Evolutionary Neural Architecture Search for Image Classification

by

Gonglin Yuan

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
2024

Abstract

Image classification serves as a cornerstone in the realm of computer vision, closely connected to various other vision-related tasks. Deep convolutional neural networks have surpassed traditional image processing techniques in the domain of image classification, earning them the status of the primary choice for many researchers. Although numerous effective convolutional neural networks have been proposed, their architectures tend to be highly task-specific. This implies that a change in data distribution for a specific task often necessitates an architecture design, a process that is not only labor-intensive but also prone to errors. Furthermore, the architectures need to be designed by experts who know deep learning and the specific task or domain information.

To address the shortcomings mentioned above, neural architecture search has made great progress and attracted great attention, which refers to searching for promising network architectures automatically for a given task. Evolutionary computation, capable of addressing non-differentiable problems, coping with discontinuous search spaces, and with promising global search ability, is suitable for neural architecture search, and a number of evolutionary neural architecture search (ENAS) methods have been proposed. However, most existing ENAS methods suffer from inaccurate evaluations (i.e., the evaluated fitness cannot represent the candidate's true performance) and inefficiencies, limiting their usage. Furthermore, the low explainability of deep neural networks remains a pressing concern, inhibiting their broader applications. Enhancing the evaluation reliability and efficiency of ENAS, coupled with improving network explainability, emerges as a crucial research direction.

The overall goal of this thesis is to advance effective and efficient ENAS methodologies by incorporating new search spaces, proposing novel architecture representations and new fitness evaluation methods, and developing innovative evolutionary operations. Furthermore, a method is proposed to explain network decisions in image classification tasks.

Firstly, this thesis proposes a novel particle swarm optimization-based ENAS method for image classification. Utilizing an autoencoder, the variable-length architecture representation is transformed into the fixed-length latent form, making it suitable for particle swarm optimization. Additionally, this thesis develops a novel hierarchical fitness evaluation method to assess the candidate's performance efficiently. The experimental results on three benchmark datasets confirm that the new method achieves promising results, improving both the classification accuracy and search efficiency.

Secondly, this thesis proposes a new ENAS method that incorporates a performance predictor to facilitate the convergence of the search, and a new weight inheritance mechanism to accelerate fitness evaluations. Specifically, the proposed method designs a new performance predictor to aid in the generation of high-performing offspring instead of directly predicting the candidate's fitness, and even the incorrect prediction will not harm the search. Additionally, an innovative weight-inheritance method is introduced to boost the efficiency of fitness evaluations. The empirical outcomes show that the proposed method achieves commendable classification outcomes and reduces the computational demands.

Thirdly, this thesis proposes an improved one-shot ENAS methodology characterized by its reliable fitness evaluations and high efficiency. In particular, an innovative supernet fine-tuning strategy is proposed to improve the fitness evaluation reliability, a new training strategy is adopted to improve the efficiency of supernet training, and a population initialization strategy is presented to enhance the evolutionary progress. Extensive experimental results confirm the method's superior performance and val-

idate each introduced strategy's efficacy.

Fourthly, this thesis proposes a novel evolutionary post-hoc, model-agnostic explanatory method to explain classifiers' decisions for image classification tasks. The proposed method leverages a stable diffusion model to help generate counterfactual images, aiding in explaining the decisions. Additionally, a multi-objective evolutionary method is proposed to identify minimal but pivotal regions, and the associated features are explainable by humans. The experimental results show the method's efficacy across diverse similar classes and classification architectures.

Acknowledgments

I wish to extend my profound and sincere gratitude to all individuals who have contributed to my Ph.D. journey.

First and foremost, I would like to express my deepest gratitude to my dear supervisors, Prof. Mengjie Zhang and Prof. Bing Xue. Their unwavering support, warm encouragement, and expert guidance have been so important to support my Ph.D. study. They have spent dedicated time providing detailed academic training and meticulous feedback, ensuring the enhancement of my research skills. Their constructive critiques and enlightening discussions have been instrumental in shaping my thoughts and refining my understanding of intricate concepts, fostering my growth as a researcher. I am incredibly fortunate to have had the opportunity to learn and evolve under the supervision of such distinguished and benevolent mentors. Their impact on my academic journey and overall personal development has been profound, and for this, I extend my sincerest appreciation.

I extend my cordial thanks to my peers and colleagues in the Evolutionary Computation Research Group (ECRG) for creating an active research environment and providing valuable encouragement to me. I would also like to thank all members of the Feature Analysis, Selection, and Learning (FASLIP) group for their generous sharing of knowledge, which has expanded my intellectual horizons and motivated me to complete this thesis. My appreciation also goes to the faculty and staff of the Victoria University of Wellington (VUW) for their support.

I wish to thank my family for their consistent understanding, support, and encouragement throughout my Ph.D. study. Their sacrifices and enduring encouragement have been my pillars of strength and a constant source of inspiration.

I would like to express my gratitude to the Wellington Doctoral Scholarship program for its financial support of my Ph.D. studies. Thanks to the School of Engineering and Computer Science (ECS) and VUW for providing me with the resources necessary to complete my thesis, especially the computing resources to support the expensive experiments.

Last but not least, I thank all reviewers and examiners who have taken the time to read my thesis and provided valuable comments and suggestions.

List of Publications

- **Gonglin Yuan**, Bing Xue and Mengjie Zhang. A Graph-Based Approach to Automatic Convolutional Neural Network Construction for Image Classification. In *Proceedings of the 35th International Conference on Image and Vision Computing New Zealand (IVCNZ 2020).* Wellington, New Zealand. 2020, pp. 1-6.
  DOI: 10.1109/IVCNZ51579.2020.9290492.

- **Gonglin Yuan**, Bin Wang, Bing Xue and Mengjie Zhang. Particle Swarm Optimization for Efficiently Evolving Deep Convolutional Neural Networks Using an Autoencoder-based Encoding Strategy. *IEEE Transactions on Evolutionary Computation.* 15pp, 2023. DOI: 10.1109/TEVC.2023.3245322.

- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. A Two-Stage Efficient Evolutionary Neural Architecture Search Method for Image Classification. In *Proceedings of the 18th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2021),* Lecture Notes in Computer Science. vol. 13031. Springer. Hanoi, Vietnam. 8-12 November 2021. pp. 469-484. DOI: 10.1007/978-3-030-89188-6_35.

- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. An Effective and Efficient Neural Architecture Search Method based on Performance Prediction and Weight Inheritance. Submitted to *IEEE Computational Intelligence Magazine (2023).*

- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. 2023. An Effective One-Shot Neural Architecture Search Method with Supernet Fine-Tuning for Image Classification. In *Proceedings of 2023 Genetic and Evolutionary Computation Conference (GECCO 2023).* ACM Press. Lisbon, Portugal. 15-19 July 2023. pp. 615-623.
  DOI: https://doi.org/10.1145/3583131.3590438.

- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. Efficient Evolutionary Neural Architecture Search using Reliable Fitness Evaluations. Conditionally accepted by *Information Sciences (2024).*

- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. Explaining Fine-Grained Image Classifications Using Evolutionary Search and the Stable Diffusion Model. To be submitted to *IEEE Transactions on Evolutionary Computation (2023).*

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The recent development of mobile internet technology has led to a substantial increase in the number of images on the internet. How to process, analyze, and understand the enormous amount of image data has become a challenging task in the field of computer vision. Image classification [82], which is the task of classifying the images into predefined categories based on the content, is the cornerstone of image analysis. Furthermore, other computer vision tasks, such as semantic segmentation and object recognition, are closely related to the image classification task.

However, image classification remains a challenging task for several reasons. Firstly, variations in an object's viewpoint, scale, deformation, and occlusion can make one object appear markedly different. Secondly, environmental factors, especially illumination conditions and background clutter, can complicate object identification. Thirdly, intra-class variation is challenging, as different objects of the same class may exhibit considerable variations in appearance. Therefore, an effective image classification model should be invariant to all aforementioned variations while being sensitive to intra-class differences.

Existing image classification techniques can be broadly categorized based

on whether the features are manually designed or automatically learned. The former often relies on human-designed feature extractors, which are labor-intensive and require domain knowledge. In contrast, techniques based on automatic feature learning can extract meaningful features directly from raw data, eliminating the need for explicit feature engineering, and they could generally outperform the former [82]. Notably, Convolutional Neural Networks (CNNs) stand out as a key image classification technique rooted in automatic feature learning. CNNs process raw images as input, learn features through various operations, and yield direct classification results. This end-to-end learning approach can effectively take advantages of the features of the image data to obtain accurate image classification results. There are some well-performed CNNs in recent years, such as VGG [126], GoogLeNet [142], ResNet [48], and DenseNet [56].

However, CNN architectures are usually task-specific, i.e., a new CNN architecture often needs to be designed when the distribution of data changes. As a result, when there is a new task, the CNN architecture needs to be designed by deep learning experts, taking into account the data's scale and distribution, which is labor-intensive. Moreover, the large computational power needed, predominantly available in large institutions or industrial giants, may not be accessible to all users. Consequently, the automatic design of CNN architectures, enabling users without CNN expertise to utilize them, has gained considerable attention.

Neural Architecture Search (NAS) is an approach that aims to automatically search for promising deep neural network architectures for a given problem [177]. Generally, NAS approaches fall into three categories based on the optimization techniques adopted [79], i.e., the Reinforcement Learning (RL)-based NAS algorithms, the gradient-based NAS algorithms, and the Evolutionary Computation (EC)-based NAS algorithms (ENAS). Given that RL-based algorithms typically require intensive computational resources [131], and gradient-based algorithms often produce ill-conditioned architectures [79], in this research, we focus on ENAS.

EC methods are population-based approaches inspired by Darwinian theory [12]. Many EC algorithms have been applied to ENAS, such as Genetic Algorithms (GAs) [92], Particle Swarm Optimization (PSO) [34], Differential Evolution (DE) [101], and Genetic Programming (GP) [64]. However, existing methods still suffer from complex architecture representations, time-consuming fitness evaluations, limited interpretability, and low search efficiency. Thus, this thesis focuses on addressing the above issues in the realm of ENAS for image classification.

Explainability [1] is also crucial for CNNs, enabling domain experts/users and developers to collect valuable insights and comprehend the underlying factors that influence the model's predictions. Such understanding can foster new discoveries and enhance decision-making processes. Moreover, understanding how CNNs arrive at their decisions can help build trust in the outputs, which is pivotal in some industries, such as finance and healthcare. Furthermore, explainable models allow developers to diagnose and address issues more effectively, as explainability can help reveal the underlying causes of incorrect decisions or predictions. As CNN models grow in size and complexity, explaining their decision-making processes becomes more challenging, and improving the explainability of deep CNNs is essential for both their development and real-world applications.

1.2 Motivations

The motivations of this research proposal will be discussed from three aspects. Firstly, the challenges of the neural architecture design and search will be presented. Next, the reasons for using EC methods to solve the problems will be briefly introduced. Finally, the limitations of the existing works will be itemized.

[1] For the purposes of this thesis, terms such as "explanability", "interpretability", and "understandability" are used interchangeably and no distinction is made between them.

1.2.1 Challenges of Neural Architecture Design and Neural Architecture Search

In order to employ CNNs to solve problems, people need to manually design the architecture of the network or use a NAS approach to search for appropriate architectures. However, designing architectures is a challenging task for the following reasons:

- CNN architectures often need to be tailored for specific tasks. Consequently, a network optimized for one particular task usually does not perform well on a different task. This means that a new problem necessitates the design of an entirely new CNN architecture. Factors such as the dataset's scale, number of categories, classification complexity, and image size typically influence the architecture of the CNNs.

- Designing a CNN architecture requires deep learning expertise. However, in practice, the experts with deep learning expertise are not always accessible. A significant number of these professionals work for major industrial firms and may not be available for the users, limiting many potential users from harnessing CNNs for their requirements.

- Manually designing CNN architectures is labor-intensive due to the trial-and-error process. Deep learning experts may need to invest substantial time on a single task, familiarizing themselves with the data distribution, training network weights, and adjusting hyperparameters over numerous iterations.

- The automatic NAS approach, while alleviating some manual labor, is resource-intensive. The vast architectural search space implies numerous candidate networks, and evaluating each network's performance can be very costly and time-consuming. Efforts to mitigate

the evaluation time may lead to another issue: less accurate assessments.

- Regardless of whether CNN architectures are designed manually or generated by NAS algorithms, their intricate designs pose interpretability challenges. While some methods offer insights into the important areas of the image, these methods cannot explain how these areas (and the corresponding features) affect the classifier's decisions, which limits the explainability of CNNs.

- Providing detailed explanations for CNN decisions on similar image classes is notably challenging. The classifications for similar image classes are difficult, and explaining these decisions is much more difficult. While attempts have been made to elucidate the logic behind different categories, research focusing on explanations for similar image categories remains rare.

1.2.2 Why EC

The process of NAS requires the optimization of several variables: the number of layers, each layer's type, and the corresponding parameters of each layer, such as the number of feature maps and the kernel size. Given these intricate considerations, NAS is a complex optimization problem, which is often nonconvex and non-differentiable since the candidate parameters are discrete [36]. This renders traditional mathematical optimization techniques less effective.

EC methods are a class of nature-inspired stochastic optimization paradigms, which are population-based and mimic the principle of Darwinian natural selection. As EC methods are insensitive to the local optima, able to address non-differentiable problems, and can cope with discrete search spaces [12], they are quite suitable for the NAS optimization problem. Specifically, GA is capable of producing high-quality solutions

by utilizing bio-inspired operators, such as crossover, mutation, and selection. We can design new operators according to the specific ENAS task, which is flexible. Besides, GA can cope with discrete representations, which are suitable for network architecture representations. PSO is of simple implementation, high efficiency, and robustness [120], and fewer parameters need to be tuned in PSO. Typically, PSO addresses continuous search spaces, and each PSO particle signifies a potential solution or network architecture. When architectural representation parameters are continuous, PSO proves effective for ENAS challenges.

In Explainable Artificial Intelligence (XAI), EC also presents promising prospects, with Multi-Objective Evolutionary Algorithms (MOEAs) standing out notably. The intrinsic capability of MOEAs to concurrently optimize multiple conflicting objectives renders them particularly suitable for counterfactual explanations. Within image classification, the counterfactual image is anticipated to significantly alter the classifier's prediction probability, thereby highlighting the importance of the modified regions and providing explanations for the cause reasoning. Besides, modifications in the counterfactual should be restricted to the smallest viable region because a limited area usually corresponds to fewer attributes or features. Altering every pixel in an image obfuscates the specific attributes responsible for any prediction alteration. As such, pinpointing pivotal regions evolves into a bi-objective optimization challenge, i.e., minimizing the area and maximizing the significance. In this context, the Nondominated Sorting Genetic Algorithm II (NSGA-II) [29], a robust MOEA, is adeptly suitable to address this challenge.

1.2.3 Limitations of Current ENAS Methods

ENAS methods offer an approach to automatically constructing CNN architectures using the EC algorithms. However, the improvement of ENAS algorithms still needs to be further explored due to the following limita-

tions.

1.2.3.1 CNN Architecture Search Space

The search space serves as a foundational component in determining the outcomes of ENAS algorithms. Typically, the design of the search space is based on existing high-performing CNN architectures. Despite the pivotal role of the search space, many ENAS methodologies need to pay more attention to the portability of the candidate networks constituting this space.

Recently, there has been a surge of interest in innovative, efficient, and portable network architectures within the computer vision domain [52, 53, 87, 115, 172]. These architectures are particularly noteworthy as they reduce the model size, optimize computational efficiency without compromising, and sometimes even enhance the overall performance. In this case, there is a need for the construction of superior search spaces. These enhanced spaces would incorporate effective and efficient network candidates, improving the searched network architecture.

1.2.3.2 CNN Architecture Representation

Before commencing the search process of an NAS algorithm, the encoding strategy must be established. Encoding strategies primarily differ based on whether the representation length is variable or fixed. For fixed-length encoding strategies [170], all individuals possess an equal length, inherently restricting the architecture's flexibility. Such strategies offer the advantage of reduced search space and compatibility with most standard evolutionary operators designed for fixed-length representations. However, these strategies present notable limitations. One significant constraint is the requirement to predetermine the architecture's length/depth, which demands human intervention. Additionally, the optimal architecture length varies across tasks, posing a challenge even for experts to anticipate. Consequently, achieving optimal performance becomes challenging when ad-

hering to a predetermined length.

Conversely, variable-length encoding strategies can represent networks of diverse depths, granting greater flexibility and expanding the search space [147]. The primary limitation is that these representations are more challenging to process using standard EC operators. Researchers then propose novel operators tailored for variable-length individuals; however, balancing exploration and exploitation capabilities is difficult. Therefore, a representation that utilizes fixed-length to depict variable-length architectures is worth exploring, especially given its compatibility with most EC algorithms.

Furthermore, exploring autoencoder-based architecture representations holds potential. Autoencoders, utilized for unsupervised representation learning [50], comprise an encoder that extracts intrinsic features of input data and produces a corresponding encoded latent representation, and a decoder that reconstructs the input from this latent representation. These encoded latent representations are continuous and typically possess fewer dimensions than the original input. Hence, the encoder can map discrete architecture representations to a latent space, yielding lower-dimensional, fixed-length, and continuous representations suitable for the PSO algorithm. Given that similar architectures often exhibit comparable performance [153,161], ensuring that similar architectures map to neighborhood regions in the latent space can further enhance the search process.

1.2.3.3 Fitness Evaluation

In NAS strategies, fitness evaluation of architectures stands as the most time-intensive procedure [78]. Typically, classification accuracy emerges as a pivotal metric. To evaluate the accuracy, an individual needs to be transferred into the corresponding CNN architecture according to the specific decoding strategy. Following this, it undergoes gradient training to get the weights, and subsequently, its performance is assessed on the evolutionary training dataset. This process usually consumes large amounts

of computational resources and costs a lot of time.

Researchers have been trying to reduce the time of the fitness evaluations. The methodologies adopted can be generally divided into three categories: (1) shallow-training methods [124], (2) surrogate-assisted methods [68], and (3) parameter-sharing/replication techniques [154].

Shallow-Training Methods:

The shallow-training fitness evaluation methods are favored by numerous researchers and adopt various strategies. Some researchers adopt an early stopping policy, i.e., only training the networks for a predefined small number of epochs [6]; some leverage a subset of the whole training data to train the candidates, with the assumption that the subset maintains a similar data distribution as the original dataset [116, 123]. A few works use a partial network, i.e., a network with lower depth and/or lower width, as the proxy to estimate the performance [76, 149]. While these methods can reduce the computational cost, they might also lead to an inaccurate performance estimation. Zhou *et al*. [176] analyzed the inconsistency of different reduction factors systematically using a model pool. However, the candidate architectures may become more similar and well-performing along the search process. In this regard, assessing the candidates under proxies may not precisely identify the actual promising candidates. Thus, investigating the effect of reducing training data used during evolution and proposing an effective fitness evaluation method are crucial.

Surrogate-Assisted Methods:

Some scholars have designed performance predictors to utilize surrogate-assisted models to directly predict network performance based on architectural features [136, 150]. However, two primary challenges persist: (1) preparing data to train these performance predictors may involve a large computational budget [135], and (2) inaccuracies in predictions may misguide the search process. For instance, NASNet [183] executed fully-training and evaluation of candidate networks and discovered that the optimal net-

work was positioned at mere 70th out of 250 contenders according to the performance predictor's prediction. Hence, designing efficient and accurate surrogate-assisted fitness evaluation methods is still a crucial research direction.

Weight Inheritance/Replication Methods:

Weight inheritance/replication is a strategy employed to expedite the fitness evaluation process. Typically, two contexts are prevalent: inheritance from parent network(s) and inheritance from a supernet comprising all candidate networks.

In the former scenario, offspring often share certain common components with their parent(s). One circumvents the necessity of training from scratch by inheriting weights of these common components as initialization. Evolutionary operators can be meticulously designed to optimize this inheritance to maximize the shared components between offspring and their parent(s).

Conversely, methods belonging to the latter context are termed *one-shot NAS* [46]. Here, only a single supernet needs to be trained initially, and all the candidate networks directly inherit weights from the supernet without further training. One-shot NAS contains two stages: the supernet training and the subsequent architecture optimization [46]. In the first stage, a large supernet that encompasses the entire search space is trained. In the second stage, the optimized architecture is searched by a search algorithm, where all candidate networks (a.k.a. subnets) inherit weights from the supernet without further training, resulting in significant time savings. However, the second stage of one-shot NAS algorithms still faces a primary challenge, i.e., unreliable evaluations, which may negatively affect the search. Specifically, it is observed that the performance ranking using inherited weights is different from that getting from stand-alone training [162], which leads to unfair comparisons among subnets. This issue has been attributed to the extensive sharing extent of weights in the supernet [96], where all subnets that contain the same operation in a layer

use the same weights. These replicated weights may not be optimal for individual subnets, leading to potential degradation in performance. To mitigate this problem, previous approaches have proposed constructing multiple sub-supernets that cover different regions of the search space to mitigate the co-adaptation problem [173], or using a dictionary of weights based on a group of supernets to combine weights for subnets [130]. However, these methods often require significant computational resources for training multiple supernets and may result in large memory consumption for saving weights. Another methodology, proposed by Zhou *et al.* [178], adjusts the sharing extent during supernet training using a curriculum learning strategy. However, the learning strategy is complicated and introduces additional computations. Therefore, improving the evaluation reliability of one-shot NAS remains a pivotal research direction.

1.2.3.4 Supernet Training

Although one-shot NAS methods have reduced computational costs compared with most NAS methods, the training of the supernet still requires significant computing resources. To address this challenge, Cai *et al.* [19] proposed a progressive shrinking scheme to train the large supernet efficiently, but the designed supernet is nested, with small subnets embedded in large subnets, resulting in deeply coupled weights that may impair fitness evaluations. Consequently, many supernets adopt a single path mechanism [46], where each subnet is a single path inside the supernet. However, research to enhance supernet training efficiency remains sparse. Given the large computational requirements of supernet training, breakthroughs in this domain would be important.

1.2.3.5 Search Efficiency

The efficiency of traditional EC methods is currently not satisfactory [114]. In the evolutionary process of ENAS, the parent candidates are usually se-

lected based on their fitness values, i.e., the ones with good performance are more likely to be chosen. Then, new offspring solutions are generated by performing genetic operations on the selected parent candidates. In this way, the offspring are expected to achieve better performance than the parents [158], promoting the evolutionary process. If the ratio of better offspring could be enhanced, the process of evolution would be accelerated, reducing the total number of generations and then reducing the computational cost. Thus, strategies to improve offspring quality and further improve EC method efficiency are needed.

Population initialization determines the initial solutions (candidate architectures in NAS), and it has been demonstrated that good initial solutions can facilitate the evolutionary process [102], reduce computational costs [62], increase the probability of finding the optimal solutions [88], and improve the stability of search results [94]. Because the population size is usually limited due to the restricted computational budget, the opportunity to cover promising areas through random initialization is low, and the effectiveness could further decrease with increasing dimensions of the search space [49]. For one-shot NAS, designing methods to leverage information obtained from supernet training to pinpoint high-potential individuals during initialization is worth exploring.

1.3 Research Goals

The overall goal of this thesis is to advance effective and efficient ENAS methodologies by developing new search spaces, novel architecture representations, pioneering fitness evaluation techniques, and innovative evolutionary operations. Furthermore, a new method will be introduced to explain the decisions of the networks for image classification tasks by generating counterfactuals. With these designs, this thesis aims at increasing the classification accuracy, diminishing the computational overhead, and improving the explainability of prevalent image classifiers. Specifically,

four specific objectives are as follows.

1. The objective is to design an effective PSO-based ENAS algorithm, and there are three sub-objectives:

    - Design an encoding scheme to represent the candidate block architectures to suit the search algorithm. An autoencoder is expected to compress the variable-length discrete integer block vectors to fixed-length continuous decimal latent vectors. The latent space is expected to be smooth and continuous, facilitating the downstream PSO search process.
    - Propose a new loss function for the autoencoder that not only considers the reconstruction loss but also takes the architecture similarity and the model scale similarity into account. In this way, the networks with a similar architecture or a similar model scale are expected to be embedded in neighborhood regions in the latent space.
    - Develop a dynamic hierarchical fitness evaluation method to efficiently and effectively estimate the performance of individuals during different stages of the search process.

2. The objective is to propose an efficient ENAS algorithm by incorporating a performance predictor to help generate well-performed offspring and employing a weight-inheritance method to accelerate fitness evaluations. Four specific sub-objectives are detailed below:

    - Propose a new lightweight backbone block architecture that can effectively construct discriminative features from the input feature maps. It is expected to have a small model scale and high computational efficiency.
    - Invent a random forest-based performance predictor. Instead of directly predicting the fitness, it is expected to predict whether

offspring outperform their parent(s) and the confidence level of the prediction. In this way, only the offspring that are most likely to perform well will be preserved, which can improve the overall performance of the offspring population, thereby accelerating the convergence of the evolutionary process. Moreover, even inaccurate predicted results will not harm the evolution.

- Propose new crossover and mutation operators to cope with the architecture representations. The newly generated offspring are expected to remain the same in most layers as one of its parents, making it easier for the performance predictor to identify whether the offspring can outperform their parent.
- Propose a new weight inheritance method to avoid random initialization of the newly generated layers of the offspring, which is expected to further improve the training efficiency.

3. The objective is to propose a powerful one-shot NAS algorithm for image classification, with accurate fitness evaluations, efficient supernet training, and a powerful population initialization to facilitate the search process. Three sub-objectives are as follows:

   - Propose a new efficient supernet training strategy to reduce the computational cost of training the weights of the supernet.
   - Propose an effective supernet fine-tuning strategy to offer accurate fitness evaluations for all candidate network architectures. The supernet weights are expected to be fine-tuned based on the distribution of the candidate network architectures, which is expected to reduce the sharing extent and provide more accurate evaluations.
   - Propose a new population initialization method to select potentially well-performing candidate network architectures as initial solutions, which is expected to facilitate the evolutionary search.

4. The objective is to propose a novel post-hoc, model-agnostic method for explaining classifiers in image classifications, and three detailed objectives are described as follows:

    - To develop a method that employs a stable diffusion model for generating counterfactual images for explanation purposes. The generated images are expected to be natural and circumvent the sharp edges that often result from directly masking specific regions.
    - To utilize an MOEA to automatically identify an ensemble of superpixels of salient impact and a minimal number of superpixels, which are essential for explaining the decisions of classifiers.
    - To propose a new objective function that assesses the impact of the selected superpixels for similar classifications.

1.4 Major Contributions

This thesis makes the following major contributions.

1. This thesis shows how to develop an efficient PSO-based NAS algorithm to search for appropriate network architectures on image classification tasks. Unlike most methods that use variable-length representations for network architectures, this method designs an autoencoder to represent variable-length network architectures as fixed-length latent vectors, which converts the original search space to a latent space that can facilitate the downstream search. Besides, an efficient and effective hierarchical fitness evaluation method guides the search process and improves efficiency. The experimental results show the proposed method is a very competitive ENAS algorithm that achieves an error rate of 2.74% on CIFAR-10 and 16.17% on

CIFAR-100, and reduces the computational cost from hundreds or thousands of GPU-days to only 2.2 and 4 GPU-days, respectively. Further analyses also confirm the effectiveness of both the proposed autoencoder and the proposed hierarchical fitness evaluation method.

Part of this contribution has been published in:

- **Gonglin Yuan**, Bin Wang, Bing Xue and Mengjie Zhang. Particle Swarm Optimization for Efficiently Evolving Deep Convolutional Neural Networks Using an Autoencoder-based Encoding Strategy. *IEEE Transactions on Evolutionary Computation.* 15pp, 2023. DOI: 10.1109/TEVC.2023.3245322.

2. This thesis shows how to develop a random forest-based performance predictor to help generate superior offspring candidates to efficiently construct good CNNs for image classification. Unlike existing NAS methods that directly use performance predictors for evaluations, which may introduce inaccurate assessments and harm the evolution, this thesis develops a new random forest-based performance preditor to help produce potentially good offspring, improving the population's performance and accelerating the evolution. In this way, even sometimes the estimated predictions being inaccurate will not harm the evolution. Furthermore, a new weight inheritance method is proposed to accelerate the evaluation processes, making the offspring inherit weights from their parents and other existing trained networks. Besides, new genetic operations assist in producing offspring that share a large percentage of the good genetic materials with one of their parents, improving the performance predictor's effectiveness and promoting weight inheritance. Finally, a new efficient backbone block structure further reduces the computational cost and helps search for lightweight networks. The experimental results demonstrate that the proposed method is a very competitive ENAS algorithm, which costs only 1.6, 2.4, and 9.4 GPU-days

on CIFAR-10, CIFAR-100, and ImageNet, and achieves error rates of 2.50%, 16.75%, 24.1%, respectively. The searched networks are also substantially smaller than those learned by most of the current ENAS methods. Further analyses reveal the superiority of the proposed block structure and confirm the effectiveness of the proposed performance predictor and the weight inheritance method.

Part of this contribution has been submitted/published in:

- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. An Effective and Efficient Neural Architecture Search Method based on Performance Prediction and Weight Inheritance. Submitted to *IEEE Computational Intelligence Magazine (2023).*

3. This thesis shows how to develop an efficient supernet training strategy and a reliable fitness evaluation method for one-shot NAS. An efficient supernet training strategy reduces the training time by leveraging a novel weight initialization method that considers the particular architecture of the supernet. Additionally, the new method fine-tunes the supernet during the evolutionary search process based on the architectures to be evaluated, providing more reliable performance estimations. Furthermore, a powerful population initialization method facilitates the search by utilizing information from the previous supernet training. Experimental results demonstrate that the proposed method achieves error rates of 2.59%, 16.67%, and 23.9% on CIFAR-10, CIFAR-100, and ImageNet, respectively, using only 0.31, 0.28, and 4.1 GPU-days. The classification results are promising, and the computational cost is lower than most competitors. Further experiments and analyses confirm the efficacy of the proposed supernet training, population initialization, and supernet fine-tuning methods.

   Part of this contribution has been submitted/published in:

- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. 2023. An Effective One-Shot Neural Architecture Search Method with Supernet Fine-Tuning for Image Classification. In *Proceedings of 2023 Genetic and Evolutionary Computation Conference (GECCO 2023).* ACM Press. Lisbon, Portugal. 15-19 July 2023. pp. 615-623. DOI: https://doi.org/10.1145/3583131.3590438.
- **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. Efficient Evolutionary Neural Architecture Search using Reliable Fitness Evaluations. Submitted to *IEEE Transactions on Evolutionary Computation (2023).*

4. This thesis shows how to develop a post-hoc, model-agnostic approach for explaining image classification decisions based on an evolutionary multi-objective method. The method employs a stable diffusion model to produce counterfactual images, thereby explaining the influence of different image regions on classifications. This algorithm optimizes both the impact and quantity of the superpixels, resulting in more straightforward explanations. Complementing this, a new fitness evaluation function measures the impact of these superpixels in classifications. Comprehensive experimental evaluations underscore the efficacy of this method across various classes and classification models. By bridging the gap between complex classifier decisions and human interpretability, this research paves the way for more transparent applications of deep learning.

   Part of this contribution has been submitted in:

   - **Gonglin Yuan**, Bing Xue, and Mengjie Zhang. Explaining Fine-Grained Image Classifications Using Evolutionary Search and the Stable Diffusion Model. To be submitted to *IEEE Transactions on Evolutionary Computation (2023).*

1.5 Organization of the Thesis

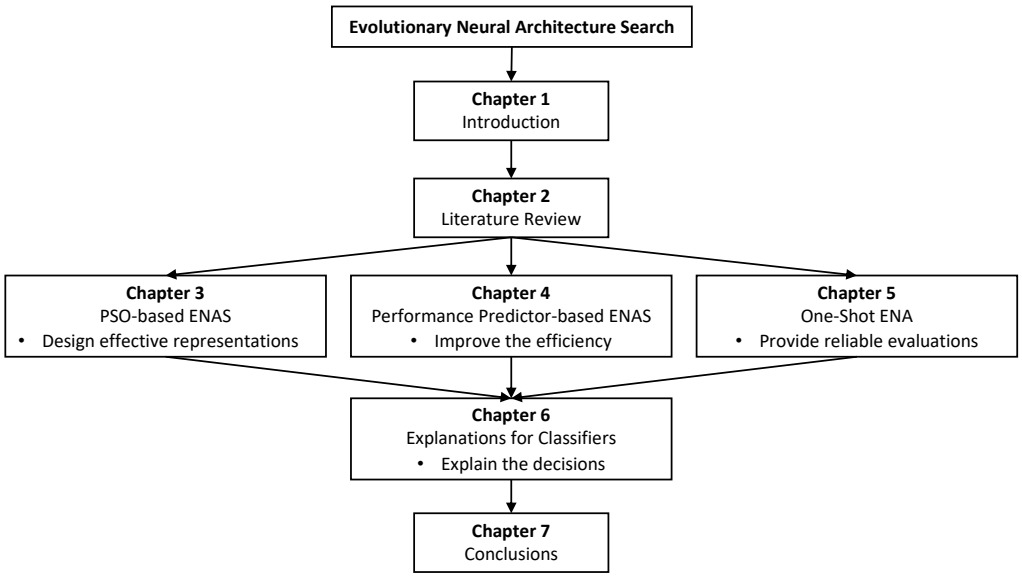


Figure 1.1: The outline of this thesis.

The following sections outline the structure and content of the remaining of this thesis, which is also presented in Figure 1.1. Chapter 2 provides the fundamental background and comprehensive literature review. Chapters 3 to 6 elucidate the primary contributions of this thesis, each spotlighting its unique objective and corresponding significant advancement. Chapter 7 concludes this thesis and indicates potential directions for subsequent research endeavors.

Chapter 2 presents a detailed background and a review of existing literature. It encompasses the concepts of computer vision, image classification, evolutionary computation, neural architecture search, and explainable deep learning. The chapter also discusses open questions and identifies the shortcomings of existing research.

Chapter 3 proposes a novel PSO-based NAS method, leveraging a newly designed autoencoder to represent the architectures of candidate networks

effectively. Additionally, it presents a dynamic hierarchical fitness evaluation method to evaluate the candidates' fitness to guide the search. This chapter subsequently describes the details of the architecture representation and the evaluation method, concluding with experimental results that validate the efficacy of the proposed method.

Chapter 4 presents an effective NAS method, incorporating a performance predictor and weight inheritance method to improve efficiency. This chapter introduces the proposed architecture search space and the corresponding encoding strategy. This is followed by an in-depth presentation of the proposed performance predictor and weight inheritance mechanism. Subsequent sections illustrate the experimental design, outcomes, and analyses, confirming the method's efficiency and efficacy.

Chapter 5 presents an enhanced one-shot ENAS method that improves the efficiency of the supernet training and the reliability of fitness evaluations. The chapter analyzes the limitations of current one-shot NAS methods. Subsequently, detailed information on the proposed supernet training, weight fine-tuning, and population initialization methods are presented. The settings of the experiments are then given, and the results are presented to confirm the effectiveness of this proposed method.

Chapter 6 introduces an innovative method to explain classifier decisions on image classifications. The chapter starts with a depiction of the overall algorithm framework, followed by the design of encoding and decoding strategies. The key point is the counterfactual image generation process, which utilizes the stable diffusion model's inpainting capability to emphasize specific features within target image regions. The subsequent section presents the novel fitness evaluation methods. A series of experiments, results, and analyses are presented, underscoring the capability of the proposed method.

Chapter 7 summarizes the research and provides the comprehensive conclusions of the thesis. It highlights the key points and the primary contributions of the thesis, and also discusses future research directions.

Chapter 2

Literature Review

This chapter presents the essential background and basic concepts of machine learning, image classification, Convolutional Neural Networks (CNNs), Neural Architecture Search (NAS), typical Evolutionary Computation (EC) algorithms, and image generation methods. Furthermore, this chapter reviews recent works on Evolutionary Neural Architecture Search (ENAS), fitness evaluations in NAS, one-shot NAS methods, and explanations for CNNs. In the end, it summarises the content and points out the limitations of existing works.

2.1 Machine Learning

Machine learning [59, 89], a subfield of Artificial Intelligence (AI), aims to enable computers to learn from data automatically, imitating the mechanism that humans learn. With the advent of the big data era, the demand for data analysis continues to increase. Consequently, developing machine learning approaches to extract knowledge from expansive datasets becomes an essential research area.

A typical machine learning model involves three main parts [2]: (1) a decision process, (2) an error function, and (3) an optimization process. Specifically, the machine learning method outcomes predictions based on

input data, and the error function evaluates the predictions by comparing them with labeled data, and the weights/parameters of the model are adjusted to achieve better prediction results through an optimization process.

Machine learning paradigms can be categorized based on the presence of labels or the types of feedback into four categories [3]:

1. **Supervised Learning**: The dataset has already been labeled, and the model can learn from the labeled data and assess how accurate the performance is. There are two major tasks for supervised learning, i.e., classification and regression.

2. **Unsupervised Learning**: The dataset is not labeled, and the models learn to find structures within the data by extracting meaningful features.

3. **Semi-Supervised Learning**: The dataset is a mix of both labeled and unlabeled data, and the models are trained to label the unlabeled data.

4. **Reinforcement Learning**: An agent takes actions within various states and receives feedback. It learns how to map states to actions to maximize the expected long-term reward.

Classification is one of the major tasks in supervised learning [63]. The training data comprises input features coupled with corresponding labels. The machine learning methods, such as decision trees [113], support vector machines [97], and neural networks [5], are fed with the training data and learn to map the features and the labels. Subsequently, these models can predict labels based on the input data. Classification can be binary or multi-class. For instance, determining if an email is spam is a binary classification task while identifying images of digitals from 0 to 9 falls into a multiclass classification task.

Typically, datasets are divided into three distinct subsets: a training set, a validation set, and a test set [9]. The training set is used to train the model to learn the features of the data and to make accurate predictions. The validation set is employed to provide an evaluation of the model's performance during the tuning of hyperparameters, assisting in avoiding overfitting and underfitting and aiding in model selection. The test set is used to assess the model's performance, testing its generalization ability and potential performance on unseen data.

2.2 Image Classification

Image classification [82] aims to classify images into specific categories according to their content information, and the main procedures are extracting the semantic information and training the classifier. Image classification is a fundamental research topic in computer vision and attracts great attention. In recent years, there have been a huge amount of images to classify with the development of big data. The traditional image classification approach extracts the hand-crafted image features and trains the classifiers accordingly. However, the human-designed features cannot effectively discriminate different classes when facing a huge amount of data. On the contrary, automatic feature learning approaches are widely employed to solve image classification problems.

There are some widely used datasets for image classification, such as MNIST [71], CIFAR-10 [65], CIFAR-100 [65], and ImageNet [112]. There are different numbers of images in each dataset, and the resolution of images is different. Figure 2.1 exhibits some examples from CIFAR-10. Specifically, the CIFAR-10 benchmark dataset is composed of $60\,000$ RGB images whose size is 32×32. These images belong to 10 categories, and each category has the same number of images.

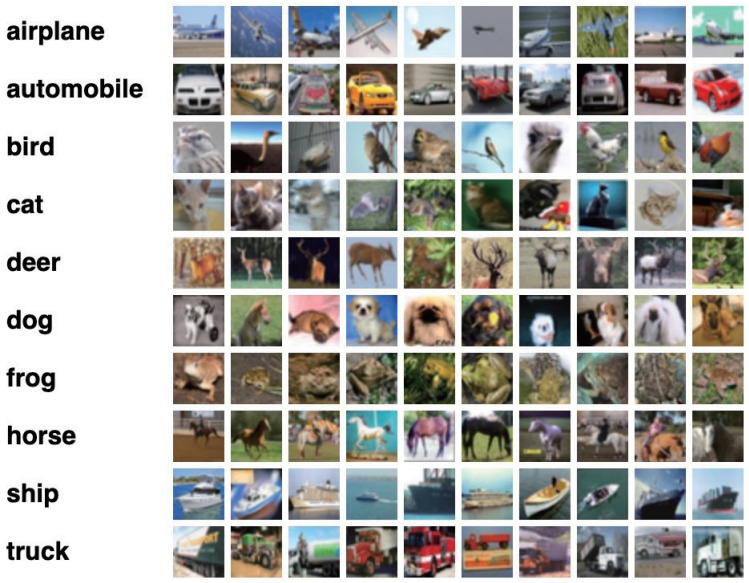


Figure 2.1: Some examples of the CIFAR-10 dataset.

2.3 CNNs and NAS

2.3.1 CNNs

Convolutional Neural Networks (CNNs) [7, 45, 72] are a kind of neural network widely used for image processing tasks, such as image classification [82], object recognition [134], and image segmentation [98]. Figure 2.2 shows a standard CNN architecture for image classification. Conventionally, CNNs mainly have three types of layers, i.e., convolutional layers, pooling layers, and fully-connected layers.

Convolutional layers connect to different local regions of the input feature maps and output the scalar product between their weights and the pixel values in the corresponding area. There are some parameters for convolutional layers, such as the number of filters, the size of kernels, and the stride size. Usually, the elementwise activation functions are employed to process the output of convolutional operations. Pooling layers are utilized to downsample the spatial dimensionality of the corresponding in-

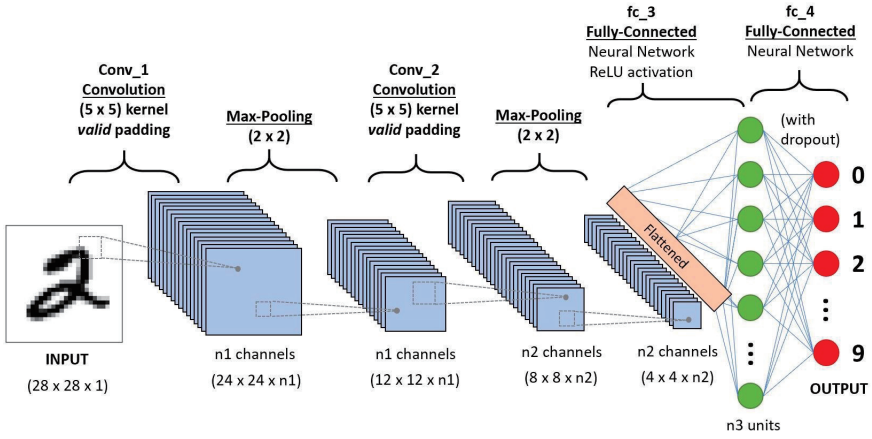


Figure 2.2: An example of a CNN architecture for image classification [1].

put. Average pooling and max-pooling are widely used in many promising CNNs. Fully-connected layers attempt to produce the class scores used for classification. Every node in a fully-connected layer is directly connected to every node in both the previous and the next layer. The number of layers, the number of neurons for each layer, and the corresponding activation function are several parameters for fully-connected layers.

Recently, some new CNN architectures have been designed to reduce the computational cost and maintain performance. One of the most fundamental technologies is the depth-wise separable convolution, which is illustrated in Figure 2.3. As we can see, it contains two separate steps — the depthwise convolution and the pointwise convolution. Another important technique for reducing the computational cost is the group convolution, which divides the input channels into several groups and applies convolution operations separately for each group. These new techniques have already been successfully used in some promising CNN architectures, such as MobileNet [53], ResNetXt [156], ShuffleNet [172], and CondenseNet [55]. Moreover, the new techniques offer more choices for the NAS tasks and expand the search space, making the design of the search strategy more challenging.

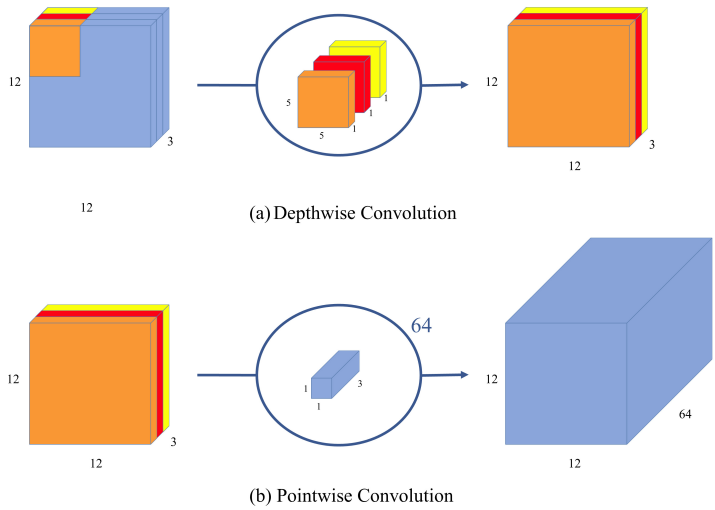


Figure 2.3: Two steps of the depth-wise separable convolution.

2.3.1.1 DenseNet

DenseNet [56] is mainly composed of dense blocks as the core units, and the blocks can extract meaningful features and alleviate the vanishing gradient problem during the training. Figure 2.4 shows the details of a standard dense block.

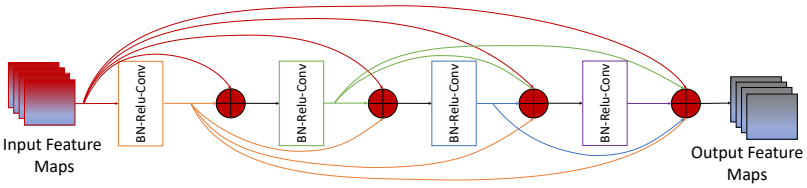


Figure 2.4: An example of a four-layer dense block (adapted from [56]).

In the example, the feature maps generated by the previous layers are input to the dense block, and the dense block could further extract powerful features from them. There are four composite layers inside the block in the example, and each layer is composed of three consecutive layers — a Batch Normalization layer, a rectified linear unit (ReLU) layer, and a

convolutional layer, which work together to extract features. The input of each composite layer is the output of the previous directly connected layer and the outputs from all the previous layers inside the same dense block, which is different from some other conventional convolutional neural networks.

Hyper-parameters of dense blocks are essential and significantly affect the performance. Firstly, the dense block's length (the number of the composite layers) affects the ability to extract features. Secondly, the growth rate of each composite layer is the number of feature maps generated by the corresponding convolutional layer and also affects the feature extraction.

2.3.1.2 MobileNetV3

MobileNetV3 [52] achieves outstanding success in image classification, object detection, and semantic segmentation tasks using relatively low computational resources, attributing to its promising structure, where the effective MobileNetV3 block is employed to construct informative features, shown in Figure 2.5.

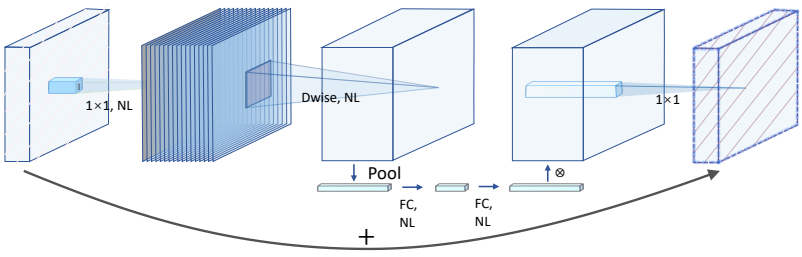


Figure 2.5: The structure of a MobileNetV3 block (adapted from [52]).

In a MobileNetV3 block, the input feature maps are processed by 1×1 convolutional layers first. The size of the output feature maps is consistent with the input, and the number of feature maps is expanded according to the predefined expansion rate, i.e., the width of the output feature

maps is the product of the number of input feature maps and the expansion rate. After that, depth-wise convolutions are constructed, followed by a Squeeze-and-Excitation (SE) module, providing different weights for different channels of the input feature maps. Specifically, the SE module contains a global average pooling layer to squeeze the spatial information, two fully-connected layers, and corresponding non-linear processors. The output feature maps are achieved by multiplying the learned weights for each channel and the original input feature maps. Essentially, the SE module is a lightweight channel-wise attention mechanism, focusing more on informative representations and overlooking the less powerful information channel-wisely. Finally, point-wise filters project the feature maps to the output with the same dimension as the block input, and a skip connection is constructed between the original block input and the output.

The expansion rate is an essential hyper-parameter to a MobileNetV3 block. It determines the number of feature maps to the depth-wise separable convolution, which significantly affects the representational power of the block. Another crucial hyper-parameter is the depth-wise convolutional kernel size, since different kernel sizes can capture different levels of information. Due to diverse MobileNetV3 blocks being in different positions inside a network, they may prefer different expansion rates and kernel sizes.

2.3.2 NAS

NAS [36] aims to automatically search for the optimal deep neural network architecture a^* within a vast search space $\mathcal{A}$ for a given dataset, defined as follows in image classification tasks:

$$a^* = \arg\min_{a \in \mathcal{A}} \mathcal{L}_{eva}(a, w_a^*) \ , \tag{2.1}$$

where $\mathcal{L}_{eva}$ denotes the classification loss on the evaluation data, which is used for measuring the performance of networks, and w_a^* represents the

learned weights for the candidate network architecture a. The weights are learned by minimizing the training loss, i.e.,

$$w_a^* = \arg\min_{w_a} \mathcal{L}_{tra}(a, w_a) \ , \tag{2.2}$$

where w_a as the weights of the network a and $\mathcal{L}_{tra}$ representing to the loss on the training data.

The computational cost of NAS is often prohibitive, as the performance of each network a must be evaluated, and learning the weights w_a^* is time-consuming. To mitigate this, some researchers employ shallow training methods to approximate w_a^* and reduce computational costs. Others develop proxy models to directly predict the performance without requiring optimized weights w_a^*. Additionally, some researchers design a large supernet $\mathcal{N}$ encompassing the entire search space and train the weights of the supernet $W_{\mathcal{A}}$. In this case, the candidate network a can directly inherit the corresponding weights from the supernet, bypassing the need for the weight optimization as Equation (2.2).

2.4 Evolutionary Computation

Evolutionary Computation (EC) algorithms, learning from the biological evolution process [12], are computational intelligence techniques for searching and optimizing [118]. As population-based algorithms, EC methods produce a population of individuals to search for promising solutions in the search domain, and each individual represents a potential solution for a specific problem. The optimization problem can be represented in flexible forms in EC algorithms, without the limitation of mathematically represented as continuous and differentiable functions. Besides, EC has good global search ability. Thus, EC is widely used in NAS problems. It is complicated to represent the network architectures by continuous and differentiable mathematic functions.

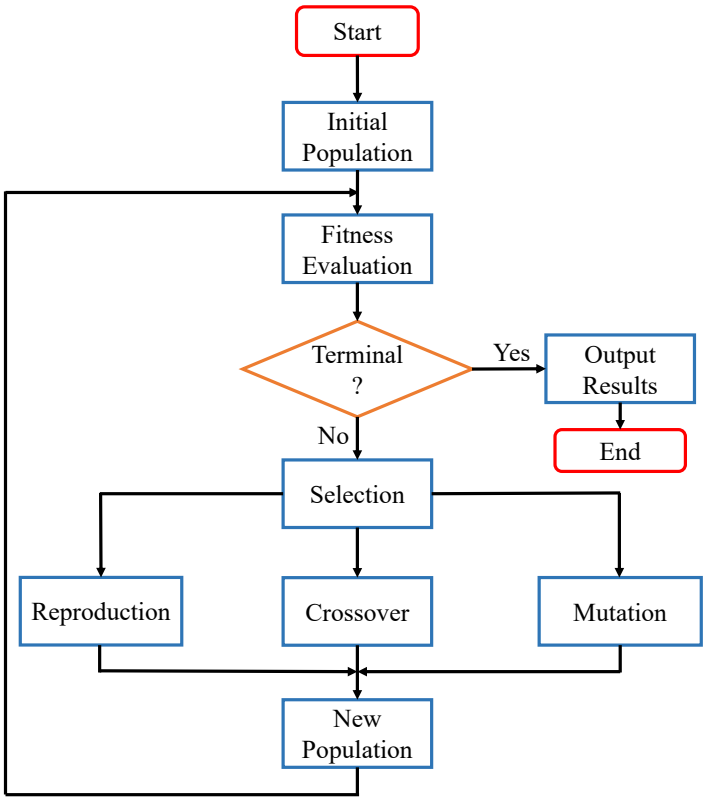


Figure 2.6: The general process of EAs.

At the beginning of EC approaches, a group of potential solutions is initialized in the search space. Then, they are evaluated and updated iteratively until the termination criterion is met. Usually, the individuals will be distributed into more promising search areas during the updating process with interaction, competition, and cooperation among them.

2.4.1 Evolutionary Algorithms

In principle, EC algorithms can be divided into Evolutionary Algorithms (EAs) and Swarm Intelligence (SI). Many EAs have been designed and achieved promising performance, such as Genetic Algorithm (GA), Genetic Programming (GP), and Evolutionary Strategy (ES). This thesis mainly

introduces GA and GP since they are closely related to NAS. The general process of EAs is shown in Figure 2.6. Firstly, a group of individuals/solutions is initialized. They compete to survive to the next generation: the parent individuals are selected according to their fitness, and evolutionary operators are employed to produce offspring. This process will continue until the termination criterion is satisfied, and usually, the individual with the highest fitness of the final population will be selected as the final solution.

2.4.1.1 Genetic Algorithms

Genetic Algorithm (GA) is an important branch of EAs, and it is widely employed in ENAS. In GA, the solutions are encoded into chromosomes according to specific encoding strategies, and the chromosomes compete to survive and generate better solutions [92]. Generally, the main steps of GA are as follows:

Step 1: Initialize the initial population according to the designed encoding strategy randomly.

Step 2: Evaluate the individuals with the fitness evaluation function.

Step 3: Select the promising individuals according to their fitness to be parent individuals.

Step 4: Perform crossover and mutation operations to the selected parent individuals, and generate the offspring.

Step 5: Evaluate the new offspring, and perform environmental selection to select the individuals who survive to the next generation.

Step 6: Go to step 3 if the termination criterion is not met. Otherwise, select the best individual of the current population and decode it as the optimal solution.

Commonly, the termination criterion is the maximal generation number. The crossover and mutation operators need to be designed corresponding to the encoding strategy.

2.4.1.2 Genetic Programming

Genetic Programming (GP) is another EA approach. GP solves problems by evolving computer programs [14, 64]. The individuals are commonly presented by trees in GP and have the functions as internal nodes and terminals as leaf nodes. The tree generation methods can be divided into three categories: full, grow, and ramped half-and-half. One of the three methods is commonly employed to build the initial population, and the sub-tree crossover and sub-tree mutation operators are utilized to produce offspring. Specifically, an example of the crossover process is shown in Figure 2.7. Two parent individuals produce two offspring individuals by exchanging the randomly chosen sub-trees. Figure 2.8 exhibits the mutation process. The mutation operation is utilized on a single parent individual. A sub-tree is randomly chosen and swapped with a new sub-tree generated by the initialization method.

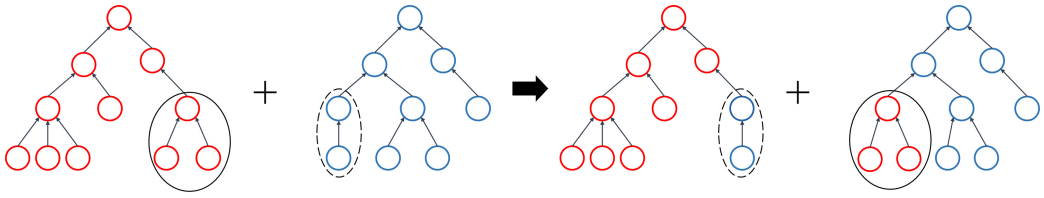


Figure 2.7: An example of the crossover operation in GP.

2.4.2 Swarm Intelligence

Swarm Intelligence (SI) is an important category of EC optimization methods. Unlike EAs, SI algorithms are mainly inspired by the self-organizing

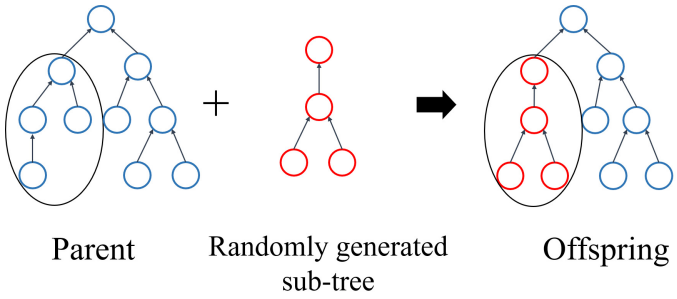


Figure 2.8: An example of the mutation operation in GP.

interaction among individuals within a group or some groups, such as animal herding, bird flocking, and ant colonies foraging. Many successful SI algorithms have been proposed, and the mainstream is the Ant Colony Optimization (ACO) algorithm [32] and the Particle Swarm Optimization (PSO) algorithm [34]. The standard ACO algorithm is often used to solve discrete optimization problems. The PSO algorithms are good at solving continuous optimization problems. This thesis transfers the discrete parameters to continuous ones by the auto-encoder to facilitate the PSO process, so the details of the PSO algorithm will be presented here.

2.4.2.1 Particle Swarm Optimization

PSO is usually used for addressing continuous optimization problems, and each particle, representing a potential solution, has both velocity and position vectors [26, 121]. The main procedures of PSO are exhibited as follows:

Step 1: Initialize the particles of the initial population randomly.

Step 2: Evaluate the particles with the fitness evaluation function.

Step 3: For each particle, select the best one from the particle's history, and record the position as $pBest_i$.

Step 4: For all particles, choose the best one from the history of all particles, and record the position as $gBest$.

Step 5: Calculate the velocity $\{v_1, ..., v_i, ...\}$ for each particle by Equation (2.3).

Step 6: Update the positon $\{p_1, ..., p_i, ...\}$ for each particle by Equation (2.4).

Step 7: Go to step 2 if the termination criterion is not met. Otherwise, return $gBest$ as the final solution.

$$v_i = w \cdot v_i + c_1 \cdot r_1 \cdot (p_g - p_i) + c_2 \cdot r_2 \cdot (p_p - p_i) \tag{2.3}$$

$$p_i = p_i + v_i \tag{2.4}$$

In Equation (2.3) and Equation (2.4) , v_i is a fixed-length vector representing the velocity of the i-th particle, and p_i expresses the position of the i-th particle. w, c_1, and c_2 are PSO parameters, which are used to fine-tune the performance of PSO. r_1 and r_2 are random numbers between 0 and 1, and p_g as well as p_p denotes the positions of $gBest$ as well as $pBest_i$, respectively.

2.4.3 Evolutionary Multi-Objective Optimization

For Evolutionary Multi-Objective (EMO) optimization methods, the sub-objectives are usually conflicting, so the improvement of one objective may negatively affect the other objectives.

2.4.3.1 NSGA-II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) [29] is a famous multi-objective optimization algorithm frequently employed for addressing problems with conflicting objectives. It provides a robust and efficient methodology underpinned by the principles of Pareto dominance.

Specifically, a solution is recognized as superior if it advances at least one objective without compromising any others. The algorithm leverages a fast non-dominated sorting technique to categorize solutions into different dominance levels. The first front contains non-dominated solutions that are optimal in all objective spaces, and subsequent fronts contain solutions dominated by those in preceding fronts. Once the sorting is over, a new population is filled by the solutions of different non-dominated fronts, and the best non-dominated front has the first priority, then is the second non-dominated front, and so on. When the last allowed front is considered, it is very likely that not all individuals can be included in the new population. Instead of discarding individuals arbitrarily, NSGA-II selects the solutions that make the diversity of the selected solutions the highest according to the crowding distances.

Specifically, the crowding distance mechanism is to maintain diversity in the population. The crowding distance is a measure of how close an individual solution is to its neighbors. A larger crowding distance indicates a larger space devoid of any other solutions, implying better diversity.

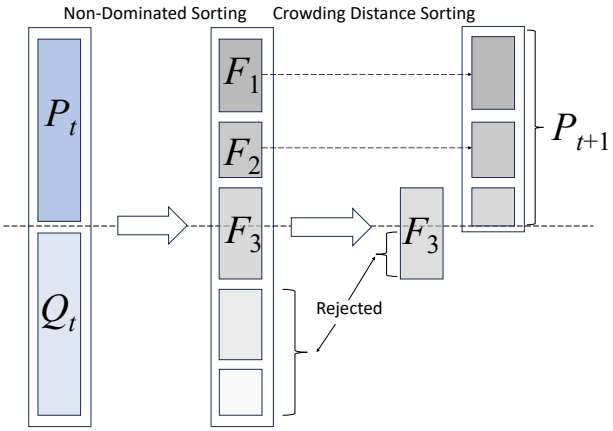


Figure 2.9: The selection process for the next iteration of NSGA-II primarily involves non-dominated sorting and crowding distance sorting. (Adapted from [29])

During the selection of individuals for the next iteration, NSGA-II predominantly utilizes non-dominated and crowding distance sorting mechanisms, as depicted in Figure 2.9. For a given population, denoted P_t, the corresponding offspring population Q_t is generated through selection, crossover, and mutation operations. Subsequently, the non-dominated sorting is applied to the combined population comprising P_t and Q_t, achieving different rank fronts, represented as $F_1, F_2, ..., F_n$. Individuals from the set F_1 are selected first, followed by those from the set F_2, until no additional sets can be accommodated. As shown in Figure 2.9, a subset of the individuals from F_3 are chosen to achieve the desired population size of P_{t+1}. Notably, within F_3, individuals with larger crowding distances are chosen.

NSGA-II has been successfully used in a variety of real-world optimization problems across diverse domains, demonstrating its effectiveness and efficiency in handling multi-objective optimization tasks [146]. Given its attributes, the NSGA-II is employed in this thesis to solve multi-objective problems.

2.5 Image Generation

In recent years, image generation has become a popular research topic, which aims to generate realistic and innovative image content through algorithms or models. The applications include image synthesis, image enhancement, super-resolution, style transformation, image filling, video generation, etc. Existing mainstream image generation models include generative adversarial networks (GANs) [42], variational autoencoders (VAEs) [8], flow-based generative models [51], and models [27]. Particularly, diffusion models become very popular recently for their excellent performance [31, 95].

The stable diffusion (SD) model [110] is a kind of conditional diffusion model [27] that excels in image synthesis tasks. It has exhibited impressive

results in class-conditional image synthesis and image inpainting. Distinctively, it showcases superior efficiency compared to traditional diffusion models in synthesizing images [181].

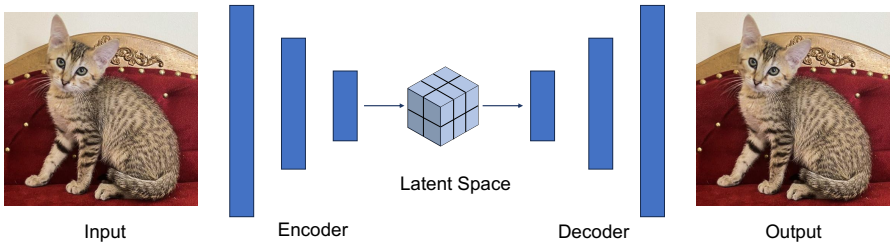


Figure 2.10: The encoder of a VAE compresses an input image to a matrix in the latent space; the decoder component can then reconstruct the image from this latent matrix.

The SD model employs a VAE to transform an image into a lower-dimensional latent matrix. The image synthesis process, or the diffusion process, takes place in the latent space rather than the pixel space, significantly boosting the model's efficiency. This efficiency gain is primarily because the latent space dimensionality is considerably smaller than that of the pixel space. Figure 2.10 depicts the usage of a VAE to compress and subsequently reconstruct an image.

Figure 2.11 depicts the synthesis process in the latent space, essentially a reverse diffusion process. Initially, a random latent space matrix is generated. Subsequently, the noise predictor estimates the noise inherent in the latent matrix, which is then subtracted from the latent matrix. This sequence of operations is repeated for a predetermined number of steps. Finally, the decoder of the VAE transforms the latent matrix into the output image. The noise predictor relies on a U-Net [111] backbone combined with a cross-attention mechanism, enabling it to learn attention-based [57] models from different text prompts.

The mechanism of the SD model renders it suitable for image inpainting tasks, where corresponding pixels in the latent matrix are fixed, while

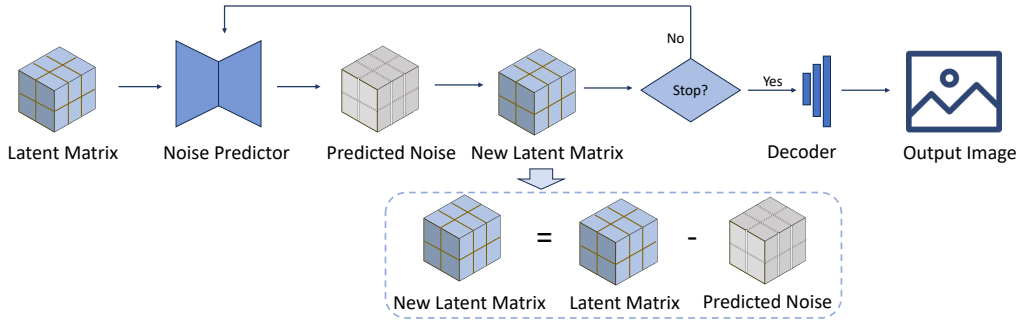


Figure 2.11: The sequence of operations involved in image synthesis within the latent space.

the others are synthesized. This distinctive characteristic motivated us to harness the SD model to generate counterfactual images that merge specific semantic meanings, assisting in explaining image classifier decisions.

2.6 Related Work

2.6.1 Evolutionary Neural Architecture Search

ENAS algorithms refer to the algorithms that employ EC methods to optimize the NAS problem [15,36,93,108,177], and researchers have employed many kinds of EC methods, such as GA, GP, PSO, and other evolutionary methods.

2.6.1.1 GA based ENAS

GA-based approaches account for the majority of existing ENAS algorithms, and many researchers developed original algorithms with state-of-the-art performance.

Some researchers fix the length of the architectures. For example, Xie *et al.* [155] designed a fixed-length GA algorithm to optimize CNN architectures for image classification. They employed fixed-length binary strings

to represent the architectures of CNNs. The number of blocks and the number of convolutional layers in each block are all predefined, which restricts the search space and limits the performance of the searched results. To break through the limitations of fixed length, Sun *et al.* [139] developed a variable-length GA algorithm to automatically evolve CNN architectures named EvoCNN. The depth of CNNs is not predefined to a specific number, which significantly increases the possibilities of the represented architectures and improves flexibility. However, the encoding strategy could only encode the CNNs with plain connections of convolutional layers, pooling layers, and fully-connected layers, not advanced structures like shortcut connections. This will limit the final performance. Then, in order to further break through the limitations of plain connections, Sun *et al.* [140] proposed another variable-length GA algorithm for ENAS called CNN-GA. CNN-GA utilized the merit of skip connections compared with EvoCNN. Specifically, CNN-GA stacked residual blocks and pooling layers according to the encoding information and achieved promising performance on the CIFAR-10 dataset and CIFAR-100 dataset.

In order to reduce the search cost, some researchers perform the search based on fixed block structures. For instance, Sun *et al.* [138] proposed a block-based GA for ENAS named AE-CNN. AE-CNN built the architecture by stacking the predefined ResNet block units, DeneNet block units, and pooling units, and there might be several residual blocks or dense blocks in each unit. The experimental results show that AC-CNN achieved good classification accuracy with a small number of parameters and short evolutionary time, i.e., 27 GPU-days in CIFAR-10.

Martín *et al.* [90] employed a GA to automatically search for deep neural network architectures along with training the parameters, such as the types of optimizers, the number of epochs, and the batch size. Note that all the search parameters are predefined to several choices.

In 2017, Fujino *et al.* [39] proposed an evolutionary method to optimize the hyperparameters in a network. Specifically, they employed AlexNet

[66] as the basic CNN architecture and used a GA to optimize its parameters, such as the number of filters, the filter size, and the pooling size. This is an early study of ENAS, and as we can see, they mainly focused on tuning the parameters of an existing network instead of exploring original architectures. Different from searching for the crucial architecture parameter, O'Neill *et al.* [99] introduced an ENAS algorithm to optimize the skip-connection structures within DenseNet blocks. This new method employed simple adjacency matrices to encode the skip-connections inside the blocks and evolved the individuals with simple crossover and mutation operators. Additionally, it introduced dynamic compression rates, which were different from traditional statistical compression rates. The experiment results show the ENAS algorithm achieves better performances than the baseline DenseNet-BC network on both CIFAR-10 and CIFAR-100. Note that this method just optimized the skip-connection structures instead of the whole network architecture.

Zhu *et al.* [179] focused on the evolutionary operators and proposed an efficient ENAS algorithm called EENA. For EENA, the crossover operations are carefully designed and are guided by the prior learning process. Similarly, the mutation operations are learned from typical architectures. In this way, EENA significantly reduces the search time.

In summary, GA is widely used in ENAS, and researchers break through the limitations of fixed-length representations and plain network connections, obtaining better performance using variable-length network representations and optimizing the skip connections.

2.6.1.2 GP based ENAS

Suganuma *et al.* [132] employed Cartesian Genetic Programming (CGP) to automatically construct CNN architectures for image classification tasks. CGP employed highly functional modules as the node functions, such as ConvBlock, ResBlock, max pooling, average pooling, concatenation, and summation. Besides, CGP only used mutation as the evolutionary opera-

tor, and there were only two offspring in each generation.

Similarly, Loni *et al.* [80] also employed CGP to optimize CNN architectures. Differently, they proposed a multi-objective CGP optimization algorithm, which considers both accuracy and model size. Specifically, after obtaining the promising network architecture, they also deployed it to the hardware devices with a limited resource budget.

Instead of utilizing CGP, many researchers also employed GP trees in ENAS algorithms because of the interpretability and flexibility of GP. For example, Zhu *et al.* [180] utilized GP trees to optimize CNN architectures. Four kinds of residual blocks with different filter sizes and convolutional layer numbers were manually designed as the terminals of GP. Four kinds of primitive functions were employed, which were capable of changing the filter number or stride size and connecting two blocks. A major drawback is that too many candidate parameters and structures are pre-defined, reducing automation.

Evans *et al.* [37] designed a multi-objective GP-based method to construct network architectures, and the two objectives are to maximize the reconstruction ability and minimize the trees' complexity. As a result, they improved the interpretability the searched networks.

2.6.1.3 PSO based ENAS

Some researchers employ PSO to evolve variable-length network architectures. For example, Junior and Yen [60] designed a novel PSO algorithm to search for variable-length deep CNN architectures, which is called psoCNN. A novel difference operator and a new velocity operator to cope with the variable-length particle. The experimental results show psoCNN takes less time than competing algorithms. Wang *et al.* [147] developed a new algorithm whose encoding strategy uses IP address to represent CNN architectures, so its name is IPPSO. IPPSO uses 'disabled layers' to transfer the variable-length architecture to a fixed-length vector so that PSO can evolve the particles. However, the disabled bits are set to 0, and the

gap between them and valid bits is big, which may harm the evolutionary process. Besides, PSO is good at processing continuous decimal vectors, but the neural architectures are discrete in nature; the representations are composed of integers, which cannot suit the PSO algorithm well. Sun *et al.* [137] proposed a flexible convolutional autoencoder with flexible numbers of convolutional layers and pooling layers and stacked them to build a CNN to solve image classification problems. The encoded CNN structures are evolved by PSO. Note that each particle keeps the same length during the evolving process as its initial length to accomplish the length diversity.

PSO can also be combined with GA for NAS. Wang *et al.* [148] developed a hybrid ENAS method called GAPSO, which employs both GA and PSO. The whole architecture is composed of a number of blocks. Specifically, PSO is employed to search for the block architectures, and the blocks are connected to build the whole network. GA is employed to explore the skip connections inside each block, for skip connections can be encoded by a binary string suitable for GA.

Instead of searching for network architectures, Lorenzo *et al.* [81] used PSO to optimize the hyper-parameters of CNNs, such as receptive field size, number of receptive fields, and stride size of each layer. This algorithm did not explore the newly generated architectures but justified that PSO is suitable for optimizing the hyper-parameters of specific architectures.

Some researchers are dedicated to improving search efficiency. Fielding *et al.* [38] proposed Swarm Optimised Block Architecture (SOBA), which can concurrently optimize the architecture and train the weights of the network. Besides, the PSO model used adaptive acceleration coefficients, and parameter sharing is employed to accelerate the optimization process. As a result, the optimization on the CIFAR-10 dataset only takes 34 hours on a single GPU. Gao *et al.* [40] proposed a gradient-priority PSO algorithm to optimize the CNN parameters that are encoded by the proposed binary

coding system, which improves the training efficiency and model performance. However, this algorithm is just used for emotion recognition and has not been tested on other image classification datasets. Wang *et al.* [149] developed an efficient ENAS algorithm utilizing the structure of the dense block, which is called EPSOCNN. EPSOCNN only encodes a single dense block and employs PSO to evolve it using a small part of the dataset. After achieving the best block structure, some blocks of the same architecture are stacked together to build the whole architecture.

The PSO-based ENAS methods face the challenge of fixed-length particle representations, and researchers have tried to break through the limitations. Besides, PSO is also flexible and can optimize network hyperparameters, block structures, and network architectures.

2.6.1.4 Other Evolutionary Computation Method-based ENAS

Some researchers foucus on designing new architecture representations and use relative simple EC strategies to evolve the candidates. Suematsu *et al.* [106] designed a simple evolutionary method to evolve graph-based CNN architectures to address image classification tasks. They encoded the architectures by graphs, and the vertices represent feature maps or activation functions, such as batch-normalization with rectified linear units or plain linear units. The edges represent identity connections or convolutions. The graphs are evolved by mutation operators. Note that the children would inherit the weights from their parents wherever possible. The results show the searched networks achieve accuracies of 94.6% and 77.0% on CIFAR-10 and CIFAR-100, respectively, which is good, but the network scales are large. The results also prove that weight inheritance can improve the evaluation efficiency significantly. Liu *et al.* [74] proposed a hierarchical genetic representation to search for network architectures. Hierarchical representations employ several motifs at different hierarchy levels, and the lower-level motifs are used as the building blocks to construct the higher-level motifs. As for the evolution process, this method

only utilizes mutation operations, and the evolved architecture is stacked several times to build a small model, then the small model is trained. The accuracy of the small model is used as the fitness of the candidate architecture. After reaching the stopping evolution criterion, the best architecture is chosen to stack into a whole large architecture.

Different from concentrating on architecture representations, Real *et al.* [105] proposed an evolutionary algorithm named 'aging evolution' to optimize CNN architectures called AmobeNet. It employs the same search space as NASNet [183] , and each cell receives the direct input from the previous cell and a skip input from the cell before the previous cell (a kind of skip-connection). The 'aging evolution' also kills the oldest model in the population and prefers the younger genotypes. AmobeNet also utilizes mutation operators to evolve individuals.

Some researchers did not employ evolutionary methods to optimize the whole structure of the network. They focused on optimizing important network parameters, such as the learning rate and the activation function. For example, Carvalho *et al.* [21] proposed AutoLR: a framework that evolves learning rate schedulers for a specific neural network architecture using structured grammatical evolution. The schedulers are functions whose inputs are the learning rate of the previous epoch and the number of the performed epochs, and they return a single learning rate. The experiment is based on a simple architecture to explore the appropriate learning rate policy. Then, the network is trained by the evolved learning rate policy, and the performance is compared with the baseline training method — using a fixed learning rate of 0.01. The results show that the proposed AutoLR could improve the accuracy of the test dataset after the same epochs of training. Bingham *et al.* [17] explored the novel activation functions for CNNs by using an Evolutionary Algorithm (EA) to search for the best individual. The proposed method employs a tree-based encoding strategy to encode the candidate activation functions. Then, novel crossover and mutation operators are used to evolve the presentations of

activation functions.

Some researchers designed a search strategy that can help improve search efficiency. For example, Ren *et al.* [107] proposed an efficient ENAS approach for image classification. The evolutionary process in EIGEN is different from traditional evolutionary algorithms. In each generation, EIGEN employs both the primary and the secondary succession to evolve individuals, and the primary succession is conducted in a larger search space compared with the second one. In order to reduce the computational cost, in each generation, EIGEN trains the individuals for a small number of epochs and extinct the bad ones two times, and only the surviving individuals would be fully trained. Additionally, in each generation, only the best individual is chosen, and the individuals of the next generation are mutated by the best individual of the former generation. Flawed individuals employ knowledge distillation to learn from the teacher network.

2.6.2 Fitness Evaluations in NAS

In NAS algorithms, the fitness of individuals needs to be evaluated, and the fitness evaluation process consumes most of the computational resources in the whole algorithm. In LargeEvo [106], 250 Graphics Processing Units (GPUs) were employed, and it took 11 days to finish running the algorithm. The prohibitive computation resources are not available for many researchers. As a result, researchers have been investigating the design of new fitness evaluation strategies to shorten the evaluation time and reduce computation. Generally, there are three categories of efficient fitness evaluation methods: shallow-training methods, surrogate-assisted methods, and one-shot methods. The first two categories will be illustrated in this section, and the related works of one-shot methods will be introduced in Section 2.6.3.

2.6.2.1 Shallow Training

The shallow training strategies are widely employed in NAS. The simplest way is to set a fixed, relatively small number of training epochs. Then, the architectures are tested to get the corresponding fitness with the weights obtained from insufficient training. Assunccao *et al.* [10] trained all the individuals for the same time each epoch, but the time could increase with the number of epochs. So *et al.* [127] stopped training weak individuals and allocated the promising individuals more training time to get a more faithful evaluation. Moreover, some researchers stopped the training until there was no significant improvement, such as [39] and [103].

Another type of shallow training is to reduce the training data and only use a subset of the data to train the candidate individuals. For example, Liu *et al.* [77] employed a subnet to explore the promising architectures and apply the searched architecture to the original large dataset by transfer learning.

Training a small-scale model is also a shallow training method. Lu *et al.* [85] reduced the number of layers and feature maps to construct a small-scale proxy, and trained the proxy to obtain the classification accuracy as the fitness of the corresponding candidate architecture.

The limitation of shallow training methods is obvious, i.e., the obtained fitness may not reflect the true performance of the candidate network architecture, which may mislead the search process and compromise the search results.

2.6.2.2 Performance Predictor

Many existing works use computationally efficient performance predictors to avoid the traditional time-consuming performance estimation process. Generally, existing predictors can be divided into two main categories: learning curves-based performance predictors and end-to-end performance predictors [79]. The first category refers to predictors that lever-

age the learning curves of neural networks during training to predict their final performance, without requiring the models to be trained to completion. For example, Rawal *et al.* [104] proposed a meta Long Short-Term Memory (LSTM) to predict networks' final performance according to partial training results.

The end-to-end performance predictors could directly predict networks' performance according to their architectures. These algorithms usually collect a group of network architectures and their corresponding performance, then a regression model is trained, and Mean Square Error (MSE) is used to measure the loss during the model training process. For example, Sun *et al.* [136] designed an offline end-to-end performance predictor based on the random forest to accelerate the fitness evaluation process in evolutionary deep learning. A set of data pairs was used to train the random forest, and the pair was composed of the encoding architecture and its performance. Later on, Sun *et al.* [135] proposed a Pairwise Ranking Indicator (PRI), employing a logistic regression model to compare the rank order between two training samples. Similarly, Wang *et al.* [150] transformed the expensive performance estimation into a binary classification task conducted by a Support Vector Machine (SVM). The SVM is used for predicting the performance comparison result between two individuals, which is enough to support the evolutionary process even without the actual fitness of each individual. In this way, the expensive computational cost is avoided.

While improving the efficiency, the predictions of the performance predictors may bring some biases compared with those obtained from fully-training the networks, which might misguide the evolution. For example, AE-CNN+E2EPP's [136] accuracy is 1.00% and 1.17% lower than AE-CNN's [138] on CIFAR-10 and CIFAR-100 [65], respectively.

2.6.3 One-Shot Neural Architecture Search

In one-shot NAS, only a single supernet needs to be trained initially, and all the candidate networks directly inherit weights from the supernet without further training. Some one-shot NAS algorithms employ continuous relaxation to parameterize the search space, and optimize the architecture parameters and supernet weights alternatively using gradient-based approaches [20]. However, this mechanism may cause interference between the architecture parameters and the supernet weights.

To address this issue, some one-shot methods separate the supernet training and the architecture search [46] into two sequential stages. In the first stage, a large supernet that encompasses the entire search space is trained. In the second stage, the optimized architecture is searched by a search algorithm, where all candidate networks (a.k.a. subnets) inherit weights from the supernet without further training, resulting in significant time savings.

Although one-shot NAS methods have reduced computational costs compared with most NAS methods, the training of the supernet still requires significant computing resources. To address this challenge, Cai *et al.* [19] proposed a progressive shrinking scheme to efficiently train a large network, but the designed supernet is nested, with small subnets embedded in large subnets, resulting in deeply coupled weights that may impair fitness evaluations. Consequently, many supernets adopt a single path mechanism [46], where each subnet is a single path inside the supernet. However, there is little research on improving the training efficiency of the supernet.

One-shot NAS also suffers from limited reliability in evaluating the performance of candidate networks (a.k.a. subnets). During the search stage, the subnets inherit weights from the supernet, and their fitness is assessed based on these weights. However, researchers have noted that the ranking correlation between inherited weights and the weights from stand-alone training is low [162], leading to unfair comparisons among

subnets. This issue has been attributed to the extensive sharing extent of weights in the supernet [96], where all subnets that contain the same operation in a layer use the same weights. These replicated weights may not be optimal for individual subnets, leading to potential degradation in performance. To mitigate this problem, some approaches have proposed constructing multiple sub-supernets that cover different regions of the search space to mitigate the co-adaptation problem [173], or using a dictionary of weights based on a group of supernets to combine weights for subnets [130]. However, these methods often require significant computational resources for training multiple supernets and may result in large memory consumption for saving weights. Additionally, the division of the search space and the effective combination of weights remain challenging issues. Another approach proposed by Zhou *et al.* [178] is to adjust the sharing extent during supernet training using a curriculum learning strategy. However, the learning strategy is complicated and introduces additional computations. Bender *et al.* [16] trained a large supernet model incorporating path dropout in order to ensure the model is robust, and then randomly zeroed out some operations and accurately evaluated the performance.

Some works focused on combining the one-shot NAS method with multi-objective optimization approaches. For instance, Yang *et al.* [160] introduced an efficient multi-objective neural architecture search method by sharing the architecture parameters with a supernet. CARS initializes a supernet with considerable cells and blocks; then, the individuals are derived from the supernet. The individuals are selected by a new multi-objective selection method called pNSGA-III, which considers the increasing speed of accuracy to protect the large-scale architectures that may perform badly during the evaluation period but has the potential for achieving higher accuracy. Corresponding crossover and mutation operators are also applied. CARS reaches a state-of-the-art performance on CIFAR-10 with a cutout preprocessing technique within only 0.4 GPU days.

2.6.4 Explainability of Deep CNNs

The explainability of deep CNNs is generally difficult, and some studies have been trying to explain the decision-making processes within these networks. The section mainly introduces three kinds of methods: gradient-based methods, class activation mapping-based methods, and model-agnostic explanation methods.

2.6.4.1 Gradient-based Methods

Gradient-based methods can offer explanations of deep CNNs' decisions by visualizing the salience regions within the input images that contribute to the decisions.

Saliency Map

Saliency Map [125] is a kind of gradient-based method, which can visualize the regions within the input image that significantly influence the CNN's output. Specifically, gradients are calculated based on the score corresponding to the true or interested class and are then visualized through a saliency map, where each pixel's intensity corresponds to its gradient

Figure 2.12: Saliency Map visualization results. (images taken from [125])

value. In this way, the pixels with higher intensity indicate they have a substantial impact on the prediction of the corresponding class, and the saliency map can show where the important regions are located.

Figure 2.12 shows the saliency maps for the top-1 predicted class in ILSVRC-2013 test images. Specifically, the input image is presented above its corresponding saliency map. The saliency maps clearly highlight the regions that contribute most to the prediction.

Guided Backpropagation

Guided Backpropagation [128] aims to show the important features that the network learns, thereby facilitating the understanding of its decisions. Different from the vanilla backpropagation, this method considers only positive gradients, discarding negative gradients by setting them to 0, which is based on the rationale that features corresponding to negative gradients are 'suppressed' by the neurons, indicating that they are not important.

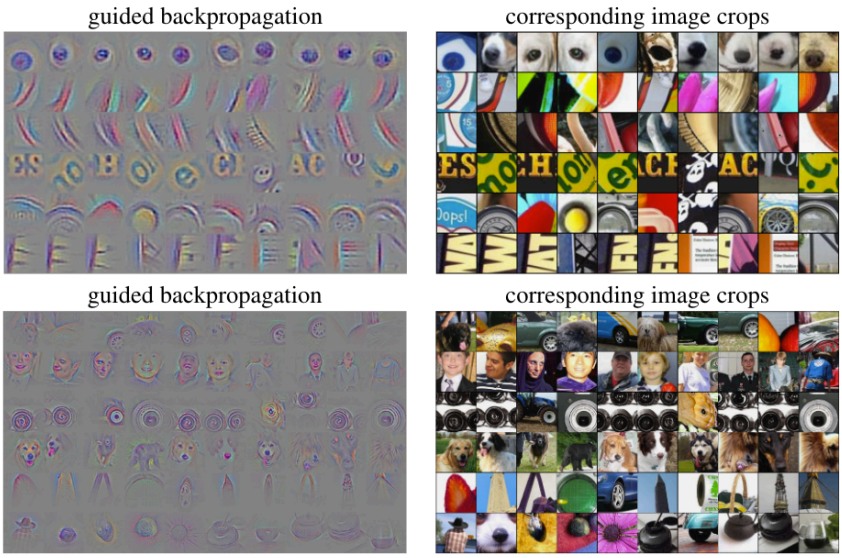


Figure 2.13: Guided Backpropagation visualization results. (images taken from [128])

Figure 2.13 presents the visualization of patterns learned from the sixth (above) and ninth (below) convolutional layers. It clearly shows that the shallow convolutional layer extracts fundamental or low-level features, and the deeper convolutional layers capture more abstract, high-level features, helping people to understand what the network/convolutional layer has learned, thus providing a kind of interpretability.

Integrated Gradient

Integrated Gradient (IG) [141] also aims to connect the output results with the input features by computing the gradients. Particularly, IG is suitable for interpreting both image processing and natural language processing models. The IG method sets a baseline first, which is defined as an *absence of a feature* in an input, and then, the calculation of the integrated gradients considers both the *sensitivity* and the *implementation invariance*.

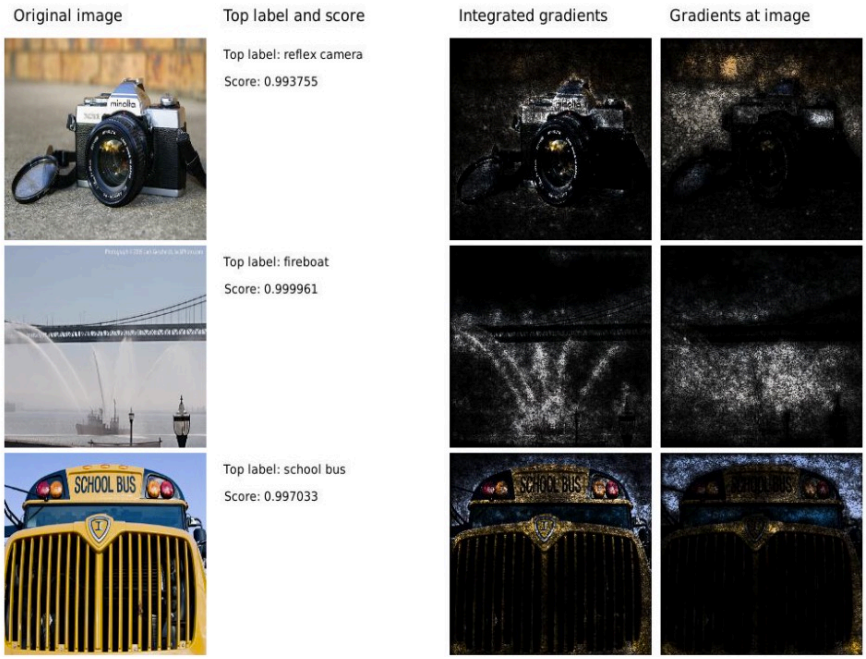


Figure 2.14: Comparison of Integrated Gradients with the vanilla gradient method. (image adapted from [141])

Using three examples, Figure 2.14 contrasts the IG method with the standard gradient method. It is obvious that the IG method can identify the salient regions in the original images more accurately than the vanilla gradient method, providing explanations for the model's predictions by signifying the important regions in the original input.

2.6.4.2 Class Activation Mapping-based Methods

Class activation mapping-based methods primarily focus on image classification tasks and generate heatmaps to indicate the importance of regions in the input images, considering the predicted results and thus providing explanations.

Class Activation Mapping (CAM)

Class Activation Mapping (CAM) [175] aims to generate a heatmap to locate the discriminative regions in the input image that are pivotal for the model's predictions for a specified class. Specifically, CAM employs a global average pooling (GAP) layer to replace the fully-connected layer(s), aiding in pinpointing the import regions.

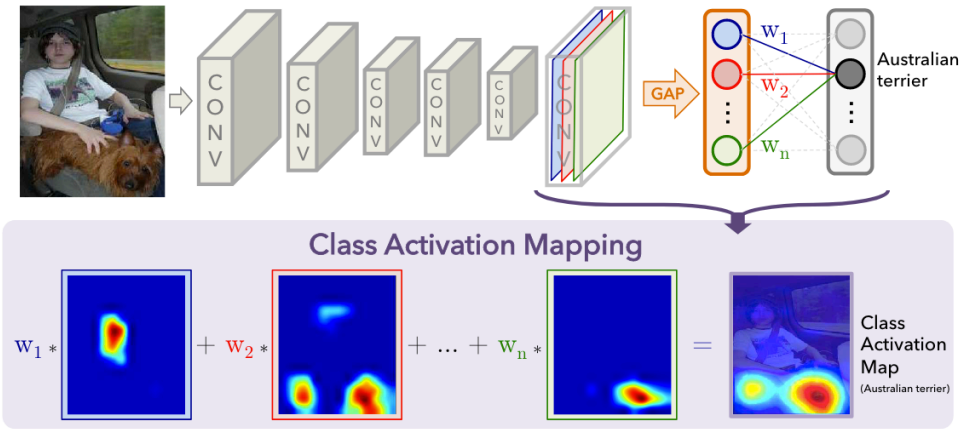


Figure 2.15: Illustration of how CAM works. (image taken from [175])

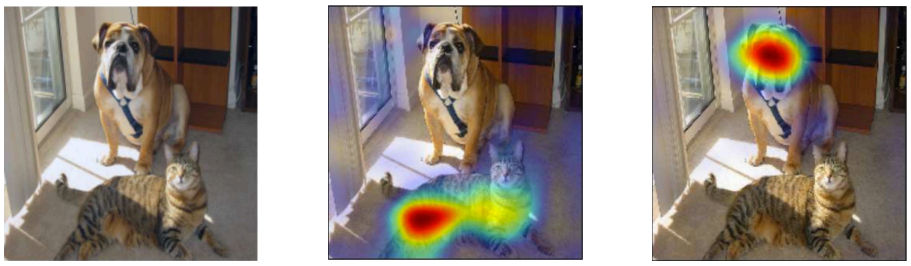
Figure 2.15 illustrates the workflow of CAM, which adds a GAP layer subsequent to the last convolutional layer of the CNN (shown in the top row), and the average value of each feature map generated by the last convolutional layer is computed. Subsequently, for the outputs of the GAP layer, linear models are trained to match them to the class labels. Once the weights of the linear models are learned, the class activation maps are synthesized by aggregating the product of the feature maps of the last convolutional layer and their correlating weights (shown in the bottom row).

In the example presented in Figure 2.15, the class activation map of the *Australian terrier* class is shown in the lower right. It shows that the head of the dog is of crucial importance (colored in red in the class activation map), and the dog's body also attracts attention from the network (colored in light blue and yellow in the class activation map). On the contrary, the other regions in the image are not quite related to the decision of *Australian terrier*.

Despite CAM's efficacy in indicating the crucial regions, the computational cost of learning the weights for the linear models is large, as N linear models have to be trained for N classes. In addition, the architectures of the CNNs need to be modified for the CAM method by removing the fully-connected layer(s) and adding the GAP layer, which is complicated.

Grad-CAM

Grad-CAM [117] also aims to interpret the knowledge learned by CNNs by visualizing the important regions for classifications. Unlike CAM, Grad-CAM does not need any model architecture modification and is suitable for a broader range of CNNs. Besides, Grad-CAM can avoid the overhead of learning the weights of the linear models. Grad-CAM utilizes feature maps from the last convolutional layer and exploits the gradient information between them and the target class to formulate the class activation map.



(a) Original Image (b) Grad-CAM 'Cat' (c) Grad-CAM 'Dog'

Figure 2.16: An example of Grad-CAM. (image taken from [117])

Figure 2.16 illustrates an example of Grad-CAM in explaining the original image containing a dog and a cat. The second figure (b) presents the results of using Grad-CAM to explain the class of 'cat', and the region where the cat locates is highlighted, signifying it contributes to the classification of 'cat' most for the classifier. Similarly, the third figure (c) shows the explanations corresponding to the class of 'dog', where the area of the head of the dog is highlighted, indicating this region is important to the prediction of the 'dog' class.

Grad-CAM++

Grad-CAM++ [22], enhanced from Grad-CAM [117], is designed to provide more fine-grained and class-discriminative visualizations, making it easier to understand and explain the decisions made by CNNs. Unlike Grad-CAM [117], where each pixel gradient has the same weight in generating the activation map, Grad-CAM++ [22] scales the pixel gradients that are important to a particular class by a larger factor and scales those that do not contribute to the prediction by a smaller factor. In this way, the 'important' pixels are better highlighted.

Figure 2.17 presents the comparisons between Grad-CAM [117] and Grad-CAM++ [22] on four example images. The above two cases belong to the situation of multiple occurrences of the same class, and Grad-CAM++ can locate all the dogs more precisely than Grad-CAM. There is only one

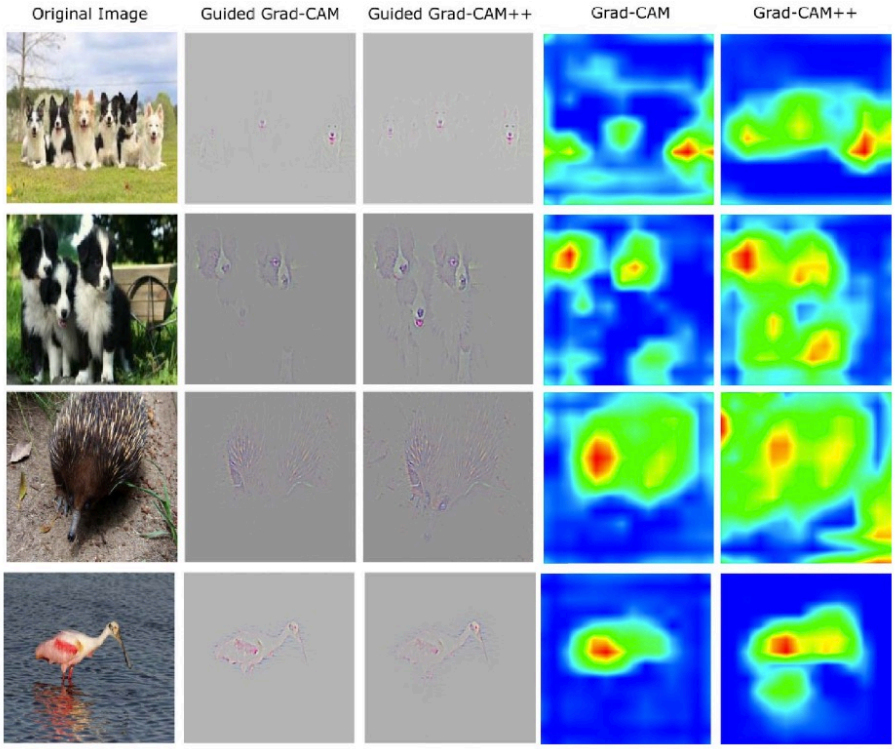


Figure 2.17: Comparative visualization between Grad-CAM and Grad-CAM++. (image taken from [22])

objective in the bottom two images, and Grad-CAM++ can present more complete contour than Grad-CAM, i.e., the nose of the hedgehog and the legs of the bird are highlighted in the heatmaps produced by Grad-CAM++. The cases show that Grad-CAM++ can present better heatmaps than Grad-CAM, thus providing more refined and precise visual explanations.

2.6.4.3 Model-Agnostic Explanation Methods

Different from the gradient-based and class activation mapping-based methods that produce explanations based on the models' weights or gradients, model-agnostic explanation methods offer interpretations for machine learning models regardless of the models' specific types/architectures.

Simple Linear Iterative Clustering (SLIC) Superpixels

Analyzing each individual pixel to pinpoint critical regions becomes computationally prohibitive due to the typically large number of pixels in images. Moreover, it is unnecessary, as approximate regions are sufficient to provide effective explanations. In this case, some methods segment the image into superpixels, forming the basis for identifying pivotal regions.

The Simple Linear Iterative Clustering (SLIC) method [4] stands out as a widely-adopted algorithm for superpixel segmentation because of its computational efficiency and adaptability in generating meaningful superpixel representations. While SLIC itself is not an explanation method, it is closely connected to several explanation techniques and is utilized in the proposed method for explanations of this thesis. A detailed overview of SLIC is presented below.

SLIC is an adaptation of the k-means clustering approach, specifically designed to efficiently generate superpixels. The algorithm operates within a five-dimensional space, encompassing the CIELAB color space and spatial coordinates, ensuring the consideration of both color and spatial proximities during clustering.

The SLIC algorithm works by first sampling equally spaced pixels across the image and considering these as cluster centers. The clusters are then iteratively refined by assigning each pixel to the cluster whose center is closest in the combined color and spatial distance metric. The distance metric D employed in SLIC is defined as:

$$D = \sqrt{d_c^2 + \left(\frac{d_s}{S}\right)^2 m^2}\,, \tag{2.5}$$

where d_c represents color distance, d_s denotes spatial distance, m is a weighting factor that balances color and spatial proximity, and S defines the grid interval. The iterative process continues until convergence, which is typically achieved in less than 10 iterations. The SLIC algorithm is advantageous due to its simplicity, efficiency, and effectiveness in generating

superpixels that adhere well to image boundaries.

Figure 2.18 presents an instance of the SLIC segmentation, with superpixel boundaries highlighted in yellow. Each superpixel potentially represents a discernible feature, which contributes to the rationale for explaining classifiers based on superpixels.

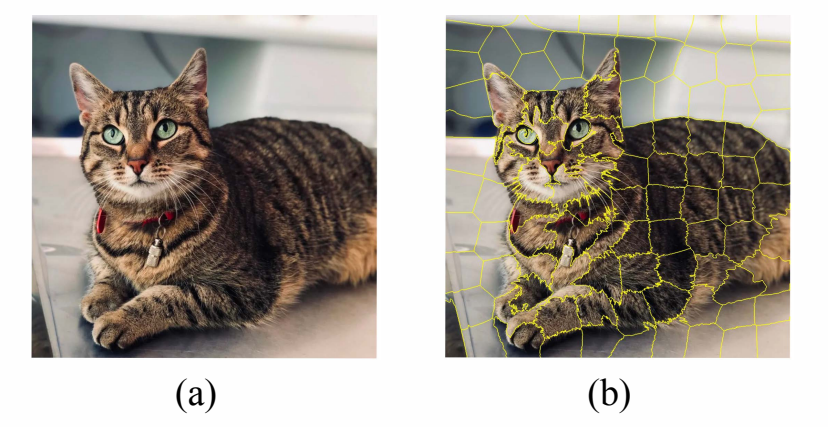


Figure 2.18: Illustration of SLIC segmentation: (a) The original image of a cat; (b) The image post SLIC segmentation.

Local Interpretable Model-agnostic Explanations (LIME)

LIME [109] is a method that can be used to explain the predictions of almost all the machine learning classifiers. The main idea is to approximate the complex model with a simpler and interpretable model, thus explaining the model's decision-making process.

To explain image classifiers, LIME segments the input image into superpixels using algorithms like SLIC. Then, LIME turns off some superpixels to generate perturbed images, which are then fed to the model to obtain the corresponding predicted scores. Subsequently, an interpretable model, such as a linear model, is trained with the perturbed images and the prediction scores. Finally, the interpretable model approximates the complex model, providing explanations. In addition, LIME generates a heatmap to

highlight the superpixels (regions) that impact the model's decision most, providing visual interpretability for the given image.

Counterfactual Generation Methods

Counterfactual generation methods can generate counterfactual images that alter the prediction probability of the classifier. Users can identify what alterations shift the model's predictions by comparing the difference between the original 'query' image and the counterfactual variant.

Goyal *et al.* [44] interchange a small patch within a 'query' image with a specific region from another 'distractor' image to generate a 'composite' image, altering the classifier's predictions (from the 'query' class to the 'composite' class) and attributing the classification result to the modified region. Figure 2.19 presents two examples of the bird breed classifications. In the first example, the head plumage of *Horned Grebe* is posted to the 'query' image, and the generated 'composite' image is classified to be

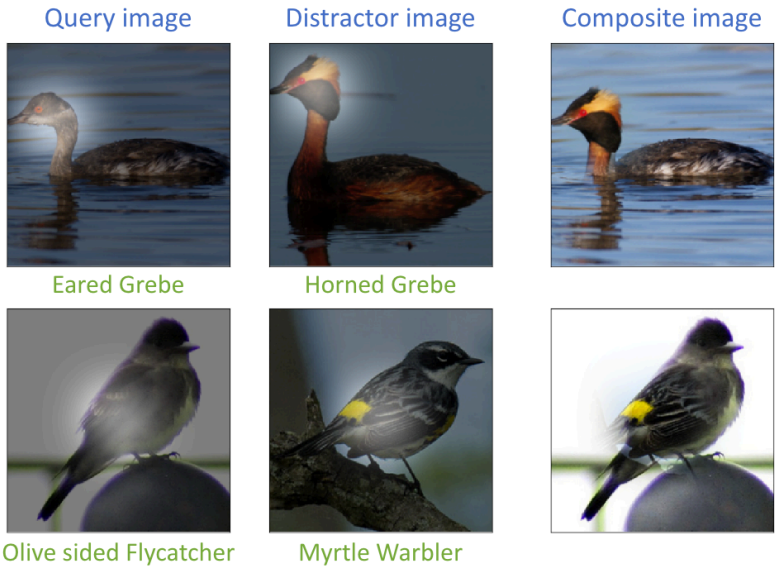


Figure 2.19: Counterfactual visual explanations for bird breeds. (image taken from [44])

Horned Grebe, implying that the head plumage is a crucial feature for the classification of *Horned Grebe*. Similarly, in the second example, the yellow wing spots of *Myrtle Warbler* are important for the classifier. Although the method is effective in identifying important features, it cannot assure that there exists a patch that can lead to a different outcome. The contours of the concatenated counterfactual image's junction regions may introduce interfering features to the classifier.

Some methods generate counterfactuals utilizing generative models like GANalyze [41], which employs GANs [42] to generate counterfactuals to study cognitive properties like memorability. Figure 2.20 shows the visualizations of GANalyze. The images in the middle columns are the baseline images, and they are modified to be characterized more (right) or less (left) by a specific property, which is quantified by the scores on the top left corner. The generated counterfactuals change the score of specific cognitive properties, but the counterfactuals alter almost all the pixels, making it challenging to determine which attribute(s) cause the different outcomes, thereby reducing the explainability.

To alleviate the above limitation, StylEx [69] generates multiple coun-

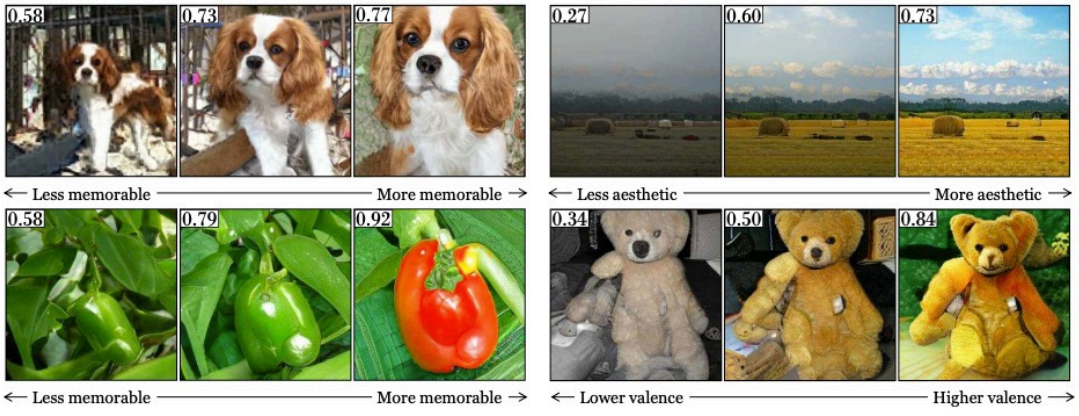


Figure 2.20: Visualizations produced by GANalyze. (image sourced from [41])

terfactual explanations for input, and each only alters one attribute. The explanations for a "cat vs. dog" classifier are presented in Figure 2.21, where the generated counterfactual images are marked with colored borders. The prediction scores of *cat* are positioned on the left top of images. These examples clearly show that features such as open or closed mouth, eye shape, and pointed or dropped ears can affect the classifier's predictions. However, a generative network named StyleGAN2 [61] needs to be trained for an input, which is time-consuming and involves parameter tuning.

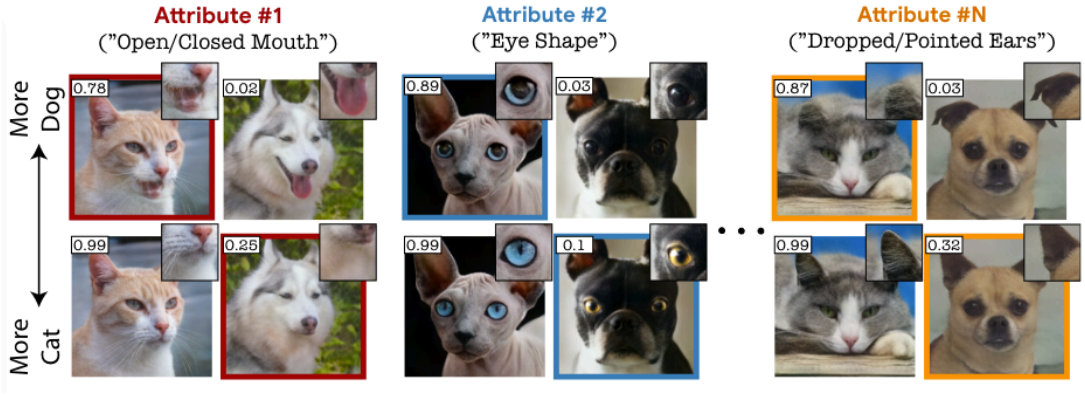


Figure 2.21: StylEx's counterfactual explanations. (image taken from [69])

2.7 Chapter Summary

This chapter first provided the essential backgrounds of the key areas involved in this research, including machine learning, image classification, architectures of CNNs and some particular networks related to this research, the concept of NAS, some famous and related EC methods, and image generation techniques.

The chapter also reviewed the related work in Section 2.6. Specifically, the ENAS methods are reviewed based on the employed EC methods, i.e.,

GA, GP, PSO, and other EC methods. Then, the fitness evaluation methods used in ENAS are introduced. The fitness evaluation is usually expensive, and researchers have proposed many strategies to alleviate this problem. Subsequently, the one-shot NAS works are reviewed, and their limitations are pointed out. Finally, the methods to explain the CNNs are reviewed. Specifically, the gradient-based methods provide explanations based on calculating the gradient information, the class activation mapping-based methods generate heatmaps to highlight saliency regions, and model-agnostic explanation methods do not rely on the weights or gradient information and can explain a broader range of models.

However, there were still limitations in the above-mentioned works, which are discussed as follows:

- The candidate architectures for NAS are expected to have various numbers of layers, because the promising depth of network architecture is hard to predict before the searching process. However, the standard EC algorithms, such as GA and PSO, are designed for processing fixed-length individuals. Many existing works have been trying to use new operations for EC algorithms to cope with variable-length individuals, which may be complicated and may affect the results of the evolution. Representing the variable-length architectures with fixed-length vectors is also a solution. In this way, the standard EC algorithms could be directly used for optimizing the architectures. However, there is little work focused on it.

- The fitness evaluation process in NAS consumes too much time. Training the networks is very time-consuming, and people have proposed many methods to reduce the computational cost, such as reducing the training epoch, using a partial training dataset, sampling the candidate network, and designing a performance predictor. However, the accuracy of the predicted fitness also decreases with the decrease of the computational cost. Research is still needed to balance

the trade-off between fitness prediction accuracy and computational cost.

- In one-shot NAS, the supernet training still consumes many computational resources, and the evaluations of the subnetworks may be inaccurate, harming the search process and the searched results. Specifically, the supernet is usually very large and difficult to train; existing methods spend a lot of consumption on the supernet training. A more efficient supernet training strategy is worth exploring. In addition, the subnets are evaluated based on the weights inherited from the supernet, but the weights might not be suitable for the subnets and could cause inaccurate evaluations.

- The interpretability of the CNNs still needs to improve. Although the counterfactual generation methods are model-agnostic and can provide straightforward explanations, some of them change all the pixels, and it is challenging to locate the critical feature(s); some involve training a generation network, which is time-consuming. Besides, there are few methods focusing on explaining the decisions between similar image classes. Classifying them is more difficult for the models, and interpretability is more important to provide insight to improve the models.

The following chapters of this thesis will show how to use evolutionary methods to tackle these issues.

Chapter 3

PSO for NAS using an Autoencoder-based Encoding Strategy

3.1 Introduction

Particle swarm optimization (PSO) [26], an effective and efficient EC algorithm, attracts a lot of attention because of its simple implementation, high efficiency, and robustness [120]. Besides, there are fewer parameters needed to be tuned in PSO, which helps improve the automation in NAS algorithms, making PSO widely used in ENAS [38,40,60,81,137,147–149]. Each particle represents a potential solution/network architecture. Generally, a standard PSO algorithm is used for processing fixed-length particles, and the length of the particles is usually associated with the depth of the corresponding network, but the optimal depth of the network is not easy to determine. To tackle this problem, some existing works [147, 150] use vectors of the same length to represent the network architectures, where some bits of the vectors can be disabled to represent no layer. However, the disabled bits are usually set to 0, and the gap between them and valid bits is big, which may harm the evolutionary process. Besides, PSO

is good at processing continuous decimal vectors. However, because the neural architectures are discrete in nature, the representations are usually composed of integers, which cannot suit the PSO algorithm well [28]. In addition, PSO is designed for continuous and fixed-length representations. However, it is not easy to represent complicated network architectures of different lengths by vectors of the same length. In this chapter, the architecture representations are transformed to fixed-length decimal vectors to suit PSO well with the help of an autoencoder.

Furthermore, the primary reason for the prohibitive computational cost in NAS is that all the candidate networks need to be trained and evaluated to get their estimated performance during the search process. Some researchers use proxies to approximate the performance of the candidates [6, 76, 116, 123, 149]. In fact, the candidate architectures may become more similar and well-performing along the search process. In this regard, assessing the candidates under proxies may not precisely identify the actual promising candidates. To alleviate this limitation, this chapter explores the effect of reducing training data used during the evolutionary process and proposes an effective hierarchal fitness evaluation method accordingly.

3.1.1 Chapter Goals

The overall goal of this chapter is to design a new PSO-based ENAS algorithm, employing a newly designed autoencoder to represent the architectures of candidate networks effectively and adopting a dynamic hierarchical fitness evaluation method to identify well-performing candidates during the search process. To achieve this goal, there are four objectives as follows:

1. Design an encoding scheme to represent the candidate block architectures to suit the search algorithm. An autoencoder could compress the variable-length discrete integer block vectors to fixed-length continuous decimal latent vectors. The latent space is expected to

be smooth and continuous, facilitating the downstream PSO search process.

2. Propose a new loss function for the autoencoder that not only considers the reconstruction loss but also takes the architecture similarity and the model scale similarity into account. In this way, the networks with a similar architecture or a similar model scale can be embedded to neighborhood regions in the latent space.

3. Investigate the effect of using different sizes of training data to estimate the fitness values and the impact on different search stages.

4. Present a dynamic hierarchical fitness evaluation method to efficiently and effectively estimate the performance of individuals during different stages of the search process.

3.1.2 Chapter Organization

The remainder of this chapter is organized as follows. Section 3.2 describes the framework and details of the proposed algorithm. Then, Section 3.3 documents the experiment design information, and Section 3.4 exhibits the experiment results and corresponding analysis. At last, the conclusions are summarized in Section 3.5.

3.2 The Proposed Method

In this section, the overall framework and the details of the proposed method will be illustrated and explained. The proposed algorithm is referred to as EAEPSO (efficient autoencoder-based PSO) for convenience.

3.2.1 Overall Framework

Figure 3.1 illustrates the overall framework of the proposed method. We employ a standard PSO search algorithm, which is shown in the yellow box in the flowchart. To facilitate the evolutionary process of PSO, the particles are expected to be fixed-length vectors of decimals. How to use fixed-length vectors to represent the variable-length candidate dense block architectures is a key focus here, and the procedure is represented in the upper green part. Besides, the procedure of the proposed efficient fitness evaluation method is represented in the left circle part.

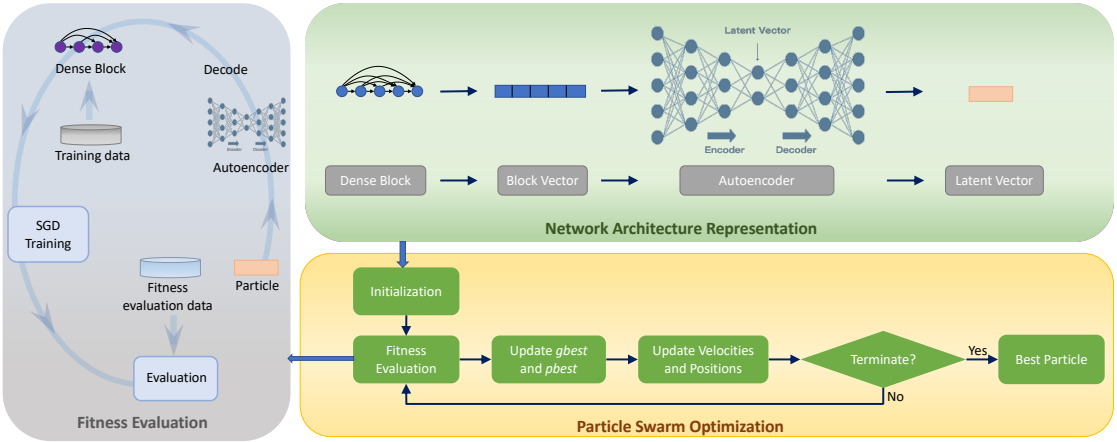


Figure 3.1: Overview: an autoencoder is used to represent the dense block architectures, a PSO algorithm is employed to implement the search process, and a new-designed fitness evaluation method is applied.

For the network architecture representation, the proposed method encodes dense blocks of different lengths into variable-length integer vectors, called block vectors (see Section 3.2.2). The block vectors are transformed by the autoencoder to fixed-length decimal-represented vectors called the latent vectors (details in Section 3.2.3). The details of the autoencoder training are illustrated in Section 3.2.4, which includes how to build the autoencoder, prepare the training samples, and the details of the

loss function. Latent vectors are the particles in the PSO algorithm and are initialized randomly for the first iteration (see Section 3.2.5).

The proposed hierarchical fitness evaluation method is used to estimate the particles' performance (see Section 3.2.6 for details). Specifically, each candidate particle will be decoded to the block vector with the help of the decoder part of the autoencoder and then to the corresponding dense block, which is a reverse procedure of the encoding process. Next, a certain proportion of the training dataset is selected to train the dense block with the Stochastic Gradient Descent (SGD) method. Finally, the dense block is measured on the fitness evaluation dataset, and the accuracy is recorded as its fitness value.

During the PSO search process, the particles are updated and evolved, which is a kind of global evolutionary search/training. Section 3.2.7 exhibits the details, and this process will continue until it meets a stopping criterion. The particle with the highest fitness will be chosen and decoded as the output dense block architecture. At last, the dense block is repeated and stacked to build the final network, and Section 3.2.8 shows how to determine the promising number of blocks.

3.2.2 Block Vectors to Encode Dense Blocks

Since the proposed method explores the number of layers and the growth rate of each layer of variable-length dense blocks [56], the hyper-parameters of the dense block need to be encoded into a vector, named block vector. Considering the common network scale and the specific hardware condition, a maximum number of layers, l_{max}, of a dense block should be defined. Besides, the minimum number of layers, l_{min}, should also be defined because too few layers in a block would not be able to learn powerful feature maps. The length of the block vector represents the length/number of layers in the dense block, so l_{max} and l_{min} are also the maximum and minimum length of a block vector. A block vector can be any length be-

tween l_{min} and l_{max}, where each element represents the growth rate of a corresponding layer.

3.2.3 Latent Vectors Transformed by Autoencoder

The block vector representations form a discrete search space, similar to most existing NAS algorithms. The reason is that the network architectures are discrete in nature. However, if we can map the discrete search space to a continuous latent space, the search in the latent space would be more efficient. In addition, if the dimension of the representation in the latent space can be reduced over the block vector representation, it will further improve the search efficiency.

Since PSO can easily process fixed-length vectors, the block vector should be transformed into a fixed-length latent vector. Although we can use fixed-length discrete strings or vectors to represent dense blocks of different lengths by padding zeros to represent the non-existing layers, this will cause significant gaps between the representations of existing layers and non-existing layers, i.e., the large difference between the existing layer's growth rate and zero, which may harm the downstream search process. Besides, PSO performs well in a continuous search space, so it would be better if the latent vector can be represented by decimal numbers. The proposed encoding strategy employs an autoencoder to extract features in the middle layer as the latent vector to achieve a fixed-length decimal-represented vector. Figure 3.2 illustrates an example of how the autoencoder transforms a variable-length block vector to a fixed-length latent vector. First of all, the maximum length of the input block vector l_{max} is set to 10 in the example in Figure 3.2, but it will be determined according to the available hardware conditions in the experiments and the applications. Secondly, to feed the variable-length block vector to the autoencoder, each variable-length block vector is padded with 0s to reach the length of l_{max}. Two 0s are padded into the block vector in the exam-

ple. Thirdly, the block vector is processed by the encoder part, composed of several fully-connected layers, and the number of nodes gets smaller. The encoder finally outputs a fixed-length latent vector, consisting of four decimals in the example. It can be directly fed into the PSO algorithm, i.e., latent vectors encoded by the autoencoder will be the particles in PSO. The gaps between different integers and between existing and non-existing layers (represented by 0s) in the block vector representations no longer exist in the latent vector. Finally, with regard to the decoding from the latent vector to the block vector, the decoder part marked in Figure 3.2 is used to perform the translation to obtain blocks.

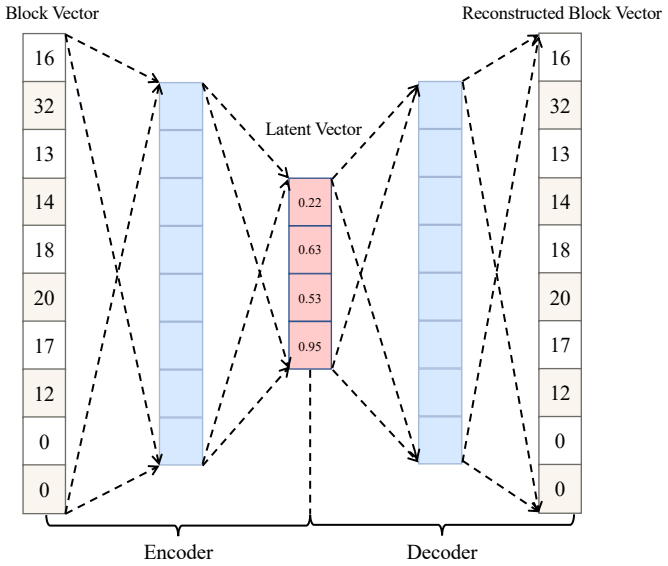


Figure 3.2: Autoencoder to transform a block vector to a latent vector.

There are several advantages of the transformation from block vectors to latent vectors. Firstly, since the autoencoder is a well-known approach for feature construction [23], it has the ability to construct meaningful features from the input vector. Therefore, the feature variables could well represent the block vector. Secondly, the autoencoder in the proposed encoding strategy has fewer dimensions in the feature variables than that of

the input vector, so the dimensionality has been reduced in the transformation. Thirdly, the discrete values of growth rates in the block vector are transformed to continuous values in the latent vector, which suits the PSO algorithm better.

3.2.4 Training of the Autoencoder

3.2.4.1 Building the Autoencoder

Generally, an autoencoder is built following the common convention, i.e., fully-connected layers are employed in both the encoder and decoder parts [50]. In particular, we consider the latent vectors' distribution, i.e., the particles' distribution in the PSO process, while designing the architecture of the autoencoder, so a batch-normalization layer is added to the end of the encoder part. This layer would adjust all the elements of the same place in a batch to obtain a Gaussian distribution with a mean of 0 and a variance of 1. In this way, the architecture representation in the latent space will become smooth, which may facilitate the downstream search on the latent space [159].

3.2.4.2 Sampling Training Data

Before the PSO search process, the autoencoder is independently trained. Specifically, while training the weights of the autoencoder, we consider the relationship between the similarity among block vectors, and that among latent vectors. Considering the similarity, the autoencoder is supposed to be trained by pair-wised vectors. So we need to create two datasets of the same size. During training, the same number of block vectors are randomly selected from the two datasets to compose pair-wised training data for each batch.

Each dataset is composed of a number of block vectors, and the block vector data generating process is shown in Algorithm 1. First of all, block vectors need to be defined by a few parameters. Besides the maximum

and minimum number of layers l_{max} and l_{min} mentioned in Section 3.2.2, the maximum growth rate g_{max} and the minimum growth rate g_{min} are also needed. The convolutional layer will not extract enough meaningful features if the corresponding growth rate is too small. While the maximum growth rate mainly depends on the hardware resource, a too-large maximum growth rate may result in the issue of out of memory in Graphic Processing Units (GPUs) and also increase the computation time. A less complex dataset would typically require a lower maximum growth rate.

Algorithm 1: Generating Block Vectors for Autoencoder Training

Input: The minimal and maximal number of layers l_{min}, l_{max}; the minimal and maximal growth rates g_{min}, g_{max}; the number of instances n.

Output: The block vector dataset D_b.

1 $D_g \leftarrow \emptyset$;
2 **for** $i = 1;\ i \leq n;\ i \leftarrow i + 1$ **do**
3 $\quad block\ vector \leftarrow \emptyset$;
4 $\quad n_{layer} \leftarrow$ Randomly generate an integer between $[l_{min}, l_{max}]$;
5 $\quad$ **for** $j = 1;\ j \leq n_{layer};\ j \leftarrow j + 1$ **do**
6 $\quad\quad n_{growth} \leftarrow$ Randomly generate an integer between $[g_{min}, g_{max}]$;
7 $\quad\quad block\ vector \leftarrow block\ vector \cup n_{growth}$;
8 $\quad$ **end**
9 $\quad$ **for** $k = n_{layer} + 1;\ k \leq l_{max};\ k \leftarrow k + 1$ **do**
10 $\quad\quad block\ vector \leftarrow block\ vector \cup 0$;
11 $\quad$ **end**
12 $\quad D_b \leftarrow D_b \cup block\ vector$
13 **end**
14 **Return** D_b.

The block vectors are generated repeatedly until reaching the predefined number of instances in the dataset n (lines 2-13). Specifically, for a

block vector, the specific number of layers is randomly chosen from the predefined range (line 4), and the number of feature maps of each layer is within the minimal and maximal growth rates to give the same possibilities to each possible value (line 6). Furthermore, the proposed algorithm sets the last few layers' growth rates to 0 by looping from $n_{layer} + 1$ to l_{max} (lines 9-10). Finally, all of the generated data are stored (line 12) and will be used to train the autoencoder.

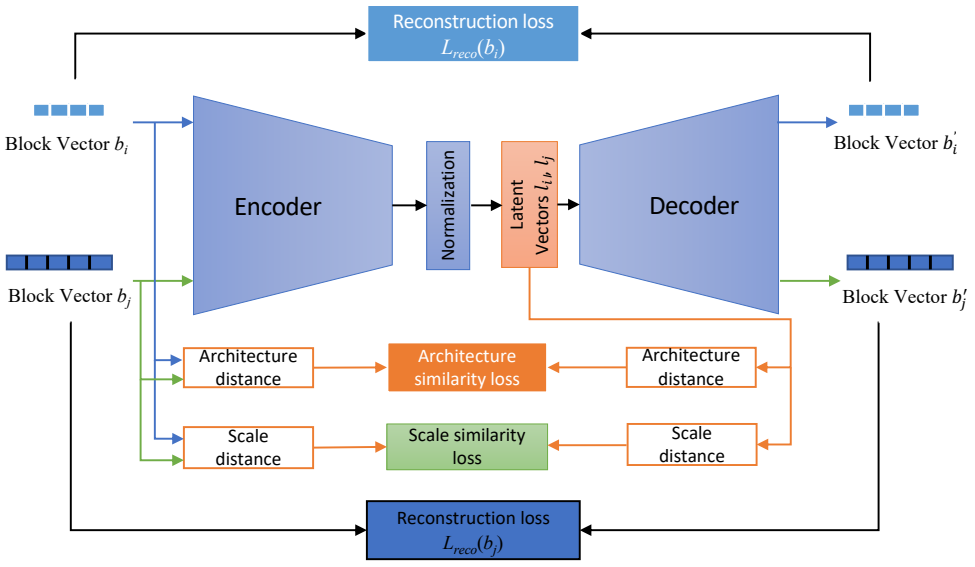


Figure 3.3: An example of the autoencoder training: Two block vectors are corresponding to two different dense blocks, and they are input into the autoencoder. The encoder part generates two latent vectors, respectively. The decoder part outputs two reconstructed block vectors. The reconstruction loss, architecture similarity loss, and scale similarity loss are considered.

3.2.4.3 The Loss Function

A new loss function is designed, and Figure 3.3 exhibits four different kinds of loss when using a pair of block vectors to train the autoencoder. The overall loss function of two block vectors b_i and b_j is shown as:

$$L(b_i, b_j) = L_{reco}(b_i) \ + \ L_{reco}(b_j) + L_{arch}(b_i, b_j) + L_{scale}(b_i, b_j) \ , \tag{3.1}$$

where $L_{reco}(b_i)$ and $L_{reco}(b_j)$ are the reconstruction loss of b_i and b_j, respectively. They can be calculated according to Equation (3.2). $L_{arch}(b_i, b_j)$ is the arechitecture similarity loss, which is calculated according to Equation (3.6), and $L_{scale}(b_i, b_j)$ represents the scale similarity loss between b_i and b_j, which is calculated according to Equation (3.9).

Specifically, the original target of an autoencoder is to minimize the reconstruction loss. The reconstruction loss of a block vector b_m is shown as:

$$L_{reco}(b_m) = \sum_{k=1}^{l_{max}} (b_{m,k} - \mathcal{A}(b_{m,k}))^2 \ , \tag{3.2}$$

where $\mathcal{A}(\cdot)$ refers to the autoencoder's processing, $b_{m,k}$ is the k-th bit in the m-th block vector, and l_{max} is the maximal length of block vectors.

To project the similar architecture to a neighborhood latent space, we also designed the architecture similarity loss L_{arch} and the scale similarity loss L_{scale} considering the similarity/difference between different dense blocks. If the dense blocks are similar, they usually lead to similar performance [161]. In this regard, their corresponding latent vectors are also expected to be similar. If the particles of similar fitness are distributed in the neighborhood space, the search could be more efficient and effective. To measure the similarity among dense blocks, we consider both the architectures and the model scales. In the employed block representation strategy, the differences among the candidate block architectures are the number of layers and the number of feature maps of each layer; we use the difference

in each layer's number of feature maps to define the architecture difference and employ the sum of all layers' feature maps to approximate the model scale. In this way, both the depth and the width of the networks are taken into account when approximating the model scale.

To calculate the architecture similarity loss L_{arch}, we need to calculate the architecture distance first. As the element values in latent vectors are between 0 and 1, we also normalize the element values in block vectors to the same range according to Equation (3.3) for the convenience of calculation, where $b^*_{m,k}$ is the k-th element value in the m-th block vector after normalization. g_{min} and g_{max} indicate the minimal and maximal growth rates, representing the minimal and maximal number of feature maps in dense block layers, respectively. Equation (3.4) and Equation (3.5) calculate the architecture distance between two block vectors b_i and b_j and between two latent vectors l_i and l_j, where l_{max} and h_{max} are the maximal length of the block vectors and latent vectors, respectively. $b^*_{i,k}$ and $b^*_{j,k}$ are the k-th element values of the i-th and j-th normalized block vectors, and $l_{i,k}$ and $l_{j,k}$ are the k-th element values of the i-th and j-th latent vectors. The architecture similarity loss is calculated according to Equation (3.6).

$$b^*_{m,k} = \frac{b_{m,k} - g_{min}}{g_{max} - g_{min}} \tag{3.3}$$

$$D_{arch}(b_i, b_j) = \frac{\sum_{k=1}^{l_{max}} \mid b^*_{i,k} - b^*_{j,k} \mid}{l_{max}} \tag{3.4}$$

$$D_{arch}(l_i, l_j) = \frac{\sum_{k=1}^{h_{max}} \mid l_{i,k} - l_{j,k} \mid}{h_{max}} \tag{3.5}$$

$$L_{arch}(b_i, b_j) = [D_{arch}(b_i, b_j) - D_{arch}(l_i, l_j)]^2 \tag{3.6}$$

For the scale similarity loss L_{scale}, the model scale distance between b_i and b_j and two corresponding latent vectors' scale distance are given by Equation (3.7) and Equation (3.8). The overall scale loss between b_i and b_j is represented by Equation (3.9).

$$D_{scale}(b_i, b_j) = \sum_{k=1}^{l_{max}} b^*_{i,k} - \sum_{k=1}^{l_{max}} b^*_{j,k} \tag{3.7}$$

$$D_{scale}(l_i, l_j) = \sum_{k=1}^{h_{max}} l_{i,k} - \sum_{k=1}^{h_{max}} l_{j,k} \tag{3.8}$$

$$L_{scale}(b_i, b_j) = [D_{scale}(b_i, b_j) - D_{scale}(l_i, l_j)]^2 \tag{3.9}$$

3.2.5 Particle Initialization

Algorithm 2 shows how to initialize the particles at the beginning of the PSO searching process. Particles are generated until reaching the predefined population size N (lines 2-6). For each particle, a block vector is randomly generated according to lines 3-12 in Algorithm 1 (line 3 in Algorithm 2). It is then encoded by the trained autoencoder to the corresponding latent vector (line 4). Finally, the first population with randomly generated particles will be generated.

Algorithm 2: Particle Initialization

Input: The population size N, the trained autoencoder.

Output: Initialized population P_0.

1 $P_0 \leftarrow \emptyset$;
2 **for** $i = 1;\ i \leq N;\ i \leftarrow i + 1$ **do**
3 $\quad b_i \leftarrow$ Randomly generate a block vector;
4 $\quad l_i \leftarrow$ Encode b_i by the encoder of the trained autoencoder;
5 $\quad P_0 \leftarrow P_0 \cup l_i$;
6 **end**
7 **Return** P_0.

3.2.6 Fitness Evaluation

Fitness evaluations are the most time-consuming process in ENAS algorithms. We investigated the use of reduced training data to propose an efficient and effective hierarchical fitness evaluation method. In most cases, training the candidate networks with a reduced dataset is likely to lead to lower test accuracy than training using the whole dataset. However, the fitness evaluation in the EC technique is usually used to guide the selection, which aims at evaluating how good each individual is to identify which network is better, so we care more about the relative goodness of the individuals, i.e., the relative fitness ranking relationship among individuals. This could be achieved by a (surrogate) measure if the measure can correctly identify the relative goodness of the individuals. There are three main considerations:

(1) Reducing the training dataset often reduces the test accuracy of the candidate, along with harming the ranking consistency between the ranking based on estimated performance and the ranking based on true performance. The less training data is used, the lower ranking consistency may lead to.

(2) The approximate ranking performance received by using reduced training data could also guide the evolutionary process, primarily at the beginning of the evolution because there is considerable variation among candidates.

(3) The best candidate found by more training data usually also performs well if trained by less training data. However, the best candidate found with less training data may not be the best one when given more training data.

Based on the above three considerations, we designed a new efficient and effective hierarchical fitness evaluation method for the PSO algorithm, which dynamically increases the amount of the training data along with

the evolutionary process. At the beginning of the evolution, in order to reduce the computational cost, we develop a basic fitness evaluation method, which only employs a small number of training data instances and consumes cheap computing resources. As there is a lot of variation among particles at the beginning of evolution, and the performance is not promising, the basic fitness evaluation method could guide the evolution effectively. As the evolution goes, when the basic fitness evaluation cannot further guide the searching process effectively, the developed progressive fitness evaluation method will be used, and its main idea is to use more training data to improve the rank consistency and select local and global best particles more accurately.

Specifically, the basic fitness evaluation method is presented in Algorithm 3. According to the predefined proportion r_b, the training data is randomly selected from the whole training dataset (line 1). Next, every particle in the population is evaluated (lines 2-12). Specifically, the particle is decoded to the corresponding dense block vector by the decoder part of the autoencoder (line 3) and then is interpreted to the dense block (line 4). The dense block will be trained until the training epoch reaches the predefined maximal epoch, or the training loss is lower than the predefined criterion (lines 6-9). The dense block will not be evaluated on the fitness evaluation dataset until the training is finished. This is different from some existing performance estimation methods, which evaluate the performance after each training epoch and choose the highest accuracy as the corresponding fitness. This is because at the early stage of the training, the candidate's performance on the evaluation dataset is relatively poor, but it will get better along with the training process. So, it will consume much unnecessary time if the evaluation is performed for each epoch.

The weight parameters and the training loss will be updated during the training (lines 7-8). After training, the candidate block will be evaluated (line 10), and its fitness is updated by the accuracy on the evaluation dataset (line 11).

Algorithm 3: Basic Fitness Evaluation

Input: The population P, the maximum training epochs k_{bmax}, the training loss criterion l_{bt}, the proportion of the training data r_b, the whole training dataset D_{train}, the fitness evaluation dataset D_{fit}.

Output: The population with fitness.

1 $D_{btrain} \leftarrow$ Randomly select r_b of D_{train};
2 **for** p *in* P **do**
3 $\quad reconstructed\ block\ vector \leftarrow$ Decode p by the decoder of the autoencoder;
4 $\quad block \leftarrow$ Decode $reconstructed\ block\ vector$ to the dense block;
5 $\quad epoch,\ loss \leftarrow 0,\ +\infty$;
6 $\quad$ **while** $epoch \leq k_{bmax}$ and $loss \geq l_{bt}$ **do**
7 $\quad\quad$ Apply SGD to train $block$ on D_{btrain};
8 $\quad\quad loss \leftarrow$ Update by the training loss;
9 $\quad$ **end**
10 $\quad acc \leftarrow$ Evaluate $block$ on D_{fit};
11 $\quad p \leftarrow$ Update the fitness of p;
12 **end**
13 **Return** P with the updated fitness.

Along with the evolution, the basic fitness evaluation method may not accurately evaluate the individuals' performance, and a progressive fitness evaluation method will then be employed. The details of the proposed progressive fitness evaluation method is presented in Algorithm 4. As mentioned earlier, the candidates that perform well on a larger training dataset usually also achieve good performance on a less training dataset. Two training datasets with different scales are used to evaluate the fitness to balance efficiency and effectiveness. The progressive training dataset D_{ptrain} is built according to the predefined portion r_{prog} (line 1), and the scale is larger than the basic training dataset mentioned in Algorithm 3.

Algorithm 4: Progressive Fitness Evaluation

Input: The population P, the population size N, the maximum progressive training epochs k_{pmax}, the progressive training loss criterion l_{pt}, the proportion of the progressive training data r_{prog}, the whole training data D_{train}, the fitness evaluation dataset D_{fit}.

Output: The population with both basic fitness and progressive fitness.

$D_{ptrain} \leftarrow$ randomly select r_{prog} of D_{train};
for $i = 1;\ i \leq N;\ i \leftarrow i + 1$ **do**
    Employ Algorithm 3 to achieve the acc_i on $Dfit$;
    $p_i \leftarrow$ Update the basic fitness with acc_i;
    $block_i \leftarrow$ Save the model with the trained weights;
end
for $i = 1;\ i \leq N;\ i \leftarrow i + 1$ **do**
    if acc_i is in the top third of P **then**
        Load $block_i$;
        $epoch, loss \leftarrow 0, +\infty$;
        while $epoch \leq k_{pmax}$ and $loss \geq l_{pt}$ **do**
            Apply SGD to train $block_i$ on D_{ptrain};
            $loss_p \leftarrow$ Update by the training loss;
        end
        $acc_p \leftarrow$ Evaluate $block$ on D_{fit};
        $p_i \leftarrow$ Update the progressive fitness with acc_p;
    else
        $p_i \leftarrow$ Update the progressive fitness with 0;
    end
end
Return P with the updated basic and progressive fitness.

Then, all the particles are trained by the basic training dataset (lines 2-4) to get their basic fitness, which is similar to Algorithm 3, but their corresponding model and trained weights will be saved (line 5). The basic fitness is used for selecting the local best particle. Next, each particle's progressive fitness is evaluated (lines 7-20). If the particle's basic fitness is good, it is more likely to perform well when training by the progressive training dataset. To save the computational resources, we only select the top third of particles (line 8), which will be further trained (lines 9-16). Specifically, the model trained by the basic training dataset is loaded (line 9) and is further trained by the larger progressive training dataset (lines 10-14). Then the network is measured by the fitness evaluation dataset (line 15), and the performance is recorded as the progressive fitness (line 16). For the particles with poor basic fitness, their progressive fitness is directly set to 0 (line 18), which can improve the search efficiency.

3.2.7 Evolving Dense Blocks

After the encoding transformation, fixed-length latent vectors are achieved, which can feed to the PSO algorithm. Then the dense blocks evolve in the PSO process based on the basic and progressive fitness evaluation methods mentioned in Section 3.2.6. Please note that there is no fixed training data proportion for the progressive fitness evaluation method. A set of gradually increasing progressive training dataset proportions will be provided as $[r_{prog1}, ..., r_{progn}]$.

Specifically, the population is initialized according to Section 3.2.5 first. After that, all the particles are evaluated based on Algorithm 3 to get their basic fitness, and then select/update the personal best particle for each one and select/update the global best one for the population, and all particles are then updated. All the particles will be evaluated by the basic fitness evaluation method and updated repeatedly until the global best fitness has not increased for five consecutive iterations, when the current evolu-

tion is considered not to be able to guide the search further. The particles will be evaluated by the proposed progressive fitness evaluation method according to Algorithm 4 with r_{prog1} as the training data proportion, and then personal best particles and the global best one are updated based on the basic fitness and progressive fitness respectively, and all the particles are then updated. Similarly, particles will be repeatedly evaluated and updated until global best fitness has not been updated for five consecutive iterations. Then, the next progressive training proportion r_{prog2} is used to fit the progressive evaluation method and help to evaluate and update the particles. The evolutionary process will continue until the last progressive training proportion r_{progn} is used. Finally, the global best particle $gBest$ is selected and decoded to the network architecture as the evolved dense block.

3.2.8 Stacking Dense Blocks

The evolved dense block needs to be stacked to get the whole network, and the promising number of blocks is hard to be determined manually. The entire training dataset is employed, and a simple grid search process is designed to determine how many blocks should be stacked together.

Before the search process, a maximum number of blocks s_{max} needs to be predefined considering the hardware resources and the image size. If s_{max} is too big, the hardware storage limit may be overloaded. Besides, a pooling layer is usually attached to each dense block, which means the size of the feature maps will be reduced to half after processing by a dense block. The minimum limit of the size of the feature maps is 1×1, so the initial input image size constrains s_{max}.

Specifically, several networks are built using the repeated dense blocks, and the numbers of blocks are increased from one to s_{max}. For each network, 80% of the training dataset is selected as the training part, and the other 20% is selected as the evaluation part. Each network is trained by

the training part and further measured on the evaluation part, and the performance on the evaluation part is used to evaluate the network. The network with the best performance on the evaluation part is selected as the final solution to the specific task.

3.3 Experiment Design

3.3.1 Benchmark Datasets

This chapter uses three common image classification datasets to evaluate the performance of the new algorithm. Among them, CIFAR-10 and CIFAR-100 [65] are selected to validate the effectiveness and efficiency of the proposed EAEPSO algorithm; CIFAR-100 and ImageNet are utilized to assess the performance of the network identified on CIFAR-10, demonstrating the notable transferability of the architecture searched by EAEPSO. These datasets are of different data scales and classification difficulties, helping to evaluate the proposed algorithm more comprehensively. Besides, many famous manual-designed networks and NAS algorithms are tested on them, which can offer a good comparison.

There are $60\,000$ color images in the CIFAR-10 dataset. The image size is 32×32, and there are three channels: R, G, and B. $50\,000$ images are used for training, and the other $10\,000$ images are for assessing the model's performance. The number of images in these ten categories is evenly distributed, i.e., for each category, there are 5000 training images and 1000 test images. As for CIFAR-100, the image size and the number of images are the same as CIFAR-10. However, there are 100 categories, i.e., there are only 600 images for each category, and 500 of them are used for training. More classes and fewer training examples for each class make the classification task on CIFAR-100 much more challenging than on CIFAR-10.

ImageNet comprises roughly 1.2 million images spanning 1000 unique categories. These images have undergone meticulous hand-annotation to

ensure the quality of their labels. Commonly, images from this dataset are resized to standardized resolutions for research and benchmarking purposes [67]. The dimension of 224×224 pixels is a frequently adopted resolution for numerous deep learning models.

The data augmentation technique is not employed during the NAS process but is performed for the post-search training process, because the fitness evaluation method used in the NAS process could effectively compare the performance of different candidate networks even without data augmentation. The augmentation method used in post-search training not only includes cropping, but also includes *cutout* [30]. However, we did not employ *dropout* or *scheduled path dropout* [183] techniques to further improve the performance.

3.3.2 Peer Competitors

To verify the effectiveness and efficiency of the proposed algorithm, we select some state-of-the-art algorithms and compare them with EAEPSO. These peer competitors could be broadly categorized into two categories according to whether the CNN is designed manually. In the first category, the CNNs are designed by human experts. They are FractalNet [70], Maxout [43], ResNet [48], DenseNet [56], Highway Network [129], VGG [126], SENet [54], and WRN-18 [165]. Most competitors are trained and tested on both CIFAR-10 and CIFAR-100. The second category automatically searches and constructs the network architectures, such as CGP-CNN [132], NAS [182], Large-scale Evolution [106], Block-QNN [174], MetaQNN [13], EIGEN [107], CNN-GA [140], PNASNet [73], AmoebaNet [105], EAS [18], NASNET [183], AECNN [138], DENSER [11], GeNet [155], CoDeepNEAT [91], Hier. repr-n, evolution [74], EffPnet [150], DARTS [75], NSGANet [84], LEMONADE [35], NSGANetV1-A1 [85], Proxyless NAS [19], AECNN+E2EPP [136], EffPnet [150], and MobileNet [115].

3.3.3 Parameter Settings

The parameter settings in EAEPSO follow the convention of the PSO algorithm and deep learning, and we also consider the computational capability of the resources available — all the experiments are implemented on the GPU cards of NVIDIA A6000.

Specifically, while employing the autoencoder to represent the dense block architecture, the minimal and maximum length of block vectors are set to 10 and 20 considering the hardware limit and the model scales of the prevalent networks [56, 150, 165]. Considering the autoencoder's scale and the convention of dimension reduction [50], the latent vectors' length is set to 8. In block vectors, the growth rates are between 10 and 32, which is similar to the width of the networks that work well. When building the dense block, following the convention, all the convolutional layers' kernel sizes are set to 3, and the stride is 1. As sampling block vectors is relatively cheap, we sample $300\,000$ block vectors as the training data for the autoencoder training, and they are trained by an Adam optimizer [33] with a learning rate of 5×10^{-3} for 500 epochs.

For the evolutionary process, the population size is 30. The regular parameters of PSO follow the convention: the inertia weight is 0.7298, and the two acceleration coefficients are both 1.49618 [145]. The velocity range is between -0.1 and 0.1. In the proposed dynamic fitness evaluation method, the proportion for the basic evaluation is 10%, and the maximal number of training epochs is set to 60. In progressive fitness evaluation, the number of training epochs is 50, and there are two predefined training dataset proportions: 20% and 40%. When stacking the block, the maximum number of blocks is 5.

For the post-search training process, the settings are followed [56]. Specifically, the searched network is trained for 300 epochs, and the initial learning rate is set to 0.1, which changes to 0.01 and 0.001 when the training epoch achieves 150 and 225, respectively.

3.4 Results and Analysis

In this section, the experimental results of EAEPSO against peer competitors are presented in Section 3.4.1. Then, the comparison results and corresponding analyses between the proposed autoencoder and the comparative one are presented in Section 3.4.2. The comparison among different fitness evaluation methods and different training dataset scales' impact on the rank consistency during the evolutionary process are investigated in Section 3.4.3. At last, the rational exploration of stacking the blocks is provided in Section 3.4.4.

3.4.1 Overall Results

EAEPSO is run on both CIFAR-10 and CIFAR-100 five times with different initial random seeds, respectively. Besides, the block architecture searched on CIFAR-10 (with the median error rate) is transferred to CIFAR-100 and ImageNet to prove its transferability. The results of EAEPSO are compared with manually designed methods from two aspects, i.e., the predictive error rate and the model size. In order to demonstrate the efficiency of EAEPSO, we also compare the computational cost with the NAS peer competitors, which is measured by the GPU-days[2] of the searching process. Please note that a substantial number of peer competitors only provide the outcomes of their best or median trials without including comprehensive statistical information, such as mean values and variances for their experiments, which hampers the feasibility of conducting statistical significance tests (SSTs) between the proposed methods in this thesis and the competitors.

Table 3.1: Performance comparisons on the **CIFAR-10** dataset.

Model	Error Rate	#Parameters	GPU-Days
FractalNet [70]	5.22%	38.6M	—
Maxout [43]	9.3%	—	—
ResNet-101 [48]	6.43%	**1.7M**	—
DenseNet (k=24) [56]	3.74%	27.2M	—
Highway Network [129]	7.72%	—	—
VGG [126]	6.66%	20.04M	—
CGP-CNN [132]	5.98%	**2.64M**	27
NAS [182]	6.01%	**2.5M**	22,400
Large-scale Evolution [106]	5.4%	5.4M	2,750
Block-QNN-S [174]	4.38%	6.1M	90
MetaQNN [13]	6.92%	—	100
EIGEN [107]	5.4%	**2.6M**	**2**
CNN-GA [140]	4.78%	**2.9M**	35
PNASNet-5 [73]	3.41%	3.2M	150
AmoebaNet-B [105]	2.98%	34.9M	3,150
EAS [18]	4.23%	23.4M	$<$10
NASNET-A [183]	2.97%	27.6M	2,000
AECNN [138]	4.3%	**2.0M**	27
DENSER [11]	5.87%	10.81M	—
GeNet from WRN [155]	5.39%	—	100
CoDeepNEAT [91]	7.3%	—	—
Hier. repr-n, evolution [74]	3.63%	—	300
EffPnet [150]	3.58%	**2.68M**	$<$3
DARTS [75]	2.82%	3.4M	**1**
NSGA-Net [84]	2.75%	3.3 M	4
LEMONADE [35]	**2.58%**	13.1M	90
NSGANetV1-A1 [85]	3.49%	**0.5M**	27
Proxyless NAS [19]	**2.08%**	5.7M	1,500
EAEPSO (Best)	2.51%	3.6M	3
EAEPSO (Median)	2.74%	2.94M	2.2
EAEPSO (Average)	2.75%	3.17M	2.8

3.4.1.1 Performance on CIFAR-10

Table 3.1 lists the number of parameters, the predictive error rate, and the GPU-days of the searching process of both EAEPSO and the other baseline methods on the CIFAR-10 dataset. EAEPSO (Median) achieves an error rate of 2.74% on CIFAR-10, and two peer competitors outperform it, whose error rates are in bold font in Table 3.1. Specifically, for LEMONADE [35] and Proxyless NAS [19], the model scales are 4.46 times and 1.94 times of EAEPSO (Median), and the computational costs are 41 and 682 times of that of EAEPSO (Median). The best error rate is only 2.51%, and the average error rate is similar to the median value. As EAEPSO is based on the DenseNet structure, we further implement a One-Sample T-Test between EAEPSO and DenseNet, and the P-value is 0.003, showing the proposed algorithm is statistically significantly better than the original DenseNet. In terms of the model scale, EAEPSO (Median)'s 2.94M ranks ninth, and eight competitors are with a slightly smaller number of parameters. Nevertheless, their predictive error rates are all worse than EAEPSO (Median)'s. The average model scale is 3.17M, which is a little larger than EAEPSO (Median). Concerning the computational cost of the searching process, EAEPSO (Median)'s 2.2 GPU-days ranks third over the NAS methods, and EIGEN's [107] 2 GPU-days and DARTS's [75] 1 GPU-day outperform it. However, EIGEN's predictive accuracy is 1.78% worse than EAEPSO (Median), and DARTS's model scale is 0.46M bigger than EAEPSO (Median) along with a worse predictive accuracy. The average computational cost is 2.8 GPU-days, which is still significantly smaller than most of the NAS methods, as many of them spend hundreds or even thousands of times of EAEPSO's GPU-days. Overall, EAEPSO demonstrates it is a very competitive method by comparing it with the 28 peer competitors on the CIFAR-10 dataset.

[2]Strictly speaking, GPU-days are not accurate since GPUs can be different, but GPU-days can be used as a good indicator, like many other papers [105, 140, 150, 155].

3.4.1.2 Performance on CIFAR-100

Table 3.2: Performance comparisons on the **CIFAR-100** dataset.

Model	Error Rate	#Parameters	GPU-Days
FractalNet [70]	22.3%	38.6M	—
Maxout [43]	38.6%	—	—
ResNet-101 [48]	25.16%	**1.7M**	—
DenseNet (k=40) [56]	17.2%	25.6M	—
Highway Network [129]	32.39%	—	—
VGG [126]	28.05	20.04M	—
SENet [54]	**15.41**	34.4M	—
Large-scale Evolution [106]	23%	40.4M	>2730
Block-QNN-S [174]	20.65%	6.1M	90
MetaQNN [13]	27.14%	—	100
EIGEN [107]	21.9%	11.8M	5
CNN-GA [140]	20.03%	**4.1M**	40
NSGA-Net [84]	20.74%	**3.3M**	8
PNASNet-5 [73]	19.53%	**3.2M**	150
ENAS [100]	19.43%	**4.6M**	**0.5**
AmoebaNet-A [105]	18.93%	**3.1M**	3,150
DARTS [75]	17.54%	**3.4M**	**1**
NSGANetV1-A1 [85]	19.23%	**0.7M**	27
AE-CNN+E2EPP [136]	22.02%	20.9M	10
EffPnet [150]	18.70%	—	—
EAEPSO	16.17%	5.42M	4
EAEPSO(Transfer)	16.94%	4.32M	2.2

Table 3.2 exhibits the performance of EAEPSO, EAEPSO(Transfer), and some peer competitors on the CIFAR-100 dataset. EAEPSO's error rate ranks second among eleven peer competitors, and only the error rate of manually-designed SENet [54] is 0.76% lower than it. Nevertheless, the number of parameters of SENet is 6.35 times of EAEPSO. In regard to the

number of parameters, eight peer competitors have more minor model scales than EAEPSO, but they are also with higher error rates. As for the computational cost, EAEPSO's 4 GPU-days ranks third among the 13 NAS methods. ENAS [100] and DARTS [75] have less searching time, however, they also have higher error rates. So we can say EAEPSO achieves very competitive performance on CIFAR-100 considering the model scale, predictive error rate, and computational cost.

3.4.1.3 Transferability Performance

In order to justify the transferability of the searched block architecture, we transfer the block searched on CIFAR-10 to other datasets — CIFAR-100 and ImageNet. The blocks sharing the same CIFAR-10 architecture are stacked together to explore the appropriate number of blocks on the new dataset. The experimental results on CIFAR-100 are shown in Table 3.2, denoted as 'EAEPSO (Transfer)'. The error rate of EAEPSO (Transfer) is 16.94%, which is only 0.77% higher than directly searching the block architecture on CIFAR-100, but the model size is only 4.32M, which is smaller than EAEPSO on CIFAR-100. EAEPSO (Transfer)'s GPU-days is only 2.2 GPU-days, which is the second lowest among all the competitors. Please note that the GPU-days of EAEPSO (Transfer) contains two parts: the computational cost of searching for the promising block architecture on CIFAR-10 and the time-consuming of stacking the block to determine the promising number of blocks on CIFAR-100.

Table 3.3 presents the results on ImageNet. Specifically, in terms of the error rate, three peer competitors outperform EAEPSO(Transfer), but their numbers of parameters are all a little larger than EAEPSO(Transfer). Their computational cost is much larger than EAEPSO(Transfer).

The experimental results indicate the block architecture evolved by EAEPSO has a very good transferability, which can lead to a satisfactory predictive accuracy on the target domain. Besides, we can take advantage of the transferability to improve the search efficiency when encountering

Table 3.3: Performance comparisons on the **ImageNet** dataset.

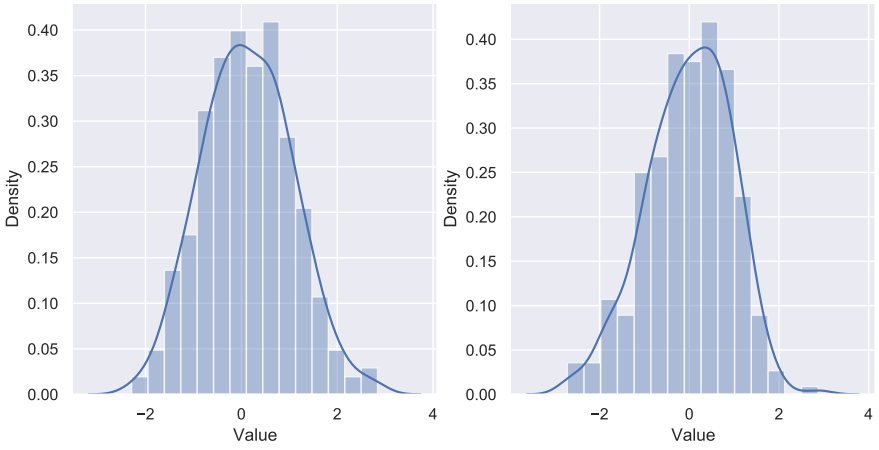
Model	Error Rate	#Parameters	GPU-Days
NAGANetV1-A1 [85]	29.1%	**3.0M**	27
MobileNet-V2 [115]	28.0%	**3.4M**	—
NSGANet-A [183]	**26.0%**	5.3M	1,575
AmoebaNet-A [105]	**25.5%**	5.1M	3,150
WRN-18 [165]	30.4%	11.7M	—
PNASNET-5 [73]	**25.8%**	5.1M	150
EffPnet [150]	27.0%	—	—
EAEPSO(Transfer)	26.9%	4.9M	**4**

a huge dataset by transferring the block architecture searched on a smaller domain dataset to the target large dataset, reducing the computational cost and help people solve larger problems.

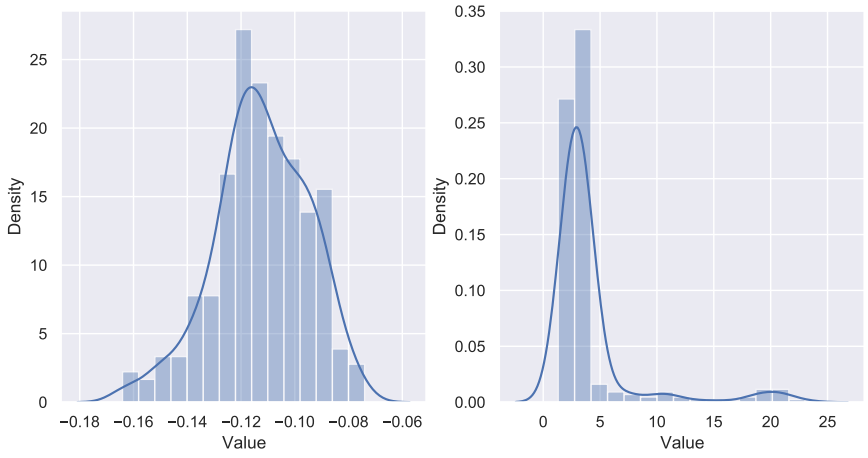
3.4.2 Analysis on Autoencoder

The objective of this section is to investigate the effectiveness of the designed autoencoder, which contains a special normalization layer at the end of the encoder part, and employs pair-wised training samples and the proposed loss function during the training process. To achieve this objective, a comparative autoencoder is built, which has the same architecture as the proposed one except for not including the specific normalization layer. The comparative autoencoder is called CAE, which is also trained with the same training dataset for the same number of training epochs, but the samples do not need to be input pair-wisely, and its loss function only contains the reconstructive error without the architecture similarity loss or the scale similarity loss.

We compare the latent space produced by the autoencoder in EAEPSO and by CAE. Specifically, we sample some random dense blocks and use the two autoencoders to represent them into 8-element latent vectors. We



(a)



(b)

Figure 3.4: The probability density curve of two element values of latent vectors. (a) Produced by the proposed autoencoder. (b) Produced by CAE.

visualize two randomly selected element values' probability density in Figure 3.4. As shown in the figure, for the autoencoder used in EAEPSO, the distribution of the values in latent blocks is very smooth and follows

the Gaussian distribution, which may facilitate the downstream search [159]. On the contrary, the latent space is not so smooth for CAE. The first probability density is relatively smooth, and most values also follow almost a Gaussian distribution. However, the second one is not smooth at all. Most values are concentrated around three and four and accompanied by a long tail distribution, which may be hard to make an effective and efficient search.

Because similar candidates are more likely to have comparable performance, we hope similar candidates could also be close in the latent space. We plot the relationship between the L_1 distance in the latent space and that in the normalized block vectors in Figure 3.5. Similar dense block architectures tend to have a small latent block distance in EAEPSO. On the other hand, the relationship between the block vector distance and the latent vector distance is obviously non-linear for CAE, which means similar candidates may have a large latent vector distance, and this may harm to the downstream search process.

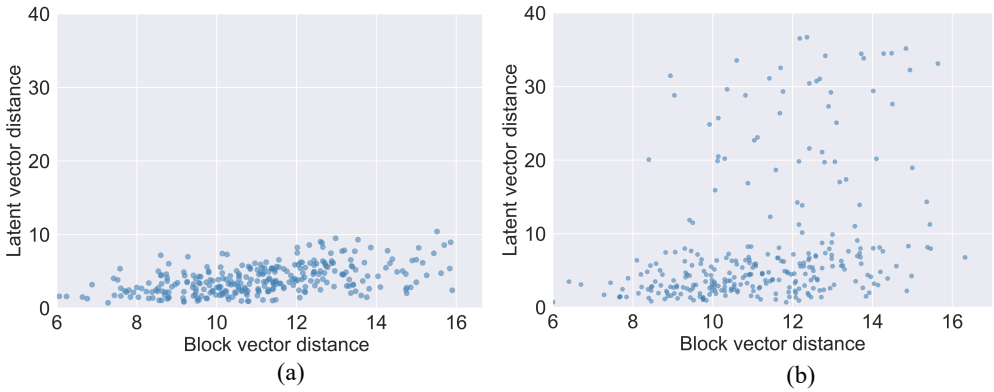


Figure 3.5: The Relationship between the L_1 distance of block vectors and that of corresponding latent vectors. (a) Produced by the proposed autoencoder, which considers the architecture similarity loss. (b) Produced by CAE.

We use the sum of all the values in the block vector to approximate the model scale and compare the relationships between the sum of values in block vectors and that in corresponding latent vectors generated by the two autoencoders, which is exhibited in Fig 3.6. With the help of the autoencoder in EAEPSO, the dense blocks of a similar scale also have a similar sum of values in the latent vectors, and the linear relationship is apparent. In this way, the candidates with the similar model scales are tend to be in the same region in the latent space, which may be helpful to the search process. On the contrary, the sum of values in latent vectors cannot reflect the model scale in CAE.

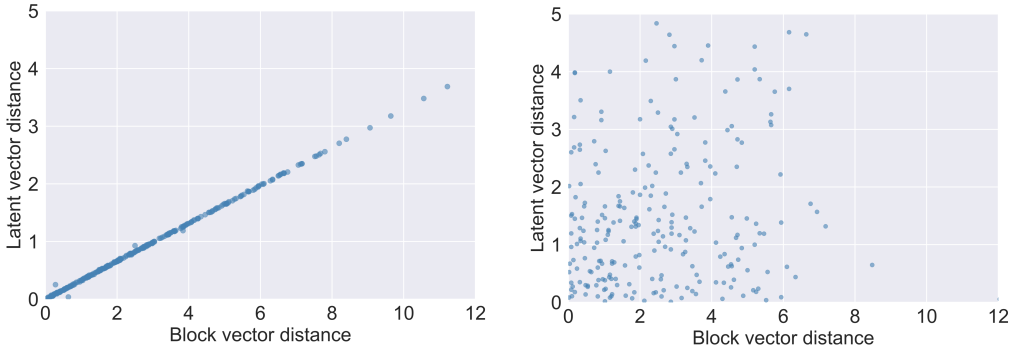


Figure 3.6: The relationship between the L_1 distance of the sum of values in block vectors and that in corresponding latent vectors. (a) Produced by the proposed autoencoder. (b) Produced by CAE.

These comparative experiments show that the proposed autoencoder could transform the original dense blocks' block vectors to a smooth latent space, reflecting the architecture similarity and the model scale similarity among different dense blocks. In this way, the candidates with similar architecture and scale are more likely to concentrate together in the latent space.

3.4.3 Analysis on different training data scales

To prove the effectiveness and efficiency of the proposed hierarchical fitness evaluation method, we use different fitness evaluation methods to replace the proposed one in the PSO process on CIFAR-10. The performance of the population in the last iteration is estimated again using the same assessment method, and the final results are presented in Table 3.4, where the proposed EAEPSO method employs both the basic fitness evaluation method and progressive fitness evaluation method, which use a variety of training data portions: 10%, 20%, and 40%.

Table 3.4: The comparisons of different evaluation methods.

Method	Average Acc.	Highest Acc.	GPU-Days
10% training data	71.62%	72.56%	0.3
40% training data	73.57%	74.68%	4
100% training data	73.92%	74.89%	14
EAEPSO	73.50%	74.44%	2

The method using only 10% training data consumes only 0.3 GPU-days, which is much smaller than the other three methods, along with the lowest average accuracy and highest accuracy. On the other hand, the method using all of the training data achieves both the best average accuracy and the best highest accuracy, but the computational cost is also much more expensive than others. EAEPSO's average accuracy is slightly lower than directly using 40% training data, and the highest accuracy is 0.24% lower. However, EAEPSO's 2 GPU-days is only half of the comparative method's 4 GPU-days. So we can say the proposed fitness evaluation method achieves a good balance between effectiveness and efficiency.

We also investigate the impact on the rank consistency between the evaluated fitness and the true performance when reducing the training dataset. We employ a standard PSO algorithm to evolve 30 particles for 10 iterations, and 80% training dataset is selected as the SGD training part to

train the candidates, and the other 20% is used for evaluating the candidates' fitness.

We record the candidates' architectures at each iteration and employ 10%, 20%, and 40% of the training dataset to re-evaluate the networks for each generation. The rank relationships of all the candidates' performance with the reduced training datasets are compared with each other. We employ the Spearman Coefficient as the metric to evaluate the rank consistency of different training data scales. The Spearman Coefficient between two ranking strings r_p and r_q is calculated by:

$$\rho_s(p,q) = 1 - \frac{\sum_{i=1}^{n}(r_{p,i} - r_{q,i})^2}{n \times (n^2 - 1)} , \tag{3.10}$$

where n is the number of the candidates in each iteration, and $r_{p,i}$ and $r_{q,i}$ refer to the i-th candidate's rank over the 30 candidates under two different conditions, respectively.

We calculate the Spearman Coefficient using different training data scales for each iteration. Specifically, we compare the ranking results using 40% training data with 10% and 20% training data, respectively. Figure 3.7 shows the comparison results. Along with the evolution, the Spearman Coefficients of both training scales (10% and 20%) show a downward trend. The reason is probably that the candidates learn from each other and gather in the neighborhood space, making their fitness tend to be similar, which inspires us to use more training data along with the evolution process. Another observation is that the Coefficient of 20% training data is almost always better than that of 10% of the training data at the same iterations, indicating that using more training data can bring more reliable fitness evaluation results.

One of the primary purposes of the fitness evaluation is to identify the best particle at each iteration to guide the downstream velocity and position update. If we can use a reduced training dataset to help identify each iteration's best particle, the searching time will be reduced. We investigate this by analyzing the performance of the best particle found by a bigger

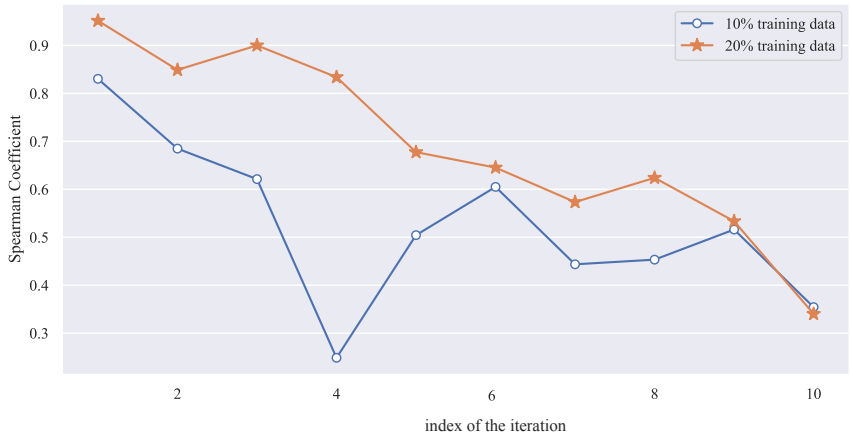


Figure 3.7: The influence of different data scales.

training dataset when trained by a smaller dataset. Figure 3.8 shows that eight particles over ten top rank third when trained by a smaller dataset, indicating that the best particle found by a larger dataset usually also performs well when trained by a smaller dataset. This inspires us to use more minor training data to evaluate the candidates and then use more training data to further evaluate the well-performing candidates to identify the best particle precisely.

3.4.4 The Rational Exploration of Stacking the Blocks

In EAEPSO, PSO searches for a good dense block structure, and then different blocks that share the same structure are stacked together to determine the best number of blocks. In this section, we design experiments to show if the whole network's performance is closely related to the performance of one block. Specifically, five different dense block structures are randomly generated first. For each one, at most four blocks are stacked together to test the performance on the CIFAR-10 dataset considering the size of the input. Figure 3.9 shows the prediction error rate with differ-

Figure 3.8: The best candidate at each iteration is found by training on 40% training data, and their rank at each iteration when trained by 20% training data is shown.

ent numbers of blocks. Table 3.5 presents the single block's performance, the network's best performance, and the number of blocks of the best networks.

From Figure 3.9, we can see the prediction error rate decreases along with the number of blocks increasing when the number is smaller than four. From Table 3.5, we can see that *Structure 5* has the lowest error rate when the network is composed of a single block, and it also achieves the best performance when consisting of 3 blocks. Generally, the network's best performance is consistent with the single block's performance. It is reasonable to select the best-performed block structure and then stack the blocks.

3.5 Chapter Summary

This chapter aims to propose an effective and efficient ENAS method for image classification. The goal has been achieved by designing an autoen-

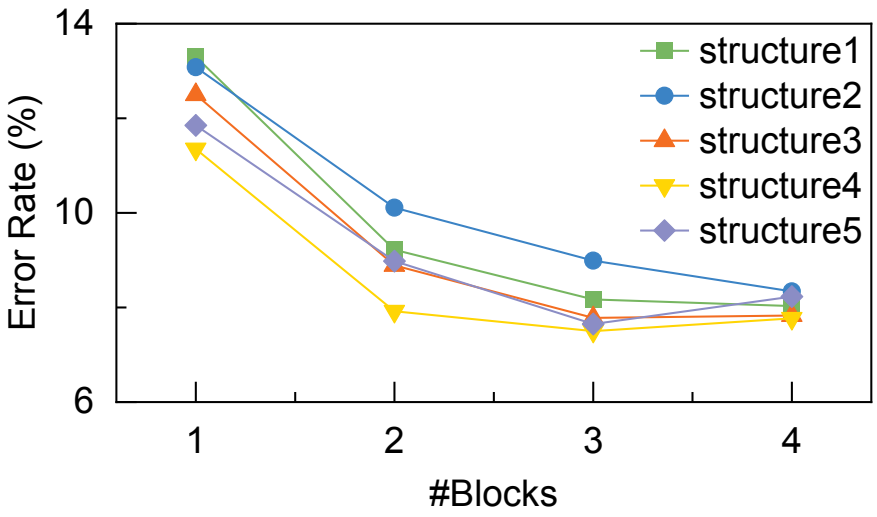


Figure 3.9: The prediction error rates of networks stacked by different numbers of blocks, and there are five different block structures.

Table 3.5: Different single blocks' prediction error rates, best networks' prediction error rates, and the corresponding numbers of blocks.

Structure	Single Block Error Rate	Lowest Network Error Rate	Corresponding Number of Blocks
1	13.30%	8.03%	4
2	13.08%	8.34%	4
3	12.50%	7.78%	3
4	11.85%	7.65%	3
5	11.35%	7.50%	3

coder and developing an efficient hierarchical fitness evaluation strategy. The proposed method is based on PSO and is named EAEPSO. Specifically, in order to represent the blocks of different depths with fixed-length individuals to facilitate the search, this chapter designs an autoencoder, which is able to encode the block vectors to fixed-length latent vectors. In addition, the proposed autoencoder can convert the discrete search space to a continuous smooth latent space, and the designed loss function can make similar candidates distribute close to facilitate the downstream search process. By investigating the reliability of using different training data scales, it is found that approximate ranking can also guide the search process, which inspired the proposed efficient hierarchical fitness evaluation method that could automatically change the scale of the training data considering the trade-off between effectiveness and efficiency.

The proposed EAEPSO method was tested on three popular benchmark datasets — CIFAR-10, CIFAR-100, and ImageNet. EAEPSO shows very good performance in terms of the searched network's performance and the computational cost, outperforming most peer competitors. Besides, it only costs less than 3 GPU-days on CIFAR-10, much smaller than most peer competitors, costing hundreds or even thousands of GPU-days. To test the transferability, the block architecture searched by CIFAR-10 was transferred to CIFAR-100 and ImageNet, which achieved competitive performance, indicating its good transferability. In addition, further experiments prove the effectiveness of the proposed autoencoder and the proposed hierarchical fitness evaluation method.

Although the proposed method achieved promising performance, it mainly searched the architecture of dense blocks. The following chapter will explore the search based on lightweight backbone architectures. Besides, this method employs a standard EC method, and the following chapter will investigate how to accelerate the convergence of the evolutionary process.

Chapter 4

NAS based on Performance Prediction and Weight Inheritance

4.1 Introduction

While numerous existing ENAS techniques, along with the methodology discussed in the previous chapter, have made strides towards minimizing the computational overhead of NAS, there remains considerable potential to further reduce the computational resources required. In contrast to other strategies, this chapter endeavors to reduce the computational overhead by accelerating the evolution.

The efficiency of traditional EAs might not be satisfactory [114]. In the evolutionary process of ENAS, parent candidates are usually selected based on their fitness values, and new offspring are generated by performing genetic operations on the selected parent candidates. In this way, the offspring are expected to achieve better performance than the parents [158], promoting the evolution. If the ratio of better offspring could be enhanced, the process of evolution would be accelerated, reducing the total number of generations and alleviating the prohibitive computational burden. This chapter proposes a performance predictor to help indicate well-performed offspring, improving the evolutionary efficiency and sav-

ing computational cost. This method overcomes the shortcomings of most existing performance predictors, as even estimated (possibly inaccurate) predictions will not harm the evolution because the offspring will be further accurately evaluated. Moreover, given the genetic overlap between offspring and their parents, this chapter introduces an enhanced weight inheritance mechanism to expedite offspring evaluation.

Some novel, efficient, and portable network architectures have gained a lot of attention in computer vision. Many of them can reduce the number of parameters and improve computing efficiency while maintaining or even enhancing performance. Notably, MobileNetV3 [52] incorporates Squeeze-and-Excitation (SE) modules complemented by the efficient h-swish activation functions. Considering MobileNetV3's superior performance and lightweight architecture, more efficient convolutional modules can be investigated based on the MobileNetV3 block to extract discriminative features. Furthermore, utilizing an appropriate search method to explore the essential hyper-parameters of the module may further improve the performance.

4.1.1 Chapter Goals

The overall goal of this chapter is to propose a new ENAS algorithm by proposing a lightweight architecture, incorporating a performance predictor to help generate well-performed offspring, and employing a weight-inheritance method to accelerate fitness evaluations. Four specific objectives are detailed below to support the overall goal:

1. Propose a new lightweight backbone block architecture that can effectively construct discriminative features from the input feature maps. It is expected to have a small model scale and high computational efficiency. The search space will be built based on the proposed block architecture, so that the searched network will be efficient and portable.

2. Invent a random forest-based performance predictor. Instead of directly predicting the fitness, it aims to predict whether offspring outperform their parent(s) and the confidence level of the prediction. In this way, only the offspring most likely to perform well will be preserved, which can improve the overall performance of the offspring population, thereby accelerating the convergence of the evolutionary process. Moreover, even inaccurate predicted results will hopefully not harm the evolution.

3. Propose new crossover and mutation operators to cope with the architecture representations. The newly generated offspring are expected to remain the same in most layers as one of its parents. This approach makes it easier for the performance predictor to identify whether the offspring can outperform their parents.

4. Investigate the influence of different weight inheritance methods on convergence efficiency and propose a new weight inheritance method to avoid random initialization of the newly generated layers of the offspring. Some layers of offspring are generated by mutation and differ from the parent individuals. Avoiding these layers' random initialization is expected to further improve the training efficiency.

4.1.2 Chapter Organization

The remainder of this chapter is organized as follows. Section 4.2 introduces the details of the proposed algorithm. Immediately after, Section 4.3 describes the experiment design information, and Section 4.4 presents the experiment results and corresponding analysis. At last, Section 4.5 concludes this chapter.

4.2 The Proposed Method

4.2.1 Algorithm Overview

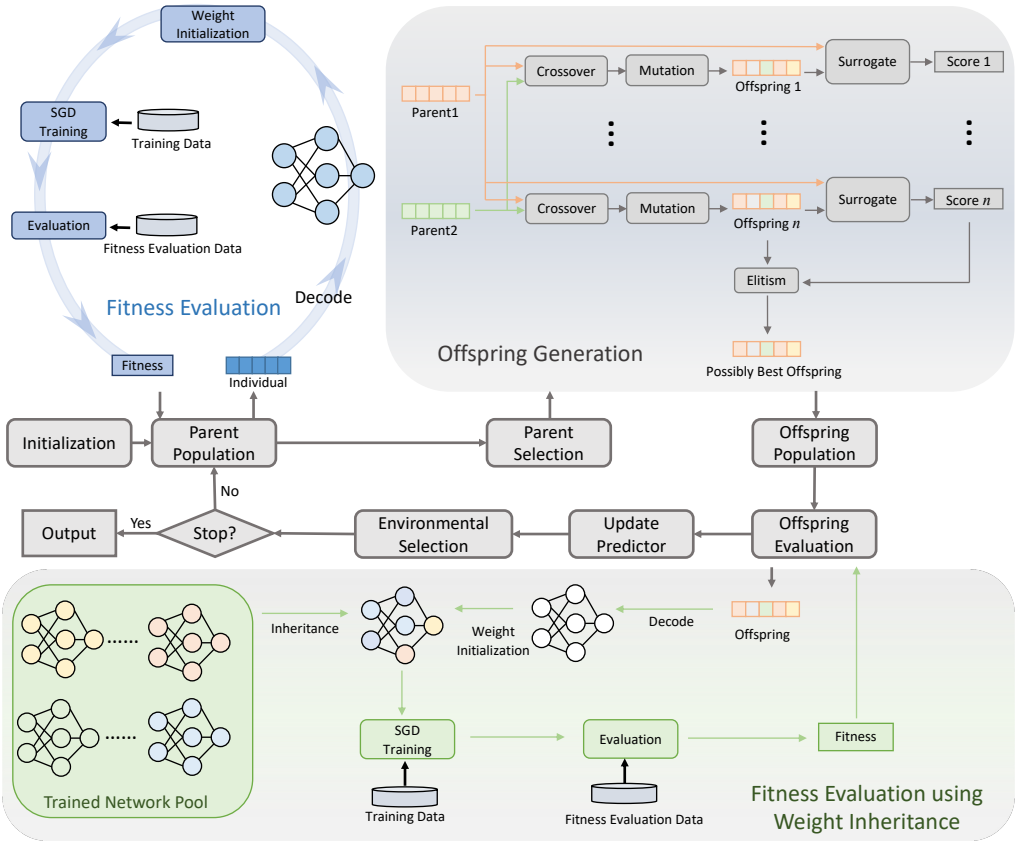


Figure 4.1: Overview: EPPGA searches for the network architecture following an evolutionary procedure.

The proposed efficient performance predictor-based genetic algorithm for NAS, called EPPGA, is illustrated in Figure 4.1. It generally follows a standard evolutionary algorithm pipeline. Prior to the evolution, a search space and an encoding strategy are predefined (see Section 4.2.2). Then, a GA is employed to search the network architecture until the termination criterion is met, at which point the best individual is selected, and its

corresponding architecture is output.

For the evolutionary process, a population of candidate network architectures is initialized first (see Section 4.2.3), and a common fitness evaluation method is applied to measure the fitness of the individuals. The top left circle of Figure 4.1 illustrates the details: an individual is decoded to the network and then initialized by a traditional weight initialization method. Subsequently, an SGD method is used to train the weights, and the network is evaluated on the fitness evaluation dataset to obtain its fitness. The process is repeated until all the individuals are evaluated. Then, the proposed offspring generation method is employed to produce offspring individuals (see Section 4.2.4), as shown in the upper middle part of Figure 4.1. Specifically, new crossover and mutation operations are performed on the two selected parents to generate new offspring a number of times. Each offspring and its corresponding parent are input to the performance predictor to obtain its score, i.e., the probability of performing better than the parent. The details of the proposed performance predictor are documented in Section 4.2.5. The offspring with the highest score is selected as the offspring of the two parents. This process will repeat a number of times to form the offspring population. Afterward, the proposed fitness evaluation method using weight inheritance is applied to evaluate the offspring (see Section 4.2.6). Figure 4.1's upper right part presents the details: one offspring is decoded to the corresponding architecture, and its initialized weights are inherited from a trained network pool, which includes all the trained networks, and the offspring's network inherits weights from its parents and other networks that share some identical block architectures. Then, the routine training and evaluation procedures are applied. Note that the weight inheritance fitness evaluation will be applied to all offspring individuals, respectively. Then, the performance predictor will be constructed/updated. Next, the environmental selection is used to select individuals to build the next generation from both the current parent population and offspring population.

4.2.2 Architecture Search Space

To investigate the appropriate network architecture, we predefine an architecture search space, and EPPGA searches for the network's depth (number of layers), width (number of feature maps), and convolution kernel size. These factors are crucial for a network and can significantly affect its feature extraction ability. This section introduces the candidate network architecture and the corresponding encoding strategy.

4.2.2.1 Network Architecture

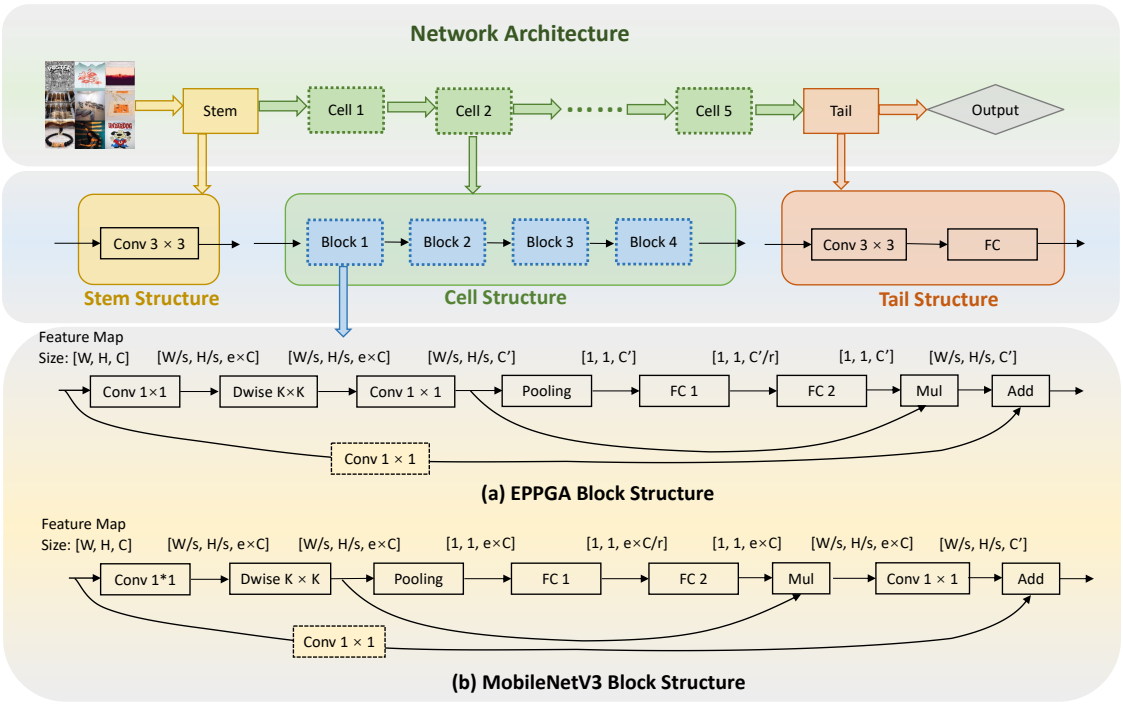


Figure 4.2: An example of the network architecture: the top module shows the overall network architecture. The middle module presents the detailed stem and tail structures. The cell is composed of four blocks in this example. The lowest module provides the details of the EPPGA block structure and the original MobileNetV3 block structure for comparison.

As illustrated in Figure 4.2, the input images are processed by a stem first, then five cells, and finally the tail. The stem and tail structures are fixed. The stem structure comprises a convolutional layer with a 3×3 kernel size (followed by a batch normalization operation and a non-linear activation layer). The tail structure contains a convolutional layer, and an FC layer, classifying the images according to the extracted features.

The network's central part consists of five continuous cells, with non-fixed cell structures and a maximum of four blocks per cell, as illustrated in Figure 4.2. The maximal number of blocks depends on the available computational resources and can be larger if more powerful Graphics Processing Units (GPUs) are accessible, potentially leading to improved performance. The number of blocks in different cells determines the network depth, which may significantly affect the final result. As different datasets may prefer different depths, EPPGA automatically explores the appropriate number of blocks for each cell.

The blocks share the same backbone structure but have different hyper-parameters. Inspired by the MobileNetV3 block, the proposed block structure has a smaller model scale and lower computational complexity, measured by the numbers of parameters and Floating Operation Points (FLOPs). Meanwhile, its feature reconstruction capability is comparable to the MobileNetV3 block, as demonstrated in Section 4.4.4.

Specifically, the input feature maps (Size [W, H, C], where W is the width, H is the height, and C is the number of channels) are first processed by an expansion layer, consisting of a 1×1 convolutional layer with an expansion rate e. The number of output channels is e times larger than the input channels. The expansion rate e is an important parameter that is difficult to determine, so EPPGA automatically searches for the optimal expansion rate for each block. A depth-wise separable convolution follows, comprising a depth-wise convolutional layer with $k \times k$ kernel size and a point-wise 1×1 convolution. The kernel size affects the quality of reconstructed features, and different blocks may prefer different kernel

sizes due to their positions in the network. Thus, each block's kernel size is also part of the search space. Additionally, the stride of the depth-wise convolutional layer influences the output size. If the block is the first in the cell, the stride can be one or two; otherwise, it is set to one. An SE channel-wise attention module is applied to the output feature maps of the depth-wise convolutional layer, comprising a global average pooling layer, two FC layers, and corresponding non-linear activation layers. The first FC layer squeezes the channels to $1/r$ of the original, and the second FC layer expands the channels to r times, achieving the same as the original, where r is the reduction rate of the SE module. Finally, the feature maps multiply the output of the SE module channel-wisely, and a residual connection is constructed with the original block input.

The primary difference between the original MobileNetV3 Block and the proposed block is the position of the SE module, which is moved from connecting to the depth-wise convolutional layer to connecting to the point-wise convolutional layer. This change reduces both the scale and computational cost of the subsequent SE model, as the point-wise convolutional layer typically has a smaller number of kernels. Specifically, the depth-wise convolution has $e \times C$ kernels, and the point-wise convolution has C' kernels, where C and C' are the numbers of feature maps of the block input and output, respectively. Note that C' is usually slightly larger than C for the first block in each cell but is the same as C for other blocks. For an SE module, excluding the biases of linear layers, the numbers of parameters and FLOPs for a fully-connected layer are calculated according to Equation (4.1) and Equation (4.2), where N_{in} and N_{out} refer to the numbers of neurons of the input and output.

$$PARAMs = N_{in} \times N_{out} \tag{4.1}$$

$$FLOPs = 2 \times N_{in} \times N_{out} \tag{4.2}$$

Consequently, the two FC layers in the proposed block architecture's

SE module share the same number of parameters and FLOPs. For each layer, the number of parameters is $C'^2/\ r$, and the number of FLOPs is $2 \times C'^2/\ r$, where r is the reduction rate of the SE module. In a traditional MobileNetV3 block, one FC layer's number of parameters and FLOPs are $e^2 \times C^2/\ r$ and $2 \times e^2 \times C^2/\ r$, respectively, where e is the expansion rate of the block.

When the input's number of channels is the same as the output, both the numbers of parameters and FLOPs for the SE model of a MobileNetV3 block are e^2 times larger than that of the proposed EPPGA's block. For the first block in each cell, where the input channel number C is smaller than the output channel number C', the number of channels for the feature maps after the expansion layer $e \times C$ is also usually larger than the output channel number C'. Thus, the proposed block architecture reduces the model scale and computational complexity, as quantitatively proven in Section 4.4.4.

4.2.2.2 Encoding Strategy

For the convenience of processing, we represent a network architecture using a string. As we investigate the cell structure within the network, we construct five cell vectors to denote the candidate network. A maximum number of blocks in a cell, b_{max}, must be predefined based on available resources, with b_{max} block vectors in one cell vector. We need to explore the hyper-parameters of a block, so a block vector consists of two numbers: the expansion rate and the kernel size. When the number of blocks in a cell is less than b_{max}, we use a vector of two zeros to represent non-existing blocks, corresponding to the expansion rate and kernel size parameters. Consequently, the cell vectors have a fixed length, which is convenient for further processing. For instance, [[4, 3], [2, 5], [0, 0]] is a cell vector representing a two-block cell with a maximum of three blocks, where the first block's expansion rate and kernel size are 4 and 3, respectively. Likewise, the second block's hyper-parameters are 2 and 5. In this manner, five cell

block vectors form a network string containing $5 \times 2 \times b_{max}$ elements.

4.2.3 Population Initialization

To initialize the population for the initial generation, individuals are generated one by one until reaching the predefined population size. For each individual, the parameters of each cell are generated repeatedly. Specifically, the number of blocks in each cell n_{block} is determined randomly. Then, the block's expansion rate and kernel size are selected from the candidate values. If the number of blocks is smaller than the predefined maximum number b_{max}, 0s are added to make the cell vector reach the length of $2 \times b_{max}$. The individual is represented by connecting the generated cell vectors. Finally, all the developed individuals make up the population of the first generation.

4.2.4 Offspring Generation

To generate an offspring, two parent individuals are chosen from the current population using binary tournament selection. We then randomly select one parent individual as the primary parent, while the other is named the secondary parent individual.

In EPPGA, both crossover and mutation operations are performed on the two parents for a predefined number of times individually, generating one offspring each time. The performance predictor receives information about each offspring and its primary parent. A score evaluating the likelihood of the offspring outperforming its primary parent is predicted by the performance predictor. The offspring with the highest score is added to the offspring population, and others become extinct. This approach ensures that the generated offspring is more likely to have better performance than its parents, adhering to the original purpose of the genetic operations. This idea is similar to "brood recombination" [143], with the key difference being the use of the performance predictor, which is more effi-

cient and time-saving. Please note that in the first generation, crossover and mutation are applied only once to each pair of parents, and the offspring is directly added to the offspring population without estimating its score since there is no trained performance predictor.

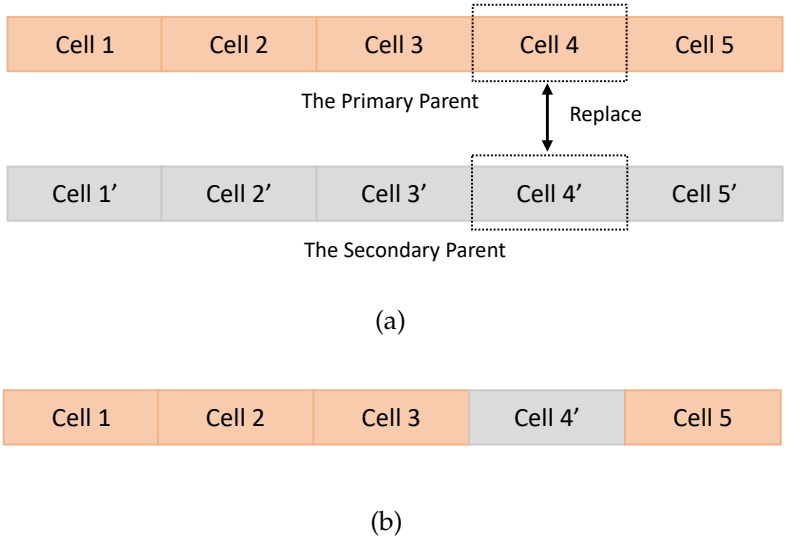


Figure 4.3: An example of a crossover operation: (a) The two parent individuals. (b) The generated offspring. The primary parent replaces the 4th cell with the secondary parent, generating the offspring.

Figure 4.3 shows an example of the crossover operation. Specifically, one of the five cells is randomly chosen, and then the corresponding cell structure of the primary parent is replaced by the corresponding cell of the secondary parent, producing a new offspring.

After generating an offspring using a crossover operation, the mutation operation is utilized to alter some blocks' hyper-parameters of the offspring, as detailed in Algorithm 5. Specifically, the number of one-block mutations n_m are generated within the maximal times of one-block mutation n_{max} (line 1). For each one-block mutation, a block is randomly selected from the entire offspring architecture (line 3), the number of blocks in the corresponding cell is calculated (line 4), and a specific mutation type

Algorithm 5: The Mutation Operation

Input: The offspring q; the maximal number of blocks in a cell b_{max}.
Output: The mutated offspring q.

```
1  n_m ← generate a number between [1, n_max];
2  for i = 1; i ≤ n_m; i ← i + 1 do
3      block ≤ Randomly choose a block from q;
4      n_b ← the number of blocks in the cell;
5      operation ← Randomly select one from
         {adding, removing, modifying};
6      if operation is adding and n_b < n_max then
7          block_new ← generate a new block;
8          Add block_new next to block;
9      else if operation is removing and n_b > 1 then
10         Remove block;
11     else
12         block_new ← generate a new block;
13         Replace block with block_new;
14     end
15 end
```

is randomly selected from three predefined types: adding, removing, and modifying (line 5). If the type is *adding* and the number of existing blocks is smaller than the maximum value (line 6), a new block vector is generated and is positioned next to the selected block (lines 7-8). If the type is *removing* and there is more than one block in the cell (line 9), the selected block is removed from the architecture (line 10). If the type is *modifying*, a new block is generated and replaces the selected one (lines 12-13).

By controlling the maximal times of one-block mutation, most block architectures remain the same between the generated offspring and the primary parent. This similarity offers several advantages. First, this can enhance the performance predictor's effectiveness. The primary parent and

the offspring will be used to construct the training samples for the performance predictor. The similarity between them can strengthen the correlation of the two parts within a training sample. The performance may focus more on the different components rather than all the information, potentially improving its performance, especially when there are limited training samples. Second, this similarity can reduce the computational cost of evaluating the offspring. Since most blocks of the offspring inherit the same architectures from the primary parent, the weights of these blocks can also be inherited from the parent, expediting the training process.

4.2.5 Performance Predictor

A performance predictor is constructed to assess the probability of offspring surpassing their primary parents in terms of the performance. This mechanism facilitates the pre-selection of superior offspring, thereby enhancing the evolutionary efficiency. Following this purpose, a pair of parents generate several offspring, and only the one with the highest score evaluated by the performance predictor will survive. Moreover, incorrect predictions are unlikely to misguide the evolution since each generated offspring (if not discarded) undergoes precise evaluation to obtain their fitness. This method offers a distinct advantage over algorithms that solely rely on a performance predictor to evaluate offspring's fitness.

To train the performance predictor, appropriate training data is required. Given the predictor's purpose, both the primary parent and candidate offspring information are collected as features, with a class label of 1 indicating that the offspring outperforms the parent individual, and 0 otherwise.

Figure 4.4 depicts the two feature components of a training sample, with the full feature set being a combination of both parts. Figure 4.4(a) presents the primary parent information, including individual representation, the number of parameters, and the number of FLOPs. The individual representation is an encoded string containing 40 elements in the example,

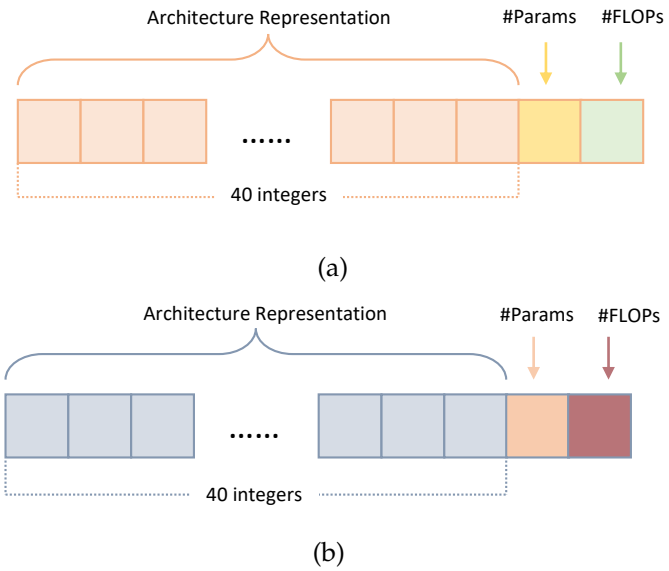


Figure 4.4: An example of the feature construction for the performance predictor contains two parts: (a) The first part comprises the primary parent's information. (b) The second part is composed of the generated offspring's information. The architecture representation length is 40 in this example.

assuming a maximum of 4 blocks per cell. Both the numbers of parameters and FLOPs are essential network features, indicating the network's scale and computational complexity, which may be related to the final predictive accuracy. Similarly, the second part contains offspring information. For each generation, after evaluating the performance of the offspring, the training dataset of the performance predictor is updated by adding the examples composed of the offspring and their primary parent information and the class labels. Then, the performance predictor is retrained on the newly expanded training dataset to work well on the newly generated offspring with a new distribution.

A random forest for binary classification is employed as the performance predictor in EPPGA for several reasons. Firstly, as an ensemble

learning approach, a random forest typically yields good performance. Secondly, it effectively handles high-dimensional features, with 84 features in an EPPGA training example, as shown in Figure 4.4. Thirdly, the computational cost of a random forest is relatively low, especially compared with neural networks, enabling faster training and prediction. The primary goal of utilizing a performance predictor is to improve evolutionary efficiency and reduce optimization time. If training and using the performance predictor is time-consuming, its design significance is substantially diminished.

The random forest comprises 200 decision trees, ensuring predictive accuracy without excessive computational resource usage. The random forest training process follows standard rules. When predicting whether an offspring can outperform its primary parent, the 200 decision trees make individual decisions, and the proportion of trees voting 1 serves as the predicted score. Consequently, the candidate offspring with the highest score is most likely to achieve good performance. If a candidate offspring's score is below 0.5, its performance may be inferior to its primary parent. For a parent pair, if all candidate offspring scores are below 0.5, the offspring will become extinct, and their primary parent will propagate to the offspring population.

4.2.6 Fitness Evaluation with Weight Inheritance

Fitness evaluation is often considered the most time-consuming process for NAS algorithms. EPPGA employs a novel fitness evaluation method using weight inheritance to expedite offspring evaluations. Note that the original fitness evaluation method without weight inheritance is applied to the initial population, as no pre-existing trained networks are available.

In EPPGA, a weight pool is established to store trained network modules, encompassing both network architectures and their trained weights. The weight pool aids in offspring evaluations. When evaluating an off-

spring, its primary and secondary parents have already been trained. According to the offspring generation method described in Section 4.2.4, four cells are similar to the primary parent's corresponding cells, and one cell is partially similar to the secondary parent, excluding mutated blocks. Thus, the non-mutated blocks' weights are directly inherited from the offspring's parents as initialized weights.

As for the mutated blocks, the weights will be inherited from the blocks with identical structures in the weight pool, rather than random initialization. If no matching block structures are present in the pool, the block's weights are initialized using *He Initialization* [47]. Consequently, most offspring blocks inherit weights from trained networks, significantly accelerating training convergence. Although directly connecting blocks inherited from different networks may cause discrepancies at junctions due to varying feature maps, the weights still converge faster during training than random initialization, as quantified in Section 4.4.3. This is probably because feature maps at similar positions in networks, despite being distinct, possess some common intrinsic features.

4.3 Experimental Settings

4.3.1 Benchmark Datasets

We employ three well-known image classification datasets, namely CIFAR-10, CIFAR-100, and ImageNet, to verify the effectiveness and efficiency of EPPGA. The reason for choosing them is that they are famous and are used in the NAS literature, bringing convenience to compare with peer competitors. Detailed information about these datasets can be found in Section 3.3.1 (see page 84).

4.3.2 Peer Competitors

To demonstrate the effectiveness and efficiency of EPPGA, we compare its performance with several state-of-the-art algorithms, which can be broadly categorized into hand-crafted and NAS competitors. For hand-crafted competitors, we consider the test error rate, model size (measured by the number of parameters), and computational complexity (measured by the number of FLOPs). When comparing with NAS algorithms, we also account for the search cost, measured in GPU-days, indicating the search process duration using a single GPU card.

Specifically, the manual-designed competitors include FractalNet [70], Maxout [43], ResNet [48], DenseNet [56], Highway Network [129], VGG [126], MobileNetV2 [115], and ShuffleNet [172]. The NAS algorithms contain CGP-CNN [132], NAS [182], Large-scale Evolution [106], Block-QNN [174], MetaQNN [13], EIGEN [107], CNN-GA [140], PNASNet [73], AmoebaNet [105], EAS [18], AECNN [138], NPENAS [152], Hier. repr-n, evolution [74], EAEPSO [164], EffPnet [150], DARTS [75], NSGA-Net [84], LEMONADE [35], MetaQNN [13], EffPnet [150], EIGEN [107], NSGANetV1 [85], Proxyless NAS [19], AE-CNN+E2EPP [136], SNAS [157], ENAS [100], SI-EvoNet-S [167], NPENAS-NP [152], Modulenet [24], CNN-GA [140], MobileNetV3 [52], SSPS [133], BanditNAS [168], and EBNAS [119].

4.3.3 Parameter Settings

Most experimental parameter settings follow the tradition of GAs and machine learning. Besides, some parameters are also affected by the availability of the computational resources, and all the experiments in this chapter are implemented using NVIDIA A6000.

For the evolutionary process, the population size is set to be 20, which is smaller than most GA methods. The number of generations is also set to be 20. Each pair of parents generates 10 offspring, and the one with the highest score can survive. While encoding the candidate architectures,

considering the hardware ability, the maximal number of blocks in one cell is 4. The maximal number of mutated blocks is 3. As for the performance predictor, 200 decision trees are utilized to build the random forest, and each decision tree selects less than 40 features, and a bootstrapping sampling method is employed.

In terms of the post-search evaluation, the number of training epochs is 300. A batch size of 96 is employed, and a cosine annealing learning rate policy is utilized with the initial learning rate set to 0.01. The dropout rate is set to be 0.2 on CIFAR-10 and CIFAR-100, and the dropout technique is not employed on ImageNet. For the data augmentation method, except for the *cropping* and *flipping* methods, *cutout* is also employed to improve the performance, which is similar to other peer methods.

4.4 Results and Analysis

4.4.1 Overall Results

EPPGA is run on both CIFAR-10 and CIFAR-100 five times with different independent random seeds. We report the average performance of the five trials and the best and median trial results. However, considering the limited available computational resources, we only perform EPPGA on ImageNet once.

4.4.1.1 Performance on CIFAR-10

Table 4.1 presents the experimental results on CIFAR-10. The best trial's error rate is only 2.44%. The median trial of EPPGA achieves an error rate of 2.50% with 2.2M parameters within 1.6 GPU-days. The average performance of the five trials closely resembles the median results. Since the three measurements are intrinsically connected, we primarily analyze the comparison results of the median trial. Considering the error rate,

Table 4.1: The comparisons with peer competitors on **CIFAR-10**.

Model	Error Rate	#Parameters	GPU-Days
FractalNet [70]	5.22%	38.6M	—
Maxout [43]	9.3%	—	—
ResNet-101 [48]	6.43%	1.7M	—
DenseNet (k=24) [56]	3.74%	27.2M	—
Highway Network [129]	7.72%	—	—
VGG [126]	6.66%	20.04M	—
Large-scale Evolution [106]	5.4%	5.4M	2,750
Block-QNN-S [174]	4.38%	6.1M	90
MetaQNN [13]	6.92%	—	100
EIGEN [107]	5.4%	2.6M	2
CNN-GA [140]	4.78%	2.9M	35
PNASNet-5 [73]	3.41%	3.2M	150
AmoebaNet-B [105]	2.98%	34.9M	3,150
AECNN [138]	4.3%	2.0M	27
Hier. repr-n, evolution [74]	3.63%	—	300
EAEPSO [164]	2.75%	3.17M	2.8
EffPnet [150]	3.58%	2.68M	<3
DARTS [75]	2.82%	3.4M	1
NSGA-Net [84]	2.75%	3.3 M	4
LEMONADE [35]	2.58%	13.1M	90
NSGANetV1-A1 [85]	3.49%	**0.5M**	27
Proxyless NAS [19]	**2.08%**	5.7M	1,500
NPENAS-NP [152]	2.54%	3.5M	1.8
SNAS [157]	2.85%	2.8M	1.5
ENAS [100]	2.89%	4.6M	0.5
SI-EvoNet-S [167]	2.69%	1.84M	**0.46**
Modulenet [24]	2.77%	—	8.0
CNN-GA [140]	3.22%	2.9M	35
EPPGA (best)	2.44%	2.39M	1.9
EPPGA (median)	2.50%	2.20M	1.6
EPPGA (average)	2.53%	2.36M	1.8

EPPGA(median)'s 2.50% ranks second among all 28 competitors. Proxyless NAS's [19] 2.08% error rate is superior; however, its model size is 2.59 times larger than EPPGA(median), and its search cost is 938 times greater. EPPGA(median)'s model size is the fifth smallest, and ResNet-101 [48], AECNN [138], NSGANetV1-A1 [85], and SI-EvoNet-S [167] have fewer parameters, but their predictive performance is much worse than EPPGA(median). Regarding the computational cost, four competitors outperform EPPGA(median)'s 1.6 GPU-days, but DARTS [75], SNAS [157], ENAS [100], and SI-EvoNet-S [167] all exhibit poorer predictive performance. From the experimental results, we conclude that EPPGA demonstrates exceptional performance and strikes an outstanding balance between effectiveness and efficiency.

4.4.1.2 Performance on CIFAR-100

Table 4.2 displays the experimental results on the CIFAR-100 dataset. Similarly, the median trial's performance is close to the average performance. Specifically, EPPGA (median) achieves an error rate of 16.75%, ranking third among the NAS competitors. SI-EvoNet-S's [167] 15.70% is lower, but its model size is 1.6 times that of EPPGA (median). EAEPSO [164] is 0.58% more accurate, but its model size is over 2.56 times that of EPPGA (median). In terms of the number of parameters, EPPGA (median)'s 2.12M is the third smallest, with only ResNet-101 [48] and NSGANetV1-A1 [85] having smaller model sizes, but their error rates are also higher. EPPGA (median) only costs 2.4 GPU-days, which is the fourth smallest. ENAS [100] and DARTS [75] have lower computational costs but also exhibit inferior prediction results. Thus, we can conclude that EPPGA achieves excellent performance on CIFAR-100: the small model scale renders it suitable for deployment in various scenarios, and the low computational cost makes it applicable for researchers without expensive computing resources.

Table 4.2: The comparisons with peer competitors on **CIFAR-100**.

Model	Error Rate	#Parameters	GPU-Days
FractalNet [70]	22.3%	38.6M	—
Maxout [43]	38.6%	—	—
ResNet-101 [48]	25.16%	1.7M	—
DenseNet (k=40) [56]	17.2%	25.6M	—
Highway Network [129]	32.39%	—	—
VGG [126]	28.05%	20.04M	—
Large-scale Evolution [106]	23%	40.4M	2,730
Block-QNN-S [174]	20.65%	6.1M	90
MetaQNN [13]	27.14%	—	100
EIGEN [107]	21.9%	11.8M	5
CNN-GA [140]	20.03%	4.1M	40
EAEPSO [164]	16.17%	5.42M	4
NSGA-Net [84]	20.74%	3.3M	8
PNASNet-5 [73]	19.53%	3.2M	150
ENAS [100]	19.43%	4.6M	**0.5**
AmoebaNet-A [105]	18.93%	3.1M	3,150
DARTS [75]	17.54%	3.4M	1
NSGANetV1-A1 [85]	19.23%	**0.7M**	27
AE-CNN+E2EPP [136]	22.02%	20.9M	10
EffPnet [150]	18.70%	—	—
AECNN [138]	22.40%	5.4M	36
SI-EvoNet-S [167]	**15.70%**	3.32M	0.81
Modulenet [24]	17.76%	—	—
CNN-GA [140]	20.53%	4.1M	40
EPPGA (best)	16.34%	2.79M	2.8
EPPGA (median)	16.75%	2.12M	2.4
EPPGA (average)	16.64%	2.43M	2.5

4.4.1.3 Performance on ImageNet

Table 4.3 displays the experimental results on the ImageNet dataset. EPPGA requires 9.4 GPU-days[3] to search for an appropriate network architecture, which is substantially more affordable than most peer competitors. Specifically, the search costs of some competitors are lower than EPPGA's, but they all have reduced accuracies. As for the performance of the search network, EPPGA attains 75.9% top-1 accuracy and 92.7% top-5 accuracy, outperforming most competitors. Among all NAS competitors, only NSGANetV1-A3's [85] top-1 accuracy is 0.3% higher than EPPGA's, along with 0.4% better top-5 accuracy. However, NSGANetV1-A3's number of FLOPs is 74M larger than EPPGA's 496M, and the search cost of NSGANetV1-A3 is almost three times that of EPPGA's. Among the manual-designed networks, DenseNet-169 [56] achieves better classification performance, but its model scale is 2.5 times larger than EPPGA's, and the number of FLOPs is 13.6 times greater. In conclusion, EPPGA exhibits highly competitive performance considering search cost, classification accuracy, model scale, and model complexity.

4.4.2 Effectiveness of Performance Predictor

In EPPGA, a random forest-based performance predictor is developed to help generate well-performed offspring, accelerating the evolutionary convergence. To verify the effectiveness of the performance predictor, we designed a comparative experiment by developing an algorithm without using the performance predictor, termed NPPGA. Specifically, a pair of parents in NPPGA produces only one offspring, while a pair of parents in EPPGA generates 10 offspring, with the performance predictor assisting in selecting the potentially best one.

[3]The cost in GPU-days also includes the training cost of all candidates, differing from some methods that exclude the training cost and report only the search cost.

Table 4.3: The comparisons with peer competitors on **ImageNet**.

Model	Top-1 Acc.	Top-5 Acc.	#Params	#FLOPs	GPU-Days
MobileNetV2 [115]	72.0%	91.0%	**3.4M**	600M	—
ShuffleNet [172]	73.7%	—	5.4M	524M	—
DenseNet-169 [56]	**76.4%**	**93.3%**	14.2M	6,740M	—
ResNet-34 [48]	73.2%	91.3%	21.8M	7,360M	—
MobileNetV3 [52]	75.2%	—	5.4M	450M	—
PNASNet-5 [73]	74.2%	91.9%	5.1M	588M	150
AmoebaNet-C [105]	75.7%	92.4%	6.4M	570M	3,150
SSPS(ResNet-34) [133]	74.3%	—	—	—	—
NSGANetV1-A3 [85]	76.2%	93.0%	5.0M	570M	27
ProxylessNAS [19]	75.1%	92.5%	7.1M	—	8.3
DARTS [75]	73.3%	91.4%	4.7M	—	4
SNAS [157]	72.7%	90.8%	4.3M	—	1.5
BanditNAS [168]	75.3%	—	5.12M	547M	1.8
EBNAS (large) [119]	67.8%	87.4%	—	**128M**	**0.04**
EPPGA	75.9%	92.7%	5.6M	496M	9.4

4.4.2.1 Overall Performance

The performance of NPPGA and EPPGA is presented in Table 4.4. EPPGA is 0.3% and 0.5% more accurate than NPPGA on CIFAR-10 and CIFAR-100, respectively. The model scales of EPPGA are also smaller than those of NPPGA on the corresponding datasets. As for the search cost, NPPGA requires 1.5 GPU-days on CIFAR-10, which is slightly less than EPPGA's 1.6 GPU-days; however, NPPGA's GPU-days is 0.2 greater than EPPGA's on CIFAR-100. Consequently, the proposed performance predictor can help improve the searched network's performance without substantially increasing the computational cost.

Table 4.4: The comparisons of NPPGA and EPPGA.

Method	Error Rate	Dataset	#Params	GPU-Days
NPPGA	2.80%	CIFAR-10	2.66M	1.5
	18.06%	CIFAR-100	2.39M	2.6
EPPGA	2.50%	CIFAR-10	2.20M	1.6
	16.75%	CIFAR-100	2.12M	2.4

4.4.2.2 Convergence Situations

Since the primary purpose of the performance predictor is to accelerate the evolutionary process, we also examine the population performance during the evolutionary stage. The experimental results on CIFAR-10 and CIFAR-100 are shown in Figure 4.5 and Figure 4.6, respectively.

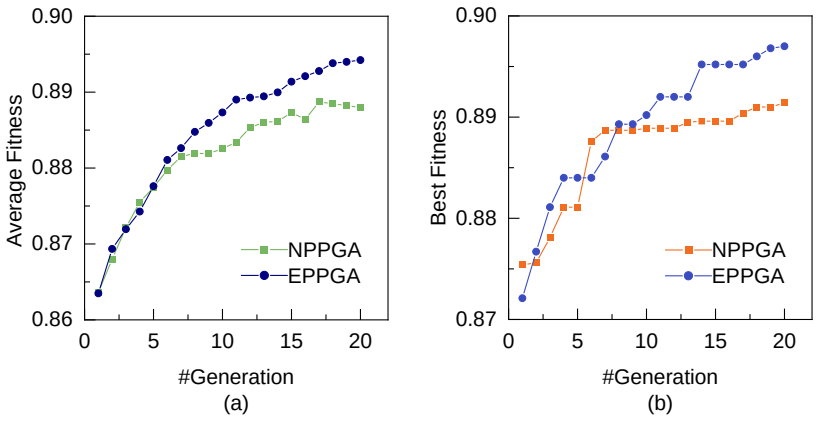


Figure 4.5: The population's performance during the evolutions of EPPGA and NPPGA on CIFAR-10. (a) The average fitness of each generation. (b) The best fitness of each generation.

On CIFAR-10, the average fitness of EPPGA and NPPGA are similar during the first seven generations, but EPPGA's average fitness noticeably outperforms NPPGA in the subsequent generations. The best fitness of each generation follows a similar tendency — there are no significant dif-

ferences between them in the first eight generations. EPPGA's best fitness continues to increase and surpasses NPPGA's in the following generations. This is because there are not enough training samples for the performance predictor in the initial generations, resulting in inferior prediction results. Along with the evolution, more parent-offspring training samples are collected, and the performance predictor is adequately trained, improving the prediction accuracy so that the population's performance of EPPGA maintains substantial growth and surpasses that of NPPGA.

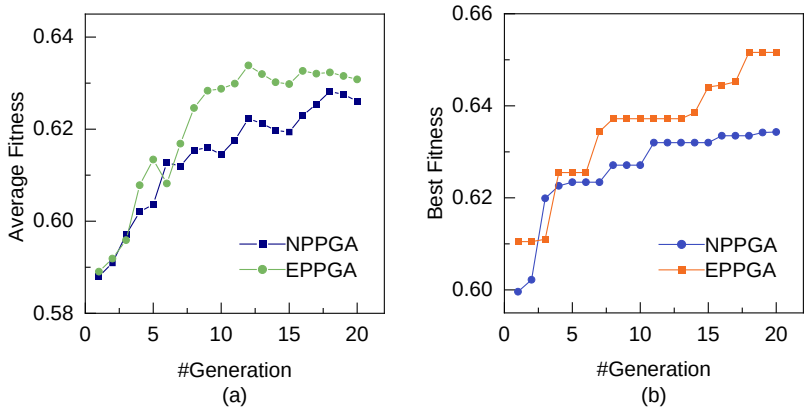


Figure 4.6: The population's performance during the evolutions of EPPGA and NPPGA on CIFAR-100. (a) The average fitness of each generation. (b) The best fitness of each generation.

From Figure 4.6, we observe that the average fitness of EPPGA and NPPGA on CIFAR-100 are similar for the first six generations, with EPPGA's average fitness reaching 0.63 after approximately 11 generations, and then volatility rises. In contrast, NPPGA's average fitness does not reach 0.63 until the 17th generation and subsequently declines in the following generations. Regarding the best fitness in each generation, there are not many differences between EPPGA and NPPGA at first, but EPPGA notably outperforms NPPGA after the sixth generation, achieving the best fitness of 0.651 in the 18th generation and ceasing to update afterward. However, NPPGA's best fitness does not improve substantially after 11 generations,

with the best result being only 0.634, which is inferior to EPPGA.

To conclude, the performance predictor employed in EPPGA can significantly enhance the population's quality after approximately six generations when sufficient training samples are collected. In subsequent generations, the performance predictor accelerates the evolution and improves both the population's quality and the best individual's fitness.

4.4.2.3 Effectiveness of the Random Forest

Within EPPGA, a random forest-based performance predictor plays a crucial role. This section explores the efficacy of random forest by comparing with other machine learning methods, i.e., Support Vector Machine (SVM) and logistic regression. Table 4.5 provides comparative results of different performance predictors.

Table 4.5: The comparisons of different performance predictors.

Predictor	Error Rate	Dataset	#Params	GPU-Days
SVM	2.55%	CIFAR10	2.35M	1.5
	17.02%	CIFAR100	2.10M	2.2
Logistic Regression	2.63%	CIFAR10	2.24M	1.6
	17.15%	CIFAR100	2.25M	2.2
Random Forest	2.50%	CIFAR10	2.20M	1.6
	16.75%	CIFAR100	2.12M	2.4

The random forest-based method demonstrates the lowest error rates on both CIFAR10 and CIFAR100 datasets. SVM and logistic regression also exhibit competitive classification performance, which is slightly worse. In terms of model size and computational cost, the differences among these performance predictors are slight.

4.4.3 Efficiency of Weight Inheritance

In EPPGA, a novel weight inheritance method is proposed: offspring inherit weights from their primary parents, secondary parents, and other networks in the weight pool. In this section, we first compare the training process of networks using the proposed weight inheritance method with that of networks without weight inheritance. Then, we present experimental results of different weight inheritance methods.

4.4.3.1 With and Without the Proposed Weight Inheritance Method

Two individuals are randomly selected from an offspring population and decoded to corresponding networks. The first individual employs a random initialization method without weight inheritance; the second one utilizes the proposed weight inheritance method. Figure 4.7 shows the network training process with SGD on the CIFAR-10 dataset. Please note that the weight inheritance training method is conducted for 20 training epochs, while the one without inheritance is operated for 40 epochs. This is because 20 training epochs are sufficient to achieve high and stable training accuracy using weight inheritance. For the first network (shown in Figure 4.7(a)), the weight inheritance method facilitates rapid weight convergence, taking about 10 epochs to achieve stable training accuracy. In contrast, the random initialization method requires approximately 35 training epochs. Regarding the other network (shown in Figure 4.7(b)), it takes about 15 training epochs to achieve stable and high training accuracy, which costs 30 more training epochs without weight inheritance.

The two network training processes with SGD on CIFAR-100 are depicted in Figure 4.8. The training accuracy becomes stable for the first network using weight inheritance after 16 training epochs, while the training protocol without weight inheritance takes more than 30 epochs to achieve similar accuracy. For the second network, the training accuracy using weight inheritance after 15 training epochs is higher than the accuracy

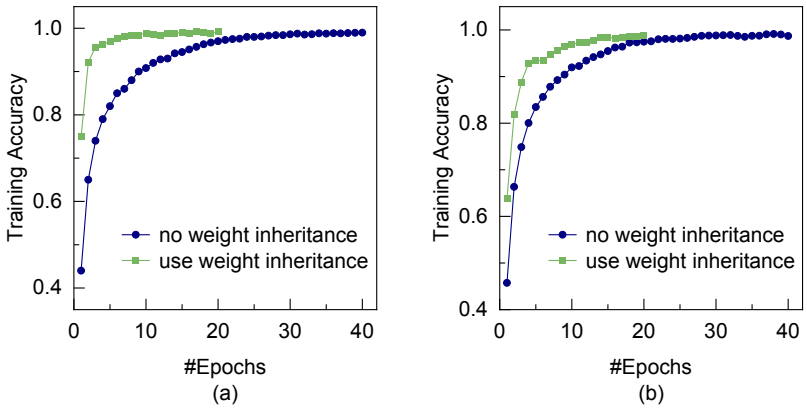


Figure 4.7: The training accuracy change situations using or not using the weight inheritance method on CIFAR-10. (a) and (b) shows the results on two different networks.

without inheritance for 40 training epochs, demonstrating the effectiveness of weight inheritance. By employing the proposed weight inheritance method, the training time of the candidates is reduced by at least 50% compared to not using weight inheritance.

4.4.3.2 Different Weight Inheritance Methods

We also compare the performance among different weight inheritance methods. Offspring can (1) inherit weights only from their primary parents (denoted as *One Parent*), (2) inherit weights from both parents (denoted as *Two Parents*), or (3) inherit weights from the parents and the weight pool (denoted as *EPPGA*). Two networks are randomly selected from the offspring, and each is evaluated using the above three weight inheritance methods mentioned above. Table 4.6 and Table 4.7 present the experimental results on CIFAR-10 and CIFAR-100, specifically reporting the training accuracies on the 5th, 10th, 15th, and 20th training epochs.

For the first network on CIFAR-10, the "two parents" inheritance method outperforms the "one parent" method for the four chosen epochs. EPPGA's

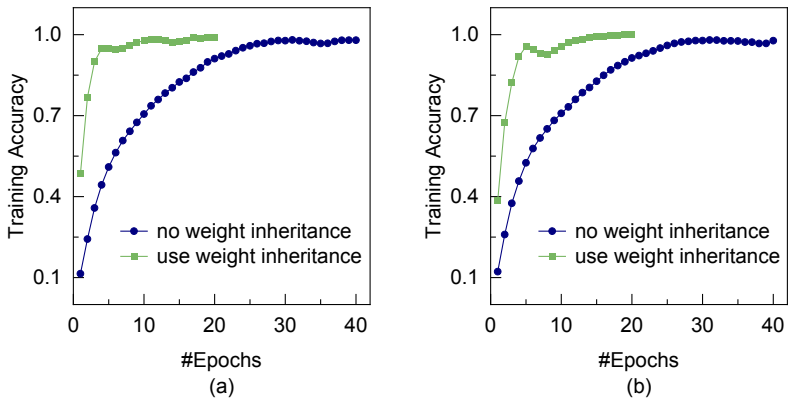


Figure 4.8: The training accuracy change situations using or not using the weight inheritance method on CIFAR-100. (a) and (b) shows the results on two different networks.

inheritance method achieves a lower accuracy at the 5th epoch than the "two parents" method. However, it achieves better performance on the three subsequent epochs, indicating the proposed weight inheritance may converge slowly at the beginning of the training but accelerates convergence as training progresses. EPPGA performs well on the selected four epochs for the second network, indicating its higher convergence speed than the other two weight inheritance methods. On CIFAR-100, EPPGA

Table 4.6: The training accuracy(%) among three weight inheritance methods comparisions on the **CIFAR-10** dataset.

Network	Inheritance Method	E5	E10	E15	E20
1	One Parent	96.89	98.34	98.76	99.17
	Two Parents	97.10	98.49	98.81	99.14
	EPPGA	97.05	98.87	99.02	99.23
2	One Parent	92.42	96.28	97.77	98.45
	Two Parents	92.86	96.90	98.05	98.48
	EPPGA	93.41	96.89	98.38	98.76

achieves significantly higher accuracies for the first network and much higher accuracies on the last two selected epochs for the second network. In general, the proposed weight inheritance method can substantially accelerate weight convergence and achieve a higher convergence speed than other weight inheritance methods.

Table 4.7: The training accuracy(%) among three weight inheritance methods comparisions on the **CIFAR-100** dataset.

Network	Inheritance Method	E5	E10	E15	E20
1	One Parent	94.79	97.05	97.65	97.68
	Two Parents	94.52	97.10	97.24	98.24
	EPPGA	94.90	97.80	97.50	98.97
2	One Parent	92.58	96.53	98.68	98.94
	Two Parents	92.86	97.02	98.54	98.64
	EPPGA	95.68	95.64	99.33	99.94

4.4.4 Analysis of the backbone block structure

In EPPGA, a new backbone block structure is proposed inspired by the MobileNetV3 block. The proposed structure constructs the search space and aims to reduce the computational cost during the evolution. Besides, the searched network is expected to perform well with a small model scale and low computational complexity. To demonstrate the effectiveness of the proposed backbone structure, a comparison algorithm is implemented called MPPGA. The search space of MPPGA is based on the MobileNetV3 block, and all the other parameters are identical to those of EPPGA. The comparisons are presented in in Table 4.8.

On CIFAR-10, MPPGA costs 3.3 GPU-days to search for an appropriate network architecture, which is more than twice the time needed by EPPGA. Regarding the searched network, MPPGA's 2.79% error rate is worse than EPPGA's 2.50%. However, the network's model size of

Table 4.8: The comparisons of MPPGA and EPPGA.

Method	Error Rate	Dataset	#Params	#FLOPs	GPU-Days
MPPGA	2.79%	CIFAR-10	3.66M	194M	3.3
	16.77%	CIFAR-100	4.11M	218M	3.6
EPPGA	2.50%	CIFAR-10	2.20M	185M	1.6
	16.75%	CIFAR-100	2.12M	207M	2.4

MPPGA is 1.66 times that of EPPGA, and the number of FLOPs is also 9M larger than EPPGA. On CIFAR-100, MPPGA's GPU-days is 1.5 times that of EPPGA. The error rate of MPPGA is 0.02% worse than EPPGA, but the model size is 1.94 times of EPPGA, along with a larger number of FLOPs. The proposed block can help reduce the search computational cost and enable the network to maintain classification performance with a much smaller model size and a lower computational complexity than the MobileNetV3 block.

4.5 Chapter Summary

This chapter aims to propose an efficient and effective ENAS method to search for lightweight networks. The objective has been achieved by designing a new backbone block structure, proposing a new performance predictor, developing a new weight inheritance method, and inventing new genetic operations. To lower the computational cost and reduce the model scale of the searched network, an efficient block architecture is designed, inspired by the MobileNetV3 block but more lightweight. A random forest-based performance predictor is proposed to estimate the performance of the offspring and help generate high-quality offspring, improving the population's performance and accelerating the evolution. Furthermore, a new weight inheritance method is proposed, which can make the offspring inherit weights from their parents and other existing trained

networks, improving the weight convergence speed while training. Finally, corresponding genetic operations are designed to produce offspring that can maintain a large proportion of the same as the primary parent, improving the predictor's performance and promoting weight inheritance.

The proposed algorithm is examined and the performance is compared with state-of-the-art peer competitors. The results show that EPPGA is an excellent ENAS algorithm regarding the search cost, the prediction error rate, and the searched network's size and computational complexity. Furthermore, the experiments indicate the proposed performance predictor can effectively accelerate the evolution and search for better candidates. The weight inheritance experiments prove the effectiveness of the proposed weight inheritance mechanism. The experiments using different search spaces show the proposed backbone structure can maintain the performance with smaller model size and lower computational complexity.

EPPGA is based on a standard NAS framework, and the following chapter will explore a new framework, i.e., one-shot NAS, which is typically of higher efficiency.

Chapter 5

One-Shot Neural Architecture Search using Reliable Fitness Evaluations

5.1 Introduction

For one-shot NAS, only a single supernet needs to be trained initially, and all the candidate networks directly inherit weights from the supernet without further training, thus saving much computational cost. However, the training of the supernet still requires significant computing resources. This chapter proposes an efficient supernet training strategy to further reduce the overall computational cost.

One-shot NAS also suffers from limited reliability in evaluating the performance of candidate networks (a.k.a. subnets). During the search stage, the subnets inherit weights from the supernet, and their fitness is assessed based on these weights. However, researchers have noted that the ranking correlation between inherited weights and the weights from stand-alone training is low [162], leading to unfair comparisons among subnets. This issue has been attributed to the extensive sharing extent of weights in the supernet [96], where all subnets that contain the same op-

eration in a layer use the same weights. These replicated weights may not be optimal for individual subnets, leading to potential degradation in performance. This chapter fine-tunes the supernet weights by directly utilizing the candidates to be evaluated as a guide for the fine-tuning process. By reducing the sharing extent from the fundamental role of the supernet, i.e., evaluating the subnets, this chapter aims to improve the reliability of performance estimation in one-shot NAS.

Accelerating the convergence of the EC-based search process can reduce the computational cost of NAS algorithms. Population initialization determines the initial solutions (candidate architectures in NAS), and it has been demonstrated that good initial solutions can facilitate the evolutionary process [102], reduce computational costs [62], increase the probability of finding the optimal solutions [88], and improve the stability of search results [94].

5.1.1 Chapter Goals

The overall goal of this chapter is to propose a one-shot ENAS algorithm for image classification, which enables accurate fitness evaluations, efficient supernet training, and a powerful population initialization to facilitate the search process. To accomplish this, four specific objectives are outlined as follows.

1. Propose a new efficient supernet training strategy to reduce the computational cost of training the weights of the supernet. The proposed strategy leverages the characteristics of the supernet architecture by training paths with 'larger' operations first, then replicating corresponding weights for 'smaller' operations, and finally, further training these paths.

2. Propose an effective supernet fine-tuning strategy to offer accurate fitness evaluations for all candidate network architectures. The su-

pernet weights are expected to be fine-tuned based on the distribution of the candidate network architectures, which can reduce the sharing extent and provide more accurate evaluations.

3. Propose a new population initialization method to select potentially well-performing candidate network architectures as initial solutions, facilitating the evolutionary search. The proposed initialization method is based on information extracted from the supernet training process.

4. Provide further analysis and discussions on the limitations of current one-shot NAS methods, and discuss the motivations behind the proposed supernet training, weight fine-tuning, and population initialization methods. This analysis aims to provide insights into the rationale and benefits of the proposed approaches.

5.1.2 Chapter Organization

The remainder of this chapter is organized as follows. Section 5.2 provides the analysis of the motivations for this work. Section 5.3 presents detailed information on the proposed algorithm. Section 5.4 describes the experimental design information, including dataset and parameter settings. Section 5.5 presents the results and analysis of the experiments. Finally, Section 5.6 provides the conclusion.

5.2 Analysis of Motivations

Sample-based one-shot NAS [46] consists of two consecutive stages: supernet training and architecture search. The supernet encompasses the entire search space $\mathcal{A}$ and its weights $W_{\mathcal{A}}$ are trained by iteratively sampling a single path or subnet a within $\mathcal{A}$, as follows:

$$W_{\mathcal{A}}^{*} = \arg\min_{W} \mathop{\mathbb{E}}_{a \sim U(\mathcal{A})} [\![\mathcal{L}_{tra}(a, W(a))]\!] , \tag{5.1}$$

where $W^*_{\mathcal{A}}$ denotes the optimized weights of the supernet, $\mathbb{E}[\![\cdot]\!]$ represents the expectation of variables, $\mathcal{L}_{tra}(a, W(a))$ signifies the training loss of subnet a with weights $W(a)$, and $a \sim U(\mathcal{A})$ indicates that subnet a follows a uniform distribution in the search space $\mathcal{A}$.

In the second stage, the goal is to find the subnet with the best performance a^* in the search space $\mathcal{A}$, expressed as:

$$a^* = \arg\max_{a \in \mathcal{A}} \mathrm{ACC}_{eva}(a, W^*_{\mathcal{A}}(a)) \ , \tag{5.2}$$

where $W^*_{\mathcal{A}}(a)$ denotes the inherited weights of subnet a from the learned supernet weights $W^*_{\mathcal{A}}$, and $\mathrm{ACC}_{eva}(a, W^*_{\mathcal{A}}(a))$ refers to the evaluation accuracy of a with the inherited weights, which serves as a measure of the subnet's performance.

Although one-shot NAS has attracted significant attention, there remain limitations when considering the underlying principles more deeply. The supernet is typically quite large since it must encompass the entire search space. A layer consists of m possible operations that are connected in parallel, i.e., $o_1, o_2, \ldots, o_m$. Assuming the total number of layers in the supernet is n, there will be m^n subnets within the supernet. For the k-th operation of the l-th layer, o^l_k, its weights are influenced by all the subnets that contain o^l_k, and the total number of such subnets is m^{n-1}, which is typically a rather large number[4]. This characteristic may result in two issues: inefficient training of the supernet and unreliable performance estimation of the subnets.

During supernet training, a subnet a is sampled from the search space $\mathcal{A}$ as shown in Equation (5.1), and the weights of each operation are influenced by the m^{n-1} subnets, resulting in a large computation budget for supernet training. A promising weight initialization could accelerate weight convergence. In a specific layer, the operations $o_1, o_2, \ldots, o_m$ typically have

[4]For simplicity, the mathematical calculations in this section do not account for the variable length of subnets, i.e., some layers in the subnet may not exist. All the subnets are assumed to be of maximum length.

different widths (number of kernels) and kernel sizes. The operation with the largest width and kernel size, denoted as o_{large}, can be optimized first, and then the weights of all other operations can be copied from the trained operation o_{large}. This supernet weight initialization mechanism is expected to facilitate the subsequent weight optimization process.

Another concern is the reliability of performance estimation using the weights inherited from the supernet. As seen from Equation (5.2), the measured performance of subnet a is closely related to its inherited weights $W^*_{\mathcal{A}}(a)$. However, the optimal weights for a subnet are often not the same as the weights trained in the supernet, as each operation's weight $W^*_{\mathcal{A}}(a, o)$ in $W^*_{\mathcal{A}}(a)$ is affected by a large number of subnets, and may deviate from the optimized weights $w^*_{a,o}$, leading to potential inaccurate performance estimation. One way to alleviate this issue is to reduce the number of subnets that affect each other, as supported by [16] — reducing the sharing extent can improve the reliability of the estimated performance. The main task of the supernet is to aid in the subnets' performance evaluation. Therefore, to enhance evaluation reliability, it would be beneficial to fine-tune the weights of the supernet to better fit the subnets to be evaluated before the evaluation, as shown in Equation (5.3):

$$W_i^* = \arg\min_{W} \mathop{\mathbb{E}}_{a \in \{a_1^i, a_2^i, \ldots, a_p^i\}} [\![\mathcal{L}_{tra}(a, W(a))]\!] \,, \tag{5.3}$$

where $\{a_1^i, a_2^i, \ldots, a_p^i\}$ refers to all the p subnets to be evaluated in the i-th iteration, and W_i^* is the optimized supernet weights for the evaluation of the subnet in the i-th iteration. Although fine-tuning may increase the computational cost, the additional computational cost will not be large, as the number of subnets to be sampled for fine-tuning in each iteration is typically much smaller (p) compared to the number of subnets used for supernet training (m_n) and the convergence of weights is expected to be fast. However, a major concern with this fine-tuning strategy is the increased time and computational consumption. Nevertheless, the additional time and computational cost for the fine-tuning could be acceptable for the fol-

lowing reasons:

(1) The initial supernet weights of the i-th iteration W_i are inherited from the optimized weights of the previous iteration W_{i-1}^*, which have already been fine-tuned by the subnets of the previous iteration. Since the populations share many common genes, the difference between W_i^* and W_{i-1}^* is expected to be small, and the fine-tuning process is unlikely to involve a significant amount of computation.

(2) The number of subnets to be sampled for fine-tuning in each iteration is typically much smaller (p) compared to the number of subnets used for supernet training (m_n). Typically, p ranges from a few dozen to a hundred, while m_n can be much larger. The fine-tuning process among this smaller number of subnets is expected to be quick, i.e., the convergence of weights is expected to be fast.

Additionally, during the supernet training process, a large number of subnets are activated and optimized, and some of these subnets may show promising performance based on their training progress and evaluation results. These potentially good subnets can serve as a valuable source of information for the architecture search process, as they can guide the search towards more promising regions of the search space. By using the potentially good subnets as the initial population for the search, the architecture search process can benefit from an informed starting point, which may help in accelerating the search process and improving the chances of finding high-performing architectures. This can be especially useful in cases where the search space is large and complex.

5.3 The Proposed Method

This section describes the comprehensive procedures and the critical components of the proposed algorithm. The proposed algorithm is called TIFNAS, an acronym standing for efficient supernet Training, effective

population Initialization, and effective supernet Fine-tuning for Neural Architecture Search.

5.3.1 Algorithm Overview

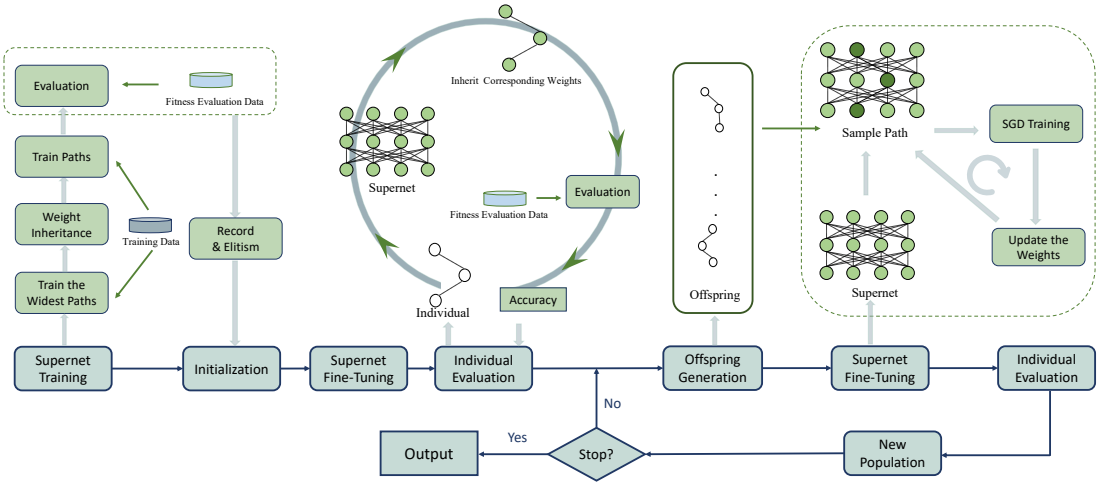


Figure 5.1: Overview: TIFNAS comprises supernet training, population initialization, and evolutionary search. The supernet is efficiently trained using the proposed training method, and the well-performing subnets during the training are selected as the initialized individuals. In the evolutionary search process, the supernet is further fine-tuned to provide more precise evaluations.

Figure 5.1 illustrates the overall framework of the proposed TIFNAS. Prior to the search, the supernet is predefined, covering the entire search space (see Section 5.3.2), and its weights are trained using the method presented in Section 5.3.3. Specifically, the three widest paths inside the supernet are trained; then, the other operations inherit weights from the three paths, which facilitates the training process; subsequently, the possible paths are trained: for each batch of data in a training epoch, a path is randomly sampled/selected, and the weights are trained employing

SGD. The performance of the selected path/subnet is evaluated on the fitness evaluation data if it is the last training epoch of SGD. The candidate individuals for the first generation are then determined by the proposed initialization method (see Section 5.3.4), which involves selecting the best subnets evaluated during the last training epoch to enhance the population quality.

Following the initialization, the search process commences, and the weights of the supernet are fine-tuned to obtain a more accurate assessment of the initial population. During the evaluation, each individual (a.k.a. subnet) inherits the weights from the supernet and is evaluated on the fitness evaluation data to obtain its classification accuracy as fitness (see Section 5.3.5). Offspring are then generated (see Section 5.3.6), and the supernet is further fine-tuned based on the offsprings (see Section 5.3.7). Specifically, for each batch of data, an offspring individual is randomly selected as the subnet, and the corresponding weights are trained using SGD and updated. Afterward, the offspring population is evaluated to obtain fitness values, and a new population is generated (see Section 5.3.8). This process is repeated for a predefined number of generations as the stopping criterion. Ultimately, the best individual/subnet is output as the searched result.

5.3.2 Supernet Architecture

The supernet of TIFNAS encompasses the entire search space, as depicted in Figure 5.2. TIFNAS aims to determine the optimal depth (number of layers/blocks), width (number of feature maps), and kernel sizes of the convolutions. The supernet architecture is designed to include DCNNs with different choices. Typically, the depth of the supernet adopts the maximum number of layers/blocks. For each layer/block, the operations of all possible combinations of widths and kernel sizes are parallelly aligned. This connection approach alleviates the weight co-adaptation problem,

especially when compared to using only one operation with the largest width and kernel size for a layer/block.

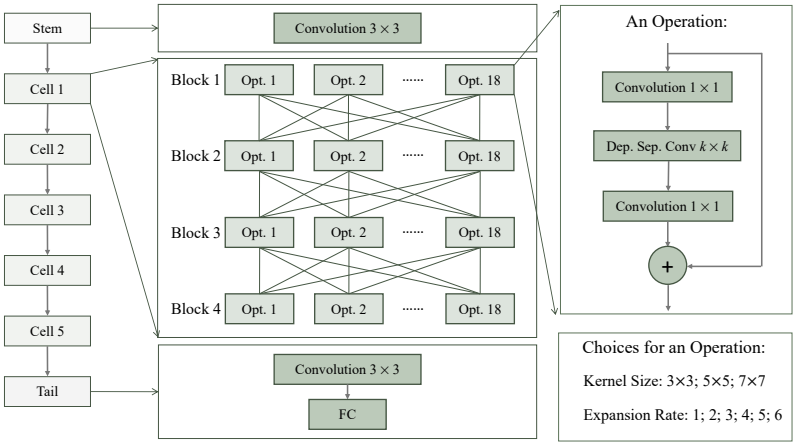


Figure 5.2: The architecture of the supernet. Each cell consists of four blocks, with each block having 18 optional operations. An operation comprises a 1×1 convolution, a depth-wise separable convolution, and another 1×1 convolution. There are 3 candidate kernel sizes and 6 expansion rates to choose from for an operation.

The supernet structure can be divided into a stem, five cells, and a tail following other methods [83]. The stem consists of a 1×1 convolution. Each cell contains four blocks, and each block has 18 optional operations. The tail consists of a 1×1 convolution followed by a fully-connected layer for classification. Specifically, an operation comprises a 1×1 convolution to expand the width, a depth-wise separable convolution to construct informative features, and another 1×1 convolution to reduce the width to the same as the input of the operation. Additionally, the input feature maps and the feature maps after convolutions are added together as the final output of the operation. For each operation, the kernel size of the depth-wise separable convolution is optional, and there are three candidate kernel sizes to choose from. The expansion rate of the first convolution is also optional, with six candidate expansion rates available. If an operation has

an expansion rate of e, the width (a.k.a. the number of feature maps) of the first two convolutions will be $e \times w$, where w represents the width of the input feature maps.

All subnets are encompassed within the large supernet, and each subnet is composed of five cells. Each cell can have a flexible number of blocks, where each block has a specific kernel size and expansion rate. In GA, an individual is represented by a string consisting of five vectors, with each vector corresponding to a cell. For example, [(3,4), (7,6), (5,2)] represents a cell with three blocks, where the kernel size of each block is 3, 7, and 5. The expansion rates of the three blocks are 4, 6, and 2, respectively.

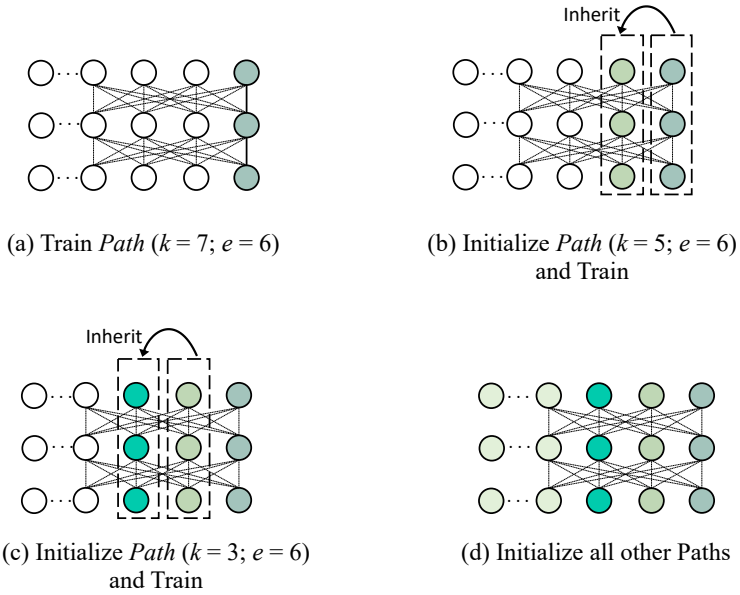
5.3.3 Supernet Training

The supernet is large and thus challenging to train. To improve the training efficiency and training effectiveness (measured by the training loss), TIFNAS comprises a proposed progressive initialization method along with the widely used single-path sampling training approach [25].

5.3.3.1 Progressive Weight Initialization

Figure 5.3 illustrates the key steps of the progressive weight initialization method for the supernet. To simplify notation, we use $Path$ (k; e) to denote the path/subnet with the greatest depth, where each cell has four blocks, and each block utilizes the expansion rate of e and a kernel size of k. Firstly, we activate and train the "largest" path, $Path$ ($k = 7$; $e = 6$); then, $Path$ ($k = 5$; $e = 6$) inherits weighs from $Path$ ($k = 7$; $e = 6$) and is then trained; subsequently, $Path$ ($k = 3$; $e = 6$) inherits weighs from $Path$ ($k = 5$; $e = 6$) and is then trained; finally, all the other paths with the largest depth are initialized by a new weight inheritance method.

The weight initializations for $Path$ ($k = 5$; $e = 6$) and $Path$ ($k = 3$; $e = 6$) belong to the category of inheriting weights from convolutions of the same width (e) but with different kernel sizes (k). On the other hand, the



(a) Train *Path* ($k = 7$; $e = 6$)

(b) Initialize *Path* ($k = 5$; $e = 6$) and Train

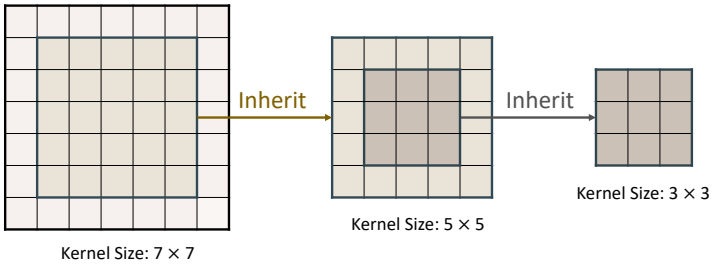
(c) Initialize *Path* ($k = 3$; $e = 6$) and Train

(d) Initialize all other Paths

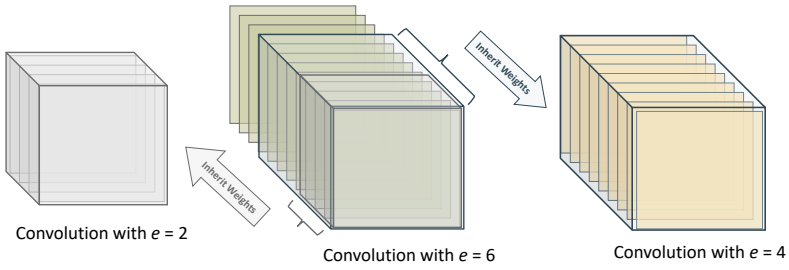
Figure 5.3: The progressive initialization method comprises four steps: (a) training the path with all blocks of $k = 7$ and $e = 6$; (b) initializing the weighs of the path with all blocks of $k = 5$ and $e = 6$, and then training; (c) initializing the weighs of the path with all blocks of $k = 3$ and $e = 6$, and then training; (d) initializing all other paths.

weight initializations of all other paths belong to the category of inheriting weights from convolutions of different widths (e) but with the same kernel size. Figure 5.4 illustrates the two types of weight inheritance methods. The first type involves each convolution kernel inheriting weights from the central part of the larger convolution kernel, with the selected central part size being consistent with the convolution kernel to be initialized. The second type applies when all other paths inherit weights from the three trained paths with the largest expansion rate. Each convolution to be initialized selects the convolution of the same kernel size with the largest expansion rate at the same block and inherits weights from the previous layers of the large convolutions, where the number of layers is the same as

the convolution to be initialized. This way, all the paths will be initialized by copying weights from the trained path, which can facilitate the training process and significantly reduce the training cost.



(a) Weight inheritance from convolutions of different kernel sizes



(b) Weight inheritance from convolutions of different widths / numbers of kernels

Figure 5.4: Two kinds of weight inheritance within the supernet. (a) The convolution of a kernel size of 5×5 inherits weights from the convolution of a kernel size of 7×7, and the convolution of a kernel size of 3×3 inherited weights from the convolution of a kernel size of 5×5. (b) The convolutions of $e = 2$ and $e = 4$ inherit weights from the convolution of $e = 6$, respectively.

5.3.3.2 Single Path Sampling Training

After initializing the weights using the progressive weight initialization method, single-path sampling training will be employed to train the weights of the supernet. The training process is described in Algorithm 6. A specific number of training epochs T and a predefined learning rate α are

used. The training continues until T training epochs have been completed (lines 1-11), with each batch of data being traversed in an epoch (lines 2-10). In each epoch, a path is randomly selected from the supernet (line 3), and the weights of the path are trained and updated using SGD (lines 4-6). If it is the last training epoch, the path is further evaluated on the evaluation data, and the loss is computed and recorded for the population initialization process (lines 7-9). Through this process, the weights of the supernet are trained to adapt to all candidate subnets.

Algorithm 6: Single Path Sampling-based Supernet Training

Input: Supernet S, training data $\mathcal{D}_{train}$, evaluation data $\mathcal{D}_{eva}$, learning rate α, number of epochs T.

Output: Weights θ of the supernet S.

```
1  for t = 1 to T do
2      for data batch D_i in D_train do
3          Sample a path p from the supernet S;
4          Compute the loss L(S_p(θ), D_i);
5          Compute the gradient ∇_θ L(S_p(θ), D_i);
6          Update the weights θ ← θ − α∇_θ L(S_p(θ), D_i);
7          if t == T then
8              Evaluate p on D_eva by computing the evaluation loss
                L(S_p(θ), D_eval);
9          end
10     end
11 end
```

The path evaluation is conducted after the training of the path instead of after the training epoch is finished. This is because the training of other paths will affect the weights of the current path, reducing the evaluation reliability of this path.

It should be noted that both the training data $\mathcal{D}_{train}$ and the evaluation data $\mathcal{D}_{eva}$ are selected from the training dataset $\mathcal{D}_{TRA}$, which is separate

from the test dataset $\mathcal{D}_{TEST}$. There is no overlap between them, as expressed by Equation (5.4):

$$\mathcal{D}_{train} \cup \mathcal{D}_{eva} = \mathcal{D}_{TRA}, \quad \mathcal{D}_{train} \cap \mathcal{D}_{eva} = \emptyset \, . \tag{5.4}$$

5.3.4 Population Initialization

Most evolutionary NAS approaches employ a *random initialization* method, randomly selecting architectures from the large search space. However, TIFNAS takes advantage of the information obtained during the supernet training process to select potentially well-performing individuals as the initial population. Specifically, during the last training epoch of the supernet training, a number of candidate architectures/subnets are evaluated, and the architectures with the lowest evaluation losses are chosen as the initial population.

There are two main reasons that TIFNAS uses this specific population initialization method:

(1) **Facilitates the search in the large search space:** The search space in NAS is typically huge, with a vast number of possible architectures to explore (as discussed in Section 5.2), but the population size is usually limited because of the available computational budget. Randomly initializing the population may result in individuals scattered across the search space without covering promising regions. However, the selected initial individuals can suggest potentially promising areas in the space, and provide a good starting point to facilitate the evolutionary architecture search.

(2) **Computationally efficient:** Evaluating a large number of individuals to select good ones as the initial population can be computationally expensive. TIFNAS, on the other hand, takes advantage of the supernet training process and evaluates the sampled subnets in the

last training epoch. Since the supernet is a single network, this evaluation can be done efficiently using existing computational resources, making the population initialization method computationally efficient.

5.3.5 Individual Evaluation

During the evolutionary search, each candidate network must be evaluated to obtain its corresponding fitness, which shows its performance and guides the search. In TIFNAS, the individual fitness evaluation is simple and efficient. The weights of the individual architectures are directly inherited from the corresponding operations in the supernet, without being trained separately. Subsequently, the networks are evaluated on the fitness evaluation data $\mathcal{D}_{eva}$, and their classification accuracies are recorded as the candidate's fitness.

5.3.6 Offspring Generation

TIFNAS generates offspring based on the current population, utilizing three main procedures following traditional GAs: (1) selecting parent individuals with promising fitness, (2) performing crossover operations, and (3) carrying out mutation operations. This subsection will provide a detailed description of these three procedures.

Parent Selection

TIFNAS utilizes a binary tournament selection strategy to select parent individuals from the current population. This process involves randomly sampling two individuals and comparing their fitness values. The individual with the better fitness value will be selected as one of the parents and included in the mating pool. This process is repeated until the number of individuals in the mating pool reaches the population size.

Two-Level Crossover

TIFNAS employs a two-level crossover operation that combines genes from both the cell level and the block level. Since each individual in TIFNAS represents a neural network architecture with multiple cells, and each cell contains several blocks, this two-level crossover operation allows for the hybridization of entire cells and the exchange of specific blocks between parents.

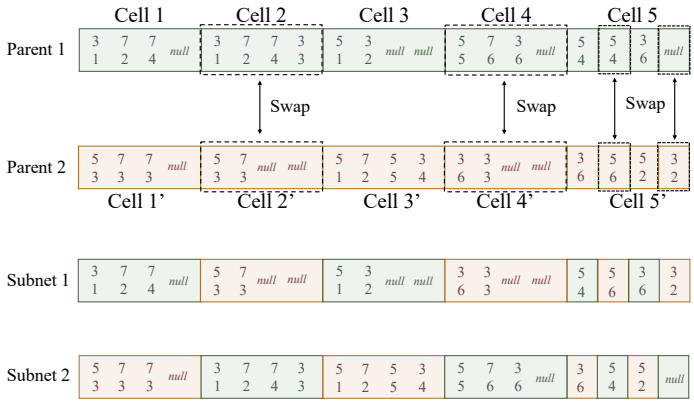


Figure 5.5: An example of the two-level crossover operation. For the cell-level crossover, the second and fourth cells are selected to swap. The fifth cell is chosen to perform the block-level crossover, and the second and fourth blocks are selected to swap.

An example of the crossover operation between two parent individuals is illustrated in Figure 5.5. In this example, the second and fourth cells are randomly selected to swap with the corresponding cell of another parent. One of the remaining three cells is randomly selected for block-level crossover, and in this case, the fifth cell is chosen. Inside the fifth cell, two blocks among the four blocks are randomly selected, and the second and fourth blocks are exchanged with the corresponding blocks from another parent. Please note that the fifth cell of *Parent* 1 only contains three blocks, and the non-existing block is represented by *null*, which can also

be swapped with other blocks. After the crossover, the fifth cell of *Offspring* 2 contains three blocks: the first and third blocks are from *Parent* 2, and the second block is from *Parent* 1. The two-level crossover operation effectively hybridizes the genes between the parents.

Parameter-Level Mutation

Algorithm 7: Parameter-Level Mutation

Input: A subnet s.
Output: A mutated subnet s.

```
1  n_m ← Determine the number of mutation operations;
2  for i = 1 to n_m do
3      cell ← Randomly select a cell from s;
4      Randomly select a type from {adding, removing, modifying};
5      n_b ← The number of blocks in cell;
6      if adding and n_b < 4 then
7          block_m ← Generate a new block;
8          Choose a position & Insert block_m;
9      else if removing and n_b > 2 then
10         Choose a block & Remove it;
11     else
12         Choose 1 or 2 blocks randomly;
13         Modify the parameter(s) of the block;
14     end
15 end
```

TIFNAS incorporates three types of mutation operations, namely *adding*, *removing*, and *modifying*. Algorithm 7 provides a detailed description of the mutation operation in TIFNAS. Specifically, the number of mutation operations to be performed is determined first (line 1). A cell is randomly selected for each operation, and a mutation type is randomly chosen (lines

3-4), followed by counting the number of blocks in the cell (line 5), and the operation is based on the selected mutation type as follows.

- *Adding*: If the number of blocks does not reach the maximum value (line 6), the parameters for a new block are generated and inserted at a random position in the cell (lines 7-8). In the example presented in Figure 5.6, a new block is added next to the second block, increasing the number of blocks from 3 to 4.
- *Removing*: If the number of blocks exceeds the minimum value (line 9), a random block is chosen and removed (line 10). In Figure 5.6, the third block is removed after the $removing$ mutation.
- $Modifying$: One or two blocks in the cell are randomly selected (line 12), and their parameters are modified (line 13). In Figure 5.6, the second block's expansion rate and the third block's kernel size are changed.

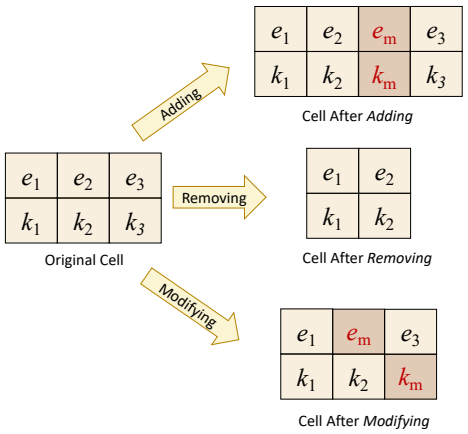


Figure 5.6: An example of three types of parameter-level mutations. For the adding mutation, a new block is generated and inserted into the cell; for the reducing mutation, a block is selected and removed; for the modification mutation, some parameters are changed.

5.3.7 Supernet Fine-Tuning

In TIFNAS, the supernet weights are fine-tuned to better fit the population to be evaluated, resulting in more accurate assessments. The supernet is fine-tuned prior to evaluating the individuals in the population for both the initial generation and the subsequent generations during the evolutionary process.

The fine-tuning process is similar to the supernet training process (as illustrated in Algorithm 6), namely, for each batch of data, a path/subset is selected and trained. There are mainly two distinctions.

One key distinction is that the paths/subnets are selected from the population to be evaluated, instead of from the extensive search space. This approach offers several advantages. First, the primary objective of the supernet is to facilitate the assessment of candidate individuals' performance, and directly employing the individuals to be evaluated for fine-tuning the supernet is sufficient for the weight fine-tuning. Second, the number of subnets for the fine-tuning (each time) is equal to the population size, which is a relatively small number, thus mitigating the issue of weight co-adaptation. Third, training the weights of the supernet with fewer paths to converge is easier. It may take a much longer time to train the weights of the supernet to convergence by randomly sampling paths in the search space, but the computational cost is essentially shrunk by only sampling from the population to be evaluated.

The other distinction is about the stopping criterion. While TIFNAS prescribes a specific number of training epochs in the supernet training stage, the supernet is trained until weight convergence during fine-tuning. This is because the weights are easy to converge during fine-tuning, and adequately trained weights can provide more accurate evaluations of candidate individuals.

5.3.8 New Population Generation

In each generation, the individuals with good performance are selected from both the current population and the offspring population to form a new population. To achieve this, a binary tournament selection is applied to the combined population of both the current and offspring populations, and the better individual is selected to move on to the next generation. Additionally, an elitism selection is applied to ensure that the best individual from the current generation is included in the new population, preserving the best architecture found so far.

5.4 Experimental Settings

5.4.1 Benchmark Datasets

In the experiments, the performance of the proposed TIFNAS method is evaluated using three commonly used benchmark datasets in image classification research, namely CIFAR-10, CIFAR-100, and ImageNet. Detailed information about these datasets can be found in Section 3.3.1 (see page 84). These datasets are selected for two primary reasons. Firstly, they offer a diverse range of difficulty levels, as they encompass varying quantities of images, image sizes, and classification categories. This enables a comprehensive evaluation of TIFNAS's performance. Secondly, numerous peer competitors have also employed these datasets in their evaluations, thereby facilitating a fair comparison between TIFNAS and other studies.

5.4.2 Parameter Settings

TIFNAS primarily consists of two main processes: supernet training and architecture search. The latter process is further divided into supernet fine-tuning and the GA process. Besides, to enable a fair comparison with other peer methods, a post-search evaluation process is required to measure the

performance of the searched architecture. The parameter settings for these processes are detailed below.

Supernet training: In TIFNAS, three single paths with the blocks of the largest width are initially trained, followed by the sampling and training of all other paths. For the widest single path training, no specific number of training epochs is provided on CIFARs, and the weights are fully trained; each path is trained for 5 epochs on ImageNet considering the available resources; for all path training, the number of training epochs is set to 100 for CIFARs and 20 for ImageNet, considering the available computing resources. Other settings remain the same and follow [46]: the learning rate is 0.025, the momentum is 0.9, the weight decay is 3×10^{-4}, and the batch size is 96. The batch size is a relatively small number because a path is sampled for each batch of data, and a smaller batch size can help traverse more paths.

Supernet fine-tuning: The training settings for fine-tuning are identical to those of supernet training, except for the stopping criterion. For fine-tuning, the weights are trained until the training loss does not decrease further for five epochs.

Genetic algorithm: The population size is set to 50, and the number of generations is set to 10. Although the number of generations is relatively small compared to most GAs, it is sufficient for TIFNAS to search for suitable architectures, as a new population initialization method is employed to facilitate the search.

Post-search evaluation: The best-found architecture must be fully evaluated to enable comparison with other methods by training and testing the corresponding network. Specifically, SGD is utilized as the optimization method; a cosine annealing learning rate policy is employed, with a warmup step of 30 epochs, and maximum and minimum learning rates of 0.1 and 0.0008, respectively. The total number of training epochs is 300. The image augmentation method of *cutout* is employed, and a dropout rate of 0.2 is used for CIFARs, but not for ImageNet, as the data amount

for ImageNet is significantly larger. Additionally, a squeeze and excitation (SE) layer is added to each block for ImageNet to enhance the feature extraction ability [52, 115].

5.5 Results and Analysis

This section presents the experimental results to demonstrate the effectiveness and efficiency of the proposed TIFNAS. Specifically, Section 5.5.1 provides the overall performance of TIFNAS and compares it with peer competitors; Section 5.5.2 illustrates the training process of the proposed supernet training method and contrasts it with the vanilla training method; Section 5.5.3 investigates the impact of the proposed population initialization method; and finally, Section 5.5.4 presents a comparison between employing and not employing the proposed supernet fine-tuning strategy.

5.5.1 Overall Results

TIFNAS is evaluated on CIFAR-10 and CIFAR-100 datasets for five trials, and the results are reported. The total computational cost is measured by GPU-days and consists of both the supernet training cost and the search cost [5]. Additionally, the model size of the searched network, measured by the number of parameters, is presented. Due to the large size and complexity of ImageNet, TIFNAS is tested only once, and the computational complexity of the network is provided, which is quantified by the number of Floating Point Operations (FLOPs).

[5]Some existing algorithms only present the search cost as the computational metric for NAS methods. However, when considering a novel dataset, the supernet needs to be trained first. Therefore, we believe that incorporating the supernet training cost is essential to accurately assess the overall computational expense of the NAS algorithm.

5.5.1.1 Performance on CIFAR-10

The overall performance comparisons on CIFAR-10 are shown in Table 5.1. There are four peer competitors belonging to manually-designed methods, and the other 29 competitors belong to NAS methods. TIFNAS achieves an average error rate of 2.594%, which ranks fifth among all the 33 competitors. The error rate is very similar to RandomNAS's [169] 2.59% and LEMONADE's [35] 2.58%, but they have much larger model sizes; their consumed GPU-days are also much larger than TIFNAS. NPENAS-NP's [152] 2.54% is 0.054% lower than TIFNAS, but its model size is 1.56 times larger than that of TIFNAS, and the computational cost is 5.8 times larger than that of TIFNAS. Proxyless NAS [19] has the lowest error rate of 2.08%, which is 0.514% lower than TIFNAS, but its model size is 2.53 times larger than that of TIFNAS, and the computational cost is 4,839 times that of TIFNAS. TIFNAS (best) achieves an impressive error rate of 2.53%, ranking second over all competitors. Only Proxyless NAS [19] demonstrates superiror classification performance, but the model size is much larger. TIFNAS has 2.25M parameters, and only AECNN [138] and NSGANetV1-A1 [85] have fewer parameters. However, the classification error rates of them are 1.706% and 0.896% higher than that of TIFNAS. TIFNAS (best)'s model size is a mere 2.18M, which is smaller than the average. TIFNAS's average computational cost is only 0.31 GPU-days, which is the lowest among all the competitors, and is actually much less than others. The computational cost of TIFNAS across different trials is similar, and TIFNAS (best) only costs 0.3 GPU-days. It can be concluded that TIFNAS achieves very good classification results in terms of the error rate and the model size, and the computational cost is much lower than existing methods.

5.5.1.2 Performance on CIFAR-100

On the CIFAR-100 dataset, five manually-designed networks and 18 NAS methods are used for comparison with TIFNAS, as illustrated in Table 5.2.

Table 5.1: Performance comparison results on **CIFAR-10**.

Model	Error Rate	#Parameters	GPU-Days
FractalNet [70]	5.22%	38.6M	—
Maxout [43]	9.3%	—	—
DenseNet (k=24) [56]	3.74%	27.2M	—
Highway Network [129]	7.72%	—	—
CGP-CNN [132]	5.98%	2.64M	27
NAS [182]	6.01%	2.5M	22,400
Large-scale Evolution [106]	5.4%	5.4M	2,750
Block-QNN-S [174]	4.38%	6.1M	90
MetaQNN [13]	6.92%	—	100
EIGEN [107]	5.4%	2.6M	2
CNN-GA [140]	4.78%	2.9M	35
PNASNet-5 [73]	3.41%	3.2M	150
AmoebaNet-B [105]	2.98%	34.9M	3,150
EAS [18]	4.23%	23.4M	$<$10
NASNET-A [183]	2.97%	27.6M	2,000
AECNN [138]	4.3%	2.0M	27
DENSER [11]	5.87%	10.81M	—
GeNet from WRN [155]	5.39%	—	100
CoDeepNEAT [91]	7.3%	—	—
Hier. repr-n, evolution [74]	3.63%	—	300
NPENAS-NP [152]	2.54%	3.5M	1.8
SNAS [157]	2.85%	2.8M	1.5
ENAS [100]	2.89%	4.6M	0.5
EffPnet [150]	3.58%	2.68M	$<$3
DARTS [75]	2.82%	3.4M	1
NSGA-Net [84]	2.75%	3.3 M	4
LEMONADE [35]	2.58%	13.1M	90
NSGANetV1-A1 [85]	3.49%	**0.5M**	27
Proxyless NAS [19]	**2.08%**	5.7M	1,500
EAEPSO [164]	2.75%	3.17M	2.8
RandomNAS [169]	2.59%	3.1M	0.7
SI-EvoNet-S [167]	15.70%	3.32M	0.81
Modulenet [24]	2.77%	—	2.0
TIFNAS (best)	2.53%	2.18M	**0.3**
TIFNAS	2.594%$\pm$0.03%	2.25M$\pm$0.36M	0.31$\pm$0.02

Table 5.2: Performance comparison results on **CIFAR-100**.

Model	Error Rate	#Parameters	GPU-Days
FractalNet [70]	22.3%	38.6M	—
Maxout [43]	38.6%	—	—
DenseNet (k=40) [56]	17.2%	25.6M	—
Highway Network [129]	32.39%	—	—
SENet [54]	**15.41**%	34.4M	—
Large-scale Evolution [106]	23%	40.4M	2,730
Block-QNN-S [174]	20.65%	6.1M	90
MetaQNN [13]	27.14%	—	100
EIGEN [107]	21.9%	11.8M	5
CNN-GA [140]	20.03%	4.1M	40
EOFGA [163]	17.07%	1.96M	0.92
EAEPSO [164]	16.17%	5.42M	4
NSGA-Net [84]	20.74%	3.3M	8
PNASNet-5 [73]	19.53%	3.2M	150
ENAS [100]	19.43%	4.6M	0.5
AmoebaNet-A [105]	18.93%	3.1M	3,150
DARTS [75]	17.54%	3.4M	1
NSGANetV1-A1 [85]	19.23%	**0.7M**	27
AE-CNN+E2EPP [136]	22.02%	20.9M	10
EffPnet [150]	18.70%	—	—
AECNN [138]	22.40%	5.4M	36
SI-EvoNet-S [167]	15.70%	3.32M	0.81
Modulenet [24]	17.76%	—	—
TIFNAS (best)	16.56%	2.20M	0.3
TIFNAS	16.672%±0.13%	2.12M±0.15M	**0.28±0.02**

TIFNAS attains an average error rate of 16.672%, ranking fourth among the 22 competitors. Although SENet [54] exhibits an error rate that is 1.262% lower, its model size is over 16 times larger than that of TIFNAS. EAEPSO [164] and SI-EvoNet-S [167] also demonstrate lower error rates;

however, their model sizes and computational costs are greater than those of TIFNAS. TIFNAS (best) achieves an error rate of 16.56%. TIFNAS possesses the third smallest number of parameters, surpassed by NSGANetV1-A1 [85] and EOFGA [163], which exhibit fewer parameters but are both less accurate than TIFNAS. In terms of the computational cost, TIFNAS requires mere 0.28 GPU-days, which is lower than all its competitors. In conclusion, TIFNAS delivers promising performance on CIFAR-100, achieving high classification accuracy with a compact model in just 0.28 GPU-days.

5.5.1.3 Performance on ImageNet

In the ImageNet dataset, TIFNAS is compared with 15 peer competitors, including four manually-designed methods (shown in Table 5.3). TIFNAS achieves a Top-1 classification accuracy of 76.1%, securing the third position. While DenseNet-169 [56] and NSGANetV1-A3 [85] marginally outperform TIFNAS by 0.3% and 0.1% respectively, DenseNet-169's model size is 2.78 times larger and its computational complexity is 14 times greater than TIFNAS. NSGANetV1-A3 exhibits a 1.19 times higher computational complexity and 6.4 times larger search cost compared to TIFNAS. Regarding Top-5 accuracy, TIFNAS attains 93.0%, with only DenseNet-196 [56] surpassing it by 0.3%.

TIFNAS's model size, with 5.1M parameters, ranks fourth smallest. MobileNetV2 [115] and DARTS [75] have fewer parameters, but underperform in classification compared to TIFNAS. NSGANetV1-A3 has marginally fewer parameters (0.1M less) but a higher computational complexity than TIFNAS. TIFNAS records 477M FLOPs, with MobileNetV3 [52], EBNAS (large) [119], and EOFGA [163] demonstrating lower complexity; however, they all exhibit inferior classification accuracies. The total computational cost for TIFNAS, encompassing supernet training and the search process, is 4.2 GPU-days, which ranks fifth. Despite having lower computational costs, BanditNAS [168], SNAS [157], EBNAS (large), and DARTS all yield

markedly poorer classification performance than TIFNAS.

Table 5.3: Performance comparison results on **ImageNet**.

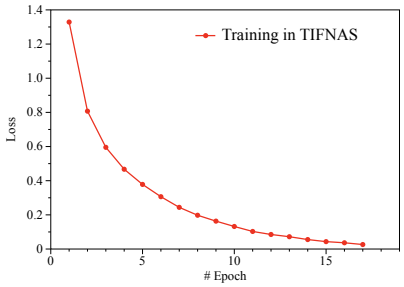
Model	Top-1 Acc.	Top-5 Acc.	#Params	#FLOPs	GPU-Days
ResNet-34 [48]	73.2%	91.3%	21.8M	7,360M	—
MobileNetV2 [115]	72.0%	91.0%	**3.4M**	600M	—
DenseNet-169 [56]	**76.4%**	**93.3%**	14.2M	6,740M	—
ShuffleNet [172]	73.7%	—	5.4M	524M	—
RL-NAS [171]	75.9%	—	—	561M	—
MobileNetV3 [52]	75.2%	—	5.4M	450M	—
PNASNet-5 [73]	74.2%	91.9%	5.1M	588M	150
Proxyless NAS [19]	75.1%	92.5%	7.1M	—	8.3
AmoebaNet-C [105]	75.7%	92.4%	6.4M	570M	3,150
BanditNAS [168]	75.3%	—	5.12M	547M	1.8
EOFGA [163]	75.6%	92.5%	5.7M	455M	8.0
SNAS [157]	72.7%	90.8%	4.3M	—	1.5
NSGANetV1-A3 [85]	76.2%	93.0%	5.0M	570M	27
EBNAS (large) [119]	67.8%	87.4%	—	**128M**	**0.04**
DARTS [75]	73.3%	91.4%	4.7M	—	4
TIFNAS	76.1%	93.0%	5.1M	477M	4.2

5.5.2 Analysis of the Supernet Training

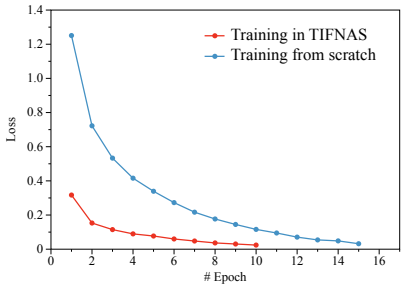
In TIFNAS, supernet training comprises two processes: training the three paths with the largest width and training all possible paths. In the first process, the three paths are trained sequentially, with the two paths of smaller kernel sizes inheriting weights prior to training. In the second process, all operations' initial weights are inherited from the three trained paths instead of randomly initialized. This section presents an analysis of these training processes on CIFAR-10.

Figure 5.7 depicts the training process for the three individual paths, as well as the training processes for the two 'smaller' paths without weight

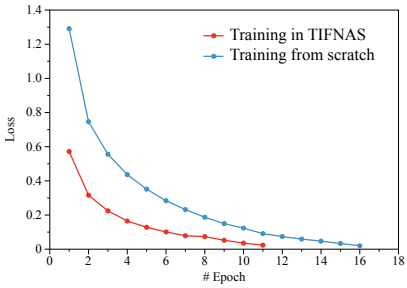
inheritance in (b) and (c). For $Path$ ($k = 7$; $e = 6$), the training loss consistently decreases throughout the training process and converges after 17 epochs. For $Path$ ($k = 5$; $e = 6$), the initial training loss using the TIFNAS method is significantly lower than that of training from scratch. Moreover, training ceases at the 10th epoch, which is notably earlier than the 15 epochs by training without weight inheritance. A similar trend is observed for $Path$ ($k = 3$; $e = 6$). These findings demonstrate the effectiveness of the weight initialization of TIFNAS in substantially reducing the number of training epochs and the computational cost.



(a) Training loss for *Path* (*k* = 7, *e* = 6)



(b) Training loss for *Path* (*k* = 5, *e* = 6)



(c) Training loss for *Path* (*k* = 3, *e* = 6)

Figure 5.7: The supernet training processes of the three widest paths on CIFAR-10. The X-axis represents the number of training epochs, and the Y-axis represents the training loss.

Figure 5.8 illustrates the training process for all possible paths within the supernet using the TIFNAS method over 100 epochs, as well as the

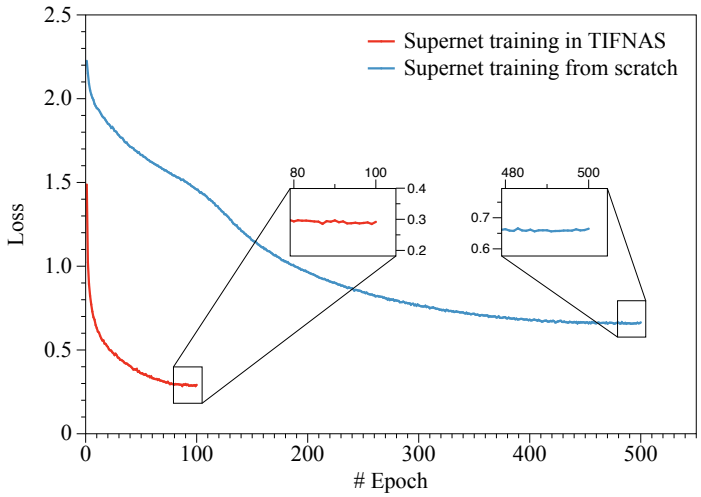


Figure 5.8: The supernet training processes of sampling possible paths to train the supernet on CIFAR-10. The X-axis represents the number of training epochs, and the Y-axis represents the training loss.

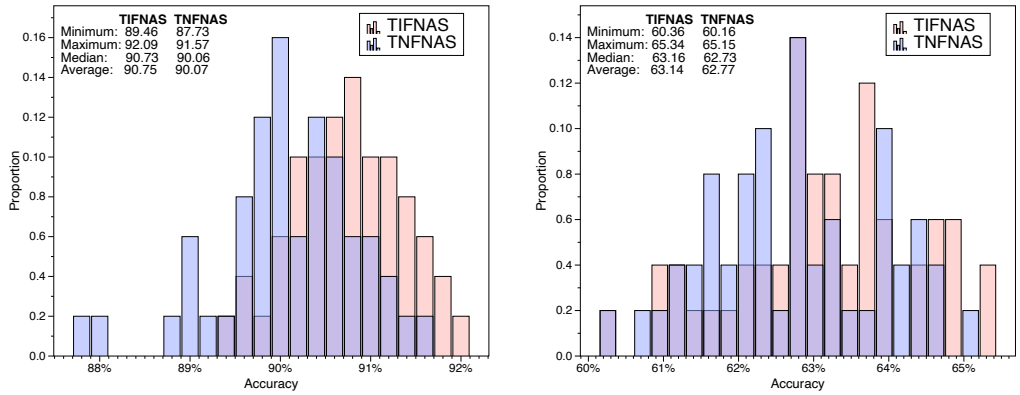
training process for training from scratch over 500 epochs. Evidently, the training loss for TIFNAS decreases much more rapidly than that of training from scratch, with the final training loss reaching approximately 0.3, compared to 0.65 for training from scratch. This indicates that the proposed training method can achieve superior training results with significantly reduced computational costs. Employing a single A6000 GPU card, the total supernet training time for TIFNAS on CIFAR-10 is approximately 175 minutes, while the training time for training from scratch over 500 epochs is around 625 minutes. Consequently, TIFNAS utilizes 28% of the training cost to achieve markedly improved training outcomes compared to the vanilla training method.

5.5.3 Effectiveness of Population Initialization

To demonstrate the effectiveness of the proposed population initialization method, comparative experiments are designed by using an algorithm

called TNFNAS. This algorithm has identical settings to TIFNAS, with the exception of employing a random population initialization method.

5.5.3.1 Analysis on the Initial Population Performance



(a) Performance on CIFAR-10

(b) Performance on CIFAR-100

Figure 5.9: The distribution of fitness for the initial population, obtained through two different initialization methods, on two image classification datasets: CIFAR-10 and CIFAR-100.

The performance of the initial population is evaluated for both TIFNAS and TNFNAS. During the evaluation, the candidates are trained from scratch to convergence, and then evaluated to obtain the classification accuracies as their fitness[6]. Figure 5.9 displays the initial population's performance distribution for TIFNAS and TNFNAS. On CIFAR-10, the average accuracy for TIFNAS is 90.75%, while it is 90.07% for TNFNAS. The minimum, maximum, and median accuracies of TIFNAS are all higher than those of TNFNAS. On CIFAR-100, the distribution of individuals of TIFNAS

[6]In the training, no data augmentation techniques are used, a simple fixed learning rate policy is employed, and the weights are randomly initialized without using weight inheritance, filtering the impact of the supernet. The training stops when the training loss does not decrease for five epochs. The achieved classification accuracy is able to represent the candidate's performance.

is more concentrated on higher accuracies compared to that of TNFNAS. Furthermore, the number of individuals with higher accuracies in TIFNAS is greater than that of TNFNAS, and these promising individuals may contain beneficial genes that facilitate the subsequent search. The probability distribution results confirm that the proposed initialization produces a higher-quality initial population.

5.5.3.2 Overall Comparasions

Table 5.4 shows the comparison results of TIFNAS and TNFNAS on CIFAR-10 and CIFAR-100. Overall, TIFNAS achieves lower error rates on both datasets compared to TNFNAS. The average error rate of TIFNAS is 0.114% and 0.485% lower than that of TNFNAS on CIFAR-10 and CIFAR-100, respectively. We used the Mann-Whitney U test to compare the performance of TIFNAS and TNFNAS: On CIFAR-10, the test reveals a significant difference between them ($U_{statistic}$ = 0, p = 0.0079); similarly, on CIFAR-100, the test also indicates a significant difference ($U_{statistic}$ = 0, p = 0.02857). The average numbers of parameters of TIFNAS on both datasets are slightly larger than TNFNAS. Regarding the computational cost, TIFNAS consumes 0.02 and 0.01 more GPU-days than TNFNAS on CIFAR-10 and CIFAR-100, respectively. The difference is very small, indicating that the proposed initialization method introduces little additional computation.

Table 5.4: The comparisons of TIFNAS and TNFNAS.

Method	Dataset	Error Rate (%)	#Params	GPU-Days
TIFNAS	CIFAR-10	2.594±0.03	2.25M±0.36M	0.31±0.02
	CIFAR-100	16.672±0.13	2.12M±0.15M	0.28±0.02
TNFNAS	CIFAR-10	2.708±0.03	2.01M±0.09M	0.29±0.02
	CIFAR-100	17.157±0.10	1.97M±0.22M	0.27±0.02

5.5.4 Effectiveness of Supernet Fine-Tuning

To test the effect of the supernet fine-tuning, we remove this step from TIFNAS, forming a method called TINNAS.

5.5.4.1 Overall Comparasions

Table 5.5 shows the overall performance of TINNAS and TIFNAS on the two datasets. The average accuracy of TIFNAS is 0.243% and 0.863% higher than that of TINNAS on CIFAR-10 and CIFAR-100, respectively. Mann-Whitney tests are conducted to examine the difference between the classification error rates of TINNAS and TIFNAS on the two datasets, and the p-values for the two datasets were both less than the commonly accepted threshold of 0.05, suggesting a significant difference between TINNAS and TIFNAS in terms of the classification error rates. There is not much difference between the number of parameters of the two methods. As for the computational cost, TINNAS consumes much fewer GPU-days compared with TIFNAS, indicating that supernet fine-tuning does require some additional computations, which is about 0.09 GPU-days on both datasets. However, considering the difference in error rates, the additional computational cost is acceptable and worthwhile.

Table 5.5: The comparisons of TIFNAS and TINNAS.

Method	Dataset	Error Rate (%)	#Params	GPU-Days
TIFNAS	CIFAR-10	2.594±0.03	2.25M±0.36M	0.31±0.02
	CIFAR-100	16.672±0.13	2.12M±0.15M	0.28±0.02
TINNAS	CIFAR-10	2.837±0.02	2.31M±0.20M	0.22±0.01
	CIFAR-100	17.535±0.15	2.16M±0.25M	0.19±0.01

5.5.4.2 Analysis on the Evaluation Reliability

The purpose of supernet fine-tuning is to provide more reliable fitness evaluations, and the evaluated fitness is expected to be similar to the true fitness, which is obtained through training from scratch until the training loss does not decrease. The Spearman Correlation Coefficient is used to measure the correlation between the predicted fitness and the true fitness.

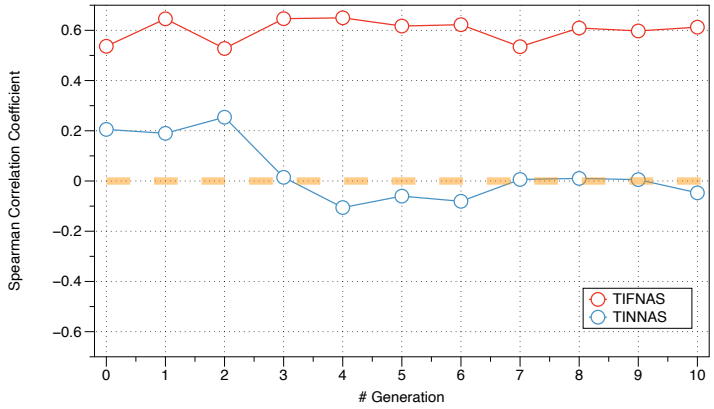


Figure 5.10: Spearman Correlation Coefficient between measured and true fitness throughout evolution on CIFAR-10.

For the population of each generation, the Spearman Correlation Coefficient of the predicted fitness and the true fitness was calculated for both TIFNAS and TINNAS. As evaluating the true fitness is very time-consuming, the experiment is only conducted on CIFAR-10 once, and the results are shown in Figure 5.10. It can be seen that the Spearman Correlation Coefficient of TIFNAS fluctuates around 0.6, indicating a strong correlation between the predicted fitness and the true fitness, and this strong correlation does not deteriorate over the course of evolution. In contrast, the Spearman Correlation Coefficient of TINNAS is around 0.2 for the first three generations and becomes approximately 0 for the following generations, suggesting a weak correlation for the first three generations and almost no correlation for the subsequent generations. These

results demonstrate that the proposed supernet fine-tuning strategy can significantly improve the reliability of the predicted fitness, and this reliability is stable and does not degrade during the evolution process.

5.6 Chapter Summary

This chapter presents an efficient one-shot evolutionary neural architecture search method, which has been achieved by developing an effective supernet training strategy, introducing a robust population initialization method, and devising a supernet fine-tuning approach to providing more precise fitness evaluations. The proposed supernet training method leverages the unique structure of the supernet and the powerful weight inheritance to accelerate the weight training process, thus reducing the overall computational cost. Moreover, the population initialization method supplies high-quality individuals for the initial population by utilizing the information gleaned from the previous supernet training process. Furthermore, the supernet weights are fine-tuned along with the evolutionary search according to the individuals to be evaluated, which requires minimal additional time but yields significantly more accurate evaluations.

The proposed method, TIFNAS, is evaluated on three widely-used datasets, and is compared with 67 peer competitors. TIFNAS achieves low error rates with compact models, and the computational costs are only 0.31, 0.28, and 4.1 GPU-days on CIFAR-10, CIFAR-100, and ImageNet, respectively, which are substantially lower than most competitors. Further analysis demonstrates the efficacy of the new training strategy, the proposed population initialization method, and the fine-tuning strategy.

While the proposed method can search for promising network architectures effectively, explaining why and how the searched architecture makes decisions is challenging. The next chapter will develop a new method to improve the interpretability of the architecture.

Chapter 6

Explaining Image Classification Using Evolutionary Search and the Stable Diffusion Model

6.1 Introduction

In contrast to Chapters 3, 4, and 5, which primarily focus on proposing innovative CNN architecture search methods, this chapter delves into explaining the decision-making processes underlying CNNs. As the complexity of CNN models has escalated, their interpretability has dwindled. Explainability, however, enables domain experts and developers to gain valuable insights and comprehend the underlying factors that influence the model's predictions. Such an understanding may lead to new discoveries and enhance decision-making. Moreover, understanding how deep models arrive at their decisions can help build trust in their outputs, which is pivotal in industries such as finance and healthcare. Furthermore, explainable models allow developers to diagnose and address issues more effectively, as explainability can help reveal the underlying causes of incorrect decisions or predictions.

Many methods aim to explain the rationale behind classifiers by pin-

pointing the salient areas [125,128,144,166]. However, their primary focus remains on discerning '*where*' the influential regions are situated, rather than explicating '*how*' these regions steer the results. This approach inadvertently curtails the depth of explainability. To alleviate this limitation, this chapter incorporates a generative model to generate counterfactuals. Such an approach allows one to identify not only 'where' the pivotal regions are but also 'how' their content changes, thereby granting a more interpretably understanding of CNNs' decision-making processes. Additionally, this proposed method possesses the capability to explain decisions on similar image classes. These classes, inherently nuanced and intricate, often pose greater challenges for the classifiers to classify. In this case, finding out how the classifiers make their predictions is crucial, and explaining the decisions also becomes much more difficult.

6.1.1 Chapter Goals

The overall goal of the chapter is to propose a novel post-hoc, model-agnostic method for explaining image classifiers, which utilizes a Stable Diffusion (SD) model to generate counterfactual images and a Multi-Objective Evolutionary Algorithm (MOEA) to identify crucial superpixels or attributes. To achieve this goal, four objectives are described as follows:

1. To develop a method that combines an SD model for generating counterfactual images for explanation purposes. Instead of directly using the SD model to generate the whole image, this method paints specific regions of an image while adhering to certain semantic information. The generated images are usually natural and circumvent the sharp edges that often result from directly masking specific regions.

2. To propose an EMO method based method to automatically identify crucial and minimal number of superpixels, which are essential for explaining the decisions of classifiers. This method should

not only identify superpixels with a significant impact on the output classification probability but also optimize the number of superpixels. Fewer superpixels contain fewer attributes, thereby simplifying the explanation process.

3. To propose a new evaluation metric that assesses the salience of the selected superpixels for classifications. This evaluation can represent the impact of the superpixels on the classifier's ability to categorize the image into a specific category rather than another similar category.

4. To validate the proposed method on extensive experiments, including several similar image categories and several classic deep learning models.

6.1.2 Chapter Organization

Subsequent sections unfold as: Section 6.2 elucidates the general framework and provides detailed information about the proposed algorithm; Section 6.3 details the experimental settings and provides details about the implementation; Section 6.4 delivers the experimental results along with their corresponding analysis; and Section 6.5 presents the conclusions.

6.2 The Proposed Method

This section delineates the framework and essential details of the proposed algorithm, termed SD-MOEX, an acronym for Stable Diffusion assisted Multi-Objective EXplaination method.

6.2.1 Algorithm Overview

Figure 6.1 outlines the overall framework of SD-MOEX. The main pipeline, illustrated on the left, begins with an input image, which is then seg-

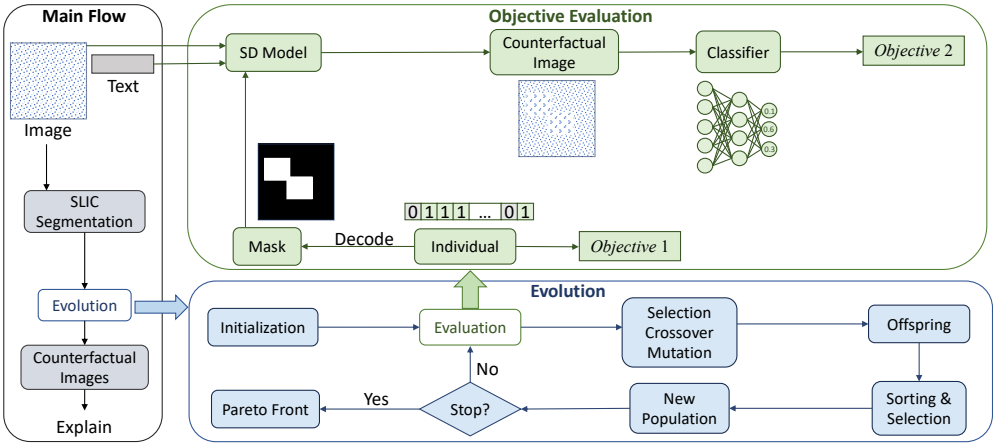


Figure 6.1: Overview of the SD-MOEX framework: the primary workflow is illustrated on the left. The bottom part displays the evolutionary process, following a typical NSGA-II procedure. The evaluation process is depicted at the top, where two objective values are calculated.

mented into superpixels using the SLIC method. An MOEA is deployed to yield non-dominated solutions, pinpointing pivotal superpixels or features in the image. Subsequently, the generated counterfactual images corresponding to the most favorable solutions will be output and provided to humans to explain the decision of the classifier (see Section 6.2.7). Please note that SD-MOEX is designed to explain classifications between closely related and similar classes, thus requiring a textual description of the comparable class, so that the image generator can produce counterfactuals considering the features of the comparable class. For instance, given an input image of a *tabby cat*, a corresponding textual description could be *Egyptian Mau cat*. SD-MOEX can explain why the classifier leans towards a *tabby cat* classification rather than an *Egyptian Mau cat*.

The evolutionary procedure is illustrated at the bottom of Figure 6.1. Each individual represents a subset of the superpixels (refer to Section 6.2.2), and a population of individuals are initialized first, as shown in Section 6.2.3. Then, the objectives of these individuals are evaluated us-

ing the proposed objective evaluation method, depicted at the top of the figure. Following this, specific individuals are selected, and the crossover and mutation operations are performed to generate an offspring population (see Section 6.2.6). The existing and new populations are merged into a combined population, where all individuals are sorted following the principle of NSGA-II, and the high-performing and less crowded individuals are selected to form the new population. This evolutionary process continues until the stopping criterion is met. The non-dominated solutions in the Pareto front are the output.

The evaluation step begins with decoding an individual to its corresponding mask (see Section 6.2.2). Simultaneously, the number of superpixels within the individual is calculated and used as $Objective$ 1. The SD model is then invoked to inpaint within the input image, utilizing the mask to pinpoint the inpainting regions, and the text description of the similar category to ascertain the inpainting content (see Section 6.2.4). Subsequently, a counterfactual image is generated, and the generated image is tested by the classifier. The second objective measure, $Objective$ 2, quantifying the chosen superpixels' impact on the classifier, is calculated based on the classifier's outputs (see Section 6.2.5).

6.2.2 Encoding and Decoding Strategies

In SD-MOEX, each individual/solution in the population represents a subset of the superpixels. To achieve this, we designed straightforward encoding and decoding strategies. An input image is segmented into N superpixels using the Simple Linear Iterative Clustering (SLIC) method, where N is manually specified. Every superpixel gets a distinctive integer tag from 1 through N. Figure 6.2 (a) offers an example where the input image is dissected into ten superpixels.

A simple binary encoding strategy is employed, wherein the encoded representation is an N-length string, equaling the number of superpixels.

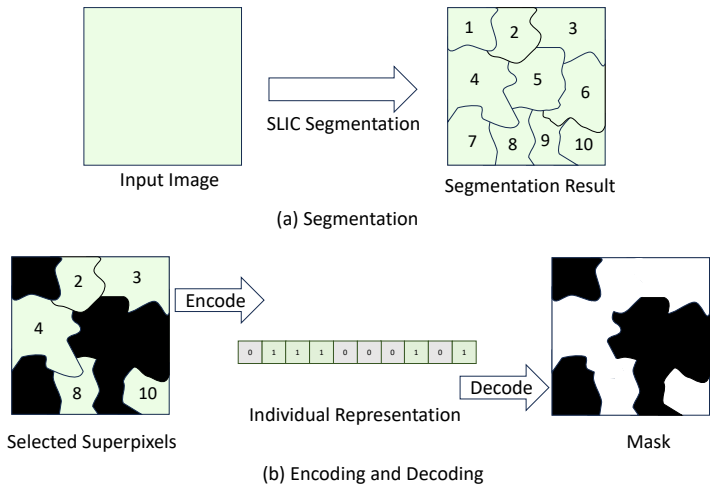


Figure 6.2: Illustration of the SLIC segmentation and the encoding-decoding process. (a) The input image is segmented into 10 superpixels via SLIC. (b) Five superpixels are selected and encoded into a string comprised of binary bit representations. Subsequently, the string is decoded into a binary mask, wherein the selected superpixels are represented in white, and the remaining portions of the image are depicted in black.

A '1' denotes the inclusion of a superpixel, while a '0' signifies its absence. Figure 6.2 (b) illustrates this, with the second, third, fourth, eighth, and tenth superpixels selected and their corresponding positions in the string set to '1'.

During the evaluation process, each individual is decoded into a mask, which is then fed to the SD model, delineating inpainting areas. The decoding process translates the selected superpixels into a white area and retains other segments as black, indicating that the corresponding content remains unchanged. Figure 6.2 (b) further clarifies this decoding process, where the individual is decoded to a mask consisting of black and white.

6.2.3 Population Initialization

The initial population is constructed with a predetermined number of individuals. As detailed in Section 6.2.2, every individual in the population corresponds to a binary string, where each bit denotes the inclusion (1) or exclusion (0) of a superpixel. The process of initializing each individual is fairly straightforward:

1) For each position in the binary string, a random binary digit (either 0 or 1) is chosen. This decision determines whether the superpixel corresponding to that position is active or not in the individual.

2) The above step is carried out iteratively until the entire length of the string matches the total number of superpixels for the given image.

3) This procedure is repeated for each individual until the desired population size is achieved.

6.2.4 Counterfactual Image Generation

To evaluate the impact of the superpixels on each individual, a counterfactual image is generated. The SD model is utilized to produce natural-looking and realistic counterfactual images. This model enables the inpainting of specific regions in an input image, guided by a textual description, termed the *prompt*. Within SD-MOEX, the mask image, decoded from the individual, denotes the region for inpainting. Simultaneously, a textual description of a related and similar category guides the inpainting content.

Figure 6.3 presents an example of the SD model's inpainting process. Three distinct inputs are required: the source image (the input image in SD-MOEX), the mask image (decoded from the individual in SD-MOEX), and the textual description (the description of the similar category in SD-MOEX). In this instance, the source image is a photograph of a cat, the

mask image indicates regions to inpaint, particularly around the eyes, and the textual description, *A picture of a dog*, suggests that the inpainted areas should adopt dog-like features. Given that the inpainting is constrained to the eye locations, the generated counterfactual image portrays a cat possessing the eyes of a dog. The image quality is high, with no discernible edges between the inpainted content and the surrounding area. The resulting image appears realistic and natural, although the content does not exist in reality.

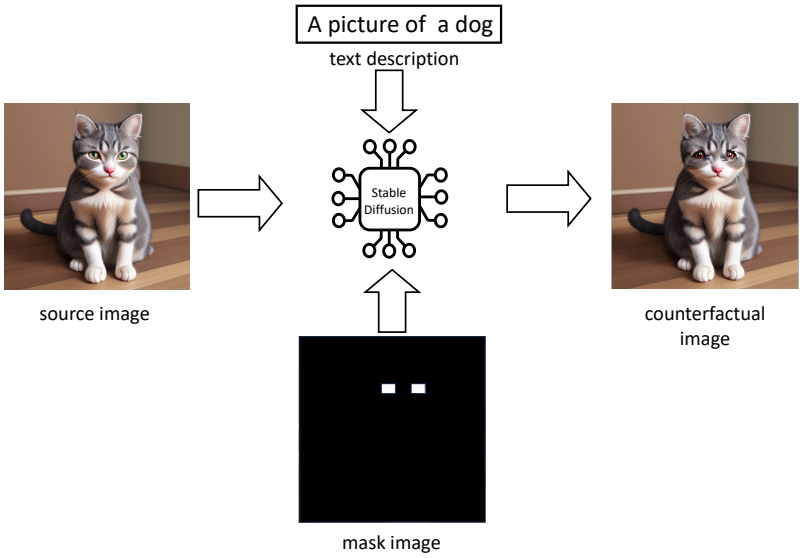


Figure 6.3: A example of the SD model's inpainting. Inputs comprise a source image of a cat, a mask image pinpointing inpainting regions, and a textual description specifying the desired inpainting content. The model synthesizes a counterfactual image based on the input information.

From the aforementioned example, several key strengths of the SD model in counterfactual image generation emerge:

1. The generated image boasts high quality, preserving pixel distribution consistency with real images.

2. The inpainted content integrates seamlessly with the surrounding

content, devoid of noticeable boundaries that could potentially impact classification and adversely affect the explanation.

3. The textual description guides the inpainting, ensuring the newly generated eyes possess 'dog' attributes. This capability allows for the analysis of the impact of 'eye' features on classifiers when telling whether an image represents a *cat* or a *dog*.

6.2.5 Objective Functions

SD-MOEX is designed to pinpoint crucial superpixels, employing the resulting counterfactual images to explain the classifier's causal reasoning in classification tasks. The evolved set of superpixels is expected to be compact and significant. Consequently, each individual is evaluated based on two corresponding objective functions.

6.2.5.1 Objective 1: Number of superpixels

The first objective is to minimize the number of the selected superpixels. Generally, fewer superpixels correspond to smaller regions [7]. Reduced regions typically represent fewer and more straightforward features; thus, a smaller number of superpixels can lead to fewer features and facilitate the interpretation. For instance, the region of a cat's eyes corresponds to a smaller number of superpixels and the features of *eyes*; conversely, a region comprising a cat's face corresponds to a larger number of superpixels and integrates multiple features like *eyes*, *nose*, *whiskers*, *mouth*, *coat color*, and *pattern*. It subsequently becomes more challenging to discern which feature(s) significantly influence the decision, thereby impeding the explainability.

[7] Technically, fewer superpixels might not necessarily lead to smaller regions, given the variable areas of different superpixels. It is possible for a set of superpixels to include fewer superpixels but correspond to a larger area. Nonetheless, this scenario is uncommon, and fewer superpixels roughly equate to smaller areas in most cases.

6.2.5.2 Objective 2: Impact of Superpixels

The second objective is to maximize the impact of the superpixel ensemble. Specifically, this objective value quantifies how drastically the prediction results would change towards another category if the corresponding superpixels are replaced by features from another category. This insight aids in explaining why the classifier designates an image into a specific category over alternatives. A novel evaluation method is proposed to quantify this impact.

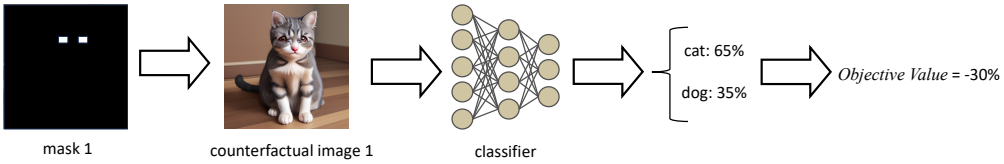
The synthesized counterfactual image is processed by the target classifier, yielding classification outcomes. The output from the last layer of neurons is a vector containing the raw classification values for each class, as illustrated in Equation (6.1), where $\vec{z}$ denotes the output vector, and z_i represents the i-th neuron's output. The Softmax function [58] is subsequently employed to calculate the predicted probabilities of belonging to different neurons/classes, following Equation (6.2), where K is the total number of neurons/classes. The objective value quantifying the impact is computed according to Equation (6.3), where $\sigma(\vec{z})_{C_{img}}$ is the input image's class predicted probability, and $\sigma(\vec{z})_{C_{text}}$ corresponds to the textual description's class predicted probability. Thus, if the counterfactual image is more likely to be classified into the text class, i.e., the inpainted class, the objective value escalates, suggesting that the selected superpixels are crucial for the classifier to categorize the image into the true class instead of the similar one.

$$\vec{z} = [z_0, z_1, ..., z_k] \tag{6.1}$$

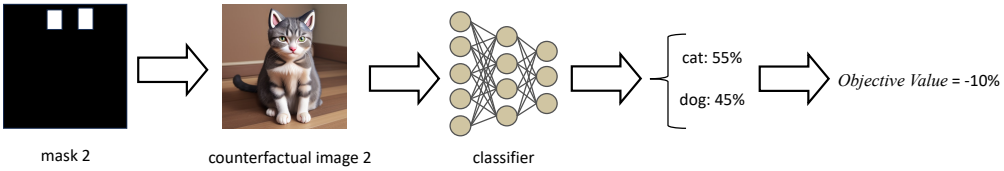
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{6.2}$$

$$Objective_{imp} = \sigma(\vec{z})_{C_{img}} - \sigma(\vec{z})_{C_{text}} \tag{6.3}$$

Figure 6.4 presents an example of the objective value calculation for impact. The first individual corresponds to a mask emphasizing the eye



(a) The impact objective value of individual 1



(b) The impact objective value of individual 2

Figure 6.4: Illustrations of impact objective values. (a) represents an individual corresponding to the eye region's superpixels. Conversely, (b) depicts an individual associated with the ear region's superpixels. The calculations imply that the impact objective value in (b) exceeds that in (a), suggesting that the 'ear' attribute assumes greater importance than the 'eye' feature for this classifier's decision-making paradigm.

regions, while the second one spotlights ear regions. Consequently, the former counterfactual image showcases a cat with canine eyes, and the latter showcases a cat with canine ears. Upon classifying these counterfactual images, the classifier determines the probabilities for each category. The latter counterfactual image exhibits a stronger inclination towards the dog classification than the former, with its impact objective value surpassing the former by 20%. This indicates that the 'ear' feature is more crucial than the 'eye' feature for this classifier to categorize an image as a *cat* rather than a *dog*.

6.2.6 Offspring Generation

The representation of individuals in SD-MOEX is relatively straightforward (refer to Section 6.2.2), and the commonly used offspring generation method is adopted. Parent individuals are chosen from the existing population (detailed information can be found in Section 2.4.3.1). Subsequently, the two-point crossover method is utilized as the mating procedure, and the flip-bit mutation operation is introduced to enrich the diversity of the population.

Specifically, the two-point crossover technique randomly chooses two points in the parent individuals and swaps the genes between these points in the parents to generate new offspring. This technique fosters exploration of the search space and facilitates the exchange of advantageous genetic information between individuals. The flip-bit mutation operator functions on a granular level, flipping each bit of the individual independently with a probability dictated by the mutation rate. This mutation operator introduces variation into the population and prevents the algorithm from prematurely converging at local optima.

6.2.7 Explanations

The evolutionary process aims to identify a set of non-dominated solutions, facilitating subsequent explanations. Each individual corresponds to a group of superpixels, highlighting salient image regions. However, it does not directly indicate 'how' these regions affect the decision. Instead of relying solely on the 'best' superpixel sets to explain the classifiers directly, SD-MOEX collects the synthesized counterfactual images corresponding to these 'best' individuals during the evaluation phase. The classification outcomes of these counterfactuals are concurrently presented, laying the foundation for comprehensive explanations. In order to provide a more comprehensive explanation, SD-MOEX selects three representative counterfactuals based on the number of superpixels in the individual: the

one with the fewest superpixels, the median, and the one with the most superpixels. This allows for showcasing influential features at different levels.

Through this approach, users can identify what features of the input image are altered, how they are changed, and how these alterations impact the classifier's decision. Thus, the two crucial questions — 'what' regions are essential and 'how' these regions can affect the decision — can be addressed in a more effective way.

6.3 Experimental Design

6.3.1 Benchmark Dataset

The experiments employ the ImageNet dataset [112] to assess the efficacy of SD-MOEX. The detailed information on ImageNet is introduced in Section 3.3.1 (see page 84). The selection of the ImageNet dataset follows three primary reasons:

1. ImageNet consists of 1000 categories, many of which are closely related and challenging to distinguish, such as the *tabby cat* versus the *Egyptian Mau cat*, and the *electric guitar* in comparison to the *acoustic guitar*. Given SD-MOEX's objective to explain decisions on similar classes, these closely related classes make it aptly suited.

2. ImageNet stands as a renowned dataset for image classification. Numerous celebrated classifiers have been trained and evaluated on ImageNet, facilitating the use of SD-MOEX in explaining their decision-making processes.

3. Several renowned explainable artificial intelligence techniques have been assessed using ImageNet, providing a conducive environment to compare SD-MOEX with other methodologies.

6.3.2 Selected Image Categories and Models

To evaluate the efficacy of SD-MOEX, specific similar image categories from the ImageNet dataset are selected. Given its model-agnostic nature, SD-MOEX can elucidate the decisions of various image classification models, and some models need to be selected in the experiments.

classes: We focus on four different classification scenarios: *Egyptian Mau cat* versus *tabby cat*, *electric guitar* versus *acoustic guitar*, *yawl* versus *schooner*, and *ambulance* versus *police van*. The *Egyptian Mau cat* and the *tabby cat* exhibit considerable morphological similarities, especially concerning facial structure and body proportions. Nevertheless, nuanced variances, such as coat shades or pattern distributions, render their distinction intricate. Similarly, the *electric* and *acoustic guitars*, despite sharing similar body contours, fret layouts, and design aesthetics, have subtle differentiators like the sound hole design and electronic components. The *yawl* and the *schooner* are both sailing vessels and share structural similarities in terms of hull design and sail arrangement. However, the number of masts and sails is different. The *ambulance* and the *police van* are specialized vehicles, but their texts on the bodies differ.

Relying solely on these superficial traits for classification proves chal-

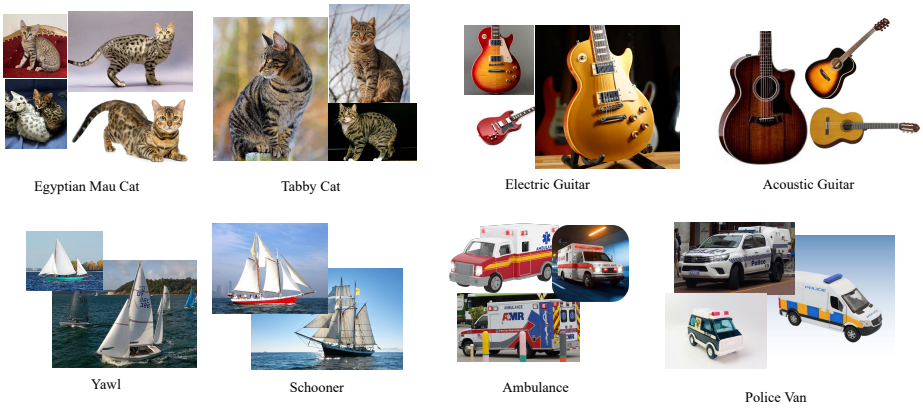


Figure 6.5: Some example cases of the classes.

lenging, highlighting their suitability for SD-MOEX exploration. Furthermore, these minor differences underscore the intricacy involved in explaining why classifiers assign an image to a specific category, not another. Figure 6.5 provides some example pictures of the similar classes.

Classification Models: We selected two image classification models for the evaluation of SD-MOEX: MobileNetV2 [115] and Wide ResNet-50 [165]. These models were chosen due to their different structural designs and computational complexities. Specifically, ResNet-50 employs a deep residual framework; on the contrary, MobileNetV2 is a lightweight, efficient network designed for mobile computing. In addition, they also achieve different classification accuracies on ImageNet, demonstrating their different capabilities for feature extraction.

It is important to emphasize that SD-MOEX is designed to be a model-agnostic method compatible with any classification model, including those searched in previous chapters of this thesis. Because MobileNetV2 and Wide ResNet-50 are two representative deep learning models, if SD-MOEX explains the decisions of both models, it will underscore its proficiency as a model interpreter with robust generalization capabilities. As a result, SD-MOEX has not been further applied to the networks searched in Chapters 3, 4, and 5.

6.3.3 Parameter Settings

We present the parameter settings of SD-MOEX across three domains: image segmentation, evolution, and image synthesis. Specifically, a critical parameter in SLIC segmentation is the number of superpixels. In SD-MOEX, this is set to 100, accounting for the dimensions of the input image and computational capacity. A larger number results in finer segmentation, increases the dimension of individual representations, and consequently augments the computational overhead. For the evolution, we set both the population size and the number of generations to 50. Typically,

larger values for these parameters yield improved results but also increase the computational expense. The crossover and mutation rates are fixed at 0.5 and 0.2, respectively. For the SD model to produce counterfactual images, specific parameters must be established. In alignment with practices found in [110], the guidance scale is 7.5, and the strength is set at 0.75.

6.4 Results and Analysis

This section delineates the experimental results of SD-MOEX, followed by in-depth analyses and discussions. Initially, the evolved counterfactual explanations are detailed in Section 6.4.1. Subsequently, a comparative analysis of SD-MOEX's counterfactual explanations with other explanation techniques is provided. Lastly, an examination of the evolution process and convergence scenario of SD-MOEX is furnished in Section 6.4.3.

6.4.1 Counterfactual Explanations

In this section, we examine SD-MOEX's explainability on four closely related image classification tasks: the *Egyptian Mau cat* versus the *tabby cat*, the *electric guitar* versus the *acoustic guitar*, the *yawl* versus the *schooner*, and the *ambulance* versus the *police van*. Given that our proposed SD-MOEX is a model-agnostic explanation method, we have chosen two prominent CNN models for examination: MobileNetV2 [115] and Wide ResNet-50 [165]. The counterfactuals corresponding to the individuals on the Pareto front can be used for explanations. Considering the practical situation, it is impossible to select all the counterfactuals corresponding to the front, and there is a great deal of overlap among some counterfactuals. Thus, it is not necessary to select all the counterfactuals for explanations. In the experiments, based on the number of superpixels corresponding to an individual, we selected three representative counterfactuals with potentially small overlaps: the smallest, the centered, and the largest number of su-

perpixels, respectively.

6.4.1.1 Egyptian Mau cat & Tabby cat

Bronze *Egyptian Mau cats* bear a striking resemblance to *tabby cats*. SD-MOEX aims to clarify why a classifier would categorize a given image as an *Egyptian Mau cat* rather than a *tabby cat*. SD-MOEX generates a series of counterfactual images accompanied by their respective classification probabilities. Three counterfactuals are selected based on their numbers of chosen superpixels to furnish detailed explanations for both MobileNetV2 and Wide ResNet-50.

Explanations for MobileNetV2:

The evolved counterfactual explanations for MobileNetV2 are presented in Figure 6.6. Specifically, in Counterfactual (a), the modified area is relatively minor, focusing on the body of the *Egyptian Mau cat*. In the original image, the skin pattern near the neck appears spotted; however, in the counterfactual, this pattern transitions to stripes. This transformation is mirrored on the lower portion of the cat's body. As a result, the classification probability for *Egyptian Mau cat* drops from 95.92% to 62.28%, while the *tabby cat* likelihood rises from 1.44% to 28.92%. Although MobileNetV2 continues to identify the counterfactual image as an *Egyptian Mau cat*, its confidence is significantly diminished. This suggests that the body pattern plays a pivotal role in MobileNetV2's decision on whether the subject is an *Egyptian Mau cat* or a *tabby cat*.

In counterfactual image (b), a more extensive modification is made compared to counterfactual image (a), especially on the front chest. In the original image, the pattern on the chest is finely speckled, but this shifts to distinct black stripes interspersed with brown and white fur colors in the counterfactual. Additionally, patterns on the back and the lower part of the body transition from black spots to strips. Relative to the original, the prediction probability for the *Egyptian Mau cat* declines from 95.92% to 59.26%, while the prediction probability for the *tabby cat* rises by 33.16% to

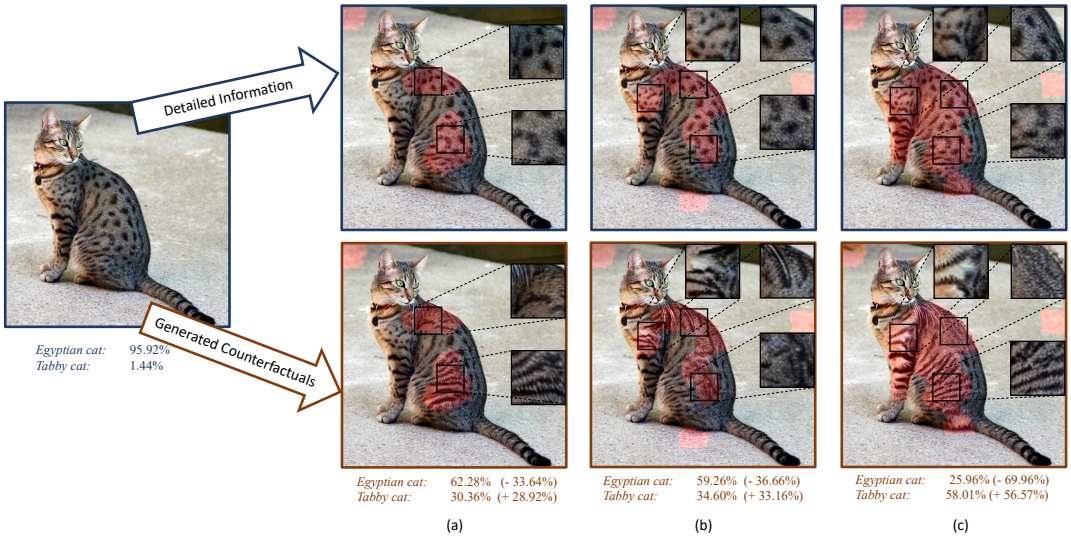


Figure 6.6: Counterfactual explanations for an *Egyptian Mau cat* image using MobileNetV2. The original image is displayed on the left. Details of selected superpixels from the original are presented above, with three variations of selected regions. The synthesized counterfactual images corresponding to these areas are displayed below. Enlarged views of altered regions facilitate comparison between the original (above) and counterfactual images (below). For each image, classification probabilities for both categories are provided, alongside changes in probabilities relative to the original image for each counterfactual image.

34.60%. This suggests that a spotted pattern in a cat's body tends to favor the classification of *Egyptian Mau cat*, while a striped pattern leans towards the identification of a *tabby cat*.

In the final counterfactual (c), modifications are made across almost the entire body. The spotted pattern characteristic of the original is eradicated in the counterfactual, replaced entirely by a striped pattern. It's noteworthy that the pinstriped pattern on the cat's abdomen in the original remains untouched, as it wasn't selected for modification. Similarly, the tail's black striped pattern in the original remains consistent, also escaping modification. Given these alterations, MobileNetV2's prediction probability for *Egyptian Mau cat* plunges to 25.96%, whereas the prediction

probability for *tabby cat* rises to 58.01%. Consequently, the final classification for this counterfactual shifts from *Egyptian Mau cat* to *tabby cat*. This counterfactual underscores that features such as the face, claws, and tail don't play a pivotal role in differentiating between these two cat breeds. Instead, it's the body pattern that emerges as the decisive factor in categorizing them.

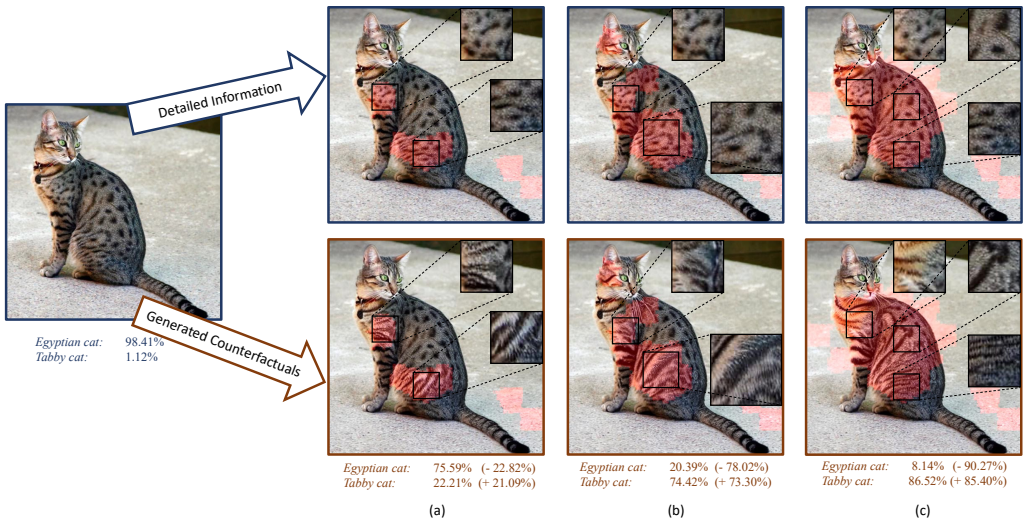


Figure 6.7: Counterfactual interpretations of Wide ResNet-50 for the classification of an *Egyptian Mau cat*. The figure comprises three synthesized counterfactual images accompanied by their corresponding prediction probabilities for both *Egyptian Mau cat* and *tabby cat* categories. Salient alterations in each counterfactual are accentuated and contrasted with their respective manifestations in the original input.

Explanations for Wide ResNet-50:

Analyzing the counterfactual explanations presented in Figure 6.7, we observe that in Counterfactual (a), a confined region, primarily encompassing the chest pattern and lower body of the cat, undergoes modification. Here, the black spots transform into black stripes. As a consequence, the prediction likelihood for the *Egyptian Mau cat* diminishes by 22.82%, settling at 75.59%, while the probability for the *tabby cat* escalates by 21.09%, reaching 22.21%. This emphasizes the pivotal role of the

pattern in these specific regions during classification. In Counterfactual (b), an expanded region is altered, with the spot patterns being predominantly transmuted into stripes. This leads to a further reduction in the prediction probability for *Egyptian Mau cat* to 20.39%, while that for the *tabby cat* surges to 74.42%. Notably, the final prediction now tilts in favor of the *tabby cat*. In Counterfactual (c), the vast majority of the body witnesses modifications. Though the fur color and posture remain consistent, the inherent pattern undergoes a transformation. The spots are replaced with stripes, culminating in a prediction probability of just 8.14% for the *Egyptian Mau cat* and a dominating 86.52% for the *Tabby cat*. This solidifies the assertion that for Wide ResNet-50, pattern differentiation remains paramount in distinguishing between the *Egyptian Mau cat* and the *tabby cat*.

6.4.1.2 Electric guitar & Acoustic guitar

Typically, *acoustic guitars* boast a hue reminiscent of wood, while *electric guitars* can span a broader color spectrum. However, when an *electric guitar* has a wooden hue, the distinction between the two becomes notably challenging. The provided counterfactuals aim to elucidate the pivotal features that classifiers rely on to distinguish these guitar variants, with the original input being an *electric guitar* rendered in wooden color.

Explanations for MobileNetV2:

As depicted in Figure 6.8, the synthesized counterfactuals highlight distinctions within the input image of an *electric guitar*. For Counterfactual (a), a localized region undergoes transformations to characteristics associated with *acoustic guitars*, with three primary features adjusted. The original image displays two pickups, components intrinsic to *electric guitars*, responsible for capturing string vibrations and transmitting them to the amplifier. Within the counterfactual, the upper pickup changes into a pickguard, while the lower one is removed, replaced by the underlying wood. Additionally, the pickup selector switch, tone control, and volume

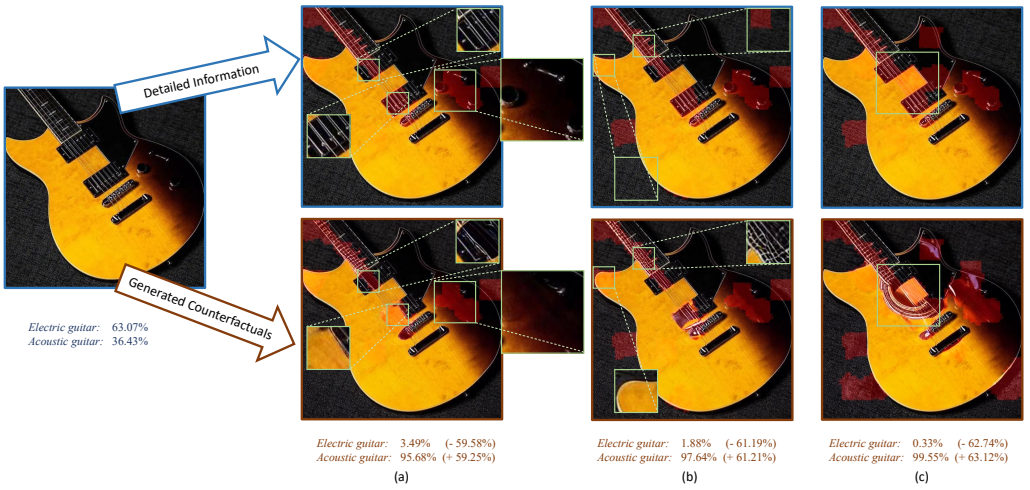


Figure 6.8: Counterfactual explanations using MobileNetV2 for an *electric guitar* depicted in a wooden hue. The original image, situated on the left, shows the classification probability for the *electric guitar* as 63.07% and 36.43% for the *acoustic guitar*. To the bottom right, three counterfactual images are juxtaposed with the original, with corresponding selected areas highlighted in the red mask. The extent of the modification area expands progressively from (a) to (c), along with the rise in the prediction probabilities for *acoustic guitar*. Key distinguishing features between the original and counterfactual images are magnified for clarity.

control are exclusive to *electric guitars* and are positioned on the guitar's right side, which are omitted in the counterfactual. Subsequent to these alterations, the classifier's prediction probability for the *electric guitar* diminishes significantly from 63.07% to 3.49%, whereas the *acoustic guitar*'s probability surges from 36.43% to 95.68%. These shifts underscore the pivotal role of the pickups, pickup selector switch, tone control, and volume control in MobileNetV2's classification mechanism.

In the second counterfactual image portrayed in Figure 6.8, a more expansive region undergoes modification compared to the first counterfactual. Similar to Counterfactual (a), the pickups, pickup selector switch, tone control, and volume control are excised. Additionally, the strings on the neck of the guitar manifest as thicker and denser, a characteristic feature of *acoustic guitars*. The previously acute angles at the top left

corner evolve to adopt a more rounded and elliptical contour, aligning more with the common shapes observed in *acoustic guitars*. With these alterations, the classifier's prediction probability for the *electric guitar* decreases to 1.88%, while that for the *acoustic guitar* increases to 97.64%. This observation strongly suggests that the presence of thick, dense strings, and the rounded and elliptical shape, are pivotal attributes for MobileNetV2's discernment between the two guitar types.

In the third counterfactual image depicted in Figure 6.8, an even more extensive set of superpixels undergoes alteration. Similar to the modifications observed in Counterfactuals (a) and (b), the pickup selector switch, tone control, and volume control on the guitar's right side are eliminated. Notably, instead of merely substituting the two pickups with the body's wood texture, a sound hole emerges in the area previously occupied by the two pickups. This sound hole, a hallmark of *acoustic guitars*, serves to amplify the instrument's sound. In contrast, *electric guitars* lack sound holes, relying instead on electric amplifiers for sound projection. In light of these modifications, the classifier's prediction probability for the *electric guitar* dwindles to merely 0.33%, while that for the *acoustic guitar* escalates to 99.55%. Evidently, the presence of a sound hole emerges as a pivotal characteristic for MobileNetV2 when discerning an *acoustic guitar*.

Explanations for Wide ResNet-50:

Figure 6.9 presents counterfactual explanations generated for Wide ResNet-50's classification of an *electric guitar* image. Three distinct counterfactuals elucidate key features contributing to the classification decision. In Counterfactual (a), the modifications are as follows: (1) the acute angles on the top-left corner of the guitar body transform into a gentler, elliptical curvature; (2)the top pickup is replaced by a sound hole. Post these alterations, the prediction probability for the *electric guitar* plummets dramatically from 95.04% to 22.29%, while the likelihood for *acoustic guitar* surges from 3.73% to 73.21%. This indicates that for Wide ResNet-50, the body's contour, presence of pickups and the sound hole are pivotal attributes in

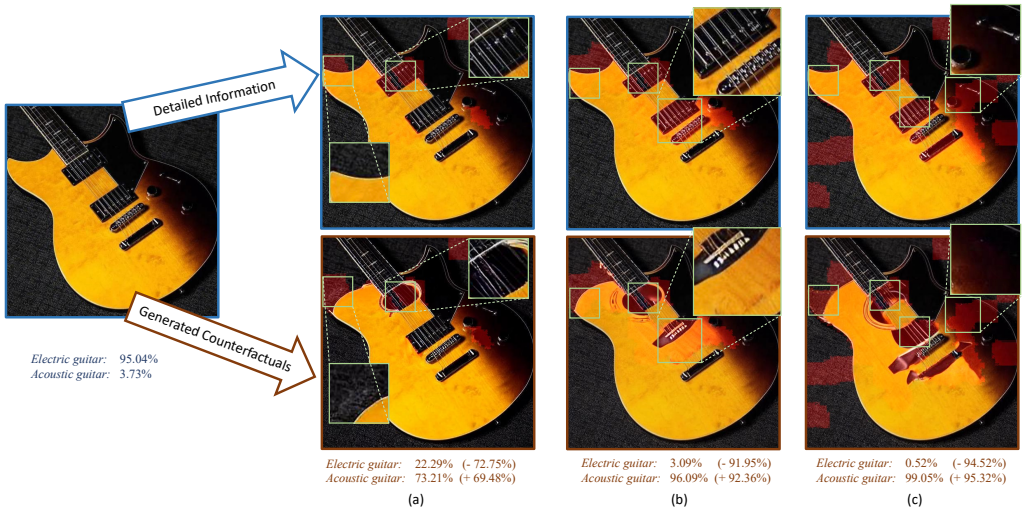


Figure 6.9: Counterfactual explanations using Wide ResNet-50 for an *electric guitar* depicted in a wooden hue. Three synthesized counterfactual images are presented, alongside the associated prediction probabilities for the *electric guitar* and *acoustic guitar* categories. Predominant modifications in each counterfactual are highlighted, juxtaposed with their original appearances in the input image.

distinguishing between guitar types. Notably, these influential features align intuitively with human discernment, thereby providing insights that are both explainable and relatable.

In Counterfactual (b), several features are altered, encompassing the modifications seen in Counterfactual (a). Specifically, the lower pickup, together with the bridge, is transmuted into a singular bridge for string support. With this additional alteration, the prediction likelihood for the *electric guitar* plunges further, reaching 3.09%. This provides evidence that the pickup is a salient feature for Wide ResNet-50 in deciding a guitar's category.

For Counterfactual (c), beyond the modifications seen in Counterfactual (b), the pickup selector switch and the volume control are obliterated, with these regions reverting to the native body surface of the guitar. The prediction probability for *electric guitar* dwindles to 0.52%, while that for *acoustic guitar* escalates to 99.05%. This showcases Wide ResNet-50's re-

liance on features such as the pickup selector switch and volume control in adjudicating a guitar's classification.

6.4.1.3 Yawl & Schooner

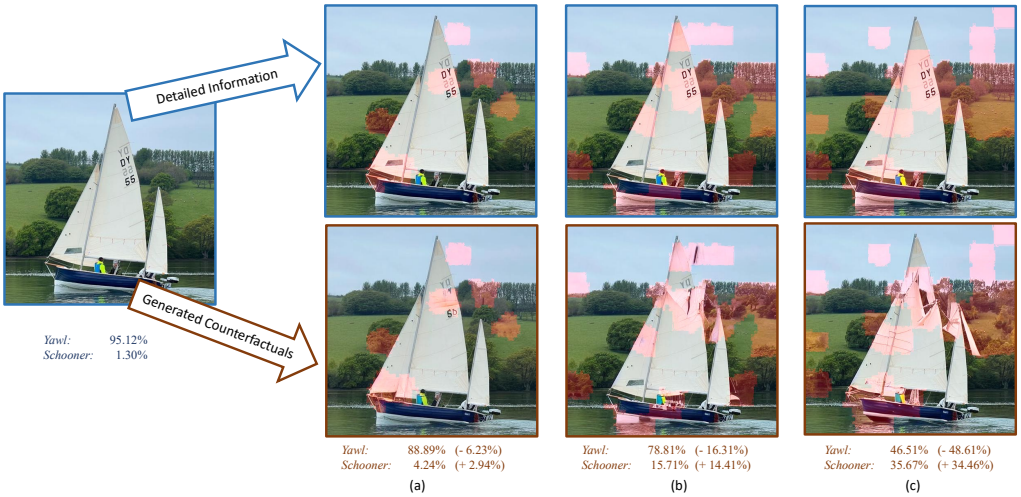


Figure 6.10: Counterfactual explanations using Wide ResNet-50 for a *yawl*. Three synthesized counterfactual images are presented alongside the associated prediction probabilities for the *yawl* and *schooner* categories. Modifications in each counterfactual are highlighted, compared with their original appearances in the input image.

A picture of a *yawl* is classified by Wide ResNet-50, and Figure 6.10 shows three counterfactual explanations to interpret why Wide ResNet-50 classifies it to be a *yawl* instead of a *schooner*. The major feature that affects the decision is the number of sails. The more sails in the counterfactuals, Wide ResNet-50 is more likely to classify the image into *schooner*.

6.4.1.4 Ambulance & Police Van

Figure 6.11 presents the counterfactual explanations for an *ambulance* image. The *ambulance* in the example looks like a *police van* in terms of the color distribution and the vehicle shape. From the generated counterfactual images and the corresponding prediction probabilities, we can see

Figure 6.11: Counterfactual explanations using MobileNetV2 for an *ambulance* which looks like a police van. Three counterfactual images are shown alongside the associated prediction probabilities for *ambulance* and *police van*. The modifications in each counterfactual are highlighted.

that, in this case, MobileNetV2 mainly makes decisions on the text on the body of the vehicle. If the text of 'AMBULANCE' is blurred, the predicted probability of it will decrease; if the text of 'POLICE' is added to the body, the classifier is very likely to classify the image into *police van*.

6.4.2 Comparisions with Other Methods

To assess the performance of SD-MOEX, it is compared against six prominent explanation methods. Among these, certain methods employ gradient-based explainability approaches: Integrated Gradients [141], which provides insights into model decisions by evaluating the integral of gradients between input features and a defined baseline, attributing importance based on the influence each feature exerts; and Grad-CAM [117], which leverages gradients concerning the target class to pinpoint crucial regions within the input image, yielding a broad localization map that denotes areas crucial for a specific prediction. On the other hand, methodologies

such as DeepLIFT [122] and SHAP [86] are importance score-based, assigning a score to quantify each input feature's contribution. LIME [109], a local explanatory technique, deciphers intricate model decisions by fitting a locally comprehensible linear model, uncovering the significance of individual features in that vicinity. Distinctively, Score-CAM [151] capitalizes on the model's activation maps and class scores to formulate a Class Activation Map (CAM), highlighting areas instrumental for classification decisions.

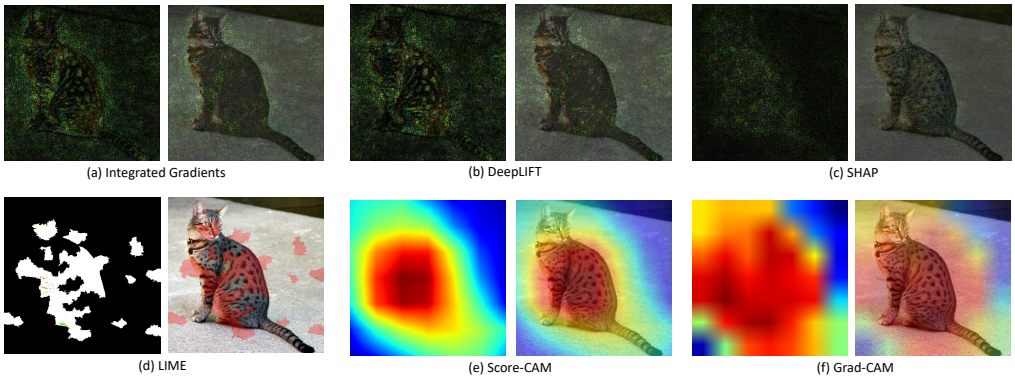


Figure 6.12: Visual explanations of MobileNetV2's decision for an image of an *Egyptian Mau cat*. Results from six distinct explanation methodologies are displayed. For each method, two images are showcased: the former presents the visualized heatmap, and the subsequent shows this heatmap upon the input, facilitating the identification of salient features.

These methods were utilized to elucidate MobileNetV2's classifications. For an input depicting an *Egyptian Mau cat*, the visualization results are illustrated in Figure 6.12. Both Integrated Gradients [141] and DeepLIFT [122] display the cat's contour and distinct markings, suggesting these features' importance; SHAP [86] accentuates the cat against its backdrop, indicating the prediction leans heavily on the cat rather than the image's other components; LIME [109] selects particular superpixels of the cat, denoting their significance for MobileNetV2. With Score-CAM [151], the cat's main body stands out while its head and tail seem less influential;

whereas Grad-CAM [117] emphasizes both the cat and its immediate surroundings. However, the explanations from these methods often span extensive regions, containing numerous features, complicating the discernment of pivotal elements. Moreover, they fall short in explaining 'how' these features steer the decision, let alone ascertaining why MobileNetV2 discerns the image as an *Egyptian Mau cat* instead of other breeds of cats.

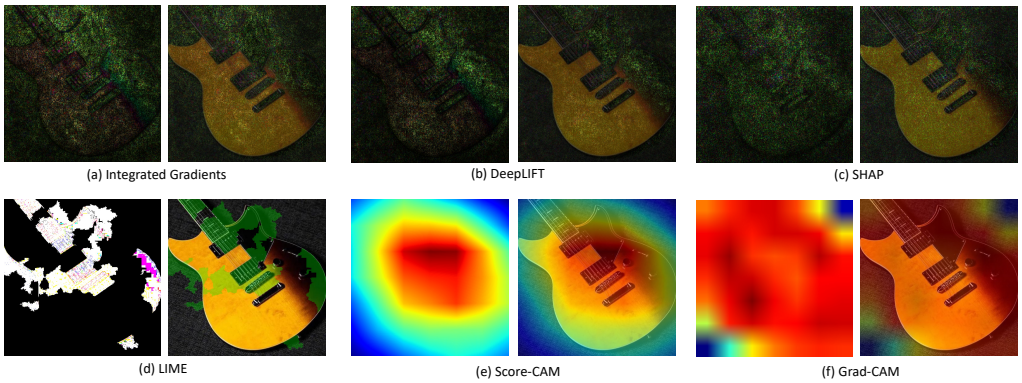
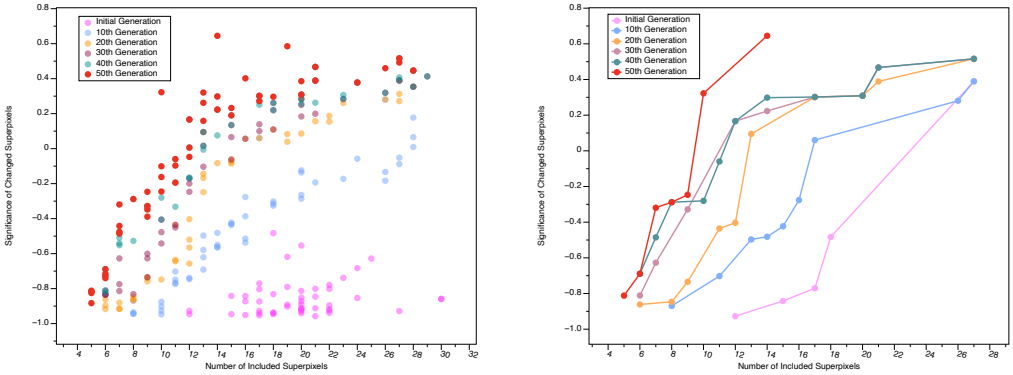


Figure 6.13: Visual explanations of MobileNetV2's decision for an image of an *electric guitar*.

Figure 6.13 showcases the explanation results for an image of an *electric guitar* for MobileNetV2. Integrated Gradients [141] and DeepLIFT [122] emphasize the significance of the guitar's right side. SHAP [86] similarly highlights the right side but also underscores the contour of the entire guitar. LIME [109] identifies key superpixels, encompassing the neck, pickups, bridge, and angle. Score-CAM [151] accentuates the guitar's central portion, while Grad-CAM [117] indicates the entirety of the guitar as being pivotal. However, these methodologies offer explanations that are broad, leveraging numerous features, and they cannot explain 'how' these features impact the classifier's decisions. On the contrary, the proposed SDMOEX can explain the classifiers' decisions much better by indicating both 'where' the important features are and 'how' they affect the classifiers.

6.4.3 Convergence Analysis



(a) The populaton distribution (b) The Pareto front

Figure 6.14: Evolutionary process for SD-MOEX with an input image of an *Egyptian Mau cat* processed by the MobileNetV2 model.

In this section, we analyze the evolutionary process across generations. Figure 6.14 showcases the distributions across generations alongside their corresponding Pareto fronts. This visual representation captures the evolution of SD-MOEX when the input is an image of an *Egyptian Mau cat* and the classifier in use is MobileNetV2. Observing the population distributions, it becomes evident that as generations progress, individuals increasingly exhibit fewer superpixels and greater impact. Analyzing the Pareto front, there is a noticeable performance improvement in the 'best' individuals during the initial 20 generations. Performance remains relatively stable between the 30th and 40th generations. However, a significant leap in performance is observed when comparing the 40th to the 50th generation. This depiction of the bjective values' evolution underscores the efficacy of the employed evolutionary algorithm.

6.5 Chapter Summary

This chapter aims to introduce a post-hoc, model-agnostic method to explain image classifiers' decisions, especially concerning similar classes. The method is expected to provide insight into both 'where' crucial features lie and 'how' they influence classifier decisions. This objective has been successfully achieved by the integration of the SD model for counterfactual generation, NSGA-II for searching for minimal and crucial superpixel set identification, and a novel evaluation method proposed in this chapter. Specifically, the SD model in SD-MOEX inpaints specific regions based on textual descriptions, resulting in naturalistic counterfactual images. Simultaneously, the tailored NSGA-II synergizes with the SD model to furnish an efficient evolutionary search mechanism. Additionally, the proposed evaluation metrics provide a comprehensive measure of superpixel quantity and impact.

The capabilities of SD-MOEX were evaluated on four classification tasks: *Egyptian Mau cat* versus *tabby cat*, *electric guitar* versus *acoustic guitar*, *yawl* versus *schooner*, and *ambulance* versus *police van*, using prominent image classifiers like MobileNetV2 and Wide ResNet-50. For the cat categorization, the derived counterfactual explanations delineate patterns that influence breed classification, highlighting the specific contributions of each pattern. In the guitar case, the interpretations shed light on influential components, rendering the results interpretable for human comprehension. For the yawl, the number of sails affects the decisions significantly. The text description on the vehicle body influences the decisions. A comparative analysis of six prevalent explanatory methods demonstrated the superiority of SD-MOEX in its explanations. Moreover, an analysis of evolutionary objective values distributions confirms the effectiveness of our evolutionary approach.

While SD-MOEX proves proficient in explaining image classifier decisions, its current scope is confined to image classification. In future work,

we plan to extend this method to broader computer vision tasks.

Chapter 7

Conclusions and Future Work

This thesis aims to enhance the effectiveness and efficiency of ENAS methodologies, primarily by improving classification accuracies and minimizing computational overhead. It also tries to explain the decisions of networks in image classification tasks. These objectives have been successfully achieved by proposing a novel PSO-based ENAS method with an autoencoder to transform architecture representations (EAEPSO in Chapter 3), an innovative performance predictor coupled with weight inheritance-assisted ENAS method (EPPGA in Chapter 4), a novel one-shot NAS method with reliable fitness evaluations and efficient supernet training (TIFNAS in Chapter 5), and a novel method to explain classifiers' decision on image classifications (SD-MOEX in Chapter 6). The first three methods primarily focus on enhancing classification performance and search efficiency via strategies such as novel encoding techniques (representations), reliable fitness evaluation techniques, advanced search spaces, efficient evolutionary processes, and novel evolutionary operations. The fourth method focuses on explaining classifiers' decisions through counterfactual generation. Experiments have been conducted to examine and contrast the proposed methods with existing ones on renowned image classification datasets of varying complexities. Results have shown the proposed methods can achieve superior classification performance and are more compu-

tationally efficient. Moreover, the interpretative approach, SD-MOEX, has been validated for its ability to explain classifier decisions.

The rest sections of this chapter outline conclusions for each thesis objective, summarize major findings, and highlight potential avenues for future research.

7.1 Achieved Objectives

This thesis has achieved the following research objectives:

1. Proposes an innovative PSO-based ENAS method for effectively and efficiently searching for network architectures. Central to this method is a uniquely designed autoencoder, purposed to help tailor the architecture representations to better utilize the ability of the search algorithm, i.e., PSO. Specifically, the autoencoder converts variable-length integer block vectors into fixed-length continuous decimal latent vectors, creating a smooth and continuous search space for the PSO search process. Notably, a multifaceted loss function for the autoencoder is developed. This function, extending beyond the traditional scope of reconstruction loss, incorporates considerations of architectural and model scale similarities, ensuring the networks with a similar architecture or model scale are embedded in neighborhood regions in the latent space. Besides, this work explores the effects of varying training data sizes on fitness value evaluations, revealing its consequential impact across diverse search stages. Accordingly, a dynamic hierarchical fitness evaluation strategy is designed to optimize the efficiency of performance estimations, which employs different training data scales throughout various stages of the search process, ensuring the accurate evaluations of the candidate networks. The proposed method is compared with 55 peer competitors on three main-stream image classification datasets, and the results confirm the outstanding performance of the proposed method.

2. Proposes a novel ENAS algorithm that employs a performance predictor and a weight-inheritance method. The method designs a novel lightweight backbone block architecture, contributing to the search for promising and portable network architectures. Furthermore, it employs a random forest-based performance predictor. Instead of the conventional fitness predictions, this predictor assesses whether offspring can outperform their parent(s), also quantifying the confidence of these predictions, which ensures that only top-performing offspring are retained, improving the evolutionary convergence speed. Furthermore, the algorithm introduces new crossover and mutation operators aiming at ensuring that the majority of newly generated offspring closely resemble one of their parents, thereby facilitating the prediction process. To optimize evaluation efficiency, a novel weight inheritance method is proposed, almost avoiding random initialization in the mutated layers of offspring. The proposed method is compared with 66 state-of-the-art competitors within the field, and the rigorous evaluation reveals a noteworthy advancement in terms of classification performance, model size, and computational cost. Further analysis also demonstrates the effectiveness of the proposed backbone block, performance predictor, and weight inheritance method, respectively.

3. Proposes a new one-shot NAS method for image classification. The method introduces an efficient supernet training strategy by leveraging the inherent characteristics of the supernet architecture and employing the weight inheritance technique. Furthermore, the method designs a supernet fine-tuning strategy to provide more accurate fitness evaluations for the candidate networks. Specifically, the fine-tuning is conducted along with the evolutionary process. In addition, a novel population initialization method is introduced to generate initial candidate architectures with potential high performance, facilitating the evolutionary search. Experimental results demon-

strate the effectiveness of the proposed method, achieving remarkable error rates of 2.59%, 16.67%, and 23.9% on CIFAR-10, CIFAR-100, and ImageNet, respectively, while demanding only 0.31, 0.28, and 4.1 GPU-days. Notably, these classification results exhibit promise, and the computational cost remains significantly lower than that of most competing methods. Extensive experiments and analyses corroborate the efficacy of the proposed supernet training, supernet fine-tuning, and population initialization methods.

4. Proposes a novel post-hoc, model-agnostic methodology to explain classifier decisions in similar classifications. The method combines a Stable Diffusion model to produce counterfactual images, thereby illuminating the significance of distinct image regions in classification outcomes. Furthermore, a multi-objective evolutionary algorithm is presented to identify both pivotal and minimal superpixels. The method can locate concise salience regions, identify important and straightforward futures, and provide intuitive explanations. Besides, a novel objective function is proposed to evaluate the impact of these superpixels. Experimental results emphasize the efficacy of this methodology across various similar classes, such as *Egyptian Mau cat* versus *Tabby cat* and *acoustic guitar* versus *electric guitar*. This robustness also extends to different classification networks.

7.2 Main Conclusions

This thesis demonstrates the efficacy of evolutionary computation in searching for promising network architectures for image classification. Additionally, it showcases the ability of evolutionary computation to provide interpretable explanations for classifiers' decisions in image classification tasks. This section summarizes the key findings drawn from the four research objectives in the contribution chapters (Chapter 3 to Chapter 6).

7.2.1 PSO for NAS using an Autoencoder-based Encoding Strategy

Chapter 3 proposes an autoencoder-based encoding strategy combined with a dynamic hierarchical fitness evaluation method to evolve CNNs, which improves the accuracy and significantly reduces computational costs.

Architecture Representation

It is observed that the architecture representation can be effectively condensed from the variable-length discrete integer block vectors to fixed-length continuous decimal latent vectors. This transformation renders the latent space become smooth and continuous, enhancing the downstream PSO search process.

The architecture representation, based on an autoencoder, is well-suited for the PSO search process. Specifically, the autoencoder captures meaningful features from the initial vector representation, reducing its dimensionality. The original block/architecture parameters are transformed from discrete values to continuous values. A newly designed batch-normalization layer is appended to the end of the encoder part, ensuring a smoother representation within the latent space. Furthermore, similar architectures are projected to neighborhood in the latent space, which is achieved by the novel loss function formulated for the autoencoder's training. The loss function takes into account the reconstruction loss, architecture similarity loss, and scale similarity loss. Experimental results validate the efficacy of the architecture representations generated by the autoencoder, outperforming those from a standard autoencoder in classification tasks. This research also shows that a novel representation strategy can help improve the searched results for ENAS tasks, because the search process is within a crafted latent space. This method opens up possibilities for incorporating various techniques and innovative concepts in building the latent space, offering a new perspective to enhance ENAS algorithms.

Hierarchical Fitness Evaluations

It is found that the dynamic hierarchical fitness evaluation method can reduce the computational overhead while providing effective evaluations for the candidate networks throughout the evolutionary process.

The proposed hierarchical fitness evaluation method dynamically adjusts the volume of the training data along with the PSO process, achieving a good balance between efficiency and accuracy. While training the candidate on a reduced dataset might compromise test accuracy compared to utilizing the entire dataset, fitness evaluations within the EC technique typically aim to guide the selection based on the relative performance of the individuals. In this context, the method adaptively modifies the scale of the training data to furnish precise relative performance comparisons. Experimental results indicate that the proposed method can achieve similar classification accuracy to using the full training data while using only a seventh of the computational resources. This method not only considers the assessment of the candidate, but also considers the role of the evaluation phase within the search process, being tailored according to the inherent characteristics of the search algorithm.

7.2.2 NAS based on Performance Prediction and Weight Inheritance

Chapter 4 introduces a method centered on performance prediction and weight inheritance to efficiently search for well-performing network architectures. It also introduces a new lightweight architecture to construct the search space.

Lightweight Architecture

A new backbone lightweight block architecture is proposed to build the basic blocks for candidate networks. Though inspired by the MobileNetV3 block, this architecture boasts greater efficiency and better per-

formance. Experimental results suggest that searches based on this newly-proposed block are markedly more efficient, with the resulting networks outperforming the accuracy achieved by those based on the conventional MobileNetV3 block. The proposed new block architecture's good performance shows that the improvement of the backbone structure affects the efficiency of the NAS methods and the performance of the searched results.

Performance Predictor

It is observed that a performance predictor can be used to help the generation of powerful offspring, thus accelerating the evolutionary process. The newly proposed random forest-based performance predictor is the key contribution of this method. While traditional preditors aim to estimate the performance of candidates directly, this proposed predictor adopts a distinct approach. Functioning as a random forest-based binary classifier, it seeks to predict the comparative outcomes between two candidates. In this way, the potentially wrong predictions will not affect the search process in the proposed method because the predictor's primary role is to aid in the selection of high-performing offspring. The actual fitness of these offspring undergoes further rigorous assessment. The experimental results show that incorporating the performance predictor significantly improves the performance of the searched network within a predefined number of evolutionary iterations. This method shows that the performance predictor can transcend mere fitness evaluations, playing a pivotal role in offspring selection, accelerating the convergence of the comprehensive search process, and consequently reducing computational expenses.

Weight Inheritance

It is noted that a powerful weight inheritance can substantially reduce the computational demands of the fitness evaluation process. Specifically,

a weight pool is constructed to collect the network modules that have undergone training. The offspring networks primarily inherit weights from their parent individuals; mutated blocks derive initial weights from this weight pool, thus avoiding random initialization. The experimental results reveal training candidates utilizing the proposed weight inheritance method outperforms training using traditional inheritance methods in efficiency and accuracy. The method shows that an efficient fitness evaluation method can help to reduce computational consumption.

7.2.3 One-Shot NAS

To further improve the efficiency of the NAS algorithm, Chapter 5 introduces a methodology grounded in the efficient one-shot framework. This method further reduces computational overhead by introducing an innovative supernet training methodology and improves the reliability of fitness evaluations through a distinct supernet fine-tuning strategy. Moreover, a novel population initialization is incorporated to enhance the evolutionary process.

Supernet Training

It is found that the novel supernet training strategy significantly boosts the training efficiency of the large supernet, subsequently diminishing the overall computational expenditure. The training strategy is based on the analysis of the complicated supernet architecture and leverages the weight replication technique, which entails the replication of weights from 'larger' operations to 'smaller' ones. The experimental results indicate that this method spends merely 28% of the training cost to achieve markedly enhanced training outcomes compared to the vanilla training methodology. Despite one-shot NAS methods being highly efficient, this methodology demonstrates further potential for optimizing supernet training efficiency.

Supernet Fine-tuning

It is suggested that the fine-tuning of the supernet can contribute to providing more reliable and precise fitness evaluations for candidate networks. The weights of the supernet undergo fine-tuning according to the distribution of the candidates awaiting evaluation, which can reduce the sharing extent and provide more reliable evaluations. The experimental results confirm the fine-tuning strategy can significantly improve fitness evaluation reliability and show that the searched networks are of higher classification accuracies when applied to the supernet fine-tuning strategy. Although the fine-tuning will consume a little more computation, amounting to 0.09 GPU-days on both the CIFAR-10 and CIFAR-100 datasets, the overall computational costs remain significantly lower than most peer competitors. Consequently, this methodology employs an efficient supernet fine-tuning strategy and successfully addresses the prevalent challenge of inconsistent fitness evaluations in one-shot NAS.

Population Initialization

It is noted that a better population initialization method can help the ENAS method in searching for network architectures with superior classification performance. This initialization method selects potential high-performing candidate network architectures as initial solutions, drawing insights from the supernet training phase. Given that the supernet undergoes training by iteratively sampling single paths, the performance metrics of the sampled paths can indicate the potential performance of the corresponding candidates. The promising candidates are then elected for the initial population. Empirical results validate that this population initialization method can significantly improve the overall performance of the initial population and lead to a better performance of the searched networks.

7.2.4 Explaining Image Classification

Different from the previous chapters focusing on searching for good network architectures, Chapter 6 introduces a method to explain the networks' decisions for image classification tasks. This method is achieved by an evolutionary multi-objective optimization technique and a counterfactual generation approach.

Counterfactual Generation

It is found that a good counterfactual generation method that integrates textual information can contribute to the explanations for the decisions made on similar classes. This generation is based on the Stable Diffusion (SD) model. Rather than directly employing the SD model to create an entire counterfactual image, this method capitalizes on the SD model to inpaint specific regions using features from a similar class. Consequently, by observing alterations in prediction outcomes, one can confirm the influence of the modified content (along with its associated features). It can be observed that the generated content blends smoothly and naturally with the surrounding areas, thereby avoiding abrupt edges that could potentially interfere with the classifier. The results confirm that this promising counterfactual generation method can assist in explaining the decisions of the classifiers.

Explainations for Similar Image Classes

It is demonstrated that the method combines the SD model and an Evolutionary Multi-Objective (EMO) method to provide explanations for similar image classes. Classifying similar classes is difficult for the classifiers, and providing explanations is more complicated. The SD model can paint content comprising the features of a similar class, and the EMO method is employed to search for smaller but pivotal superpixels/regions. Generally, a large region has more impact on the decision but also contains more features. The more features are selected, the harder the explanations will

be because it will be challenging to identify which feature(s) contribute most to the impact. The experimental results show that the pivotal regions are found using the EMO method, and the corresponding features help humans understand the causal reasoning behind the classifiers' decisions.

7.3 Future Work

This section highlights key research directions for future work.

7.3.1 Multi-Objective Neural Architecture Search

In this thesis, the ENAS methods (i.e., in Chapters 3, 4, and 5) have only a single objective, i.e., to maximize the accuracy of the candidate architectures. These methods achieve good classification performance. However, when distributing these models to real-world applications, there are some additional considerations. For instance, when deploying models onto mobile devices, the model's size becomes imperative. Similarly, in domains such as autonomous driving, the inference time of a model is of great importance. By adopting a multi-objective NAS approach, one can accommodate a number of objectives, generating models with different trade-offs between the objectives to meet the different requirements of the users.

7.3.2 Neural Architecture Search for Broader Computer Vision Tasks

This thesis shows ENAS can achieve good performance in image classification tasks, consistently performing well across a number of datasets with varying classification complexities. However, there are some more computer vision tasks, such as image generation, semantic segmentation, and object recognition. Contrasted with image classification, the backbone net-

work architectures for these tasks differ significantly, and the evaluation metrics are also quite different from classification. Assessing the candidate's performance on these tasks may cost more computational resources, and some existing methods spend a lot of computational resources on the evaluations. Hence, it is worthwhile to design new efficient ENAS methods for these tasks to expand the application fields of NAS.

7.3.3 Network Architecture Search based on Novel Backbone Structures

This thesis explores the NAS methodologies based on the search space constructed from different typical network structures, including DenseNet and MobileNet. This thesis also refines and enhances the backbone structure to build a better search space. However, state-of-the-art network structures are continuously invented and redefined. The NAS methods based on the structures of potentially good performance are more likely to achieve promising outcomes. Consequently, there emerges a compelling need to develop NAS methods that are flexible and can accommodate the emergent backbone structures.

7.3.4 Efficient and Accurate Fitness Evaluations

In this thesis, several methods are proposed to evaluate the performance of candidate network architectures, keeping a balance between evaluation efficiency and accuracy. However, these evaluations are still resource-intensive, demanding significant computational power. Besides, for most methods, the increase in efficiency comes at the cost of a decrease in accuracy. In the future, we will keep refining the fitness evaluation strategies, aiming for a good balance between efficiency and accuracy. Specifically, there are several existing efficient fitness evaluation methods, and an ensemble of them may achieve a trade-off between efficiency and accuracy, which is worth exploring.

7.3.5 Explanations for Networks across Varied Applications

This thesis develops a methodology to explain the networks' decisions for image classifications and achieves good results. However, the realm of interpretability extends far beyond classification tasks alone. The deep networks for other applications, such as segmentation, generation, and recognition, also need to be explained. Compared with classification, the networks may be of higher complexity; thus, interpreting them may be much more challenging. In the future, we plan to develop new methods to explain the deep networks across an expanded spectrum of application domains.

7.3.6 Explanations for Deep Network Architectures

This thesis explains the networks' decisions on image classification tasks from the point of the input data. It is also important to explain why a specific deep network architecture can achieve good performance on specific application fields. In this way, researchers are more likely to design more powerful network architectures, and the NAS methods may also achieve better results.

7.3.7 Real-World Applications

In addition to the benchmark image classification datasets, it is important to apply the proposed NAS methods to solve real-world image classification tasks, such as aquaculture image classification. Specifically, the ENAS methods can be applied to analyze shell-fish data on New Zealand green mussels, and the fin-fish image data on New Zealand snappers.

Bibliography

[1] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[2] https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/.

[3] https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning.

[4] ACHANTA, R., SHAJI, A., SMITH, K., LUCCHI, A., FUA, P., AND SÜSSTRUNK, S. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 11 (2012), 2274–2282.

[5] ADELI, H. Neural networks in civil engineering: 1989–2000. *Computer-Aided Civil and Infrastructure Engineering 16*, 2 (2001), 126–142.

[6] AHMED, A. A., DARWISH, S. M. S., AND EL-SHERBINY, M. M. A novel automatic cnn architecture design approach based on genetic algorithm. In *International Conference on Advanced Intelligent Systems and Informatics* (2019), Springer, pp. 473–482.

[7] ALBAWI, S., MOHAMMED, T. A., AND AL-ZAWI, S. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)* (2017), Ieee, pp. 1–6.

[8] AN, J., AND CHO, S. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE 2*, 1 (2015), 1–18.

[9] ANTONIOU, A., EDWARDS, H., AND STORKEY, A. How to train your maml. *arXiv preprint arXiv:1810.09502* (2018).

[10] ASSUNÇÃO, F., CORREIA, J., CONCEIÇÃO, R., PIMENTA, M. J. M., TOMÉ, B., LOURENÇO, N., AND MACHADO, P. Automatic design of artificial neural networks for gamma-ray detection. *IEEE Access 7* (2019), 110531–110540.

[11] ASSUNÇÃO, F., LOURENÇO, N., MACHADO, P., AND RIBEIRO, B. Evolving the topology of large scale deep neural networks. In *European Conference on Genetic Programming* (2018), Springer, pp. 19–34.

[12] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. Handbook of evolutionary computation. *Release 97*, 1 (1997), B1.

[13] BAKER, B., GUPTA, O., NAIK, N., AND RASKAR, R. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167* (2016).

[14] BANZHAF, W. *Genetic Programming: First European Workshop, EuroGP'98, Paris, France, April 14-15, 1998, Proceedings*, vol. 1. Springer Science & Business Media, 1998.

[15] BAYMURZINA, D., GOLIKOV, E., AND BURTSEV, M. A review of neural architecture search. *Neurocomputing 474* (2022), 82–93.

[16] BENDER, G., KINDERMANS, P.-J., ZOPH, B., VASUDEVAN, V., AND LE, Q. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning* (2018), PMLR, pp. 550–559.

[17] BINGHAM, G., MACKE, W., AND MIIKKULAINEN, R. Evolutionary optimization of deep learning activation functions. *arXiv preprint arXiv:2002.07224* (2020).

[18] CAI, H., CHEN, T., ZHANG, W., YU, Y., AND WANG, J. Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2018), vol. 32.

[19] CAI, H., ZHU, L., AND HAN, S. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations* (2018).

[20] CAI, Z., CHEN, L., AND LIU, H.-L. BHE-DARTS: Bilevel optimization based on hypergradient estimation for differentiable architecture search. In *Proceedings of ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2023), pp. 1–5.

[21] CARVALHO, P., LOURENÇO, N., ASSUNÇÃO, F., AND MACHADO, P. Autolr: an evolutionary approach to learning rate policies. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (2020), pp. 672–680.

[22] CHATTOPADHAY, A., SARKAR, A., HOWLADER, P., AND BALASUBRAMANIAN, V. N. Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks. In *Proceedings of 2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (2018), pp. 839–847.

[23] CHEN, X., KINGMA, D. P., SALIMANS, T., DUAN, Y., DHARIWAL, P., SCHULMAN, J., SUTSKEVER, I., AND ABBEEL, P. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731* (2016).

[24] CHEN, Y., GAO, R., LIU, F., AND ZHAO, D. Modulenet: Knowledge-inherited neural architecture search. *IEEE Transactions on Cybernetics* (2021), 1–11.

[25] CHU, X., ZHANG, B., XU, R., AND LI, J. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845* (2019).

[26] CLERC, M. *Particle swarm optimization*, vol. 93. John Wiley & Sons, 2010.

[27] CROITORU, F.-A., HONDRU, V., IONESCU, R. T., AND SHAH, M. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023), 1–20.

[28] DATTA, D., AND FIGUEIRA, J. R. A real-integer-discrete-coded particle swarm optimization for design problems. *Applied Soft Computing 11*, 4 (2011), 3625–3633.

[29] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6*, 2 (2002), 182–197.

[30] DEVRIES, T., AND TAYLOR, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552* (2017).

[31] DHARIWAL, P., AND NICHOL, A. Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems 34* (2021), 8780–8794.

[32] DORIGO, M., BIRATTARI, M., AND STUTZLE, T. Ant colony optimization. *IEEE Computational Intelligence Magazine 1*, 4 (2006), 28–39.

[33] DP, K., AND BA, J. Adam: A method for stochastic optimization. In *Proc. of the 3rd International Conference for Learning Representations (ICLR)* (2015).

[34] EBERHART, R., AND KENNEDY, J. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science* (1995), Ieee, pp. 39–43.

[35] ELSKEN, T., METZEN, J. H., AND HUTTER, F. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081* (2018).

[36] ELSKEN, T., METZEN, J. H., AND HUTTER, F. Neural architecture search: A survey. *The Journal of Machine Learning Research 20*, 1 (2019), 1997–2017.

[37] EVANS, B. P., XUE, B., AND ZHANG, M. What's inside the black-box? a genetic programming method for interpreting complex machine learning models. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 1012–1020.

[38] FIELDING, B., AND ZHANG, L. Evolving image classification architectures with enhanced particle swarm optimisation. *IEEE Access 6* (2018), 68560–68575.

[39] FUJINO, S., NAOKI, M., AND MATSUMOTO, K. Deep convolutional networks for human sketches by means of the evolutionary deep learning. fuzzy systems association and 9th international conference on soft computing and intelligent systems (ifsa-scis). In *2017 Joint 17th World Congress of International. IEEE* (2017).

[40] GAO, Z., LI, Y., YANG, Y., WANG, X., DONG, N., AND CHIANG, H.-D. A gpso-optimized convolutional neural networks for eeg-based emotion recognition. *Neurocomputing 380* (2020), 225–235.

[41] GOETSCHALCKX, L., ANDONIAN, A., OLIVA, A., AND ISOLA, P. Ganalyze: Toward visual definitions of cognitive image properties. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2019).

[42] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial networks. *Commun. ACM 63*, 11 (oct 2020), 139–144.

[43] GOODFELLOW, I., WARDE-FARLEY, D., MIRZA, M., COURVILLE, A., AND BENGIO, Y. Maxout networks. In *International Conference on Machine Learning* (2013), PMLR, pp. 1319–1327.

[44] GOYAL, Y., WU, Z., ERNST, J., BATRA, D., PARIKH, D., AND LEE, S. Counterfactual visual explanations. In *Proceedings of the 36th International Conference on Machine Learning* (09–15 Jun 2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 2376–2384.

[45] GU, J., WANG, Z., KUEN, J., MA, L., SHAHROUDY, A., SHUAI, B., LIU, T., WANG, X., WANG, G., CAI, J., AND CHEN, T. Recent advances in convolutional neural networks. *Pattern Recognition 77* (2018), 354–377.

[46] GUO, Z., ZHANG, X., MU, H., HENG, W., LIU, Z., WEI, Y., AND SUN, J. Single path one-shot neural architecture search with uniform sampling. In *Proceedings of European Conference on Computer Vision* (2020), Springer, pp. 544–560.

[47] HE, K., ZHANG, X., REN, S., AND SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1026–1034.

[48] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[49] HELWIG, S., AND WANKA, R. Theoretical analysis of initial particle swarm behavior. In *Parallel Problem Solving from Nature – PPSN X* (Berlin, Heidelberg, 2008), G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, Eds., Springer Berlin Heidelberg, pp. 889–898.

[50] HINTON, G. E., AND ZEMEL, R. S. Autoencoders, minimum description length, and helmholtz free energy. *Advances in Neural Information Processing Systems 6* (1994), 3–10.

[51] HO, J., CHEN, X., SRINIVAS, A., DUAN, Y., AND ABBEEL, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the 36th International Conference on Machine Learning* (09–15 Jun 2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 2722–2730.

[52] HOWARD, A. G., SANDLER, M., CHU, G., CHEN, L.-C., CHEN, B., TAN, M., WANG, W., ZHU, Y., PANG, R., VASUDEVAN, V., LE, Q. V., AND ADAM, H. Searching for mobilenetv3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), 1314–1324.

[53] HOWARD, A. G., ZHU, M., CHEN, B., KALENICHENKO, D., WANG, W., WEYAND, T., ANDREETTO, M., AND ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[54] HU, J., SHEN, L., AND SUN, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 7132–7141.

[55] HUANG, G., LIU, S., VAN DER MAATEN, L., AND WEINBERGER, K. Q. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 2752–2761.

[56] HUANG, G., LIU, Z., VAN DER MAATEN, L., AND WEINBERGER, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 4700–4708.

[57] HUANG, Z., WANG, X., HUANG, L., HUANG, C., WEI, Y., AND LIU, W. Ccnet: Criss-cross attention for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2019).

[58] JANG, E., GU, S., AND POOLE, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[59] JORDAN, M. I., AND MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science 349*, 6245 (2015), 255–260.

[60] JUNIOR, F. E. F., AND YEN, G. G. Particle swarm optimization of deep neural networks architectures for image classification. *Swarm and Evolutionary Computation 49* (2019), 62–74.

[61] KARRAS, T., LAINE, S., AITTALA, M., HELLSTEN, J., LEHTINEN, J., AND AILA, T. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[62] KIMURA, S., AND MATSUMURA, K. Genetic algorithms using low-discrepancy sequences. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation* (New York, NY, USA, 2005), GECCO '05, Association for Computing Machinery, pp. 1341–1346.

[63] KOTSIANTIS, S. B., ZAHARAKIS, I., PINTELAS, P., ET AL. Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering 160*, 1 (2007), 3–24.

[64] KOZA, J. R. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing 4*, 2 (1994), 87–112.

[65] KRIZHEVSKY, A., HINTON, G., ET AL. Learning multiple layers of features from tiny images.

[66] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25* (2012), 1097–1105.

[67] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM 60*, 6 (may 2017), 84–90.

[68] KUŞ, Z., AKKAN, C., AND GÜLCÜ, A. Novel surrogate measures based on a similarity network for neural architecture search. *IEEE Access 11* (2023), 22596–22613.

[69] LANG, O., GANDELSMAN, Y., YAROM, M., WALD, Y., ELIDAN, G., HASSIDIM, A., FREEMAN, W. T., ISOLA, P., GLOBERSON, A., IRANI, M., AND MOSSERI, I. Explaining in style: Training a gan to explain a classifier in stylespace. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2021), pp. 693–702.

[70] LARSSON, G., MAIRE, M., AND SHAKHNAROVICH, G. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648* (2016).

[71] 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.

[72] Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[73] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 19–34.

[74] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436* (2017).

[75] Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[76] Liu, J., Gong, M., Miao, Q., Wang, X., and Li, H. Structure learning for deep neural networks based on multiobjective optimization. *IEEE transactions on Neural Metworks and Learning Systems 29*, 6 (2017), 2450–2463.

[77] Liu, P., El Basha, M. D., Li, Y., Xiao, Y., Sanelli, P. C., and Fang, R. Deep evolutionary networks with expedited genetic algorithms for medical image denoising. *Medical Image Analysis 54* (2019), 306–315.

[78] Liu, S., Zhang, H., and Jin, Y. A survey on computationally efficient neural architecture search. *Journal of Automation and Intelligence 1*, 1 (2022), 100002.

[79] LIU, Y., SUN, Y., XUE, B., ZHANG, M., YEN, G. G., AND TAN, K. C. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems* (Aug 06, 2021). doi: 10.1109/TNNLS.2021.3100554.

[80] LONI, M., MAJD, A., LONI, A., DANESHTALAB, M., SJÖDIN, M., AND TROUBITSYNA, E. Designing compact convolutional neural network for embedded stereo vision systems. In *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)* (2018), IEEE, pp. 244–251.

[81] LORENZO, P. R., NALEPA, J., KAWULOK, M., RAMOS, L. S., AND PASTOR, J. R. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2017), pp. 481–488.

[82] 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.

[83] LU, Z., DEB, K., GOODMAN, E., BANZHAF, W., AND BODDETI, V. N. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In *Computer Vision – ECCV 2020* (Cham, 2020), A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Springer International Publishing, pp. 35–51.

[84] LU, Z., WHALEN, I., BODDETI, V., DHEBAR, Y., DEB, K., GOODMAN, E., AND BANZHAF, W. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 419–427.

[85] LU, Z., WHALEN, I., DHEBAR, Y., DEB, K., GOODMAN, E., BANZHAF, W., AND BODDETI, V. N. Multi-objective evolutionary

design of deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation* (2020).

[86] LUNDBERG, S. M., AND LEE, S.-I. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems 30* (2017).

[87] MA, N., ZHANG, X., ZHENG, H.-T., AND SUN, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)* (2018), pp. 116–131.

[88] MA, Z., AND VANDENBOSCH, G. A. E. Impact of random number generators on the performance of particle swarm optimization in antenna design. In *2012 6th European Conference on Antennas and Propagation (EUCAP)* (2012), pp. 925–929.

[89] MAHESH, B. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet] 9*, 1 (2020), 381–386.

[90] MARTÍN, A., LARA-CABRERA, R., FUENTES-HURTADO, F., NARANJO, V., AND CAMACHO, D. Evodeep: a new evolutionary approach for automatic deep neural networks parametrisation. *Journal of Parallel and Distributed Computing 117* (2018), 180–191.

[91] MIIKKULAINEN, R., LIANG, J., MEYERSON, E., RAWAL, A., FINK, D., FRANCON, O., RAJU, B., SHAHRZAD, H., NAVRUZYAN, A., DUFFY, N., ET AL. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.

[92] MITCHELL, M. *An introduction to genetic algorithms*. MIT press, 1998.

[93] MO, H., CUSTODE, L. L., AND IACCA, G. Evolutionary neural architecture search for remaining useful life prediction. *Applied Soft Computing 108* (2021), 107474.

[94] MORRISON, R. W. Dispersion-based population initialization. In *Genetic and Evolutionary Computation — GECCO 2003* (Berlin, Heidelberg, 2003), E. Cantú-Paz, J. A. Foster, K. Deb, L. D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. Miller, Eds., Springer Berlin Heidelberg, pp. 1210–1221.

[95] NICHOL, A., DHARIWAL, P., RAMESH, A., SHYAM, P., MISHKIN, P., MCGREW, B., SUTSKEVER, I., AND CHEN, M. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741* (2021).

[96] NING, X., TANG, C., LI, W., ZHOU, Z., LIANG, S., YANG, H., AND WANG, Y. Evaluating efficient performance estimators of neural architectures. In *Advances in Neural Information Processing Systems* (2021), M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., pp. 12265–12277.

[97] NOBLE, W. S. What is a support vector machine? *Nature Biotechnology 24*, 12 (2006), 1565–1567.

[98] NOH, H., HONG, S., AND HAN, B. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1520–1528.

[99] O'NEILL, D., XUE, B., AND ZHANG, M. Neural architecture search for sparse densenets with dynamic compression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (2020), pp. 386–394.

[100] PHAM, H., GUAN, M., ZOPH, B., LE, Q., AND DEAN, J. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning* (2018), PMLR, pp. 4095–4104.

[101] PRICE, K., STORN, R. M., AND LAMPINEN, J. A. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[102] RAHNAMAYAN, S., TIZHOOSH, H. R., AND SALAMA, M. M. Quasi-oppositional differential evolution. In *2007 IEEE Congress on Evolutionary Computation* (2007), pp. 2229–2236.

[103] RAPAPORT, E., SHRIKI, O., AND PUZIS, R. Eegnas: Neural architecture search for electroencephalography data analysis and decoding. In *International Workshop on Human Brain and Artificial Intelligence* (2019), Springer, pp. 3–20.

[104] RAWAL, A., AND MIIKKULAINEN, R. From nodes to networks: Evolving recurrent neural networks. *arXiv preprint arXiv:1803.04439* (2018).

[105] REAL, E., AGGARWAL, A., HUANG, Y., AND LE, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 4780–4789.

[106] REAL, E., MOORE, S., SELLE, A., SAXENA, S., SUEMATSU, Y. L., TAN, J., LE, Q. V., AND KURAKIN, A. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), pp. 2902–2911.

[107] REN, J., LI, Z., YANG, J., XU, N., YANG, T., AND FORAN, D. J. Eigen: Ecologically-inspired genetic approach for neural network structure searching from scratch. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 9059–9068.

[108] REN, P., XIAO, Y., CHANG, X., HUANG, P.-Y., LI, Z., CHEN, X., AND WANG, X. A comprehensive survey of neural architecture

search: Challenges and solutions. *ACM Computing Surveys (CSUR) 54*, 4 (2021), 1–34.

[109] RIBEIRO, M. T., SINGH, S., AND GUESTRIN, C. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386* (2016).

[110] ROMBACH, R., BLATTMANN, A., LORENZ, D., ESSER, P., AND OMMER, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2022), pp. 10684–10695.

[111] RONNEBERGER, O., FISCHER, P., AND BROX, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (Cham, 2015), N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Springer International Publishing, pp. 234–241.

[112] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., ET AL. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision 115*, 3 (2015), 211–252.

[113] SAFAVIAN, S. R., AND LANDGREBE, D. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics 21*, 3 (1991), 660–674.

[114] SALOMON, R. Raising theoretical questions about the utility of genetic algorithms. In *International Conference on Evolutionary Programming* (1997), Springer, pp. 275–284.

[115] SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A., AND CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 4510–4520.

[116] SAPRA, D., AND PIMENTEL, A. D. Constrained evolutionary piecemeal training to design convolutional neural networks. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (2020), Springer, pp. 709–721.

[117] SELVARAJU, R. R., COGSWELL, M., DAS, A., VEDANTAM, R., PARIKH, D., AND BATRA, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (Oct 2017).

[118] SEN, S. A survey of intrusion detection systems using evolutionary computation. In *Bio-inspired Computation in Telecommunications*. Elsevier, 2015, pp. 73–94.

[119] SHI, C., HAO, Y., LI, G., AND XU, S. Ebnas: Efficient binary network design for image classification via neural architecture search. *Engineering Applications of Artificial Intelligence 120* (2023), 105845.

[120] SHI, Y. Particle swarm optimization. *IEEE connections 2*, 1 (2004), 8–13.

[121] SHI, Y., AND EBERHART, R. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)* (1999), vol. 3, pp. 1945–1950 Vol. 3.

[122] SHRIKUMAR, A., GREENSIDE, P., AND KUNDAJE, A. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning* (06–11 Aug 2017), D. Precup and Y. W. Teh, Eds., vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 3145–3153.

[123] SHU, H., AND WANG, Y. Automatically searching for u-net image translator architecture. *arXiv preprint arXiv:2002.11581* (2020).

[124] SHU, Y., WANG, W., AND CAI, S. Understanding architectures learnt by cell-based neural architecture search. *arXiv preprint arXiv:1909.09569* (2019).

[125] SIMONYAN, K., VEDALDI, A., AND ZISSERMAN, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034* (2013).

[126] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[127] SO, D., LE, Q., AND LIANG, C. The evolved transformer. In *International Conference on Machine Learning* (2019), PMLR, pp. 5877–5886.

[128] SPRINGENBERG, J., DOSOVITSKIY, A., BROX, T., AND RIEDMILLER, M. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)* (2015).

[129] SRIVASTAVA, R. K., GREFF, K., AND SCHMIDHUBER, J. Highway networks. *arXiv preprint arXiv:1505.00387* (2015).

[130] SU, X., YOU, S., ZHENG, M., WANG, F., QIAN, C., ZHANG, C., AND XU, C. K-shot nas: Learnable weight-sharing for nas with k-shot supernets. In *Proceedings of the 38th International Conference on Machine Learning* (18–24 Jul 2021), M. Meila and T. Zhang, Eds., vol. 139 of *Proceedings of Machine Learning Research*, PMLR, pp. 9880–9890.

[131] SUCH, F. P., MADHAVAN, V., CONTI, E., LEHMAN, J., STANLEY, K. O., AND CLUNE, J. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017).

[132] SUGANUMA, M., SHIRAKAWA, S., AND NAGAO, T. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2017), pp. 497–504.

[133] SUN, Q., LI, X., JIAO, L., REN, Y., SHANG, F., AND LIU, F. Fast and effective: A novel sequential single-path search for mixed-precision-quantized networks. *IEEE Transactions on Cybernetics* (2022), 1–13.

[134] SUN, Q.-S., ZENG, S.-G., LIU, Y., HENG, P.-A., AND XIA, D.-S. A new method of feature fusion and its application in image recognition. *Pattern Recognition 38*, 12 (2005), 2437–2448.

[135] SUN, Y., SUN, X., FANG, Y., YEN, G. G., AND LIU, Y. A novel training protocol for performance predictors of evolutionary neural architecture search algorithms. *IEEE Transactions on Evolutionary Computation 25*, 3 (2021), 524–536.

[136] SUN, Y., WANG, H., XUE, B., JIN, Y., YEN, G. G., AND ZHANG, M. Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Transactions on Evolutionary Computation 24*, 2 (2019), 350–364.

[137] SUN, Y., XUE, B., ZHANG, M., AND YEN, G. G. A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE Transactions on Neural Networks and Learning Systems 30*, 8 (2018), 2295–2309.

[138] SUN, Y., XUE, B., ZHANG, M., AND YEN, G. G. Completely automated cnn architecture design based on blocks. *IEEE Transactions on Neural Networks and Learning Systems 31*, 4 (2019), 1242–1254.

[139] SUN, Y., XUE, B., ZHANG, M., AND YEN, G. G. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation 24*, 2 (2019), 394–407.

[140] SUN, Y., XUE, B., ZHANG, M., YEN, G. G., AND LV, J. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics 50*, 9 (2020), 3840–3854.

[141] SUNDARARAJAN, M., TALY, A., AND YAN, Q. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning* (06–11 Aug 2017), D. Precup and Y. W. Teh, Eds., vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 3319–3328.

[142] SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., AND RABINOVICH, A. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1–9.

[143] TACKETT, W. A., AND CARMI, A. The unique implications of brood selection for genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence* (1994), IEEE, pp. 160–165.

[144] TOMSETT, R., HARBORNE, D., CHAKRABORTY, S., GURRAM, P., AND PREECE, A. Sanity checks for saliency metrics. *Proceedings of the AAAI Conference on Artificial Intelligence 34*, 04 (Apr. 2020), 6021–6029.

[145] VAN DEN BERGH, F., AND ENGELBRECHT, A. P. A study of particle swarm optimization particle trajectories. *Information Sciences 176*, 8 (2006), 937–971.

[146] VERMA, S., PANT, M., AND SNASEL, V. A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems. *IEEE Access 9* (2021), 57757–57791.

[147] WANG, B., SUN, Y., XUE, B., AND ZHANG, M. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (2018), IEEE, pp. 1–8.

[148] WANG, B., SUN, Y., XUE, B., AND ZHANG, M. A hybrid ga-pso method for evolving architecture and short connections of deep convolutional neural networks. In *Pacific Rim International Conference on Artificial Intelligence* (2019), Springer, pp. 650–663.

[149] WANG, B., XUE, B., AND ZHANG, M. Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks. In *2020 IEEE Congress on Evolutionary Computation (CEC)* (2020), IEEE, pp. 1–8.

[150] WANG, B., XUE, B., AND ZHANG, M. Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[151] WANG, H., WANG, Z., DU, M., YANG, F., ZHANG, Z., DING, S., MARDZIEL, P., AND HU, X. Score-cam: Score-weighted visual explanations for convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (June 2020).

[152] WEI, C., NIU, C., TANG, Y., WANG, Y., HU, H., AND LIANG, J. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems* (2022), 1–15.

[153] WEI, C., TANG, Y., NIU, C. N. C., HU, H., WANG, Y., AND LIANG, J. Self-supervised representation learning for evolutionary neural

architecture search. *IEEE Computational Intelligence Magazine 16*, 3 (2021), 33–49.

[154] WEI, H., YANG, Y., WU, H., TANG, Y., LIU, M., AND LI, J. Automatic feature selection by one-shot neural architecture search in recommendation systems. In *Proceedings of the ACM Web Conference 2023* (New York, NY, USA, 2023), WWW '23, Association for Computing Machinery, pp. 1765–1772.

[155] XIE, L., AND YUILLE, A. Genetic CNN. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 1379–1388.

[156] XIE, S., GIRSHICK, R., DOLLÁR, P., TU, Z., AND HE, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1492–1500.

[157] XIE, S., ZHENG, H., LIU, C., AND LIN, L. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).

[158] XU, Z.-B., LEUNG, K.-S., LIANG, Y., AND LEUNG, Y. Efficiency speed-up strategies for evolutionary computation: fundamentals and fast-gas. *Applied Mathematics and Computation 142*, 2-3 (2003), 341–388.

[159] YAN, S., ZHENG, Y., AO, W., ZENG, X., AND ZHANG, M. Does unsupervised architecture representation learning help neural architecture search? *Advances in Neural Information Processing Systems 33* (2020).

[160] YANG, Z., WANG, Y., CHEN, X., SHI, B., XU, C., XU, C., TIAN, Q., AND XU, C. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 1829–1838.

[161] YOU, J., LESKOVEC, J., HE, K., AND XIE, S. Graph structure of neural networks. In *International Conference on Machine Learning* (2020), PMLR, pp. 10881–10891.

[162] YU, K., SUITO, C., JAGGI, M., MUSAT, C.-C., AND SALZMANN, M. Evaluating the search phase of neural architecture search. In *ICRL 2020 Eighth International Conference on Learning Representations* (2020), no. CONF.

[163] YUAN, G., BING, X., AND MENGJIE, Z. An effective one-shot neural architecture search method with supernet fine-tuning for image classification. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2023).

[164] YUAN, G., WANG, B., XUE, B., AND ZHANG, M. Particle swarm optimization for efficiently evolving deep convolutional neural networks using an autoencoder-based encoding strategy. *IEEE Transactions on Evolutionary Computation* (2023).

[165] ZAGORUYKO, S., AND KOMODAKIS, N. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).

[166] ZEILER, M. D., AND FERGUS, R. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014* (Cham, 2014), D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Springer International Publishing, pp. 818–833.

[167] ZHANG, H., JIN, Y., CHENG, R., AND HAO, K. Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance. *IEEE Transactions on Evolutionary Computation 25*, 2 (2020), 371–385.

[168] ZHANG, J., GONG, X., LIU, Y., WANG, W., WANG, L., AND ZHANG, B. Bandit neural architecture search based on performance

evaluation for operation selection. *Science China Technological Sciences* (2023), 481–488.

[169] ZHANG, M., LI, H., PAN, S., CHANG, X., ZHOU, C., GE, Z., AND SU, S. One-shot neural architecture search: Maximising diversity to overcome catastrophic forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence 43*, 9 (2020), 2921–2935.

[170] ZHANG, T., LEI, C., ZHANG, Z., MENG, X.-B., AND CHEN, C. L. P. As-nas: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning. *IEEE Transactions on Evolutionary Computation 25*, 5 (2021), 830–841.

[171] ZHANG, X., HOU, P., ZHANG, X., AND SUN, J. Neural architecture search with random labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2021), pp. 10907–10916.

[172] ZHANG, X., ZHOU, X., LIN, M., AND SUN, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 6848–6856.

[173] ZHAO, Y., WANG, L., TIAN, Y., FONSECA, R., AND GUO, T. Few-shot neural architecture search. In *Proceedings of the 38th International Conference on Machine Learning* (18–24 Jul 2021), M. Meila and T. Zhang, Eds., vol. 139 of *Proceedings of Machine Learning Research*, PMLR, pp. 12707–12718.

[174] ZHONG, Z., YAN, J., WU, W., SHAO, J., AND LIU, C.-L. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 2423–2432.

[175] ZHOU, B., KHOSLA, A., LAPEDRIZA, A., OLIVA, A., AND TORRALBA, A. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016).

[176] ZHOU, D., ZHOU, X., ZHANG, W., LOY, C. C., YI, S., ZHANG, X., AND OUYANG, W. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 11396–11404.

[177] ZHOU, X., QIN, A. K., SUN, Y., AND TAN, K. C. A survey of advances in evolutionary neural architecture search. In *2021 IEEE Congress on Evolutionary Computation (CEC)* (2021), IEEE, pp. 950–957.

[178] ZHOU, Z., NING, X., CAI, Y., HAN, J., DENG, Y., DONG, Y., YANG, H., AND WANG, Y. Close: Curriculum learning on the sharing extent towards better one-shot nas. In *Computer Vision – ECCV 2022* (Cham, 2022), S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds., Springer Nature Switzerland, pp. 578–594.

[179] ZHU, H., AN, Z., YANG, C., XU, K., ZHAO, E., AND XU, Y. Eena: efficient evolution of neural architecture. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops* (2019), pp. 0–0.

[180] ZHU, Y., YAO, Y., WU, Z., CHEN, Y., LI, G., HU, H., AND XU, Y. Gp-cnas: Convolutional neural network architecture search with genetic programming. *arXiv preprint arXiv:1812.07611* (2018).

[181] ZHUANG, H., ZHANG, Y., AND LIU, S. A pilot study of query-free adversarial attack against stable diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (June 2023), pp. 2384–2391.

[182] ZOPH, B., AND LE, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[183] ZOPH, B., VASUDEVAN, V., SHLENS, J., AND LE, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 8697–8710.