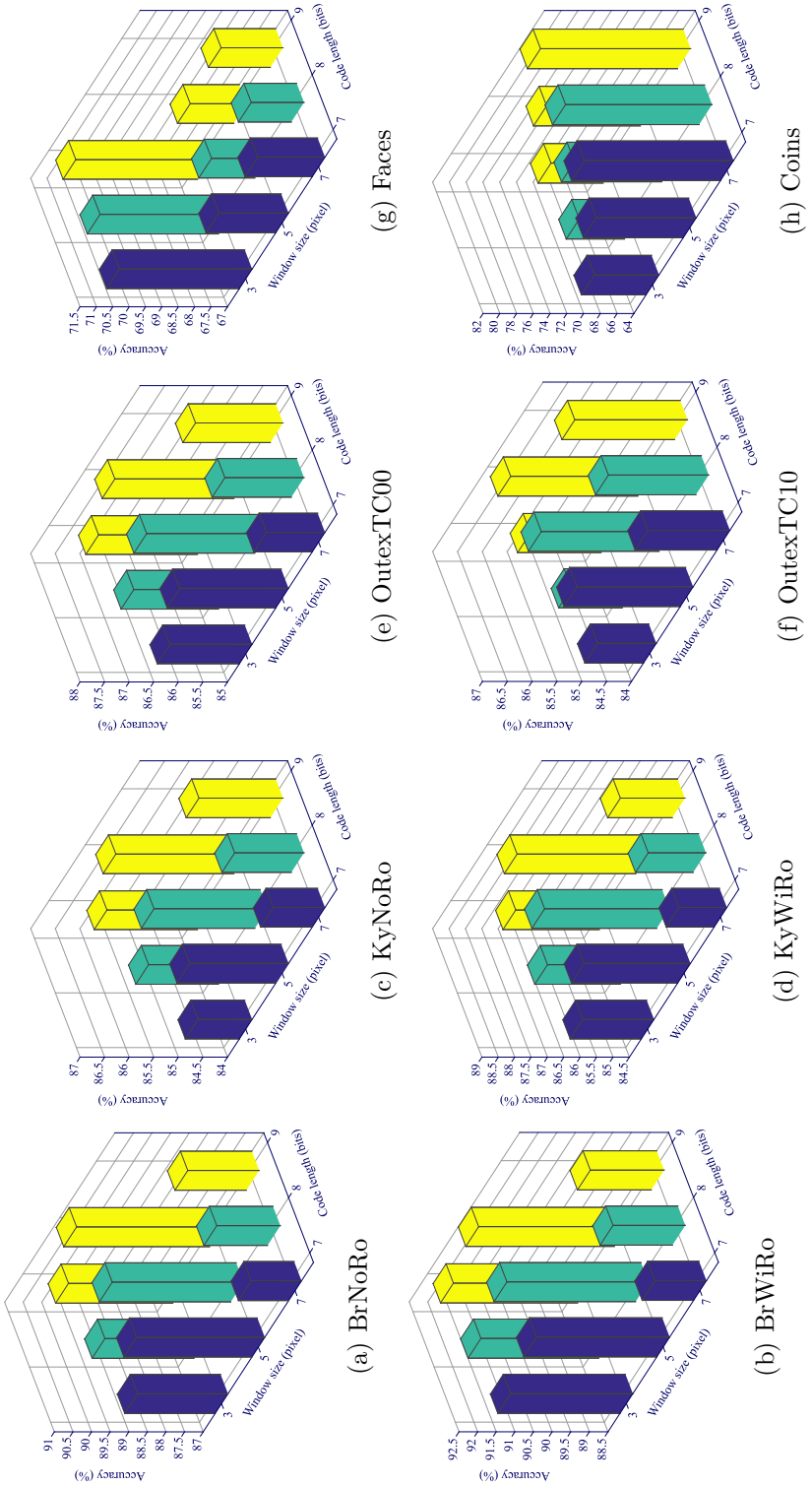
Table 5.2: Accuracy (%) for using different combinations of code lengths and window sizes on the six datasets ($\bar{x} \pm s$).

		Window size		
Dataset	Code length	3×3 pixels	5×5 pixels	7×7 pixels
BrNoRo	7-bits	89.43 ± 1.98	90.44 ± 1.88	88.35 ± 2.00
	8-bits	89.74 ± 2.13	90.53 ± 1.80	88.71 ± 1.98
	9-bits	90.15 ± 2.06	90.92 ± 1.94	88.92 ± 2.04
BrWiRo	7-bits	91.78 ± 1.35	92.07 ± 1.14	89.88 ± 1.01
	8-bits	92.01 ± 1.37	92.30 ± 1.14	90.44 ± 1.00
	9-bits	92.18 ± 1.42	92.49 ± 1.14	90.48 ± 0.92
KyNoRo	7-bits	85.10 ± 2.71	86.01 ± 1.87	85.05 ± 1.92
	8-bits	85.68 ± 2.73	86.31 ± 1.80	85.41 ± 1.84
	9-bits	86.10 ± 2.65	86.66 ± 1.79	85.71 ± 1.78
KyWiRo	7-bits	86.67 ± 1.91	87.74 ± 1.31	85.96 ± 0.99
	8-bits	87.11 ± 1.92	88.31 ± 1.24	86.26 ± 1.21
	9-bits	87.44 ± 1.91	88.51 ± 1.39	86.47 ± 1.12
OutexTC00	7-bits	86.67 ± 2.53	87.21 ± 1.79	86.19 ± 2.14
	8-bits	86.98 ± 2.41	87.46 ± 1.83	86.61 ± 2.17
	9-bits	87.27 ± 2.57	87.68 ± 1.87	86.78 ± 2.23
OutexTC10	7-bits	85.18 ± 1.56	86.34 ± 1.79	85.65 ± 1.80
	8-bits	85.28 ± 1.62	86.63 ± 1.84	86.02 ± 1.79
	9-bits	85.68 ± 1.59	86.82 ± 1.93	86.27 ± 1.91

size, and z-axis is the average accuracy (%) over 300 independent runs as shown in Table 5.2 and Figure 5.3.

GP-criptor$^{\text{ri}}$ has achieved on average the minimum of 88.35% and 89.88% accuracy on the BrNoRo and BrWiRo datasets, respectively, using the combination of window size 7×7 and code length 7-bits as depicted in Figure 5.3(a),(b). The maximum average accuracies on these datasets were achieved when the window size is set to 5×5 and a 9-bits code length, which are 90.92% and 92.49% respectively.

Similarly, using a window of size 7×7 pixels and a code of length 7-bits, GP-criptor$^{\text{ri}}$ on the KyNoRo and KyWiRo datasets has achieved the lowest average performance that are, respectively, 85.05% and 85.96% as shown in Figure 5.3(c),(d). The best average performances are scored on these two datasets when the code length is increased to 9-bits and the window size is reduced to 5×5 pixels, which are 86.66% (KyNoRo) and 88.51% (KyWiRo).



(a) BrNoRo (b) BrWiRo (c) KyNoRo (d) KyWiRo (e) OutexTC00 (f) OutexTC10 (g) Faces (h) Coins

Figure 5.3: The results of the first experiment, which presents the impact of the window size and code length on the performance on the (a) BrNoRo, (b) BrWiRo, (c) KyNoRo, (d) KyWiRo, (e) OutexTC00, (f) OutexTC10, (g) Faces, and (h) Coins datasets.

While the results of the OutexTC00 dataset show a similar pattern to that of the previous four where a minimum average performance (86.19%) is achieved with a window of size 7×7 pixels and code length 7-bits, the OutexTC10 dataset results show a different pattern as the minimum performance was achieved with a window of size 3×3 and 7-bits code length as, respectively, presented in Figure 5.3(e),(f). However, the combination of a 5×5 pixels window and 9-bits code length still gives the best average accuracies 87.68% and 86.82%, respectively.

The Faces and Coins datasets also follow the same pattern regarding the length of the code such that the performance is improved when the number of bits in the code increases as shown in Figure 5.3(g),(h). However, increasing the window size has a completely different influence on the performance on these two datasets. Increasing the window size degrades the performance in the case of the Faces dataset, whilst large windows give better performance than small windows in Coins. The cause of this phenomenon is investigated and discussed in Section 5.4.3.

In summary, the six texture datasets show a similar pattern, that is, a better performance has been achieved with a code of length 9-bits than that of length 7- and 8-bits as depicted in Figure 5.3(a)–(f). Similarly, the proposed method has achieved slightly better performance when the window size is 5×5 pixels than the other two experimented sizes, i.e., 3×3 and 7×7 pixels. Hence, in our subsequent experiments we have used the combination of window size 5×5 and code length 9-bits as it has been shown to give the best average performance. A code of length 9-bits and a sliding window with size 3×3 pixels is used in the case of the Faces dataset. The Coins dataset has been further experimented using larger window sizes and is discussed in Section 5.4.3.

5.4.2 Texture Image Classification

The results for the second set of experiments on the six texture datasets are presented in Tables 5.3 to 5.5. Horizontally each table is made up of 11 columns (one to list the descriptors and 10 for the different classifiers). The values in these tables are the average accuracy (over 300 independent runs) percentage $\pm$ the standard deviation resulting from using a classifier (column) with an image descriptor (row).

These results are statistically tested using the *Wilcoxon signed-rank* test [320, 66, 68] with a significance level of 5%. The statistical test has been applied twice, first, to check whether the proposed method with a simple classifier (1-NN) can compete with the baseline methods (i.e. descriptors) with more powerful classifiers; and second, to test whether the GP-criptor$^{\text{ri}}$ method can compete with the baseline methods using the same classifier. The symbols "$*$" and "$-$" are used to, respectively, represent significantly better and significantly worse in the first test, whereas significantly better and significantly worse in the second test are indicated by "↑" and "↓", respectively. For each dataset, the corresponding method with best average performance for each classifier is made **bold**; whilst the best performance amongst all methods and classifiers is underlined. If two or more methods have the same best average performance, the one with the smallest standard deviation is underlined.

For all texture image datasets, the code length is set to 9-bits and a window of size 5×5 pixels is used as the majority of these datasets have performed well with this combination in preliminary testing. Similarly, the code length is set to 9-bits for non-texture datasets; however, the window size is set to 3×3 and 7×7 for Faces and Coins datasets, respectively.

5.4.2.1 BrNoRo

On the BrNoRo dataset, the proposed method with 1-NN has achieved on average 90.9% accuracy which is significantly better than all other methods as presented in the top half of Table 5.3. Meanwhile, those features extracted by the proposed method have positive influence on the performance of all other classifiers, apart from MLP with $\text{CLBP}_{24,3}$, compared to the performances achieved using the baseline methods.

5.4.2.2 BrWiRo

The bottom half of Table 5.3 shows the results of the BrWiRo dataset which is the rotated version of BrNoRo. The proposed method with 1-NN has achieved the best performance over all other methods with more sophisticated classifiers. On average, the GP-criptor$^{\text{ri}}$ method has scored 92.5% which is comparable to the performance

on the without-rotation version, i.e., BrNoRo, of this dataset. This reveals the ability of the GP-criptor^{ri} method to handle rotations. Although the GP-criptor^{ri} method has slightly degraded the performance of MLP and SVM with $CLBP_{24,3}$, using the extracted features by GP-criptor^{ri} with the other classifiers shows a significant improvement in their performances compared to other descriptors.

5.4.2.3 KyNoRo

The results presented in the top half of Table 5.4 correspond to the KyNoRo dataset. The proposed method shows on average 86.7% accuracy on the unseen data of KyNoRo. Meanwhile, both $CLBP_{24,3}$ and $CLBC_{24,3}$ have significantly outperformed the GP-criptor^{ri} method by scoring on average 90.3% and 91.1%, respectively. The same also happened when the features extracted by GP-criptor^{ri} are used with MLP, NNge, and SVM. In all other cases, the impact of the GP-criptor^{ri} features either significantly improved the performance or achieved a comparable level, i.e., best or in the top-three ranked performances, to that of the expert designed methods.

5.4.2.4 KyWiRo

The results of the experiments on the rotated version of KyNoRo (KyWiRo) are listed in the bottom half of Table 5.4. The results on this dataset are quite similar to those that have been observed on KyNoRo. The performance achieved by any classifier using the proposed method's features is either the first or in the top-three best performances compared to the use of hand-crafted descriptors.

5.4.2.5 OutexTC00

The results on OutexTC00 are presented in the top half of Table 5.5. GP-criptor^{ri} has scored the second overall best performance (the first is $LBP^{u2}_{8,1}$ with NNge) with 87.7% accuracy on this dataset, and has outperformed all the competitor methods.

5.4.2.6 OutexTC10

On the OutexTC10 dataset, the proposed method has achieved the best performance with 86.8% accuracy on average over all the baseline methods as shown in the

Table 5.3: Accuracy (%) of ten classifiers using eight image descriptors on the **BrNoRo** and **BrWiRo** texture image datasets ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
BrNoRo	DIF	36.8 ± 2.7*	18.0 ± 3.6*↑	28.5 ± 4.9*↑	36.4 ± 3.2*↑	35.3 ± 2.5*↑	21.0 ± 6.4*↑	33.4 ± 4.7*↑	34.3 ± 3.5*↑	31.7 ± 3.5*↑	33.5 ± 2.8*↑
	$LBP^{u2}_{8,1}$	68.9 ± 3.8*	37.2 ± 5.9*↑	25.4 ± 3.9*↑	61.9 ± 5.3*↑	69.7 ± 2.5*↑	52.1 ± 8.4*↑	64.0 ± 7.7*↑	71.9 ± 3.3*↑	51.9 ± 3.6*↑	60.2 ± 4.4*↑
	GLCM	51.5 ± 7.3*	17.0 ± 8.6*↑	33.6 ± 6.1*↑	48.9 ± 7.7*↑	53.6 ± 7.0*↑	38.7 ± 6.5*↑	44.6 ± 6.3*↑	48.0 ± 5.1*↑	43.9 ± 6.2*↑	47.3 ± 6.3*↑
	$LBP^{riu2}_{24,3}$	66.7 ± 2.2*	22.0 ± 6.6*↑	37.4 ± 2.8*↑	60.9 ± 2.3*↑	69.6 ± 2.8*↑	36.8 ± 2.8*↑	46.2 ± 4.2*↑	65.0 ± 3.0*↑	48.4 ± 2.7*↑	61.7 ± 3.2*↑
	$CLBP_{24,3}$	83.7 ± 3.7*	36.5 ± 4.9*↑	33.7 ± 3.8*↑	79.5 ± 3.6*↑	**83.6 ± 3.2***	69.7 ± 5.8*↑	76.6 ± 3.6*↑	84.1 ± 4.0*	60.9 ± 1.8*↑	69.0 ± 5.1*
	$LBC_{24,3}$	64.1 ± 3.0*	22.6 ± 6.1*↑	36.2 ± 3.7*↑	58.2 ± 2.4*↑	67.7 ± 2.7*↑	33.4 ± 6.1*↑	41.6 ± 5.5*↑	60.8 ± 2.6*↑	46.6 ± 1.9*↑	60.3 ± 4.1*↑
	$CLBC_{24,3}$	78.6 ± 4.0*	35.4 ± 3.0*↑	34.9 ± 5.0*↑	78.1 ± 3.8*↑	79.4 ± 4.4*↑	66.9 ± 7.0*↑	74.9 ± 3.2*↑	79.4 ± 4.0*↑	59.3 ± 1.5*↑	60.1 ± 5.8*↑
	GP-criptor^{ri}	**90.9 ± 1.9**	**60.9 ± 3.1***	**50.6 ± 1.2***	**85.4 ± 1.4***	82.4 ± 1.9*	**79.9 ± 2.6***	**82.2 ± 1.4***	**85.7 ± 1.5***	**69.9 ± 1.7***	**70.5 ± 2.3***
BrWiRo	DIF	36.5 ± 2.9*	14.8 ± 4.1*↑	30.9 ± 4.1*↑	34.9 ± 2.5*↑	32.5 ± 2.4*↑	21.2 ± 5.8*↑	34.1 ± 4.1*↑	34.2 ± 4.3*↑	34.1 ± 3.6*↑	34.1 ± 3.4*↑
	$LBP^{u2}_{8,1}$	37.6 ± 1.4*	23.7 ± 2.3*↑	18.9 ± 3.0*↑	31.1 ± 1.5*↑	36.3 ± 1.9*↑	24.6 ± 4.8*↑	37.7 ± 5.0*↑	40.6 ± 1.9*↑	30.1 ± 1.8*↑	33.5 ± 3.0*↑
	GLCM	41.1 ± 6.4*	13.9 ± 5.3*↑	27.8 ± 4.7*↑	39.9 ± 7.3*↑	38.7 ± 4.6*↑	29.1 ± 5.6*↑	39.6 ± 4.6*↑	37.9 ± 5.0*↑	35.3 ± 4.5*↑	38.2 ± 6.0*↑
	$LBP^{riu2}_{24,3}$	68.3 ± 3.4*	21.8 ± 9.5*↑	41.4 ± 2.3*↑	62.7 ± 3.6*↑	68.5 ± 2.7*↑	40.1 ± 4.9*↑	50.8 ± 3.4*↑	66.6 ± 1.9*↑	51.5 ± 1.8*↑	62.2 ± 4.3*↑
	$CLBP_{24,3}$	86.1 ± 3.1*	38.6 ± 4.3*↑	33.8 ± 3.8*↑	78.9 ± 3.5*↑	**86.3 ± 3.3***	73.9 ± 2.5*↑	79.4 ± 2.8*↑	85.3 ± 2.8*	62.1 ± 1.6*↑	**72.4 ± 6.1***
	$LBC_{24,3}$	64.0 ± 3.5*	22.7 ± 8.3*↑	36.9 ± 3.9*↑	59.5 ± 4.0*↑	65.1 ± 2.3*↑	35.3 ± 4.3*↑	47.0 ± 2.3*↑	62.5 ± 2.5*↑	48.0 ± 1.9*↑	57.7 ± 3.3*↑
	$CLBC_{24,3}$	84.9 ± 2.7*	36.5 ± 6.4*↑	36.7 ± 4.1*↑	82.9 ± 3.0*↑	84.6 ± 2.4*	75.0 ± 3.4*↑	78.7 ± 3.7*↑	84.1 ± 2.1*↑	62.3 ± 1.2*↑	68.7 ± 6.3*
	GP-criptor^{ri}	**92.5 ± 1.1**	**59.7 ± 3.4***	**48.9 ± 0.9***	**86.7 ± 1.5***	83.7 ± 1.4*	**81.1 ± 3.0***	**83.2 ± 0.9***	**86.5 ± 2.0***	**69.6 ± 1.2***	71.3 ± 1.8*

Table 5.4: Accuracy (%) of ten classifiers using eight image descriptors on the **KyNoRo** and **KyWiRo** texture image datasets ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
KyNoRo	DIF	22.2 ± 1.7∗	10.5 ± 2.7∗↑	28.4 ± 1.8∗↑	18.7 ± 1.1∗↑	23.0 ± 2.0∗↑	19.4 ± 2.8∗↑	20.0 ± 2.6∗↑	27.1 ± 2.5∗↑	22.4 ± 0.9∗↑	21.1 ± 1.8∗↑
	$LBP^{u2}_{8,1}$	62.9 ± 4.2∗	31.1 ± 7.4∗↑	29.2 ± 3.3∗↑	67.2 ± 4.0∗↑	63.8 ± 3.2∗↑	62.3 ± 4.0∗↑	61.6 ± 4.3∗↑	65.0 ± 3.5∗↑	54.8 ± 1.6∗↑	57.8 ± 4.1∗↑
	GLCM	68.0 ± 3.0∗	18.2 ± 3.9∗↑	**44.5 ± 5.0**∗	68.9 ± 1.8∗↑	64.5 ± 3.7∗↑	48.1 ± 5.7∗↑	55.1 ± 3.1∗↑	65.7 ± 3.8∗↑	56.0 ± 2.1∗↑	67.0 ± 2.8∗
	$LBP^{riu2}_{24,3}$	68.2 ± 2.7∗	21.5 ± 4.7∗↑	43.5 ± 2.7∗↓	67.0 ± 2.5∗↑	67.9 ± 2.0∗↑	40.9 ± 4.1∗↑	51.6 ± 3.0∗↑	65.3 ± 3.2∗↑	55.1 ± 2.2∗↑	65.9 ± 2.1∗
	$CLBP_{24,3}$	90.3 ± 0.7−	43.5 ± 1.9∗↑	34.9 ± 3.1∗↑	40.1 ± 2.8∗↑	90.3 ± 1.2−↓	66.6 ± 14.4∗	**78.7 ± 3.3**∗↓	90.1 ± 1.4−↓	63.7 ± 0.8∗	78.5 ± 4.5∗↓
	$LBC_{24,3}$	67.5 ± 3.0∗	17.8 ± 2.7∗↑	41.0 ± 4.1∗	65.4 ± 2.8∗↑	69.2 ± 2.7∗↑	44.5 ± 4.1∗↑	53.0 ± 3.6∗↑	65.6 ± 2.9∗↑	53.5 ± 1.7∗↑	64.8 ± 3.4∗
	$CLBC_{24,3}$	**91.1 ± 1.1**−	41.4 ± 3.0∗↑	36.3 ± 4.2∗↑	72.0 ± 2.7∗↑	**90.7 ± 1.3**−↓	66.8 ± 14.3∗	76.3 ± 3.4∗	**91.1 ± 1.3**−↓	**64.5 ± 1.3**∗	**79.2 ± 3.7**∗↓
	GP-criptor^{ri}	86.7 ± 1.8	**56.8 ± 2.5**∗	41.1 ± 1.3∗	**82.7 ± 1.8**∗	78.8 ± 1.7∗	**70.7 ± 4.2**∗	75.1 ± 1.6∗	82.6 ± 2.1∗	64.4 ± 1.2∗	66.9 ± 1.7∗
KyWiRo	DIF	23.0 ± 0.9∗	12.8 ± 2.3∗↑	28.5 ± 2.9∗↑	19.8 ± 0.9∗↑	20.0 ± 1.6∗↑	21.9 ± 0.9∗↑	22.1 ± 1.4∗↑	22.0 ± 0.1∗↑	23.3 ± 0.5∗↑	17.7 ± 2.7∗↑
	$LBP^{u2}_{8,1}$	41.0 ± 0.6∗	21.2 ± 1.9∗↑	23.7 ± 2.2∗↑	37.7 ± 0.6∗↑	36.3 ± 1.1∗↑	44.1 ± 1.9∗↑	46.4 ± 3.8∗↑	40.7 ± 1.4∗↑	37.2 ± 0.4∗↑	32.5 ± 1.3∗↑
	GLCM	49.0 ± 0.4∗	16.7 ± 2.8∗↑	32.8 ± 2.1∗↑	49.8 ± 0.7∗↑	47.3 ± 0.3∗↑	33.8 ± 2.8∗↑	41.8 ± 2.0∗↑	46.1 ± 0.4∗↑	40.0 ± 0.9∗↑	47.0 ± 1.1∗↑
	$LBP^{riu2}_{24,3}$	69.1 ± 2.2∗	25.0 ± 7.3∗↑	**43.3 ± 4.8**∗	66.4 ± 2.6∗↑	67.1 ± 2.8∗↑	42.9 ± 7.4∗↑	51.6 ± 3.9∗↑	67.2 ± 3.0∗↑	54.7 ± 2.5∗↑	66.9 ± 3.5∗
	$CLBP_{24,3}$	90.6 ± 2.2−	38.9 ± 2.1∗↑	34.2 ± 2.6∗↑	39.2 ± 2.3∗↑	**91.4 ± 1.9**−↓	72.4 ± 5.1∗	77.8 ± 6.5∗	90.6 ± 2.4−↓	62.2 ± 1.1∗↑	77.9 ± 5.4∗↓
	$LBC_{24,3}$	68.2 ± 1.9∗	17.7 ± 3.4∗↑	39.2 ± 4.0∗	65.1 ± 2.4∗↑	65.9 ± 3.0∗↑	42.7 ± 6.8∗↑	50.7 ± 3.6∗↑	66.2 ± 2.7∗↑	53.0 ± 2.0∗↑	65.6 ± 2.7∗↑
	$CLBC_{24,3}$	**91.0 ± 2.1**−	37.9 ± 4.9∗↑	36.3 ± 4.8∗↑	71.4 ± 3.4∗↑	91.2 ± 2.2−↓	**73.1 ± 4.5**∗	**78.6 ± 5.3**∗	**91.2 ± 2.7**−↓	63.8 ± 1.3∗↑	**79.9 ± 4.1**∗↓
	GP-criptor^{ri}	88.5 ± 1.4	**56.2 ± 1.4**∗	41.1 ± 1.7∗	**84.4 ± 1.8**∗	80.5 ± 1.2∗	72.7 ± 3.8∗	76.3 ± 1.6∗	84.0 ± 1.4∗	**65.3 ± 1.1**∗	68.3 ± 1.8∗

bottom half of Table 5.5. Furthermore, 7 out of 10 classifiers, i.e., 1-NN, AdaBoost, J48, K*, NB, NBTree, and RF, are ranked number one when GP-criptor^{ri} features are used. The differences are also significant in most of these cases.

5.4.3 Non-texture Image Classification

All previously discussed datasets are for texture classification. Many descriptors in the literature have been originally designed to detect and extract features from such images, and some typical examples are LBP and its variants, and GLCM (Haralick features). In order to study the ability of the proposed method to handle different applications than texture classification, the Faces and Coins datasets are used as discussed in Section 5.3.1.

5.4.3.1 Faces

Using a code of length 9-bits and a 3×3 pixels window, the results on the Faces dataset are depicted in Table 5.6. The proposed method was not the best method on this dataset. However, its performance is the second best when the 1-NN classifier is used. Moreover, GP-criptor^{ri} has significantly better performance than most of the other methods with more powerful classifiers. Similarly, using the GP-criptor^{ri} features with other classifiers, apart from AdaBoost, has either significantly improved or only slightly degraded the performances of those classifiers. This demonstrates the capability of the proposed method to handle datasets of different flavours. Figure 5.3(g) shows that the best performance on this dataset is achieved when the window size is 3×3 pixels, and increasing the window size has degraded the performance. This was expected mainly because the size of this dataset's instances is relatively smaller than all other datasets (19×19 pixels). Smaller windows allow the system to detect more specific (less cluttered) and effective keypoints than larger windows. The eye region is one of those very important regions of a human face and domain-experts commonly select these as features for face detection tasks. If the window is small enough, it can effectively detect the eye regions without including the surrounding regions, as demonstrated in Figure 5.4.

Table 5.5: Accuracy (%) of ten classifiers using eight image descriptors on the **OutexTC00** and **OutexTC10** texture image datasets ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
OutexTC00	DIF	17.4 ± 1.7∗	9.0 ± 3.3∗↑	13.2 ± 4.7∗↑	14.0 ± 3.0∗↑	15.4 ± 2.1∗↑	11.1 ± 2.8∗↑	12.3 ± 4.1∗↑	15.7 ± 2.2∗↑	13.0 ± 1.5∗↑	14.7 ± 2.9∗↑
	$LBP^{u2}_{8,1}$	87.6 ± 3.3	45.5 ± 8.7∗↑	33.1 ± 8.8∗↑	85.5 ± 2.5∗	**84.9 ± 3.0**∗	75.8 ± 5.8∗	76.1 ± 4.2∗↑	**89.3 ± 1.6** ↓	65.5 ± 6.2∗↑	**80.3 ± 7.2**∗↓
	GLCM	70.5 ± 2.4∗	20.6 ± 7.2∗↑	35.9 ± 7.4∗↑	59.8 ± 5.6∗↑	65.7 ± 4.8∗↑	39.5 ± 13.0∗↑	50.8 ± 7.6∗↑	69.0 ± 3.1∗↑	49.1 ± 5.5∗↑	69.9 ± 2.8∗↑
	$LBP_{24,3}$	69.3 ± 3.6∗	29.4 ± 9.8∗↑	39.3 ± 4.4∗↑	66.5 ± 4.1∗↑	64.9 ± 2.6∗↑	39.0 ± 5.1∗↑	50.7 ± 3.9∗↑	66.7 ± 4.3∗↑	52.2 ± 2.1∗↑	68.9 ± 2.7∗↑
	$CLBP_{24,3}$	79.3 ± 2.9∗	42.8 ± 5.9∗↑	26.5 ± 3.4∗↑	27.8 ± 1.5∗↑	81.3 ± 1.9∗↑	63.8 ± 5.2∗↑	67.5 ± 3.7∗↑	80.3 ± 2.0∗↑	55.7 ± 1.4∗↑	69.4 ± 3.3∗↑
	$LBC_{24,3}$	67.8 ± 3.0∗	22.3 ± 5.0∗↑	34.8 ± 5.8∗↑	66.1 ± 3.0∗↑	63.6 ± 3.1∗↑	37.5 ± 7.6∗↑	46.2 ± 5.0∗↑	65.4 ± 3.5∗↑	49.9 ± 2.0∗↑	66.5 ± 2.0∗↑
	$CLBC_{24,3}$	77.9 ± 2.5∗	34.5 ± 12.1∗↑	27.7 ± 2.8∗↑	62.5 ± 2.1∗↑	79.6 ± 1.8∗↑	65.5 ± 3.7∗↑	65.6 ± 4.3∗↑	78.5 ± 1.3∗↑	57.4 ± 1.4∗↑	71.3 ± 4.2∗
	GP-$criptor^{ri}$	**87.7 ± 1.9**	**57.1 ± 2.7**∗	**47.3 ± 2.2**∗	**87.6 ± 2.4**	83.8 ± 1.3∗	**72.0 ± 3.8**∗	**79.9 ± 1.4**∗	85.4 ± 2.2∗	**72.6 ± 1.1**∗	73.9 ± 1.6∗
OutexTC10	DIF	8.2 ± 0.7∗	6.6 ± 1.5∗↑	11.4 ± 2.1∗↑	7.4 ± 0.7∗↑	8.4 ± 0.7∗↑	7.5 ± 1.6∗↑	7.7 ± 1.7∗↑	9.4 ± 0.7∗↑	9.1 ± 0.8∗↑	7.8 ± 1.2∗↑
	$LBP^{u2}_{8,1}$	37.4 ± 2.3∗	18.9 ± 3.0∗↑	27.6 ± 4.0∗↑	34.3 ± 1.7∗↑	34.4 ± 2.9∗↑	31.6 ± 3.1∗↑	42.1 ± 3.9∗↑	36.1 ± 2.4∗↑	27.4 ± 0.8∗↑	32.7 ± 2.6∗↑
	GLCM	43.8 ± 2.5∗	17.3 ± 4.8∗↑	26.4 ± 6.5∗↑	41.2 ± 6.2∗↑	36.5 ± 3.3∗↑	28.0 ± 4.9∗↑	38.0 ± 3.8∗↑	43.3 ± 3.6∗↑	34.4 ± 3.9∗↑	44.2 ± 3.8∗↑
	$LBP^{riu2}_{24,3}$	67.6 ± 2.0∗	21.9 ± 6.9∗↑	36.4 ± 3.9∗↑	66.4 ± 1.1∗↑	66.0 ± 2.5∗↑	38.8 ± 6.1∗↑	49.8 ± 2.7∗↑	63.6 ± 4.0∗↑	48.2 ± 2.1∗↑	64.1 ± 2.4∗↑
	$CLBP_{24,3}$	85.4 ± 2.2	34.1 ± 7.1∗↑	26.6 ± 4.7∗↑	15.5 ± 2.3∗↑	85.8 ± 2.0 ↓	58.1 ± 10.0∗↑	72.0 ± 6.2∗↑	85.4 ± 2.3	58.2 ± 1.2∗↑	72.2 ± 4.0∗
	$LBC_{24,3}$	63.8 ± 2.0∗	19.6 ± 4.4∗↑	34.7 ± 3.8∗↑	62.6 ± 2.0∗↑	60.9 ± 2.9∗↑	37.6 ± 6.3∗↑	47.0 ± 3.3∗↑	58.9 ± 4.1∗↑	46.1 ± 1.3∗↑	59.0 ± 1.5∗↑
	$CLBC_{24,3}$	86.2 ± 2.1	40.9 ± 6.1∗↑	28.2 ± 4.6∗↑	70.4 ± 2.3∗↑	**86.8 ± 1.8** ↓	63.5 ± 7.1∗↑	74.3 ± 3.5∗↑	**86.4 ± 2.2**	61.1 ± 1.2∗↑	**77.2 ± 5.2**∗↓
	GP-$criptor^{ri}$	**86.8 ± 1.9**	**51.2 ± 3.1**∗	**41.5 ± 2.0**∗	**86.2 ± 1.6**	83.4 ± 1.5∗	**70.5 ± 4.6**∗	**78.1 ± 1.9**∗	85.6 ± 2.1	**68.6 ± 1.7**∗	72.2 ± 1.1∗

Table 5.6: Accuracy (%) of ten classifiers using eight image descriptors on the **Faces** image dataset ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
Faces	DIF	64.4 ± 10.6	60.7 ± 11.9∗↓	54.6 ± 9.60∗	58.9 ± 5.5∗↑	62.5 ± 13.2	65.9 ± 9.40	56.6 ± 10.2∗↑	60.9 ± 11.2∗	58.0 ± 7.4∗↑	62.9 ± 11.5
	$LBP^{u2}_{8,1}$	69.4 ± 5.90	58.1 ± 5.80∗↓	**65.3 ± 14.4**	**69.4 ± 6.5**	73.4 ± 9.50	69.0 ± 5.30	**76.3 ± 5.10** –↓	**73.0 ± 6.20**	**70.0 ± 4.1** ↓	71.2 ± 8.10
	GLCM	**73.6 ± 10.6**	**65.8 ± 16.0** ↓	62.3 ± 12.9	67.1 ± 8.2	**74.0 ± 13.1**	**73.4 ± 10.3**	64.4 ± 7.80	69.6 ± 13.9	64.8 ± 6.4∗	**75.1 ± 11.0**
	$LBP^{riu2}_{8,1}$	59.3 ± 10.9∗	60.4 ± 10.8∗↓	58.2 ± 10.3∗	60.4 ± 8.7∗	60.2 ± 6.50∗↑	59.6 ± 5.70∗↑	61.3 ± 7.00∗↑	61.10 ± 7.9∗↑	58.3 ± 6.6∗↑	60.1 ± 6.60∗↑
	$CLBP_{8,1}$	61.3 ± 9.90∗	51.3 ± 5.30∗	54.7 ± 6.40∗↑	66.1 ± 8.1	64.9 ± 10.6	59.0 ± 6.00∗↑	65.9 ± 11.7	62.1 ± 8.70∗	57.6 ± 4.5∗↑	64.1 ± 9.50
	$LBC_{8,1}$	59.1 ± 9.70∗	60.1 ± 11.0∗↓	54.4 ± 10.3∗	55.8 ± 9.0∗↑	56.4 ± 6.70∗↑	53.6 ± 6.10∗↑	57.6 ± 7.90∗↑	54.6 ± 6.20∗↑	56.0 ± 5.9∗↑	55.6 ± 6.70∗↑
	$CLBC_{8,1}$	63.5 ± 9.80∗	52.2 ± 5.90∗	54.5 ± 6.70∗↑	69.0 ± 9.9	65.5 ± 14.5	61.0 ± 6.00∗↑	70.5 ± 11.7	64.1 ± 12.4	56.7 ± 5.8∗↑	67.2 ± 14.0
	GP-criptor^{ri}	71.1 ± 5.00	45.6 ± 8.30∗	60.2 ± 3.80∗	66.1 ± 5.6∗	68.4 ± 5.00	69.5 ± 7.50	67.5 ± 5.30	68.3 ± 5.10	65.6 ± 5.1∗	68.0 ± 4.80

Table 5.7: Accuracy (%) of ten classifiers using eight image descriptors on the **Coins** image dataset ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
Coins	DIF	**94.7 ± 3.20**	**80.9 ± 8.00** ↓	**75.4 ± 10.3** ↓	**88.1 ± 6.90** ↓	**94.4 ± 3.30** ↓	**87.8 ± 10.5** ↓	**94.9 ± 1.30** ↓	**94.9 ± 2.80** ↓	**80.5 ± 6.40** ↓	**94.9 ± 2.10** ↓
	$LBP^{u2}_{8,1}$	75.3 ± 12.4	64.4 ± 17.5∗↓	73.5 ± 17.5	67.0 ± 10.4∗↑	70.5 ± 14.3	64.8 ± 10.8∗	74.3 ± 15.5	68.1 ± 13.1∗	71.3 ± 10.5	71.1 ± 12.8
	GLCM	80.5 ± 8.10	54.7 ± 10.9∗↓	65.5 ± 10.6∗	77.2 ± 8.40	79.5 ± 8.90	69.3 ± 10.7∗	75.1 ± 9.00	77.2 ± 9.10	71.6 ± 6.90	74.4 ± 14.3
	$LBP^{riu2}_{8,1}$	68.9 ± 7.20∗	62.8 ± 9.50∗↓	61.2 ± 14.3∗	67.9 ± 8.80∗↑	70.8 ± 11.3	70.6 ± 9.40	70.6 ± 9.90	74.8 ± 6.90	64.7 ± 6.30∗	71.3 ± 10.3
	$CLBP_{8,1}$	65.9 ± 11.0∗	51.1 ± 8.20∗↓	50.2 ± 7.40∗↑	66.9 ± 7.80∗↑	68.2 ± 8.70∗	66.5 ± 6.90∗	72.3 ± 4.80	66.3 ± 9.30∗	57.5 ± 3.30∗↑	73.4 ± 9.60
	$LBC_{8,1}$	66.0 ± 7.40∗	61.5 ± 9.00∗↓	56.9 ± 10.6∗	64.1 ± 7.00∗↑	68.9 ± 8.70∗	65.4 ± 8.70∗	68.7 ± 7.70∗	70.6 ± 7.40∗	60.9 ± 6.70∗↑	70.8 ± 8.30
	$CLBC_{8,1}$	66.8 ± 6.00∗	50.4 ± 7.20∗↓	49.2 ± 7.30∗↑	69.2 ± 8.10∗↑	65.5 ± 6.60∗	63.6 ± 6.40∗↑	66.7 ± 5.90∗	62.8 ± 5.10∗↑	56.8 ± 2.40∗↑	71.0 ± 7.20
	GP-criptor^{ri}	81.9 ± 15.1	41.6 ± 5.40∗	61.4 ± 5.60∗	77.1 ± 10.9	72.5 ± 10.8	73.0 ± 8.20	75.4 ± 13.2	73.2 ± 10.9	69.2 ± 9.50∗	71.0 ± 10.3

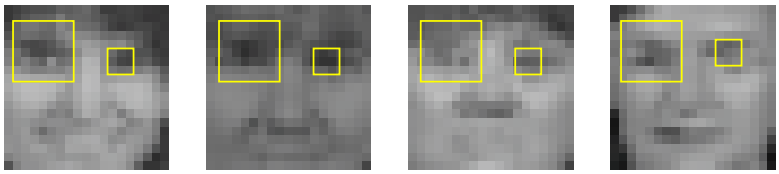
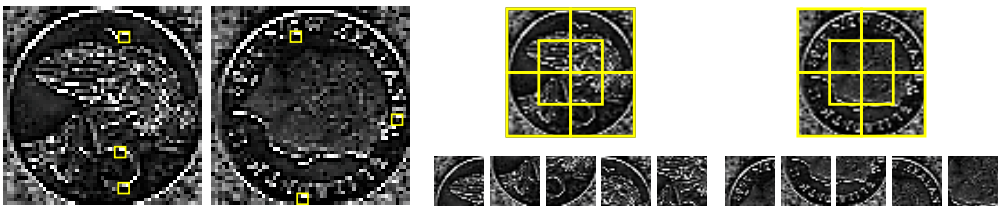


Figure 5.4: Examples demonstrate the difference between using 3×3 and 7×7 windows to highlight the eye regions on an example from the Faces dataset.

5.4.3.2 Coins

The results of the second non-texture dataset (i.e. Coins) are presented in Table 5.7, where a window of size 7×7 pixels and code of length 9-bits are used. Surprisingly, all state-of-the-art methods as well as the proposed method have struggled to perform well on this dataset. Meanwhile, the simple DIF method shows extremely good performance that significantly outperforms all other methods. Moreover, all classifiers show best performance when DIF (domain-independent features [337]) features are used. The proposed method, in general, shows a comparable performance to other image descriptors, and has significantly outperformed some of them. The poor performance of all LBP and LBP-like methods (including the proposed method), and the good performance of DIF on this dataset were expected. This is because the LBP-based methods use a relatively small window that captures small regions, which are very likely to appear in the instances of both classes, as presented in Figure 5.5(a). This increases the difficulty of finding distinctive keypoints to discriminate the head and tail instances. DIF, on the other hand, divides the instances into 5 big regions (excluding the horizontal and vertical lines), i.e., 27×27 pixels each, which allows more information (especially the central region of the instances) to be captured as presented in Figure 5.5(b). Therefore, two more experiments have been conducted on this dataset using a 9×9 pixel window in the first and a window of size 11×11 pixels in the second. The results of these experiments along with the previous window sizes (3×3, 5×5, and 7×7) are presented in Table 5.8. Increasing the window size has allowed the system to capture more informative features, which is clearly reflected by the improvement in the performance. Comparing results for the 3×3 and 11×11 windows, the minimum improvement is 6.7% (J48), whilst in the case of AdaBoost the difference



(a) (b)

Figure 5.5: Examples of the Coins dataset show (a) some 3×3 windows that have identical mean and standard deviation, and (b) the regions of the DIF method (excluding the horizontal and vertical line features).

Table 5.8: Accuracy (%) of using GP-criptor$^{\text{ri}}$ with different window sizes on the **Coins** dataset ($\bar{x} \pm s$)

	3×3	5×5	7×7	9×9	11×11
1-NN	71.8 ± 13.3	76.7 ± 14.4	81.9 ± 15.1	85.7 ± 15.1	87.9 ± 14.6
AdaBoost	39.9 ± 6.2	35.4 ± 7.2	41.6 ± 5.4	62.5 ± 7.2	63.3 ± 6.2
J48	57.5 ± 7.7	58.7 ± 6.1	61.4 ± 5.6	63.5 ± 6.1	64.3 ± 6.6
KStar	69.3 ± 13.1	72.8 ± 12.5	77.1 ± 10.9	81.3 ± 12.4	83.5 ± 12.2
MLP	66.0 ± 10.7	69.1 ± 10.7	72.5 ± 10.8	73.4 ± 11.3	75.4 ± 11.6
NB	64.9 ± 9.6	68.0 ± 10.1	73.0 ± 8.2	75.7 ± 9.2	79.4 ± 9.5
NBTree	68.0 ± 12.3	71.1 ± 13.1	75.4 ± 13.2	79.2 ± 13.6	82.2 ± 12.7
NNge	66.1 ± 10.7	70.7 ± 10.2	73.2 ± 10.9	74.6 ± 11.6	76.8 ± 12.0
RF	63.7 ± 10.2	65.9 ± 9.9	69.2 ± 9.5	72.2 ± 10.3	75.8 ± 10.5
SVM	64.5 ± 9.9	67.6 ± 10.3	71.0 ± 10.3	71.4 ± 9.8	73.5 ± 11.0

is 23.4%. Comparing those newly obtained results (11×11 window) to the results of the other methods in Table 5.7 reveals that GP-criptor$^{\text{ri}}$, apart from J48, has either second or third best performance. Furthermore, GP-criptor$^{\text{ri}}$ with 1-NN has significantly outperformed the vast majority of the competitive methods with more powerful classifiers.

5.4.4 Summary

From the results above, the following observations can be deduced.

- The proposed method has the ability to automatically evolve an image descriptor using the raw pixel values and without human intervention;

- The system uses only two instances of each class and yet it has been shown to outperform most domain-expert designed descriptors;
- The evolved descriptors do not solely detect a specific and predetermined set of keypoints, e.g., lines, corners, and spots; rather, it automatically detects a set of good keypoints;
- The proposed method is not limited to a specific domain, i.e., texture classification; it has also showed better or comparable performance to state-of-the-art methods on other domains such as face classification and coin classification tasks (further investigation can still be made);
- The features of the GP-criptor^{ri} method have, in the majority of the cases, positive influence on the performance of classifiers of different types;
- Unlike domain-expert designed descriptors, the proposed method evolves a rotation-invariant descriptor that does not require human intervention to handle this issue;
- It is easy to change the parameters, e.g., the window size and the number of bits in the code, in the proposed method where the system can automatically handle those changes without human intervention, while domain-expert involvement is required to alter these parameters in other methods; and
- The program structure of the proposed method is flexible and could allow different types of functions to be used for feature extraction.

5.5 Further Analysis

In this section, a comparison of the robustness of GP-criptor (Chapter 4) and GP-criptor^{ri} to evolve rotation-invariant image descriptors is discussed first. Then an individual program evolved by GP-criptor^{ri} is analysed and discussed in order to provide more understanding on why and how the proposed method can perform well.

5.5.1 Comparison between GP-criptor and GP-criptor^{ri}

To support our hypothesis of the inability of GP-criptor to handle the rotation variance (see Section 4.2 on page 119), we have experimented with this method using the same six texture image datasets discussed in Section 5.3.1. The results of the non-rotated and rotated versions for each dataset (Brodatz, Kylberg and OutexTC) are grouped in a single table in this section. A **bold face** font is used in those tables to present the results of statistical testing (significantly better). Wilcoxon signed-rank statistical test with a significance level of 5% is also used in this experiment.

The results of using the 10 previously used classification methods (Section 5.4) with GP-criptor and GP-criptor^{ri} automatically evolved descriptors on the BrNoRo and BrWiRo datasets are presented in Table 5.9. Apart from AdaBoost and J48, all other classifiers have achieved significantly better performance on BrNoRo (rotation-free) using the descriptors evolved by the GP-criptor method than those evolved by GP-criptor^{ri}. However, AdaBoost and J48 show, respectively, 12% and 11.5% improvement using GP-criptor^{ri} descriptors. More importantly the GP-criptor^{ri} method shows significantly positive influence on the performances of *all* the 10 classifiers on BrWiRo (with rotation) compared to the GP-criptor method, where the improvement is ranging on average between 37% (NB) and 23% (J48).

Table 5.10 presents the results of these two methods' (GP-criptor and GP-criptor^{ri}) evolved descriptors for the 10 classification methods on the KyNoRo and KyWiRo datasets. Although the classifiers showed better performance on the non-rotated version of the Kylberg dataset (KyNoRo) using the features extracted by the GP-criptor method descriptors, GP-criptor^{ri} has achieved comparable performance and the gap is only 3.7% in the worst case (MLP). Noticeably, GP-criptor^{ri} has improved the performances of AdaBoost and J48 by 12.8% and 7.2%, respectively. The results on the rotated version of Kylberg (KyWiRo) show the ability of the GP-criptor^{ri} method to handle the rotation variance where the GP-criptor method has struggled to preserve the same level of performance on the rotation-free version of this dataset.

Table 5.9: Accuracy (%) of 10 classifiers using GP-criptor and GP-criptor^{ri} evolved image descriptors on the **BrNoRo** and **BrWiRo** datasets ($\bar{x} \pm s$).

	BrNoRo		BrWiRo	
	GP-criptor	GP-criptor^{ri}	GP-criptor	GP-criptor^{ri}
1-NN	**96.3 ± 0.8**	90.9 ± 1.9	60.3 ± 1.7	**92.5 ± 1.1**
AdaBoost	48.9 ± 2.7	**60.9 ± 3.1**	26.7 ± 0.8	**59.7 ± 3.4**
J48	39.1 ± 1.0	**50.6 ± 1.2**	25.9 ± 1.3	**48.9 ± 0.9**
K*	**95.1 ± 0.4**	85.4 ± 1.4	51.8 ± 1.9	**86.7 ± 1.5**
MLP	**94.4 ± 0.9**	82.4 ± 1.9	52.6 ± 2.1	**83.7 ± 1.4**
NB	**89.3 ± 1.1**	79.9 ± 2.6	44.1 ± 3.7	**81.1 ± 3.0**
NBTree	**84.5 ± 1.0**	82.2 ± 1.4	50.4 ± 2.0	**83.2 ± 0.9**
NNge	**95.1 ± 0.7**	85.7 ± 1.5	56.3 ± 2.4	**86.5 ± 2.0**
RF	**71.1 ± 2.0**	69.9 ± 1.7	39.9 ± 0.9	**69.6 ± 1.2**
SVM	**81.0 ± 1.2**	70.5 ± 2.3	42.9 ± 2.1	**71.3 ± 1.8**

Finally, the results presented in Table 5.11 are obtained on the OutexTC00 and OutexTC10 datasets. The pattern of the results on these two datasets is very similar to that previously observed on the Kylberg and Brodatz datasets. The GP-criptor method shows significantly better performance than that achieved by the GP-criptor^{ri} method on the rotation-free version of OutexTC (OutexTC00), whereas the GP-criptor^{ri} method shows significantly better performance on the rotated version, revealing the robustness of this method to evolve rotation-invariant image descriptor. The differences between the performances of the two methods are ranging between 22.3% (NB) and 12.1% (J48).

In summary, the following observations can be made:

- Although the GP-criptor method has better performance on the rotation-free datasets, the GP-criptor^{ri} method has achieved comparable performance to GP-criptor and still outperformed the other image descriptors studied in Section 5.4;
- The GP-criptor^{ri} method has the potential to handle the rotation variance, whilst the GP-criptor method struggled to preserve a satisfactory level of performance when the dataset has rotated instances;
- The proposed method achieved more stable or consistent results between the rotated and rotation-free datasets, compared to those results achieved by

Table 5.10: Accuracy (%) of 10 classifiers using GP-criptor and GP-criptor^{ri} evolved image descriptors on the **KyNoRo** and **KyWiRo** datasets ($\bar{x} \pm s$).

	KyNoRo		KyWiRo	
	GP-criptor	GP-criptor^{ri}	GP-criptor	GP-criptor^{ri}
1-NN	**89.5 ± 1.3**	86.7 ± 1.8	56.3 ± 2.4	**88.5 ± 1.4**
AdaBoost	44.0 ± 1.2	**56.8 ± 2.5**	23.5 ± 1.1	**56.2 ± 1.4**
J48	33.9 ± 1.1	**41.1 ± 1.3**	25.1 ± 1.6	**41.1 ± 1.7**
K*	**85.9 ± 1.5**	82.7 ± 1.8	50.5 ± 2.0	**84.4 ± 1.8**
MLP	**82.5 ± 2.0**	78.8 ± 1.7	43.1 ± 1.5	**80.5 ± 1.2**
NB	72.9 ± 5.4	70.7 ± 4.2	38.4 ± 2.2	**72.7 ± 3.8**
NBTree	75.8 ± 1.7	75.1 ± 1.6	46.7 ± 1.7	**76.3 ± 1.6**
NNge	**85.4 ± 1.7**	82.6 ± 2.1	49.4 ± 2.1	**84.0 ± 1.4**
RF	64.5 ± 1.4	64.4 ± 1.2	38.7 ± 1.1	**65.3 ± 1.1**
SVM	**70.5 ± 2.0**	66.9 ± 1.7	37.0 ± 1.4	**68.3 ± 1.8**

the GP-criptor method as the drop in performance was significant on the rotated images; and

- The proposed method has a noticeable positive influence on improving the performances of AdaBoost and J48 classification methods.

5.5.2 Analysis of a GP-criptor^{ri} evolved descriptor

Here, one of the good performing and relatively small programs that has been evolved on the BrWiRo dataset is chosen. The tree representation of the program is depicted in Figure 5.6. This program uses a 7-bit code length and a sliding window of size 5×5 pixels. Overall, there are 106 nodes in this program with 56 terminals and 50 functions. Hence, this program performs on average 7 operations $((50 - 1\ (\text{code}))\ /7 = 7)$ to calculate the value of each bit. Most of these tree branches can be interpreted easily such as the second $(min + 2\,(mean - max))$, third $(mean^2 - ((min \times max)))$, fourth $((2 \times mean) - max)$, and the simplified sixth $(min^2 - stdev)$ bit branches; whereas other branches are more complicated. In order to present how this program responds to different patterns, two window samples are fed into the program as depicted in Figure 5.7. In the first case (Figure 5.7(a)), the window comprises the same values which represents an example of a homogeneous region; meanwhile, in the second case (Figure 5.7(b)), the window

Table 5.11: Accuracy (%) of 10 classifiers using GP-criptor and GP-criptor^{ri} evolved image descriptors on the **OutexTC00** and **OutexTC10** datasets ($\bar{x} \pm s$).

	OutexTC00		OutexTC10	
	GP-criptor	GP-criptor^{ri}	GP-criptor	GP-criptor^{ri}
1-NN	**95.9 ± 2.0**	87.7 ± 1.9	69.3 ± 2.7	**86.8 ± 1.9**
AdaBoost	55.5 ± 1.2	57.1 ± 2.7	30.5 ± 1.0	**51.2 ± 3.1**
J48	45.9 ± 2.9	47.3 ± 2.2	29.4 ± 2.2	**41.5 ± 2.0**
K*	**93.4 ± 2.4**	87.6 ± 2.4	65.5 ± 2.8	**86.2 ± 1.6**
MLP	**95.8 ± 1.5**	83.8 ± 1.3	66.5 ± 3.1	**83.4 ± 1.5**
NB	**86.7 ± 3.0**	72.0 ± 3.8	48.2 ± 2.9	**70.5 ± 4.6**
NBTree	**85.7 ± 2.0**	79.9 ± 1.4	58.2 ± 2.4	**78.1 ± 1.9**
NNge	**95.4 ± 2.4**	85.4 ± 2.2	66.8 ± 2.9	**85.6 ± 2.1**
RF	**78.4 ± 1.1**	72.6 ± 1.1	47.4 ± 2.4	**68.6 ± 1.7**
SVM	**84.8 ± 1.6**	73.9 ± 1.6	54.9 ± 1.9	**72.2 ± 1.1**

comprises only one value that is smaller than the other values which represents an example of a dark spot region. Clearly, the generated code in the two cases are different that each contributes toward a different bin of the generated histogram. These two examples show that the system has successfully defined those formulae which returns different responses for different patterns, i.e., keypoints, without any prior knowledge or human intervention.

In terms of accuracy, this program has achieved 94.6% accuracy on the unseen data; the confusion matrix on BrWiRo (where the row indices represent the actual classes, while column indices the predicted classes) is presented in Table 5.12. The program has successfully classified all instances (100%) of 6 out of the 20 classes, over 90% accuracy on the other 11 classes, and only 3 classes are below 80%.

The average fitness value at each generation of the 30 runs, using the same training set that was used to evolve the program in Figure 5.6, is presented in Figure 5.8. Clearly, the system has made fast jumps in the fitness values over the first 16 generations as the fitness value has decreased from approximately 0.2 to approximately 0.089; while the progress after that was slower and the fitness value dropped to approximately 0.067 over the remaining 34 generations.

In order to shed light on the time required to evolve a descriptor by GP-criptor^{ri}, the CPU time for each evolutionary run has been measured from the beginning of generating the initial population to the end when a stopping criterion is met.

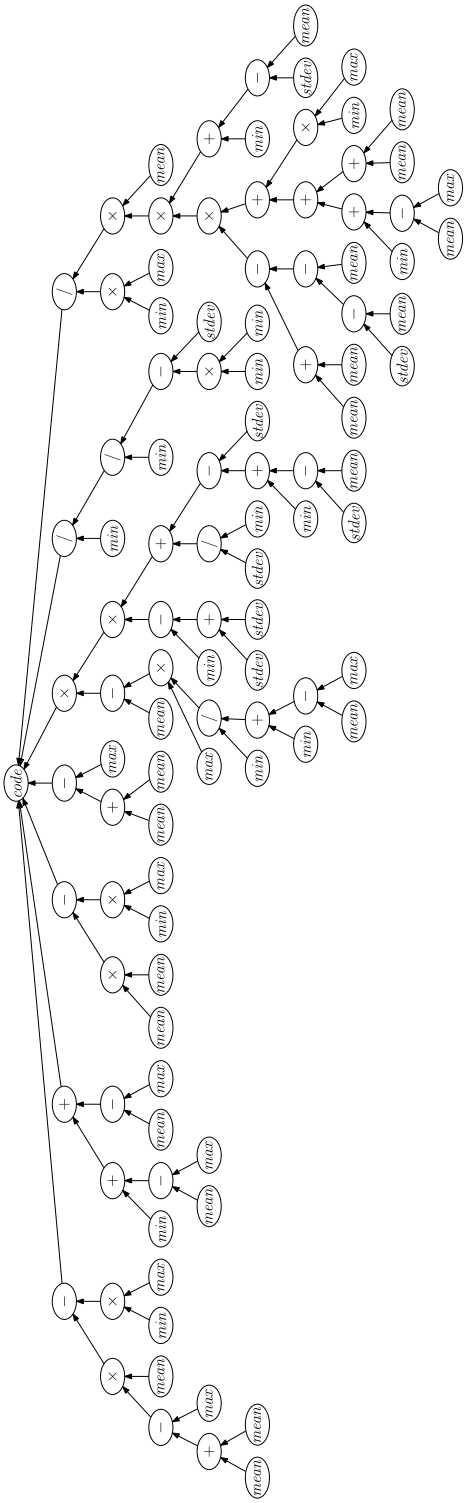


Figure 5.6: The tree representation of a program evolved on the BrWiRo dataset.

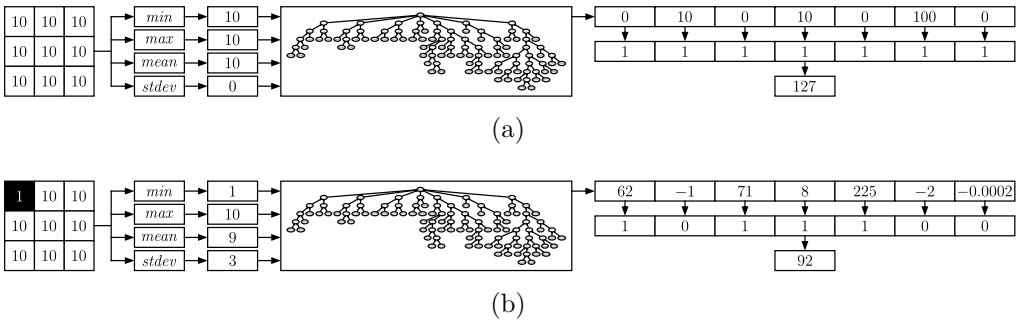


Figure 5.7: Two examples demonstrate how the program presented in Figure 5.6 generates codes for a (a) homogeneous region, and (b) dark spot region.

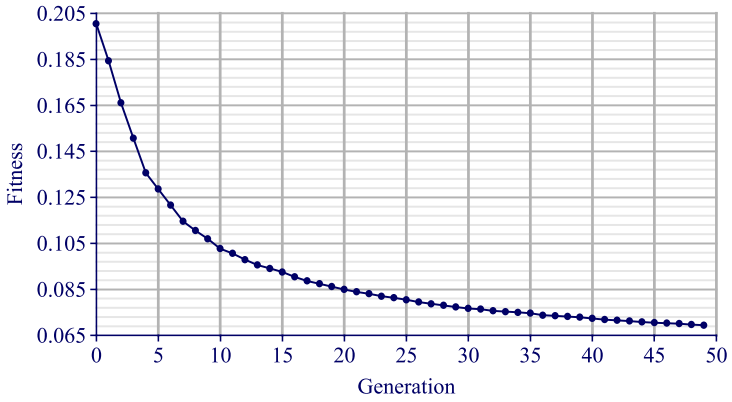


Figure 5.8: The average fitness value per generation.

Figure 5.9 presents the average time required to evolve an image descriptor by GP-criptor^{ri} for different window sizes and code lengths. Unlike GP-criptor (see Section 4.5.2 on page 139), the larger the window size, the longer time that is needed to obtain a good descriptor. This is mainly because in GP-criptor the system uses the pixel values of the sliding window directly, whilst GP-criptor^{ri} iterates over those values in order to calculate the mean and standard deviation. Therefore, larger windows require longer loops, i.e., more values to iterate on, at each position of the sliding window.

A stacked representation of the resulting feature vectors by this program for 40 instances, 2 from each class, that were randomly drawn from the BrWiRo dataset is presented in Figure 5.10. The class labels for those instances are printed on the

Table 5.12: The confusion matrix on BrWiRo for the program presented in Figure 5.6

	D01	D03	D04	D05	D06	D09	D11	D14	D15	D16	D17	D18	D20	D21	D24	D34	D37	D46	D47	D49	Total %
D01	504	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
D03	0	466	0	0	0	0	0	0	14	0	0	0	0	0	24	0	0	0	0	0	92.5
D04	0	0	389	0	0	63	52	0	0	0	0	0	0	0	0	0	0	0	0	0	77.2
D05	0	0	1	493	0	0	0	0	5	0	0	5	0	0	0	0	0	0	0	0	97.8
D06	0	0	0	0	504	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
D09	0	0	87	0	0	385	3	0	7	0	1	0	0	0	21	0	0	0	0	0	76.4
D11	0	0	18	0	0	4	482	0	0	0	0	0	0	0	0	0	0	0	0	0	95.6
D14	0	0	0	0	0	0	0	501	1	0	0	0	0	0	2	0	0	0	0	0	99.4
D15	0	21	10	2	0	6	0	0	451	0	0	0	0	0	14	0	0	0	0	0	89.5
D16	0	0	0	0	0	0	2	0	0	489	13	0	0	0	0	0	0	0	0	0	97.0
D17	0	0	2	0	0	0	0	0	0	0	502	0	0	0	0	0	0	0	0	0	99.6
D18	0	0	0	109	0	0	0	0	0	0	0	395	0	0	0	0	0	0	0	0	78.4
D20	0	0	0	0	0	0	0	0	0	0	0	0	504	0	0	0	0	0	0	0	100.0
D21	0	0	0	0	0	0	0	0	0	0	5	0	0	499	0	0	0	0	0	0	99.0
D24	0	3	1	0	0	2	0	0	0	0	0	0	0	0	498	0	0	0	0	0	98.8
D34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	504	0	0	0	0	100.0
D37	2	0	2	0	0	0	1	0	0	0	0	20	0	0	0	0	478	0	1	0	94.8
D46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	504	0	0	100.0
D47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	504	0	100.0
D49	0	12	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	488	96.8

horizontal axis, whereas the vertical axis shows the relative frequency. The aim of this figure is to show that a program evolved by GP-criptor^{ri} is responding in a similar way to those instances belonging to the same class, and differently to those instances belonging to other classes. Each feature vector comprises 128 values (2^7), each of which has been represented with a different colour in the bars of Figure 5.10. Hence, each colour indicates the same exact feature across those bars. A closer inspection of this figure reveals how those feature vectors belonging to the same class are similar and have, to some extent, a distinctive fingerprint. Some typical examples are D09, D17, D24, and D49 as they are visually easier to compare than others. The figure also shows two important facts: firstly, the system has detected some keypoints that appeared in one class but not, or very seldom, in the other classes; and secondly, the system is able to find keypoints that are shared between all those classes but appear more frequently in one class than in the other classes. An example of the former case is the feature indicated in dark-blue at the middle part of class D34, whereas the features indicated by light-blue and light-green of, respectively, classes D21 and D49 are examples of the latter case.

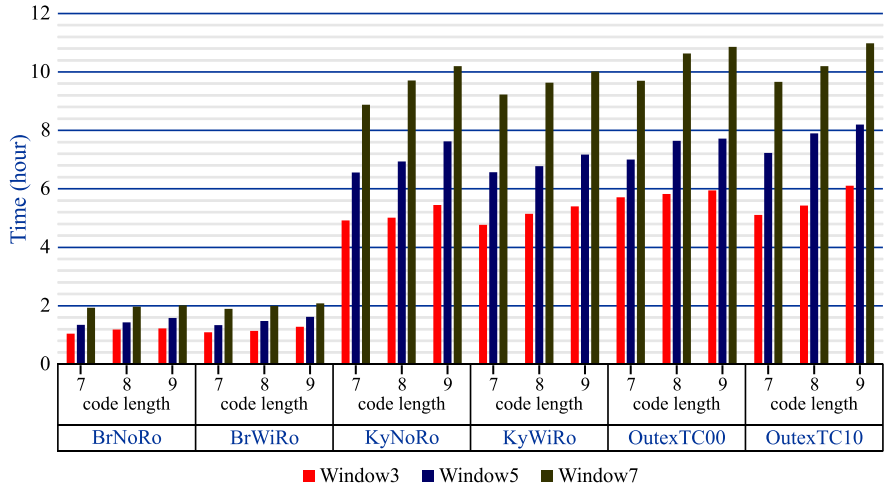


Figure 5.9: The average time in hours required to evolve a descriptor.

Further to our earlier discussion on the number of training instances (Section 5.1), and from the example studied in this section, it becomes evident that the system did not stop the evolutionary process when only one or a few distinctive keypoints have been detected. Instead, the proposed method continued the process of evolving a better program that can capture more prominent or good keypoints.

The main idea of using only a few instances matches reasonably well the process of how humans teach young children to identify different objects. There are three amazing facts in this learning process: (1) humans do not need to use thousands or hundreds of images where only one or a few are enough [256]; (2) different children may detect or use different characteristics to identify the object of each category, e.g., some may focus on the body shape while others may focus on the head/face characteristics in the case of discriminating between cow and horse examples; and (3) humans do not specify keypoints for objects of different categories and teach the children to use them to perform the categorisation task, instead, the children identify those keypoints themselves.

Figure 5.10 shows that the evolved program (presented in Figure 5.6) captured and extracted similar patterns for the two instances of each class at the end of the evolutionary process, but the patterns for different classes are distinguished. This shows that this method is very good at distinguishing examples in different classes.

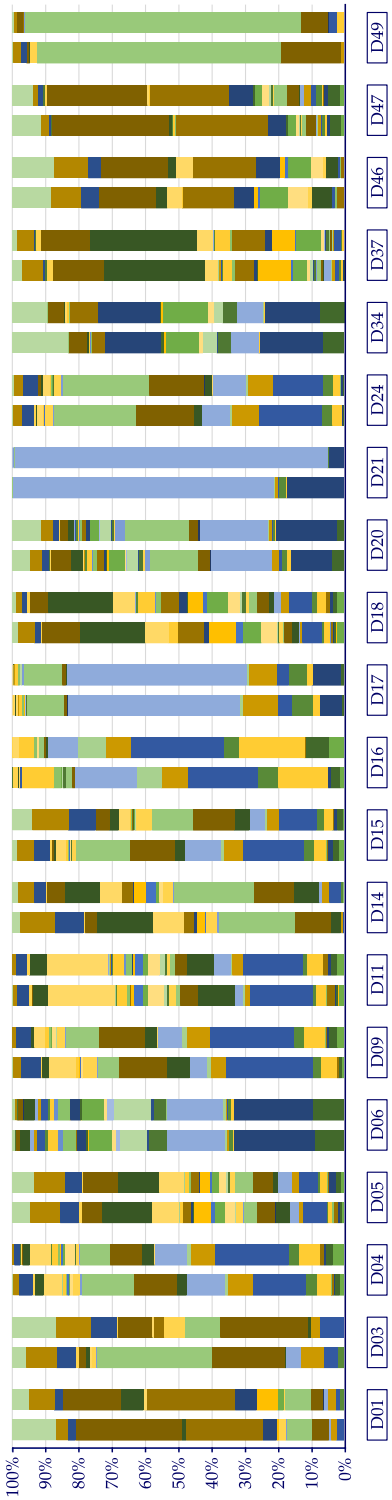


Figure 5.10: A stacked representation of the resulting feature vectors by the program shown in Figure 5.6 for 40 instances (2 from each class) drawn from the BrWiRo. The same features are indicated using the same colours to highlight the similarities/differences between instances of the same/different class(es).

5.6 Chapter Summary

In this chapter, a GP approach was proposed to automatically evolve a rotation-invariant image descriptor to detect keypoints and extract informative features simultaneously for image classification. Different from existing methods, the proposed GP approach does not require any human intervention, needs only two instances per class, and aims to tackle rotation (in)variance by using simple rotation-invariant features in the terminal set. This method is suitable for problems where only a small number of labelled instances are available, and for the situations that cannot afford a long time for training. To examine the performance of the proposed GP approach, a large number of experiments have been conducted on six texture and two non-texture image classification datasets of varying difficulty, with different degrees of rotations. The performance of the GP approach is compared with seven state-of-the-art domain-expert designed image descriptors and ten well-known classifiers are used in the experiments. The results show that the proposed GP method, using only two instances per class, performed comparably or significantly outperformed the other methods in most cases. Furthermore, the GP approach is robust in handling different degrees of rotations, and the evolved GP tree, i.e., image descriptor, is understandable and interpretable by humans, although it is automatically constructed without human intervention.

Extending GP-criptor^{ri} to automatically specify the length of the code, i.e., number of children under the *code* node, is discussed in the next chapter.

6

A Dynamic GP Representation for Evolving Image Descriptors

6.1 Introduction

The process of developing an image classifier requires an appropriate set of features and a classification method. Finding appropriate features may be more important than designing an effective classification algorithm; in many cases, it requires a domain-expert [341]. If the extracted features are informative/good, even a very simple classification model, e.g., k-Nearest Neighbour, can be sufficient to achieve good classification performance [229]. In order to construct a feature vector for an image, a set of *keypoints*, i.e., regions of interest, need to be identified first. The number of features to be extracted from each keypoint specifies the length of the feature vector. The majority of machine learning algorithms were not designed to handle feature vectors of varying length of one problem, i.e., performing classification where each instance has a different number of features. Therefore, the length of the feature vector extracted from an image is often predetermined

and static. For example, conventional LBP produces a feature vector with length 2^p where p is the number of neighbouring pixels; whereas uniform LBP (LBP^{u2}) [218] generates a feature vector with length $p(p-1)+3$. Specifying the length of the feature vector increases the number of the parameters required to be set. This task can be accomplished empirically; however, computationally it can be a very expensive task to perform. Automatically determining the length of the feature vector during the learning phase represents an alternative solution that can largely reduce the computations and experiments.

6.1.1 Chapter Goals

This chapter aims at using GP to automate the process of constructing a rotation-invariant image descriptor that detects a set of automatically designed keypoints, i.e., the user does not specify those keypoints (such as corners and edges), and extracts informative features from those keypoints simultaneously, and the system automatically determines the length of the feature vector. Motivated by the success of GP-criptor$^{\text{ri}}$ (Chapter 5), a set of simple arithmetic operators and first-order statistics (e.g. mean and standard deviation) are automatically synthesised as a set of formulae that form an image descriptor, i.e., an evolved GP program. More importantly, this method does not require human intervention to design the keypoints and features; instead it uses a small sample (only two instances) of each class to evolve a descriptor. Moreover, the length of the feature vector is dynamic and will be determined during the evolutionary process. Quantitatively, the performance of an image descriptor automatically constructed by GP will be compared to that of seven domain-expert designed descriptors using ten commonly used machine learning classification methods on seven image benchmarks for multi-class texture classification. Those datasets are of varying difficulty, and comprised of a different number of classes and rotations. Qualitatively, on the other hand, a constructed descriptor will be closely examined to shed light on how the proposed method can perform well. The following objectives will be investigated in this chapter.

- Develop a new tree-based [243] GP program representation that allows a node to have a dynamic number of children.

- Assess the evolved descriptors quantitatively and compare to seven domain-expert designed descriptors on seven texture image benchmarks.
- Provide a qualitative assessment by investigating the interpretability and other aspects of an evolved descriptor.

6.1.2 Chapter Organisation

The remainder of the chapter is organised as follows. Section 6.2 describes the proposed method. Section 6.3 presents the experiment design. The results are presented and discussed in Section 6.4. A descriptor evolved by the proposed method is thoroughly examined in Section 6.5. Section 6.6 summarises this chapter.

6.2 The Proposed Method

The proposed *Rotation-invariant Evolutionary Image Descriptor* (EID^{ri}) method is described in this section. As the method is extended from previous methods (Chapter 5 and Chapter 4), the shared components will not be discussed and only the main differences are highlighted. The overall algorithm is discussed first. Then the program representation is explained. Finally, this section describes the fitness measure and the procedure of extracting the feature vector from an image.

6.2.1 The Overall Algorithm

The overall process is depicted in Figure 6.1. The instances of each class are equally split between the training and test sets. A key factor of the proposed method is that only a few instances of each class are required to evolve a descriptor. Therefore, the system randomly selects a subset (two instances per class) of the training set and feeds it into GP. The final result of the GP evolutionary process is an image descriptor that takes an image and produces a feature vector. The randomly selected training instances, i.e., those used during the evolutionary process, and the test set are then fed into the evolved descriptor to generate the transformed training set ($\mathbf{S_{tr}}$) and transformed test set ($\mathbf{S_{ts}}$) respectively. The $\mathbf{S_{tr}}$ is used to train a classifier which is then evaluated using $\mathbf{S_{ts}}$. Here $\alpha = \{(\vec{x}_i, c_i)\}$ $\alpha \in \mathbf{S_{tr}} \cup \mathbf{S_{ts}}$ and

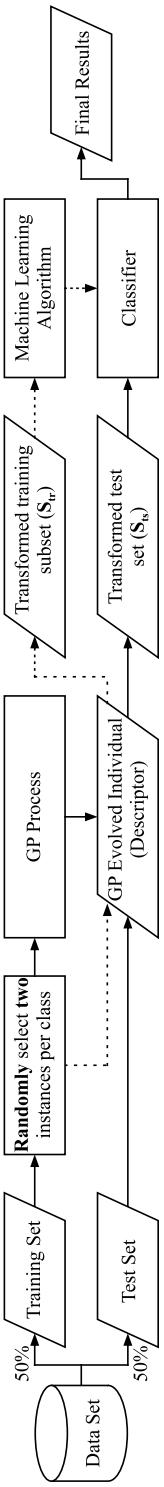


Figure 6.1: Parts of the overall EID^{ri} algorithm.

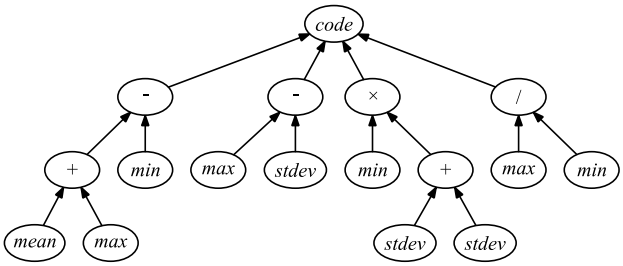
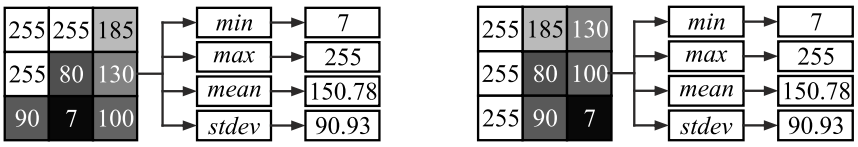


Figure 6.2: Example of an evolved EID$^{\text{ri}}$ program.



(a) (b)

Figure 6.3: Example demonstrates the rotation-invariant property of the terminal nodes, (a) the original window, and (b) a 45° rotated version of the same window.

$i \in \{1, 2, \ldots, z\}$, where $\vec{x}_i \in \mathbb{R}_{\geq 0} = \{l \in \mathbb{R} \mid l \geq 0\}$ and $c_i \in \{t_1, t_2, \ldots, t_C\}$ denote the i^{th} instance's feature vector and corresponding class label. The total number of classes is C, and the total number of instances in α is z.

6.2.2 Program Representation

Similar to GP-criptor$^{\text{ri}}$, the terminal set in EID$^{\text{ri}}$ consists of the *min*, *max*, *mean* and *stdev* nodes as presented in Figure 6.2. Each of these nodes performs a simple first-order statistic on a set of values. The *min* and *max* nodes, respectively, return the minimum, i.e., $\min(\cdot)$, and maximum, i.e., $\max(\cdot)$, value of a vector. The *mean* and *stdev* nodes, on the other hand, calculate and return the average and standard deviation using, respectively, Equation (3.5) and Equation (3.6) (see page 79). These functions are order-independent, i.e., do not consider the indices of the elements in the vector, which is an important property to tackle the rotation variants as demonstrated in Figure 6.3.

In order to keep the search space small, the function set in EID$^{\text{ri}}$ consists of five functions. Four of these functions are the arithmetic $+$, $-$, $\times$ and protected

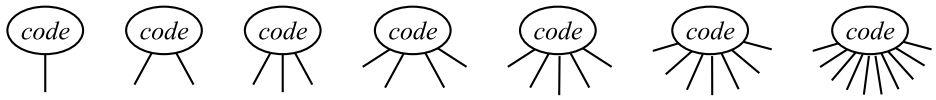


Figure 6.4: Examples of *code* nodes with a different number of children.

/ operators. The / function will return 0 if the denominator is 0, which is very important to prevent the occurrence of a "division by zero" exception. These functions are identical to that of GP-criptor^{ri} where each performs the corresponding operator on two inputs and returns the resulting value.

The main difference between GP-criptor^{ri} and EID^{ri} is the *code* function, which is the fifth component of the function set. This function is different from the other four functions in three ways. First, the number of inputs, i.e., children, is dynamic and is automatically determined by GP during the evolutionary process. However, the number of children must be greater than 0; otherwise, this node will not return any value. Second, this function thresholds the input values such that all negative values are substituted with 0 and all positive and zero values are substituted with 1; which makes the type of the input and output values different. Therefore, the *code* node can not appear anywhere apart from being the root node of a program tree. Third, each individual can have only a single *code* node due to the input-output type-mismatch. The system generates individuals that have a different number of children under the *code* node. To accomplish this, the system is provided with a list of *code* nodes each of which has a specific number of children, e.g., ranging between 1 and 10, and the system randomly chooses one of these nodes to build an individual as depicted in Figure 6.4.

6.2.3 Fitness Measure

An important objective of this research is to evolve image descriptors using only a few training instances. When the number of training instances of each class is relatively small, e.g., less than 10, the use of accuracy often becomes insufficient, mainly because the system can easily capture features that are good enough to discriminate between the training instances, but not sufficient to classify the unseen data. This is an example of the well-known phenomenon in machine learning

Algorithm 2 Measuring the fitness for an EID$^{\text{ri}}$ individual

function FITNESS($\mathbf{S}, r, ind, c, t$)▷ Training images, radius, individual, number of classes, and number of instances per class
  $z \leftarrow (c \times t)$ ▷ Total number of instances in $\mathbf{S}$
  $\mathcal{A} \leftarrow \emptyset$ ▷ Empty set
  for i **in** $\{1, 2, \ldots, z\}$ **do**
    $\vec{f_i} \leftarrow$ FEATURES($\mathbf{S}_i, r, ind$) ▷ Algorithm 3 on i^{th} image
    $c_i \leftarrow$ CLASS($\mathbf{S}_i$) ▷ Class label of the i^{th} image
    $\mathcal{A} \leftarrow \mathcal{A} \cup \left\{\left(\vec{f_i}, c_i\right)\right\}$ ▷ Concatenate the i^{th} image tuple
  end for
  $\{D_{\text{w}}, D_{\text{b}}\} \leftarrow$ DISTANCES($\mathcal{A}, c, t$) ▷ Algorithm 4
  return $1 \Big/ \left(1 + e^{-5(D_{\text{w}} - D_{\text{b}})}\right)$ ▷ Fitness of ind on $\mathbf{S}$
end function

called *over-fitting* [196]. Therefore, it is necessary to derive an alternative fitness measure that encourages the system towards identifying a good set of representative keypoints. The aim is to detect keypoints that have a different pattern between instances belonging to different classes, and meanwhile, ensure that instances belonging to the same class are following the same pattern. Measuring the distance between the feature vectors is an alternative approach as shown in Chapter 5 (see page 145), which is also used in this chapter as presented in Algorithm 4. Measuring the fitness for an individual evolved by EID$^{\text{ri}}$ is presented in Algorithm 2.

A large number of distance measures have been proposed in the literature. One of the widely used measures is χ^2 which measures the distance between two normalised vectors [48] as presented and discussed in Section 4.2.5 (see page 123).

Inherited from GP-criptor$^{\text{ri}}$, this fitness function considers the *within-class* and *between-class* distances. Considering only the within-class distance, the system may evolve a program that can generate nearly similar feature vectors for instances belonging to different groups. In other words, the system will form only a single group/cluster as the aim is to minimise the distance between the instances. Meanwhile, the system may evolve a program that separates the instances such that each instance can form a cluster if the between-class distance considered alone. Therefore, the aim is to find a trade-off between these two distances by minimising

Algorithm 3 Extracting the feature vector

function FEATURES($\mathbf{S}_i, r, ind$) ▷ Image, radius, and individual
  $q \leftarrow |code.children|$ ▷ Number of the children under *code*
  $\vec{f} \leftarrow (0_1, 0_2, \ldots, 0_{2^q})$ ▷ Set of length 2^q of zeros
  for each pix **in** $image$ **do**
    $\vec{x} \leftarrow$ PIXELS(pix, r) ▷ Neighbouring pixels
    $min \leftarrow$ MIN($\vec{x}$)
    $max \leftarrow$ MAX($\vec{x}$)
    $mean \leftarrow$ MEAN($\vec{x}$)
    $stdev \leftarrow$ STDV($\vec{x}$)
    $\mathbf{L} \leftarrow \{min, max, mean, stdev\}$
    $i \leftarrow$ EVALUATE($ind, \mathbf{L}$) ▷ Evaluate the tree on the inputs
    $\vec{f}_{\{i\}} \leftarrow \vec{f}_{\{i\}} + 1$ ▷ Increment the i^{th} element in $\vec{f}$
  end for
  return $\vec{f}$ ▷ Feature vector
end function

the average distance between instances belonging to the same class and maximising the average distance between instances belonging to different classes as shown in Algorithm 4. In this way, GP will try to form a group/cluster for each class that keeps its instances close to each other, and simultaneously, makes those clusters separated apart as much as possible. The aim is that an unseen instance will be put into a cluster consisting of instances from the same class.

6.2.4 Feature Vector Extraction

The process of extracting the feature vector from an image in EID$^{\text{ri}}$ is identical to that in GP-criptor$^{\text{ri}}$. However, the length of the feature vector in GP-criptor$^{\text{ri}}$ is predetermined by the user; whereas in EID$^{\text{ri}}$ it is automatically determined by the system during the evolutionary process. The number of children (q) of the *code* (root) node is used to initiate an empty histogram, i.e., feature vector, of length 2^q bins. This histogram is populated using a sliding window of a predetermined size (r) that scans the image being evaluated row-wise from the left-top corner to the right-bottom corner. At each pixel, i.e., position of the sliding window,

Algorithm 4 The between-class and within-class distances

function DISTANCES($\bar{\mathbf{S}}, c, t$) ▷ Set of tuples $\{(\vec{x}_i, c_i)\}$ where $i \in \left\{1, 2, \ldots, \left|\bar{\mathbf{S}}\right|\right\}$, number of classes, and number of instances per class
  $z \leftarrow (c \times t)$ ▷ Number of instances in $\bar{\mathbf{S}}$, i.e., $\left|\bar{\mathbf{S}}\right|$
  for each $(\vec{x}_i, c_i)$ **in** $\bar{\mathbf{S}}$, $i \in \{1, 2, \ldots, z\}$ **do**
    for each $(\vec{x}_j, c_j)$ **in** $\bar{\mathbf{S}}$, $j \in \{1, 2, \ldots, z\} \setminus \{i\}$ **do**
      $distance \leftarrow \text{DIST}(\vec{x}_i, \vec{x}_j)$ ▷ Algorithm 5
      if $c_i \neq c_j$ **then** ▷ Different classes
        $D_\text{b} \leftarrow D_\text{b} + distance$ ▷ Between-class sum
      else
        $D_\text{w} \leftarrow D_\text{w} + distance$ ▷ Within-class sum
      end if
    end for
  end for
  $D_\text{b} \leftarrow D_\text{b} / (z \times (z - t))$ ▷ Average between-class distances
  $D_\text{w} \leftarrow D_\text{w} / (z \times (t - 1))$ ▷ Average within-class distances
  return $\{D_\text{w}, D_\text{b}\}$ ▷ Between- and within-class distances
end function

the following operations are performed. The inputs (terminals) of the program's tree are calculated, e.g., minimum, maximum, mean, and standard deviation of the pixel values in the window. As the terminals become available, the program tree can be evaluated starting from the leaves up to the root (the standard GP procedure to evaluate an individual). At this point, the root node thresholds the values of its children in order to generate a binary code. All negative values map to 0; whereas zero and positive values map to 1. Similar to LBP, the generated code is then converted into the corresponding decimal value, and the bin at the index of this decimal value is incremented. This procedure is depicted in Algorithm 3.

6.3 Experiment Design

The performance of EID^{ri} is assessed by conducting a number of experiments using seven texture image datasets, and compared to well-known image descriptors in

Algorithm 5 Measure the distance between two vectors

function DIST($\vec{u}, \vec{v}$) ▷ Two vectors
  $E \leftarrow |\vec{u}|$ ▷ Number of elements ($|\vec{u}| = |\vec{v}|$)
  $ds \leftarrow 0$
  for i **in** $\{1, 2, \ldots, E\}$ **do**
    if $(\vec{u}_i + \vec{v}_i) \neq 0$ **then** ▷ Different classes
      $ds \leftarrow ds + \left((\vec{u}_i - \vec{v}_i)^2 \,/\, (\vec{u}_i + \vec{v}_i)\right)$
    else ▷ To prevent division the by zero exception
      $ds \leftarrow ds + 0$ ▷ Based on Cha [48]
    end if
  end for
  $distance \leftarrow (0.5 \times ds)$
  return *distance* ▷ Distance between $\vec{u}$ and $\vec{v}$
end function

computer vision. The details of these datasets, parameter settings, methods for comparison, and implementation are provided in this section.

6.3.1 Datasets

Similar to GP-criptor$^{\text{ri}}$, image classification is considered in this chapter to evaluate the performance of EID$^{\text{ri}}$. Seven multi-class image classification datasets are used in this chapter, which are BrNoRo, BrWiRo, KyNoRo, KyWiRo, OutextTC00, OutexTC10, and KySinHw. These datasets are presented and discussed in Section 2.4 (see page 62). Notice that these datasets comprises different number of classes, number of instances in each class, rotation angles, illuminations, and instance dimensions.

6.3.2 Benchmark Methods for Comparison

Since an image descriptor is only responsible for detecting keypoints and extracting features from those keypoints, the goodness of the extracted features to perform classification or detection is used in the literature as a measure to assess the performance of the descriptor [217, 185, 4, 26]. Therefore, and similar to GP-

criptor^{ri}, different machine learning algorithms such as SVM, NB, AdaBoost, J48, RF, NBTree, K*, NNge, k-NN, and MLP, are used in this chapter to evaluate the performance of EID^{ri}. These classifiers are briefly discussed in Section 3.3.3 (see page 89) and Section 4.3.2 (see page 125), and more details are in [323]. The effectiveness of EID^{ri} is also investigated in this study by comparing its performance to a number of common state-of-the-art image descriptors such as DIF, GLCM, $\text{LBP}^{u2}_{p,r}$, $\text{LBP}^{riu2}_{p,r}$, $\text{CLBP}_{p,r}$, $\text{LBC}_{p,r}$, and $\text{CLBC}_{p,r}$ (see Section 2.1.2 on page 29).

6.3.3 Experiments

Two sets of experiments are designed and conducted that aim at investigating different criteria (e.g. window size and performance in terms of image classification). Apart from the evolutionary parameters, the proposed method has only one parameter that requires manual setting, that is the sliding window size. Therefore, the impact of using 3 window sizes, i.e., 3×3, 5×5, and 7×7, on the performance of a simple instance-based classifier (k-NN) using the features of an evolved image descriptor is investigated in the first experiment. This set of experiments can be seen as a parameter tuning stage. Investigating the influence of the evolved descriptors by EID^{ri} on the performance of different types of classification methods and compared to other image descriptors (baseline methods) is the aim of the second set of experiments.

Similar to other EC methods, GP is a stochastic search method initialised with a random *seed* value. Therefore, for each experiment the proposed method has been independently executed 30 times using a different seed value each time and the best evolved program at the end of each run is reported. Moreover, the proposed method randomly selects two instances from each class to evolve an image descriptor. Using different instances for training could affect the evolved program. Hence, the process of 30 independent runs was further repeated 10 more times using different training instances each time. Then the average performance of those 300 (= 30 (runs) $\times$ 10 (repetitions)) best individuals along with the standard deviation are reported. Saying that, the total number of runs for the first experiment is 7 (datasets) $\times$ 3 (windows sizes) $\times$ 10 (repetitions) $\times$ 30 (runs) = 6300. Meanwhile, there are 7 baseline methods, 1 new method (using only the best window size found

Table 6.1: The GP parameters

Parameter	Value	Parameter	Value
Generations	50	Crossover Rate	0.80%
Population Size	300	Mutation Rate	0.20%
Minimum Depth	2	Maximum Depth	10
Selection Type	Tournament	Reproduction	Keep the best
Tournament size	5	Initial Population	Half-and-half

in the first experiment with 30 independent runs), 8 deterministic (single run each) and 2 stochastic (30 runs each) classifiers, 10 repetitions, and 7 datasets; therefore, there are 176120 experiments/runs in total in the second set of experiments.

6.3.4 Parameter Settings

This section discusses the parameter settings for the proposed method, methods for comparison, and the different classification algorithms.

6.3.4.1 Parameter settings for EID$^{\text{ri}}$

The evolutionary parameter settings for EID$^{\text{ri}}$ are identical to that of GP-criptor$^{\text{ri}}$ (see Section 5.3.3 on page 152) apart from the population size that is set to 300 individual instead of 200. Although it is more expensive to use larger population size, allowing the system to have more individuals is worth investigating. Due to the high computation costs of dealing with images, the population size is restricted to 300 individuals. These parameters are summarised in Table 6.1.

6.3.4.2 Parameters of the baseline methods

Similar to [339] and [251], we have observed that the radius (r) and the number of neighbouring pixels (p) for LBP, CLBP, LBC and CLBC have shown very good performance in most of the cases when they are, respectively, set to 3 and 24. Meanwhile, these two parameters, i.e., r and p, have been set to 1 and 8 for the $\text{LBP}^{u2}_{p,r}$ and $\text{DRLBP}_{p,r}$ methods as our experiments revealed that this combination shows a better performance than other settings.

6.3.4.3 Parameters of the classifiers

The parameter settings of the classifiers used in this chapter have been discussed in Section 4.3.3 (see page 127). As the same classification algorithms are used in this chapter, the parameters are also kept consistent between the two chapters.

6.3.5 Implementation

EID^{ri} is implemented using the platform provided by the Evolutionary Computation Java-based (ECJ) package version 23 [190]. The Waikato Environment for Knowledge Analysis (WEKA) package version 3.8 [116] implementations for all other aforementioned classifiers (see Section 3.3.3.1 on page 89) are used.

6.4 Results and Discussions

The results of the experiments are presented and discussed in this section.

6.4.1 Window Size

The aim of the first experiment is to investigate the impact of using different window sizes on the performance of EID^{ri}, and the obtained results are presented in Figure 6.5. On the BrNoRo and BrWiRo datasets, the observed performances for different window sizes were 90.21% and 92.12% for a 3×3 pixels window, 91.04% and 92.59% for a 5×5 pixels window, and 89.17% and 90.78% for a 7×7 pixels window, respectively. The results obtained on the KyNoRo and KyWiRo datasets in this experiment were 86.27% and 87.67% for a 3×3 pixels window, 86.90% and 88.60% for a 5×5 pixels window, and 85.78% and 86.51% for a 7×7 pixels window, respectively. On OutexTC00 and OutexTC10, the proposed method has achieved 87.51% and 85.90% for a 3×3 pixels window, 87.90% and 87.09% for a 5×5 pixels window, and 86.94% and 86.37% for a 7×7 pixels window, respectively. Finally, EID^{ri} obtained 92.85%, 94.06%, and 93.85% average accuracy using a window of size 3×3, 5×5, and 7×7 pixels, respectively. Clearly, the results of these datasets are following a similar pattern and the differences between using different window sizes is not large.

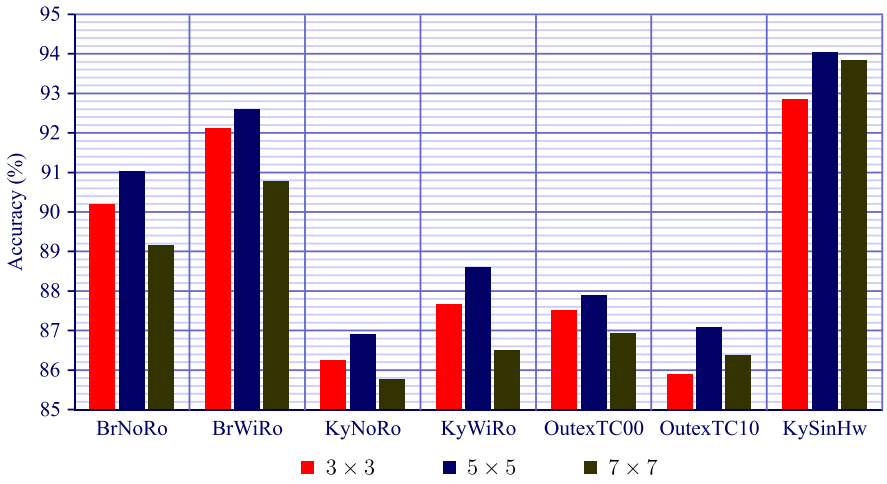


Figure 6.5: The results of the first experiment, which presents the impact of the window size on the performance on the seven datasets.

In summary, EID^{ri} has achieved better results with a window of size 5×5 pixels than those of the other two experimented window sizes, i.e., 3×3 and 7×7 pixels, for all datasets as depicted in Figure 6.5 and Table 6.2. Larger window sizes, e.g., 9×9 and 11×11, were not investigated mainly because the results of the 7×7 window are generally worse than 5×5 window. Based on the results of this experiment, the window size has been set to 5×5 pixels in the second experiment.

6.4.2 Image Classification

The aim of the second set of experiments is to compare the effectiveness of EID^{ri} descriptors against a variety of LBP based and non-LBP based hand-crafted image descriptors that were developed by domain-experts. The results for all datasets are presented in tables in this section, where the first column of each table lists the dataset name; meanwhile the name of the feature extraction methods, i.e., image descriptor, are listed in the second column. The performances of the different classifiers are listed in columns 3 to 12 each of which reports the average accuracy and standard deviation ($\bar{x} \pm s$).

In order to correctly assess the significance of the results, it is very important to use a suitable statistical test. Testing the normality and homoscedasticity are

Table 6.2: The impact of the window size on the performance of EID^{ri} on the seven experimented datasets ($\bar{x} \pm s$).

	Window size (pixels)		
	3×3	5×5	7×7
BrNoRo	90.21 ± 1.97	91.04 ± 2.01	89.17 ± 2.23
BrWiRo	92.12 ± 1.48	92.59 ± 1.08	90.78 ± 1.00
KyNoRo	86.27 ± 2.71	86.90 ± 1.88	85.78 ± 1.73
KyWiRo	87.67 ± 1.89	88.60 ± 1.31	86.51 ± 1.17
OutexTC00	87.51 ± 2.50	87.90 ± 1.84	86.94 ± 2.16
OutexTC10	85.90 ± 1.75	87.09 ± 1.91	86.37 ± 1.77
KySinHw	92.85 ± 1.52	94.06 ± 1.63	93.85 ± 2.06

required prior to the use of a parametric statistical test such as t-test and *analysis of variance* (ANOVA) [66, 68]. The results of the normality test revealed the skewness of the data, i.e., obtained results, which means the normality assumption is not accomplished. Therefore, the significance of the obtained results are tested using a non-parametric statistical test that is the *Wilcoxon signed-rank* test [320, 66] with a significance level of 5%.

The statistical significance test has been performed to investigate how EID^{ri} with a simple 1-NN classification method competes with the other image descriptors using 1-NN and more powerful classifiers. The symbol "$*$" appears next to the method that has been significantly outperformed by EID^{ri}, and a "$-$" is used to indicate that the corresponding method has significantly better performance than that of EID^{ri}. Moreover, the statistical significance test has been applied again to assess whether the proposed method can compete with the baseline methods using the same classifier. In this case, the symbols "$\uparrow$" and "$\downarrow$" are used to, respectively, indicate that EID^{ri} is significantly better and significantly worse than the corresponding method. The overall best performance on each dataset is underlined, and the method with the best performance for each classification method is made **bold**.

6.4.2.1 BrNoRo

The results of the BrNoRo dataset are presented in the first block of Table 6.3. The proposed method has achieved the overall best performance using 1-NN classification

method with an average accuracy of 91.0%. The results of the first significance test show that EID^{ri} has significantly outperformed all the baseline methods even with more sophisticated classifiers. The second significance test revealed that EID^{ri} has significantly better performance than all the 8 baseline descriptors in 5 classifiers (excluding 1-NN), and at least better than 5 descriptors on the other 4 classification methods.

6.4.2.2 BrWiRo

On the BrWiRo dataset, i.e., the rotated version of BrNoRo, the results in the second block of Table 6.3 show that EID^{ri} has achieved the overall best average performance that is 92.6% on this dataset. Similarly, the proposed method with a simple instance-based classifier (1-NN) has significantly outperformed the other baseline hand-crafted descriptors on all 10 classifiers. Apart from $CLBP_{24,3}$ with MLP and SVM, EID^{ri} with the same classification method has achieved significantly better performance than all the 8 baseline descriptors as shown by the results of the second significance test.

6.4.2.3 KyNoRo

The first block of Table 6.4 shows the results obtained on the KyNoRo dataset. The newly introduced method with 1-NN shows the 4^{th} overall best performance as $CLBP_{24,3}$ achieved the first (MLP), second (1-NN), and third (NNge). Using 1-NN, EID^{ri} has significantly better performance than the other descriptors in the vast majority cases, and shows a significant improvement in the performance of 76.25% (61/80) of the 10 classifiers compared to the use of those domain-expert designed descriptors. On the other 23.75% (19/80) of the cases, the proposed method has comparable performance to the domain-expert methods for 11 out of 19 cases, and significantly degraded the performance on the other 8 cases.

6.4.2.4 KyWiRo

On the KyWiRo dataset, the proposed method shows nearly a similar pattern to that on the rotation-free version of this dataset (KyNoRo) as presented in the second block of Table 6.4. Using the simple 1-NN classification method, EID^{ri} scored

Table 6.3: The average accuracy (%) of ten classifiers using nine image descriptors on the **BrNoRo** and **BrWiRo** texture images datasets ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
BrNoRo	DIF	36.8 ± 2.7*	18.0 ± 3.6*↑	28.5 ± 4.9*↑	36.4 ± 3.2*↑	35.3 ± 2.5*↑	21.0 ± 6.4*↑	33.4 ± 4.7*↑	34.3 ± 3.5*↑	31.7 ± 3.5*↑	33.5 ± 2.8*↑
	GLCM	51.5 ± 7.3*	17.0 ± 8.6*↑	33.6 ± 6.1*↑	48.9 ± 7.7*↑	53.6 ± 7.0*↑	38.7 ± 6.5*↑	44.6 ± 6.3*↑	48.0 ± 5.1*↑	43.9 ± 6.2*↑	47.3 ± 6.3*↑
	$LBP^{u2}_{8,1}$	84.0 ± 2.0*	40.4 ± 4.7*↑	33.6 ± 5.1*↑	**86.2 ± 2.4***	83.4 ± 1.6*	62.8 ± 8.3*↑	71.4 ± 5.8*↑	82.1 ± 2.7*↑	56.5 ± 1.6*↑	**75.3 ± 5.3***
	$LBP^{u2ri}_{24,3}$	68.5 ± 3.2*	25.1 ± 4.3*↑	39.0 ± 5.0*↑	60.8 ± 3.5*↑	68.8 ± 2.9*↑	34.7 ± 7.2*↑	46.7 ± 2.6*↑	63.3 ± 3.9*↑	48.4 ± 2.1*↑	62.8 ± 3.6*↑
	$CLBP_{24,3}$	82.4 ± 4.9*	37.3 ± 8.0*↑	35.3 ± 4.4*↑	81.2 ± 4.5*↑	**85.1 ± 3.5***	71.5 ± 5.0*↑	77.4 ± 5.3*↑	83.7 ± 4.7*	61.3 ± 1.8*↑	70.9 ± 5.7*
	$LBC_{24,3}$	66.3 ± 2.8*	23.5 ± 8.0*↑	36.0 ± 6.8*↑	58.2 ± 3.3*↑	67.5 ± 3.1*↑	30.2 ± 7.3*↑	45.0 ± 3.1*↑	61.7 ± 2.8*↑	45.3 ± 2.3*↑	62.9 ± 2.8*↑
	$CLBC_{24,3}$	63.6 ± 2.5*	32.5 ± 3.9*↑	32.7 ± 3.9*↑	67.1 ± 2.9*↑	68.0 ± 2.2*↑	61.9 ± 4.0*↑	67.8 ± 5.9*↑	65.9 ± 1.1*↑	54.5 ± 1.4*↑	50.7 ± 4.3*↑
	$DRLBP_{8,1}$	83.2 ± 2.5*	39.0 ± 4.7*↑	35.7 ± 4.3*↑	80.3 ± 3.2*↑	83.6 ± 2.7*	58.0 ± 10.1*↑	71.0 ± 4.9*↑	82.9 ± 3.8*	59.7 ± 1.5*↑	73.3 ± 6.4*
	EID^{ri}	**91.0 ± 2.0**	**47.2 ± 2.6**	**51.8 ± 1.3**	85.4 ± 1.6	83.1 ± 1.7	**80.3 ± 2.7**	**82.2 ± 1.5**	**85.7 ± 1.9**	**70.3 ± 1.5**	71.8 ± 2.6
BrWiRo	DIF	36.5 ± 2.9*	14.8 ± 4.1*↑	30.9 ± 4.1*↑	34.9 ± 2.5*↑	32.5 ± 2.4*↑	21.2 ± 5.8*↑	34.1 ± 4.1*↑	34.2 ± 4.3*↑	34.1 ± 3.6*↑	34.1 ± 3.4*↑
	GLCM	41.1 ± 6.4*	13.9 ± 5.3*↑	27.8 ± 4.7*↑	39.9 ± 7.3*↑	38.7 ± 4.6*↑	29.1 ± 5.6*↑	39.6 ± 4.6*↑	37.9 ± 5.0*↑	35.3 ± 4.5*↑	38.2 ± 6.0*↑
	$LBP^{u2}_{8,1}$	42.3 ± 1.7*	18.8 ± 2.7*↑	20.1 ± 3.0*↑	41.8 ± 2.0*↑	40.9 ± 2.5*↑	26.1 ± 3.6*↑	33.0 ± 3.2*↑	38.3 ± 3.3*↑	26.3 ± 1.2*↑	36.3 ± 4.0*↑
	$LBP^{u2ri}_{24,3}$	67.6 ± 2.6*	22.9 ± 6.4*↑	40.1 ± 5.3*↑	62.5 ± 2.4*↑	68.6 ± 1.7*↑	39.4 ± 4.5*↑	49.1 ± 4.2*↑	65.8 ± 2.7*↑	51.2 ± 1.9*↑	63.0 ± 4.9*↑
	$CLBP_{24,3}$	85.8 ± 2.7*	34.5 ± 8.1*↑	33.2 ± 3.0*↑	79.2 ± 2.1*↑	**85.8 ± 1.6***↓	71.1 ± 6.5*↑	78.7 ± 4.3*↑	84.7 ± 2.1*↑	62.1 ± 1.4*↑	**74.7 ± 2.8***↓
	$LBC_{24,3}$	64.5 ± 3.0*	22.5 ± 4.8*↑	39.3 ± 2.9*↑	59.9 ± 3.5*↑	64.0 ± 3.1*↑	37.7 ± 3.1*↑	46.9 ± 5.6*↑	61.8 ± 3.3*↑	47.5 ± 1.7*↑	59.7 ± 5.0*↑
	$CLBC_{24,3}$	70.8 ± 3.2*	32.3 ± 7.7*↑	34.4 ± 3.1*↑	73.8 ± 4.0*↑	73.4 ± 3.2*↑	67.1 ± 5.3*↑	72.3 ± 4.1*↑	72.8 ± 2.9*↑	57.2 ± 2.2*↑	58.3 ± 5.6*↑
	$DRLBP_{8,1}$	69.7 ± 2.4*	29.4 ± 7.3*↑	34.3 ± 3.8*↑	66.8 ± 3.0*↑	71.2 ± 3.7*↑	52.7 ± 5.8*↑	60.8 ± 6.4*↑	69.9 ± 3.4*↑	52.2 ± 1.0*↑	63.1 ± 3.8*↑
	EID^{ri}	**92.6 ± 1.1**	**48.9 ± 2.4**	**49.4 ± 1.6**	**87.0 ± 1.7**	84.0 ± 1.4	**81.3 ± 3.0**	**83.0 ± 1.0**	**86.6 ± 1.9**	**70.1 ± 1.2**	72.0 ± 1.9

88.6% accuracy on average which represents the fourth overall best performance on this dataset (KyWiRo) and has significantly outperformed the 8 baseline methods in over 88% of the cases for the 10 classifiers. The results of the second statistical test show that EID^{ri} has a significantly positive influence on the performance of the 10 classifiers compared to the use of the other descriptors' features.

6.4.2.5 OutexTC00

As shown in the first block of Table 6.5, the proposed method is the overall best performing method on the OutexTC00 dataset with 87.9% average accuracy. Apart from $\text{LBP}^{u2}_{8,1}$ with 1-NN, K*, MLP, and NNge, results of the first statistical test show that EID^{ri} with 1-NN has significantly outperformed all other methods with 1-NN and more powerful classifier. Using the same classification method, on the other hand, EID^{ri} has also shown a significant positive impact on the performance in the vast majority (over 87%) of those classifiers. However, in six cases the improvement was not significant and on other four EID^{ri} has either slightly or significantly degraded the performance.

6.4.2.6 OutexTC10

On the rotated version of the OutexTC00, i.e., OutexTC10, the proposed method shows the overall best performance that is 87.1% average accuracy as presented in the second block of Table 6.5. The statistical results of using EID^{ri} with a 1-NN classifier against the baseline methods with 1-NN and other classifiers reveal that EID^{ri} has significantly better performance than the competitor methods apart from $\text{CLBP}_{24,3}$ with 1-NN, MLP, and NNge classifiers. The results of the second significance test show that the features extracted by the proposed method have significantly improved 67 out of 72 of the cases, and only slightly improved or degraded the performance of 5 cases. Only one case, i.e., $\text{CLBP}_{24,3}$ with MLP, shows significantly negative impact of EID^{ri} features.

6.4.2.7 KySinHw

Table 6.6 lists the results obtained on the KySinHw dataset. The winner of the overall best performance on this dataset was $\text{CLBP}_{24,3}$ with NNge classification

Table 6.4: The average accuracy (%) of ten classifiers using nine image descriptors on the **KyNoRo** and **KyWiRo** texture images datasets ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
KyNoRo	DIF	22.2 ± 1.7∗	10.5 ± 2.7∗↑	28.4 ± 1.8∗↑	18.7 ± 1.1∗↑	23.0 ± 2.0∗↑	19.4 ± 2.8∗↑	20.0 ± 2.6∗↑	27.1 ± 2.5∗↑	22.4 ± 0.9∗↑	21.1 ± 1.8∗↑
	GLCM	68.0 ± 3.0∗	18.2 ± 3.9∗↑	**44.5 ± 5.0**∗	68.9 ± 1.8∗↑	64.5 ± 3.7∗↑	48.1 ± 5.7∗↑	55.1 ± 3.1∗↑	65.7 ± 3.8∗↑	56.0 ± 2.1∗↑	67.0 ± 2.8∗
	$LBP^{u2}_{8,1}$	75.5 ± 2.0∗	31.8 ± 3.1∗↑	36.7 ± 4.3∗↑	74.2 ± 2.2∗↑	74.6 ± 2.1∗↑	53.0 ± 7.4∗↑	62.4 ± 4.6∗↑	73.6 ± 3.1∗↑	56.9 ± 1.3∗↑	66.5 ± 3.2∗
	$LBP^{u2ri}_{24,3}$	67.4 ± 2.9∗	20.2 ± 6.5∗↑	38.2 ± 3.3∗↑	66.1 ± 3.2∗↑	66.1 ± 2.6∗↑	40.2 ± 3.7∗↑	50.6 ± 3.7∗↑	64.4 ± 2.8∗↑	53.5 ± 2.7∗↑	66.1 ± 2.9∗
	$CLBP_{24,3}$	**90.6 ± 1.2**−	41.8 ± 2.4∗	33.3 ± 3.5∗↑	37.8 ± 1.3∗↑	**91.0 ± 1.6**−↓	67.5 ± 9.4∗	**77.1 ± 4.0**∗	**90.6 ± 1.6**−↓	63.5 ± 1.0∗↑	**78.7 ± 4.5**∗↓
	$LBC_{24,3}$	66.1 ± 2.8∗	18.4 ± 4.0∗↑	39.8 ± 4.6∗	63.8 ± 3.2∗↑	66.6 ± 2.7∗↑	39.7 ± 5.0∗↑	52.0 ± 4.0∗↑	64.4 ± 3.4∗↑	52.4 ± 3.3∗↑	65.3 ± 2.7∗↑
	$CLBC_{24,3}$	76.7 ± 4.1∗	38.3 ± 3.9∗↑	32.8 ± 3.9∗↑	68.0 ± 3.6∗↑	78.1 ± 3.8∗	64.2 ± 7.4∗↑	69.4 ± 4.5∗↑	77.8 ± 2.5∗↑	59.7 ± 1.6∗↑	51.3 ± 2.4∗↑
	$DRLBP_{8,1}$	86.3 ± 1.1	32.3 ± 6.8∗↑	37.1 ± 3.6∗↑	**85.0 ± 1.5**∗↓	86.9 ± 1.3 ↓	64.3 ± 5.1∗↑	73.0 ± 3.8∗	85.9 ± 1.8 ↓	62.5 ± 1.8∗↑	76.1 ± 3.4∗↓
	EID^{ri}	86.9 ± 1.9	**43.3 ± 1.0**	41.4 ± 1.3	82.4 ± 2.0	80.1 ± 2.0	**70.9 ± 4.1**	75.7 ± 1.7	83.1 ± 2.2	**65.1 ± 1.4**	68.3 ± 1.6
KyWiRo	DIF	23.0 ± 0.9∗	12.8 ± 2.3∗↑	28.5 ± 2.9∗↑	19.8 ± 0.9∗↑	20.0 ± 1.6∗↑	21.9 ± 0.9∗↑	22.1 ± 1.4∗↑	22.0 ± 0.1∗↑	23.3 ± 0.5∗↑	17.7 ± 2.7∗↑
	GLCM	49.0 ± 0.5∗	17.0 ± 2.9∗↑	32.7 ± 2.3∗↑	49.7 ± 0.7∗↑	47.3 ± 0.2∗↑	34.1 ± 2.8∗↑	42.0 ± 2.1∗↑	46.1 ± 0.5∗↑	39.9 ± 1.0∗↑	46.9 ± 1.1∗↑
	$LBP^{u2}_{8,1}$	42.6 ± 1.8∗	19.7 ± 2.3∗↑	25.8 ± 3.3∗↑	39.1 ± 1.8∗↑	45.2 ± 2.1∗↑	27.0 ± 4.6∗↑	38.4 ± 4.2∗↑	44.2 ± 4.7∗↑	32.1 ± 1.2∗↑	37.3 ± 2.6∗↑
	$LBP^{u2ri}_{24,3}$	69.5 ± 3.3∗	20.3 ± 5.2∗↑	**42.9 ± 5.0**∗	67.0 ± 3.7∗↑	66.5 ± 2.6∗↑	43.1 ± 4.8∗↑	51.3 ± 3.5∗↑	66.1 ± 2.5∗↑	54.8 ± 1.6∗↑	66.3 ± 2.3∗↑
	$CLBP_{24,3}$	**89.0 ± 2.8**	40.9 ± 6.3∗	32.4 ± 3.6∗↑	40.1 ± 2.4∗↑	**90.8 ± 1.8**−↓	69.1 ± 8.4∗	**78.1 ± 5.1**∗	**89.5 ± 1.9** ↓	62.1 ± 1.3∗↑	**79.3 ± 3.9**∗↓
	$LBC_{24,3}$	68.3 ± 3.6∗	17.6 ± 4.2∗↑	41.2 ± 4.1∗	65.2 ± 4.0∗↑	64.9 ± 2.9∗↑	41.8 ± 5.0∗↑	50.3 ± 4.2∗↑	64.7 ± 2.9∗↑	52.9 ± 2.0∗↑	64.1 ± 2.9∗↑
	$CLBC_{24,3}$	76.6 ± 3.8∗	37.2 ± 2.1∗↑	30.3 ± 3.0∗↑	66.9 ± 3.3∗↑	77.4 ± 4.4∗↑	65.4 ± 5.4∗↑	69.0 ± 3.6∗↑	75.7 ± 3.8∗↑	59.0 ± 1.6∗↑	56.5 ± 5.8∗↑
	$DRLBP_{8,1}$	74.1 ± 2.1∗	29.1 ± 5.3∗↑	36.0 ± 2.7∗↑	71.9 ± 2.0∗↑	75.7 ± 2.4∗↑	59.9 ± 5.8∗↑	65.2 ± 3.0∗↑	75.4 ± 3.1∗↑	56.0 ± 1.3∗↑	64.9 ± 3.8∗↑
	EID^{ri}	88.6 ± 1.3	**43.0 ± 2.0**	41.2 ± 1.1	**84.4 ± 1.7**	80.5 ± 1.1	**72.4 ± 3.8**	76.5 ± 1.6	84.0 ± 1.4	**65.9 ± 0.9**	68.6 ± 1.8

Table 6.5: The average accuracy (%) of ten classifiers using nine image descriptors on the **OutexTC00** and **OutexTC10** texture images datasets ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
OutexTC00	DIF	17.8 ± 1.4*	8.30 ± 2.7*↑	12.8 ± 4.8*↑	14.8 ± 1.9*↑	15.7 ± 1.9*↑	11.5 ± 2.6*↑	12.2 ± 4.4*↑	16.3 ± 1.3*↑	13.2 ± 1.4*↑	15.0 ± 2.8*↑
	GLCM	70.9 ± 2.2*	20.0 ± 7.3*↑	36.0 ± 7.9*↑	60.8 ± 4.8*↑	66.1 ± 4.9*↑	41.2 ± 12.5*↑	52.0 ± 7.1*↑	69.4 ± 3.0*↑	49.7 ± 5.4*↑	70.2 ± 2.8*↑
	$LBP^{u2}_{8,1}$	**87.9 ± 1.2**	40.8 ± 12.0*	36.9 ± 3.8*↑	87.5 ± 1.7	**87.5 ± 1.2** ↓	67.2 ± 10.3*	73.5 ± 3.1*↑	**86.8 ± 1.4**	65.6 ± 1.5*↑	**80.4 ± 3.0***↓
	$LBP^{u2ri}_{24,3}$	69.5 ± 2.9*	26.8 ± 7.9*↑	39.2 ± 6.1*↑	67.9 ± 2.4*↑	66.5 ± 3.3*↑	44.3 ± 6.9*↑	51.6 ± 4.2*↑	68.8 ± 2.2*↑	53.0 ± 2.1*↑	69.7 ± 2.8*↑
	$CLBP_{24,3}$	81.0 ± 2.9*	37.4 ± 7.6*↑	23.4 ± 3.6*↑	27.0 ± 2.2*↑	82.1 ± 1.3*↑	61.3 ± 4.2*↑	68.2 ± 4.3*↑	80.7 ± 2.5*↑	56.8 ± 1.3*↑	70.2 ± 6.5*
	$LBC_{24,3}$	68.2 ± 3.8*	20.2 ± 4.6*↑	36.2 ± 3.8*↑	68.0 ± 4.1*↑	63.9 ± 3.7*↑	39.6 ± 5.8*↑	48.5 ± 3.0*↑	66.6 ± 3.9*↑	50.5 ± 1.7*↑	66.8 ± 3.0*↑
	$CLBC_{24,3}$	72.8 ± 2.9*	37.1 ± 5.8*↑	30.4 ± 3.6*↑	69.1 ± 2.9*↑	73.8 ± 2.9*↑	64.3 ± 4.9*↑	66.1 ± 4.9*↑	71.1 ± 3.5*↑	57.3 ± 1.5*↑	55.0 ± 4.7*↑
	$DRLBP_{8,1}$	85.0 ± 1.8*	34.1 ± 10.7*↑	40.9 ± 4.3*↑	83.4 ± 1.4*↑	83.7 ± 1.5*	64.1 ± 5.5*↑	71.6 ± 3.8*↑	83.3 ± 1.7*↑	66.9 ± 1.5*↑	79.5 ± 4.8*↓
	EID^{ri}	**87.9 ± 1.8**	**46.0 ± 2.1**	**47.8 ± 2.3**	**87.7 ± 2.2**	84.6 ± 1.7	**72.2 ± 3.6**	**80.1 ± 1.9**	86.0 ± 2.0	**73.3 ± 1.2**	74.5 ± 1.6
OutexTC10	DIF	8.20 ± 0.7*	6.60 ± 1.5*↑	11.4 ± 2.1*↑	7.40 ± 0.7*↑	8.40 ± 0.7*↑	7.50 ± 1.6*↑	7.70 ± 1.7*↑	9.40 ± 0.7*↑	9.10 ± 0.8*↑	7.80 ± 1.2*↑
	GLCM	43.8 ± 2.5*	17.3 ± 4.8*↑	26.4 ± 6.5*↑	41.2 ± 6.2*↑	36.5 ± 3.3*↑	28.0 ± 4.9*↑	38.0 ± 3.8*↑	43.3 ± 3.6*↑	34.4 ± 3.9*↑	44.2 ± 3.8*↑
	$LBP^{u2}_{8,1}$	34.4 ± 1.8*	16.1 ± 3.5*↑	17.7 ± 4.3*↑	32.8 ± 1.2*↑	34.8 ± 2.1*↑	20.4 ± 3.4*↑	28.2 ± 4.7*↑	32.8 ± 1.2*↑	22.5 ± 0.9*↑	28.9 ± 4.4*↑
	$LBP^{u2ri}_{24,3}$	64.5 ± 1.0*	25.9 ± 5.3*↑	38.4 ± 4.4*	64.3 ± 2.0*↑	64.2 ± 2.4*↑	34.5 ± 7.0*↑	49.6 ± 3.3*↑	61.2 ± 2.8*↑	47.1 ± 1.2*↑	62.3 ± 1.7*↑
	$CLBP_{24,3}$	86.1 ± 2.4	33.8 ± 5.6*↑	26.6 ± 2.7*↑	16.1 ± 1.5*↑	**86.9 ± 2.3** ↓	51.6 ± 7.5*↑	74.3 ± 6.8*	**86.5 ± 2.6**	59.0 ± 1.0*↑	**74.8 ± 5.0***
	$LBC_{24,3}$	60.5 ± 1.8*	20.9 ± 4.3*↑	32.8 ± 5.7*↑	59.8 ± 1.7*↑	59.4 ± 2.7*↑	33.9 ± 4.1*↑	45.4 ± 3.2*↑	55.6 ± 3.6*↑	44.0 ± 1.5*↑	57.8 ± 2.4*↑
	$CLBC_{24,3}$	75.5 ± 2.2*	31.5 ± 7.0*↑	29.2 ± 4.0*↑	70.2 ± 2.3*↑	77.3 ± 2.5*↑	59.9 ± 6.9*↑	71.7 ± 4.6*↑	75.2 ± 2.4*↑	58.8 ± 1.9*↑	53.4 ± 3.9*↑
	$DRLBP_{8,1}$	64.0 ± 2.6*	27.2 ± 5.5*↑	29.8 ± 5.9*↑	59.4 ± 1.9*↑	61.3 ± 3.5*↑	42.2 ± 8.8*↑	59.3 ± 5.4*↑	64.7 ± 3.8*↑	52.0 ± 0.8*↑	55.4 ± 5.1*↑
	EID^{ri}	**87.1 ± 1.9**	**41.8 ± 2.5**	**41.2 ± 1.6**	**86.2 ± 1.6**	84.3 ± 1.5	**71.6 ± 4.4**	**78.4 ± 2.2**	86.0 ± 1.9	**68.8 ± 1.6**	72.3 ± 1.3

method, which achieved 97.9% accuracy on average. However, EID^{ri} is ranked fourth best performance (94.1%) amongst the other methods on the KySinHw dataset. EID^{ri} with 1-NN has also significantly outperformed over 96% of the other methods with different classification methods as revealed by the results of the first statistical test. Checking the influence of EID^{ri} on the performance of each classification method compared to that of the other methods shows that EID^{ri} features have significantly improved over 81% of the cases, but also significantly degraded the performance of 11% of the cases. The performances of other 8% cases show either slightly improvement or deterioration when the features of the proposed method are used.

6.4.3 Summary

The following observations can be deduced from the results above.

- The proposed method does not require human intervention to automatically evolve a rotation-invariant image descriptor;
- The system does not require domain knowledge and only uses two instances of each class to find a set of good keypoints that can lead to a significantly better performance than using domain-expert designed descriptors in most cases;
- The newly introduced method does not solely detect a set of predefined/designed keypoints such as lines and corners; rather, it automatically designs a set of keypoints and determines how to detect those keypoints;
- The use of EID^{ri} features have, in the majority of the cases, improved the performance of the different classification methods compared to the use of the other descriptors' features;
- The proposed method **dynamically** sets the length of the feature vector by automatically selecting the number of the bits in the binary codes, i.e., children of the *code* node;

Table 6.6: The average accuracy (%) of ten classifiers using nine image descriptors on the **KySinHw** texture images dataset ($\bar{x} \pm s$).

		1-NN	AdaBoost	J48	K*	MLP	NB	NBTree	NNge	RF	SVM
KySinHw	DIF	11.6 ± 1.1*	9.00 ± 1.6*↑	20.5 ± 1.9*↑	8.20 ± 1.0*↑	10.9 ± 1.6*↑	11.5 ± 2.5*↑	13.2 ± 3.6*↑	11.9 ± 1.8*↑	15.8 ± 1.4*↑	11.6 ± 1.8*↑
	GLCM	69.6 ± 2.3*	13.2 ± 3.9*↑	41.0 ± 3.6*	67.6 ± 2.6*↑	64.9 ± 3.0*↑	51.5 ± 6.8*↑	61.6 ± 3.2*↑	67.2 ± 3.0*↑	55.2 ± 2.0*↑	67.3 ± 2.3*↑
	$LBP^{u2}_{8,1}$	54.8 ± 2.2*	22.8 ± 3.3*↑	28.2 ± 3.3*↑	48.6 ± 1.6*↑	54.8 ± 3.8*↑	27.1 ± 7.8*↑	45.0 ± 3.0*↑	51.5 ± 6.1*↑	35.0 ± 1.3*↑	48.4 ± 4.3*↑
	$LBP^{u2ri}_{24,3}$	81.8 ± 1.5*	26.1 ± 5.6*↑	**48.4 ± 3.4***↓	78.8 ± 1.6*↑	81.0 ± 1.7*↑	46.4 ± 6.5*↑	60.3 ± 4.2*↑	78.9 ± 2.3*↑	63.5 ± 1.5*↑	78.8 ± 1.7*↓
	$CLBP_{24,3}$	**97.3 ± 0.7**−	**46.7 ± 4.7***	31.6 ± 3.9*↑	23.1 ± 1.9*↑	**96.5 ± 1.1**−↓	74.8 ± 5.4*↑	83.1 ± 4.1*	**97.9 ± 1.0**−↓	67.0 ± 1.1*	**87.3 ± 4.6***↓
	$LBC_{24,3}$	80.7 ± 1.5*	20.6 ± 7.6*↑	46.3 ± 4.1*↓	77.7 ± 1.7*↑	79.5 ± 2.0*↑	46.5 ± 6.8*↑	60.2 ± 2.1*↑	77.5 ± 2.8*↑	60.6 ± 1.5*↑	77.6 ± 1.6*↓
	$CLBC_{24,3}$	89.0 ± 1.9*	46.2 ± 2.7*	37.0 ± 4.1*	87.3 ± 2.0*↑	89.5 ± 1.5*↓	73.0 ± 4.4*↑	78.2 ± 5.1*↑	89.2 ± 1.9*↑	**67.6 ± 1.3***	69.7 ± 3.5*↑
	$DRLBP_{8,1}$	85.3 ± 2.3*	34.9 ± 8.6*↑	37.8 ± 5.1*	80.6 ± 2.3*↑	87.1 ± 2.9*	55.0 ± 7.9*↑	72.4 ± 3.8*↑	86.6 ± 2.6*↑	62.7 ± 2.0*↑	77.3 ± 5.2*
	EID^{ri}	94.1 ± 1.6	44.8 ± 1.8	39.3 ± 1.3	**91.4 ± 2.3**	87.1 ± 1.8	**80.2 ± 3.0**	**84.1 ± 1.5**	92.1 ± 1.4	67.3 ± 1.3	74.8 ± 1.5

- Unlike other hand-crafted descriptors, changing the window size does not require changing or redesigning the system since it is handled automatically in EID^{ri} as the results of the first experiment suggest; and
- The new method can build complicated functions using the combination of simple ones, while it also has the flexibility to use different functions in the function set.

6.5 Further Analysis

In this section, EID^{ri} is investigated in depth by considering how and why the proposed method can perform well. The convergence, time required to evolve a descriptor, and number of children of the *code* node and the window size effect on the number of children are discussed in the first subsection. A program evolved by EID^{ri} is analysed and discussed in details in the second subsection.

6.5.1 Overall Analysis

The average fitness value per generation for 30 independent runs using different seed values on two randomly selected instances per class of the BrWoRo dataset is depicted in Figure 6.6. A closer inspection of this graph reveals that on average the programs have made larger jumps in the first few generations than the later generations. The fitness value is decreased from 2.10 to 0.088 in the first 20 generations compared to the decrease in fitness from 0.088 to 0.070 over the remainder 30 generations. The standard deviation bars of those 30 independent runs show a similar pattern where the earlier generations have more variations than the later ones.

The average time needed to evolve an image descriptor by EID^{ri} is presented in Figure 6.7 and Table 6.7. Clearly, the time required to evolve an individual is affected by the number of classes, size of each instance, and size of the sliding window. Having more classes results in having more instances to scan by each GP individual in each generation. Similarly, a larger instance means more pixels are required to be scanned. Although increasing the window size is assumed to reduce the evolutionary time as more pixels (the edges) are ignored, Figure 6.7

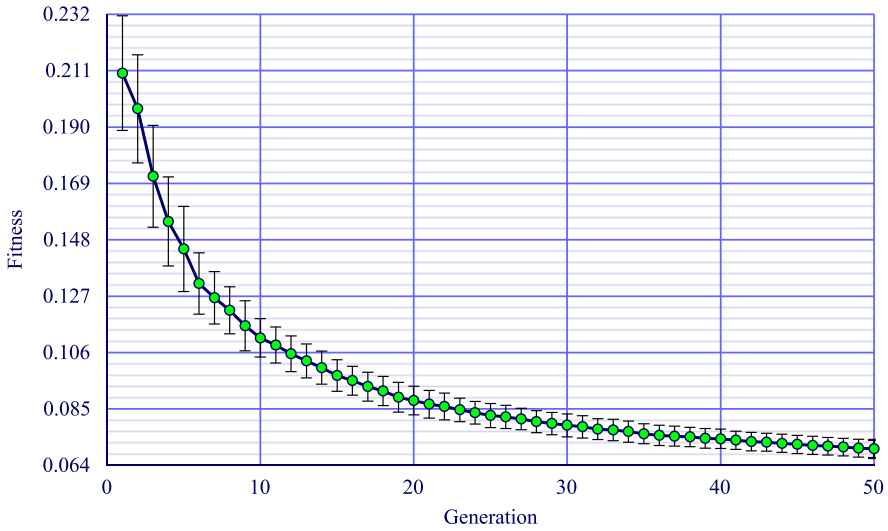


Figure 6.6: The average fitness value per generation on the BrWoRo dataset (the whiskers represent the standard deviation).

shows that increasing the window size considerably increases the evolution time. Calculating the minimum, maximum, mean, and standard deviation values require iterations proportional to the number of pixels. This process can be optimised by pre-calculating those values instead of re-calculating them for each individual at each generation. However, storing those pre-calculated values for each instance requires a large amount of memory as each pixel, i.e., a single integer value, will be associated with four floating values.

The frequency of the number of children under the *code* node has been analysed as it is automatically determined during the evolutionary process. The bar plot presented in Figure 6.8 (and Table 6.8) shows the distribution of code lengths, i.e., number of children for the *code* node, across all independent runs of the seven datasets, and categorised based on the different window sizes. The plot shows clearly that the vast majority of the evolved descriptors have a *code* node with 8, 9 or 10 children. The percentage of those programs with a *code* node having more than 7 children occupy 85.6%, 86.1%, and 86.4% of the evolved programs for window sizes 3×3, 5×5, and 7×7, respectively. Also none of the best evolved programs has less than 3 children under the *code* node, only one program with 3, and only six with 4 nodes. Note that the length of the feature vector is doubled

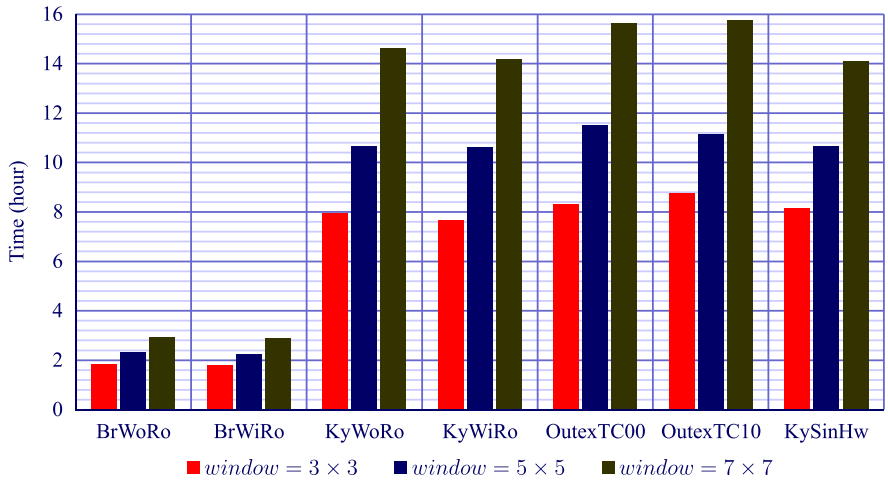


Figure 6.7: The impact of the window size on the average time required to evolve a program by EID^{ri}.

for every node added to the children list of the *code* node, which means more space is required to store those feature vectors. Moreover, from the analysis of those programs with 9 or 10 children, it has been observed that only a few cells of the feature vectors have values/counts, whilst the other cells have zero values across all instances. Removing those unused cells can potentially reduce the memory required to store the feature vectors.

6.5.2 An Evolved Descriptor

A sample program evolved by EID^{ri} on the BrNoRo dataset is presented in Figure 6.9. Using a window of size 5×5 pixels, this program has achieved 92.14% accuracy on the unseen data. Overall, there are 118 nodes in this program where 61 nodes are leaves and the other 57 are functions. The *code* node has 5 children/branches, which means that the feature vector for an image is with a length of $2^5 = 32$ values. Although some of those branches are difficult to interpret, other branches can be simplified and presented as mathematical formulae. For instance, the corresponding formulae for the third, fourth and fifth branches can be presented as $2mean - (min + max)$, $(2mean - max) \,/\, (stdev - max)$, and $(mean + max + \frac{mean-min}{stdev}) - \Big((mean - min)\,\frac{stdev}{max}\Big)$, respectively. As only a few simple mathematical operations

Table 6.7: The impact of the window size on the average time (hour) required to evolve a program by EID^{ri} on the seven experimented datasets.

	Window size (pixels)		
	3×3	5×5	7×7
BrNoRo	1.84	2.32	2.95
BrWiRo	1.81	2.25	2.89
KyNoRo	7.94	10.66	14.65
KyWiRo	7.67	10.65	14.19
OutexTC00	8.30	11.52	15.64
OutexTC10	8.75	11.16	15.78
KySinHw	8.16	10.67	14.09

Table 6.8: The relative frequency of the number of children for the *code* node of different window sizes.

	Window size (pixels)		
	3×3	5×5	7×7
Code length = 1	0.0000	0.0000	0.0000
Code length = 2	0.0000	0.0000	0.0000
Code length = 3	0.0000	0.0000	0.0005
Code length = 4	0.0005	0.0005	0.0019
Code length = 5	0.0205	0.0162	0.0224
Code length = 6	0.0581	0.0557	0.0624
Code length = 7	0.0648	0.0667	0.0486
Code length = 8	0.2371	0.2638	0.2700
Code length = 9	0.3357	0.3014	0.3119
Code length = 10	0.2833	0.2957	0.2824

are required to generate the feature vector for an image, such descriptors are efficient, i.e., they have the potential to operate very fast, and therefore they can be used for online/real-time applications.

Table 6.9 presents the confusion matrix for the program depicted in Figure 6.9. The first column of this table lists the actual class labels, whereas the first row lists the predicted class labels. The proportion of correctly classified instances for

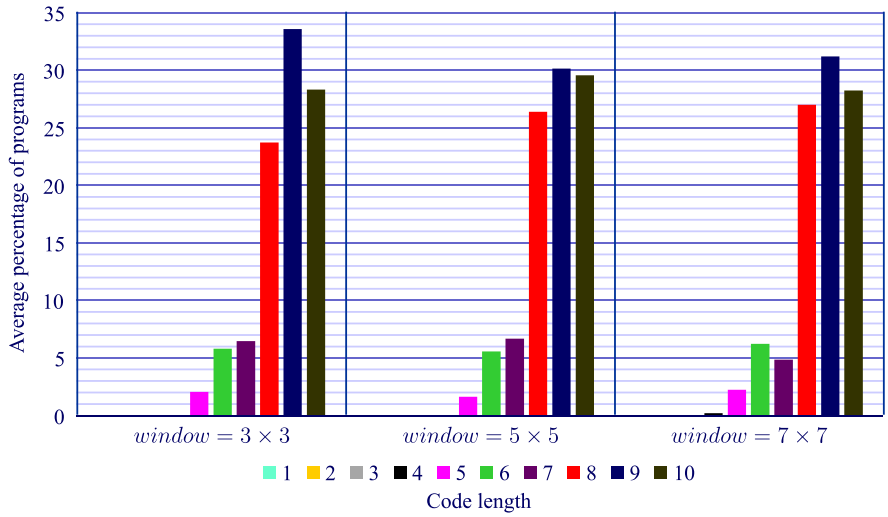


Figure 6.8: The relative frequency of the number of children for the *code* node of different window sizes.

each class is presented in the last column of Table 6.9. The program has correctly classified either all instances (100% accuracy) or achieved over 90% accuracy for 17 out of the 20 classes, scored over 70% accuracy on 2 further classes, and did not perform well on only one class (D09) with 55% accuracy. The confusion matrix shows that 19 out of 42 instances of the D09 class have been misclassified as D04. A sample instance from each of these two classes, i.e., D04 and D09, are randomly selected and two tiles of each are enlarged (zoomed-in) in Figure 6.10. A close look at the enlarged tiles reveals the similarity between the instances of the two classes especially when the rotation variation is considered (the position order of the pixel values is ignored within the sliding window).

The program presented in Figure 6.9 is further analysed by feeding two randomly selected instances from each class of the BrNoRo dataset to generate the corresponding feature vectors that are visually depicted in Figure 6.11. In this figure, the two instances drawn from each class are positioned side-by-side to ease the comparison task, and the corresponding class label is shown on the horizontal axis. The values of each feature vector are normalised and represented as the percentage that are placed on top of each other. The $2^5 = 32$ features $\{f_1, f_2, \ldots, f_{32}\}$ are indicated using different colours as depicted in the legend of Figure 6.11. The

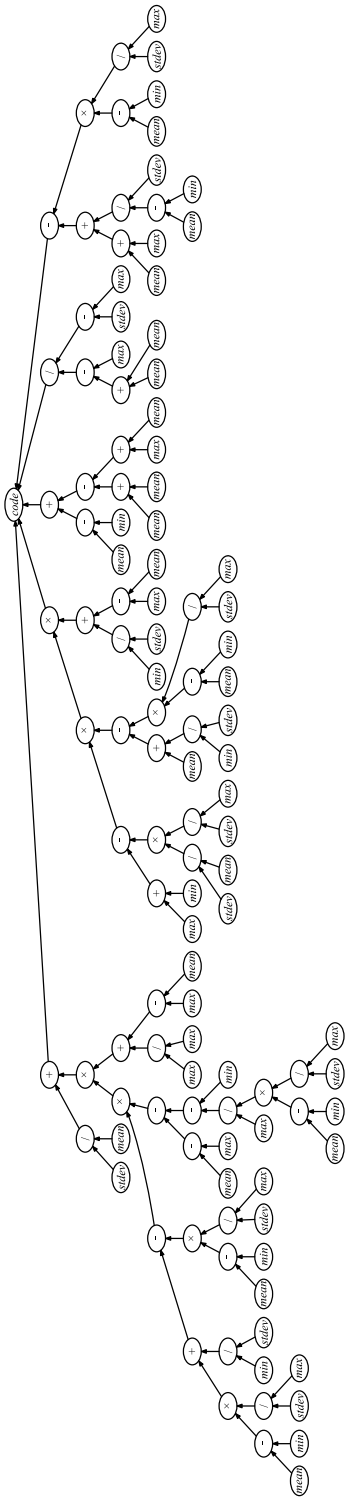


Figure 6.9: A sample program evolved by EID^{ri} on the BrNoRo dataset (5 bits).

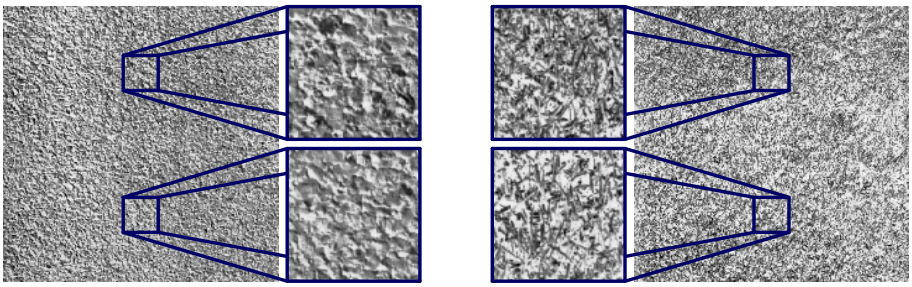
Table 6.9: The confusion matrix for the program presented in Figure 6.9 on the **BrNoRo** dataset.

	D01	D03	D04	D05	D06	D09	D11	D14	D15	D16	D17	D18	D20	D21	D24	D34	D37	D46	D47	D49	Total %
D01	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	90.5
D03	0	40	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	95.2
D04	0	0	30	0	0	9	2	0	0	0	0	0	0	0	1	0	0	0	0	0	71.4
D05	0	0	0	40	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	95.2
D06	0	0	0	0	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
D09	0	0	19	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54.8
D11	0	0	2	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	95.2
D14	0	1	0	0	0	0	0	41	0	0	0	0	0	0	0	0	0	0	0	0	97.6
D15	0	0	2	0	0	0	0	0	38	0	0	0	0	0	2	0	0	0	0	0	90.5
D16	0	0	0	0	0	0	0	0	0	39	2	0	0	1	0	0	0	0	0	0	92.9
D17	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	0	0	0	0	0	100
D18	0	0	0	9	0	0	0	0	0	0	0	31	0	0	0	0	2	0	0	0	73.8
D20	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	0	0	0	100
D21	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	0	0	100
D24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	0	100
D34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0	0	100
D37	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	38	1	0	0	90.5
D46	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	40	0	0	95.2
D47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	0	100
D49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42	100

figure clearly shows that the evolved program has generated a "fingerprint" for the instances belonging to the same class that is different from all other instances from the other classes. Some examples are the class D16 that has a very high percentage of f_3, class D21 with a high percentage of f_9, and class D49 with high f_{31} percentage. The program also identified some features that occur in some classes but not in others, e.g., feature f_{12} that does not appear in the vectors of classes D01, D03, D06, D14, D16, D21, D46, and D49. The similarity between D04 and D09 instances is also noticeably high, which explains why a large number of instances of these two classes were misclassified.

6.6 Chapter Summary

In this chapter, GP has been successfully utilised to automatically synthesise a set of mathematical formulae that form an image descriptor. Unlike other image



(a) (b)

Figure 6.10: A sample instance and two enlarged tiles of the (a) D04 class, and (b) D09 class.

descriptors, the proposed method does not require human intervention to design a set of keypoints, or any mechanism for detecting those keypoints and extracting the feature values. Domain knowledge is not required either. In particular, the proposed method *automatically determine the length of the feature vector* during the evolutionary process, and uses two instances of each class to evolve a descriptor that is capable of generating distinctive feature vectors for instances belong to different classes. As this method uses only a few training instances, it is suitable for problems where the number of labelled data is limited. Another impact of using a few training instances is that the overall complexity of the system in terms of time and physical computer memory will be reduced, which makes the system suitable for the situations that cannot afford a long time for training. Similar to GP-criptor$^{\text{ri}}$ (Chapter 5), the proposed method uses a set of first-order statistics to evolve a rotation-invariant descriptor relying on the order-independent characteristic of those statistics. An evolved descriptor works in a pixel-by-pixel fashion, using a sliding window of a predefined size. To assess the effectiveness of the proposed method, seven texture image classification datasets with different degrees of rotations were used and compared to the effectiveness of eight state-of-the-art expert-designed and hand-crafted image descriptors using ten widely used classification algorithms. Quantitatively, the results of the experiments reveal the effectiveness of the proposed GP method to evolve an image descriptor using only two instances per class and yet can significantly outperform or achieve comparable results to the hand-crafted descriptors. Qualitatively, the analysis of the proposed

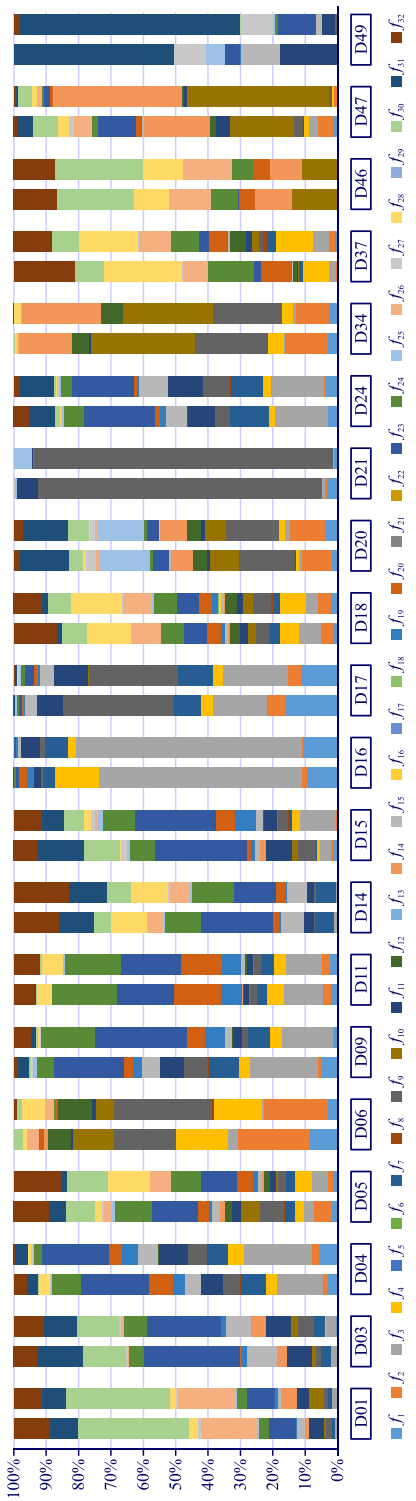


Figure 6.11: A visual representation for the feature vectors of 40 randomly selected instances from the 20 classes (two from each class) of the BrNoRo dataset, which were generated by the program presented in Figure 6.9.

system and an evolved descriptor demonstrates the robustness to tackle the rotation variation and the interpretability of the evolved descriptor.

The use of a descriptor evolved by GP on one dataset directly on another dataset, i.e., learning by knowledge transfer, is thoroughly investigated in the next chapter.

7

Transfer Learning in GP: A Study on the Generalisability of GP Evolved Image Descriptors

7.1 Introduction

The majority of the algorithms in machine learning are designed based on two assumptions. The first assumption is that both of the training and future (unseen) data having the same distribution or follow the same pattern [227, 86]. This assumption is not satisfied by many real-world applications, and hence, a well-trained model, e.g., classifier, may perform poorly on the unseen data. A typical example is that a model trained on a set of clean (noise-free) instances, while the test set comprises instances that show a severe level of noise. The second assumption is that a large number of training instances are available in order to build a good model [134, 252, 77]. In some situations, it may be difficult or expensive to acquire a sufficiently large number of instances such as in the medical

domain [342]. Transfer learning, or learning by knowledge transfer, has emerged inspired by the learning ability of humans where future problems can be tackled based on the knowledge gained from previously solved problems [141]. Different approaches and methods have been proposed where transfer learning can be used to tackle different problems of the same as well as different domains (see Section 2.1.3.5 on page 38 and Section 2.3.4 on page 59).

The vast majority of existing image descriptors were designed to detect keypoints that are believed to be reliable and are common across different domains. LBP and its variants, and SIFT and its variants are typical examples that have been widely used to perform texture classification on different datasets, object detection for diverse type of objects, and face recognition and many more. This is a very important attribute of such descriptors, as it reduces the requirement to develop different descriptors for different domains and applications.

7.1.1 Chapter Goals

By adopting transfer learning, specifically learning by model transfer, the overall aim of this chapter is to examine the generalisability of the image descriptors evolved by GP-criptor$^{\text{ri}}$ to facilitate performing classification on datasets of the same domain as well as of different domains. In other words, a descriptor evolved on the source domain will be directly used to tackle the problem in the target domain using only two instances per class of the target domain. Specifically, this chapter intends to investigate the following objectives.

- Investigate the impact of different source domains on the performance in the problem of the target domain.
- Investigate the impact of the window size on the performance between the source and target domains.
- Investigate the impact of the feature vector length on the performance between the source and target domains.
- Investigate the impact of the number of classes between the source and target domains.

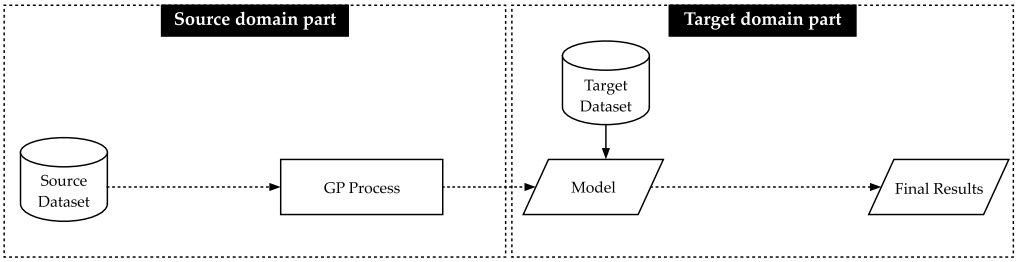


Figure 7.1: The overall transfer learning framework used in this chapter, where dashed arrows mean there are hidden states.

- Compare the generalisability of the automatically evolved descriptors to that of eight state-of-the-art domain-expert designed image descriptors.

7.1.2 Chapter Organisation

The remainder of the chapter is organised as follows. The general frame-work of how to transfer a model from the source domain to the target domain is discussed in Section 7.2. Section 7.3 presents the aims and details of the different experiments. The results are presented and discussed in Section 7.4. Section 7.5 summarises the discussion of this chapter.

7.2 A GP Transfer Learning Framework

This section provides detailed discussion regarding the GP framework of transfer learning adopted in this study. The framework comprises two parts each of which aims at performing a specific task. The first part is performed on the source domain, whereas the second part is performed on the target domain. The overall flowchart of this framework is depicted in Figure 7.1.

7.2.1 The Source Domain Part

The first part of the framework is performed on the source domain. In transfer learning, typically the entire source domain is used to build a model [86, 160]. However, in this chapter the evolved programs by GP-criptor$^{\text{ri}}$ (see Section 5.2 on

page 147) are examined. Moreover, using a large training set to evolve a program in GP-criptor$^{\text{ri}}$ is a very expensive and time-consuming process mainly because the method has been specifically designed to cope with only a few examples. As depicted in Figure 7.2, only two instances of each class of the source domain are randomly selected and fed into GP. The GP system will run until the last generation is reached, and then the best evolved program is returned.

Due to the stochastic nature of GP, the evolved program can be very different based on the seed value used between the different runs which represents the first random component of this part. Hence, using the same two instances per class, the process is repeated 30 independent times using a different seed value each time. The best program that is evolved at the end of each run is reported so it can be used later on the target domain. Therefore, we will have 30 different programs (image descriptors) resulting from the use of different seed values.

The second random component of this part is the selected instances to evolve a descriptor. The two instances that are selected to evolve the descriptor can largely affect the final result. Therefore, the entire process has been further repeated 10 times using different instances each time. Using different instances to evolve a descriptor requires executing the 30 independent runs with different seed values as discussed earlier. The final result of this part of the framework is 300 (= 10 (repeats) × 30 (runs)) image descriptors.

7.2.2 The Target Domain Part

The second part of the framework deals with the instances of the target domain. As depicted in Figure 7.2, each descriptor ω resulting from a single run of the first part of this framework, i.e., on the source domain, is used to generate a transformed target domain dataset by generating the feature vector for each instance. If there are α source datasets and β target datasets, the i^{th} source and j^{th} target datasets are denoted as $\mathbf{S}_i$ and $\mathbf{T}_j$, respectively. Here $\mathbf{S}_i = \left\{ \left(\vec{x}_1^i, c_1^i\right), \left(\vec{x}_2^i, c_2^i\right), \ldots, \left(\vec{x}_{z_i}^i, c_{z_i}^i\right) \right\}$, where the j^{th} instance, i.e., feature vector and the corresponding class label, of the i^{th} source dataset is denoted as $\left(\vec{x}_j^i, c_j^i\right)$ and z_i is the total number of instances in $\mathbf{S}_i$. Similarly, the i^{th} dataset in the target domain is denoted as $\mathbf{T}_i = \left\{ \left(\vec{x}_1^i, c_1^i\right), \left(\vec{x}_2^i, c_2^i\right), \ldots, \left(\vec{x}_{z_i}^i, c_{z_i}^i\right) \right\}$. The feature vector for each instance in a

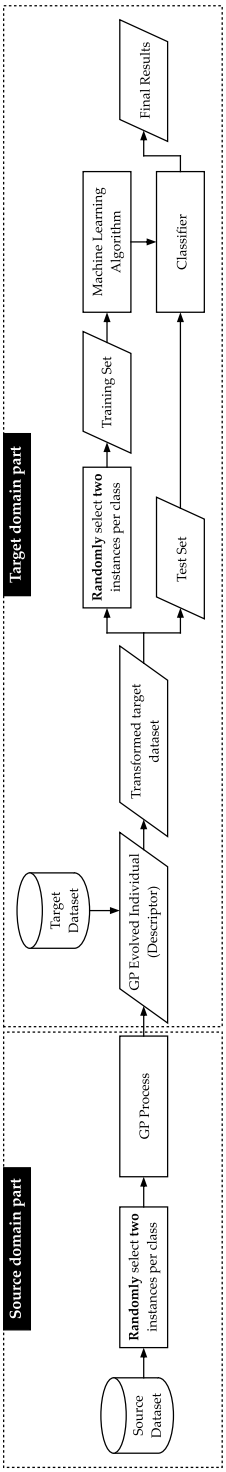


Figure 7.2: The detailed transfer learning used in this chapter.

target domain dataset is generated based on the GP-criptor$^{\text{ri}}$ process for feature vector extraction discussed in Section 5.2.3 (see page 149).

7.3 Experiment Design

This section aims at providing details of the experiments conducted in this chapter. The discussion of this section also includes the datasets, methods for comparison, parameter settings, and implementation.

7.3.1 Datasets

A core aim of this chapter is to investigate domain generalisation of GP-criptor$^{\text{ri}}$ evolved image descriptors (Chapter 5). Therefore, the source datasets are BrNoRo, BrWiRo, KyNoRo, KyWiRo, OutexTC00, and OutexTC10. To investigate the generalisability of those descriptors, datasets from similar, i.e., texture, and different domains are used. Therefore, the target domain datasets are the same six source datasets and the following seven benchmark datasets: Coins, Faces, KySinHw, Webcam, Amazon, Dslr, and CUReT. The details about each of these 13 datasets such as the number of classes, number of instances, rotations, illumination, and dimensions are discussed in Section 2.4 (see page 62).

7.3.2 Benchmark Methods for Comparison

The goodness of the image descriptors automatically evolved by GP-criptor$^{\text{ri}}$ is compared to that of six state-of-the-art domain-expert designed descriptors: $\text{LBP}^{u2}_{p,r}$, $\text{LBP}^{riu2}_{p,r}$, $\text{CLBP}_{p,r}$, $\text{LBC}_{p,r}$, $\text{CLBC}_{p,r}$, and $\text{DRLBP}_{p,r}$. Similar to GP-criptor, GP-criptor$^{\text{ri}}$, and EID$^{\text{ri}}$, the average accuracy of a classifier is used to assess the goodness of each of those descriptors (evolve by GP-criptor$^{\text{ri}}$ and the six baseline descriptors). Only the k-NN classification algorithm is used in this chapter.

7.3.3 Experiments

In order to investigate different aspects of transfer learning in GP, the experiments are designed to investigate different aspects. In total there are six experiments that

have been conducted which can be categorised into two groups: identical source and target datasets, and different source and target datasets. In each experiment, the settings are kept fixed such that after transforming the instances of the target dataset the system splits the transformed dataset into a training set and a test set. Two instances from each class are randomly selected to form the training set, whereas the remainder of the instances are used to form the test set. A classifier is then trained using the training instances, evaluated on the unseen (test) set, and the performance (accuracy) is reported. As the instances of the training set were randomly chosen, the system is set to further repeat the previous steps 10 times. At the end of the 10 repetitions, the mean and standard deviation of the accuracies are reported.

7.3.3.1 Identical source and target datasets

The experiments of this group aim to examining different aspects of the evolved descriptors on the same datasets that they were evolved on, i.e., the source dataset is identical to the target dataset. The core aim of these experiments is to provide baseline results for the subsequent experiments.

The first experiment investigates the performance of an evolved image descriptor on the same dataset, i.e., standard supervised learning where the source and target datasets are the same. Therefore, an image descriptor evolved on a dataset is used to transform the instances of that dataset, and two instances are then randomly selected and used for training with k-NN and the reminder of the instances are used for testing.

In the first experiment, the window size of the evolved descriptor is kept fixed. For example, an individual evolved using a window of size 3×3 pixels is also used on the target dataset with the exact same window size. Hence, the aim of the second experiment is to investigate the effect of using different window sizes on the same dataset, i.e., the source and target datasets are the same. Although it is not limited to any window size, only three window size have been used in this study: 3×3 pixels, 5×5 pixels, and 7×7 pixels. This experiment will allow investigating whether an evolved descriptor can, or is best to, be used with the same window size that was used during the evolutionary process or a comparable performance can be achieved using other window sizes.

The third experiment investigates the impact of using different code lengths, i.e., different feature vector length, which was kept fixed in the previous two experiments. Investigating this factor will allow the examination of the correlation between the number of features in the feature vector and the total number of classes in the dataset. Moreover, the experiment will examine the impact of having different numbers of classes in the source and target datasets on the performance.

7.3.3.2 Different source and target datasets

The second group of experiments aims at investigating the performance when using different datasets in source and target domains, which is a core aim of this study.

The fourth experiment in this study investigates the effectiveness of evolving an image descriptor on one dataset and using it on different datasets. Unlike the first experiment experiment, the source and target datasets in the fourth experiment are different. It is worth mentioning that the two datasets can be from the same domain, e.g., texture to texture, or different domains, e.g., texture to object.

Similar to the second experiment, the fifth experiment is designed to investigate the impact of using different window sizes between the source and target datasets; however, the source and target datasets are different. Moreover, this experiment will allow investigating the correlation between the window size and the size of the instances in both domains which was not possible in the second experiment as the source and target domain are identical (the instances have the same dimensions).

The sixth experiment is also designed to investigate the impact of using different code lengths between the source and target domains on the performance similar to the third experiment of the first group.

7.3.4 Parameter Settings

The parameters of GP-criptor$^{\text{ri}}$ are kept identical to that presented and discussed in Section 5.3.3 (see page 152), mainly because here the focus is on the transferability of GP-criptor$^{\text{ri}}$ evolved descriptors to tackle problems in related and unrelated domains. Similarly, parameter settings of the baseline methods are also kept identical to that were used in Chapter 5 (see page 145).

As mentioned earlier in this chapter, only the k-NN classification algorithm is

used to measure the goodness of the features extracted by an image descriptor to classify the instances of each dataset. As there are only two instances of each class in the training set, the parameter k is set to 1, i.e., the class label of the closest neighbour/instance is returned.

7.4 Results and Discussions

The results of the six different experiments are presented and discussed in this section. To prevent redundancy, the results of the experiments are combined and presented in tables for each dataset in the subsequent subsections. However, the performance of the baseline descriptors is examined first as it provides the basis for comparison; followed by the results and discussions on the automatically evolved GP-criptor$^{\text{ri}}$ descriptors.

7.4.1 The Baseline Descriptors

Table 7.1 presents the results of applying the k-NN classification algorithm on the 13 image datasets using the features of six state-of-the-art image descriptors. These descriptors were designed by domain-experts and have been used to perform different tasks, e.g., classification, and clustering, in a wide range of computer vision applications. The process to measure the goodness of each image descriptor in Table 7.1 comprises three steps: firstly, the instances of the dataset being evaluated are transformed using one of the six image descriptors; secondly, two instances are randomly selected from the dataset to form the training set and the remaining instances are used to form the test set; and thirdly, k-NN is applied to classify the instances of the test set and the accuracy is reported. As the two instances for training are randomly drawn, these three steps are further repeated 10 more times using different instances for training each time. Therefore, each pair of values in Table 7.1 represent the mean and the standard deviation ($\bar{x} \pm s$) of the accuracy over 10 repetitions.

The image descriptor with the best average performance on each dataset is made **bold** in Table 7.1; and if two or more image descriptors have the same best average performance, the one with the smallest standard deviation is made **bold**.

Table 7.1: The average accuracy (%) of k-NN using six image descriptors on 13 images dataset ($\bar{x} \pm s$).

	$LBP^{u2}_{8,1}$	$LBP^{u2ri}_{24,3}$	$CLBP_{24,3}$	$LBC_{24,3}$	$CLBC_{24,3}$	$DRLBP_{8,1}$
BrNoRo	86.48 ± 1.31	70.52 ± 3.01	**87.38 ± 2.57**	67.91 ± 3.67	68.83 ± 2.70	85.01 ± 1.91
BrWiRo	47.42 ± 1.54	69.88 ± 2.66	**85.06 ± 3.23**	66.63 ± 2.88	70.07 ± 4.19	70.30 ± 2.29
KyNoRo	82.25 ± 2.01	73.99 ± 2.15	**93.05 ± 1.77**	73.91 ± 1.91	81.05 ± 2.84	88.56 ± 1.87
KyWiRo	46.24 ± 2.17	71.75 ± 2.80	**91.95 ± 1.20**	71.16 ± 2.66	78.93 ± 2.20	77.21 ± 1.83
KySinHw	58.18 ± 1.94	81.34 ± 1.40	**96.63 ± 0.73**	79.58 ± 1.73	87.79 ± 2.23	85.33 ± 2.73
OutexTC00	**94.77 ± 1.95**	83.89 ± 3.22	93.17 ± 1.19	82.80 ± 2.39	82.55 ± 3.76	93.45 ± 1.37
OutexTC10	65.64 ± 1.71	74.17 ± 2.53	**88.05 ± 1.78**	71.39 ± 2.54	76.15 ± 2.98	74.97 ± 2.14
CUReT	51.09 ± 1.14	45.91 ± 2.33	**55.42 ± 2.40**	44.45 ± 2.42	46.29 ± 1.94	54.71 ± 1.81
Coins	62.50 ± 7.28	63.68 ± 10.5	**65.37 ± 9.50**	64.39 ± 10.1	61.74 ± 11.2	61.79 ± 5.98
Faces	**75.79 ± 9.54**	73.29 ± 8.57	71.43 ± 9.84	69.88 ± 7.08	64.92 ± 4.66	70.01 ± 7.72
Dslr	40.76 ± 3.16	36.40 ± 4.23	42.75 ± 3.45	34.77 ± 4.69	**43.74 ± 3.38**	39.08 ± 3.98
Amazon	18.50 ± 1.63	12.66 ± 1.59	**18.81 ± 1.30**	12.41 ± 1.72	18.53 ± 1.24	15.12 ± 1.26
Webcam	34.69 ± 3.50	29.67 ± 3.15	37.44 ± 3.21	27.39 ± 3.04	**37.87 ± 2.69**	31.61 ± 2.90

7.4.2 The GP-criptor^{ri} Descriptors

The performance of using those image descriptors automatically evolved by GP-criptor^{ri} using six source datasets (Chapter 5) on each of the 13 experimented target datasets is presented in this subsection. For each target dataset, the results are presented in a table and a boxplot. Each table horizontally consists of 9 columns, where the length of the code, i.e., number of children under the *code* node, is presented in the first column, the second and third columns show the source and target window sizes, and the remaining six columns list the average performance resulting from using the descriptors evolved on each of the six source datasets. Meanwhile, each table vertically consists of three blocks each of which comprises of three sub blocks that presents the results of using different source and target window sizes combinations. It is important to notice that in Chapter 5, on each dataset, GP-criptor^{ri} has produced 300 image descriptors that resulting from using 30 seeds and 10 repetitions (using different training instances). Therefore, each pair of values in the following tables is the average accuracy and the standard deviation of using k-NN on the features of 300 image descriptors. The boxplots, on the other hand, are provided to ease comparing the performance achieved by the programs evolved on each source dataset on the target dataset being evaluated. Each figure

comprises three boxplots each of which corresponds to using a different code length (7-, 8-, and 9-bits), whilst the different colours represent the three experimented window sizes (3×3 pixels, 5×5 pixels, and 7×7 pixels). Unlike the tables, the boxplots only show the cases where the source window and the target window are of the same size. The source datasets are listed on the horizontal axis and the vertical axis show the accuracy (%).

7.4.2.1 The BrNoRo Dataset

The results of using the descriptors evolved by GP-criptor$^{\text{ri}}$ on different source datasets to classify the BrNoRo dataset are presented in Table 7.2 and Figure 7.3. The results in Table 7.1 show that $\text{CLBP}_{24,3}$ has achieved the best performance amongst the baseline descriptors with 87.38% average accuracy. The GP-criptor$^{\text{ri}}$ evolved descriptors, on the other hand, show comparable or better performance than the best of the baseline descriptors. Table 7.2 shows that over 57% (93 out of 162) of the cases have achieved a better performance than the best of the baseline descriptors. Noticeably, some programs that were evolved on other source datasets have achieved better performance than the baseline methods. Clearly, the programs evolved on BrNoRo (source is the same as the target) show the best performance amongst all other programs as depicted in Figure 7.3. Similarly, those programs that were evolved on the rotation version of the BrNoRo have also achieved a very comparable results to that of the non-rotated version. The programs of all other source datasets also show a good level of performance where the median is greater than or equal to 80% accuracy. Generally, using a 7×7 pixels window size shows worse performance than using the other two window sizes. The code length, on the other hand, does not show large impact on the performance, and increasing the length has slightly improved the average accuracies of the automatically evolved descriptors.

7.4.2.2 The BrWiRo Dataset

Table 7.3 and Figure 7.4 present the results of using GP-criptor$^{\text{ri}}$ programs evolved on different source datasets to classify the BrWiRo dataset. Similar to BrNoRo, the programs evolved on this dataset, i.e., BrWiRo, show very good performance that

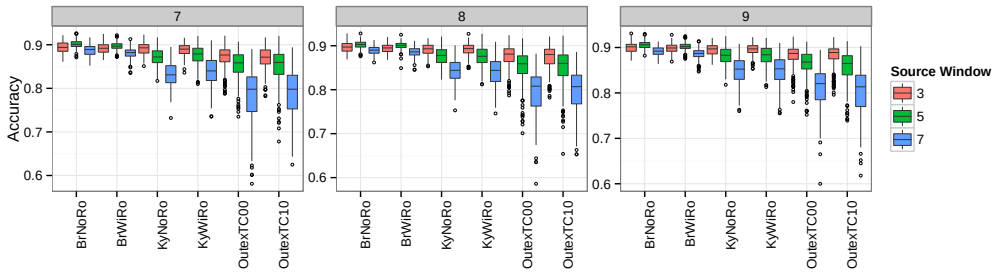


Figure 7.3: The performance of using individuals evolved on different datasets, with different code lengths on the **BrNoRo** dataset using the same source and target window sizes.

in most cases is above 80% accuracy as shown in Figure 7.4. The baseline methods have achieved on average 85.06% accuracy in the best case ($CLBP_{24,3}$), and the second best is 70.30% ($DRLBP_{8,1}$). The results in Table 7.3 show that over 70% (114 out of 162) of the cases have achieved a better performance than the best of the baseline descriptors. Furthermore, over 78% (89 out of 114) of those cases are from programs that were not evolved on the BrWiRo. More importantly is that the programs evolved on the unrotated version (BrNoRo) show comparable or better performance to the programs evolved on the rotated version. This supports the observation that those descriptors are rotation-invariant as has been investigated in Chapter 5. Similar to BrNoRo, using a 7×7 pixels window size show worse performance than using 3×3 pixels and 5×5 pixels window sizes. Moreover, the impact of the code length on the performance is minimal and increasing the length has slightly improved the average accuracy.

7.4.2.3 The KyNoRo Dataset

The results in Table 7.4 and Figure 7.5 show the performance for classifying the KyNoRo dataset using the programs evolved from the different source datasets. Following the same pattern to the previous datasets, the programs evolved on this dataset (KyNoRo) show better trend to classify the instances of the dataset compared to other source datasets. Although the average performance of the GP-criptor^{ri} evolved programs is not better than the best of the baseline methods

Table 7.2: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **BrNoRo** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	**89.4 ± 1.16**	89.1 ± 1.10	89.1 ± 1.36	88.8 ± 1.35	87.2 ± 2.49	86.9 ± 2.46
		5	**87.4 ± 2.09**	87.3 ± 1.81	87.1 ± 2.33	86.6 ± 2.49	81.8 ± 4.91	81.5 ± 5.04
		7	83.5 ± 3.02	**83.5 ± 2.65**	82.7 ± 3.56	81.9 ± 4.26	73.4 ± 7.37	73.2 ± 7.66
	5	3	**89.8 ± 0.97**	89.5 ± 0.92	88.7 ± 1.27	89.0 ± 1.33	88.7 ± 1.60	88.7 ± 1.83
		5	**90.1 ± 0.90**	89.7 ± 0.80	87.2 ± 1.89	87.6 ± 2.19	85.4 ± 3.08	85.1 ± 3.73
		7	**88.0 ± 1.17**	87.7 ± 1.20	84.1 ± 2.78	84.1 ± 3.38	78.3 ± 5.38	77.8 ± 6.12
	7	3	**89.4 ± 1.30**	88.8 ± 1.39	88.0 ± 1.41	88.4 ± 1.44	88.9 ± 1.51	89.1 ± 1.38
		5	**90.1 ± 1.00**	89.5 ± 0.98	86.5 ± 1.80	87.1 ± 2.09	85.5 ± 3.27	85.7 ± 2.87
		7	**88.7 ± 1.21**	88.0 ± 1.17	83.2 ± 2.64	84.0 ± 3.15	78.5 ± 5.50	79.0 ± 5.10
8	3	3	**89.6 ± 1.14**	89.4 ± 0.97	89.2 ± 1.31	89.1 ± 1.37	87.7 ± 2.39	87.4 ± 2.53
		5	**87.8 ± 2.13**	87.7 ± 1.86	87.2 ± 2.07	87.0 ± 2.44	82.5 ± 4.62	82.0 ± 5.08
		7	84.1 ± 3.14	**84.2 ± 2.85**	83.1 ± 3.33	82.5 ± 4.07	74.4 ± 7.05	73.8 ± 7.80
	5	3	**89.9 ± 0.83**	89.7 ± 0.86	89.1 ± 1.09	89.1 ± 1.23	89.0 ± 1.73	88.9 ± 1.70
		5	**90.2 ± 0.85**	90.0 ± 0.80	87.7 ± 1.88	87.5 ± 2.19	85.2 ± 3.59	85.2 ± 3.75
		7	**88.3 ± 1.16**	88.1 ± 1.15	84.4 ± 3.00	83.9 ± 3.35	78.3 ± 5.91	78.2 ± 6.17
	7	3	**89.5 ± 1.04**	89.1 ± 1.28	88.5 ± 1.32	88.8 ± 1.33	89.2 ± 1.47	89.3 ± 1.46
		5	**90.2 ± 0.74**	89.8 ± 0.87	87.3 ± 1.70	87.3 ± 2.02	86.0 ± 3.04	86.2 ± 2.92
		7	**89.0 ± 0.89**	88.6 ± 1.03	84.3 ± 2.55	84.1 ± 3.04	79.3 ± 5.01	79.6 ± 4.91
9	3	3	**90.0 ± 1.10**	89.8 ± 1.00	89.5 ± 1.30	89.6 ± 1.28	88.2 ± 2.23	88.3 ± 2.25
		5	**88.3 ± 1.95**	88.2 ± 1.66	87.5 ± 2.24	87.9 ± 2.12	83.4 ± 4.44	83.5 ± 4.09
		7	84.8 ± 2.86	**85.1 ± 2.38**	83.5 ± 3.53	84.1 ± 3.47	75.5 ± 7.06	75.9 ± 6.38
	5	3	**90.3 ± 0.86**	90.0 ± 0.97	89.5 ± 1.19	89.6 ± 1.23	89.5 ± 1.50	89.4 ± 1.45
		5	**90.6 ± 0.80**	90.2 ± 0.81	88.2 ± 1.90	88.1 ± 2.20	86.4 ± 2.94	85.8 ± 3.40
		7	**88.5 ± 1.07**	88.3 ± 1.08	85.2 ± 2.98	84.6 ± 3.37	80.0 ± 5.19	78.9 ± 5.65
	7	3	**89.8 ± 1.08**	89.3 ± 1.25	88.9 ± 1.38	89.1 ± 1.39	89.7 ± 1.35	89.7 ± 1.30
		5	**90.4 ± 0.87**	89.9 ± 0.89	87.8 ± 1.76	87.8 ± 2.20	87.1 ± 2.55	86.7 ± 2.75
		7	**89.1 ± 1.01**	88.6 ± 1.05	85.0 ± 2.79	84.9 ± 3.26	81.1 ± 4.39	80.3 ± 4.90

($\text{CLBP}_{42,3}$ with 93.05% accuracy), over 48% (78 out of 162) of the cases show over 84% accuracy on average. Generally, the window size has a small impact between programs evolved on the same source dataset; however, smaller window sizes show better performance than using larger ones. Using a larger feature vector (code length) has positive influence on the average performance as presented in

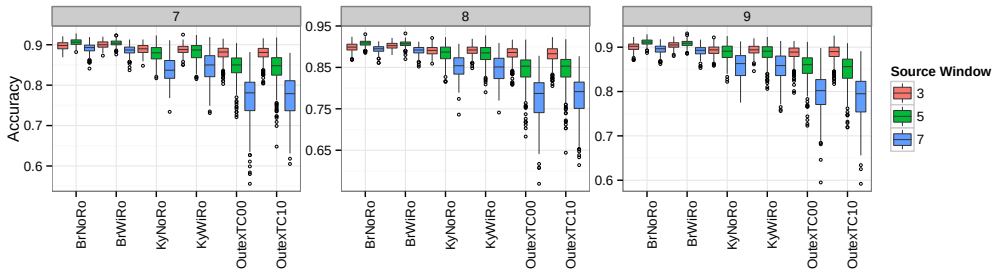


Figure 7.4: The performance of using individuals evolved on different datasets, with different code lengths on the **BrWiRo** dataset using the same source and target window sizes.

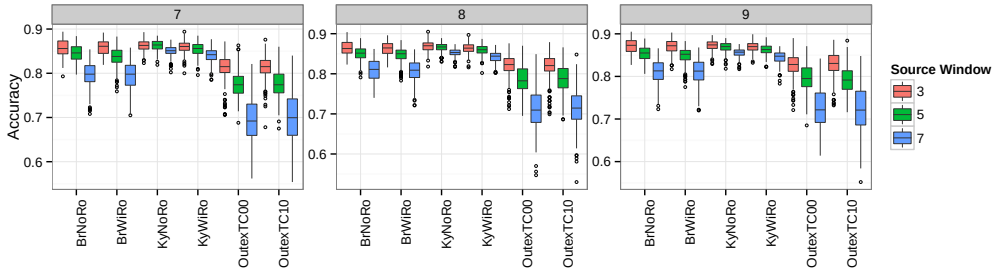


Figure 7.5: The performance of using individuals evolved on different datasets, with different code lengths on the **KyNoRo** dataset using the same source and target window sizes.

Table 7.4. The programs evolved on OutexTC00 and OutexTC10 datasets show worse performances compared to other datasets.

7.4.2.4 The KyWiRo Dataset

Table 7.5 presents the results obtained on the KyWiRo dataset using the GP-criptor^{ri} programs that were evolved on the six source datasets. Compared to the baseline methods, the results show that the achieved performance is not as good as the best of the baseline method $CLBP_{24,3}$ with 91.95% average accuracy. However, over 87% (141 out of 162) of the situations have achieved over 80% accuracy on average, and over 58% (83) of those 141 situation are above 85% accuracy. Apart

Table 7.3: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **BrWiRo** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	89.8 ± 1.03	**90.0 ± 0.89**	88.9 ± 1.16	88.8 ± 1.16	87.8 ± 2.05	87.7 ± 2.03
		5	88.1 ± 2.05	**88.2 ± 1.77**	87.8 ± 2.21	87.3 ± 2.44	81.1 ± 5.24	80.9 ± 5.45
		7	83.7 ± 3.45	**84.1 ± 3.07**	83.3 ± 3.70	82.0 ± 4.53	71.8 ± 7.62	71.5 ± 7.79
	5	3	89.9 ± 0.95	**89.5 ± 0.97**	88.3 ± 1.37	88.6 ± 1.30	88.9 ± 1.32	88.8 ± 1.58
		5	**90.7 ± 0.78**	90.4 ± 0.73	87.9 ± 2.01	88.2 ± 2.10	84.5 ± 3.21	84.2 ± 3.87
		7	**88.7 ± 1.05**	88.5 ± 1.11	84.5 ± 3.02	84.4 ± 3.38	76.6 ± 5.54	75.9 ± 6.36
	7	3	88.8 ± 1.51	88.1 ± 1.71	87.8 ± 1.53	88.2 ± 1.40	89.0 ± 1.34	**89.1 ± 1.17**
		5	**90.5 ± 1.07**	89.9 ± 1.11	87.1 ± 2.17	87.9 ± 2.26	84.7 ± 3.31	85.0 ± 3.00
		7	**89.1 ± 1.19**	88.6 ± 1.18	83.8 ± 2.97	84.6 ± 3.28	76.8 ± 5.58	77.3 ± 5.33
8	3	3	89.8 ± 1.04	**90.2 ± 0.78**	89.0 ± 1.11	89.1 ± 1.19	88.2 ± 1.98	87.9 ± 2.11
		5	88.2 ± 2.28	**88.5 ± 1.84**	88.0 ± 2.02	87.7 ± 2.36	81.7 ± 4.96	81.2 ± 5.46
		7	84.4 ± 3.62	**84.9 ± 3.10**	83.6 ± 3.56	82.8 ± 4.18	72.7 ± 7.20	72.1 ± 7.93
	5	3	**90.0 ± 0.79**	89.7 ± 0.96	88.7 ± 1.19	88.8 ± 1.26	89.1 ± 1.45	88.9 ± 1.46
		5	**90.8 ± 0.74**	90.7 ± 0.70	88.3 ± 2.01	88.1 ± 2.31	84.4 ± 3.74	84.3 ± 3.95
		7	**89.0 ± 1.02**	88.9 ± 0.96	85.1 ± 3.14	84.2 ± 3.57	76.7 ± 6.23	76.6 ± 6.44
	7	3	89.1 ± 1.40	88.6 ± 1.51	88.3 ± 1.35	88.5 ± 1.30	89.2 ± 1.37	**89.2 ± 1.25**
		5	**90.7 ± 0.77**	90.3 ± 0.92	88.0 ± 1.89	88.1 ± 2.12	85.2 ± 3.17	85.3 ± 2.98
		7	**89.4 ± 0.85**	89.1 ± 0.98	85.2 ± 2.76	84.7 ± 3.18	77.6 ± 5.28	77.9 ± 5.09
9	3	3	90.1 ± 0.98	**90.5 ± 0.74**	89.3 ± 1.13	89.4 ± 1.10	88.5 ± 1.81	88.7 ± 1.85
		5	88.8 ± 1.96	**89.1 ± 1.55**	88.2 ± 2.11	88.5 ± 1.99	82.4 ± 4.73	82.7 ± 4.45
		7	85.2 ± 3.30	**85.9 ± 2.54**	84.1 ± 3.66	84.3 ± 3.63	73.8 ± 7.12	74.1 ± 6.56
	5	3	**90.2 ± 0.83**	89.9 ± 1.01	89.0 ± 1.23	89.1 ± 1.24	89.4 ± 1.26	89.4 ± 1.30
		5	**91.1 ± 0.69**	90.8 ± 0.74	88.8 ± 1.89	88.6 ± 2.17	85.5 ± 3.16	84.8 ± 3.47
		7	**89.2 ± 0.96**	89.0 ± 0.97	85.8 ± 3.05	85.0 ± 3.47	78.3 ± 5.43	77.2 ± 5.78
	7	3	89.2 ± 1.50	88.7 ± 1.67	88.6 ± 1.43	88.8 ± 1.32	89.4 ± 1.17	**89.5 ± 1.07**
		5	**90.8 ± 0.93**	90.4 ± 0.96	88.6 ± 1.84	88.5 ± 2.23	86.3 ± 2.52	86.0 ± 2.75
		7	**89.5 ± 1.03**	89.2 ± 1.04	85.9 ± 2.88	85.5 ± 3.26	79.5 ± 4.57	78.6 ± 5.14

from OutexTC00 and OutexTC10 in some cases, the majority of the programs evolved on other than KyWiRo have achieved a reasonably good performance. Similar to BrWiRo, the programs evolved on the unrotated version of Kylberg have achieved similar or better performance even though they have not been evolved on any rotated instances as presented in Figure 7.6. The window size has a negative

Table 7.4: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **KyNoRo** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	85.7 ± 1.81	85.8 ± 1.66	**86.2 ± 1.20**	85.9 ± 1.37	81.3 ± 2.80	81.4 ± 2.51
		5	84.0 ± 2.17	**84.6 ± 1.97**	83.2 ± 2.18	82.7 ± 2.57	74.3 ± 4.34	74.2 ± 4.15
		7	79.1 ± 3.86	**80.9 ± 2.87**	78.3 ± 3.26	77.8 ± 4.02	64.5 ± 6.83	64.3 ± 6.67
	5	3	85.9 ± 1.65	85.2 ± 1.70	86.8 ± 1.20	**86.6 ± 1.18**	83.5 ± 1.88	83.5 ± 1.96
		5	84.5 ± 1.88	83.7 ± 2.13	**86.3 ± 1.17**	85.4 ± 1.50	77.6 ± 3.12	77.6 ± 3.28
		7	81.4 ± 2.36	80.9 ± 2.83	**83.4 ± 1.81**	81.8 ± 2.39	68.6 ± 5.18	68.6 ± 5.58
	7	3	84.9 ± 2.01	83.9 ± 2.24	85.9 ± 1.50	**86.4 ± 1.35**	83.7 ± 1.72	84.1 ± 1.66
		5	83.0 ± 2.32	82.3 ± 2.65	**86.7 ± 0.97**	86.3 ± 1.12	78.0 ± 3.11	78.6 ± 3.24
		7	79.7 ± 2.88	79.4 ± 3.20	**85.0 ± 1.08**	84.0 ± 1.50	69.4 ± 5.18	70.0 ± 5.65
8	3	3	86.3 ± 1.66	86.3 ± 1.61	**86.9 ± 1.21**	86.3 ± 1.31	82.1 ± 2.69	81.8 ± 3.03
		5	84.6 ± 2.09	**85.2 ± 1.86**	84.2 ± 1.85	83.3 ± 2.16	75.4 ± 4.11	75.3 ± 4.49
		7	80.1 ± 3.44	**81.7 ± 2.83**	79.8 ± 2.81	78.6 ± 3.37	66.1 ± 6.33	66.2 ± 7.00
	5	3	86.3 ± 1.34	85.7 ± 1.52	87.2 ± 1.03	**87.1 ± 1.06**	84.2 ± 2.04	84.3 ± 1.82
		5	85.1 ± 1.56	84.7 ± 1.81	**86.6 ± 1.01**	85.9 ± 1.29	78.7 ± 3.41	78.8 ± 3.40
		7	82.0 ± 2.18	82.0 ± 2.35	**83.9 ± 1.48**	82.6 ± 2.02	69.9 ± 5.68	70.0 ± 5.72
	7	3	85.6 ± 1.63	84.8 ± 2.14	86.4 ± 1.36	**86.7 ± 1.28**	84.4 ± 1.79	84.7 ± 1.61
		5	84.0 ± 2.06	83.4 ± 2.32	87.0 ± 0.82	**86.4 ± 1.08**	79.3 ± 3.03	79.6 ± 2.85
		7	80.9 ± 2.57	80.6 ± 2.80	85.2 ± 0.93	**84.2 ± 1.35**	71.3 ± 4.98	71.5 ± 4.77
9	3	3	87.1 ± 1.62	87.0 ± 1.55	**87.3 ± 1.16**	86.9 ± 1.17	82.7 ± 2.56	82.9 ± 2.72
		5	85.5 ± 1.92	**85.6 ± 1.82**	84.5 ± 1.99	83.8 ± 2.20	76.5 ± 4.16	76.6 ± 3.94
		7	81.2 ± 3.11	**82.1 ± 2.66**	80.1 ± 3.07	79.1 ± 3.38	67.3 ± 6.65	67.7 ± 6.20
	5	3	86.9 ± 1.39	86.2 ± 1.58	**87.5 ± 1.16**	87.4 ± 1.08	84.8 ± 1.73	84.7 ± 1.70
		5	85.4 ± 1.65	84.8 ± 1.89	**86.9 ± 1.11**	86.3 ± 1.24	79.7 ± 3.09	79.1 ± 3.17
		7	82.3 ± 2.32	82.0 ± 2.54	**84.2 ± 1.61**	83.0 ± 1.93	71.7 ± 5.15	70.7 ± 5.35
	7	3	85.9 ± 1.71	85.1 ± 2.10	86.8 ± 1.45	**87.1 ± 1.34**	85.1 ± 1.60	85.2 ± 1.56
		5	84.2 ± 2.14	83.6 ± 2.51	**87.3 ± 0.87**	86.8 ± 1.12	80.3 ± 2.92	80.3 ± 3.13
		7	81.1 ± 2.66	80.9 ± 3.01	**85.5 ± 0.93**	84.5 ± 1.47	72.6 ± 4.75	72.4 ± 5.22

influence on the performance as increasing the window size, generally, degraded the accuracy. The code length has a similar impact to the previous datasets on the performance.

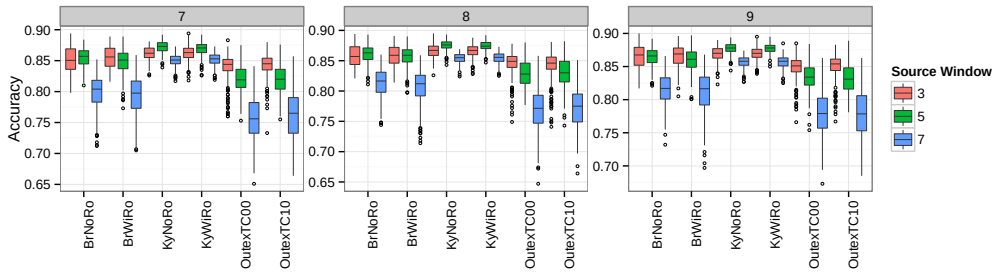


Figure 7.6: The performance of using individuals evolved on different datasets, with different code lengths on the **KyWiRo** dataset using the same source and target window sizes.

7.4.2.5 The KySinHw Dataset

The average classification accuracies of using the programs evolved on different source datasets to classify the KySinHw dataset are presented in Table 7.6 and Figure 7.7. As presented in Table 7.1, $\text{CLBP}_{24,3}$ has scored the best performance with 96.63% accuracy on average. The programs evolved on different source datasets have achieved a very good level of accuracy on this dataset, where over 86% (140 out of 162) of the cases show average accuracy higher than 90%. It important to notice that this dataset (KySinHw) is not one of the source datasets; which means, all the descriptors have evolved on datasets that are completely different from the target dataset. Figure 7.7 shows that the median performance of those automatically evolved image descriptors, apart from OutexTC00 and OutexTC10 with a 3×3 pixels windows size, is above 90%. Increasing the code length has positive impact on the performance of the evolved programs. Furthermore, increasing the window size from 3×3 pixels to 5 pixels shows a positive influence on the performance, whereas increasing the window size from 5×5 pixels to 7×7 pixels have slightly deteriorated the performance.

7.4.2.6 The OutexTC00 Dataset

Table 7.7 and Figure 7.8 present the results obtained from using the descriptors evolved on the six source datasets to classify the OutexTC00 dataset. The $\text{LBP}^{u2}_{8,1}$

Table 7.5: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **KyWiRo** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	85.2 ± 1.86	85.5 ± 1.72	**86.2 ± 1.11**	86.2 ± 1.14	84.0 ± 2.00	84.3 ± 1.72
		5	85.6 ± 1.74	**86.1 ± 1.36**	85.7 ± 1.45	85.5 ± 1.72	79.4 ± 3.29	79.6 ± 2.93
		7	81.2 ± 3.08	**82.5 ± 2.19**	81.8 ± 2.29	81.6 ± 2.63	71.2 ± 5.48	71.4 ± 5.02
	5	3	85.3 ± 1.93	84.6 ± 1.99	86.1 ± 1.20	**86.3 ± 1.02**	85.4 ± 1.15	85.3 ± 1.25
		5	85.5 ± 1.52	84.8 ± 2.00	**87.2 ± 0.94**	86.9 ± 1.05	82.1 ± 2.25	82.0 ± 2.34
		7	82.4 ± 1.87	81.8 ± 2.53	**84.6 ± 1.32**	84.0 ± 1.60	75.1 ± 3.68	75.0 ± 3.85
	7	3	84.2 ± 2.29	83.2 ± 2.70	84.9 ± 1.68	**85.9 ± 1.41**	85.3 ± 1.06	85.4 ± 1.07
		5	83.4 ± 2.26	82.7 ± 2.70	86.9 ± 1.01	**87.3 ± 0.87**	82.4 ± 2.24	82.7 ± 2.32
		7	80.0 ± 2.66	79.3 ± 3.09	85.0 ± 0.93	**85.2 ± 1.00**	75.7 ± 3.51	76.2 ± 3.77
8	3	3	85.8 ± 1.70	85.8 ± 1.65	86.6 ± 1.13	**86.6 ± 1.04**	84.5 ± 1.90	84.3 ± 2.20
		5	86.1 ± 1.64	**86.6 ± 1.35**	86.5 ± 1.25	86.0 ± 1.43	80.4 ± 2.83	80.4 ± 3.19
		7	82.2 ± 2.68	**83.1 ± 2.07**	82.9 ± 1.89	82.3 ± 2.14	72.7 ± 4.99	72.8 ± 5.22
	5	3	85.6 ± 1.63	85.1 ± 1.76	86.4 ± 1.10	**86.6 ± 0.92**	85.7 ± 1.21	85.7 ± 1.10
		5	86.1 ± 1.42	85.6 ± 1.62	**87.6 ± 0.80**	87.5 ± 0.81	82.9 ± 2.29	83.0 ± 2.37
		7	83.0 ± 1.82	82.7 ± 1.89	**85.1 ± 1.00**	84.7 ± 1.17	76.1 ± 3.97	76.3 ± 4.02
	7	3	84.8 ± 2.07	84.2 ± 2.51	85.4 ± 1.57	**86.0 ± 1.30**	85.7 ± 1.10	85.8 ± 1.03
		5	84.5 ± 2.02	83.9 ± 2.45	87.3 ± 0.87	**87.5 ± 0.83**	83.2 ± 2.22	83.5 ± 2.02
		7	81.3 ± 2.35	80.7 ± 2.70	**85.4 ± 0.77**	85.4 ± 0.94	77.1 ± 3.55	77.3 ± 3.25
9	3	3	86.5 ± 1.67	86.6 ± 1.58	86.9 ± 1.06	**87.0 ± 0.94**	84.9 ± 1.64	85.1 ± 1.76
		5	86.7 ± 1.42	**87.0 ± 1.29**	86.6 ± 1.26	86.3 ± 1.42	81.0 ± 2.92	81.3 ± 2.68
		7	82.8 ± 2.36	**83.5 ± 1.95**	83.0 ± 1.92	82.6 ± 2.28	73.7 ± 4.94	73.9 ± 4.42
	5	3	86.2 ± 1.62	85.6 ± 1.83	86.8 ± 1.17	**87.0 ± 1.00**	86.1 ± 1.05	86.0 ± 0.99
		5	86.4 ± 1.38	85.9 ± 1.71	**87.8 ± 0.78**	87.7 ± 0.75	83.4 ± 2.07	83.3 ± 2.11
		7	83.3 ± 1.84	82.9 ± 2.12	**85.3 ± 1.01**	85.1 ± 1.07	77.2 ± 3.48	76.7 ± 3.52
	7	3	85.1 ± 2.07	84.4 ± 2.45	85.9 ± 1.65	**86.4 ± 1.31**	86.1 ± 1.09	86.1 ± 0.97
		5	84.7 ± 2.10	84.3 ± 2.56	87.5 ± 0.88	**87.7 ± 0.88**	83.9 ± 2.02	84.0 ± 2.13
		7	81.5 ± 2.38	81.1 ± 3.02	**85.7 ± 0.82**	85.7 ± 0.93	78.0 ± 3.23	77.9 ± 3.44

descriptor with 94.77% average accuracy represents the best method amongst all other descriptors on this dataset. The vast majority of the descriptors evolved by GP-criptor$^{\text{ri}}$ have achieved over 80% accuracy and more than 36% (59 out of 162) of the cases have a higher performance than 85% accuracy as presented in Table 7.7. Figure 7.8 shows that some descriptors that were evolved on the

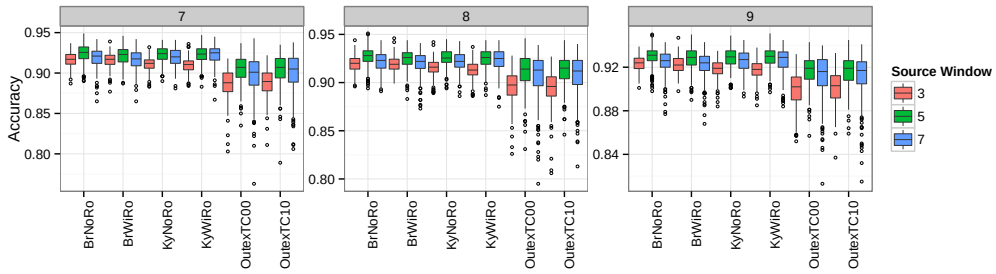


Figure 7.7: The performance of using individuals evolved on different datasets, with different code lengths on the **KySinHw** dataset using the same source and target window sizes.

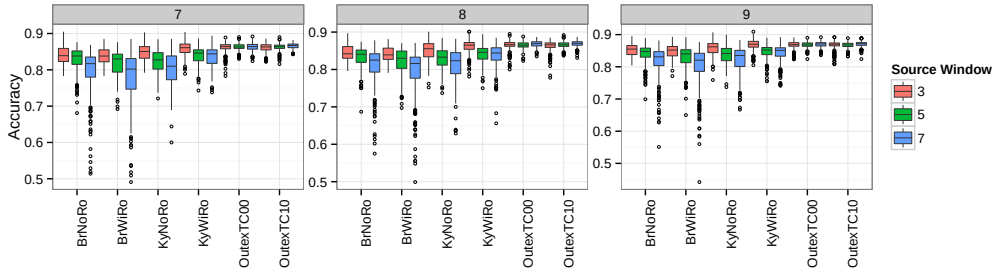


Figure 7.8: The performance of using individuals evolved on different datasets, with different code lengths on the **OutexTC00** dataset using the same source and target window sizes.

Brodatz (BrNoRo and BrWiRo) and Kylberg (KyNoRo and KyWiRo) datasets have achieved even higher (over 90%) accuracy than those descriptors evolved on OutexTC00. Generally, increasing the window size has negative impact on the performance of those descriptors; whilst increasing the code length shows a positive influence on the performance.

7.4.2.7 The OutexTC10 Dataset

On OutexTC10, the performances of using those automatically evolved descriptors are presented in Table 7.8 and Figure 7.9. On average, none of the experimented descriptors (baseline and GP-criptor^{ri} descriptors) have achieved higher perfor-

Table 7.6: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **KySinHw** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	**91.7 ± 0.88**	91.6 ± 0.84	91.1 ± 0.82	91.0 ± 0.87	88.8 ± 1.70	88.8 ± 1.68
		5	92.3 ± 1.18	**92.4 ± 1.13**	91.8 ± 1.08	91.6 ± 1.21	88.7 ± 2.75	88.7 ± 2.73
		7	91.5 ± 1.58	**92.0 ± 1.35**	91.3 ± 1.44	91.2 ± 1.68	86.4 ± 4.70	86.3 ± 4.65
	5	3	91.1 ± 1.17	90.8 ± 1.25	91.0 ± 1.01	**91.1 ± 0.96**	89.6 ± 1.33	89.6 ± 1.35
		5	**92.4 ± 1.12**	92.1 ± 1.14	92.3 ± 0.92	92.3 ± 1.09	90.5 ± 1.72	90.4 ± 1.87
		7	**92.4 ± 0.99**	92.2 ± 1.00	92.0 ± 1.05	91.8 ± 1.37	89.4 ± 2.47	89.3 ± 2.83
	7	3	90.1 ± 1.41	89.9 ± 1.60	90.2 ± 1.28	**90.8 ± 1.14**	89.7 ± 1.15	89.9 ± 1.09
		5	91.8 ± 1.29	91.5 ± 1.39	91.9 ± 1.09	**92.4 ± 1.07**	90.7 ± 1.56	90.9 ± 1.57
		7	91.9 ± 1.18	91.6 ± 1.21	92.0 ± 1.05	**92.2 ± 1.13**	89.7 ± 2.41	90.1 ± 2.35
8	3	3	**92.0 ± 0.82**	91.9 ± 0.77	91.6 ± 0.86	91.3 ± 0.89	89.6 ± 1.57	89.4 ± 1.67
		5	**92.6 ± 1.06**	92.7 ± 0.99	92.2 ± 1.05	91.9 ± 1.15	89.7 ± 2.33	89.5 ± 2.64
		7	**92.0 ± 1.47**	92.3 ± 1.31	91.8 ± 1.29	91.6 ± 1.53	87.8 ± 3.94	87.6 ± 4.41
	5	3	91.4 ± 0.97	91.2 ± 1.05	91.3 ± 0.86	**91.4 ± 0.85**	90.3 ± 1.34	90.3 ± 1.13
		5	**92.7 ± 0.93**	92.4 ± 1.00	92.5 ± 0.89	92.5 ± 0.96	91.2 ± 1.75	91.3 ± 1.51
		7	**92.7 ± 0.87**	92.6 ± 0.89	92.2 ± 1.03	92.1 ± 1.18	90.2 ± 2.56	90.4 ± 2.29
	7	3	90.7 ± 1.33	90.3 ± 1.50	90.6 ± 1.15	**91.0 ± 1.06**	90.4 ± 1.06	90.4 ± 1.04
		5	92.1 ± 1.18	91.9 ± 1.32	92.2 ± 1.01	**92.5 ± 0.99**	91.4 ± 1.49	91.4 ± 1.42
		7	92.1 ± 1.05	92.0 ± 1.14	92.2 ± 0.95	**92.3 ± 1.10**	90.7 ± 2.25	90.8 ± 2.01
9	3	3	**92.3 ± 0.72**	92.2 ± 0.73	91.9 ± 0.81	91.7 ± 0.82	90.0 ± 1.59	90.1 ± 1.53
		5	**93.1 ± 0.92**	93.1 ± 0.84	92.5 ± 1.09	92.3 ± 1.05	90.0 ± 2.32	90.3 ± 2.24
		7	**92.4 ± 1.36**	92.8 ± 1.00	92.0 ± 1.44	92.0 ± 1.41	88.3 ± 3.89	88.6 ± 3.62
	5	3	91.7 ± 1.03	91.5 ± 1.06	91.6 ± 0.92	91.7 ± 0.92	90.7 ± 1.17	90.6 ± 1.07
		5	93.0 ± 0.89	92.8 ± 1.00	92.8 ± 0.89	92.9 ± 0.92	91.6 ± 1.40	91.6 ± 1.41
		7	92.9 ± 0.80	92.8 ± 0.90	92.5 ± 0.97	92.6 ± 1.01	90.9 ± 1.90	90.8 ± 2.05
	7	3	90.9 ± 1.44	90.6 ± 1.51	91.0 ± 1.20	**91.3 ± 1.07**	90.7 ± 1.03	90.7 ± 0.97
		5	92.5 ± 1.17	92.2 ± 1.28	92.6 ± 1.01	**92.8 ± 1.03**	91.8 ± 1.38	91.8 ± 1.34
		7	92.5 ± 1.03	92.2 ± 1.15	92.6 ± 0.96	**92.6 ± 1.11**	91.2 ± 1.97	91.3 ± 1.94

mance than the 88.05% that was scored by $\text{CLBP}_{24,3}$. However, all the cases presented in Table 7.8 show that the automatically evolved programs on average have achieved better performance than the second best domain-expert designed descriptor. Moreover, over 54% (88 out of 162) of the cases have achieved over 84% accuracy on average. Figure 7.9 shows that OutexTC10 is following the same

Table 7.7: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **OutexTC00** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	84.0 ± 2.18	83.8 ± 2.06	84.8 ± 2.17	85.7 ± 1.85	**86.2 ± 1.36**	86.1 ± 1.07
		5	83.5 ± 2.33	83.6 ± 2.09	83.9 ± 2.01	84.6 ± 1.82	84.8 ± 1.58	**84.9 ± 1.31**
		7	83.2 ± 2.94	83.6 ± 2.62	83.9 ± 2.24	**84.4 ± 2.14**	83.7 ± 2.18	84.0 ± 1.98
	5	3	83.0 ± 3.64	82.2 ± 3.60	82.7 ± 4.13	85.1 ± 3.27	**87.1 ± 0.95**	87.1 ± 1.11
		5	82.9 ± 3.17	81.9 ± 3.40	82.2 ± 3.06	83.8 ± 2.51	**86.3 ± 0.80**	86.3 ± 0.87
		7	82.8 ± 3.65	81.5 ± 4.05	82.8 ± 2.92	84.1 ± 2.35	85.9 ± 0.98	**86.0 ± 0.92**
	7	3	81.4 ± 5.23	79.9 ± 6.63	78.8 ± 6.29	82.7 ± 4.79	87.0 ± 1.12	**87.2 ± 1.09**
		5	80.8 ± 5.44	79.4 ± 6.18	79.6 ± 4.46	82.5 ± 3.35	**86.4 ± 0.84**	**86.6 ± 0.66**
		7	79.6 ± 6.03	78.1 ± 6.86	80.2 ± 4.62	83.5 ± 2.76	86.3 ± 1.02	**86.6 ± 0.80**
8	3	3	84.5 ± 1.98	84.0 ± 1.90	85.1 ± 2.42	86.1 ± 1.77	**86.5 ± 1.22**	86.3 ± 1.39
		5	83.9 ± 2.00	83.9 ± 1.88	84.2 ± 2.09	85.0 ± 1.66	**85.2 ± 1.50**	85.0 ± 1.73
		7	84.1 ± 2.58	84.1 ± 2.39	84.3 ± 2.05	**84.9 ± 1.95**	84.1 ± 2.16	84.0 ± 2.39
	5	3	83.4 ± 3.11	82.3 ± 3.79	83.4 ± 3.72	85.0 ± 3.12	87.3 ± 1.01	**87.4 ± 0.94**
		5	83.3 ± 2.70	82.3 ± 3.31	82.8 ± 2.74	84.1 ± 2.27	86.5 ± 0.86	**86.6 ± 0.84**
		7	83.4 ± 2.99	82.1 ± 3.83	83.6 ± 2.55	84.6 ± 1.85	86.2 ± 1.06	**86.2 ± 0.95**
	7	3	81.9 ± 4.56	80.6 ± 5.82	80.1 ± 5.59	82.7 ± 4.51	87.2 ± 1.03	**87.4 ± 0.98**
		5	81.6 ± 4.48	80.3 ± 5.56	80.2 ± 4.15	82.6 ± 3.33	86.7 ± 0.73	**86.7 ± 0.70**
		7	81.1 ± 4.81	79.4 ± 6.17	81.2 ± 4.30	83.7 ± 3.06	86.8 ± 0.92	**86.8 ± 0.86**
9	3	3	85.2 ± 1.83	84.8 ± 1.92	85.7 ± 2.21	86.8 ± 1.63	**86.8 ± 1.08**	86.7 ± 1.31
		5	84.6 ± 1.79	84.5 ± 1.77	84.5 ± 1.94	85.5 ± 1.51	**85.6 ± 1.23**	85.6 ± 1.36
		7	84.5 ± 2.42	84.6 ± 2.41	84.7 ± 2.11	85.3 ± 1.68	84.7 ± 1.69	**84.8 ± 1.80**
	5	3	84.2 ± 3.17	83.3 ± 3.50	84.4 ± 3.66	85.8 ± 2.89	**87.6 ± 0.90**	87.6 ± 0.95
		5	83.9 ± 2.78	83.0 ± 3.35	83.7 ± 2.77	84.6 ± 2.23	86.8 ± 0.83	**86.8 ± 0.82**
		7	83.9 ± 3.08	82.7 ± 3.87	84.2 ± 2.54	85.2 ± 1.92	86.5 ± 0.92	**86.5 ± 0.91**
	7	3	82.8 ± 5.01	81.3 ± 6.12	81.1 ± 5.86	83.7 ± 4.59	**87.7 ± 0.90**	87.7 ± 0.98
		5	82.4 ± 4.43	81.0 ± 5.48	81.3 ± 4.21	83.3 ± 3.18	**87.0 ± 0.76**	**87.0 ± 0.76**
		7	81.7 ± 4.72	80.2 ± 5.95	82.1 ± 4.16	84.3 ± 2.66	87.0 ± 0.86	**87.0 ± 0.80**

pattern of the unrotated version of this dataset, i.e., OutexTC00, of the window size and code length impacts on the performance. Noticeably, over 59% (16 out of 27) of the OutexTC00 cases (the unrotated version) have similar or better performance than the programs evolved on OutexTC10 as presented in Table 7.8.

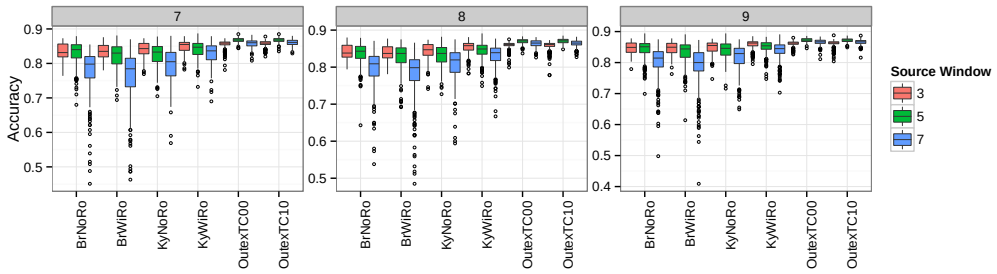


Figure 7.9: The performance of using individuals evolved on different datasets, with different code lengths on the **OutexTC10** dataset using the same source and target window sizes.

7.4.2.8 The CUReT Dataset

The results obtained from using different image descriptors on CUReT are presented in Table 7.9 and Figure 7.10. The CUReT dataset is more challenging than all other texture datasets as it comprises a very large number of classes (61 in total). Similar to KySinHw, the CUReT dataset was not used as a source dataset to evolve image descriptors, and hence, all the programs were evolved on other datasets. On average, all baseline descriptors have outperformed the automatically evolved image descriptors. GP-criptor^{ri} programs have struggled to achieve higher than 35% average accuracy on this dataset, whereas the minimum reported average performance by the baseline methods is 44.45%. Increasing the number of features, i.e., the number of children under *code*, the performance has improved compared to those programs with small number of features. The use of 5×5 pixels windows have shown to achieve better performance than the other two window sizes as depicted in Figure 7.10.

7.4.2.9 The Coins Dataset

Table 7.10 and Figure 7.11 show the results of applying classifying the Coins dataset by k-NN using the features of the automatically evolved image descriptors on the six source datasets. This dataset comprises only two classes and the task is to discriminate between a 5 cent New Zealand coin head and tail instances. The

Table 7.8: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **OutexTC10** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	83.5 ± 2.17	83.5 ± 2.03	84.0 ± 2.09	84.9 ± 1.69	85.6 ± 1.13	**85.7 ± 0.83**
		5	84.6 ± 2.14	84.8 ± 1.90	85.0 ± 1.74	85.5 ± 1.44	85.8 ± 1.33	**85.9 ± 1.06**
		7	83.0 ± 2.84	83.5 ± 2.48	83.7 ± 2.26	**84.1 ± 2.16**	83.5 ± 2.08	83.8 ± 1.84
	5	3	81.9 ± 4.15	81.2 ± 4.24	80.2 ± 4.87	83.0 ± 3.59	**86.2 ± 0.71**	86.1 ± 0.84
		5	83.1 ± 3.22	82.1 ± 3.45	82.4 ± 3.21	84.1 ± 2.43	**86.8 ± 0.60**	86.8 ± 0.64
		7	82.0 ± 3.52	80.6 ± 4.07	82.3 ± 3.00	83.8 ± 2.39	85.5 ± 1.02	**85.6 ± 1.00**
	7	3	79.2 ± 6.18	77.8 ± 7.72	74.9 ± 8.03	79.6 ± 5.91	86.0 ± 0.85	**86.1 ± 0.72**
		5	79.8 ± 5.73	78.5 ± 6.54	79.1 ± 5.11	82.4 ± 3.47	86.8 ± 0.70	**86.8 ± 0.61**
		7	77.9 ± 6.28	76.5 ± 7.04	79.5 ± 4.96	82.9 ± 2.86	85.8 ± 1.11	**86.0 ± 0.94**
8	3	3	84.1 ± 1.87	83.8 ± 1.83	84.4 ± 2.16	85.4 ± 1.52	**85.9 ± 0.99**	85.8 ± 1.15
		5	85.2 ± 1.79	85.3 ± 1.66	85.3 ± 1.72	86.0 ± 1.28	**86.1 ± 1.28**	86.0 ± 1.47
		7	84.0 ± 2.45	84.1 ± 2.30	84.3 ± 2.01	**84.7 ± 1.91**	84.0 ± 2.06	83.9 ± 2.28
	5	3	82.6 ± 3.50	81.5 ± 4.30	81.1 ± 4.40	83.1 ± 3.25	86.3 ± 0.83	**86.3 ± 0.70**
		5	83.7 ± 2.77	82.7 ± 3.46	83.1 ± 2.75	84.4 ± 2.12	87.0 ± 0.65	**87.0 ± 0.60**
		7	82.7 ± 3.00	81.5 ± 3.87	83.2 ± 2.56	84.3 ± 1.84	85.9 ± 1.02	**86.0 ± 0.94**
	7	3	80.2 ± 5.47	78.9 ± 6.87	76.7 ± 7.15	79.9 ± 5.53	**86.2 ± 0.82**	86.3 ± 0.76
		5	81.1 ± 4.63	79.8 ± 5.86	80.0 ± 4.67	82.5 ± 3.51	87.1 ± 0.62	**87.1 ± 0.61**
		7	79.7 ± 4.87	78.1 ± 6.25	80.7 ± 4.49	83.1 ± 3.05	86.3 ± 1.01	**86.4 ± 0.87**
9	3	3	84.7 ± 1.77	84.6 ± 1.84	84.9 ± 1.96	85.9 ± 1.40	86.1 ± 0.77	**86.2 ± 0.99**
		5	85.8 ± 1.51	85.7 ± 1.56	85.7 ± 1.60	86.4 ± 1.08	**86.4 ± 0.98**	86.4 ± 1.11
		7	84.5 ± 2.27	84.5 ± 2.24	84.6 ± 2.12	**85.1 ± 1.61**	84.4 ± 1.65	84.6 ± 1.73
	5	3	83.3 ± 3.55	82.6 ± 3.87	82.3 ± 4.11	83.9 ± 3.13	**86.5 ± 0.62**	86.5 ± 0.65
		5	84.3 ± 2.80	83.3 ± 3.50	84.0 ± 2.74	85.0 ± 2.15	87.2 ± 0.61	**87.2 ± 0.60**
		7	83.3 ± 3.05	81.9 ± 3.87	83.9 ± 2.45	84.9 ± 1.91	86.2 ± 0.94	**86.2 ± 0.91**
	7	3	81.2 ± 5.88	79.7 ± 7.23	77.9 ± 7.31	81.0 ± 5.65	**86.5 ± 0.63**	86.4 ± 0.74
		5	81.8 ± 4.86	80.5 ± 5.87	81.1 ± 4.71	83.4 ± 3.32	**87.3 ± 0.61**	87.3 ± 0.64
		7	80.3 ± 4.97	78.8 ± 6.21	81.6 ± 4.24	83.8 ± 2.81	86.6 ± 0.91	**86.6 ± 0.84**

best average performance reported on this dataset by the baseline descriptors is 65.37% ($\text{CLBP}_{24,3}$). GP-criptor$^{\text{ri}}$ descriptors on the other hand, have outperformed all baseline methods in over 89% (145 out of 162) of the cases as presented in Table 7.10. Unlike previous datasets, increasing the code length shows negative influence on the performance; whereas using large window sizes tend to result in

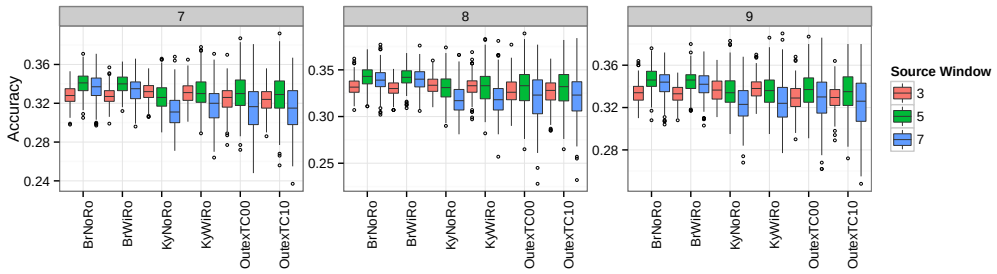


Figure 7.10: The performance of using individuals evolved on different datasets, with different code lengths on the **CUReT** dataset using the same source and target window sizes.

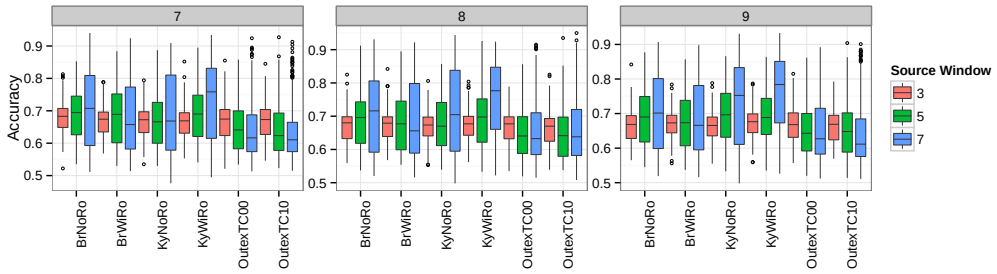


Figure 7.11: The performance of using individuals evolved on different datasets, with different code lengths on the **Coins** dataset using the same source and target window sizes.

better performance than using small windows as presented in Figure 7.11. This behaviour was expected as it has been previously observed (see Section 5.4.3on page 162). Over 15% (25 out of 162) of the cases in Table 7.10 have a higher performance than 70% accuracy on average.

7.4.2.10 The Faces Dataset

The results on the Faces dataset are presented in Table 7.11 and Figure 7.12. Similar to the Coins dataset, the Faces dataset comprises only two classes and the task is to classify those instances that show a face from others, i.e., a non-face. The majority of the automatically evolved descriptors have achieved above 60%

Table 7.9: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **CUReT** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	32.8 ± 0.93	32.7 ± 0.86	**33.2 ± 1.00**	33.0 ± 1.05	32.4 ± 1.35	32.3 ± 1.31
		5	32.3 ± 1.53	32.3 ± 1.31	32.6 ± 1.55	**32.7 ± 1.68**	31.8 ± 2.23	31.6 ± 2.24
		7	30.9 ± 2.00	30.9 ± 1.74	31.1 ± 1.96	**31.3 ± 2.13**	30.0 ± 2.76	29.8 ± 2.80
	5	3	**33.5 ± 0.94**	33.4 ± 0.88	32.8 ± 0.99	33.1 ± 1.03	32.8 ± 1.18	32.8 ± 1.29
		5	**34.0 ± 1.12**	33.9 ± 1.01	32.7 ± 1.43	33.2 ± 1.63	33.0 ± 1.90	32.9 ± 2.11
		7	33.3 ± 1.31	**33.3 ± 1.22**	31.4 ± 1.78	32.0 ± 1.99	31.7 ± 2.28	31.5 ± 2.59
	7	3	**33.6 ± 1.05**	33.2 ± 1.08	32.3 ± 0.96	32.6 ± 1.09	32.5 ± 1.15	32.5 ± 1.14
		5	**34.3 ± 1.22**	34.0 ± 1.19	32.2 ± 1.31	32.8 ± 1.53	32.7 ± 1.84	32.7 ± 1.80
		7	**33.6 ± 1.41**	33.4 ± 1.29	31.1 ± 1.63	31.8 ± 1.86	31.5 ± 2.36	31.5 ± 2.34
8	3	3	33.2 ± 0.92	33.0 ± 0.87	**33.4 ± 0.97**	33.3 ± 1.02	32.7 ± 1.36	32.6 ± 1.40
		5	32.7 ± 1.54	32.6 ± 1.32	32.8 ± 1.57	**33.0 ± 1.56**	32.2 ± 2.25	32.0 ± 2.37
		7	31.4 ± 2.00	31.4 ± 1.64	31.5 ± 2.04	**31.8 ± 1.97**	30.6 ± 2.75	30.4 ± 2.85
	5	3	33.7 ± 0.88	**33.7 ± 0.86**	33.1 ± 0.96	33.2 ± 1.12	32.9 ± 1.26	32.9 ± 1.25
		5	34.2 ± 1.11	**34.2 ± 1.03**	33.1 ± 1.43	33.2 ± 1.68	33.1 ± 2.08	33.1 ± 2.12
		7	33.6 ± 1.38	**33.6 ± 1.25**	31.9 ± 1.72	32.1 ± 1.99	31.9 ± 2.61	31.9 ± 2.63
	7	3	**33.9 ± 0.90**	33.8 ± 1.06	32.7 ± 0.96	32.8 ± 1.00	32.7 ± 1.19	32.8 ± 1.08
		5	**34.6 ± 1.05**	34.5 ± 1.10	32.8 ± 1.28	32.9 ± 1.41	33.1 ± 1.94	33.1 ± 1.80
		7	**34.0 ± 1.20**	34.0 ± 1.23	31.9 ± 1.56	31.9 ± 1.72	32.0 ± 2.44	32.1 ± 2.33
9	3	3	33.4 ± 0.94	33.3 ± 0.85	33.6 ± 1.07	**33.7 ± 1.07**	32.9 ± 1.27	32.9 ± 1.19
		5	32.9 ± 1.55	33.0 ± 1.29	33.1 ± 1.57	**33.6 ± 1.52**	32.5 ± 2.13	32.6 ± 1.93
		7	31.6 ± 2.02	31.8 ± 1.71	31.8 ± 1.98	**32.4 ± 1.89**	31.0 ± 2.72	31.1 ± 2.37
	5	3	**34.1 ± 0.95**	34.0 ± 0.92	33.5 ± 1.11	33.6 ± 1.01	33.3 ± 1.13	33.2 ± 1.14
		5	**34.7 ± 1.12**	34.5 ± 1.09	33.5 ± 1.50	33.6 ± 1.46	33.7 ± 1.79	33.5 ± 1.91
		7	**34.0 ± 1.29**	33.8 ± 1.25	32.4 ± 1.77	32.6 ± 1.77	32.7 ± 2.14	32.4 ± 2.37
	7	3	**34.1 ± 1.04**	33.9 ± 1.00	33.1 ± 1.12	33.2 ± 1.20	33.2 ± 1.16	33.1 ± 1.08
		5	**34.9 ± 1.12**	34.8 ± 1.09	33.3 ± 1.37	33.4 ± 1.61	33.8 ± 1.78	33.5 ± 1.82
		7	**34.3 ± 1.28**	34.2 ± 1.19	32.4 ± 1.70	32.5 ± 1.95	32.9 ± 2.22	32.5 ± 2.39

accuracy on average, and over 82% (134 out of 162) of those cases show more than 65% accuracy. The baseline descriptors, specifically $\text{LBP}^{u2}_{8,1}$ with 75.79% average accuracy, have achieved reasonably better performance than the automatically evolved descriptors; however the gap is about 4.17% between the best average of the two groups (baseline and GP-criptor$^{\text{ri}}$ descriptors). Increasing the window

Table 7.10: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **Coins** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	**67.7 ± 4.79**	67.0 ± 4.67	66.6 ± 4.94	66.5 ± 5.21	66.7 ± 5.69	66.7 ± 5.54
		5	**68.6 ± 8.17**	68.3 ± 8.45	66.4 ± 7.77	65.6 ± 7.84	66.3 ± 8.21	66.1 ± 7.95
		7	71.7 ± 11.3	**71.9 ± 11.9**	69.1 ± 10.6	67.6 ± 10.9	68.1 ± 10.4	67.9 ± 9.86
	5	3	**68.3 ± 4.19**	68.0 ± 4.51	67.2 ± 4.77	68.0 ± 4.63	67.1 ± 5.63	66.7 ± 6.03
		5	**68.5 ± 7.47**	68.4 ± 8.25	66.9 ± 8.04	68.5 ± 7.83	64.9 ± 7.47	64.2 ± 7.66
		7	71.4 ± 11.0	70.9 ± 11.7	69.1 ± 11.0	**71.6 ± 11.1**	66.8 ± 9.66	65.5 ± 9.36
	7	3	**69.1 ± 4.67**	68.3 ± 4.69	66.6 ± 4.66	68.4 ± 4.11	66.2 ± 5.97	65.7 ± 5.83
		5	**68.3 ± 8.06**	66.8 ± 7.85	67.0 ± 8.88	69.5 ± 7.75	63.1 ± 7.10	62.6 ± 7.21
		7	70.2 ± 11.4	68.2 ± 11.0	69.6 ± 12.3	**73.5 ± 11.2**	63.9 ± 8.62	63.4 ± 8.42
8	3	3	66.8 ± 4.55	**67.4 ± 4.49**	66.8 ± 4.52	67.3 ± 4.63	66.9 ± 5.07	66.3 ± 5.23
		5	67.9 ± 8.22	**69.2 ± 8.82**	67.0 ± 7.81	67.5 ± 7.93	66.3 ± 7.59	65.5 ± 7.47
		7	71.0 ± 11.1	**72.5 ± 11.9**	69.3 ± 10.9	70.4 ± 10.9	68.0 ± 9.61	67.0 ± 9.94
	5	3	**68.5 ± 4.32**	67.7 ± 4.03	67.6 ± 4.31	68.3 ± 4.55	66.8 ± 5.20	66.7 ± 5.37
		5	69.0 ± 8.14	67.8 ± 8.24	67.8 ± 8.08	**69.3 ± 8.13**	65.1 ± 7.56	64.8 ± 7.62
		7	71.0 ± 11.0	69.5 ± 11.4	70.3 ± 11.6	**72.6 ± 11.4**	66.9 ± 9.74	66.9 ± 10.2
	7	3	68.4 ± 4.33	68.1 ± 4.34	68.1 ± 4.60	**68.4 ± 3.76**	66.6 ± 5.26	66.4 ± 5.63
		5	68.5 ± 8.08	67.7 ± 7.69	68.9 ± 8.78	**71.1 ± 7.73**	64.6 ± 7.53	64.1 ± 7.40
		7	70.1 ± 11.4	68.5 ± 11.2	71.3 ± 12.4	**75.5 ± 11.1**	65.7 ± 9.48	65.4 ± 9.33
9	3	3	66.3 ± 4.50	66.9 ± 4.29	66.3 ± 4.10	**67.3 ± 4.44**	66.8 ± 5.28	66.4 ± 4.94
		5	66.5 ± 8.47	**69.0 ± 8.76**	66.3 ± 7.16	66.6 ± 7.48	65.3 ± 7.35	65.2 ± 7.47
		7	68.7 ± 11.4	**72.5 ± 11.8**	69.2 ± 10.4	68.7 ± 10.4	66.9 ± 9.72	67.3 ± 10.0
	5	3	68.3 ± 4.34	67.5 ± 4.05	67.9 ± 3.91	**68.3 ± 3.98**	67.1 ± 4.91	67.4 ± 5.21
		5	68.6 ± 7.79	67.5 ± 7.75	**69.6 ± 7.72**	69.1 ± 7.56	64.9 ± 7.07	65.4 ± 7.54
		7	70.9 ± 11.2	68.6 ± 11.2	**73.3 ± 10.8**	72.9 ± 10.9	66.8 ± 9.92	66.9 ± 9.91
	7	3	**68.9 ± 4.43**	68.4 ± 4.14	67.8 ± 4.21	68.6 ± 3.30	66.5 ± 5.21	65.5 ± 5.25
		5	68.5 ± 7.66	67.5 ± 7.24	69.8 ± 8.44	**71.4 ± 7.30**	63.8 ± 6.89	63.2 ± 6.68
		7	70.1 ± 10.9	68.5 ± 10.6	72.9 ± 12.0	**75.9 ± 10.4**	65.3 ± 9.09	64.0 ± 9.11

size shows negative influence on the performance on this dataset as depicted in Figure 7.12. Meanwhile, the effect of the code length is minimal, although it shows some improvement with longer feature vectors (more children under the *code* node). This behaviour was also expected as has been previously observed in Section 5.4.3.1 (see page 162), and it is due mainly to the size of each instance (19×19 pixels).

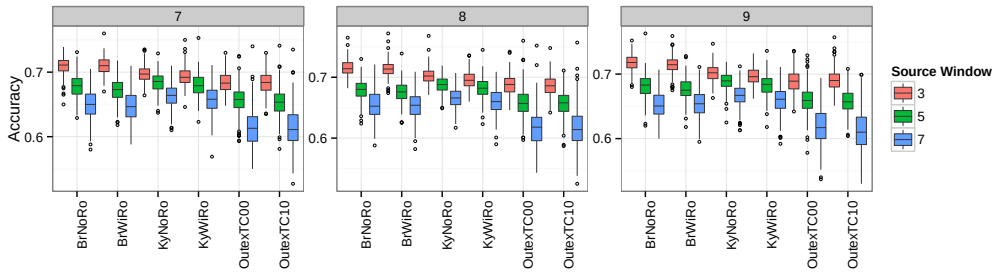


Figure 7.12: The performance of using individuals evolved on different datasets, with different code lengths on the **Faces** dataset using the same source and target window sizes.

7.4.2.11 The Dslr Dataset

The results on the first domain of the Office dataset (Dslr) are presented in Table 7.12 and Figure 7.13. The instances of Dslr are larger than all other datasets' instances, which each of size 1000×1000 pixels. Although each instance in this dataset presents an object, the pose of those objects are uncontrolled, the objects of each category are different (e.g. phones with different types, shapes, and colours), and the background of each instance is different. As presented in Table 7.1, the best average performance on this dataset by the baseline descriptors has been achieved by $\text{CLBC}_{24,3}$ with 43.74% accuracy. Meanwhile, the best average accuracy that was achieved by the automatically evolved descriptors is 47.80%. Moreover, over 57% (93 out of 162) of the cases in Table 7.12 show on average better performance than the best of the baseline descriptors. Unlike other datasets, increasing the target window size has positive influence on the performance as depicted in Figure 7.13. This is expected as the size of those instances are very large and small windows can capture very small amount of information. The use of smaller source window size has better performance than those descriptors that were evolved with a larger window. The impact of the code length, on the other hand, is minimal and only a slight improvement has been observed by increasing this factor.

Table 7.11: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **Faces** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	**71.0 ± 1.38**	70.9 ± 1.23	69.7 ± 1.24	69.3 ± 1.37	68.4 ± 1.57	68.4 ± 1.64
		5	68.4 ± 1.97	**68.6 ± 1.65**	67.6 ± 1.83	66.9 ± 2.07	64.9 ± 2.19	64.7 ± 2.18
		7	65.7 ± 2.44	**66.2 ± 2.26**	64.7 ± 2.45	63.7 ± 2.73	61.3 ± 2.67	61.0 ± 2.77
	5	3	69.1 ± 1.47	68.3 ± 1.60	**69.3 ± 1.45**	69.0 ± 1.33	68.3 ± 1.60	68.0 ± 1.76
		5	67.8 ± 1.68	67.2 ± 1.82	**68.4 ± 1.49**	67.9 ± 1.69	65.6 ± 2.05	65.4 ± 2.19
		7	65.4 ± 2.15	65.2 ± 2.19	**66.2 ± 1.98**	65.7 ± 2.33	62.0 ± 2.85	61.7 ± 2.89
	7	3	68.1 ± 1.73	67.6 ± 1.97	**68.3 ± 1.90**	68.3 ± 1.66	67.8 ± 1.57	67.8 ± 1.61
		5	67.3 ± 1.90	66.6 ± 1.87	**68.4 ± 1.78**	67.7 ± 1.76	65.0 ± 2.19	65.0 ± 2.38
		7	65.0 ± 2.28	64.8 ± 2.21	**66.3 ± 1.80**	65.7 ± 2.06	61.3 ± 2.93	61.5 ± 3.06
8	3	3	**71.5 ± 1.27**	71.4 ± 1.24	70.2 ± 1.30	69.5 ± 1.31	68.8 ± 1.58	68.6 ± 1.64
		5	68.8 ± 1.72	**69.1 ± 1.48**	68.0 ± 1.76	67.1 ± 2.02	65.2 ± 2.14	65.1 ± 2.14
		7	66.1 ± 2.46	**66.5 ± 2.06**	65.0 ± 2.52	63.8 ± 2.82	61.5 ± 2.73	61.4 ± 2.77
	5	3	69.2 ± 1.49	68.9 ± 1.45	**69.6 ± 1.17**	69.4 ± 1.35	68.7 ± 1.60	68.5 ± 1.66
		5	67.9 ± 1.64	67.6 ± 1.54	**68.8 ± 1.35**	68.2 ± 1.60	65.9 ± 2.33	65.7 ± 2.12
		7	65.9 ± 2.03	65.8 ± 2.02	**66.5 ± 1.84**	65.7 ± 2.26	62.1 ± 2.99	62.2 ± 2.75
	7	3	68.2 ± 1.75	67.9 ± 1.58	68.7 ± 1.64	**68.7 ± 1.53**	68.0 ± 1.35	67.9 ± 1.62
		5	67.5 ± 1.77	67.1 ± 1.86	**68.5 ± 1.51**	68.2 ± 1.66	65.2 ± 2.21	65.1 ± 2.35
		7	65.5 ± 2.25	65.3 ± 2.20	**66.5 ± 1.60**	66.1 ± 2.03	61.8 ± 3.06	61.6 ± 3.08
9	3	3	**71.8 ± 1.18**	71.4 ± 1.23	70.2 ± 1.29	69.7 ± 1.27	68.9 ± 1.71	69.1 ± 1.60
		5	**69.1 ± 1.60**	69.0 ± 1.68	68.0 ± 1.79	67.3 ± 1.86	65.4 ± 2.08	65.6 ± 2.00
		7	66.3 ± 2.39	**66.5 ± 2.23**	64.8 ± 2.79	63.9 ± 2.74	61.7 ± 2.77	61.9 ± 2.64
	5	3	69.3 ± 1.42	68.8 ± 1.46	**69.5 ± 1.33**	69.4 ± 1.26	68.8 ± 1.63	68.6 ± 1.48
		5	68.1 ± 1.75	67.6 ± 1.67	**68.9 ± 1.36**	68.3 ± 1.57	65.9 ± 2.08	65.8 ± 1.92
		7	65.9 ± 2.30	65.6 ± 2.11	**66.8 ± 1.69**	66.0 ± 2.11	62.1 ± 2.83	62.0 ± 2.66
	7	3	68.2 ± 1.72	67.9 ± 1.76	68.7 ± 1.78	**68.7 ± 1.56**	68.3 ± 1.51	68.1 ± 1.55
		5	67.4 ± 1.65	67.1 ± 1.82	**68.6 ± 1.67**	68.1 ± 1.75	65.6 ± 2.14	65.1 ± 2.45
		7	65.3 ± 2.15	65.3 ± 2.18	**66.7 ± 1.64**	66.0 ± 1.94	62.0 ± 2.83	61.3 ± 3.21

7.4.2.12 The Amazon Dataset

Table 7.13 and Figure 7.14 show the results on the Amazon dataset (the second domain of Office). This dataset is more challenging compared to the other datasets of Office due to the variations of its instances' content. For example, two objects of

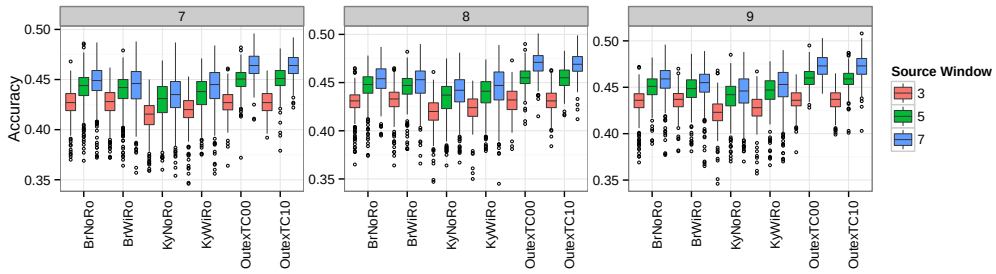


Figure 7.13: The performance of using individuals evolved on different datasets, with different code lengths on the **Dslr** dataset using the same source and target window sizes.

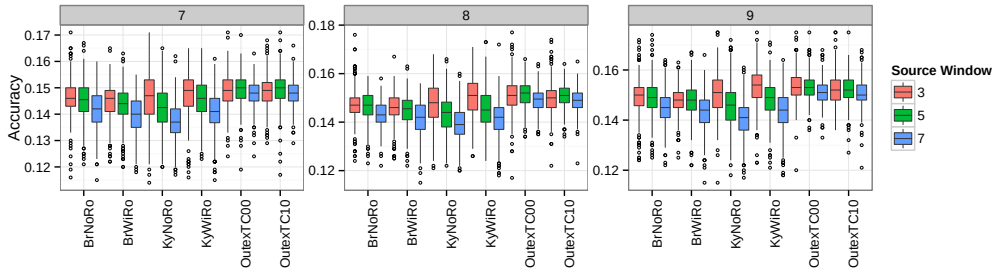


Figure 7.14: The performance of using individuals evolved on different datasets, with different code lengths on the **Amazon** dataset using the same source and target window sizes.

the same group or from different groups can be on the same instance as shown in Figure 2.24 (see page 71). Overall, the baseline and GP-criptor^{ri} image descriptors did not achieve over 18.81% ($CLBP_{24,3}$) accuracy on average. In the best case scenario of the automatically evolved descriptors shows on average 15.5% accuracy. The impact of the code length on the performance is similar to that on the Dslr dataset, which shows a slightly improvement with increased number of children under the *code* node as depicted in Figure 7.14. Similarly, increasing the source window has negative influence on the performance; whilst the target window with size 5×5 shows better performance than the other two sizes.

Table 7.12: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **Dslr** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	42.6 ± 1.51	42.7 ± 1.55	41.4 ± 1.63	41.8 ± 1.66	42.7 ± 1.26	**42.7 ± 1.24**
		5	45.1 ± 1.47	45.2 ± 1.46	44.1 ± 1.63	44.7 ± 1.63	**45.3 ± 1.13**	45.2 ± 1.09
		7	46.5 ± 1.51	46.8 ± 1.46	45.6 ± 1.66	46.2 ± 1.70	**46.9 ± 1.34**	46.8 ± 1.29
	5	3	41.0 ± 1.86	40.7 ± 2.03	39.6 ± 2.19	40.3 ± 1.93	**42.5 ± 1.41**	42.4 ± 1.46
		5	44.1 ± 1.72	43.8 ± 1.89	42.7 ± 2.16	43.6 ± 1.80	**45.0 ± 1.21**	45.0 ± 1.27
		7	45.8 ± 1.67	45.5 ± 1.77	44.3 ± 2.14	45.1 ± 1.85	46.4 ± 1.30	**46.5 ± 1.44**
	7	3	39.7 ± 2.20	39.2 ± 2.56	38.4 ± 2.35	39.3 ± 2.26	42.4 ± 1.46	**42.5 ± 1.37**
		5	43.0 ± 1.98	42.5 ± 2.29	41.6 ± 2.35	42.6 ± 2.19	45.0 ± 1.18	**45.0 ± 1.11**
		7	44.6 ± 1.99	44.2 ± 2.22	43.3 ± 2.33	44.1 ± 2.29	46.4 ± 1.31	**46.4 ± 1.19**
8	3	3	42.9 ± 1.48	43.1 ± 1.46	41.9 ± 1.59	42.3 ± 1.46	43.1 ± 1.37	**43.1 ± 1.17**
		5	45.5 ± 1.38	45.7 ± 1.45	44.6 ± 1.47	45.1 ± 1.38	**45.7 ± 1.17**	45.6 ± 1.14
		7	47.1 ± 1.46	47.3 ± 1.53	46.1 ± 1.55	46.6 ± 1.46	**47.3 ± 1.30**	47.2 ± 1.32
	5	3	41.4 ± 1.78	41.3 ± 1.67	40.2 ± 2.03	40.7 ± 1.92	**43.0 ± 1.37**	42.9 ± 1.33
		5	44.6 ± 1.65	44.5 ± 1.58	43.3 ± 2.05	43.8 ± 1.80	**45.5 ± 1.11**	45.5 ± 1.15
		7	46.3 ± 1.57	46.1 ± 1.52	44.8 ± 2.09	45.3 ± 1.78	**47.1 ± 1.27**	47.0 ± 1.33
	7	3	40.4 ± 1.79	40.0 ± 1.96	39.1 ± 2.24	39.5 ± 2.29	42.8 ± 1.32	**42.9 ± 1.26**
		5	43.7 ± 1.54	43.3 ± 1.89	42.4 ± 2.17	42.8 ± 2.25	**45.5 ± 1.14**	45.4 ± 1.12
		7	45.3 ± 1.60	45.0 ± 1.89	44.0 ± 2.09	44.4 ± 2.28	**47.0 ± 1.26**	46.9 ± 1.23
9	3	3	43.4 ± 1.58	43.6 ± 1.20	42.3 ± 1.59	42.7 ± 1.59	43.6 ± 1.17	**43.6 ± 1.14**
		5	45.9 ± 1.52	**46.2 ± 1.21**	45.0 ± 1.50	45.6 ± 1.34	46.1 ± 0.96	46.0 ± 1.11
		7	47.5 ± 1.61	**47.8 ± 1.29**	46.5 ± 1.57	47.1 ± 1.40	47.7 ± 1.12	47.7 ± 1.36
	5	3	41.8 ± 1.67	41.6 ± 1.78	40.7 ± 2.00	41.3 ± 1.90	**43.5 ± 1.13**	43.3 ± 1.27
		5	44.9 ± 1.43	44.7 ± 1.61	43.9 ± 1.92	44.5 ± 1.89	**46.0 ± 1.06**	45.8 ± 1.10
		7	46.6 ± 1.45	46.4 ± 1.58	45.5 ± 1.93	45.9 ± 1.96	**47.6 ± 1.20**	47.4 ± 1.25
	7	3	40.6 ± 2.02	40.2 ± 2.17	39.5 ± 2.39	40.1 ± 2.16	**43.1 ± 1.21**	43.1 ± 1.27
		5	43.9 ± 1.91	43.5 ± 1.99	42.8 ± 2.20	43.4 ± 2.15	**45.9 ± 1.05**	45.8 ± 1.18
		7	45.6 ± 1.87	45.2 ± 1.98	44.3 ± 2.18	45.0 ± 2.22	**47.3 ± 1.20**	47.2 ± 1.32

7.4.2.13 The Webcam Dataset

The third dataset of Office is Webcam, and the results obtained on this dataset are presented in Table 7.14 and Figure 7.15. Amongst the baseline descriptors, $\text{CLBC}_{24,3}$ has achieved the best performance with 37.87% accuracy on average.

Table 7.13: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **Amazon** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	14.6 ± 0.66	14.5 ± 0.68	14.7 ± 0.89	14.8 ± 0.83	**14.9 ± 0.63**	14.8 ± 0.61
		5	14.7 ± 0.61	14.7 ± 0.63	14.7 ± 0.76	**14.9 ± 0.73**	14.8 ± 0.55	14.8 ± 0.56
		7	14.6 ± 0.61	14.6 ± 0.60	14.6 ± 0.67	**14.8 ± 0.67**	14.7 ± 0.55	14.7 ± 0.55
	5	3	14.3 ± 0.74	14.2 ± 0.76	14.0 ± 0.89	14.4 ± 0.84	**14.8 ± 0.69**	14.8 ± 0.73
		5	14.5 ± 0.70	14.3 ± 0.69	14.2 ± 0.81	14.6 ± 0.77	**14.9 ± 0.59**	14.9 ± 0.64
		7	14.4 ± 0.65	14.3 ± 0.61	14.2 ± 0.75	14.5 ± 0.70	**14.8 ± 0.54**	14.8 ± 0.58
	7	3	14.1 ± 0.85	13.8 ± 0.92	13.4 ± 0.91	13.8 ± 0.97	**14.6 ± 0.64**	14.6 ± 0.67
		5	14.2 ± 0.79	14.0 ± 0.82	13.7 ± 0.82	14.0 ± 0.86	14.8 ± 0.58	**14.9 ± 0.55**
		7	14.2 ± 0.71	14.0 ± 0.74	13.7 ± 0.75	14.0 ± 0.78	14.7 ± 0.54	**14.8 ± 0.48**
8	3	3	14.7 ± 0.68	14.6 ± 0.61	14.8 ± 0.82	15.0 ± 0.75	**15.1 ± 0.68**	15.0 ± 0.64
		5	14.8 ± 0.62	14.8 ± 0.58	14.9 ± 0.75	**15.2 ± 0.66**	15.0 ± 0.64	15.0 ± 0.57
		7	14.7 ± 0.61	14.6 ± 0.55	14.8 ± 0.68	**15.1 ± 0.59**	14.9 ± 0.61	14.8 ± 0.56
	5	3	14.5 ± 0.65	14.4 ± 0.64	14.2 ± 0.89	14.3 ± 0.89	**15.0 ± 0.64**	14.9 ± 0.61
		5	14.7 ± 0.60	14.5 ± 0.62	14.3 ± 0.78	14.5 ± 0.82	15.1 ± 0.54	**15.1 ± 0.49**
		7	14.6 ± 0.56	14.4 ± 0.59	14.3 ± 0.71	14.5 ± 0.76	15.0 ± 0.49	**15.0 ± 0.47**
	7	3	14.3 ± 0.66	14.1 ± 0.78	13.7 ± 0.92	13.9 ± 0.92	14.8 ± 0.63	**14.8 ± 0.60**
		5	14.4 ± 0.62	14.3 ± 0.73	13.9 ± 0.78	14.1 ± 0.87	15.0 ± 0.54	**15.0 ± 0.52**
		7	14.3 ± 0.55	14.2 ± 0.67	13.9 ± 0.70	14.1 ± 0.79	14.9 ± 0.50	**14.9 ± 0.49**
9	3	3	14.9 ± 0.72	14.8 ± 0.56	15.1 ± 0.81	15.3 ± 0.79	**15.3 ± 0.62**	15.2 ± 0.60
		5	15.0 ± 0.60	15.0 ± 0.50	15.1 ± 0.77	**15.5 ± 0.73**	15.2 ± 0.56	15.2 ± 0.57
		7	14.9 ± 0.56	14.9 ± 0.49	15.0 ± 0.70	**15.3 ± 0.67**	15.1 ± 0.53	15.0 ± 0.54
	5	3	14.7 ± 0.72	14.7 ± 0.73	14.4 ± 0.98	14.6 ± 0.87	**15.2 ± 0.63**	15.1 ± 0.67
		5	14.9 ± 0.70	14.7 ± 0.68	14.6 ± 0.89	14.8 ± 0.81	**15.3 ± 0.53**	15.2 ± 0.57
		7	14.7 ± 0.64	14.6 ± 0.61	14.6 ± 0.77	14.8 ± 0.76	**15.2 ± 0.47**	15.1 ± 0.52
	7	3	14.4 ± 0.83	14.3 ± 0.85	13.9 ± 0.99	14.1 ± 1.01	**15.1 ± 0.61**	15.0 ± 0.66
		5	14.6 ± 0.73	14.4 ± 0.77	14.1 ± 0.84	14.3 ± 0.87	**15.2 ± 0.52**	15.2 ± 0.55
		7	14.5 ± 0.70	14.3 ± 0.70	14.1 ± 0.74	14.3 ± 0.78	**15.1 ± 0.48**	15.1 ± 0.53

Meanwhile, the best average performance reported on the Webcam dataset was 44.10% on by those descriptors automatically evolved on the BrWiRo source dataset with a code of length 9-bits and source and target windows, respectively, 3×3 pixels and 7×7 pixels. Table 7.14 shows that there are over 56% (92 out of 162) cases have achieved better performance than the best of the baseline descriptors. Following

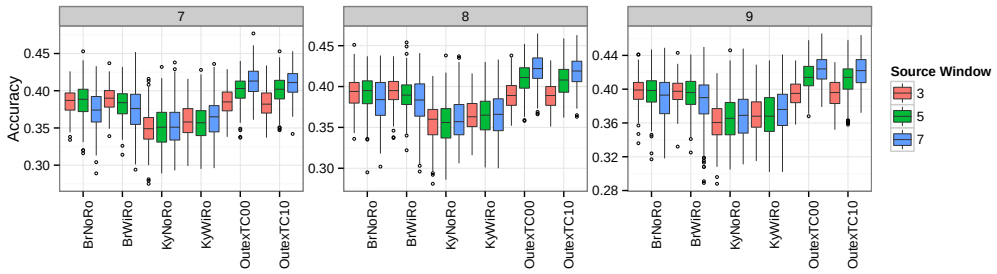


Figure 7.15: The performance of using individuals evolved on different datasets, with different code lengths on the **Webcam** dataset using the same source and target window sizes.

the same pattern of Dslr and Amazon, increase the code length shows slightly improvement in the performance as depicted in Figure 7.15. The combination of a large target window and a small source window shows better performance than small target windows or large source windows only as presented in Table 7.14.

7.4.2.14 Summary

In summary, the following observations can be drawn from the obtained results of the conducted experiments:

- The automatically evolved image descriptors can be directly used on other datasets without changing or modifying any part of those descriptors.
- The performance achieved from those automatically designed descriptors is comparable or better than that of the domain-expert hand-crafted descriptors in most cases.
- The descriptors can be applied to datasets that have a different number of classes, rotation, illumination, and instances sizes.
- The descriptors can be used to perform classification on datasets of a related domain to the source dataset, e.g., texture, or different domains, e.g., object and face classification.

Table 7.14: The average accuracy (%) of k-NN using descriptors evolved on six datasets with different code length, source window size, and target window size combinations on the **Webcam** dataset ($\bar{x} \pm s$).

	Window size		Source dataset					
Code	source	target	BrNoRo	BrWiRo	KyNoRo	KyWiRo	OutexTC00	OutexTC10
7	3	3	38.6 ± 1.69	**38.9 ± 1.64**	34.9 ± 2.33	35.9 ± 2.23	38.6 ± 1.76	38.3 ± 1.72
		5	41.5 ± 1.64	**42.0 ± 1.55**	38.6 ± 2.11	39.3 ± 1.99	41.4 ± 1.45	41.2 ± 1.49
		7	42.5 ± 1.71	**43.1 ± 1.68**	40.3 ± 1.97	40.9 ± 1.85	42.4 ± 1.43	42.3 ± 1.50
	5	3	34.8 ± 2.46	34.3 ± 2.16	31.5 ± 2.91	32.4 ± 2.77	**36.8 ± 2.07**	36.7 ± 2.16
		5	38.6 ± 2.28	38.2 ± 1.99	35.1 ± 2.68	35.8 ± 2.57	**40.1 ± 1.87**	40.1 ± 1.91
		7	40.4 ± 2.21	40.2 ± 1.95	37.1 ± 2.59	37.8 ± 2.43	**41.6 ± 1.85**	41.6 ± 1.87
	7	3	31.5 ± 3.00	31.5 ± 3.04	29.6 ± 2.97	30.8 ± 3.02	**36.0 ± 2.23**	35.8 ± 2.13
		5	35.3 ± 2.70	35.2 ± 2.96	33.3 ± 2.72	34.3 ± 2.67	**39.6 ± 2.04**	39.5 ± 1.84
		7	37.5 ± 2.51	37.4 ± 2.82	35.4 ± 2.64	36.4 ± 2.57	**41.2 ± 1.95**	41.1 ± 1.72
8	3	3	39.2 ± 1.85	**39.4 ± 1.67**	35.8 ± 2.29	36.5 ± 2.25	38.9 ± 1.72	38.9 ± 1.60
		5	42.0 ± 1.63	**42.5 ± 1.52**	39.4 ± 2.05	39.9 ± 1.91	41.8 ± 1.39	41.8 ± 1.32
		7	43.1 ± 1.66	**43.6 ± 1.51**	41.0 ± 1.97	41.5 ± 1.78	43.0 ± 1.40	42.9 ± 1.35
	5	3	35.4 ± 2.30	35.1 ± 2.04	32.1 ± 2.81	33.1 ± 2.74	**37.6 ± 2.08**	37.2 ± 2.17
		5	39.3 ± 2.07	39.0 ± 1.87	35.7 ± 2.60	36.5 ± 2.46	**41.0 ± 1.85**	40.7 ± 1.94
		7	41.0 ± 2.01	40.8 ± 1.87	37.8 ± 2.43	38.5 ± 2.29	**42.5 ± 1.82**	42.2 ± 1.84
	7	3	32.6 ± 2.93	32.3 ± 2.92	30.2 ± 3.14	30.9 ± 2.95	**36.9 ± 2.18**	36.6 ± 2.32
		5	36.3 ± 2.82	36.0 ± 2.89	33.8 ± 2.84	34.5 ± 2.63	**40.6 ± 1.99**	40.3 ± 1.94
		7	38.4 ± 2.69	38.1 ± 2.80	36.0 ± 2.68	36.6 ± 2.56	**42.1 ± 1.92**	41.9 ± 1.75
9	3	3	**39.8 ± 1.58**	39.7 ± 1.47	36.1 ± 2.23	36.9 ± 2.27	39.5 ± 1.53	39.5 ± 1.64
		5	42.7 ± 1.41	**42.8 ± 1.36**	39.8 ± 1.87	40.4 ± 1.88	42.3 ± 1.30	42.4 ± 1.42
		7	43.8 ± 1.51	**44.1 ± 1.51**	41.5 ± 1.83	42.1 ± 1.76	43.4 ± 1.33	43.5 ± 1.48
	5	3	35.6 ± 2.35	35.4 ± 2.26	32.8 ± 2.82	33.2 ± 2.99	**38.0 ± 2.00**	37.7 ± 2.06
		5	39.6 ± 2.05	39.4 ± 2.04	36.5 ± 2.56	37.0 ± 2.69	**41.5 ± 1.70**	41.2 ± 1.80
		7	41.5 ± 1.94	41.4 ± 2.03	38.6 ± 2.45	39.1 ± 2.50	**43.0 ± 1.69**	42.8 ± 1.79
	7	3	32.9 ± 2.73	32.7 ± 3.04	30.9 ± 3.12	31.8 ± 3.14	**36.8 ± 2.15**	36.7 ± 2.18
		5	36.8 ± 2.61	36.4 ± 2.86	34.7 ± 2.81	35.4 ± 2.73	**40.6 ± 1.89**	40.5 ± 1.88
		7	39.1 ± 2.53	38.6 ± 2.76	36.9 ± 2.65	37.5 ± 2.61	**42.3 ± 1.79**	42.1 ± 1.78

- The descriptors can be used on the target dataset with a different window size from the one used to evolve that descriptor on the source dataset. This flexibility is not available in the vast majority of the existing descriptors.

- The results show that the window size is highly dependent on the size of

the instances, such that larger instances require a larger window in order to capture more distinctive features.

- The results also show that the impact of the code length is less than the window size.

7.5 Chapter Summary

In this chapter, the generalisability of the automatically evolved image descriptors by GP-criptor^{ri} has been investigated via adopting transfer learning in order to examine the possibility of directly use an image descriptor to tackle problems in related and unrelated domains. The experiments have been designed to assess different aspects of those descriptors, which were evaluated using 13 image classification benchmarks that vary in number of classes, instances per class, rotations, illumination, type (texture, faces, and object classification tasks), and instance size/dimension. The performance of those automatically evolved descriptors are compared to six state-of-the-art domain-expert designed image descriptors. The results of the conducted experiments show that GP-criptor^{ri} evolved descriptors have achieved comparable or better performance to that of the competitive descriptors. Moreover, those automatically designed descriptors have successfully tackled problems of the same domain of the datasets they were originally used to evolve them as well as different domains. More importantly, the automatically evolved descriptors have the flexibility to apply them on the target dataset using different window sizes without adjusting or changing any part of the descriptor. The majority of hand-crafted descriptors lack this flexibility and changing such a parameter most likely needs human intervention to undertake the required modifications.

Although a number of aspects have been investigated in this chapter, more deep and through investigation is needed. The initial results are promising and show the potential of those automatically evolved descriptors to tackle different problems.

8

Conclusions and Future Work

This chapter concludes the discussion of this thesis, highlights the main findings, and outlines directions for future work.

The overall goal of this thesis was to develop a new domain independent GP approach to image classification by utilising GP to evolve programs that are capable of automatically detecting diverse and informative keypoints, designing a set of features, and performing feature extraction using only a small number of training instances to facilitate image classification, and are robust to different image changes such as illumination and rotation. This goal was successfully achieved by developing a number of new GP representations to automatically evolve image descriptors that operate directly on the raw pixel values of an image and generate the corresponding feature vectors. The proposed methods were evaluated on a range of image classification benchmarks and compared with existing state-of-the-art methods. The results show that the newly proposed methods in this thesis have achieved either competitive performance or outperformed the prior state-of-the-art methods. The proposed methods have been also thoroughly examined and analysed to assess their capability to handle different image changes such as illumination and rotation.

The remainder of the chapter provides conclusions for the individual objectives and highlights the main findings from each chapter. Then an insightful discussion is provided to suggest several potential research directions for future work.

8.1 Achieved Objectives

This thesis has successfully fulfilled the following research objectives:

- Through this thesis, two new GP representations were proposed to evolve models for binary classification in images using only a few instances per class. The two methods correspond to One-shot GP and Compound-GP (Chapter 3) and aim at detecting image regions that best classify the instances of the two classes. The two methods rely on measuring the distances between the feature vectors of the instances from the same class and those from a different class. The experimental results show that the programs evolved by those methods have either significantly outperformed or achieved comparable performance to both GP and non-GP well-known methods in the literature.

- This thesis developed three new GP representations to automatically construct image descriptors, namely GP-criptor (Chapter 4), GP-criptor$^{\text{ri}}$ (Chapter 5), and EID$^{\text{ri}}$ (Chapter 6), for multi-class image classification tasks using only a few instances per class. Instead of evolving a classifier to classify the instances of more than two classes, the new methods take a step backward and tackle the problem from the feature generation step such that they transform instances of the same class to have similar feature vectors that are distinctive from instances of the other classes. The results of the conducted experiments have made it evident that these methods have significantly better performance than both automatically designed and domain-expert hand-crafted features, and are robust to different image changes such as illumination and rotation.

- This thesis proposed a transfer learning methodology in GP (Chapter 7) by investigating the generalizability of those image descriptors automatically synthesised on one dataset to perform the classification on different datasets

that are from related and unrelated domains. The experimental results demonstrate the transferability of those descriptors and their goodness to achieve comparable or better results to domain-expert designed image descriptors, particularly for problems in the same domain and some tasks in a different domain.

8.2 Main Conclusions

Overall, this thesis finds that GP is effective to address the problem of having a few training instances for image classification by automatically evolving programs that can detect informative keypoints, and extract features that are more representative of each class. Most of the newly proposed methods in this thesis have successfully provided better classification performance than the prior state-of-the-art algorithms.

The main conclusions drawn from each of the five contribution chapters (Chapter 3 through Chapter 7) for the three research objectives are presented and discussed in this section.

8.2.1 Sparse Keypoints and Dense Features for Image Classification

Chapter 3 proposes two new GP methods for automatic sparse keypoints detection, feature extraction and image classification in a single program. The proposed methods combine the concepts of both sparse and dense image descriptors. The performance of the proposed methods have been compared with Two-tier GP [7], conventional GP and six well-known classification algorithms from the literature with domain-expert hand-crafted features. The 95% confidence intervals from multiple runs of these two newly proposed methods, i.e., One-shot GP and Compound-GP, show that they potentially outperformed the competitive methods in many cases. These two methods have achieved over 90.0% average accuracy in many cases.

Following is a summary of the main findings in terms of: 1) program representation, 2) fitness measure, and 3) generality of the evolved programs.

8.2.1.1 Representation

Chapter 3 proposes two methods that demonstrate how the attributes of sparse descriptors and dense descriptors can be integrated into a single GP individual. Combining the attributes of sparse descriptors to detect some good regions of the image, and dense descriptors to extract more effective features from each region rather than relying on the pixel values (low-features) has been found to improve the classification performance.

8.2.1.2 Fitness measure

It has been observed (Chapter 3) that those methods that rely on the accuracy measure to build/evolve a classifier have performed poorly when the number of training examples is limited. The accuracy measure is inappropriate mainly because the learning algorithm can simply select a few unreliable features to perfectly discriminate the training instances. Therefore, to address the problem of having only a few learning instances, the proposed methods rely on the goodness of multiple keypoints through considering the distances (separability) between those keypoints in instances of different classes.

8.2.1.3 Generality

Although the evolved GP models have been shown to outperform the other competitive methods, the detected keypoints and extracted features are not biased to a specific type of classification algorithm. Using the features extracted by the proposed GP methods has been found to be effective and has the potential to improve the performance of different classification algorithms.

8.2.2 Illumination-invariant Dense Image Descriptors

This thesis proposes the first illumination-invariant dense image descriptor for image classification using a few instances per class (Chapter 4). From the experimental results of Chapter 4, it has been observed that GP has the potential to automatically synthesise a set of formulae to form an LBP-like image descriptor (operating in a pixel-by-pixel fashion), and yet can significantly outperform domain-expert image

descriptors. From Chapter 4, it is found that previous GP methods for multi-class image classification cannot cope with a large number of classes, larger size instances, or having a few instances per class; whereas the developed method in this thesis has successfully addressed all these difficulties. The experiments yielded substantial classification performance improvement of 86% on average compared to the two GP-based methods. Meanwhile, the automatically synthesised image descriptors have outperformed the hand-crafted descriptors in more than 89.3% of the cases. It has also been found that this newly proposed method in this thesis is illumination-invariant through assessing its performance using datasets that comprises instances that were captured under uncontrolled environment (lighting).

8.2.3 Rotation-invariant Dense Image Descriptors

This thesis proposes the first approach to automatically evolving rotation-invariant image descriptors for multi-class image classification tasks, where human-intervention is not needed to handle the design of the required formulae (Chapter 5). Although an instances can appear in any degree of rotation, it has been found from the results in Chapter 5 that GP is capable of evolving rotation-invariant image descriptors using a small sample of the instances in each class. In other words, the proposed method does not need a sample from each and every rotation situation. The results show that automatically evolved descriptors have outperformed the state-of-the-art methods in more than 83.7% of the cases; where over 90.0% average accuracy has been achieved on some datasets.

8.2.4 Parameter Self-tuning GP Representation

Chapter 6 proposes a new GP representation that allows the system to automatically self-tune a parameter through introducing a set of similar nodes with a different number of arguments. Tuning a parameter can be a very time-consuming task that may require executing expensive experiments in order to find a good value as has been observed in Chapter 5. From the experimental results in Chapter 6, it has been found that a significantly fewer runs are needed to evolve the descriptors compared to Chapter 5 where 30 independent runs are needed for each code length. The EID^{ri} method was able to improve upon several benchmark hand-crafted descriptors with

an improvement of 2.9% compared to the best or second best performing methods. The EID^{ri} method has outperformed 8 hand-crafted image descriptors in more than 91.9% of the cases. It has been found that descriptors with a small feature vector (small number of children under the *code* node) cannot achieve the desirable performance, and the system tends towards evolving descriptors with larger and more effective feature vectors (but not over too large).

8.2.5 Transferable Image Descriptors

This thesis investigates and shows the generalizability of those automatically evolved image descriptors to perform classification on other datasets that are different from the source dataset (Chapter 7). Previous works assume that both the training set and test set are drawn from the same distribution, and therefore, models evolved by such methods are not expected to perform well on different datasets. From Chapter 7, it has been observed that an image descriptor automatically constructed by GP-criptor^{ri} on one dataset can (or has the potential to) be directly used on other datasets from the same domain or different domain to that of the source dataset. The 95% confidence intervals from multiple runs of the automatically evolved descriptors demonstrate that they often outperformed the hand-crafted methods. In over 56.1% of the cases, those automatically evolved descriptors on other datasets (source domain) has improved the performance compared to the state-of-the-art descriptors. Furthermore, those automatically evolved descriptors do not require any modification to change the window size, which is not the case in the vast majority of existing descriptors. It has been also found from the experimental results in Chapter 7 that those descriptors can cope very well in most cases with the target datasets even though the source dataset has completely different number of classes, size of instances, rotation, illumination, and type.

8.3 Future Work

Finally, this section provides some possible research directions for future work.

8.3.1 GP and Convolutional Neural Networks

Over the past 30 years, Convolutional Neural Networks (CNNs) became very popular and have received considerable attention in many domains [166]. They have been used to tackle many problems in computer vision and pattern recognition domains [286, 75]. Unlike conventional neural networks, CNNs use a set of shared weights, also called filters or kernels, between layers which potentially reduces the number of weights, i.e., free parameters, that need to be learnt. Moreover, each filter is applied to the entire image (convolution operator) in order to generate a feature map. This makes such an algorithm invariant to object shifting. Another important operator is *pooling*, where a single value of a region on a feature map is returned. This operator has the potential to reduce the size of the generated feature map in the subsequent layer and to select prominent/good features. Clearly, the ability of GP to automatically evolve a model has been successfully presented in this thesis. Furthermore, the automatically evolved image descriptors in this thesis share some properties and objectives, e.g., keypoints detection, feature extraction, and being invariant to image deformations, with CNNs. One of the main issues with CNNs is the requirement to specify the structure of the network beforehand, which imposes some challenges to determine the number of layers, nodes, filters, and how to connect the nodes in adjacent layers. GP can be utilised to all or some of these engineering issues. For example, the coefficients of the filters can be automatically utilised by GP which may improve the performance of CNNs, or those filters can be used as a starting point for further improvements by CNNs which can speed-up the convergence of the network.

8.3.2 Scale-invariant Image Descriptors

This thesis has successfully developed illumination-invariant and rotation-invariant image descriptors. However, handling scale invariance is more challenging and extending the proposed methods to be scale-invariant requires careful consideration. One way to deal with the scale factor could be through using different window sizes simultaneously, similar to SIFT and most of its variants. However, using multiple window sizes means each image needs to be scanned multiple times. This may significantly slow-down the evolutionary process.

8.3.3 Multi-objective Approach for Evolving Image Descriptor

In this thesis, only a single-objective approach has been adopted to develop the different methods. Adopting a multi-objective approach could help in evolving programs that are fast to execute, small in size, and have a good performance.

8.3.4 Colour Image Descriptors

There is much less work on colour image descriptors compared to grey-scale image descriptors. Colours can provide richer information than grey-scale. In this thesis, the proposed methods have been developed to operate on grey-scale images only and extending them to consider colours is important and possible. One way to accomplish this task is by changing the program representation to generate a number of sub feature vectors (one from each channel) and then combine (fusion or simply concatenate) them together to produce the complete feature vector. Another possibility is by generating multiple programs (one for each channel).

8.3.5 Beyond Image Classification

This thesis is concerned mainly with image classification as it is a cornerstone task in computer vision. Image descriptors can be used to perform other tasks that are also important such as object detection and image segmentation. Therefore, both of these two tasks (object detection and image segmentation) can be formulated and performed as image classification by slicing the image being evaluated into a number of sub images (tiles), and then assigning a class label for each of those tiles.

8.3.6 GP for Unsupervised Image Clustering

GP has seldom been used for unsupervised learning, e.g., clustering. All the methods proposed in this thesis are supervised and the class label is needed in order to measure the between-class and within-class distances. Extending GP to consider unlabelled instances could largely help to improve the classification task by borrowing instances from a related domain and transferring the knowledge to

tackle the problem at hand. However, utilising GP to perform clustering is not an easy task and several factors have to be taken into consideration such as the number of clusters and evaluation of clustering results.

8.3.7 Transfer Learning

This thesis only investigates some preliminary works in applying transfer learning in GP for computer vision tasks. However, the investigation carried out in this thesis is not through and more work needs to be done in the future to investigate deeply the concepts of transfer learning in GP. One direction is investigating the impact of reusing some building blocks of those descriptors evolved by GP to build more powerful descriptors. Another direction is to evolve image descriptors using a combination of datasets from different domains and investigate whether this can help improving the performance.

Bibliography

[1] Abdulhamid, F., Neshatian, K., and Zhang, M. Image recognition using genetic programming with loop structures. In *Proceedings of the 26th International Conference on Image and Vision Computing New Zealand* (2011), vol. 29, pp. 553–558.

[2] Abegaz, T., Dozier, G. V., Bryant, K. S., Adams, J., Baker, B., Shelton, J., Ricanek, K., and Woodard, D. L. Genetic-based selection and weighting for LBP, oLBP, and eigenface feature extraction. In *Proceedings of the 22nd Midwest Artificial Intelligence and Cognitive Science Conference* (2011), pp. 221–224.

[3] Abend, K., Harley, T., and Kanal, L. Classification of binary random patterns. *IEEE Transactions on Information Theory 11*, 4 (1965), 538–544.

[4] Ahonen, T., Hadid, A., and Pietikäinen, M. Face recognition with local binary patterns. In *Proceedings of the 8th European Conference on Computer Vision* (2004), T. Pajdla and J. Matas, Eds., vol. 3021 of *Lecture Notes in Computer Science*, Springer, pp. 469–481.

[5] Ahonen, T., Hadid, A., and Pietikäinen, M. Face description with local binary patterns: Application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 12 (2006), 2037–2041.

[6] Al-Sahaf, H., Song, A., Neshatian, K., and Zhang, M. Extracting image features for classification by two-tier genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2012), IEEE, pp. 1–8.

[7] Al-Sahaf, H., Song, A., Neshatian, K., and Zhang, M. Two-tier genetic programming: Towards raw pixel-based image classification. *Expert Systems with Applications 39*, 16 (2012), 12291–12301.

[8] Alba, E., and Troya, J. M. A survey of parallel distributed genetic algorithms. *Complexity 4*, 4 (1999), 31–52.

[9] Albaradei, S., and Wang, Y. Object classification using a semantic hierarchy. In *Proceedings of the 10th International Symposium on Advances in Visual Computing: Part I* (2014), Springer, pp. 228–237.

[10] Albukhanajer, W., Briffa, J., and Jin, Y. Evolutionary multiobjective image feature extraction in the presence of noise. *IEEE Transactions on Cybernetics 45*, 9 (2015), 1757–1768.

[11] Alcantarilla, P. F., Bartoli, A., and Davison, A. J. KAZE features. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI* (2012), Springer, pp. 214–227.

[12] Ali, A., Khan, R., Ullah, I., Khan, A. D., and Munir, A. Minutiae based automatic fingerprint recognition: Machine learning approaches. In *Proceedings of 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing* (2015), University of Auckland, pp. 1148–1153.

[13] Alpaydin, E. *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[14] Alpaydin, E. *Introduction to Machine Learning*, 3rd ed. Adaptive Computation and Machine Learning series. The MIT Press, 2014.

[15] Andreopoulos, A., and Tsotsos, J. K. 50 years of object recognition: Directions forward. *Computer Vision and Image Understanding 117*, 8 (2013), 827–891.

[16] Anuradha Purohit, Narendra S. Choudhari, A. T. Code code bloat bloat problem problem problem in genetic genetic genetic programming

programming programming programming. *International Journal of Scientific and Research Publications 3*, 4 (2013), 1–5.

[17] Ardakany, A. R., Nicolescu, M., and Nicolescu, M. An extended local binary pattern for gender classification. In *Proceedings of 2013 IEEE International Symposium on Multimedia* (2013), IEEE, pp. 315–320.

[18] Atkins, D. L., Neshatian, K., and Zhang, M. A domain independent genetic programming approach to automatic feature extraction for image classification. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2011), IEEE, pp. 238–245.

[19] Awad, A. I. Machine learning techniques for fingerprint identification: A short review. In *Proceedings of the 1st International Conference on Advanced Machine Learning Technologies and Applications* (2012), Springer, pp. 524–531.

[20] Awate, S., and Whitaker, R. Unsupervised, information-theoretic, adaptive image filtering for image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 3 (2006), 364–376.

[21] Back, T., Fogel, D. B., and Michalewicz, Z. *Handbook of Evolutionary Computation*, 1st ed. Computational Intelligence Library. IOP Publishing Ltd., 1997.

[22] Bäck, T., Hammel, U., and Schwefel, H. P. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 3–17.

[23] Ball, W. W. R. *A Short Account of the History of Mathematics*, 4th ed. Macmillan, 1924.

[24] Banterle, F., Corsini, M., Cignoni, P., and Scopigno, R. A low-memory, straightforward and fast bilateral filter through subsampling in spatial domain. *Computer Graphics Forum 31*, 1 (2012), 19–32.

[25] Bart, E., and Ullman, S. Cross-generalization: Learning novel classes from a single example by feature replacement. In *Proceedings of the IEEE*

Computer Society Conference on Computer Vision and Pattern Recognition (2005), IEEE Computer Society, pp. 672–679.

[26] Bay, H., Ess, A., Tuytelaars, T., and van Gool, L. Speeded-up robust features (SURF). *Computer Vision and Image Understanding 110*, 3 (2008), 346–359.

[27] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. *Machine Learning 79*, 1 (2010), 151–175.

[28] Beni, G., and Wang, J. *Swarm Intelligence in Cellular Robotic Systems*. Springer, 1993, pp. 703–712.

[29] Benson, K. A. Evolving finite state machines with embedded genetic programming for automatic target detection within SAR imagery. In *Proceedings of the 2000 Congress on Evolutionary Computation* (2000), IEEE, pp. 1543–1549.

[30] Beyer, H.-G., and Schwefel, H.-P. Evolution strategies: A comprehensive introduction. *Natural Computing 1*, 1 (2002), 3–52.

[31] Bhanu, B., and Lin, Y. Learning composite operators for object detection. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2002), Morgan Kaufmann, pp. 1003–1010.

[32] Bhanu, B., and Lin, Y. Object detection in multi-modal images using genetic programming. *Applied Soft Computing 4*, 2 (2004), 175–201.

[33] Bhanu, B., Yingqiang, L., and Krawiec, K. *Evolutionary Synthesis of Pattern Recognition Systems*. Monographs in Computer Science. Springer, 2006.

[34] Bhowan, U., Zhang, M., and Johnston, M. Genetic programming for image classification with unbalanced data. In *Proceedings of the 24th International Conference Image and Vision Computing New Zealand* (2009), pp. 316–321.

[35] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[36] Bonabeau, E., Dorigo, M., and Theraulaz, G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[37] Boureau, Y.-L., Bach, F., LeCun, Y., and Ponce, J. Learning mid-level features for recognition. In *Proceedings of the 23rd IEEE Conference on Computer Vision and Pattern Recognition* (2010), IEEE, pp. 2559–2566.

[38] Bovolo, F., and Bruzzone, L. A context-sensitive technique based on support vector machines for image classification. In *Proceedings of the 1st International Conference on Pattern Recognition and Machine Intelligence* (2005), Springer, pp. 260–265.

[39] Bratko, A., Cormack, G. V., Filipic, B., Lynam, T. R., and Zupan, B. Spam filtering using statistical data compression models. *Journal of Machine Learning Research 6* (2006), 2673–2698.

[40] Breiman, L. Random forests. *Machine Learning 45*, 1 (2001), 5–32.

[41] Bridle, J. S., and Cox, S. J. RecNorm: Simultaneous normalisation and classification applied to speech recognition. In *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. Morgan-Kaufmann, 1991, pp. 234–240.

[42] Brodatz, P. *Textures: A Photographic Album for Artists and Designers*. Dover Publications, 1999.

[43] Brown, M., and Seales, W. Image restoration of arbitrarily warped documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 10 (2004), 1295–1306.

[44] Browne, W., Kawamura, K., Krichmar, J., Harwin, W., and Wagatsuma, H. Cognitive robotics: new insights into robot and human intelligence by reverse engineering brain functions. *IEEE Robotics Automation Magazine 16*, 3 (2009), 17–18.

[45] Camps-Valls, G., and Bruzzone, L. Kernel-based methods for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing 43*, 6 (2005), 1351–1362.

[46] Canny, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8*, 6 (1986), 679–698.

[47] Castro, L., and Timmis, J. *Artificial Immune Systems: A New Computational Intelligence Paradigm*. Springer, 2002.

[48] Cha, S.-H. Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences 1*, 4 (2007), 300–307.

[49] Chandrasekhar, V., Chen, D. M., Lin, A., Takacs, G., Tsai, S. S., Cheung, N.-M., Reznik, Y. A., Grzeszczuk, R., and Girod, B. Comparison of local feature descriptors for mobile visual search. In *Proceedings of the 17th International Conference on Image Processing* (2010), IEEE, pp. 3885–3888.

[50] Chen, J., Shan, S., He, C., Zhao, G., Pietikäinen, M., Chen, X., and Gao, W. WLD: A robust local image descriptor. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 9 (2010), 1705–1720.

[51] Chen, J., Zhao, G., Kellokumpu, V., and Pietikäinen, M. Combining sparse and dense descriptors with temporal semantic structures for robust human action recognition. In *IEEE International Conference on Computer Vision Workshops* (2011), IEEE, pp. 1524–1531.

[52] Chicotay, S., David, O., and Netanyahu, N. Image registration of very large images via genetic programming. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2014), IEEE, pp. 329–334.

[53] Ciżnicki, M., Kierzynka, M., Kopta, P., Kurowski, K., and Gepner, P. Benchmarking data and compute intensive applications on modern CPU and GPU architectures. In *Proceedings of the International Conference on Computational Science* (2012), Elsevier, pp. 1900–1909.

[54] Cleary, J. G., and Trigg, L. E. K*: An instance-based learner using an entropic distance measure. In *Proceedings of the 12th International Conference on Machine Learning* (1995), Morgan Kaufmann, pp. 108–114.

[55] Coello Coello, C. A., Lamont, G. B., and Veldhuizen, D. A. V. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer, 2006.

[56] Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[57] Cramer, N. L. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms* (1985), L. Erlbaum Associates Inc., pp. 183–187.

[58] Dai, W., Xue, G.-R., Yang, Q., and Yu, Y. Transferring naive bayes classifiers for text classification. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (2007), AAAI Press, pp. 540–545.

[59] Dai, W., Yang, Q., Xue, G.-R., and Yu, Y. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine Learning* (2007), ACM, pp. 193–200.

[60] Dana, K. J., van Ginneken, B., Nayar, S. K., and Koenderink, J. J. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics 18*, 1 (1999), 1–34.

[61] Darwin, C. *On the Origin of Species: By Means of Natural Selection Or the Preservation of Favored Races in the Struggle for Life*. Cosimo Classics, 2007.

[62] Dasgupta, D. *An Overview of Artificial Immune Systems and Their Applications*. Springer, 2000, pp. 3–21.

[63] Dash, N., Priyadarshini, R., and Misra, R. An ANN model to classify multinomial datasets with optimized target using particle swarm optimization technique. In *Proceedings of the International Conference on Computational Intelligence in Data Mining* (2015), Springer, pp. 355–364.

[64] Davis, J., and Domingos, P. Deep transfer via second-order Markov logic. In *Proceedings of the 26th Annual International Conference on Machine Learning* (2009), ACM, pp. 217–224.

[65] Dawood, H., Dawood, H., and Guo, P. Texture image classification with improved weber local descriptor. In *Proceedings of the 13th International Conference on Artificial Intelligence and Soft Computing* (2014), Springer, pp. 684–692.

[66] Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research 7* (2006), 1–30.

[67] Deng, H., Zhang, W., Mortensen, E. N., Dietterich, T. G., and Shapiro, L. G. Principal curvature-based region detector for object recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2007), IEEE Computer Society, pp. 1–8.

[68] Derrac, J., García, S., Molina, D., and Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation 1*, 1 (2011), 3–18.

[69] DigitalCameraWorld. Camera sensors at work: how your digital camera turns light into an image, 2012. [Online]. Available: `http://media.digitalcameraworld.com/wp-content/uploads/sites/123/2012/08/Photography_cheat_sheet_digital_processing1.jpg` (visited on Nov. 12, 2013).

[70] Dorigo, M. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[71] Dorigo, M., and Caro, G. D. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation* (1999), IEEE, pp. 1470–1477.

[72] Dorigo, M., Maniezzo, V., and Colorni, A. Positive feedback as a search strategy. Tech. Rep. 91-016, Politecnico di Milano, Italy, 1991.

[73] Dorigo, M., and Stützle, T. *Ant Colony Optimization*. Bradford Company, 2004.

[74] Downey, C., and Zhang, M. Multiclass object classification for computer vision using linear genetic programming. In *Proceedings of the 24th International Conference on Image and Vision Computing New Zealand* (2009), IEEE, pp. 73–78.

[75] Druzhkov, P. N., and Kustikova, V. D. A survey of deep learning methods and software tools for image classification and object detection. *Pattern Recognition and Image Analysis 26*, 1 (2016), 9–15.

[76] Duda, R., and Hart, P. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[77] Duin, R. P. Small sample size generalization. In *Proceedings of the 9th Scandinavian Conference on Image Analysis* (1995), vol. 2, pp. 957–964.

[78] Duyvesteyn, K., and Kaymak, U. Genetic programming in economic modelling. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2005), IEEE, pp. 1025–1031.

[79] Eberhart, R. C., and Kennedy, J. A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science* (1995), IEEE, pp. 39–43.

[80] Ebner, M., and Zell, A. Evolving a task specific image operator. In *Evolutionary Image Analysis, Signal Processing and Telecommunications*, vol. 1596 of *Lecture Notes in Computer Science*. Springer, 1999, pp. 74–89.

[81] Elhoseiny, M., Huang, S., and Elgammal, A. Weather classification with deep convolutional neural networks. In *Proceedings of the 2015 IEEE International Conference on Image Processing* (2015), IEEE, pp. 3349–3353.

[82] Espejo, P. G., Ventura, S., and Herrera, F. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 40*, 2 (2010), 121–144.

[83] Fan, J., El-Kwae, E. A., Hacid, M.-S., and Liang, F. Novel tracking-based moving object extraction algorithm. *Journal of Electronic Imaging 11*, 3 (2002), 393–403.

[84] Fan, Z.-G., Wang, K.-A., and Lu, B.-L. Feature selection for fast image classification with support vector machines. In *Neural Information Processing*, vol. 3316 of *Lecture Notes in Computer Science*. Springer, 2004, pp. 1026–1031.

[85] Fei-Fei, L. Knowledge transfer in learning to recognize visual object classes. In *Proceedings of IEEE International Conference on Development and Learning* (2006), IEEE, pp. 1–6.

[86] Fei-Fei, L., Fergus, R., and Perona, P. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 4 (2006), 594–611.

[87] Fei-Fei, L., VanRullen, R., Koch, C., and Perona, P. Rapid natural scene categorization in the near absence of attention. *Proceedings of the National Academy of Sciences 99*, 14 (2002), 9596–9601.

[88] Feng, J., and Darrell, T. Learning the structure of deep convolutional networks. In *Proceedings of the 2015 IEEE International Conference on Computer Vision* (2015), IEEE, pp. 2749–2757.

[89] Ferchichi, S., Zidi, S., Laabidi, K., Ksouri, M., and Maouche, S. A new feature extraction method based on clustering for face recognition. In *Engineering Applications of Neural Networks*, L. Iliadis and C. Jayne, Eds., vol. 363 of *IFIP Advances in Information and Communication Technology*. Springer Berlin Heidelberg, 2011, pp. 247–253.

[90] Ferencz, A., Learned-Miller, E., and Malik, J. Building a classification cascade for visual identification from one example. In *Proceedings of the 10th IEEE International Conference on Computer Vision* (2005), vol. 1, pp. 286–293.

[91] Fix, E., and Hodges, J. Discriminatory analysis-nonparametric discrimination: Consistency properties. Tech. Rep. 4, USAF School of Aviation Medicine, Randolf Field, Texas, 1951.

[92] Fogel, L., Owens, A., and Walsh, M. *Artificial Intelligence Through Simulated Evolution*. Wiley, 1966.

[93] Fogel, L. J. Autonomous automata. *Industrial Research 4* (1962), 14–19.

[94] Fogel, L. J. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. Wiley, 1999.

[95] Forsyth, D., and Ponce, J. *Computer Vision: A Modern Approach*. Pearson Education, Limited, 2011.

[96] Forsyth, R. BEAGLE-A darwinian approach to pattern recognition. *Kybernetes 10*, 3 (1981), 159–166.

[97] Freund, Y., and Schapire, R. E. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning* (1996), Morgan Kaufmann, pp. 148–156.

[98] Frintrop, S. Attentive robots. In *From Human Attention to Computational Attention: A Multidisciplinary Approach*, vol. 10 of *Springer Series in Cognitive and Neural Systems*. Springer, 2016, pp. 413–443.

[99] Fu, W., Johnston, M., and Zhang, M. Automatic construction of invariant features using genetic programming for edge detection. In *Proceedings of the 25th Australasian Joint Conference on Artificial Intelligence*, vol. 7691 of *Lecture Notes in Computer Science*. Springer, 2012, pp. 144–155.

[100] Fu, W., Johnston, M., and Zhang, M. Distribution-based invariant feature construction using genetic programming for edge detection. *Soft Computing 19*, 8 (2015), 2371–2389.

[101] Galerne, B., Gousseau, Y., and Morel, J. M. Random phase textures: Theory and synthesis. *IEEE Transactions on Image Processing 20*, 1 (2011), 257–267.

[102] Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine Learning 63*, 1 (2006), 3–42.

[103] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., 1989.

[104] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing*, 2nd ed. Prentice Hall, 2002.

[105] Gould, A. J. Visual perception and attention, 2012. [Online]. Available: `https://wiki.ucl.ac.uk/display/UCLICACS/Visual+perception+and+attention` (visited on Nov. 12, 2013).

[106] Grauman, K., and Leibe, B. *Visual Object Recognition*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool publishers, 2011.

[107] Green, R. D., Burne, J. A., and Guan, L. Video analysis of gait for diagnosing movement disorders. *Journal of Electronic Imaging 9*, 1 (2000), 16–21.

[108] Gruau, F. On using syntactic constraints with genetic programming. In *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. MIT Press, 1996, ch. 19, pp. 377–394.

[109] Grundland, M., and Dodgson, N. A. Decolorize: Fast, contrast enhancing, color to grayscale conversion. *Pattern Recognition 40*, 11 (2007), 2891–2896.

[110] Guo, Z., Zhang, L., and Zhang, D. A completed modeling of local binary pattern operator for texture classification. *IEEE Transactions on Image Processing 19*, 6 (2010), 1657–1663.

[111] Guo, Z., Zhang, L., and Zhang, D. Rotation invariant texture classification using LBP variance (LBPV) with global matching. *Pattern Recognition 43*, 3 (2010), 706–719.

[112] Guruvareddy, A., Sri Rama Krishna, K., and Giri Prasad, M. N. An effective local contrast enhancement technique by blending of local statistics

and genetic algorithm. *Pattern Recognition and Image Analysis 21*, 4 (2011), 606–615.

[113] Gustafson, S., Burke, E., and Krasnogor, N. On improving genetic programming for symbolic regression. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2005), vol. 1, IEEE, pp. 912–919.

[114] Guven, A. Linear genetic programming for time-series modelling of daily flow rate. *Journal of Earth System Science 118*, 2 (2009), 137–146.

[115] Guyon, I., and Elisseeff, A. An introduction to variable and feature selection. *The Journal of Machine Learning Research 3* (2003), 1157–1182.

[116] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter 11*, 1 (2009), 10–18.

[117] Haralick, R., Shanmugam, K., and Dinstein, I. Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics SMC-3*, 6 (1973), 610–621.

[118] Harding, S., Leitner, J., and Schmidhuber, J. Cartesian genetic programming for image processing. In *Genetic Programming Theory and Practice X*, Genetic and Evolutionary Computation. Springer, 2013, pp. 31–44.

[119] Harris, C., and Stephens, M. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference* (1988), Alvey Vision Club, pp. 147–151.

[120] Harris, Z. Distributional structure. *Word 10*, 2/3 (1954), 146–162.

[121] Hartigan, J., and Wong, M. Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics 28*, 1 (1979), 100–108.

[122] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2 ed. Springer Series in Statistics. Springer, 2009.

[123] Heisele, B., Poggio, T., and Pontil, M. Face detection in still gray images. A.I. memo 1687, Center for Biological and Computational Learning, MIT, 2000.

[124] Hermosilla, G., Gallardo, F., Farias, G., and Martin, C. S. Fusion of visible and thermal descriptors using genetic algorithms for face recognition systems. *Sensors 15*, 8 (2015), 17944–17962.

[125] Hindmarsh, S., Andreae, P., and Zhang, M. Genetic programming for improving image descriptors generated using the scale-invariant feature transform. In *Proceedings of the 27th International Conference on Image and Vision Computing New Zealand* (2012), ACM, pp. 85–90.

[126] Hoai, N. X., McKay, R. I., and Abbass, H. A. Tree adjoining grammars, language bias, and genetic programming. In *Proceedings of the 6th European Conference on Genetic Programming* (2003), C. Ryan, T. Soule, M. Keijzer, E. P. K. Tsang, R. Poli, and E. Costa, Eds., vol. 2610 of *Lecture Notes in Computer Science*, Springer, pp. 335–344.

[127] Holland, J. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.

[128] Holland, J. H. Outline for a logical theory of adaptive systems. *Journal of the ACM 9*, 3 (1962), 297–314.

[129] Holland, J. H., Booker, L. B., Colombetti, M., Dorigo, M., Goldberg, D. E., Forrest, S., Riolo, R. L., Smith, R. E., Lanzi, P. L., Stolzmann, W., and Wilson, S. W. *What Is a Learning Classifier System?* Springer, 2000, pp. 3–32.

[130] Holland, J. H., and Reitman, J. S. Cognitive systems based on adaptive algorithms. *SIGART Bulletin*, 63 (1977), 49–49.

[131] Holmes, G., Pfahringer, B., Kirkby, R., Frank, E., and Hall, M. Multiclass alternating decision trees. In *Proceedings of the 13th European Conference on Machine Learning* (2002), Springer, pp. 161–172.

[132] Humphrey, E. J., Bello, J. P., and Lecun, Y. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In *Proceedings of the 13th International Society for Music Information Retrieval Conference* (2012), pp. 403–408.

[133] Humphrey, E. J., Bello, J. P., and LeCun, Y. Feature learning and deep architectures: new directions for music informatics. *Journal of Intelligent Information Systems 41*, 3 (2013), 461–481.

[134] Jain, A., and Chandrasekaran, B. Dimensionality and sample size considerations in pattern recognition practice. In *Proceedings of the Classification Pattern Recognition and Reduction of Dimensionality* (1982), vol. 2, Elsevier, pp. 835–855.

[135] Joachims, T. Transductive inference for text classification using support vector machines. In *Proceedings of the 16th International Conference on Machine Learning* (1999), Morgan Kaufmann, pp. 200–209.

[136] John, G. H., Kohavi, R., and Pfleger, K. Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning* (1994), W. W. Cohen and H. Hirsh, Eds., Morgan Kaufmann, pp. 121–129.

[137] John, G. H., and Langley, P. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence* (1995), Morgan Kaufmann, pp. 338–345.

[138] Johnson, S. *Stephen Johnson on Digital Photography*. O'Reilly Media, Incorporated, 2006.

[139] Jong, K. A. D. *Evolutionary computation: A unified approach*. MIT Press, 2006.

[140] Kanan, C., and Cottrell, G. W. Color-to-grayscale: Does the method matter in image recognition? *PLoS ONE 7*, 1 (2012), 1–7.

[141] Kandaswamy, C., Silva, L. M., Alexandre, L. A., Sousa, R., Santos, J. M., and de Sá, J. M. Improving transfer learning accuracy by reusing

stacked denoising autoencoders. In *Proceedings of 2014 IEEE International Conference on Systems, Man, and Cybernetics* (2014), IEEE, pp. 1380–1387.

[142] Karaboga, D., and Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing 8*, 1 (2008), 687–697.

[143] Ke, Y., and Sukthankar, R. PCA-SIFT: A more distinctive representation for local image descriptors. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2004), IEEE Computer Society, pp. 506–513.

[144] Keerthi, S., and Lin, C.-J. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation 15*, 7 (2003), 1667–1689.

[145] Kennedy, J., Eberhart, R., and Shi, Y. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[146] Kinnear, K., Spector, L., and Angeline, P. *Advances in Genetic Programming*, 3rd ed. MIT Press, 1999.

[147] Kinzett, D., Johnston, M., and Zhang, M. Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence 2*, 4 (2009), 151–168.

[148] Kira, K., and Rendell, L. A. A practical approach to feature selection. In *Proceedings of the 9th International Workshop on Machine Learning* (1992), Morgan Kaufmann, pp. 249–256.

[149] Kohavi, R. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining* (1996), AAAI Press, pp. 202–207.

[150] Koller, D., and Sahami, M. Toward optimal feature selection. In *Proceedings of the 13th International Conference on Machine Learning* (1996), L. Saitta, Ed., Morgan Kaufmann, pp. 284–292.

[151] Kolmogorov, V., and Zabih, R. Multi-camera scene reconstruction via graph cuts. In *Proceedings of the 7th European Conference on Computer Vision-Part III* (2002), Springer-Verlag, pp. 82–96.

[152] Koza, J. R. A genetic approach to the truck backer upper problem and the inter-twined spiral problem. In *Proceedings of the International Joint Conference on Neural Networks, Volume IV* (1992), IEEE Press, pp. 310–318.

[153] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[154] Koza, J. R., Andre, D., Bennett, F. H., and Keane, M. A. *Genetic Programming III: Darwinian Invention & Problem Solving*. Morgan Kaufmann, 1999.

[155] Krig, S. *Computer Vision Metrics: Survey, Taxonomy, and Analysis*, 1st ed. Apress, 2014.

[156] Kylberg, G. Centre for Image Analysis. [Online]. Available: `http://www.cb.uu.se/~gustaf/KylbergSintornRotation/` (visited on Jul. 26, 2016).

[157] Kylberg, G. The Kylberg texture dataset v. 1.0. External report (Blue series) 35, Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, Uppsala, Sweden, 2011.

[158] Kylberg, G. *Automatic Virus Identification using TEM: Image Segmentation and Texture Analysis*. PhD thesis, Division of Visual Information and Interaction, Uppsala University, Uppsala, Sweden, 2014.

[159] Lake, B. M., Salakhutdinov, R., Gross, J., and Tenenbaum, J. B. One shot learning of simple visual concepts. In *Proceedings of the 33rd Annual Conference of the Cognitive Science Society* (2011), Cognitive Science Society, pp. 2568–2573.

[160] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science 350*, 6266 (2015), 1332–1338.

[161] Langdon, W. B., Modat, M., Petke, J., and Harman, M. Improving 3D medical image registration CUDA software with genetic programming.

In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation* (2014), ACM, pp. 951–958.

[162] Langdon, W. B., and Poli, R. Why ants are hard. In *Proceedings of the 3rd Annual Conference on Genetic Programming* (1998), J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds., Morgan Kaufmann, pp. 193–201.

[163] Langdon, W. B., Poli, R., McPhee, N. F., and Koza, J. R. Genetic programming: An introduction and tutorial, with a survey of techniques and applications. In *Computational Intelligence: A Compendium*, J. Fulcher and L. Jain, Eds., vol. 115 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 2008, pp. 927–1028.

[164] Larochelle, H., and Bengio, Y. Classification using discriminative restricted Boltzmann machines. In *Proceedings of the 25th International Conference on Machine Learning* (2008), ACM, pp. 536–543.

[165] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[166] Lecun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Müller, U., Sackinger, E., Simard, P., and Vapnik, V. Comparison of learning algorithms for handwritten digit recognition. In *Proceedings of the International Conference on Artificial Neural Networks* (1995), F. Fogelman and P. Gallinari, Eds., Springer-Verlag, pp. 53–60.

[167] Li, X., and Ciesielski, V. Using loops in genetic programming for a two class binary image classification problem. In *Proceedings of the 17th Australian Joint Conference on Artificial Intelligence* (2004), G. I. Webb and X. Yu, Eds., vol. 3339 of *Lecture Notes in Computer Science*, Springer, pp. 898–909.

[168] Li, Y., Ma, J., and Zhao, Q. Two improvements in genetic programming for image classification. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2008), pp. 2492–2497.

[169] Liang, Y., Zhang, M., and Browne, W. Image segmentation: A survey of methods based on evolutionary computation. In *Simulated Evolution and Learning*, vol. 8886 of *Lecture Notes in Computer Science*. Springer, 2014, pp. 847–859.

[170] Liang, Y., Zhang, M., and Browne, W. A supervised figure-ground segmentation method using genetic programming. In *Proceedings of the 18th European Conference on the Applications of Evolutionary Computation* (2015), vol. 9028 of *Lecture Notes in Computer Science*, Springer, pp. 491–503.

[171] Liao, S., Gao, Y., Oto, A., and Shen, D. Representation learning: A unified deep learning framework for automatic prostate MR segmentation. In *Proceedings of the 16th International Conference on Medical Image Computing and Computer-Assisted Intervention* (2013), Springer, pp. 254–261.

[172] Lim, K.-L., and Galoogahi, H. Shape classification using local and global features. In *Proceedings of the 4th Pacific-Rim Symposium on Image and Video Technology* (2010), IEEE Computer Society, pp. 115–120.

[173] Lin, J.-Y., Ke, H.-R., Chien, B.-C., and Yang, W.-P. Classifier design with feature selection and feature extraction using layered genetic programming. *Expert Systems with Applications 34*, 2 (2008), 1384–1393.

[174] Lindeberg, T. Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention. *International Journal of Computer Vision 11*, 3 (1993), 283–318.

[175] Lindeberg, T. Feature detection with automatic scale selection. *International Journal of Computer Vision 30*, 2 (1998), 79–116.

[176] Lisin, D. A., Mattar, M. A., Blaschko, M. B., Learned-Miller, E. G., and Benfield, M. C. Combining local and global image features for object class recognition. In *Proceedings of the IEEE Computer Society*

Conference on Computer Vision and Pattern Recognition (2005), vol. 3, IEEE Computer Society, pp. 47–55.

[177] Liu, H., and Motoda, H. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academicpublishers, 1998.

[178] Liu, L., Shao, L., Li, X., and Lu, K. Learning spatio-temporal representations for action recognition: A genetic programming approach. *IEEE Transactions on Cybernetics 46*, 1 (2016), 158–172.

[179] Liu, L., Shao, L., and Rockett, P. Genetic programming-evolved spatio-temporal descriptor for human action recognition. In *Proceedings of the British Machine Vision Conference* (2012), BMVA Press, pp. 18.1–18.12.

[180] Liu, L., Zhao, L., Long, Y., Kuang, G., and Fieguth, P. Extended local binary patterns for texture classification. *Image and Vision Computing 30*, 2 (2012), 86–99.

[181] Liu, Y., Wang, G., Chen, H., Dong, H., Zhu, X., and Wang, S. An improved particle swarm optimization for feature selection. *Journal of Bionic Engineering 8*, 2 (2011), 191–200.

[182] Loderer, M., and Pavlovičová, J. Optimization of LBP parameters. In *Proceedings of the 56th International Symposium on Electronics in Marine* (2014), IEEE, pp. 1–4.

[183] Loukas, C., Kostopoulos, S., Tanoglidi, A., Glotsos, D., Sfikas, C., and Cavouras, D. Breast cancer characterization based on image classification of tissue sections visualized under low magnification. *Computational and Mathematical Methods in Medicine 2013* (2013), 1–7.

[184] Loveard, T., and Ciesielski, V. Representing classification problems in genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2001), vol. 2, IEEE, pp. 1070–1077.

[185] Lowe, D. G. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision* (1999), IEEE, pp. 1150–1157.

[186] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (2004), 91–110.

[187] Lu, D., and Weng, Q. A survey of image classification methods and techniques for improving classification performance. *International Journal of Remote Sensing 28*, 5 (2007), 823–870.

[188] Lu, J., Tan, Y.-P., and Wang, G. Discriminative multimanifold analysis for face recognition from a single training sample per person. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 1 (2013), 39–51.

[189] Lu, X., Lin, Z., Jin, H., Yang, J., and Wang, J. Z. RAPID: Rating pictorial aesthetics using deep learning. In *Proceedings of the 22nd ACM International Conference on Multimedia* (2014), ACM, pp. 457–466.

[190] Luke, S. *Essentials of Metaheuristics*, second ed. Lulu, 2013. [Online] Available: `http://cs.gmu.edu/~sean/book/metaheuristics/`.

[191] Martin, B. Instance-based learning: Nearest neighbor with generalization. Master's thesis, University of Waikato, Hamilton, New Zealand, 1995.

[192] Matas, J., Chum, O., Urban, M., and Pajdla, T. Robust wide baseline stereo from maximally stable extremal regions. In *Proceedings of the British Machine Vision Conference* (2002), P. L. Rosin and A. D. Marshall, Eds., British Machine Vision Association, pp. 384–393.

[193] Matsugu, M., Mori, K., Mitari, Y., and Kaneda, Y. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks 16*, 5 (2003), 555–559.

[194] McKay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., and O'Neill, M. Grammar-based genetic programming: A survey. *Genetic Programming and Evolvable Machines 11*, 3-4 (2010), 365–396.

[195] Mclachlan, G. J. *Discriminant Analysis and Statistical Pattern Recognition*. Wiley, 2004.

[196] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd ed. Springer-Verlag, 1994.

[197] Michalski, R. S. *A Theory and Methodology of Inductive Learning*. Springer, 1983, pp. 83–134.

[198] Mihalkova, L., and Mooney, R. J. Transfer learning by mapping with minimal target data. In *Proceedings of the AAAI-08 Workshop on Transfer Learning For Complex Tasks* (2008), AAAI Press, pp. 31–36.

[199] Mikolajczyk, K., and Matas, J. Improving descriptors for fast tree matching by optimal linear projection. In *Proceedings of the 11th International Conference on Computer Vision* (2007), IEEE, pp. 1–8.

[200] Miller, A. *Subset Selection in Regression*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 2002.

[201] Miller, E., Matsakis, N., and Viola, P. Learning from one example through shared densities on transforms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2000), vol. 1, pp. 464–471.

[202] Miller, J. F., and Harding, S. L. Cartesian genetic programming. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation* (2008), ACM, pp. 2701–2726.

[203] Miller, J. F., and Smith, S. L. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation 10*, 2 (2006), 167–174.

[204] Mirjalili, S., Mirjalili, S. M., and Lewis, A. Grey wolf optimizer. *Advances in Engineering Software 69* (2014), 46–61.

[205] Mizuno, S., Yamaguchi, T., Fukushima, A., Matsuyama, Y., and Ohashi, Y. Overlap coefficient for assessing the similarity of pharmacokinetic data between ethnically different populations. *Clinical Trials 2*, 2 (2005), 174–181.

[206] Molina, L. C., Belanche, L., and Nebot, A. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of the IEEE International Conference on Data Mining* (2002), IEEE Computer Society, pp. 306–313.

[207] Montana, D. J. Strongly typed genetic programming. *Evolutionary Computation 3*, 2 (1995), 199–230.

[208] Moravec, H. Obstacle avoidance and navigation in the real world by a seeing robot rover. Tech. Rep. CMU-RI-TR-80-03, Robotics Institute, Stanford University, 1980.

[209] Morris, T. *Computer Vision and Image Processing.* Cornerstones of computing. Palgrave Macmillan, 2004.

[210] Munder, S., and Gavrila, D. M. An experimental study on pedestrian classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 11 (2006), 1863–1868.

[211] Muni, D. P., Pal, N. R., and Das, J. Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man, and Cybernetics, Part A 36*, 1 (2006), 106–117.

[212] Nanni, L., Brahnam, S., and Lumini, A. A simple method for improving local binary patterns by considering non-uniform patterns. *Pattern Recognition 45*, 10 (2012), 3844–3852.

[213] Neshatian, K., Zhang, M., and Andreae, P. A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. *IEEE Transections Evolutionary Computation 16*, 5 (2012), 645–661.

[214] Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. Dynamic multi-objective job shop scheduling: A genetic programming approach. In *Automated Scheduling and Planning*, A. S. Uyar, E. Ozcan, and N. Urquhart, Eds., vol. 505 of *Studies in Computational Intelligence*. Springer, 2013, pp. 251–282.

[215] Nixon, M., and Aguado, A. *Feature Extraction and Image Processing for Computer Vision*. Academic Press, 2012.

[216] Ojala, T., Mäenpää, T., Pietikäinen, M., Viertola, J., Kyllonen, J., and Huovinen, S. Outex - new framework for empirical evaluation of texture analysis algorithms. In *Proceedings of the 16th International Conference on Pattern Recognition* (2002), vol. 1, IEEE, pp. 701–706.

[217] Ojala, T., Pietikäinen, M., and Harwood, D. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In *Proceedings of the 12th International Conference on Pattern Recognition* (1994), vol. 1, IEEE, pp. 582–585.

[218] Ojala, T., Pietikäinen, M., and Harwood, D. A comparative study of texture measures with classification based on feature distributions. *Pattern Recognition 29*, 1 (1996), 51–59.

[219] Ojala, T., Pietikäinen, M., and Mäenpää, T. Gray scale and rotation invariant texture classification with local binary patterns. In *Proceedings of the 6th European Conference on Computer Vision* (2000), no. 1842 in Lecture Notes in Computer Science, Springer, pp. 404–420.

[220] Ojala, T., Pietikäinen, M., and Mäenpää, T. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 7 (2002), 971–987.

[221] Olague, G., and Trujillo, L. A genetic programming approach to the design of interest point operators. In *Bio-inspired Hybrid Intelligent Systems for Image Analysis and Pattern Recognition*, vol. 256 of *Studies in Computational Intelligence*. Springer, 2009, pp. 49–65.

[222] Olague, G., and Trujillo, L. Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image and Vision Computing 29*, 7 (2011), 484–498.

[223] Oltean, M., Grosan, C., Diosan, L., and Mihaila, C. Genetic programming with linear representation: a survey. *International Journal on Artificial Intelligence Tools 18*, 2 (2009), 197–238.

[224] O'Neill, M., and Ryan, C. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academicpublishers, 2003.

[225] O'Neill, M., Vanneschi, L., Gustafson, S., and Banzhaf, W. Open issues in genetic programming. *Genetic Programming and Evolvable Machines 11*, 3 (2010), 339–363.

[226] Ortiz, R. FREAK: Fast retina keypoint. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), IEEE Computer Society, pp. 510–517.

[227] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 10 (2010), 1345–1359.

[228] Parker, J. R. *Algorithms for image processing and computer vision*, 2nd ed. Wiley, 2010.

[229] Pechenizkiy, M. The impact of feature extraction on the performance of a classifier: kNN, Naïve Bayes and C4.5. In *Proceedings of the 18th Conference of the Canadian Society for Computational Studies of Intelligence: Advances in Artificial Intelligence* (2005), vol. 3501 of *Lecture Notes in Computer Science*, Springer, pp. 268–279.

[230] Peng, Y., Wu, Z., and Jiang, J. A novel feature selection approach for biomedical data classification. *Journal of Biomedical Informatics 43*, 1 (2010), 15–23.

[231] Perez, C., and Olague, G. Learning invariant region descriptor operators with genetic programming and the F-measure. In *Proceedings of the 19th International Conference on Pattern Recognition* (2008), IEEE, pp. 1–4.

[232] Perez, C., and Olague, G. Evolving local descriptor operators through genetic programming. In *Applications of Evolutionary Computing*, vol. 5484 of *Lecture Notes in Computer Science*. Springer, 2009, pp. 414–419.

[233] Perez, C. B., and Olague, G. Evolutionary learning of local descriptor operators for object recognition. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation* (2009), ACM, pp. 1051–1058.

[234] Perez, C. B., and Olague, G. Genetic programming as strategy for learning image descriptor operators. *Intelligent Data Analysis 17*, 4 (2013), 561–583.

[235] Pietroni, N., Cignoni, P., Otaduy, M., and Scopigno, R. Solid-texture synthesis: A survey. *IEEE Computer Graphics and Applications 30*, 4 (2010), 74–89.

[236] Piszcz, A., and Soule, T. A survey of mutation techniques in genetic programming. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (2006), ACM, pp. 951–952.

[237] Platt, J. C. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods* (1999), B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds., MIT Press, pp. 185–208.

[238] Poli, R. Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. Tech. Rep. CSRP-96-14, University of Birmingham, School of Computer Science, 1996. Presented at 3rd International Conference on Artificial Neural Networks and Genetic Algorithms.

[239] Poli, R. A simple but theoretically-motivated method to control bloat in genetic programming. In *Proceedings of the 6th European Conference on Genetic Programming* (2003), Springer, pp. 204–217.

[240] Poli, R. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications 2008* (2008), 1–10.

[241] Poli, R., and Cagnoni, S. Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement. In *Genetic Programming 1997: Proceedings of the Second Annual Conference* (1997), Morgan Kaufmann, pp. 269–277.

[242] Poli, R., Kennedy, J., and Blackwell, T. Particle swarm optimization. *Swarm Intelligence 1*, 1 (2007), 33–57.

[243] Poli, R., Langdon, W. B., and McPhee, N. F. *A Field Guide to Genetic Programming*. Published via `http://lulu.com`, 2008. (with contributions by J. R. Koza).

[244] Potter, M. C. Short-term conceptual memory for pictures. *Journal of Experimental Psychology. Human learning and Memory 2*, 5 (1976), 509–522.

[245] Price, K., Storn, R. M., and Lampinen, J. A. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005.

[246] Quinlan, J. R. Induction of decision trees. *Machine Learning 1*, 1 (1986), 81–106.

[247] Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. *Dataset Shift in Machine Learning*. The MIT Press, 2009.

[248] Ramamoorthy, S., Kirubakaran, R., and Subramanian, R. Texture feature extraction using MGRLBP method for medical image classification. In *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems*, vol. 324 of *Advances in Intelligent Systems and Computing*. Springer, 2015, pp. 747–753.

[249] Ramirez, R., and Puiggros, M. A genetic programming approach to feature selection and classification of instantaneous cognitive states. In *Applications of Evolutionary Computing*, M. Giacobini, Ed., vol. 4448 of *Lecture Notes in Computer Science*. Springer, 2007, pp. 311–319.

[250] Rao, D. H., and Panduranga, P. A survey on image enhancement techniques: Classical spatial filter, neural network, cellular neural network,

and fuzzy filter. In *Proceedings of the IEEE International Conference on Industrial Technology* (2006), pp. 2821–2826.

[251] Rassem, T., and Khoo, B. E. Completed local ternary pattern for rotation invariant texture classification. *The Scientific World Journal 2014* (2014), 1–10.

[252] Raudys, S. J., and Jain, A. K. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 3 (1991), 252–264.

[253] Rechenberg, I. *Evolutionsstrategie*. Frommann-Holzboog, 1994.

[254] Rifkin, R., and Klautau, A. In defense of one-vs-all classification. *The Journal of Machine Learning Research 5* (2004), 101–141.

[255] Rodner, E., and Denzler, J. Learning with few examples by transferring feature relevance. In *Proceedings of the 31st DAGM Symposium on Pattern Recognition* (2009), vol. 5748 of *Lecture Notes in Computer Science*, Springer, pp. 252–261.

[256] Rodner, E., and Denzler, J. Learning with few examples for binary and multiclass classification using regularization of randomized trees. *Pattern Recognition Letters 32*, 2 (2011), 244–251.

[257] Rosenblatt, F. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.

[258] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *Proceedings of the 9th European Conference on Computer Vision-Part I* (2006), Springer, pp. 430–443.

[259] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. MIT Press, 1986, pp. 318–362.

[260] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[261] Ryan, C., Fitzgerald, J., Krawiec, K., and Medernach, D. Image classification with genetic programming: Building a stage 1 computer aided detector for breast cancer. In *Handbook of Genetic Programming Applications* (2015), Springer, pp. 245–287.

[262] Saenko, K., Kulis, B., Fritz, M., and Darrell, T. Adapting visual category models to new domains. In *Proceedings of the 11th European Conference on Computer Vision*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Springer, 2010, pp. 213–226.

[263] Salakhutdinov, R., Tenenbaum, J. B., and Torralba, A. One-shot learning with a hierarchical nonparametric Bayesian model. In *ICML Unsupervised and Transfer Learning* (2012), vol. 27 of *JMLR Proceedings*, JMLR.org, pp. 195–206.

[264] Santini, F., and Rucci, M. Depth perception in an anthropomorphic robot that replicates human eye movements. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation* (2006), IEEE, pp. 1293–1298.

[265] Santini, F., and Rucci, M. Active estimation of distance in a robotic system that replicates human eye movement. *Robotics and Autonomous Systems 55*, 2 (2007), 107–121.

[266] Schneiderman, H., and Kanade, T. A statistical method for 3D object detection applied to faces and cars. In *Proceedings of Computer Vision and Pattern Recognition* (2000), IEEE Computer Society, pp. 1746–1759.

[267] Segaran, T. *Programming Collective Intelligence*, 1st ed. O'Reilly, 2007.

[268] Shao, L., Liu, L., and Li, X. Feature learning for image classification via multiobjective genetic programming. *IEEE Transactions on Neural Networks and Learning Systems 25*, 7 (2014), 1359–1371.

[269] Shi, H., Li, C., Xie, Y., and Zou, Q. Identification number recognition based on random forests. In *Proceedings of 2012 International Conference on Software Engineering and Knowledge Engineering: Theory and Practice* (2012), Springer, pp. 745–753.

[270] Sibson, R. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal 16*, 1 (1973), 30–34.

[271] Sinha, A., Banerji, S., and Liu, C. New color GPHOG descriptors for object and scene image classification. *Machine Vision and Applications 25*, 2 (2014), 361–375.

[272] Sivic, J., and Zisserman, A. Efficient visual search of videos cast as text retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 4 (2009), 591–606.

[273] Skinner, C., Riddle, P. J., and Triggs, C. Mathematics prevents bloat [genetic programming]. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (2005), IEEE, pp. 390–395.

[274] Smart, W., and Zhang, M. Using genetic programming for multiclass classification by simultaneously solving component binary classification problems. In *Proceedings of the 8th European Conference on Genetic Programming* (2005), Springer, pp. 227–239.

[275] Smart, W. R., and Zhang, M. Classification strategies for image classification in genetic programming. In *Proceedings of the 18th International Conference on Image and Vision Computing New Zealand* (2003), Massey University, pp. 402–407.

[276] Smith, M. G., and Bull, L. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines 6*, 3 (2005), 265–281.

[277] Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel Distributed Processing. Volume 1: Foundations*, D. E. Rumelhart, J. L. McClelland, and PDP Research Group, Eds. MIT Press, 1986, ch. 5, pp. 282–317.

[278] Snavely, N., Simon, I., Goesele, M., Szeliski, R., and Seitz, S. M. Scene reconstruction and visualization from community photo collections. *Proceedings of the IEEE 98*, 8 (2010), 1370–1390.

[279] Sobel, I., and Feldman, G. A 3×3 isotropic gradient operator for image processing. Presented at a talk at the Stanford Artificial Project, 1968.

[280] Song, A., Chen, D., and Zhang, M. Contribution based bloat control in genetic programming. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation* (2010), IEEE, pp. 1–8.

[281] Song, A., and Ciesielski, V. Texture analysis by genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2004), IEEE Press, pp. 2092–2099.

[282] Song, A., and Ciesielski, V. Texture segmentation by genetic programming. *Evolutionary Computation 16*, 4 (2008), 461–481.

[283] Song, A., Loveard, T., and Ciesielski, V. Towards genetic programming for texture classification. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence* (2001), vol. 2256 of *Lecture Notes in Computer Science*, Springer, pp. 461–472.

[284] Song, Z., Chen, Q., Huang, Z., Hua, Y., and Yan, S. Contextualizing object detection and classification. In *Proceedings of the 24th IEEE Conference on Computer Vision and Pattern Recognition* (2011), IEEE, pp. 1585–1592.

[285] Sonka, M., Hlavac, V., and Boyle, R. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007.

[286] Srinivas, S., Sarvadevabhatla, R. K., Mopuri, K. R., Prabhu, N., Kruthiventi, S. S. S., and Babu, R. V. A taxonomy of deep convolutional neural nets for computer vision. *Frontiers in Robotics and AI 2* (2016), 36–54.

[287] Stephenson, M., O'Reilly, U.-M., Martin, M. C., and Amarasinghe, S. Genetic programming applied to compiler heuristic optimization. In *Proceedings of the 6th European Conference on Genetic Programming* (2003), Springer-Verlag, pp. 238–253.

[288] Stockman, G., and Shapiro, L. G. *Computer Vision*, 1st ed. Prentice Hall, 2001.

[289] Storn, R., and Price, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11*, 4 (1997), 341–359.

[290] Stringer, H., and Wu, A. S. Winnowing wheat from chaff: The chunking GA. In *Proceedings of the 6th Annual Conference on Genetic and Evolutionary Computation* (2004), Springer, pp. 198–209.

[291] Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning 3*, 1 (1988), 9–44.

[292] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. Bradford, 1998.

[293] Szeliski, R. *Computer Vision: Algorithms and Applications*, 1st ed. Springer-Verlag, 2010.

[294] Tabbone, S. Corner detection using Laplacian of Gaussian operator. In *Proceedings of the 8th Scandinavian Conference on Image Analysis* (1993), Norwegian Society for Image Processing and Pattern Recognition, pp. 1055–1059.

[295] Tackett, W. A. Genetic programming for feature discovery and image discrimination. In *Proceedings of the 5th International Conference on Genetic Algorithms* (1993), pp. 303–311.

[296] Tax, D., and Duin, R. Using two-class classifiers for multiclass classification. In *Proceedings of the 16th International Conference on Pattern Recognition* (2002), vol. 2, pp. 124–127.

[297] Thorpe, S., Fize, D., and Marlot, C. Speed of processing in the human visual system. *Nature 381*, 6582 (1996), 520–522.

[298] Trenn, S. Multilayer perceptrons: Approximation order and necessary number of hidden units. *IEEE Transactions on Neural Networks 19*, 5 (2008), 836–844.

[299] Trujillo, L., Legrand, P., Olague, G., and Pérez, C. Optimization of the Hölder image descriptor using a genetic algorithm. In *Proceedings*

of the 12th Annual Conference on Genetic and Evolutionary Computation (2010), ACM, pp. 1147–1154.

[300] Trujillo, L., and Olague, G. Synthesis of interest point detectors through genetic programming. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (2006), ACM, pp. 887–894.

[301] Trujillo, L., and Olague, G. Automated design of image operators that detect interest points. *Evolutionary Computation 16*, 4 (2008), 483–507.

[302] Trzcinski, T., Christoudias, M., Lepetit, V., and Fua, P. Learning image descriptors with the boosting-trick. In *Advances in Neural Information Processing Systems 25*. Curran Associates, 2012, pp. 269–277.

[303] Tuytelaars, T., and Mikolajczyk, K. Local invariant feature detectors: A survey. *Foundations and Trends in Computer Graphics and Vision 3*, 3 (2008), 177–280.

[304] Ullmann, J., and Kidd, P. Recognition experiments with typed numerals from envelopes in the mail. *Pattern Recognition 1*, 4 (1969), 273–289.

[305] Ulusoy, I., and Bishop, C. M. Comparison of generative and discriminative techniques for object detection and classification. In *Toward Category-Level Object Recognition* (2006), J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, Eds., vol. 4170 of *Lecture Notes in Computer Science*, Springer, pp. 173–195.

[306] Umbaugh, S. E. *Computer Vision and Image Processing: A Practical Approach Using CVIPtools*, 1st ed. Prentice Hall PTR, 1997.

[307] Unler, A., and Murat, A. A discrete particle swarm optimization method for feature selection in binary classification problems. *European Journal of Operational Research 206*, 3 (2010), 528–539.

[308] Vanneschi, L., and Poli, R. *Genetic Programming — Introduction, Applications, Theory and Open Issues*. Springer, 2012, pp. 709–739.

[309] Vera-Díaz, F., and Doble, N. The human eye and adaptive optics. In *Topics in Adaptive Optics*, B. Tyson, Ed. InTech, 2012, pp. 119–150.

[310] Viola, P., and Jones, M. J. Rapid object detection using a boosted cascade of simple features. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2001), IEEE Computer Society, pp. 511–518.

[311] Vojodi, H., Fakhari, A., and Moghadam, A. M. E. A new evaluation measure for color image segmentation based on genetic programming approach. *Image and Vision Computing 31*, 11 (2013), 877–886.

[312] Wang, X., Wang, Y., Yang, X., and Zuo, H. Texture classification based on SIFT features and bag-of-words in compressed domain. In *Proceedings of the 5th International Congress on Image and Signal Processing* (2012), IEEE, pp. 941–945.

[313] Weber, M., Welling, M., and Perona, P. Unsupervised learning of models for recognition. In *Proceedings of the European Conference on Computer Vision* (2000), vol. 2, Springer, pp. 101–108.

[314] Weise, T. *Global Optimization Algorithms-Theory and Application*, 2nd ed. Self-Published, 2009. Available online: `http://www.it-weise.de/research/publications/W2009GOATAA/index.html`.

[315] Weiss, G. M., and Provost, F. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research 19*, 1 (2003), 315–354.

[316] Weiss, K., and Khoshgoftaar, T. A survey of transfer learning. *Journal of Big Data 3*, 1 (2016), 1–40.

[317] Whigham, P. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (1995), pp. 33–41.

[318] Whigham, P., and Dick, G. Implicitly controlling bloat in genetic programming. *IEEE Transactions on Evolutionary Computation 14*, 2 (2010), 173–190.

[319] Wikipedia. Eye—Wikipedia, the free encyclopedia, 2003. [Online]. Available: `http://en.wikipedia.org/wiki/Eye` (visited on Nov. 12, 2013).

[320] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin 1*, 6 (1945), 80–83.

[321] Willis, A., and Sui, Y. An algebraic model for fast corner detection. In *Proceedings of the 12th IEEE International Conference on Computer Vision* (2009), IEEE, pp. 2296–2302.

[322] Wilson, S. W. Get real! XCS with continuous-valued inputs. In *Learning Classifier Systems: From Foundations to Applications*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Springer, 2000, pp. 209–219.

[323] Witten, I. H., and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques*, second ed. Morgan Kaufmann, 2005.

[324] Wu, T.-F., Lin, C.-J., and Weng, R. C. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research 5* (2004), 975–1005.

[325] Xiao, Q. Biometrics: Technology, application, challenge, and computational intelligence solutions. *IEEE Computational Intelligence Magazine 2*, 2 (2007), 5–25.

[326] Xie, H., and Zhang, M. Depth-control strategies for crossover in tree-based genetic programming. *Soft Computing 15*, 9 (2011), 1865–1878.

[327] Xue, B., Zhang, M., Browne, W., and Yao, X. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation 20*, 4 (2016), 606–626.

[328] Yang, B., and Chen, S. A comparative study on local binary pattern (LBP) based face recognition: LBP histogram versus LBP image. *Neurocomputing 120* (2013), 365–379.

[329] Yang, L., Hanneke, S., and Carbonell, J. A theory of transfer learning with applications to active learning. *Machine Learning 90*, 2 (2013), 161–189.

[330] Yang, X.-S. Ant algorithm and cuckoo search: A tutorial. In *Artificial Intelligence, Evolutionary Computing and Metaheuristics: In the Footsteps of Alan Turing* (2013), Springer, pp. 421–434.

[331] Yu, J., Yu, J., Almal, A., Dhanasekaran, S., Ghosh, D., Worzel, W., and Chinnaiyan, A. Feature selection and molecular classification of cancer using genetic programming. *Neoplasia 9*, 4 (2007), 292–303.

[332] Zadrozny, B. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the 21st International Conference on Machine Learning* (2004), ACM, pp. 114–121.

[333] Zaharie, D., Perian, L., Negru, V., and Zamfirache, F. Evolutionary pruning of non-nested generalized exemplars. In *Proceedings of the 6th IEEE International Symposium on Applied Computational Intelligence and Informatics* (2011), IEEE, pp. 57–62.

[334] Zhang, M., Andreae, P., and Bhowan, U. A two phase genetic programming approach to object detection. In *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 3215 of *Lecture Notes in Computer Science*. Springer, 2004, pp. 224–231.

[335] Zhang, M., Andreae, P., and Chow, R. Pixel statistics based neural networks for domain independent multiclass object detection. In *Proceedings of Image and Vision Computing New Zealand* (2002), University of Auckland, pp. 279–284.

[336] Zhang, M., and Ciesielski, V. Genetic programming for multiple class object detection. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence* (1999), vol. 1747 of *Lecture Notes in Computer Science*, Springer, pp. 180–192.

[337] Zhang, M., Ciesielski, V., and Andreae, P. A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Advances in Signal Processing 2003*, 8 (2003), 841–859.

[338] Zhang, M., and Johnston, M. A variant program structure in tree-based genetic programming for multiclass object classification. In *Evolutionary*

Image Analysis and Signal Processing, vol. 213 of *Studies in Computational Intelligence*. Springer, 2009, pp. 55–72.

[339] Zhao, Y., Huang, D.-S., and Jia, W. Completed local binary count for rotation invariant texture classification. *IEEE Transactions on Image Processing 21*, 10 (2012), 4492–4497.

[340] Zheng, S., Yuille, A., and Tu, Z. Detecting object boundaries using low-, mid-, and high-level information. *Computer Vision and Image Understanding 114*, 10 (2010), 1055–1067.

[341] Zhu, X., Vondrick, C., Fowlkes, C. C., and Ramanan, D. Do we need more training data? *International Journal of Computer Vision 119*, 1 (2016), 76–92.

[342] Zhuang, L., Gao, H., Luo, J., and Lin, Z. Regularized semi-supervised latent dirichlet allocation for visual concept learning. *Neurocomputing 119* (2013), 26–32.