Deep Learning-based Image Analysis for High-content Screening

by

Dylon Zhiheng Zeng

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Data Science.

Victoria University of Wellington
2021

Abstract

High-content screening is an empirical strategy in drug discovery to identify substances capable of altering cellular phenotype — the set of observable characteristics of a cell — in a desired way. Throughout the past two decades, high-content screening has gathered significant attention from academia and the pharmaceutical industry. However, image analysis remains a considerable hindrance to the widespread application of high-content screening. Standard image analysis relies on feature engineering and suffers from inherent drawbacks such as the dependence on annotated inputs. There is an urging need for reliable and more efficient methods to cope with increasingly large amounts of data produced.

This thesis centres around the design and implementation of a deep learning-based image analysis pipeline for high-content screening. The end goal is to identify and cluster hit compounds that significantly alter the phenotype of a cell. The proposed pipeline replaces feature engineering with a k-nearest neighbour-based similarity analysis. In addition, feature extraction using convolutional autoencoders is applied to reduce the negative effects of noise on hit selection. As a result, the feature engineering process is circumvented. A novel similarity measure is developed to facilitate similarity analysis. Moreover, we combine deep learning with statistical modelling to achieve optimal results. Preliminary explorations suggest that the choice of hyperparameters have a direct impact on neural network performance. Generalised estimating equation models are used to predict the most suitable neural network architecture for the input data.

Using the proposed pipeline, we analyse an extensive set of images acquired from a series of cell-based assays examining the effect of 282 FDA

approved drugs. The analysis of this data set produces a shortlist of drugs that can significantly alter a cell's phenotype, then further identifies five clusters of the shortlisted drugs. The clustering results present groups of existing drugs that have the potential to be repurposed for new therapeutic uses. Furthermore, our findings align with published studies. Compared with other neural networks, the image analysis pipeline proposed in this thesis provides reliable and better results in a shorter time frame.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Dr. Binh Nguyen, for his time, illuminating ideas, and research guidelines. Conducting this master's research seemed like a daunting task initially due to immense input data. Without my supervisor's generous support and guidance, I could not have finished this research.

A massive thank you to Prof. Ivy Liu and Prof. Philip Morrison for inviting me to work on an analytics project, increasing my knowledge and understanding in the statistics area. Thank you to Kang Wang, who has helped with the design of research methods. I also extend my thanks to my friends and colleagues for their moral support.

Most importantly, my family has my heartfelt gratitude. They have been nothing but supportive of my life choices. Thank you for always keeping faith in me, giving me the mental energy to finish this degree.

Last but not least, I thank all the people who have assisted with my studies at Victoria University of Wellington, including the friendly staff in the School of Mathematics and Statistics, and the Student and Academic Services team in the Faculty of Science office.

I was financially supported by the Wellington Master's by Thesis Scholarship from Victoria University of Wellington, New Zealand.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis centres around the design and implementation of a deep learning-based image analysis pipeline for high-content screening. The tremendous amounts of input data in this research are microscopic images of cells — treated with over 282 FDA approved drugs — from large scale assays. Subcellular structures, including cell membrane and nucleus, are clearly visible. The end goal, or the overall output, is first to produce a shortlist of drugs that significantly modulate cellular phenotype, then cluster the said drugs for repurposing reasons.

1.1 Motivation

In recent years, phenotypic screening [58] has fallen back into favour in the pharmaceutical industry. While target-based screening relies on a predefined disease-modifying target such as proteins [68], phenotypic screening identifies candidate drug compounds without knowing their biological target. In phenotypic screening, compounds are tested in assays to select ones with a desirable therapeutic effect on cells. The main advantage of phenotypic screening is that it broadens the search space by lifting the limitation of having a target, increasing the chance of discovering pioneering drugs. Besides, phenotypic screening could provide direct information on

how a compound affects a disease-relevant phenotype [9].

When conducted on a large scale with fluorescence imaging, advanced microscopy, robotic handling, and image analysis, this type of phenotypic screening is known as high-content screening [36]. First described in a paper by Giuliano (1997) [29], high-content screening has been used extensively in the pharmaceutical industry to increase the scope of data collected from assays. In contrast to traditional high-throughput screening, which has a single read-out averaged over all cells within a microplate well, high-content screening allows simultaneous monitoring of various cellular phenotypes through some form of image analysis [9].

In high-content screening, vast quantities of candidate compounds are tested in a microplate format for their ability to induce phenotypic changes in cell populations. These cells are then visualised by staining subcellular structures, like cell membrane and nucleus, with fluorescent dyes; and captured by automated high-resolution microscopy. Once obtained, the microscopic images are analysed to assess complex spatiotemporal effects of drugs on a cell's particular morphology. A drug compound with the desired impact on cells is called a hit. In this case, the desired impact is significant modulation of phenotype. With the rise of Artificial Intelligence, various parts of high-content screening have been supported by machine learning algorithms [84]. Image analysis, however, remains a major hurdle in high-content screening. Phenotypic changes induced by drug compounds can be highly variable, demanding considerable efforts in identifying them.

Traditional image analysis relies on feature engineering to find drug compounds that modulate phenotype [33]. Simply put, feature engineering is a process to create quantitative variables from the spatial characteristics of image data, such as cell size, shape, nucleus size, intensity, etc. [72]. The features created vary in significance. As a result, feature engineering is closely followed by feature selection — choosing the most relevant subset of features for predictions. In standard practice, the images

of drug-treated cell populations are first segmented into single cells. Then features are created manually, and their values are measured by some carefully designed automated process [33]. Knowing what features to create requires hands-on experience, thorough understanding, and an eye for detail. Moving on to feature selection. The irrelevant features are removed either manually or by a machine learning algorithm. The phenotypic profile of a drug, hence, is a multidimensional measurement of the selected features. Finally, the drugs profiles are compared and clustered to determine anomalies.

When features are well defined, it is easier for machine learning algorithms to pick up on patterns and detect anomalies. As a result, the algorithms can be more concise and accurate. The feature engineering process is so vital to the overall success yet so elusive and circumstantial that some describe it as an art [21]. Therefore, feature engineering is inevitably subject to personal bias and extremely time-consuming to perform right. There is always a need for a more time-efficient unsupervised machine learning approach in image analysis that circumvents feature engineering. To fill this void, we propose a deep learning-based image analysis pipeline for high-content screening.

1.2 Related Works

Most image analysis approaches in high-content screening have followed two paths: supervised profiling with feature engineering and unsupervised clustering with feature extraction. In this context, 'unsupervised' means that features are not predefined. Feature extraction is a dimensionality reduction technique that produces distilled representation of the input [83].

Supervised approaches identify drugs that modulate phenotype using predefined features. Giuliano et al. (2004) [28] used the measurements of multiple features to create phenotypic profiles for 22 drug compounds

and performed hierarchical cluster analysis. On a similar theme, Perlman et al. (2004) [77] quantified phenotypic changes using multidimensional features and profiled the effects of drugs in human cells. Loo et al. (2007) [60] classified treated and untreated cancer cells using a support vector machine with up to 300 features, then used the classification results to create drug profiles.

On the other hand, unsupervised approaches work by clustering intrinsic cellular structures in the input images, mostly using some form of convolutional neural network. Therefore these approaches do not require predefined features. Pawlowski et al. (2016) [75] performed unsupervised feature extraction on microscopy images using convolutional neural networks and achieved an overall classification accuracy of up to 91%. In a similar vein, Zhang et al. (2017) [98] classified cervical cells with convolutional neural networks. By using deep features, Zhang et al. achieved a classification accuracy of 98.3%. Caicedo et al. (2018) [11] went one step further with convolutional neural networks as they proposed to classify each treatment against each other, lifting the requirement for negative control. While effective, this approach had an underlying assumption that most treatments, i.e. drugs, did not induce phenotypic changes. Akram et al. (2019) [4] proposed a convolutional neural network-based framework to detect defects in photovoltaic cells. They achieved a 93.0% accuracy on an electroluminescence image data set.

In recent years, autoencoders have entered the horizon in medical data analysis. The task of autoencoders is to extract refined representations of inputs without feature engineering. Miotto et al. (2016) [67] derived patient representations from health records, such as doctor visits and test results, using autoencoders. Based on 76214 patient representations, Miotto et al. predicted the chances of patients developing 78 diseases. Their results with autoencoders outperformed the predictions based on raw data as well as the predictions based on other feature learning algorithms. Chen et al. (2017) [13] proposed a framework based on convolutional autoen-

coders to analyse computed tomography (CT) images. Lung nodule representations learned by the framework could help assist diagnosis. Results from experiments showed that the framework was superior to other supervised approaches. Chen et al. stated that the framework could also be extended to similarity analysis. Khamparia et al. (2020) [52] used a combination of convolutional and variational autoencoders to detect abnormal cells that might develop into cervical cancer. The use of variational autoencoder further reduced data dimensionality. Their hybrid network achieved an accuracy of up to 99.4%, better than traditional machine learning methods.

Unsupervised feature extraction in microscopic image analysis is still new. Therefore, there is no consensus on optimal practice. Furthermore, the high variability in different image data sets demands a tailored solution for each problem. As a result, the accuracy numbers of neural networks on different data sets cannot be compared directly. In this thesis, an alternative approach to image analysis is presented. Specifically, we propose a deep learning-based image analysis pipeline for high-content screening that circumvents feature engineering. This pipeline is primarily designed for an extraordinary data set comprising images of cells treated with over 282 FDA approved drugs. However, with minor modifications on the neural network such as adjusting input dimensions, the pipeline is also generalisable to other data sets.

1.3 Pipeline Overview

This section gives an overview of the pipeline covered in this thesis, as illustrated in Figure 1.1.

The input data to the pipeline are microscopic images of cell populations — treated with over 282 FDA approved drugs — from large scale assays. Subcellular structures, including cell membrane and nucleus, are clearly visible. These are grayscale images in which only one colour chan-

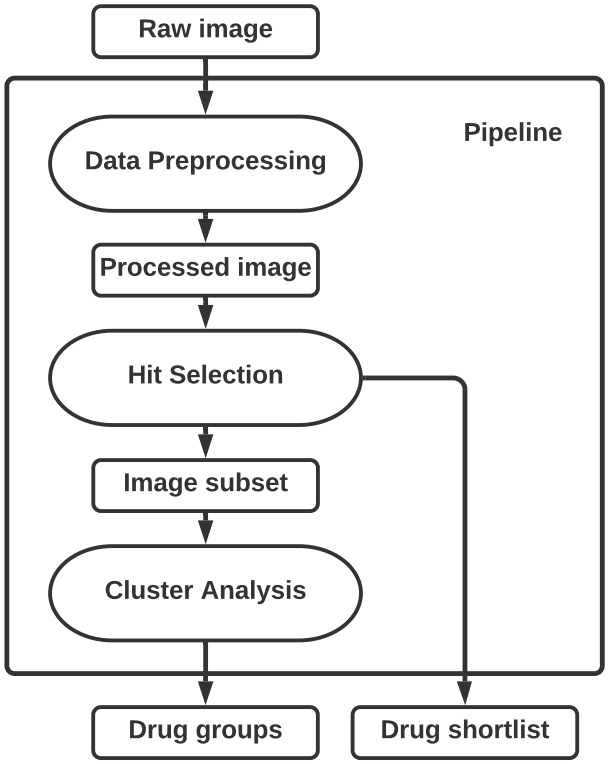


Figure 1.1: A conceptual outline of the image analysis pipeline for high-content screening.

nel exists. Each pixel in a grayscale image has an intensity value, representing the amount of blackness.

There are three key stages in the pipeline. Raw image data are first processed at **Data Preprocessing** to downsize and get cleansed. The processed images then proceed to **Hit Selection** to produce a shortlist of drugs. Finally, images corresponding to the shortlisted drugs — a subset of the processed images — move on to **Cluster Analysis** to identify groups hidden in the shortlist.

There are two outputs from the pipeline. The first one is a shortlist of drugs that significantly modulate cellular phenotype, produced at the end of the hit selection stage. The second one, produced at the end of the

cluster analysis stage, is clustering information of the shortlist indicating groups of drugs that induce similar phenotypic changes. This is valuable information for drug repurposing reasons. Instead of image-wise prediction, we are only interested in group-wise prediction in this research. For example, we want to evaluate the effect of a drug based on a collection of images, rather than an evaluation for each image.

1.4 Issues Addressed

Due to high cell-to-cell variability and noise, unsupervised feature extraction for image analysis can be challenging to achieve. While supervised approaches produce detailed profiles of cellular phenotype for each drug compound based on predefined features, unsupervised learning relies on detecting anomalous drugs (i.e. hits) that are different from the negative control. With increasing computational power, deep neural networks play a more critical role in machine learning as they are, in general, more powerful and more robust algorithms [8]. This section briefly describes the key issues addressed in designing an unsupervised deep learning-based image analysis pipeline for high-content screening.

1.4.1 Data Preprocessing Stage

Data preprocessing is a critical first stage in the proposed pipeline. The later tasks — hit selection and cluster analysis — require well-presented input data to be accurate. Since the raw data are microscopic images of cell populations, they are not ready to proceed to hit selection right away. The assays were conducted in batches; thus, we must first recalibrate pixel values in the images. Cell localisation is the second step. Similar to facial recognition, where faces need to be located before anything else, we need to locate and crop out single cells. Last but not least, cells that are not distinctly visible or not in complete shapes must be removed.

1.4.2 Hit Selection Stage

Hit selection is the second stage in the pipeline and the most important one. In this research, a drug compound with the ability to modulate cellular phenotype significantly is considered a hit. To identify hits, we first need to define non-hits. Fortunately, Dimethyl Sulfoxide (DMSO) — a widely used safe solvent which offered no impact on phenotype [6] [43] — is among the tested drug compounds. DMSO is therefore considered the negative control. Drugs that induce phenotypic changes significantly different from the negative control are flagged as hits.

We propose a hit selection procedure with similarity analysis using unsupervised feature extraction, circumventing the feature engineering process. An effective comparison scheme must be developed to enable similarity analysis. In the proposed pipeline, hit selection is assisted by a convolutional autoencoder — an unsupervised deep neural network. The task of the convolutional autoencoder is to construct condensed representations of the microscopic images. These lower-dimensional representations are compared with the negative control using a novel similarity measure in order to select hits. The purpose we use lower-dimensional representations is threefold [67]:

1. The condensed representations are less susceptible to noise.

2. Intrinsic patterns are more visible in condensed representations.

3. The condensed representations are smaller in terms of storage usage.

In summary, the distilled version of images boosts hit selection accuracy and requires less computational resources.

The hyperparameters used to define an autoencoder architecture must be determined. They are not trivial as the performance of the autoencoder depends closely on their values. A generalised estimating equation model is used to predict the optimal hyperparameters.

1.4.3 Cluster Analysis Stage

The final stage in the proposed pipeline is cluster analysis, where we find clusters among the hit compounds. Once we have a shortlist of drug compounds, we are interested in finding groups that induce similar phenotypic changes. Since the tested drugs are all FDA approved to begin with, those within the same group have the potential to be repurposed. Of course, more experiments and clinical trials must be conducted to make the repurposing decision scientifically sound, but those are out of the scope of image analysis in this pipeline.

In clustering, we must measure the similarity between each pair of hit compounds. Fortunately, the comparison scheme used in the hit selection stage also fits this occasion. Thus, instead of comparing drug compounds to the negative control, we compare them with each other in the shortlist.

1.4.4 Summary

The key issues addressed in the proposed pipeline can be summarised as follows.

1. Data preprocessing stage:
    - Recalibrate intensities from different assay batches;
    - Detect and crop out single-cell structures;
    - Filter out incomplete cells and impurities.
2. Hit selection stage:
    - Develop a novel scheme for comparing representations;
    - Construct a generalised estimating equation model to predict optimal hyperparameters of convolutional autoencoder tailored for the input images;

- Use the convolutional autoencoder to obtain condensed representations of input images;
- Compare representations with the negative control to obtain similarity measures;
- Determine an appropriate similarity threshold at which drugs are considered as hits.

3. Cluster analysis stage:
   - Compare representations of each pair of hit compounds to obtain similarity measures;
   - Select an appropriate approach for cluster analysis;
   - Cluster hit compounds based on the similarity measures.

1.5 Contributions

This thesis contributes in the following ways:

1. This thesis shows how to select an optimal set of hyperparameters for a deep neural network using statistical modelling. Compared with other practices, this method provides more solid scientific grounding. An experiment with labelled data provides empirical evidence on the effectiveness of this method.

2. This thesis presents a novel comparison scheme, including a novel similarity measure, useful in similarity analysis. In general, this comparison scheme can apply to image classification of any kind. A key feature of this scheme is that it is essentially nonparametric, meaning there is no assumption about the input. Explorations show that, when used as an image classifier, the convolutional autoencoder-aided comparison scheme outperforms common neural networks by a wide margin.

3. This thesis explores the properties of convolutional autoencoders with empirical evidence. Results show that autoencoders are useful in dimensionality reduction. However, they are not as useful in cluster analysis on their own. Results also show that the encoder component is less susceptible to overfitting.

4. This thesis presents the design and the implementation of a deep learning-based image analysis pipeline for high-content analysis. A key feature of this pipeline is that it does not require feature engineering, saving a great amount of time and human efforts while producing better results. After testing, the pipeline is applied to an immense and intricate image data set to select drugs that have the potential to be repurposed. The results of this research can ultimately help discover new therapeutic uses for existing drugs.

1.6 Thesis Outline

The rest of this thesis is structured as follows.

- ***Chapter 2*** reviews aspects of statistical modelling and machine learning relevant to this research, emphasising generalised estimating equations and convolutional autoencoder.

- ***Chapter 3*** discusses the data preprocessing stage where images are transformed into a standard format compatible with the neural network used.

- ***Chapter 4*** discusses the hit selection stage where drug compounds that induce significant phenotypic changes are identified using a novel convolutional autoencoder-aided comparison scheme.

- ***Chapter 5*** discusses the cluster analysis stage where shortlisted drugs are clustered into groups based on the way they alter cellular pheno-

type. The clustering results help discover drugs that have the potential for repurposing.

- ***Chapter 6*** briefly concludes the content of this thesis.

Chapter 2

Background

This chapter reviews aspects of statistical modelling and machine learning relevant to this research. The emphasis is on generalised estimating equations and convolutional autoencoder.

2.1 Linear Regression

Linear regression is a statistical technique that models a continuous dependent variable (Y) — the outcome — as a linear function of one or more independent variables (X). The purpose of regression models is to summarise data or predict the results of interventions.

2.1.1 Types of Outcome

Independent Outcomes

In statistics, two events are independent if the occurrence of one does not depend on the occurrence of the other [81]. The independence of events A and B can be written as

$$P(A \; and \; B) = P(A) \times P(B), \tag{2.1}$$

where P is the notation for probability.

The independence of outcomes is a common assumption in research, meaning that the outcome from a sample subject is independent of those from other subjects. While independence is a statistical assumption, its implementation relies on the correct conduct of the experiment. Thus, reasonable experimental control should be in place to ensure that observations on the same variable do not influence each other.

Correlated Outcomes

For various reasons, correlated outcomes are collected in many kinds of research. Two outcomes are correlated if not independent, that is

$$P(A \text{ and } B) \neq P(A) \times P(B). \tag{2.2}$$

Correlated outcomes tend to be similar. Outcomes are correlated if they are repeated observations of a variable over time from the same subject, as in longitudinal studies, or observations of multiple variables from the same subject. Outcomes from different research subjects may also be correlated if they are from the same group, such as the heights of family members. Statistical inferences are only valid if the research method accounts for any correlation among outcomes.

2.1.2 Linear Regression Model Structure

Predicting the continuous outcome (Y) by the values of one or more explanatory variables (X), sometimes called predictors, is intuitive and straightforward logic. Linear regression has this prediction property embedded and assumes that X and Y's relationship is linear. Depending on the circumstances, one can enhance the predictive capability of a model by adding more explanatory variables. The GPA of a student, for example, may depend on hours studied, number of courses taken, GPA from the previous year etc. A linear regression model with only one explanatory

variable is called simple linear regression; the one with several explanatory variables is called multiple linear regression [25]. The multiple linear regression model takes the standard form as in Eq. (2.3).

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i \tag{2.3}$$

For each subject i, $i = 1, \ldots, n$, y_i is the continuous outcome given $\mathbf{x}_i$; x_{ij} is the j^{th} explanatory variable, $j = 1, \ldots, p$; β_j is the j^{th} unknown parameter describing the strength of association between x_{ij} and y_i; ε_i represents the residual, the difference between true and model-predicted values of y_i (i.e. $\varepsilon_i = y_i - \mathbf{x}_i^T \boldsymbol{\beta}$), $E(\varepsilon_i) = 0$; $\mathbf{x}_i^T$ denotes the transpose of vector $\mathbf{x}_i$, $\mathbf{x}_i^T = [1, x_{i1}, x_{i2}, \ldots, x_{ip}]$.

For a linear regression model to work properly, there are four assumptions to be satisfied [25]:

1. Linearity: The true relationship between any explanatory variable x_{ij} and conditional outcome y_i is linear.

2. Normality: Each conditional outcome y_i follows a normal distribution [74] (means of these distributions are not necessarily the same), implying that each ε_i is also normally distributed. There is no assumption on the distribution of observed outcomes Y (i.e. $Y = \{y_1, y_2, \ldots, y_i\}$).

3. Independence: The observations, especially the outcomes, from different research subjects are independent. This is an underlying assumption for linear regression, e.g. y_i's are independent of each other.

4. Homoscedasticity: The residuals have the same finite variance across all values of x_{ij}.

In addition to the assumptions above, researchers must also beware of multicollinearity and outliers. Multicollinearity arises when the correlation between explanatory variables is high, and it will compromise the

validity of model inferences. In the case of multicollinearity, a common remedy is to drop one of the correlated predictors. Outliers are also harmful to linear regression models, leading to inaccurate predictions for a wide range of predictor values. It is a general practice to inspect and remove extreme outliers before model fitting.

$\hat{\boldsymbol{\beta}}$ — the estimator of model parameter vector $\boldsymbol{\beta}$ — is found by minimising the sum of squared residuals [51]. In other words, $\hat{\boldsymbol{\beta}}$ solves the following minimisation problem.

$$\text{Find } \min_{\boldsymbol{\beta}} Q(\boldsymbol{\beta}), \text{ for } Q(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - \hat{\mathbf{x}}_i^T \hat{\boldsymbol{\beta}})^2,$$

where Q is the objective function.

2.2 Generalised Linear Models

Generalised linear models (GLMs) are a generalisation of regression models that alleviates the requirement for normality on the conditional distribution of outcome by introducing a link function [71]. These models are flexible and convenient, accommodating to a wide range of research circumstances while keeping the familiar structure of linear regression.

2.2.1 Link function

In reality, the conditional distribution of outcome is not always normal. GLMs assume that, given a set of predictor values, the outcome follows one of the distributions from the exponential family [5], such as normal, binomial, gamma, and Poisson. GLMs connect the vector of explanatory variables $\mathbf{x}_i$ with the conditional mean of outcome μ_i, i.e. $\mu_i = E(y_i)$, through an additional monotonic function — the link function g. A monotonic function is either ever-increasing or ever-decreasing as the value of independent variable increases. Eq. (2.4) below expresses this relationship.

$$g(\mu_i) = \mathbf{x}_i^T \boldsymbol{\beta} \tag{2.4}$$

Various link functions are available based on the characteristics of data and the assumed conditional distribution of outcome [64]. Some examples are displayed in Table 2.1.

Table 2.1: Common link functions.

Model	Distribution of y_i	Link	$g(\mu_i)$
Linear Regression	Normal	Identity	μ_i
Logistic Regression	Binomial	Logit	$\ln\left(\frac{\mu_i}{1-\mu_i}\right)$
Gamma GLM	Gamma	Log	$\ln \mu_i$
Gamma GLM	Gamma	Inverse	μ_i^{-1}
Poisson Regression	Poisson	Log	$\ln \mu_i$

2.2.2 GLM Model Structure

GLMs take the standard form shown in Eq. (2.4). There is no residual term because it has no standard form. In linear regression, residuals are normally distributed with zero mean and are independent of any explanatory variables. Whereas in GLMs, residual depends closely on both the distribution and the value of predictors. A GLM fitted on binary data [15], for example, has residual in the form of probability. Bounded by 0 and 1, this limited range of probability suggests that the residual's distribution inevitably depends on the predicted probability. The lack of comparability makes residual less informative in GLMs than in linear regression models. Consequently, homoscedasticity is no longer an assumption for GLMs.

In Eq. (2.4), the transformed conditional mean $g(\mu_i)$ is linearly related to explanatory variables. It is the conditional mean μ_i, rather than the conditional outcome y_i itself, that gets transformed. Therefore, a generalised

linear model, e.g. $\ln \mu_i = \mathbf{x}_i^T \boldsymbol{\beta}$, is not the same thing as a linear regression model with transformed outcome, e.g. $\ln y_i = \mathbf{x}_i^T \boldsymbol{\beta}$. An exception is when the assumed conditional distribution of outcome is normal and g is the identity link, as shown in Eq. (2.5), then linear regression becomes a special case of GLMs. In this regard, Eq. (2.3) and Eq. (2.5) describe the same model.

$$E(y_i) = \mu_i = \mathbf{x}_i^T \boldsymbol{\beta} \tag{2.5}$$

For each subject i, $i = 1, \ldots, n$, μ_i is the expected value of conditional outcome y_i; $\mathbf{x}_i^T$ denotes the transpose of predictor vector $\mathbf{x}_i$, $\mathbf{x}_i^T = [1, x_{i1}, x_{i2}, \ldots, x_{ip}]$; $\boldsymbol{\beta}$ is the vector of unknown parameters.

For a generalised linear model to work properly, there are three assumptions to be satisfied [20]:

1. Linearity: The true relationship between any explanatory variable x_{ij} and transformed conditional mean of outcome $g(\mu_i)$ is linear.

2. Normality: Each conditional outcome y_i follows the same distribution from the exponential family (parameters of these distributions are not necessarily the same). There is no assumption on the distribution of observed outcomes Y (i.e. $Y = \{y_1, y_2, \ldots, y_i\}$).

3. Independence: The observations, especially the outcomes, from different research subjects are independent. This is an underlying assumption for linear regression, e.g. y_i's are independent of each other.

Similar to multiple linear regression, researchers must also beware of multicollinearity and outliers in GLMs. Multicollinearity and outliers are briefly discussed in Section 2.1.2.

The estimation of parameters is usually done on computers because it uses an iterative procedure called maximum likelihood estimation [70]. This procedure repeats until successive estimates change by a small enough

amount. Intuitively, $\hat{\beta}$ maximises the likelihood of obtaining the observed outcomes. Therefore, a large amount of data is required for reliable predictions.

2.3 Generalised Estimating Equations

Generalised estimating equations (GEEs) take the generalisation idea of GLMs one step further — they can be thought of as an extension of GLMs to correlated outcomes. GEEs on their own are not regression models but a technique for estimating model parameters. In this subchapter, we refer the term 'GEEs' to the regression models solved by GEEs.

2.3.1 Motivation

The distilling of useful information from raw data into prediction models has become effortless thanks to increasing computational power. The validity of such models, however, still hinges on the basic assumptions to hold true.

A fundamental assumption of GLMs is that the observed outcomes are independent of each other, which is not always guaranteed. For instance, a nature reserve ranger tries to model the predator and prey dynamics and records the antelope population (predictor) and the lion population (outcome) every year. Lions have a long lifespan. The number of lions this year depends heavily on the number from last year. Therefore the low variability in predator population reduces the informative sample size. Although the model parameter estimator will remain unbiased in this case, the variance estimator will be unreliable [65]. Fitting GLMs to correlated outcomes is likely to overstate the effect of predictors, resulting in more false positives [82], i.e. an effect deemed significant when it is not.

However, correlated data are still useful data. In general, the correlation in outcomes is determined by the structure of the experiment. It is

hard to eradicate once data are collected. Repeating an experiment is often out of the equation because data collection is a slow and costly task. At the same time, correlated outcomes carry helpful information that could improve prediction accuracy. In GLMs, more data points help the algorithm choose a solution that is the most representative of the data and the least affected by noise. In some scenarios, especially when explanatory variables are categorical, insufficient data points could fail to estimate the desired model. Thus, it is better to account for correlation rather than to work around it.

2.3.2 Working Correlation Structure

Generalised estimating equations (GEEs), as proposed by Liang and Zeger (1986) [59], are designed for data of correlated and clustered nature. Specifically, research subjects are independent of each other but observations within each subject are correlated. The research subject can be an individual or a group of individuals with common characteristics. For example if we measure mental wellbeing in a neighbourhood, results from the same household are likely to be correlated because family members tend to have similar wellbeing. Results from different households, however, are still independent. GEEs account for the within-subject correlation by using a working correlation structure $R(\alpha)$, where α is a set of parameters characterising $R(\alpha)$. Although the true correlation may vary, the same $R(\alpha)$ is assumed for all research subjects.

The correlation structure $R(\alpha)$ is referred to as a 'working' correlation matrix by Liang and Zeger because it does not have to be true for the estimators of model parameters and variances to be consistent [59]. This means, even if the working correlation structure $R(\alpha)$ is misspecified, GEEs provide consistent estimates as long as the regression model for marginal mean is correct and the sample size goes to infinity. However, infinite sample size is not achievable in practice. These estimators will be-

come increasingly inefficient, i.e. large variance, and untrustworthy as $R(\alpha)$ gets further away from the true correlation structure. As a result, reliable inferences still depend on a reasonable choice of $R(\alpha)$. Some widely used correlation structures are shown in Table 2.2, where R_{ab} denotes the $(a,b)^{th}$ element of $R(\alpha)$.

Table 2.2: Common working correlation structures.

Structure	Definition	Example
Independent	$R_{ab} = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Exchangeable	$R_{ab} = \begin{cases} 1 & a = b \\ \alpha & a \neq b \end{cases}$	$\begin{pmatrix} 1 & \alpha & \alpha \\ \alpha & 1 & \alpha \\ \alpha & \alpha & 1 \end{pmatrix}$
Unstructured	$R_{ab} = \begin{cases} 1 & a = b \\ \alpha_{ab} & a \neq b \end{cases}$	$\begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & 1 & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & 1 \end{pmatrix}$
Autoregressive-1	$R_{ab} = \begin{cases} 1 & a = b \\ \alpha^{\lvert a-b \rvert} & a \neq b \end{cases}$	$\begin{pmatrix} 1 & \alpha & \alpha^2 \\ \alpha & 1 & \alpha \\ \alpha^2 & \alpha & 1 \end{pmatrix}$

2.3.3 GEE Model Structure

The generalised estimating equations (GEEs) extend the generalised linear models to correlated outcomes while keeping the familiar idea of link function. GEEs model the marginal mean of outcome, rather than the joint distribution of outcomes. To be more precise, GEEs estimate the average outcome for all subjects across the population who share common values of the explanatory variables [97], rather than the subject-specific outcome. GEEs arise from quasi-likelihood [95] principles which only require the

marginal mean to be correctly specified given a link function and predictors, and its variance to be specified given the marginal mean. As a result, GEEs do not postulate distributional assumptions.

This section borrows notation and expression extensively from Liang and Zeger (1986) [59] and Wang (2014) [93]. To begin, suppose m is the maximum number of observed outcomes for each subject. For each subject i, $i = 1, \ldots, n$, $\mathbf{y}_i^T = [y_{i1}, y_{i2}, \ldots, y_{im}]$ where y_{it} denotes the t^{th} outcome with the expectation $E(y_{it}) = \mu_{it}$, $t = 1, \ldots, m$; $\boldsymbol{\mu}_i$ is the vector of marginal means, $\boldsymbol{\mu}_i^T = [\mu_{i1}, \mu_{i2}, \ldots, \mu_{im}]$; $\mathbf{x}_{it}^T$ denotes the transpose of corresponding t^{th} predictor vector, $\mathbf{x}_{it}^T = [1, x_{it1}, x_{it2}, \ldots, x_{itp}]$. Let g be a link function and $\boldsymbol{\beta}$ be the vector of unknown model parameters. GEEs have a standard form for the marginal mean μ_{it} similar to GLMs, as shown in Eq. (2.6).

$$g(\mu_{it}) = \mathbf{x}_{it}^T \boldsymbol{\beta} \tag{2.6}$$

The variance of y_{it} is written as

$$var(y_{it}) = \nu(\mu_{it})\phi^{-1} \tag{2.7}$$

where $\nu(\mu_{it})$ is the variance function in terms of the marginal mean and ϕ is a scale parameter. $\nu(\mu_{it})$ and ϕ depend on the data type of outcomes. For continuous outcomes, $\nu(\mu_{it}) = 1$ and ϕ represents the error variance. For binary outcomes, $\nu(\mu_{it}) = \mu_{it}(1 - \mu_{it})$ and ϕ is estimated based on overdispersion. For count outcomes, $\nu(\mu_{it}) = \mu_{it}$ and ϕ is estimated based on overdispersion. Let $R(\alpha)$ be the working correlation structure characterised by α, the covariance matrix for $\mathbf{y}_i$ is given by

$$V_i(\alpha) = A_i^{\frac{1}{2}} R(\alpha) A_i^{\frac{1}{2}} \phi^{-1}. \tag{2.8}$$

where A_i is a diagonal matrix with $\nu(\mu_{it})$ as the t^{th} element. $V_i(\alpha)$ is identical to $cov(\mathbf{y}_i)$ if the working correlation structure is indeed true. Finally, the estimator of parameters $\hat{\boldsymbol{\beta}}$ solves the estimating equation as in Eq. (2.9).

$$U(\boldsymbol{\beta}) = \sum_{i=1}^{n} D_i^T V_i(\hat{\alpha})^{-1}(\mathbf{y}_i - \mu_i) = 0 \tag{2.9}$$

where $D_i = \frac{\partial \mu_i}{\partial \boldsymbol{\beta}}$, and $\hat{\alpha}$ is a consistent estimator of α. If $V_i(\hat{\alpha})$ is properly specified, then $E[U(\boldsymbol{\beta})] = 0$ and $cov[U(\boldsymbol{\beta})] = D_i^T V_i^{-1} D_i$ are both asymptotically true. In that case, $U(\boldsymbol{\beta})$ resembles a score function, i.e. the derivative of log-likelihood with respect to the parameter vector, ensuring consistent parameter and variance estimators.

GEEs have milder assumptions compared to GLMs, they are [20]:

1. The marginal mean of outcome μ_{it} in terms of a link function g is a linear function of explanatory variables $\mathbf{x}_{it}$. This is the most crucial assumption of GEEs.

2. The variance of marginal mean is a function of marginal mean, as described in Eq. (2.7). A variance function is assumed. However, this variance function does not have to be true for the parameter estimator and the variance estimator to be consistent. It assumes the marginal distribution, but not the full joint distribution.

3. Outcomes are correlated within a subject but independent between subjects. A within-subject correlation structure is assumed. However, this correlation structure does not have to be true for the parameter estimator in the mean model and the variance estimator to be consistent, using the sandwich method.

Considering that consistent estimators are more efficient when the effective sample size is large, GEEs require a large quantity of independent subjects or independent clusters. Similar to multiple linear regression, researchers must also beware of multicollinearity and outliers in GEE models. Multicollinearity and outliers are briefly discussed in Section 2.1.2.

Due to the sheer amount of computation involved, GEEs are usually solved by an iterative procedure on computers.

2.3.4 Model Selection

Model selection in GEEs refers to a three-step process corresponding to the assumptions:

1. Select a regression model for the marginal mean μ, i.e. the model described in Eq. (2.6). This step is the most crucial since the consistency of parameter and variance estimators ultimately depends only on a correctly specified regression model [20]. Therefore, one should carefully inspect the data and choose a link function suitable for the outcome data type and the characteristics of the data. Furthermore, the transformed marginal mean has to be a linear function of the explanatory variables. The choice of explanatory variables affects not only the performance of the regression model but also all downstream inferences. How to select significant explanatory variables is further elaborated on later in this section.

2. Select an appropriate variance function, i.e. $\nu(\mu)$ in Eq. (2.7). A correctly specified $\nu(\mu)$ increases the efficiency of the whole selection process [19]. Instead of a complete distribution, only the relationship between marginal mean and its variance needs to be specified. Similar to link function, the choice of $\nu(\mu)$ is usually determined by the outcome data type and the characteristics of the data [94]. Therefore, it is the most straightforward selection of the three. Zeger and Liang (2014) [96] described that, although less efficient, an incorrect choice of $\nu(\mu)$ still produces consistent parameter and variance estimators.

3. Select an appropriate correlation structure, i.e. $R(\alpha)$ as discussed in Section 2.3.2. Although a misspecified working correlation structure still leads to consistent parameter and variance estimators, a correctly specified one enhances the efficiency of these estimators [50]. When the sample size is small, the interpretation of variance becomes more trustworthy as the correlation structure gets closer to

the true one. And variance, in turn, plays a huge role in the significance test for the predictors. There are several selection criteria available for the correlation structure, such as RJ [79], ESS [87], QIC [73], and CIC [39] to name a few. The selection is straightforward using these criteria, either jointly or separately.

In this section, the selection of explanatory variables is discussed in more detail. Considering that GEEs is an approach based on quasi-likelihood principles that do not explicitly specify the full likelihood, all likelihood-based model selection methods are unavailable for GEEs. There is no consensus on how to perform model selection, as in choosing the significant predictors. Some suggest that QIC is also suitable for model selection, but its performance is not well when neither the predictors nor the correlation structure is confirmed [16] [93] [17].

Backward Elimination

A logical solution to this problem is backward elimination [3], the simplest yet reliable model selection method. Compared to those selection methods with a single goodness-of-fit statistic for the whole model, backward elimination is a more intuitive algorithm that considers all relevant effects of candidate variables and removes the most insignificant ones early on. Another advantage is that backward elimination can be made fully automated by an algorithm.

A flowchart for this algorithm is demonstrated in Figure 2.1. Backward elimination starts with a full model, which includes all candidate predictors as well as the possible interactions of these predictors depending on circumstances. Then, all these terms are tested for significance and the most insignificant one is deleted. The model is updated and refitted. This process repeats until all terms remaining are significant at desired significance level [78]. For better interpretability, it is a general practice to keep all lower-order terms in the model regardless of their statistical signifi-

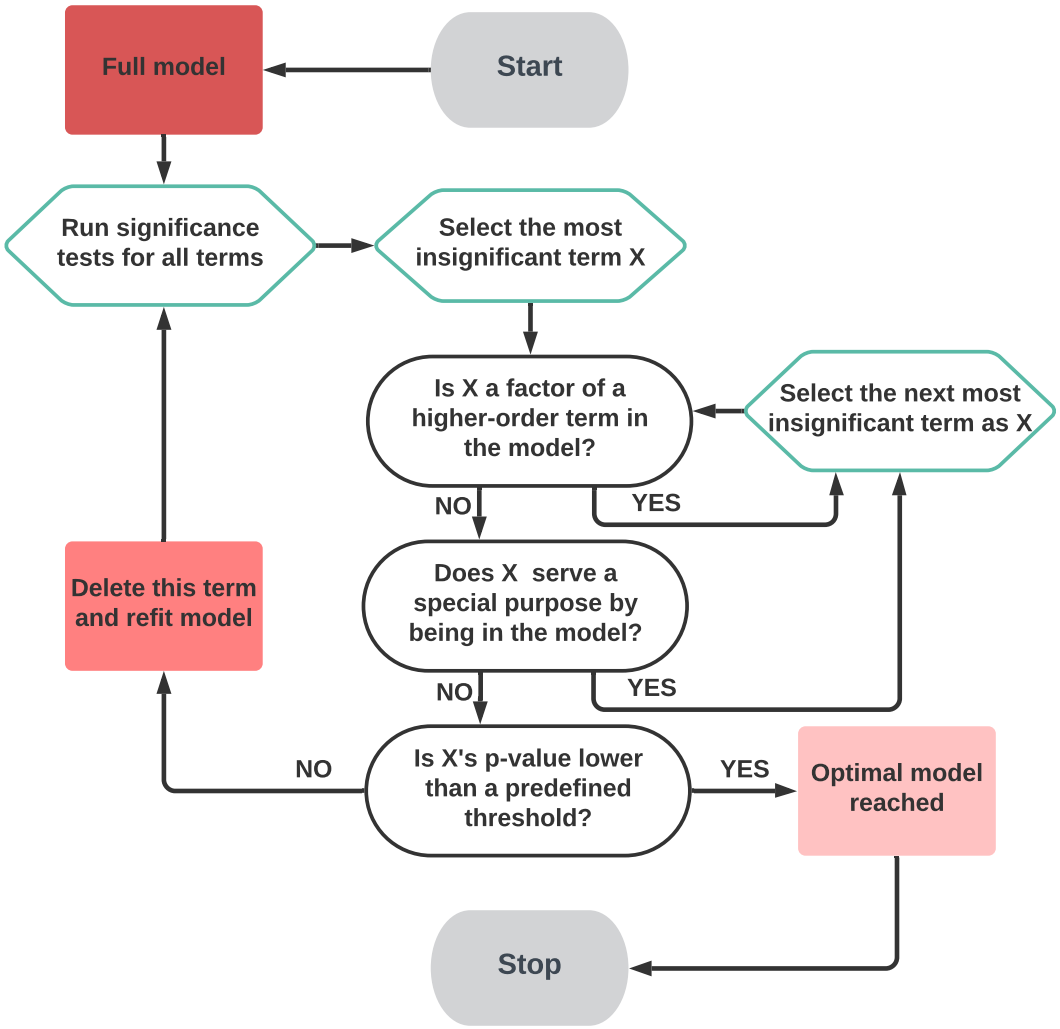


Figure 2.1: A flowchart of the backward elimination algorithm.

cance if they are a factor of a significant higher-order term. For example, A, B, C, AB, BC, AC are factors of the term ABC, however AD is not a factor. It is also advisable to keep the terms meaningful to the research purposes in the model, even when they are deemed insignificant by tests. If a researcher tries to identify the relationship between sunlight exposure and skin cancer, these two variables should be included in the model regardless. Sometimes, the insignificance is a finding in itself.

Test of Significance

In a significance test, a null hypothesis — which usually claims there is no relationship between variables — is assessed by measuring how well the observed data match this claim [69]. The test result is given by a probability value, i.e. p-value, which is a continuous number from 0 to 1 describing the chances of obtaining the observed data as a random occurrence if the null hypothesis is in fact true. For example, a p-value of 0.05 suggests that the probability of wrongly rejecting a null hypothesis is 5% [38]. In practice, 0.05 is a common threshold for p-value — any number larger than 0.05 shows statistical evidence that supports the null hypothesis.

Wald test [91] is a common significance test used in regression model selection. For a null hypothesis $H_0 : \beta_x = 0$, implying the predictor x is not a significant factor in the outcome, the Wald statistic W is defined as

$$W = \frac{(\hat{\beta}_x - 0)^2}{\widehat{var}(\hat{\beta}_x)} \tag{2.10}$$

if x is numerical or binary. $\hat{\beta}_x$ is the parameter estimator for x and $\widehat{var}(\hat{\beta}_x)$ is the variance estimator of $\hat{\beta}_x$. Here, W follows an asymptotic χ^2 distribution [34] with one degree of freedom if the null hypothesis is true. If $\boldsymbol{x}$ is categorical with more than two levels, W can be written as

$$W = (\hat{\boldsymbol{\beta}}_{\boldsymbol{x}} - 0)^T [\widehat{cov}(\hat{\boldsymbol{\beta}}_{\boldsymbol{x}})]^{-1} (\hat{\boldsymbol{\beta}}_{\boldsymbol{x}} - 0) \tag{2.11}$$

where $\hat{\boldsymbol{\beta}}_{\boldsymbol{x}}$ is the estimator of parameter vector $\boldsymbol{\beta}_{\boldsymbol{x}}$ associated with the levels of $\boldsymbol{x}$, and $\widehat{cov}(\hat{\boldsymbol{\beta}}_{\boldsymbol{x}})$ denotes the covariance matrix estimator of $\hat{\boldsymbol{\beta}}_{\boldsymbol{x}}$ associated with the levels of $\boldsymbol{x}$. Under null hypothesis, this statistic W follows an asymptotic χ^2 distribution with d degrees of freedom where d is the number of nonredundant parameters in $\boldsymbol{\beta}_{\boldsymbol{x}}$ [2].

2.3.5 Result Analysis

Following model selection, fitting the best model yields parameter and variance estimates for the significant predictors. A typical research objective is to determine whether intervention on predictors significantly alters model outcome. For GEEs, this objective is realised by comparing two marginal means of outcome derived from different predictor values or, more specifically, by examining confidence intervals.

Confidence Interval

A confidence interval provides a range of more likely estimates of the fixed and unknown parameter. To be especially precise, a 95% confidence interval means that if an experiment is repeated multiple times and a confidence interval is constructed for each experiment, in the long run, 95% of these intervals will capture the true parameter [18]. A 95% confidence interval for a parameter β in a GEE model is written as

$$\left(\hat{\beta} - t_{2.5\%,n-k}[\widehat{var}(\hat{\beta})]^{\frac{1}{2}} \ , \ \hat{\beta} + t_{2.5\%,n-k}[\widehat{var}(\hat{\beta})]^{\frac{1}{2}}\right),$$

where $\hat{\beta}$ is a point estimator of β based on the observed data, $t_{2.5\%,n-k}$ is the critical value of Student's t-distribution [23] with $n-k$ degrees of freedom and a 95% confidence level (2.5% two-tailed), k is the number of parameters in the model including intercept, $\widehat{var}(\hat{\beta})$ is the variance estimator of $\hat{\beta}$. A common rule of thumb is that when the degree of freedom df is at least 30, $t_{2.5\%,df}$ can be approximated by the critical value of a standard normal distribution with mean 0 and variance 1, or $z_{2.5\%}$ [54]. The lower and the upper limits of this confidence interval satisfy

$$P\left(\hat{\beta} - t_{2.5\%,n-k}[\widehat{var}(\hat{\beta})]^{\frac{1}{2}} \ \leq \ \beta \ \leq \ \hat{\beta} + t_{2.5\%,n-k}[\widehat{var}(\hat{\beta})]^{\frac{1}{2}}\right) = 0.95, \quad (2.12)$$

where P is the notation for probability [18].

Comparing Marginal Means

When determining whether two marginal means are significantly different in GEEs, a common pitfall is to construct one confidence interval for each marginal mean and check for overlap. While this procedure is easy to perform, it is not statistically accurate. The correct approach is to examine if the confidence interval of their difference includes zero [31]. Considering that the link function is monotonic, it is easier to compare the transformed marginal means instead of the marginal means themselves for the same result. Suppose we have two vectors of predictor values $\mathbf{x}_1$ and $\mathbf{x}_2$. The 95% confidence interval of the difference between the transformed marginal means, i.e. $g(\hat{\mu_1}) - g(\hat{\mu_2})$, takes the following form

$$\left(M - t_{2.5\%,n-k}[\widehat{var}(M)]^{\frac{1}{2}} \ , \ M + t_{2.5\%,n-k}[\widehat{var}(M)]^{\frac{1}{2}}\right),$$

where $M = \mathbf{x}_1^T\hat{\boldsymbol{\beta}} - \mathbf{x}_2^T\hat{\boldsymbol{\beta}} = (\mathbf{x}_1 - \mathbf{x}_2)^T\hat{\boldsymbol{\beta}}$, $\hat{\boldsymbol{\beta}}$ is the estimator of parameter vector $\boldsymbol{\beta}$, and $\widehat{var}(M)$ is the variance estimator of M. $\widehat{var}(M)$ can then be calculated as in Eq. (2.13), where $\widehat{cov}(\hat{\boldsymbol{\beta}})$ is the covariance matrix estimator of $\hat{\boldsymbol{\beta}}$ [20].

$$\widehat{var}(M) = \widehat{var}(\mathbf{x}_1^T\hat{\boldsymbol{\beta}} - \mathbf{x}_2^T\hat{\boldsymbol{\beta}}) = (\mathbf{x}_1 - \mathbf{x}_2)^T\widehat{cov}(\hat{\boldsymbol{\beta}})(\mathbf{x}_1 - \mathbf{x}_2) \tag{2.13}$$

The decision rule is simple: if this confidence interval includes zero, the marginal means derived from $\mathbf{x}_1$ and $\mathbf{x}_2$ are not significantly different at a 5% (i.e. 1-95%) significance level; if this confidence interval does not include zero, the marginal means derived from $\mathbf{x}_1$ and $\mathbf{x}_2$ are significantly different at a 5% significance level. By examining confidence intervals in this manner, researchers are able to answer research questions such as whether smoking increases the chance of developing lung cancer.

2.4 Machine Learning

Machine learning is one of the hottest research topics and is incredibly prevalent in data-abundant industries. Machine learning is about designing computer algorithms that can learn from experience and improve problem-solving ability automatically [27].

2.4.1 Definitions

Machine Learning

Machine learning is generally considered a subfield of Artificial Intelligence that progressively parses data and makes informed decisions for a given intellectual task, including pattern identification, speech recognition, image classification etc. [55]. Machine learning algorithms are usually autonomous problem-solving tools that can 'learn' from a training data set without being explicitly programmed, then independently translates this 'knowledge' to actions or predictions for new cases. For instance, the autocorrect feature on computers and smartphones — guessing the correct word you mean to type — has machine learning behind it. The more you type and the more you use autocorrect, the better its suggestions become. Such suggestions are made by finding distinct typing patterns based on past inputs. More specifically, this results from complex decision-making systems that adapt themselves every time a suggestion is accepted or denied. In machine learning, the decision-making process is not programmed to be static but dynamic and data-driven through adjusting algorithm parameters [85]. Furthermore, mathematics and statistics play a huge role in machine learning, where parameters are in the hundreds if not thousands [61]. Therefore, state-of-the-art techniques are usually employed to enhance performance, including but not limited to mathematical optimisation, probability analysis, and cluster analysis.

Deep Learning

Deep learning is a subfield of machine learning. In deep learning, algorithms are organised in multiple layers called an artificial neural network [48]. Consecutively, each layer builds on the output of the previous layer and serves a different function. The artificial neural network structure mimics the biological neural network of a human brain, resulting in advanced learning capability and superior problem-solving power [8]. In general, deep learning algorithms require a larger data set for training. However, once properly trained, deep learning usually outperforms basic machine learning by a considerable margin [8] [27].

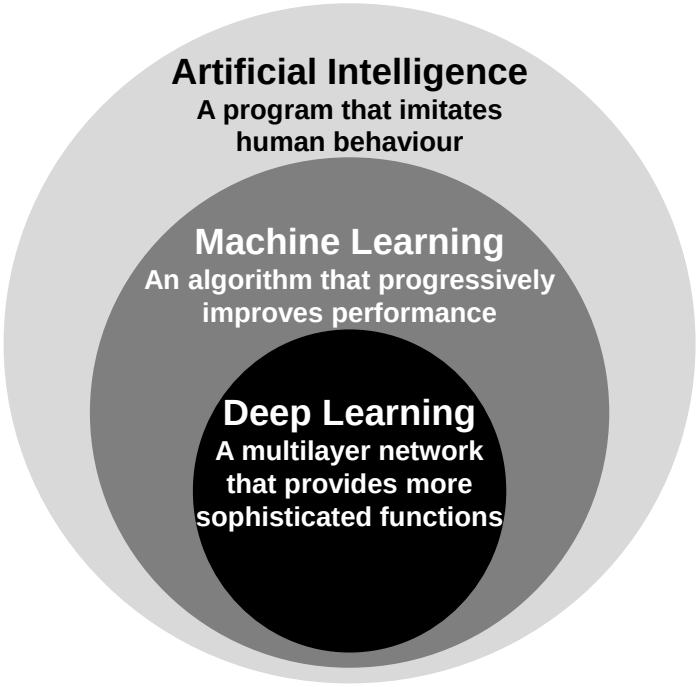


Figure 2.2: Overview of Artificial Intelligence. Machine learning and deep learning are both subfields of Artificial Intelligence. Deep learning is also a subfield of machine learning.

2.4.2 Categories of Machine Learning

There are a few broad categories machine learning algorithms can fit into, based on the type of input data and the nature of task involved. This section briefly describes one classification approach related to human guidance.

Supervised Learning

In supervised learning, the input data fed to the learning algorithm include the desired output [27]. Typical supervised learning tasks include classification and regression. Take image classification as an example, described in Figure 2.3. Animal pictures and the correct label of the pictures, such as 'dog' or 'horse', are put into an algorithm as training data. This algorithm then tries to develop a mapping function from pictures to correct animal labels. For each prediction, the algorithm compares its output to the given label and adjust accordingly. Here, the labels act as human guidance to teach this algorithm about the expected solution. Supervised learning typically demands immense labelled data, which can be a luxurious resource on a large scale.

Unsupervised Learning

Contrary to supervised learning, unsupervised learning is applied when data is unlabelled. Meaning that the desired output is not available to guide decision-making [27]. Such algorithms rely on clustering data and making predictions accordingly. A typical application of unsupervised learning is personalised product recommendation. As shown in Figure 2.4, if a customer has purchased a flight ticket in recent history, there is a higher chance that this individual is also interested in a travel insurance. Unsupervised learning can make personalised recommendations based on purchase history.

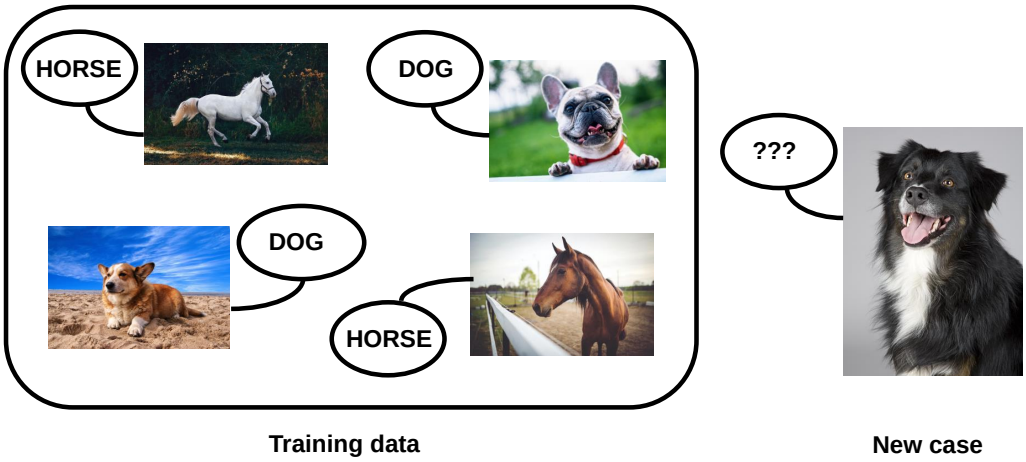


Figure 2.3: An example of a supervised learning task. The machine learning algorithm is taught to classify new cases into known clusters. (image source: Pexels)

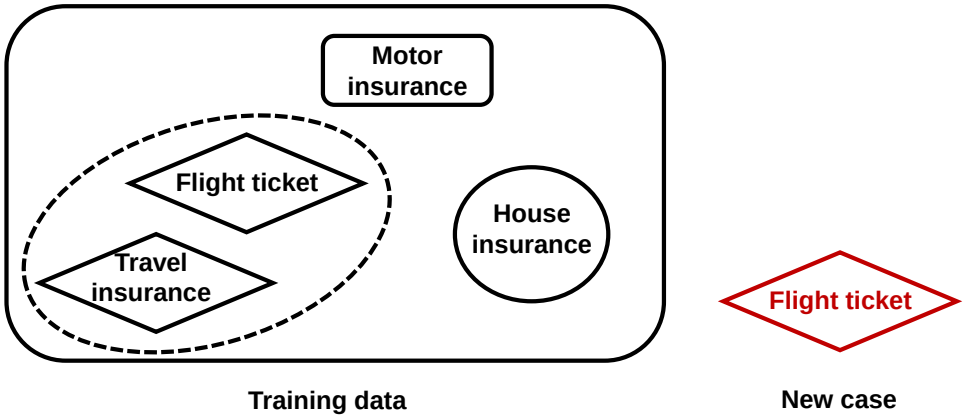


Figure 2.4: An example of an unsupervised learning task. The machine learning algorithm groups available products into clusters and makes product recommendations based on clustering results.

Reinforcement Learning

Reinforcement learning is preferred over supervised/unsupervised learning when the objective is to make a sequence of circumstantial decisions

[27]. This system, called an agent, differs from other learning approaches in that it gets a reward or a penalty each time an action is performed. Then, in an iterative manner, the agent learns from the current environmental state and the reward/penalty received before updating its policy (i.e. a set of rules for selecting action in a given environmental state) and selecting an action based on the policy. The end goal is to maximise reward over time. This process is demonstrated in Figure 2.5. Typical applications of reinforcement learning include chess-playing program, self-balance walking robot, and dialogue creation.

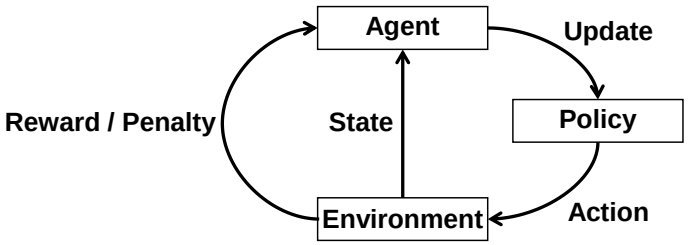


Figure 2.5: An overview of reinforcement learning. The machine learning agent observes the environmental state and selects an action based on current policy. Then, subject to the reward or the penalty received for this action, the agent updates its policy and select a new action accordingly. This process iterates until an optimal policy is reached.

2.5 Basic Autoencoder

An autoencoder is an unsupervised deep learning neural network that aims to recreate its input as output through an information bottleneck. Autoencoders are particularly useful in feature extraction or dimensionality reduction [41]. In this section, we briefly discuss the basic principles behind an autoencoder.

2.5.1 Neuron

The simplest computational unit of a neural network is a neuron which is a mathematical analogue to brain neurons [32]. Neurons act in parallel to form a layer. Layers work in series to form a neural network. Each neuron in the neural network receives input from neurons in the previous layer, and its output, in turn, proceeds to neurons in the next layer. This type of connection is kept between layers. Neurons within the same layer typically do not connect. Figure 2.6 shows how a neuron processes information.

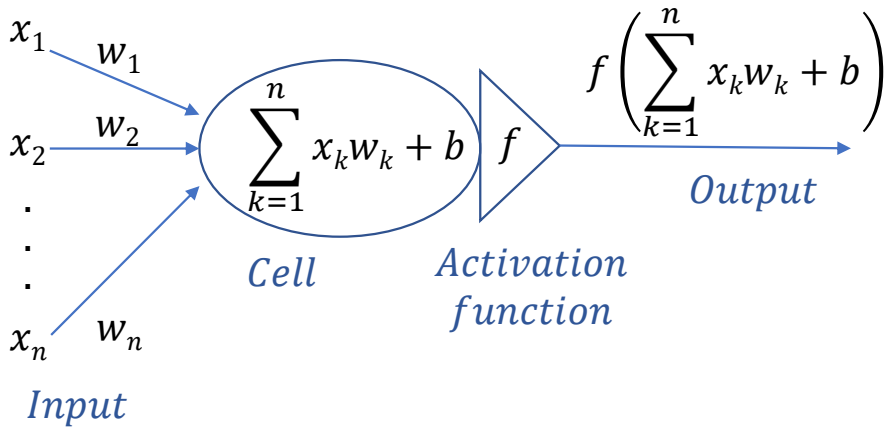


Figure 2.6: The mathematical model of a single neuron in a neural network. Adapted from Karpathy (2015) [49].

In Figure 2.6, information flows from input to output without circling back, referred to as a feed-forward network [49]. First, the neuron receives input data x_k from multiple different neurons in the previous layer and multiplies them by weights w_k assigned to each input respectively. Then, bias b is added to the sum of products before an activation function f is applied to transform this sum. Activation functions are later discussed in the next section. The final output proceeds to feed other neurons in the next layer. Neurons 'learn' by adjusting weights w_k to produce a favorable output.

2.5.2 Activation Function

An activation function is applied to each neuron at the very end. It can be considered as a final modification to the output in order to introduce nonlinearity, which helps the neural network learn complex features [40]. Due to the linear nature of the sum of weighted input in a neuron, it is sometimes difficult to model nonlinear patterns without an activation function. Consider a model with a single neuron, i.e. $y = xw + b$. In the absence of an activation function, the output y is always on a straight line once the optimal weight $\hat{w}$ and bias $\hat{b}$ are found. An activation function also functions as a threshold, keeping output values restricted to the desired range [40]. In a deep neural network, the total number of model parameters, including weights and biases, can be in the millions. An activation function with a range from zero to one, for example, can keep the output from skyrocketing. Examples of common activation functions are rectified linear units (ReLU) [30], scaled exponential linear units (SELU, with default values $\lambda = 1.0507$, $\alpha = 1.6733$) [53], sigmoid, and tanh. Their definitions are written below as in Eq. (2.14-2.17). The shape of these activation functions are displayed in Figure 2.7.

$$ReLU(x) = max(0, x) \tag{2.14}$$

$$SELU(x) = \begin{cases} \lambda\alpha(exp^x - 1) & x < 0 \\ \lambda x & x \geq 0 \end{cases} \tag{2.15}$$

$$sigmoid(x) = \frac{1}{1 + exp^{-x}} \tag{2.16}$$

$$tanh(x) = \frac{exp^x - exp^{-x}}{exp^x + exp^{-x}} \tag{2.17}$$

The choice of activation function can also affect training efficiency significantly. During neural network training, the change in objective function with respect to model parameters are computed by the chain rule in each iteration. This computation starts from the last layer and iterates

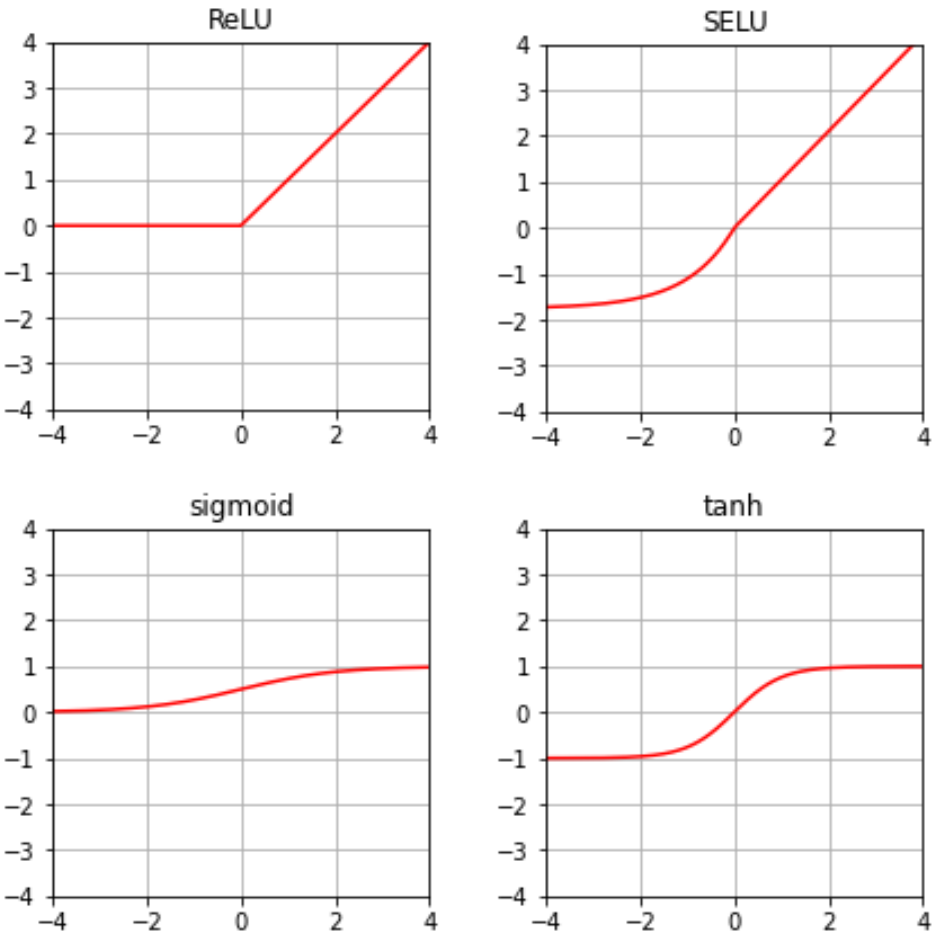


Figure 2.7: Shape of common activation functions.

backwards, resulting in a gradually diminishing gradient across layers if the activation function has a gradient range between zero and one [42]. The vanishing gradient problem, where training stops prematurely due to diminished gradients and static parameters, worsens when the neural network has many layers. This problem is further elaborated on in Section 2.5.4. Glorot et al. (2011) [42] concluded that ReLU can prevent the vanishing gradient problem as its linearity helps the gradients flow well through the layers. Starting with ReLU in network construction has become a rule of thumb in deep learning. Other activation functions are only preferred

circumstantially.

2.5.3 Basic Autoencoder Architecture

Autoencoder is a type of neural network that comprises multiple layers of parallel neurons. The idea behind autoencoder is to force high-dimensional input data through an information bottleneck, transforming it into a low-dimensional representation in the process, and reproduce the input data from that representation [41]. By mapping the input onto a space of reduced dimensionality, the neural network is able to learn and extract a variety of features. The architecture of an autoencoder is defined by its hyperparameters — the preset values that cannot be updated during training [32]. Hyperparameters of an autoencoder include: the activation function of neurons, the number of filters in each layer, the total number of layers, and the loss function. An example of autoencoder performing feature extraction is illustrated as in Figure 2.8.

In Figure 2.8, every circle represents a neuron. The latent vector $\boldsymbol{B}$, or the bottleneck information, is the result of input vector $\boldsymbol{X}$ ($\boldsymbol{X}^T = [x_1, x_2, x_3, x_4]$) mapped onto a latent space through a series of interconnecting neurons called an encoder. Subsequently, the decoder — a mirror version of the encoder — maps $\boldsymbol{B}$ back onto the same space as $\boldsymbol{X}$. The reconstruction of $\boldsymbol{X}$ is $\boldsymbol{X}_o$ ($\boldsymbol{X}_o^T = [x_{o1}, x_{o2}, x_{o3}, x_{o4}]$). Hence an autoencoder can be abstractly described as Eq. (2.18-2.19), where En denotes the encoding process and De denotes the decoding process.

$$En(\boldsymbol{X}) = \boldsymbol{B} \tag{2.18}$$

$$\boldsymbol{X}_o = De(\boldsymbol{B}) \tag{2.19}$$

The autoencoder progressively learns and selects essential characteristics of the input, or features, by minimising a reconstruction loss which is closely related to the difference between $\boldsymbol{X}$ and $\boldsymbol{X}_o$ [41].

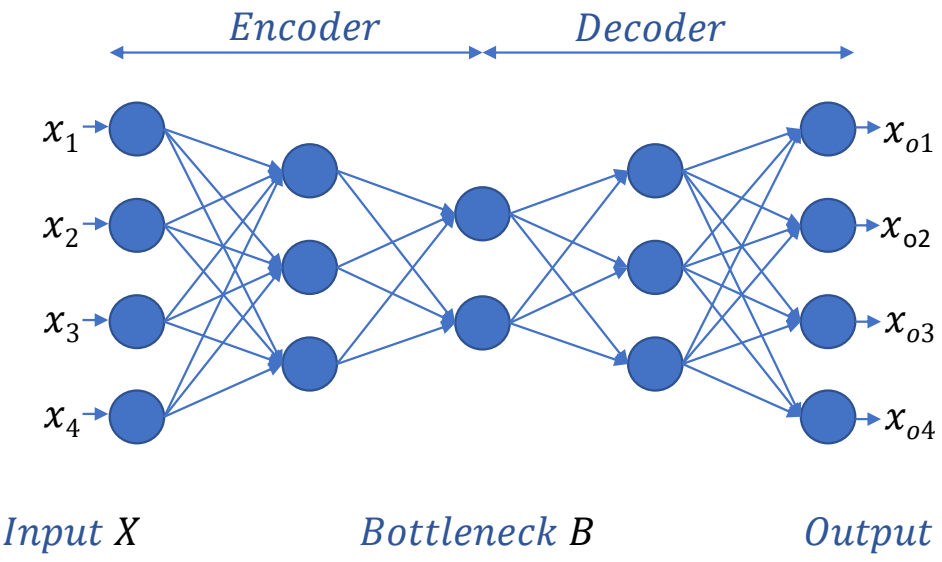


Figure 2.8: An example of autoencoder architecture. The neural network consists of an encoder and a decoder. The goal is to minimise the reconstruction loss, defined as a function of input $\boldsymbol{X}$ and output $\boldsymbol{X}_o$ which represents the difference in information between the two.

In this basic setting, layers are fully connected. The term 'fully connected' refers to a connection arrangement where each neuron is connected to all the neurons in its adjacent layers. Neurons within the same layer, however, are not connected. The hourglass shape structure is particularly crucial in autoencoder. Without forcing such an information bottleneck, the autoencoder could have learned to ignore the compression process entirely and copy information from input directly to output for the smallest possible reconstruction loss. Consequently, the bottleneck $\boldsymbol{B}$ would have carried roughly the same amount of information as the input $\boldsymbol{X}$, defeating the purpose of feature extraction.

In essence, the training of an autoencoder can be described as the following minimisation problem.

$$\text{Find } \min_{\boldsymbol{m}} L(\boldsymbol{X}, \boldsymbol{X}_o),$$

where m represents all the model parameters (weights and biases) and L is the desired objective function, also called loss function. The choice of loss function L is circumstantial. Some of the widely used ones are mean squared error, mean absolute error, and cross-entropy [32]. Therefore L should be chosen in an integrated manner, giving attention to the nature of problem, the objective of research, and the type of data. Ideally when loss function is minimised, $\boldsymbol{X}$ and $\boldsymbol{X}_o$ become almost identical. Nonetheless, the output will never be exactly the same as the input. Since some information, no matter how trivial, is inevitably lost while squeezing through the bottleneck. A 100% restoration would have implied that there is no dimensionality reduction inside the encoder. Once properly trained, the bottleneck $\boldsymbol{B}$ becomes a lower-dimensional representation of $\boldsymbol{X}$.

As a dimensionality reduction technique, autoencoders help filter out non-important information, leaving the essential features most representative of the input. The potential applications are vast.

2.5.4 Backpropagation

The model parameter estimators have no closed forms in a neural network. Moreover, the enormous number of parameters means that solving by exhaustion is not practical. Therefore, their optimal values are found by an iterative process. During the training of a feed-forward neural network, parameters are iteratively updated by an optimisation algorithm called backpropagation [80]. Backpropagation works by passing on gradient information from the last layer backwards to the first one. A simple example is illustrated in Figure 2.9.

In Figure 2.9, the neural network has three layers: input layer $\boldsymbol{a}$, hidden layer $\boldsymbol{b}$, and output layer $\boldsymbol{c}$. To find out how to adjust the parameters, Rumelhart et al. (1986) [80] suggested the use of partial derivatives and

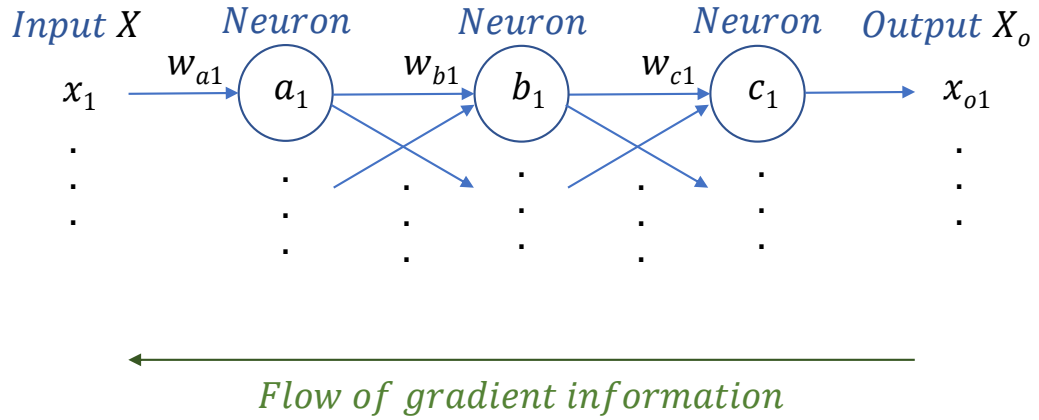


Figure 2.9: An example of backpropagation. Input information propagates from left to right, whereas gradient information propagates backward from right to left. Hence the name backpropagation.

the chain rule. To demonstrate his ideas, we look at a simple example. Suppose L is the loss function of input $\boldsymbol{X}$ and output $\boldsymbol{X}_o$, then the update in weight w_{c1} of neuron c_1 in the last layer $\boldsymbol{c}$ can be expressed by

$$w_{c1}^{i+1} = w_{c1}^{i} - \eta \frac{\partial L}{\partial w_{c1}},$$

where η is the learning rate, w_{c1}^{i} is the current weight, w_{c1}^{i+1} is the new weight, and the partial derivative $\frac{\partial L}{\partial w_{c1}}$ is the gradient of L with respect to w_{c1}. Recall that, described in Section 2.5.1 and Section 2.5.3, loss function L is a function of the output $\boldsymbol{X}_o$, whose component can be expressed in terms of an activation function and a linear function of weighted inputs and bias. Therefore, by the chain rule, the gradient $\frac{\partial L}{\partial w_{c1}}$ can be further written as

$$\frac{\partial L}{\partial w_{c1}} = \frac{\partial L}{\partial \boldsymbol{X}_o} \frac{\partial \boldsymbol{X}_o}{\partial x_{o1}} \frac{dx_{o1}}{dA_{c1}} \frac{dA_{c1}}{dS_{c1}} \frac{\partial S_{c1}}{\partial w_{c1}},$$

where A_{c1} and S_{c1} are the activation function and linear function for neuron c_1 respectively, notation d represents total derivative.

The neuron c_1 is related to only one of the output components, i.e. x_{o1}. For a neuron in the second last layer, b_1, its weights affect multiple components of the output. Hence the update of one of its weight w_{b1} can be expressed by

$$w_{b1}^{i+1} = w_{b1}^{i} - \eta \frac{\partial L}{\partial w_{b1}},$$

where

$$\frac{\partial L}{\partial w_{b1}} = \sum_{i=1}^{n} \frac{\partial L}{\partial \boldsymbol{X}_o} \frac{\partial \boldsymbol{X}_o}{\partial x_{oi}} \frac{dx_{oi}}{dA_{ci}} \frac{dA_{ci}}{dS_{ci}} \frac{\partial S_{ci}}{\partial A_{b1}} \frac{dA_{b1}}{dS_{b1}} \frac{\partial S_{b1}}{\partial w_{b1}},$$

and n is the number of neurons in the next layer $\boldsymbol{c}$ that are connected to b_1. To update the weights of a neuron in this second last layer, we need to use the product of two gradients (e.g. $\frac{dA_{c1}}{dS_{c1}}$ and $\frac{dA_{b1}}{dS_{b1}}$). Image a deep neural network with 100 layers. We would need the product of 100 gradients to update the weights in the first layer. If the gradients have a range between zero and one, the change in loss function with respect to the weights in the first layer will become minuscule. This phenomenon is called a vanishing gradient problem.

Learning rate η indicates how large a step the weight takes in the right direction. If η is too large, the precision of the optimisation algorithm is reduced, possibly resulting in a constant back-and-forth motion around the optimal value. If η is too small, the training time of neural network is prolonged. Therefore in some cases, η is set as an adaptive value based on a convergence measure for the loss function. The training process usually halts when the convergence measure is small enough [32].

There are three optimisation approaches to implement backpropagation: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent [32].

1. Batch gradient descent calculates the loss function for each training sample but only updates model parameters after all samples in the training data set are evaluated. The parameters are updated based

on an aggregated gradient. This approach is considered computationally efficient as parameters are adjusted once per epoch — a complete evaluation cycle through the training data set. Due to less frequent updates, convergence is more stable. However, it may take a long time to converge for large data sets.

2. Stochastic gradient descent, contrarily, calculates loss function and updates model parameters for each training sample in the data set. This approach is effective in reaching global minima yet computationally expensive, which may lead to a longer training time. In addition, this stochastic approach also fluctuates the optimisation process because frequent updates amplify the impact of noise.

3. Mini-batch gradient descent finds a middle ground between batch and stochastic gradient descent, where training samples are split into small batches before employing batch gradient descent to each batch. Mini-batch is the most common approach among the three as it incorporates the benefits of the other two. To use mini-batch gradient descent, one must first choose a batch size. A rule of thumb is to select a power of two to fit the memory size of hardware. For example, 16, 32, 64, 128 are common choices of batch size. Small batch size converges quickly but is noisy. Large batch size converges slowly but more stable. Masters et al. (2018) [63] and Bengio (2012) [7] both suggested that 32 is a good default value. Once selected, it is advisable to keep batch size fixed while optimising model parameters.

2.6 Convolutional Autoencoder

A convolutional autoencoder is a variant of the basic autoencoder where a convolutional layer is used. This type of neural network preserves the spatial patterns of data, therefore it is preferable when dealing with images [62] [22]. The core constituents of a convolutional autoencoder are the

convolutional layer and the pooling layer.

2.6.1 Convolutional Layer

The convolutional layer is usually the first layer in convolutional neural network (CNN) [26] [56]. In CNN, input data such as images are assumed to possess spatial structures which cannot be fully appreciated using standard fully connected layers. A convolutional layer, on the other hand, extracts information locally and therefore retains the pattern.

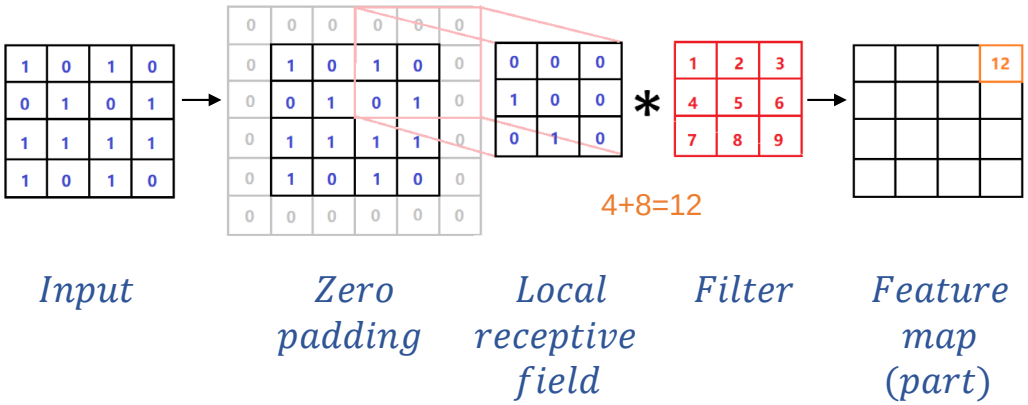


Figure 2.10: An illustration of how an unactivated feature map is made. The local receptive field and the filter are multiplied element by element. In this case, 12 is the sum of all elements in the product matrix; therefore, it is put in the top right corner of the feature map corresponding to the position of local receptive field.

In a convolutional layer, convolutional kernels known as filters are administered. A filter is a small weight matrix with the same number of channels as the input image. It moves across the input image, striding one pixel at a time, extracting a patch of input with the same dimensions as the filter. This patch, called a local receptive field, is then multiplied element-wise by the filter. Finally, the elements of the product matrix are

summed to yield a convolved output value representing a pixel in the unactivated feature map. This procedure repeats as the filter shifts through the whole input image until all pixels in the feature map are filled [62]. Padding the input image with ring(s) of zero pixels along its edge ensures that the output image is of the same size as the input. Once the feature map is complete, a bias is added to each feature map before it gets activated by an activation function. The activated feature map then proceeds to the next layer. Figure 2.10 shows a filter at work.

As each feature map has shared filter weights and bias, a convolutional layer has much fewer parameters than a fully connected layer [35]. In essence, a filter is trained to extract key features in the input image which are the most useful in reconstructing it. Depending on the weights, a filter can act as an edge detector, a corner detector, etc.

2.6.2 Pooling Layer

Closely following a convolutional layer is a pooling layer. The primary function of pooling layers is to perform pooling operation and reduce dimensionality. The concepts of filter and local receptive field apply in pooling layers. However, unlike convolutional layers, the filters have no trainable weights and the local receptive fields are not overlapping at all [62]. The untrainable filters in a pooling layer usually serve one of two purposes: finding the maximum or finding the average. Figure 2.11 demonstrates both max and average pooling with the same input.

Pooling layers are essential in a convolutional autoencoder as they downsamples the input and thus enables the signature information bottleneck. Furthermore, on account of lower-dimensional output, pooling layers further reduce the overall trainable parameters in the neural network. This forces the filters to be more applicable.

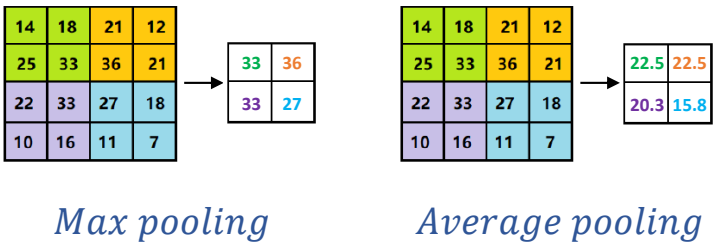


Figure 2.11: Illustrations of max pooling and average pooling. In max pooling, the maximum value of each local receptive field is selected. In average pooling, the average of each local receptive field is selected.

2.6.3 Convolutional Autoencoder Architecture

In summary, a convolutional autoencoder is nothing more than an assembly of alternating convolutional layers and pooling layers. In the decoder, where dimensionality needs to increase, upsampling layers replace pooling layers. Upsampling layers repeat each row and column of the input to upsize it.

Intuitively, the number of filters is the number of features that a neural network can potentially learn. Feature maps become more abstract after each convolution, showing fewer features. Thus, for better convergence, a rule of thumb is to reduce the number of filters in a convolutional layer deeper in the encoder.

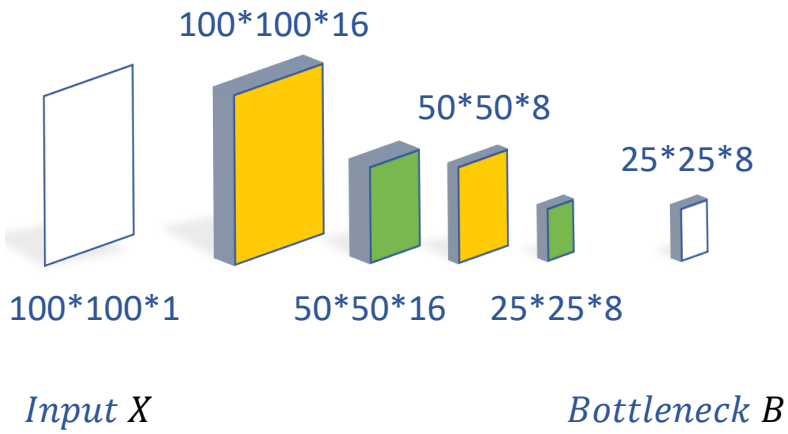
Training deep neural networks can be challenging, especially when initial weights are random. The ever-changing weights trigger instability in the distribution of inputs to layers deep in the neural network, forcing the optimisation algorithm to adapt indefinitely. To address this problem, Ioffe and Szegedy (2015) [45] proposed a technique called batch normalisation. In batch normalisation, input data are standardised using the mean and the standard deviation of each batch to pursue a stable data distribution passing on to the next layer. This operation is written as in Eq. (2.20), where x is the input value, x_{norm} is the normalised value, μ and σ are the

mean and the standard deviation of a given batch of inputs respectively.

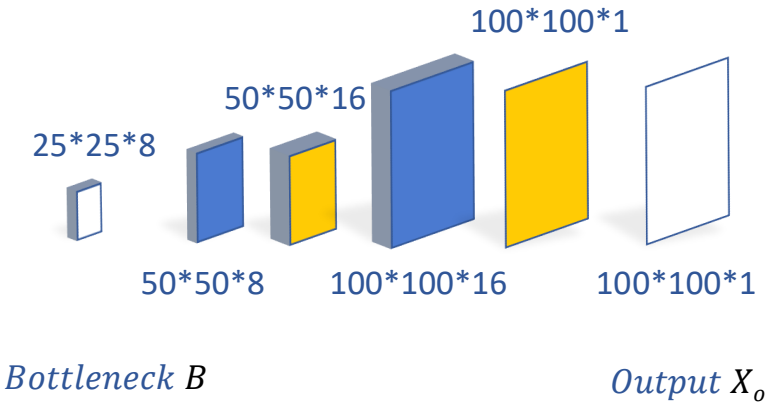
$$x_{norm} = \frac{x - \mu}{\sigma} \tag{2.20}$$

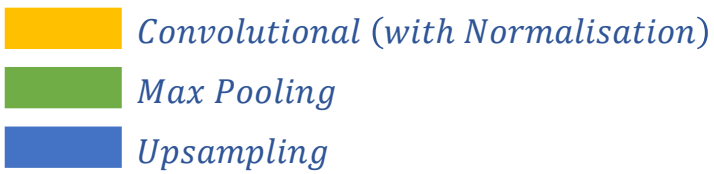
Normalising the inputs has a significant effect in reducing the number of epochs required in training. Ioffe et al. (2015) [45] found that convolutional layers with batch normalisation helped reduce the number of training steps by half while maintaining the same level of accuracy. Both Ioffe et al. (2015) [45], and Kaiming He, et al. (2015) [37] suggested that the batch normalisation operation should be placed right before the nonlinearity. Pooling layers have no trainable weights and thus are not in need of stable distribution of inputs. Therefore, batch normalisation can be an integral part of a convolutional layer right before the activation function.

Figure 2.12 shows the architecture of an example convolutional autoencoder with two convolutional layers in its encoder component. In this example, the input image $\boldsymbol{X}$ is 100 by 100 in size and has one colour channel, thus 100*100*1. A convolutional layer with 16 filters processes $\boldsymbol{X}$ and produces a 100*100*16 output feeding into a max pooling layer. The pooling layer downsizes the image into 50*50*16 without reducing the number of feature maps. Another pair of convolutional/pooling layers with 8 filters further condenses the image into a 25*25*8 bottleneck $\boldsymbol{B}$. Compared with the original image $\boldsymbol{X}$, which has 100*100*1=10000 data points, this bottleneck $\boldsymbol{B}$ has 5000 data points — half of the initial value. Therefore $\boldsymbol{B}$ is a condensed representation of $\boldsymbol{X}$. Opposite operations happen inside the decoder. The bottleneck $\boldsymbol{B}$ is first upsampled by an upsampling layer. Then it passes on to a convolutional layer with 16 filters, mirroring the position of its counterpart in the encoder. Following that is another pair of upsampling and convolutional layers with 1 filter to resconstruct the image back to 100*100*1. The final output is $\boldsymbol{X}_o$, with the same dimensions as input $\boldsymbol{X}$. At first glance, the encoder and the decoder may seem unbalanced due to different numbers of filters. In fact, they are perfectly balanced in terms of trainable parameters. Since we assume filter number



(a)





(b)

Figure 2.12: Architecture of an example convolutional autoencoder. (a) The encoder component. Two convolutional layers with 16 and 8 filters respectively. 1320 trainable parameters in total. (b) The decoder component. Two convolutional layers with 16 and 1 filter respectively. 1313 trainable parameters in total.

decreases deeper into the encoder, and the decoder has a mirrored structure, the two non-trainable hyperparameters left are the layer number and the filter number in the first convolutional layer.

There are two major differences between convolutional neural network (CNN) and convolutional autoencoder (CAE). CAE has an information bottleneck, and CNN has fully connected layers before final output [47] [62].

2.6.4 Challenges

The rapid development of Artificial Intelligence and machine learning technologies presents enormous opportunities. General analytical tasks, such as pattern recognition and classification, can be accomplished more effectively and at a lower cost using neural networks. Applications of AI-powered products and services will be numerous in traditional and modern industries alike. While neural networks are efficient and effective in replacing human workforce, they still face several major challenges. This section briefly discusses some of these challenges in deep neural networks.

Selection of hyperparameters

Deep neural networks have hyperparameters that define their architecture. Unfortunately, there is no universal solution as to how to select these hyperparameters [88]. This problem is especially noticeable with unsupervised learning, where there is no validation of performance due to the lack of labels. As a result, researchers have been relying on labelled benchmark data sets to evaluate the performance of neural networks. The reality, however, is that a neural network that works fine with one data set may struggle with another one. This is because both the choice of neural network and its hyperparameters are circumstantial. For example, 32 filters may be too many for clustering handwriting digits, but it is not enough for clustering animal pictures. Like there are multiple solutions to a problem,

multiple neural networks can serve the same purpose with similar accuracy as long as the hyperparameters are set appropriately. This calls for a practical and efficient hyperparameter selection approach. We elaborate on this matter in the next chapter.

Interpretability

Deep neural networks are infamously challenging to interpret. The interpretability worsens with unlabeled data. Some describe them as a black-box approach, and adequately so. Even when weights and feature maps are on hand, the predictions made by a neural network are not easily interpretable in terms of logic [89]. Ideally, a user should be able to tell which features do the filters capture or the logic behind selecting these filters. Sadly, the plausibility of such predictions still relies almost solely on our trust in the learning algorithm. Sometimes, even neural network designers cannot understand the theoretical grounding of the predictions. Thus, there is an urging need for accurate and transparent learning algorithms in high-stakes sectors such as criminal justice, banking and healthcare.

Data preprocessing

Well-presented input data is key to machine learning success. Data preprocessing quality affects the performance of neural networks significantly. One of the reasons researchers often rely on benchmark data sets for evaluation is that benchmark data sets are well preprocessed. However, in real-world scenarios, that luxury is no longer there. Since real-world data can be pretty unpredictable, preprocessing the data and making them compatible with existing neural networks is a tall order. This, again, is particularly difficult in unsupervised learning such as clustering, where validation is not possible.

2.7 Chapter Summary

This chapter briefly reviewed the key concepts used in this research and the scientific foundations upon which they are built. We refer to the content of this chapter later in this thesis. Starting from the next chapter, we present the proposed pipeline in three sequential stages: data preprocessing, hit selection, and cluster analysis.

Chapter 3

Data Preprocessing Stage

This chapter discusses the first stage in the pipeline — data preprocessing. The content of this chapter includes an overview, preliminary explorations, the final design, results, and a brief summary.

3.1 Data Preprocessing Overview

Data preprocessing is the crucial first stage in the pipeline, as shown in Figure 3.1. The 61560 raw images are 2048*2048*1 in dimensions, meaning there are 2048 pixels in height and width and there is only one colour channel (grayscale). These are 16-bit images. Therefore, pixel intensity has a range from 0 to $2^{16} - 1$ (or 65535). As shown in Figure 3.2, each image contains cell populations treated by a given drug the identity of which we do not know. However, we are given the identity of drug 0, which is the negative control Dimethyl Sulfoxide (DMSO). Raw images are organised under 282 FDA approved drugs, each with varying numbers of images. The total storage size of raw images is around 500 GB. Considering the assays are conducted in batches, we need to recalibrate the pixel intensities so that all images share the same zero background intensity. To do that, we assume the lowest intensity of all images in each batch is the background intensity. Background intensity is a benchmark used in recalibration. Then

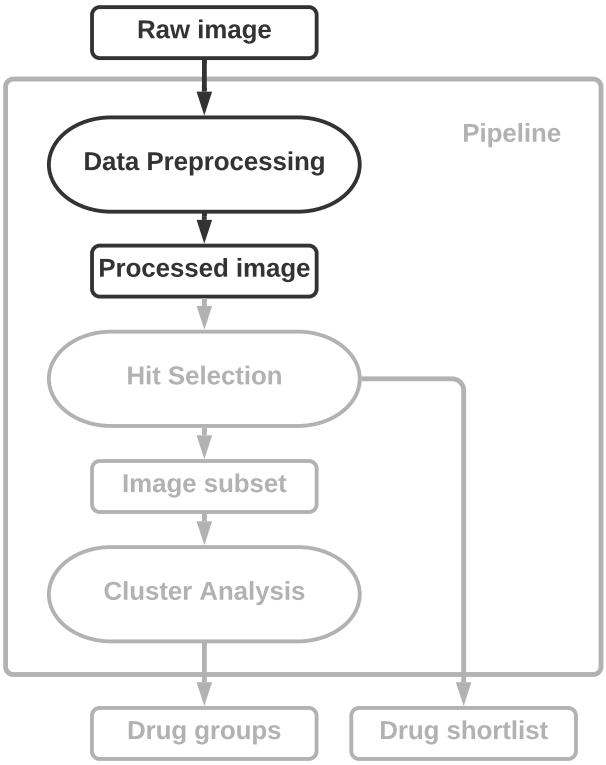


Figure 3.1: Data preprocessing is the first stage in the proposed pipeline.

single-cell structures are detected and cropped out. These cropped out images are referred to as 'windows' in this chapter. The size of the windows should be large enough to contain one cell comfortably. Finally, windows that contain an incomplete cell are no good for the research, thus deleted. At the end of this stage, the desired outputs are windows of clear and complete single cell ready to be passed to the hit selection stage.

Three issues are addressed in this stage:

1. Recalibrate intensities from different assay batches;

2. Detect and crop out single-cell structures;

3. Filter out incomplete cells and impurities.

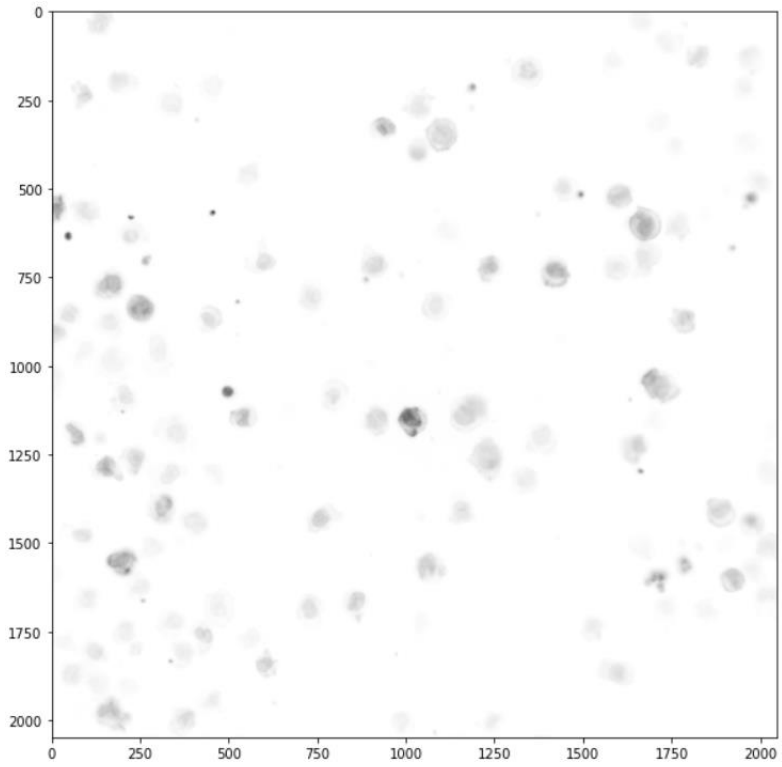


Figure 3.2: An example of raw data. The 2048*2048 image shows cell population after treatment.

Assumption made in data preprocessing:

1. The lowest intensity of all images in each batch is the background intensity.

3.2 Preliminary Explorations

This section describes some of the necessary explorations of data and methods. The explorations provide a better demonstration of the task as well as a collection of considerations behind the choices made in the final design.

3.2.1 Intensity Recalibration

The first task in the line is intensity recalibration. We avoid calling it 'normalisation for batches' here because batch normalisation is a different technique entirely. Intensity could be affected by on-site factors, such as equipment and lighting, when the image was captured. The assays assigned to 282 FDA approved drugs were not conducted at the same time but in 21 batches. External factors could vary for different assay batches, but they were considered the same within each batch. Intensity is additive. Any effects caused by external factors, therefore, add to the existing intensity. As a result, the most intuitive solution to recalibration is to deduct the background intensity from each batch. In doing so, we bring all intensities from different assay batches together on an equal footing. The raw data are 16-bit images with intensities ranging from 0 to 65536. Table 3.1 includes the minimum intensity for each batch, which is assumed to be the background intensity, as well as the intended adjustments corresponding to the minimum. The recalibration imposes no effect on spatial structures. Once recalibrated, background intensities from different batches should all become zero.

Table 3.1: Minimum pixel intensity and intended adjustment by batch.

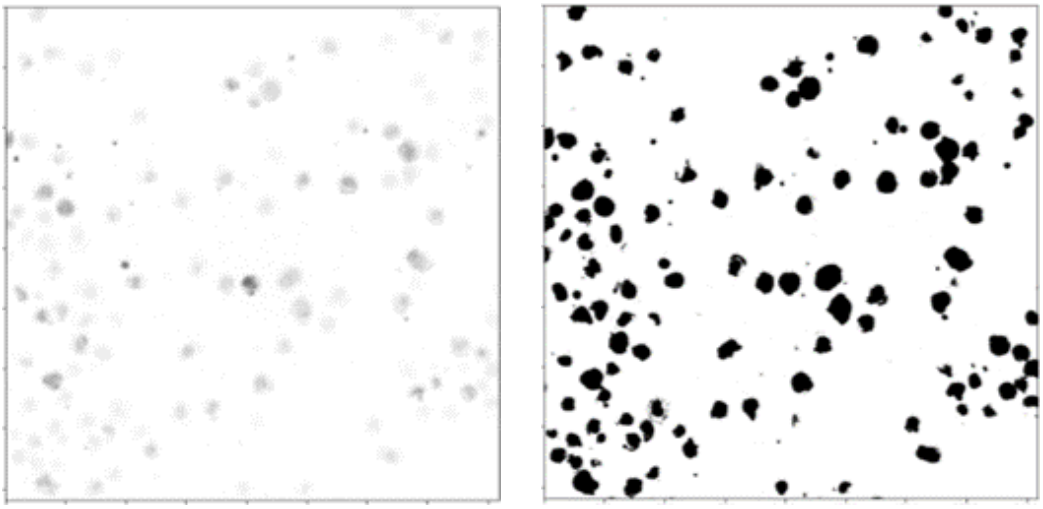
Assay batch	Minimum intensity	Intended adjustment
1	108	-108
2	108	-108
3	110	-110
4	110	-110
5	110	-110
6	110	-110
7	110	-110
8	110	-110
9	110	-110
10	110	-110
11	112	-112
12	106	-106
13	106	-106
14	108	-108
15	108	-108
16	106	-106
17	110	-110
18	108	-108
19	112	-112
20	110	-110
21	112	-112

3.2.2 Cell Detection and Cropping

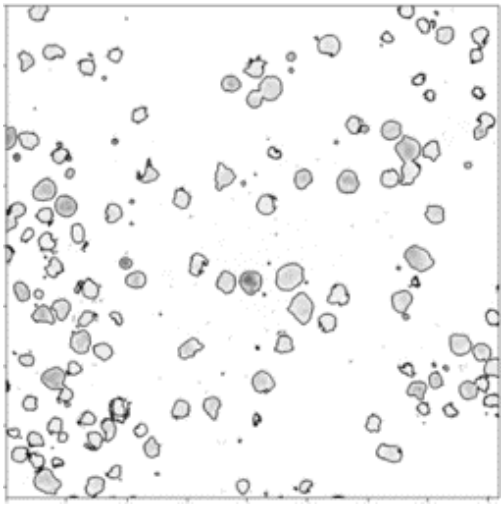
Next comes cell detection and cropping. A raw input image with dimensions of 2048*2048 is too large for any nontrivial algorithm to learn directly. To boost accuracy, we must segment the image into windows that contain only one cell. There are two free tools available in Python [90] for the task; both are functions of OpenCV [10]. These two tools are *findContours* and *watershed*.

findContours uses contour lines similar to the topographic maps for mountain climbing. Contour lines mark the regions in which intensities are higher than a predefined threshold. Figure 3.3 demonstrates the way *findContours* works. First, an input image is converted into a binary image with intensities of either 1 or 0. The business rule is that any intensity equal to or higher than a predefined threshold is converted into 1. Then for each object containing 1's, a curve is created joining all the points along the boundary. These curves are contour lines for object detection.

Compared with *findContours*, *watershed* is a more advanced tool. *watershed* works by first identifying regions of sure background and sure objects, then determining the boundary between them using an algorithm. Figure 3.4 demonstrates the way *findContours* works. Two thresholds are required for *watershed*, one for regions that are definitely background (white areas), the other for regions that are definitely objects (white areas surrounding by black bands). The black bands, therefore, are regions of uncertainty. A *watershed* algorithm refines the bands, thinning them in a controlled manner to mark the boundary.

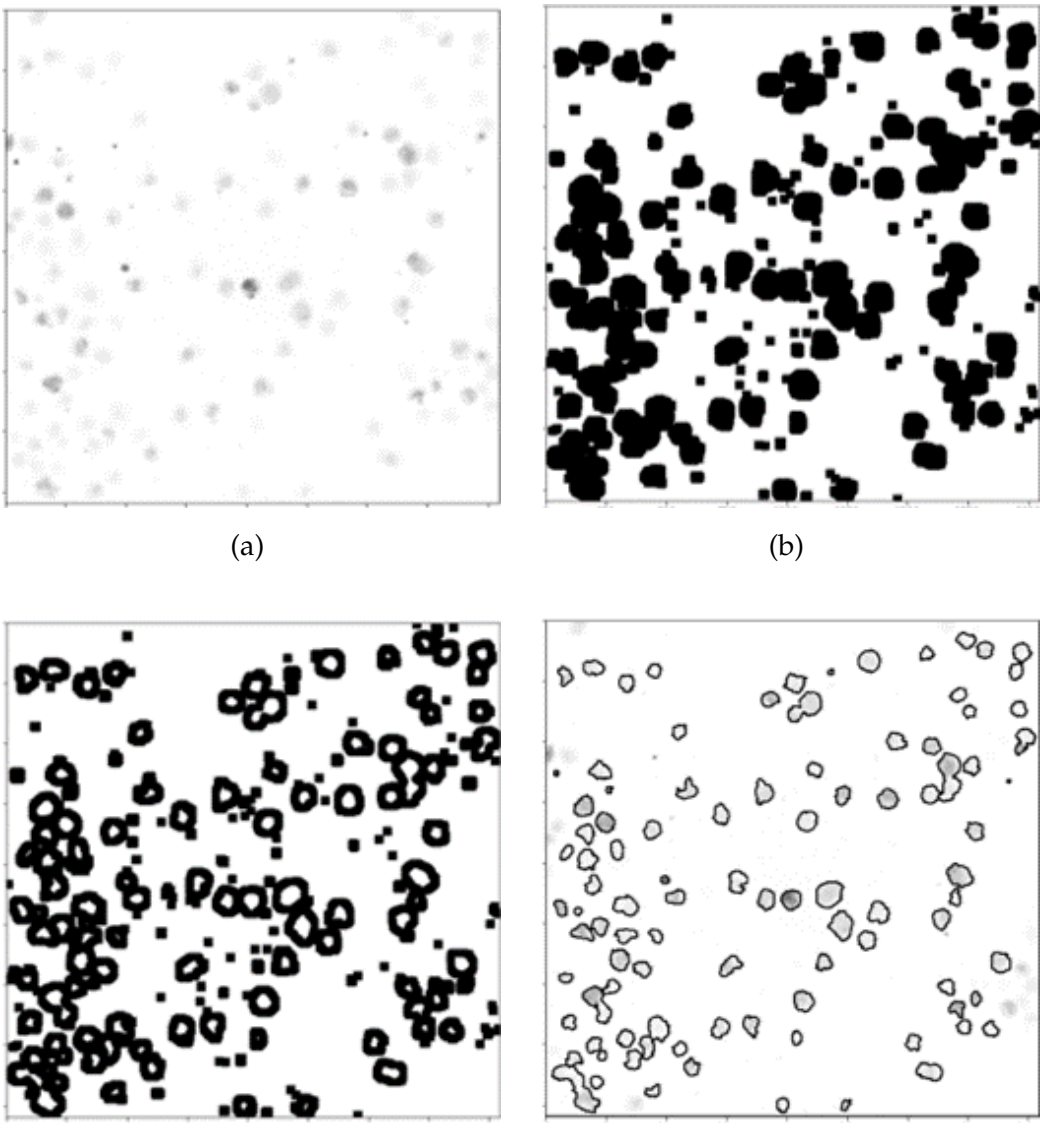


(a) (b)



(c)

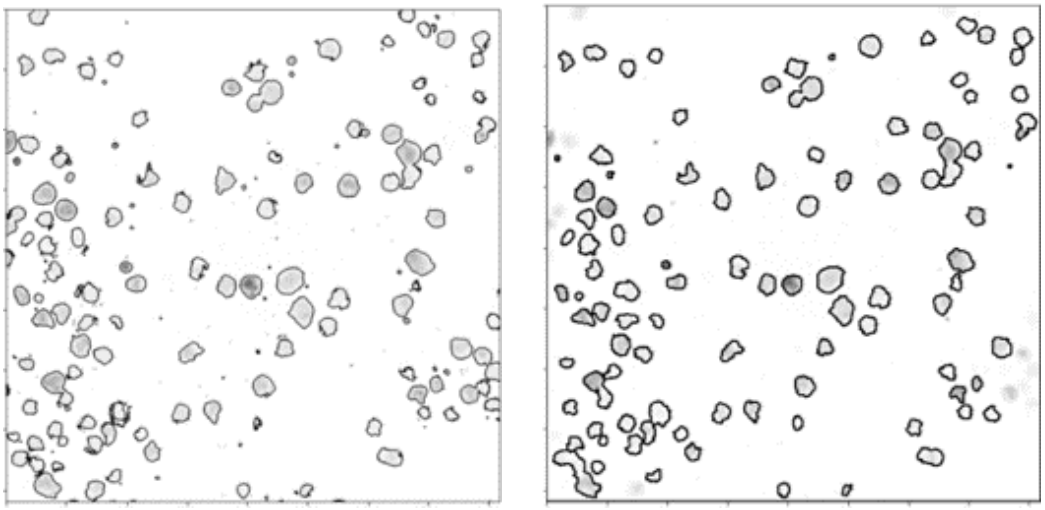
Figure 3.3: Demonstrations of *findContours*. (a) The original image. (b) The binary image based on a predefined threshold. (c) Cell detection using contour lines.



(a) (b)

(c) (d)

Figure 3.4: Demonstrations of *watershed*. (a) The original image. (b) White areas are sure background. (c) White areas surrounding by black bands are sure objects. (d) Cell detection using an algorithm.



(a) (b)

Figure 3.5: Comparison of *findContours* and *watershed*. (a) Cell detection using *findContours*. (d) Cell detection using *watershed*.

Figure 3.5 displays a comparison of *findContours* and *watershed* in cell detection, each with unique advantages. *findContours* is easier to employ, requiring only one threshold. Boundaries are well defined. However, it cannot detect objects that are touching the edge of the image, which is important in filtering out incomplete cells. *watershed* provides thicker, less refined boundaries. However, it automatically deselects objects that are too small (i.e. noise) as well as objects that are touching the edge. In this research, where well-presented inputs are desired, we go for the more thorough yet less automatic approach — *findContours*. Although a considerable amount of extra coding is required for *findContours* to work properly, it offers a significantly clearer boundary of cell. We set the threshold to a recolibrated intensity of 5. This threshold is somewhat arbitrary. It does not matter if we pick up small impurities during the process; they are removed in the next step.

Following cell detection, each cell is cropped out and placed in the middle of an empty window. The dimensions of these windows determine the

size of inputs to the next stage. If the windows are too large, extra computational resources are required to process them. By close examinations, we choose a 120*120 window that is adequate for most single-cell structures. Any object that does not fit into this window is considered to have at least two closely positioned cells and is deleted consequently.

3.2.3 Data Cleansing

Cleansing is the final step in the data preprocessing stage. The task for data cleansing is to delete the windows containing incomplete cells and the windows containing only impurities. This step is quite straightforward. We delete windows if the cell they contain is touching the edge in the original image. Moreover, we delete windows if the average intensity of the entire window does not reach a certain threshold. Failing to exceed the threshold indicates that the window's content is small impurities rather than a complete cell. The threshold is arbitrary to some extent because the gap in average intensity between impurities and a full cell is huge. We set this average intensity threshold to 5. Figure 3.6 displays some of the discarded windows. The first two windows from the left in Figure 3.6 contain only impurities; the next two windows contain an incomplete cell (cut by edge); the last window from the left contains a cell cluster that manages to squeeze inside a window because part of the cluster is an incomplete cell (cut by edge). This kind of clusters is discarded due to close proximity to the edge.

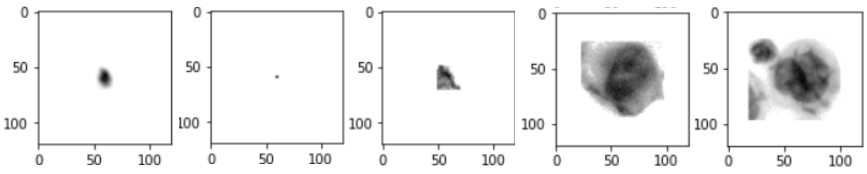


Figure 3.6: Some examples of the discarded windows.

3.3 Data Preprocessing Design

This section provides a concise and final design of the data preprocessing stage. Figure 3.7 outlines the steps taken in data preprocessing. Considerations behind the design are included in Section 3.2.

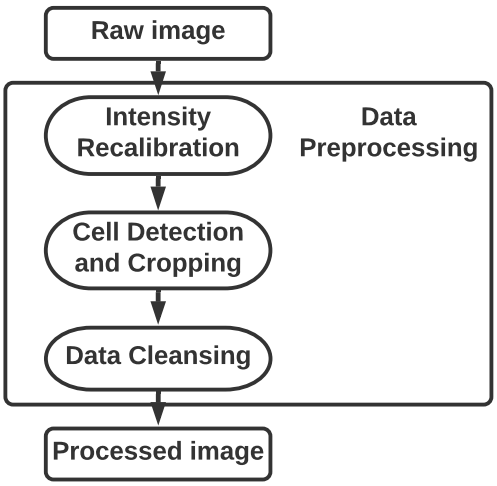


Figure 3.7: Detailed steps in the data preprocessing stage.

Intensity Recalibration

1. Adjust pixel intensities of raw images based on the background intensity of assay batch. Intended adjustments are shown in Table 3.1.

Cell Detection and Cropping

1. Apply the *findContours* function from the OpenCV package to the recalibrated images for cell detection. Binary threshold used in *findContours* is set to 5.

2. Crop out the detected cells and place them in the middle of an empty window of size 120*120. Objects larger than 120*120 are discarded.

Data Cleansing

1. Discard windows containing a cell that is touching the edge in the corresponding raw image.

2. Discard windows with an average intensity below 5.

3.4 Results and Discussion

Most windows discarded during cropping were cell clusters formed by multiple closely positioned cells. They were not single-cell structures. Meanwhile, most windows discarded during data cleansing were impurities as well as incomplete cells. A small proportion of these windows were cell clusters with an incomplete cell cut by an edge. Figure 3.8 presents some example windows produced after preprocessing. Each window contains exactly one complete cell. Furthermore, subcellular structures such as membrane and nucleus are clearly visible. From 61560 raw images, we obtained 1935920 such windows.

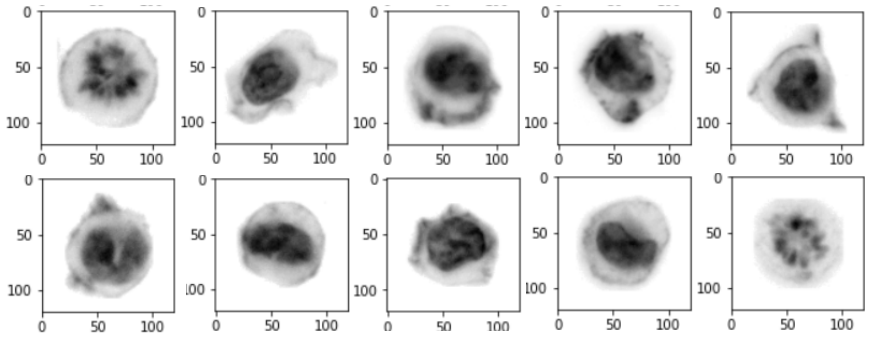


Figure 3.8: Some examples of the processed images, referred to as windows.

The data preprocessing stage is crucial to the overall success of this pipeline. Well presented inputs boost the precision and accuracy of a neu-

ral network significantly. In addition, the storage size of inputs was significantly reduced by cutting out the empty spaces between cells. As a result, while the original raw images had a total storage size of around 500 GB, the windows had a total storage size of just 111 GB in float32 data type. Float32 is the default format of input that goes into a neural network in Keras [14], the deep learning platform we used. Keras also allows float16 as input which takes 16 bits of memory instead. The float16 data type, however, comes with a cost of lower computation precision. Therefore, we chose float32 as the input data type because it provided a good balance between precision and storage space.

3.5 Chapter Summary

This chapter described the design and the implementation of data preprocessing steps. We managed to distil 61560 raw images into 1935920 smaller windows containing clear and complete single-cell structures using these steps. In doing so, we reduced the total storage size to 111 GB, around 22% of the original size. Moreover, cells were well positioned — right at the centre of a window. Such arrangement was beneficial to the overall precision and accuracy of the neural network. Next, the outputs from data preprocessing, in a compatible data type, proceed to the hit selection stage as inputs.

Chapter 4

Hit Selection Stage

This chapter discusses the second stage in the pipeline — hit selection. The content of this chapter includes an overview, preliminary explorations, the final design, results, and a brief summary.

4.1 Hit Selection Overview

Hit selection is the second stage in the pipeline, as shown in Figure 4.1. The outputs from the previous stage, 1935920 windows containing single-cell structures, are the inputs that feed into hit selection. These windows, organised under 282 drug codes, are 120*120 in size and have only one colour channel. We do not know the identities of drug codes until the very end of this research, except for drug 0 which is the negative control Dimethyl Sulfoxide (DMSO). The task is to select a shortlist of drugs out of the remaining 281 that induce significant phenotypic changes, i.e. the hits. In other words, we are tasked to select drugs whose associated windows have significantly different content from the DMSO's. To do that, we must assume most images associated with a certain drug are homogeneous. If one drug induces two distinctly different phenotypic changes, this pipeline might not work as planned. Since we do not intend to go the feature engineering route and do not have a true shortlist, this becomes an

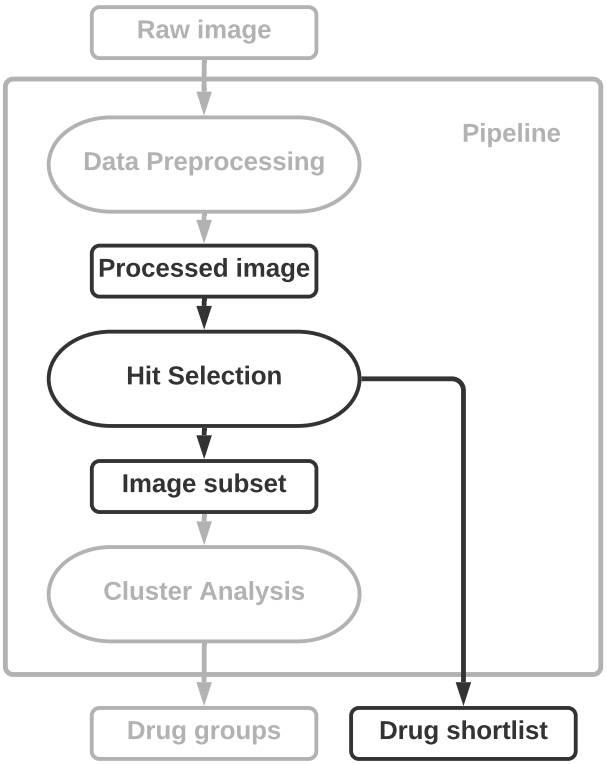


Figure 4.1: Hit selection is the second stage in the proposed pipeline.

unsupervised image classification problem in terms of both features and labels. At the end of this stage, there are two desired outputs. One is the shortlist of drug codes; the other is the windows associated with the shortlisted drugs.

To select hits, we must address the following five issues:

1. Develop a novel scheme for comparing representations;
2. Construct a generalised estimating equation model to predict optimal hyperparameters of convolutional autoencoder tailored for the input images;
3. Use the convolutional autoencoder to obtain condensed representations of input images;

4. Compare representations with the negative control to obtain similarity measures;

5. Determine an appropriate similarity threshold at which drugs are considered as hits.

Assumptions made in hit selection:

1. Each drug induces only one type of phenotypic changes.

4.2 Preliminary Explorations

This section describes some of the necessary explorations of data and methods. The explorations provide a better demonstration of the task as well as a collection of considerations behind the choices made in the final design.

4.2.1 Novel Comparison Scheme

This section describes a novel comparison scheme that can produce a similarity measure about two unlabelled image groups. There are some objectives we would like to achieve using the comparison scheme:

- Groups of different spatial structures are easily distinguishable.
- Groups of similar spatial structures are not distinguishable.
- Read-out is concise, preferably a single score describing the similarity between two groups of images.
- The comparison scheme is nonparametric and only requires image inputs.
- The comparison scheme is time-efficient.
- The comparison scheme is generalisable to other similar data sets.

According to the objectives above, the ideal comparison scheme would take two groups of images as inputs and nothing else, then produce a similarity measure based on the spatial structures in the images. One seemingly intelligent solution is a deep convolutional neural network. However, such a network is difficult to build and even more time-consuming and resource-demanding to train appropriately. Sometimes, working smart is more efficient than working hard. Instead, we can take advantage of the fact that only two groups of images are compared at one time. We present a novel comparison scheme that meets all the requirements. It may not seem intellectual at first. However, results suggest that the proposed scheme is reliable and effective.

Brief Outline

We are inspired by the idea of confusion in decision-making, where one is unsure of how to choose. This uncertainty can translate to a 50-50 prediction. In the proposed comparison scheme, two groups of input images (e.g. A and B) are first converted into multidimensional vectors. Then for each input image (i.e. the target), a certain number of its nearest neighbours are identified using a distance metric of choice. Next, a simple algorithm predicts an indicator for the target based on the distances and the nearest neighbours information. The indicator is an educated guess of where the target comes from, either group A or group B. Finally, we compare the predicted indicators with the true information and calculate a classification accuracy. If the two image groups contain the same spatial structures, the algorithm is confused. Consequently, the predicted indicators would be a random guess of either A or B. A classification accuracy close to 50% would suggest similar spatial structures in both groups. On the other hand, if the two image groups contain distinctively different structures, the classification accuracy would be close to 100%. We use this accuracy as a similarity measure.

Distance Metrics

There are many distance metrics available such as Euclidean, Manhattan, and Cosine [44]. When measuring the distance between point P and point Q, Euclidean distance $Dist_E$ is the length of the shortest line segment between them. $Dist_E$ can be written as in Eq (4.1).

$$Dist_E(P,Q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{4.1}$$

In Eq (4.1), p_i and q_i are the Euclidean vectors starting from origin; n is the dimension of space that P and Q occupy. Manhattan distance, also known as city block distance, is the sum of absolute differences between the elements of two vectors. With the same notations, Manhattan distance $Dist_M$ can be written as in Eq (4.2) below.

$$Dist_M(P,Q) = \sum_{i=1}^{n}|p_i - q_i| \tag{4.2}$$

Finally, cosine distance is cosine of the angle between two vectors. Cosine distance $Dist_C$ is defined in Eq (4.3).

$$Dist_C(P,Q) = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}} \tag{4.3}$$

Figure 4.2 below illustrates an example of the distance metrics in 2D. Euclidean distance, equal to $\sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$ in this case, is the most intuitive one among the three as it represents the shortest distance from P to Q in a straight line. Manhattan distance is the total distance from P to Q along axes, or $|x_p - x_q| + |y_p - y_q|$. Finally, cosine distance is simply the cosine of the angle between P and Q.

In high-dimensional spaces, data points become increasingly sparse that common distance metrics, e.g. Euclidean, are less meaningful. The sparsity causes data points to further differ from their neighbours. As a result, data points become increasingly equidistant to each other, making

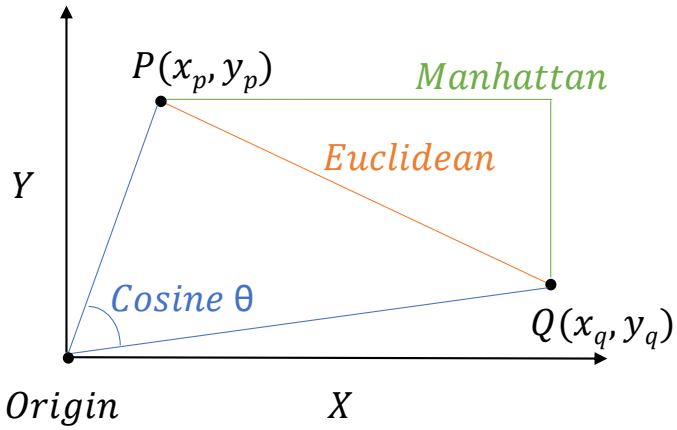


Figure 4.2: Illustrations of three popular distance metrics.

the distance information insignificant. This phenomenon is referred to as the curse of dimensionality [44]. A logical workaround is to decrease dimensionality, which we employ. In addition, the lower exponent a metric has, the better it adapts to high-dimensional spaces [1]. For example, Manhattan distance, being a sum of differences, has an exponent of 1, whereas Euclidean and cosine distance have an exponent of 2. However, it should still be a case by case decision. Thus, we leave this choice open for now as a hyperparameter.

Indicator Prediction

The proposed comparison scheme uses an algorithm to predict an indicator for each input image, referred to as a target in this section. This indicator is an educated guess of the image group the target comes from. To make such a guess, we select a fixed number of neighbour vectors closest to the target in terms of a distance metric. The nearest neighbours, and their distances to the target, are inputs of the algorithm. This algorithm can be described as follows.

1. **Input**: any two groups of vectors.

2. Mark the two input groups as group A and group B, and record this true indicator for each input vector.

3. For each vector from either group, i.e. the target, find its K nearest neighbours among the two groups with regard to a distance metric of choice.

4. Record the true indicators and the distances from the target for the nearest neighbours.

5. Calculate the sum of squared distances S_A for the nearest neighbours from group A.

6. Calculate the sum of squared distances S_B for the nearest neighbours from group B.

7. Compare S_A and S_B. If S_A is smaller, the predicted indicator for this target is A; otherwise, the predicted indicator is B.

8. **Output**: a predicted group indicator for each input vector.

Figure 4.3 shows a simple 2D example in which there are four vectors from group A (green) and four vectors from group B (red). Suppose $K = 3$ and the distance metric of choice is Euclidean. For a specific target vector from group B, as shown in Figure 4.3, we select three neighbours closest to the target in terms of Euclidean distance. In summary, one group B neighbour with distance 2, and two group A neighbours with distances 4 and 5 respectively. $S_A = 4^2 + 5^2 = 41$. $S_B = 2^2 = 4$. S_B is smaller; thus, the target is given a predicted indicator B.

The number of nearest neighbours directly affects prediction accuracy. Therefore it is treated as a hyperparameter. Moreover, fast queries of nearest neighbours has been enabled by powerful similarity search tools such as Faiss [46]. It only takes seconds to run for an entire data set. The predicted indicators and the true indicators are collected to produce a similarity measure. This process is detailed below.

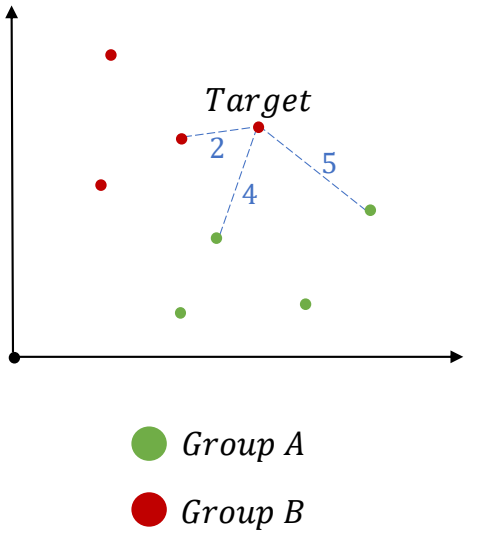


Figure 4.3: An example demonstrating indicator selection.

Similarity Measure

Following indicator prediction, we calculate a classification accuracy based on the predicted indicators and the real indicators. Figure 4.4 shows a confusion matrix used in calculating a classification accuracy. When a predicted indicator matches the actual one, it counts as a true A or a true B. Otherwise, it counts as a false A or a false B. The balanced accuracy is given by Eq. (4.4).

$$Balanced\ \ accuracy = (\frac{TA}{TA + FA} + \frac{TB}{TB + FB}) \div 2 \times 100\% \tag{4.4}$$

An accuracy close to 50% suggests a close resemblance between the vectors in group A and those in group B. If the vectors are different, the algorithm would be able to distinguish where a vector comes from. Consequently, the majority of predictions would be TA's or TB's and the ac-

curacy would be close to 100%. This accuracy serves as an effective and intuitive similarity measure.

The comparison scheme is tested later in this chapter.

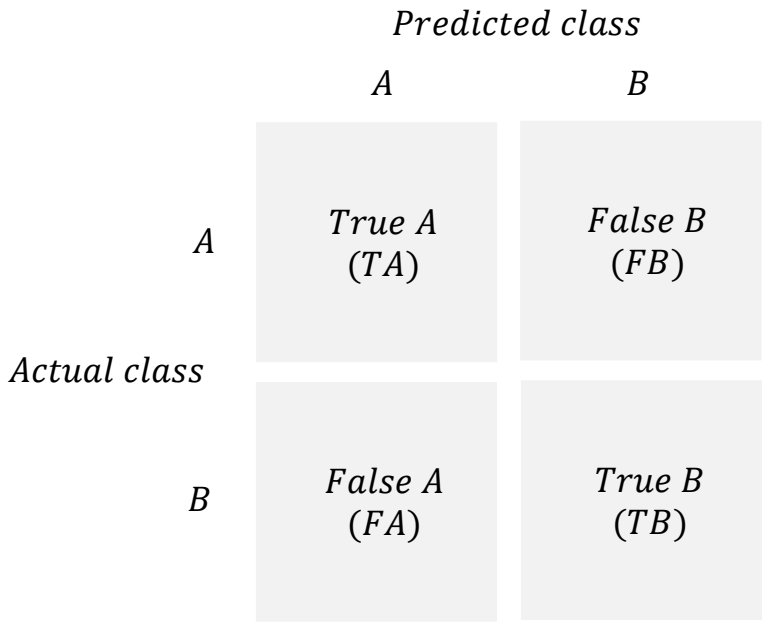


Figure 4.4: A confusion matrix for calculating accuracy.

4.2.2 Convolutional Autoencoder

In this section, we discuss some explorations about convolutional autoencoder. Background of convolutional autoencoders is discussed in Section 2.6. Considering that the windows of cells are not labelled, the explorations conducted in this section rely on a well presented labelled data set of handwritten digits — MNIST [57]. MNIST contains around 70000 handwritten samples of numbers from 0 to 9. The images have dimensions 28*28*1. The two data sets, windows of cells and MNIST, are strikingly similar. Both of them are organised in groups and in similar formats — they are all squared images with darker content at the centre surrounded

by a white background. The groups of digits can well imitate the groups of drugs, where some are more similar than others.

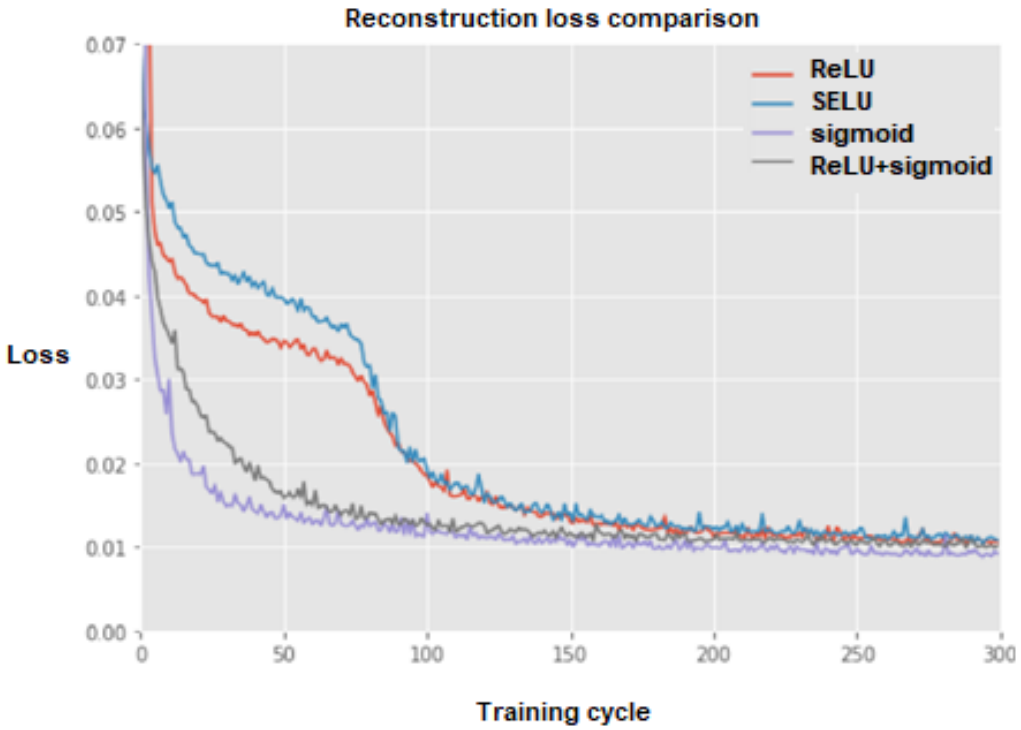


Figure 4.5: Evolution of reconstruction losses with different activation functions. While all activation functions converge eventually, fast convergence saves training time and is preferred. Hyperparameters: 2 convolutional layers in encoder, 16 filters in the first convolutional layer

Figure 4.5 shows how activation functions affect training time. We trained four convolutional autoencoders with the same 5000 samples of handwritten '5' in a preliminary test run. The hyperparameters, as discussed in Section 2.6.3, were 2 layers and 16 filters. In addition, training samples were augmented slightly to include randomness in the training set, a technique called online augmentation [86]. The augmentation randomly selected samples from the training set. As a result, each training

cycle contained 128 batches of 32 samples instead of an entire epoch. As discussed in Section 2.5.2, ReLU helps tackle the vanishing gradient problem. However, ReLU being linear reduced the network's ability to learn patterns fast. Sigmoid, on the other hand, was the first to converge. To achieve a balance, we chose ReLU as the activation function except for the last layer, where we used sigmoid. The combination of ReLU and sigmoid had a similar short convergence time as sigmoid while resisting vanishing gradient. Next we look at how other hyperparameters that directly affect the reconstruction, keeping activation functions fixed.

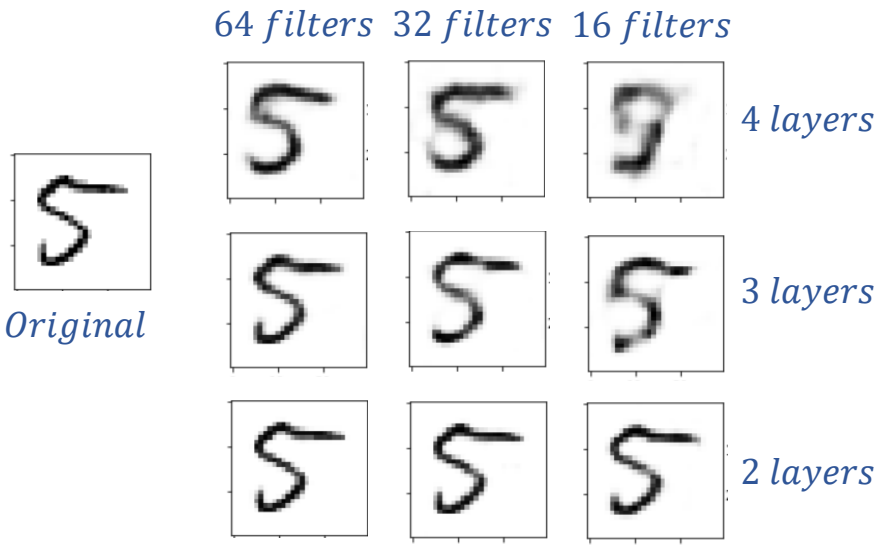


Figure 4.6: Reconstruction examples using different convolutional autoencoder architectures. The varying hyperparameters are the number of convolutional layers in the encoder, and the number of filters in the first convolutional layer. Activation function of choice is ReLU+sigmoid.

Figure 4.6 shows some reconstructions of the same original image of '5' using nine different convolutional autoencoder architectures. It is clear that filter number and layer number affect the reconstruction significantly. Intuitively, as the number of filters increases, a convolutional neural net-

work can learn more features. In the meantime, as the number of layers increases, the features learned can be more complex. Nevertheless, more layers and filters bring in more trainable parameters and therefore require more training samples. No enough trainable parameters, on the other hand, lead to poor performance. Looking across the columns in Figure 4.6, 16 filters are not enough for a 4-layer encoder. 32 filters and 64 filters provide better results. Looking vertically, 2-layer architectures seem to perform well. However, there is a pitfall in selecting an autoencoder architecture where the bottleneck carries equal or more information than the original image. As discussed in Section 2.5.3, we need an information bottleneck for dimensionality reduction. The bottleneck is related to pooling layers. In this exploration, 2-32 and 2-64 did not force a bottleneck. Consequently, these two architectures are not suitable. 4-16 and 4-32 had a bottleneck that was too small in terms of information volume, resulting in poor reconstructions. For another data set, however, this bottleneck may be just enough.

This exploration shows that layer number and filter number should be situational choices based on the complexity, size, and available data of input. Next, we explore the functionality of convolutional autoencoders. The architecture of autoencoders, as discussed in Section 2.5, is in an hourglass shape. We gradually decrease the number of filters in the encoder (halve at each consecutive layer), and mirror the encoder to construct a decoder. As a result, the filter problem is simplified. We only need to determine the filter number in the first convolutional layer.

Figure 4.7 displays some reconstruction examples from the test data set. The convolutional autoencoder, trained by digit '5' only, tried to morph other digits into a hybrid form similar to the training data. The reconstruction loss for '5' is logically the smallest. If a digit is not '5', e.g. '3', the morphing creates a difference between the original and the reconstructed image, increasing the reconstruction loss. We can see the reconstructed '3' becomes a '5' in Figure 4.7. One would presume that reconstruction loss

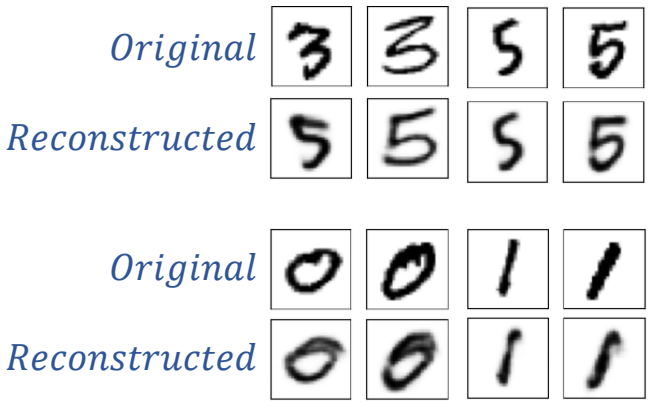


Figure 4.7: Reconstruction examples of MNIST digit '3', '5', '0', '1'. The convolutional autoencoder was trained solely on the digit '5'. Hyperparameters: 2 convolutional layers in encoder, 16 filters in the first convolutional layer

increases as a digit bears less resemblance to '5'. This presumption, however, is not true. The original and reconstructed images of '1', for example, are very similar, resulting in a small reconstruction loss.

Convolutional autoencoders on their own, or with additional fully connected layers at the end, can be used for classification purposes utilising reconstruction loss. Nevertheless, the results are shown to be poor. The fact that the basic shapes and orientation are retained in Figure 4.7 suggests that the encoder did an excellent job condensing an original image into distilled representations. The decoder, however, tended to overfit training data. This theory is later confirmed in Section 4.2.4.

In summary, the convolutional autoencoder on its own is a potent dimensionality reduction tool but not much else. As a classifier, its accuracy would not be good enough.

4.2.3 Hyperparameters Selection with GEE

Hyperparameters selection is a multi-step process. The proposed pipeline has several hyperparameters to be defined. For example, the hyperparameters in the convolutional autoencoder are: convolutional layer number in the encoder, filter number in the first convolutional layer, activation function, and batch size. In addition, there are two more hyperparameters in the comparison scheme: distance metric and number of nearest neighbours. Some of these are already selected based on literature reviews and explorations described in Section 2.5.4 and Section 4.2.2. Table 4.1 summarises the hyperparameters and the intended choices in this pipeline.

Table 4.1: The choices of hyperparameters for the MNIST data set.

Component	Hyperparameter	Choice
Conv. autoencoder	Layer number	To be defined
Conv. autoencoder	Filter number	To be defined
Conv. autoencoder	Activation function	ReLU+sigmoid
Conv. autoencoder	Batch size	32
Comparison scheme	Distance metric	To be defined
Comparison scheme	Nearest neighbours number	To be defined

We use generalised estimating equations (GEE), discussed in Section 2.3, to select four hyperparameters: layer number, filter number, distance metric, and nearest neighbour number. These hyperparameters are categorical. For example, 4 layers are not exactly twice as good as 2 layers. The hyperparameter selection is a composite process. We aim to select a set of hyperparameters that helps produce higher classification accuracies.

To simulate the hit selection problem, we divide the MNIST data set into 20 equal-size groups — two for each digit. Images within the same group are associated with the same digit, the identity of which is not disclosed. The idea is to select 1 of these 20 groups as a negative control, then select the groups that are significantly different from negative control us-

ing a comparison scheme. At the end of this exploration, we can validate the results with their true identity. Table 4.2 summarises these 20 groups. Group A is the negative control.

Table 4.2: Summary of the 20 image groups from the MNIST data set.

Group	Samples	Label	Group	Samples	Label
A	3000	'5'	K	3000	Unknown
B	3000	Unknown	L	3000	Unknown
C	3000	Unknown	M	3000	Unknown
D	3000	Unknown	N	3000	Unknown
E	3000	Unknown	O	3000	Unknown
F	3000	Unknown	P	3000	Unknown
G	3000	Unknown	Q	3000	Unknown
H	3000	Unknown	R	3000	Unknown
I	3000	Unknown	S	3000	Unknown
J	3000	Unknown	T	3000	Unknown

We need a negative control and a mild positive control to select the best set of hyperparameters for the input data. The positive control can be selected using the comparison scheme introduced in Section 4.2.1. It does not need to be a strong negative control. Anything mildly dissimilar will suffice, no matter what distance metric or number of nearest neighbours we use. In this case, our first try — group B — has a classification accuracy of 95.3% (Manhattan, $K = 3$). Thus, group B is good enough to be a mild positive control. (In hindsight, 95.3% was one of the lowest out of 19 groups, and that was before feature extraction. A solid piece of evidence showing the robustness of the comparison scheme.) 300 samples from the negative control, i.e. group A, are set aside for testing. This 300 samples serve as testing samples.

The next step is to train several convolutional autoencoders with different sets of hyperparameters. The training data set can be a random

selection of the image groups. For example, we randomly select K, M, T, plus the negative control A and mild positive control B. The neural network does not have any classification function at all, only dimensionality reduction. Nonetheless, it is essential to diversify the training data set to increase generalisability. Limited by length, we do not show the training details here.

Next, we select 300 independent sample groups from group A and B. Each sample group is randomly selected without replacement and contains 300 unique samples from group A and 300 unique samples from group B. Two sample groups are considered independent if the selection method is independent [76]. Therefore, while we only have 5700 unique samples total in group A and B (300 from group A are set aside), we make up 300 independent and unique combinations of 600 handwritten samples.

In the final step, we select the optimal hyperparameters. One of the trained convolutional autoencoders is assigned to each sample group and perform feature extraction, i.e. extract a condensed representation for each sample using a convolutional autoencoder. Then, we compare the condensed representations in each sample group with the testing samples using a randomly selected distance metric. 10 comparisons are performed for each sample group — five nearest neighbour choices (1, 3, 5, 7, 9) and two group choices (A and B) — resulting in 3000 classification accuracies. This accuracy is discussed in Section 4.2.1. Each pair of accuracies (from group A and group B) are then translated to a decision score using Eq. (4.5). The decision scores are the correlated outcomes for GEE models.

$$S = |50\% - M_A| + |100\% - M_B| \tag{4.5}$$

In Eq. (4.5), S is the decision score, M_A is the classification accuracy obtained by comparing group A samples with the testing samples, M_B is the classification accuracy obtained by comparing group B samples with the testing samples. As discussed in Section 4.2.1, 50% and 100% are the ideal

values for M_A and M_B respectively. Low decision scores, therefore, are preferable.

By backward elimination, described in Section 2.3.4, we arrive at an optimal GEE model for the decision scores. Eq. (4.6) defines this model.

$$S_{i,K} = \beta_0 + \beta^F_{f_i} + \beta^L_{l_i} + \beta^M_{m_i} + \beta^N_K + \beta^{FL}_{f_i,l_i} + \beta^{FM}_{f_i,m_i} + \beta^{FN}_{f_i,K} + \beta^{LM}_{l_i,m_i} + \beta^{LN}_{l_i,K} + \beta^{MK}_{m_i,K} + \beta^{FLM}_{f_i,l_i,m_i} + \beta^{NLM}_{K,l_i,m_i} + \beta^{FNM}_{f_i,K,m_i} + \beta^{FLN}_{f_i,l_i,K} + \beta^{FLMN}_{f_i,l_i,m_i,K} \quad (4.6)$$

In Eq. (4.6), all predictors are categorical. For each sample group i, $i = 1, 2, \ldots, 300$, $S_{i,K}$ is the decision score for K nearest neighbours, $K \in \{1, 3, 5, 7, 9\}$; f_i represents the filter number choice for sample group i, $f_i \in \{16, 32, 64\}$; l_i represents the layer number choice, $l_i \in \{2, 3, 4\}$; m_i represents the distance metric choice, $m_i \in \{Euclidean,\ Manhattan,\ Cosine\}$. The constraints are: any β that includes at least one of the subscripts $f_i = 16$, $l_i = 2$, $m_i = Euclidean$, $K = 1$, is equal to zero.

This GEE model has an identity link function. Therefore, normality in residuals would suggest that GEE assumptions are all met. Using Figure 4.8, we conclude that normality is met moderately. The mean of residuals (-0.000386) is very close to zero and the standard deviation (0.0121) is relatively small, indicating a good fit.

Figure 4.9 displays the model predictions, Decision scores close to zero indicates the set of hyperparameters is more suitable for this particular data set. The lowest decision score 0.0742 belongs to a combination of 4 layers, 64 filters, Manhattan distance, and 5 nearest neighbours. The five best sets of hyperparameters are summarised in Table 4.3.

Although the five best sets of hyperparameters are not statistically different from each other at a significance level of 5% (method discussed in Section 2.3.5), the 2-layer 64-filter architectures do not force an information bottleneck for the MNIST data set. This is because the MNIST data set only has one colour channel. Therefore, the second best combination is 4 layers, 64 filters, cosine distance, and 9 nearest neighbours.

Table 4.3: Five best sets of hyperparameters for the MNIST data set.

Layer	Filter	Distance metric	Nearest neighbours	Decision score
4	64	Manhattan	5	0.0742
2	64	Euclidean	3	0.0760
2	64	Euclidean	7	0.0761
2	64	Euclidean	9	0.0786
4	64	Cosine	9	0.0788

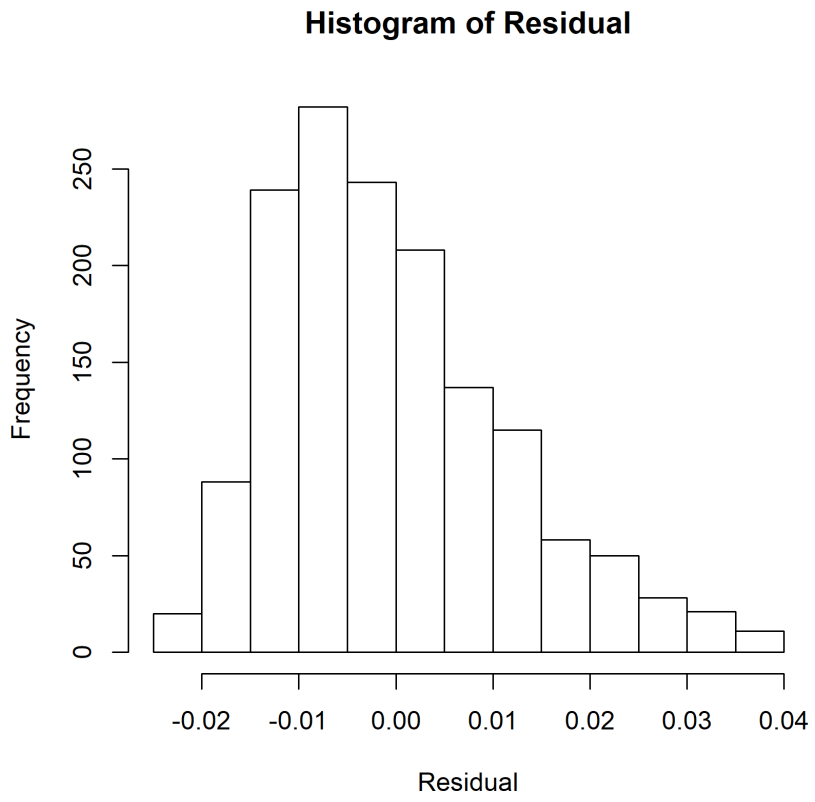


Figure 4.8: Visual inspection for normality in residuals for a GEE model with identity link function.

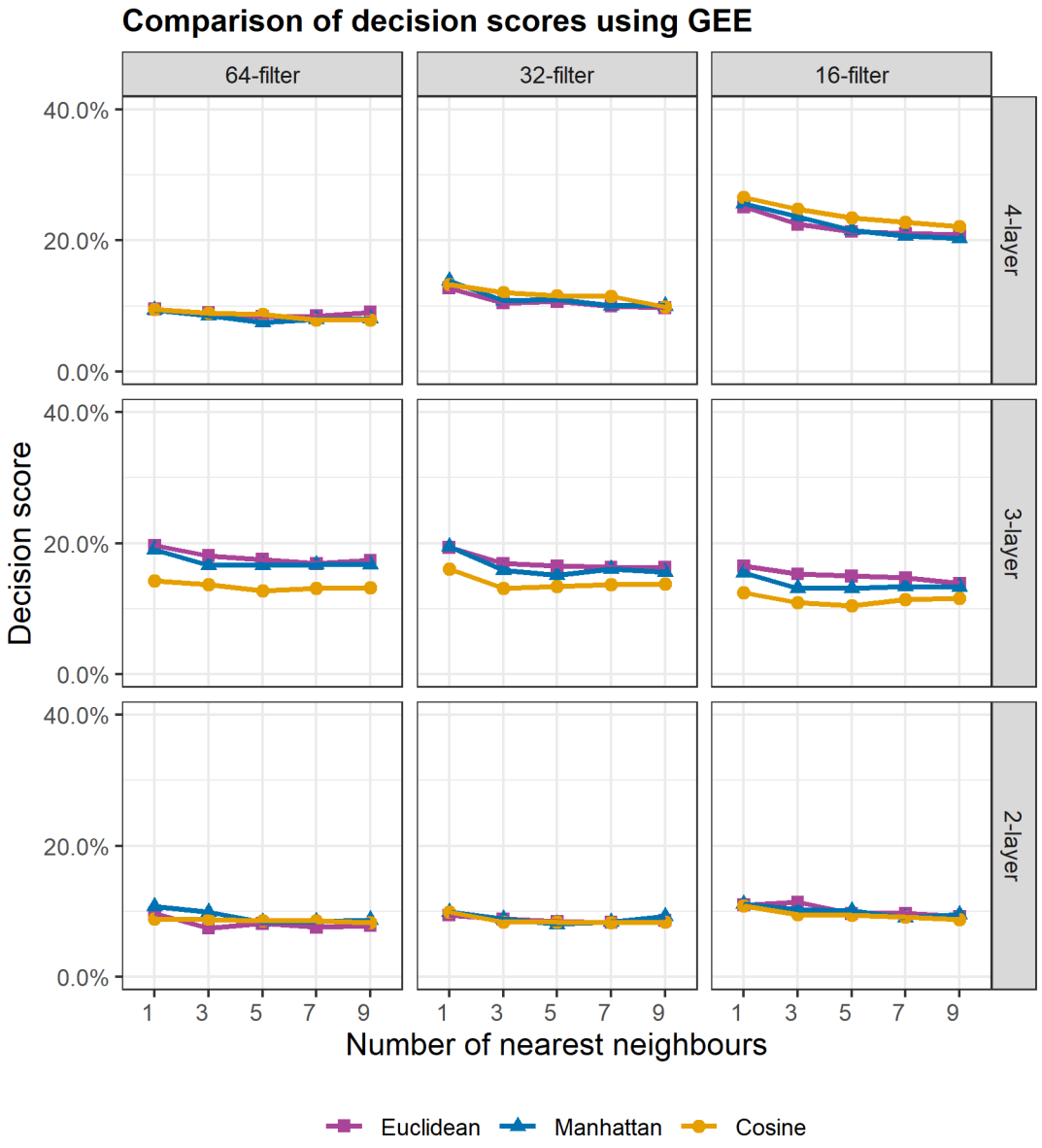


Figure 4.9: Visual comparison of decision scores using GEE.

4.2.4 Hit Selection

Finally, with the help from a convolutional autoencoder, we can select hits using the comparison scheme. The choices of hyperparameters are summarised in Table 4.4.

Table 4.4: The choices of hyperparameters for the MNIST data set (complete).

Component	Hyperparameter	Choice
Conv. autoencoder	Layer number	4
Conv. autoencoder	Filter number	64
Conv. autoencoder	Activation function	ReLU+sigmoid
Conv. autoencoder	Batch size	32
Comparison scheme	Distance metric	Manhattan
Comparison scheme	Nearest neighbours number	5

As discussed in Section 4,2,3, image groups B-T are compressed using the convolutional autoencoder (CAE) described. We then compare the condensed representations of each image group with group A, the negative control. Results are shown in Table 4.5. Classification accuracy is all over 99% for non-'5'. For another group of digit '5', the accuracy is around 50%. This accuracy is an excellent similarity measure for obvious reasons. Furthermore, we now know that the convolutional autoencoder was trained on digits '3', '2', '5', and '8' only. It had never seen the other six digits, yet it was able to distinguish non-'5' with exceptional levels of accuracy. This exploration shows that convolutional autoencoders as dimensionality reduction tools are highly generalisable to unseen data, given that the unseen data are in a similar format or style. This also suggests that the encoder component is far less susceptible to overfitting than the decoder component (issue discussed in Section 4.2.2). Moreover, the classification accuracy of group A and B is 95.3% without CAE, and 99.3% aided by CAE. This suggests feature extraction boosts accuracy signifi-

cantly.

Table 4.5: Similarities between image groups and the negative control.

Group	Accuracy	Label revealed	Group	Accuracy	Label revealed
A	N/A	'5'	K	0.990	'3'
B	0.993	'8'	L	0.992	'9'
C	0.994	'6'	M	0.998	'2'
D	0.995	'1'	N	0.501	'5'
E	0.992	'8'	O	0.995	'1'
F	0.996	'0'	P	0.995	'7'
G	0.995	'4'	Q	0.995	'7'
H	0.990	'3'	R	0.994	'4'
I	0.993	'6'	S	0.995	'0'
J	0.993	'9'	T	0.998	'2'

In a broader sense, this novel CAE-aided comparison scheme can serve as an effective unsupervised clustering tool for new samples. The proposed comparison scheme can work on its own. It is not a neural network. However, when feature extraction is performed by a convolutional autoencoder (CAE) before classification, accuracy goes up. Considering that the CAE only serves as a dimensionality reduction tool, it requires significantly less training samples than other neural networks with additional functionality. We conduct an experiment using the MNIST data set. Three clustering tools are used to classify new handwritten samples into unknown clusters. Table 4.6 shows that the proposed comparison scheme outperforms other common clustering tools such as support vector machine (SVM) [92] and deep adaptive clustering (DAC) [12] by a large margin in terms of overall performance. Due to limited length, the other tools are not discussed in this thesis. The training information displayed for the proposed scheme is for CAE only.

We used the cross-validation method to calculate a balanced accuracy

Table 4.6: The CAE-aided comparison scheme outperforms common unsupervised clustering tools for the MNIST data set.

	SVM	DAC	DAC	Proposed scheme
Training set	30000	30000	21000 (choose 7)	21000 (choose 7)
Test set	30000	30000	30000	30000
Training time	4min	42min	30min	4min
Evaluation time	30s	5s	5s	50s
Binary accuracy	83.2%	98.6%	78.1%	99.6%
Multi-class accuracy	N/A	97.5%	62.3%	97.1%

for unsupervised clustering. Seven out of ten digits were randomly chosen to simulate a scenario where some of the test data were new to the model. Compared with SVM and DAC, the proposed scheme required far less training time yet produced a much higher balanced accuracy. This is because the proposed scheme only used training for dimensionality reduction. More sophisticated functionality would have required more training time. Furthermore, both SVM and DAC required careful tuning of their hyperparameters. In comparison, the proposed scheme tuned hyperparameters automatically using statistical modelling. DAC achieved excellent results when test data were known clusters. However, its accuracy dropped drastically with unseen test data. The proposed scheme, on the other hand, was excellent in both scenarios. It is safe to say that the proposed scheme outperformed the other neural networks by a wide margin.

4.3 Hit Selection Design

This section provides a concise and final design of the hit selection stage. Figure 4.10 outlines the steps taken in hit selection. Considerations behind the design are included in Section 4.2.

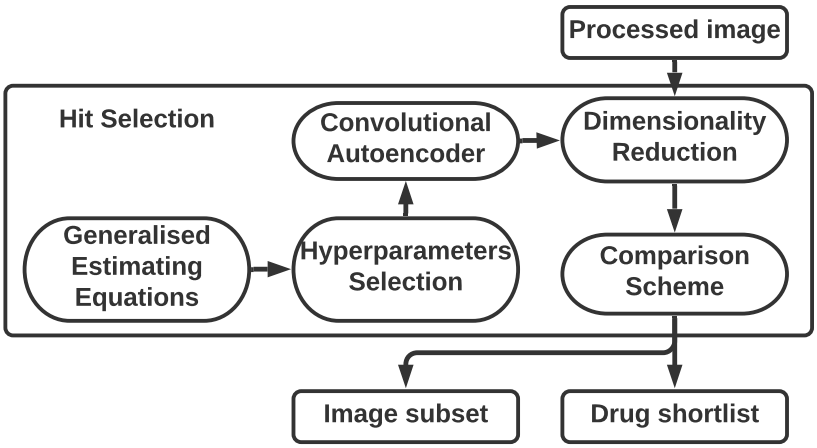


Figure 4.10: Detailed steps in the hit selection stage.

Control Selection

1. Select a mild positive control using a novel comparison scheme discussed in Section 4.2.1. Anything moderately different from negative control will suffice.
2. Set aside a small testing group of samples from negative control (DMSO) for hyperparameters selection.

Hyperparameters Selection

1. Train several convolutional autoencoders with various hyperparameters on a small subset of input data.
2. Select 300 independent sample groups from the negative control and the mild positive control.
3. Use different sets of hyperparameters as treatments, then compare the sample groups with the testing group in the comparison scheme.
4. Collect classification accuracies and use them as similarity measures to calculate decision scores.

5. Construct a GEE model to predict the most suitable set of hyperparameters.

Hit Selection

1. Apply dimensionality reduction to the processed images using a convolutional autoencoder with the selected hyperparameters.

2. Compare condensed representations of drug-treated cells and DMSO-treated cells in the comparison scheme.

3. Select a similarity threshold for the hits.

4. Select hits based on the similarity measure.

5. Collect a subset of images associated with hit compounds for the next stage.

4.4 Results and Discussion

According to design (Section 4.3), we first selected a mild positive control using the proposed comparison scheme without dimensionality reduction. This process took 39 seconds for each trial. In the twelfth trial, drug 13 with a classification accuracy of 73% was selected as the mild positive control.

In hyperparameters selection, the optimal GEE model (discussed in Section 2.3) selected by backward elimination (discussed in Section 2.3.4) can be described as Eq (4.7). This model uses an identity link function and contains all main effects and interactions. We were not interested in deciphering the relationships between variables but only selecting the best set of hyperparameters. Therefore, it was acceptable to have a relatively complicated model.

$$S_{i,K} = \beta_0 + \beta^F_{f_i} + \beta^L_{l_i} + \beta^M_{m_i} + \beta^N_K + \beta^{FL}_{f_i,l_i} + \beta^{FM}_{f_i,m_i} + \beta^{FN}_{f_i,K} + \beta^{LM}_{l_i,m_i} + \beta^{LN}_{l_i,K} + \beta^{MK}_{m_i,K} +$$

$$\beta^{FLM}_{f_i,l_i,m_i} + \beta^{NLM}_{K,l_i,m_i} + \beta^{FNM}_{f_i,K,m_i} + \beta^{FLN}_{f_i,l_i,K} + \beta^{FLMN}_{f_i,l_i,m_i,K} \tag{4.7}$$

In Eq. (4.6), all predictors are categorical. For each sample group i, $i = 1, 2, \ldots, 300$, $S_{i,K}$ is the decision score for K nearest neighbours, $K \in \{1, 3, 5, 7, 9\}$; f_i represents the filter number choice for sample group i, $f_i \in \{16, 32, 64\}$; l_i represents the layer number choice, $l_i \in \{2, 3, 4\}$; m_i represents the distance metric choice, $m_i \in \{Euclidean,\ Manhattan,\ Cosine\}$. The constraints are: any β that includes at least one of the subscripts $f_i = 16$, $l_i = 2$, $m_i = Euclidean$, $K = 1$, is equal to zero. Normality of residuals can be examine using Figure 4.11.

As observed, the normality assumption is moderately met. The residuals have a mean of -0.000612 and a standard deviation of 0.0115. A mean close to zero and a small standard deviation combined indicate a good fit. Therefore, all GEE assumptions are met.

Table 4.7 summarises the best set of hyperparameters selected by the GEE model. This set had a decision score of 0.109, the best of combinations. Interestingly, the most suitable number of layers was 3 for the cell data set and was 4 for the MNIST data set. This choice can be partly explained by the fact that a 3-layer 64-filter architecture compresses input to 25% of the original size, while a 4-layer 64-filter architecture compresses input to 4% of the original size. The compression rate for a 4-layer architecture is acceptable for a simpler data set such as MNIST, but too high for a complex data set such as the cell images. Nonetheless, 25% of the original size in terms of storage space is impressive enough. This process further reduced the total storage size to 28 GB, much smaller than the unprocessed raw image data set (over 500 GB).

The convolutional autoencoder was trained on windows of cells associated with 15 different drugs, including negative control and mild positive control, to ensure diversity in training samples. The training set included

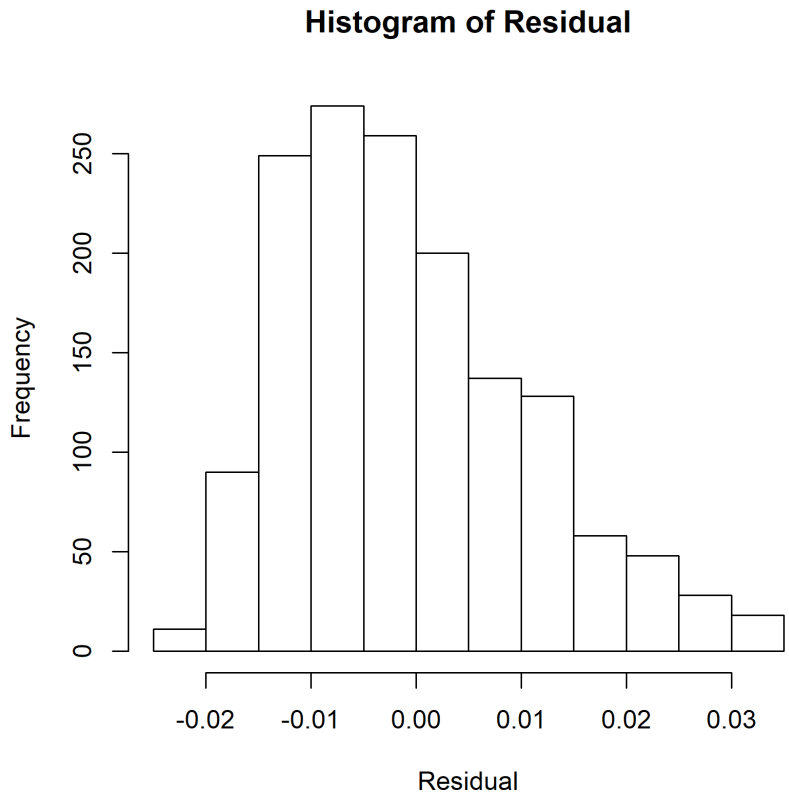


Figure 4.11: Visual inspection for normality in residuals for a GEE model with identity link function.

101763 windows, whereas the validation set included 15420 windows — a ratio of 6.6 to 1. Figure 4.12 displays the evolution of losses. The training process was smooth in general, and it took 26 minutes to converge. Windows were augmented online before training. The augmentation algorithm randomly selected samples from the training set and randomly rotated them at an angle between -20 to 20 degrees. This was to increase generalisability. As a result, each training cycle contained 128 batches of 32 samples instead of an entire epoch. Early stopping was in place to halt the training when convergence was reached.

Table 4.7: The choices of hyperparameters for the cell image data set (complete).

Component	Hyperparameter	Choice
Conv. autoencoder	Layer number	3
Conv. autoencoder	Filter number	64
Conv. autoencoder	Activation function	ReLU+sigmoid
Conv. autoencoder	Batch size	32
Comparison scheme	Distance metric	Manhattan
Comparison scheme	Nearest neighbours number	5

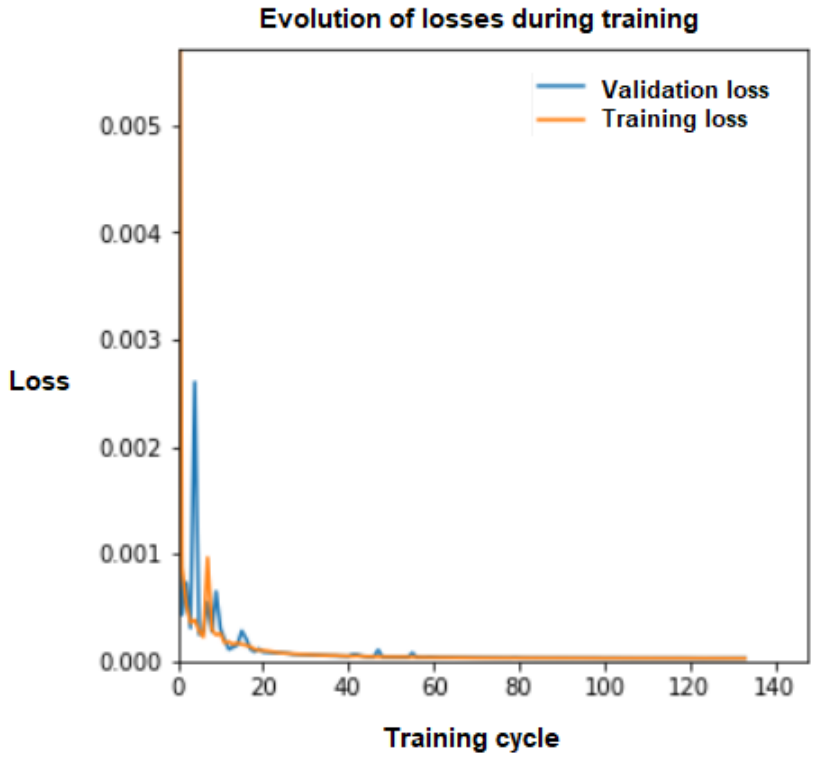


Figure 4.12: Evolution of reconstruction losses for the training set and the validation set. Both losses converge at around 130 training circles.

After feature extraction, we used the proposed comparison scheme to compare the negative control (DMSO) and other drugs, then calculate classification accuracies. The classification accuracies can be considered as a similarity measure, where an accuracy close to 50% indicates high levels of similarity and an accuracy close to 100% indicates low levels of similarity. Figure 4.13 shows the distribution of the similarity measures. The majority of drugs are considerably similar to the negative control. If we use a threshold of 0.8, 5 drugs are shortlisted; a threshold of 0.75 shortlists 9 drugs; 0.7 shortlists 16 drugs. We considered all three scenarios. Table 4.8 shows the three shortlists and the corresponding classification accuracies. The condensed representations associated with the shortlisted drugs proceed to the next stage in the pipeline — clustering analysis — to identify groups of drugs that induce similar phenotypic changes.

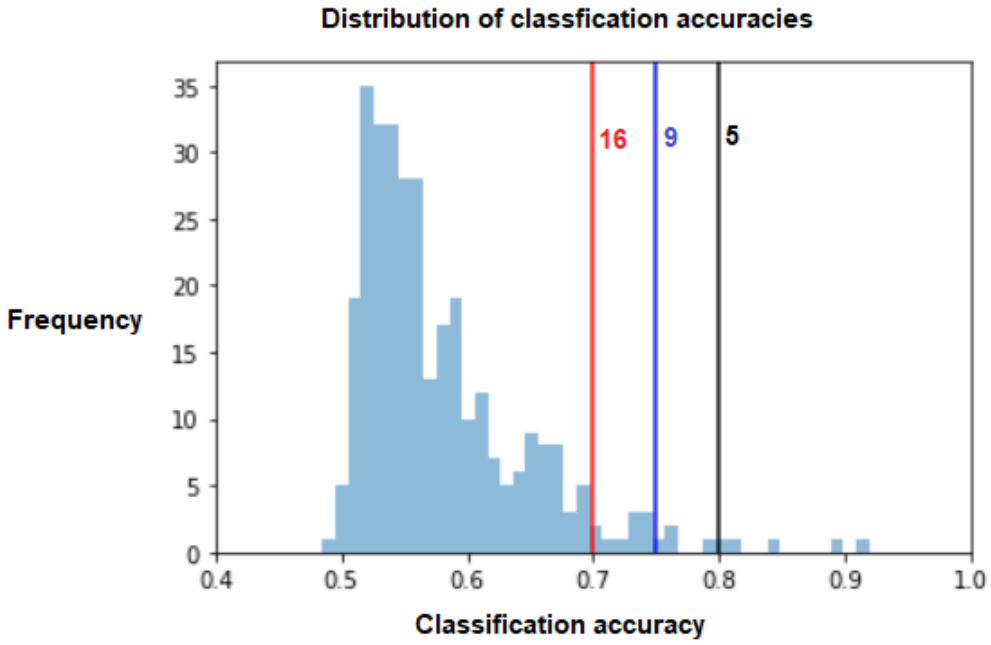


Figure 4.13: Distribution of similarity measures.

Table 4.8: Drug codes in each shortlist based on different thresholds. Acc.=Classification accuracy

Shortlist(0.8)	Acc.	Shortlist(0.75)	Acc.	Shortlist(0.7)	Acc.
27	0.917	27	0.917	27	0.917
63	0.894	63	0.894	63	0.894
252	0.842	252	0.842	252	0.842
87	0.814	87	0.814	87	0.814
86	0.808	86	0.808	86	0.808
		132	0.788	132	0.788
		115	0.767	115	0.767
		44	0.760	44	0.760
		209	0.755	209	0.755
				199	0.744
				177	0.740
				241	0.738
				39	0.736
				13	0.733
				88	0.728
				212	0.718

4.5 Chapter Summary

This chapter described the design and the implementation of hit selection steps. We managed to further compress cell windows to a total storage size of 28 GB using a convolutional autoencoder. This was a distilled version of the original raw data over 500 GB, retaining the most representative features. Furthermore, we compared features of the negative control (DMSO) with those of each drug. The result was a shortlist of 5 drugs that were the most capable of inducing significant phenotypic changes. Based on the threshold chosen, the shortlist could extend to 9 or 16 drugs. Next, we aim to identify clusters of drugs in the shortlist using their image features.

Chapter 5

Cluster Analysis Stage

This chapter discusses the last stage in the pipeline — cluster analysis. The content of this chapter includes an overview, preliminary explorations, the final design, results, and a brief summary.

5.1 Cluster Analysis Overview

Cluster analysis is the last stage in the pipeline, as shown in Figure 5.1. This stage is very similar to hit selection. In hit selection, condensed representations of cells (i.e. the extracted features) were compared with the negative control. Whereas in cluster analysis, representations associated with the shortlisted drugs are compared with each other. The result is a distance matrix showing similarity measures of each pair of shortlisted drugs. Due to the proven generalisability of the convolutional autoencoder as a dimensionality reduction tool, as discussed in Section 4.2.4, we can use the trained autoencoder for new unseen samples. Thus, no new training is required. A novel comparison scheme introduced in Section 4.2.1 produces all the similarity measures. The desired outputs at the end of this stage are clusters of drugs that have the potential to be repurposed.

Two issues are addressed in this stage:

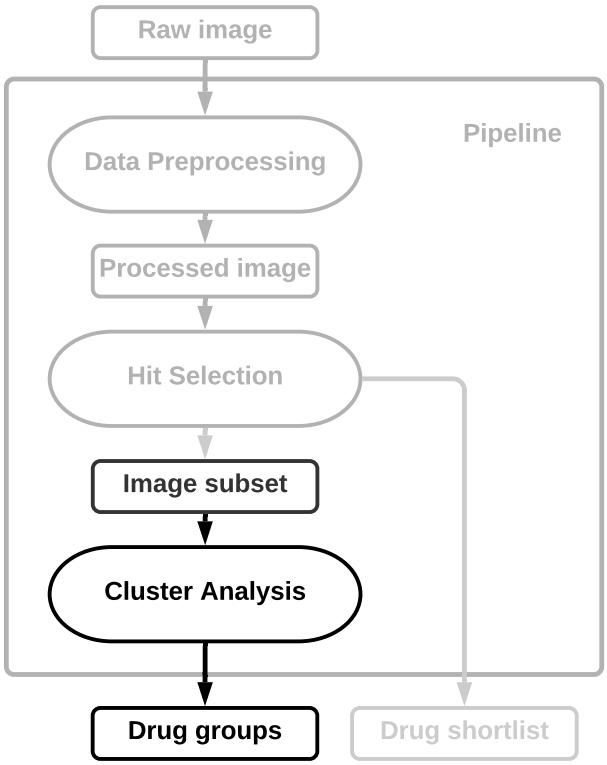


Figure 5.1: Cluster analysis is the last stage in the proposed pipeline.

1. Compare shortlisted drugs with each other and obtain a distance matrix;

2. Perform hierarchical clustering to obtain groups of drugs that have the potential to be repurposed.

5.2 Preliminary Explorations

This section describes some of the necessary explorations of data and methods. The explorations provide a better demonstration of the task as well as a collection of considerations behind the choices made in the final design.

5.2.1 Distance Matrix

The first step in cluster analysis is to obtain a distance matrix with all the similarity measures. This process is identical to the one we described in Section 4.2.4, except that the convolution autoencoder is already set up and trained. As a result, this stage is the most straightforward one in the pipeline. We use the comparison scheme to calculate the similarity measures. To better demonstrate the process, we again use the example from Section 4.2.4. From Section 4.2.4, we have a shortlist of 18 image groups significantly different from the negative control (i.e. the digit '5'). Next, we calculate the classification accuracy for each pair of the shortlisted image groups. Figure 5.2 shows the distance matrix obtained.

	0	0	1	1	2	2	3	3	4	4	6	6	7	7	8	8	9	9
0	0.497	0.491	0.999	0.999	0.987	0.991	0.997	0.997	0.997	0.997	0.994	0.994	0.998	0.998	0.991	0.991	0.994	0.993
0	0.491	0.502	0.999	0.999	0.987	0.987	0.995	0.997	0.997	0.997	0.994	0.993	0.997	0.997	0.991	0.991	0.993	0.993
1	0.999	0.999	0.503	0.495	0.994	0.995	0.995	0.997	0.998	0.997	0.998	0.998	0.994	0.994	0.995	0.996	0.998	0.997
1	0.999	0.999	0.495	0.507	0.995	0.994	0.997	0.997	0.998	0.997	0.998	0.998	0.995	0.994	0.995	0.996	0.998	0.998
2	0.987	0.987	0.994	0.995	0.495	0.504	0.988	0.989	0.993	0.992	0.996	0.995	0.985	0.985	0.985	0.984	0.992	0.990
2	0.991	0.987	0.995	0.994	0.504	0.505	0.989	0.991	0.993	0.994	0.996	0.996	0.986	0.985	0.981	0.983	0.991	0.991
3	0.997	0.995	0.995	0.997	0.988	0.989	0.499	0.505	0.996	0.997	0.997	0.998	0.991	0.991	0.971	0.975	0.986	0.986
3	0.997	0.997	0.997	0.997	0.989	0.991	0.505	0.506	0.998	0.998	0.998	0.998	0.991	0.990	0.972	0.974	0.985	0.986
4	0.997	0.997	0.998	0.998	0.993	0.993	0.996	0.998	0.496	0.503	0.994	0.994	0.988	0.990	0.993	0.992	0.947	0.944
4	0.997	0.997	0.997	0.997	0.992	0.994	0.997	0.998	0.503	0.502	0.992	0.993	0.991	0.990	0.993	0.992	0.948	0.952
6	0.994	0.994	0.998	0.998	0.996	0.996	0.997	0.998	0.994	0.992	0.505	0.499	0.999	0.999	0.992	0.993	0.997	0.998
6	0.994	0.993	0.998	0.998	0.995	0.996	0.998	0.998	0.994	0.993	0.499	0.501	1.000	0.999	0.992	0.992	0.998	0.998
7	0.998	0.997	0.994	0.995	0.985	0.986	0.991	0.991	0.988	0.991	0.999	1.000	0.499	0.495	0.992	0.993	0.969	0.969
7	0.998	0.997	0.994	0.994	0.985	0.985	0.991	0.990	0.990	0.990	0.999	0.999	0.495	0.501	0.992	0.991	0.969	0.971
8	0.991	0.991	0.995	0.995	0.985	0.981	0.971	0.972	0.993	0.993	0.992	0.992	0.992	0.992	0.503	0.490	0.984	0.984
8	0.991	0.991	0.996	0.996	0.984	0.983	0.975	0.974	0.992	0.992	0.993	0.992	0.993	0.991	0.490	0.499	0.981	0.983
9	0.994	0.993	0.998	0.998	0.992	0.991	0.986	0.985	0.947	0.948	0.997	0.998	0.969	0.969	0.984	0.981	0.502	0.505
9	0.993	0.993	0.997	0.998	0.990	0.991	0.986	0.986	0.944	0.952	0.998	0.998	0.969	0.971	0.984	0.983	0.505	0.502

Figure 5.2: Distance matrix for the shortlisted image groups that are different from digit '5', MNIST.

Considering that the digit labels were only revealed at the end, the comparison scheme did an excellent job classifying the same digits together. As shown in Figure 5.2, image groups associated with the same

digit have a classification accuracy of around 50%, indicating high levels of similarity. In addition, image groups associated with different digits have a classification accuracy of around 100%, indicating low levels of similarity. Furthermore, the classification process only took 19 seconds because we already had the features extracted from the hit selection stage. The classification accuracy and short process time were solid evidence that the comparison scheme worked as intended.

5.2.2 Hierarchical Clustering

Hierarchical clustering is a simple process that groups similar objects together one at a time based on a numerical similarity measure. For example, the most similar pair is clustered first, then the similarity measure for this cluster becomes the average of its elements' similarity measures. This step repeats until all objects are clustered into one group eventually. In this pipeline, the similarity measure has always been classification accuracy. However, we can add an additional layer of robustness to the clustering results using the correlation of classification accuracies as the similarity measure for clustering. Thus, instead of evaluating which pair are the most similar, we look at which pair has the most similar pattern in their corresponding classification accuracies with all other objects. Correlation can be calculated by Eq. (5.1), where r is the correlation, x_i's are classification accuracies of drug X with others, $\bar{x}$ is the mean accuracy for X, y_i's are classification accuracies of drug Y with others, $\bar{y}$ is the mean accuracy for Y.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5.1}$$

There are various packages available to automate this straightforward process, the built-in *hclust* function in R is a convenient choice. Figure 5.3 shows the clustering results. A correlation of 1 indicates high levels of correlation, while a correlation of 0 indicates low levels of correlation. For

the MNIST data set, '4', '9', and '7' are correlated at a very low level, '3' and '8' are correlated at a very low level.

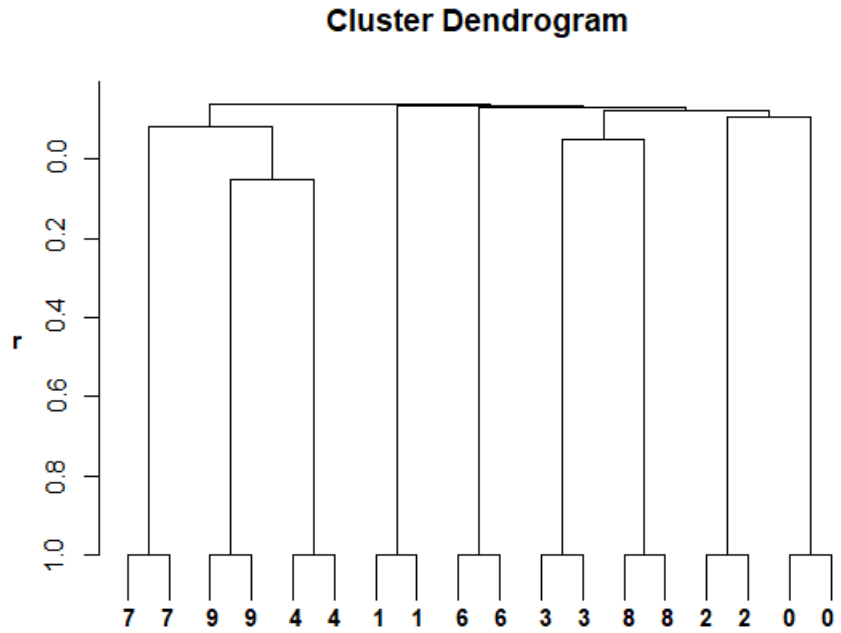


Figure 5.3: Dendrogram of hierarchical clustering using correlation (average).

There are various packages available to automate this straightforward process, the built-in *hclust* function in R is a convenient choice. Figure 5.3 shows the clustering results. A correlation of 1 indicates high levels of correlation, while a correlation of 0 indicates low levels of correlation. For the MNIST data set, image groups associated with the same digit are clustered together with a correlation around 1. This suggests impressive clustering results. Moreover, '4', '9', and '7' are correlated at a shallow level; '3' and '8' are correlated at a shallow level. These clustering results align with common perceptions.

5.3 Cluster Analysis Design

This section provides a concise and final design of the cluster analysis stage. Figure 5.4 outlines the steps taken in cluster analysis. Considerations behind the design are included in Section 5.2.

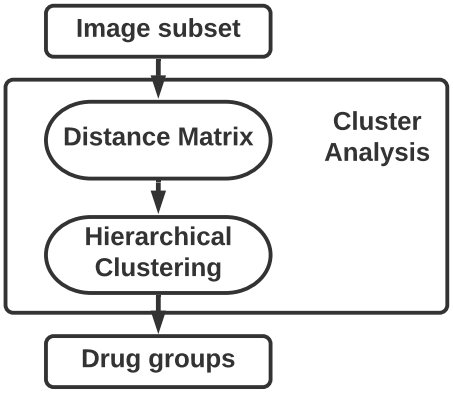


Figure 5.4: Detailed steps in the cluster analysis stage.

Distance Matrix

1. Calculate classification accuracies for each pair of drugs in the shortlist, and contruct a distance matrix.

2. Calculate the correlation between drugs based on classification accuracies.

Hierarchical Clustering

1. Apply the built-in *hclust* function in R to perform clustering analysis.

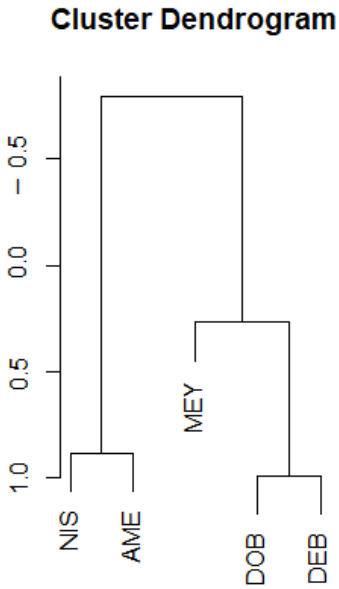
2. Examine the cluster dendrogram and make grouping decisions accordingly.

5.4 Results and Discussion

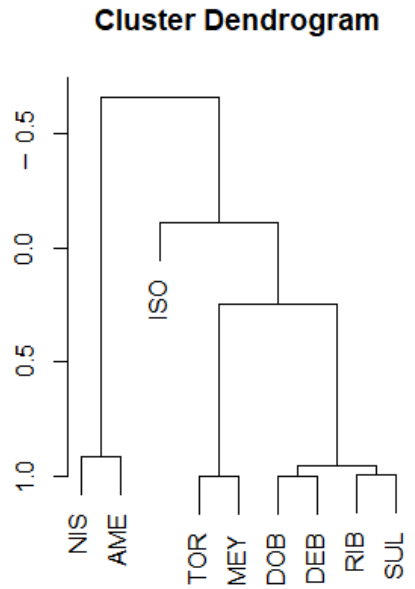
Figure 5.5 shows the distance matrix comprising classification accuracies of each pair of drugs. This distance matrix was for the shortlist with a 0.7 threshold, one of the hit selection outputs. Each drug on this shortlist had a classification accuracy of at least 0.7 with the negative control. Considering the classification was by pair, the accuracy for a particular pair of drugs did not change with the number of drugs in the list. Therefore, Figure 5.5 includes the distance matrice for the other two shortlists (threshold=0.75 and 0.80) as well. We put the abbreviations of drug names instead of drug codes because this was the final stage. Some clusters are obvious just from looking at the distance matrix. For example, TOR and MEY have almost identical classification accuracies.

	TOR	MEY	MET	CAR	ITO	DIF	NIS	ISO	GUA	DOB	ADE	BEN	DEB	AME	RIB	SUL
TOR	0.498	0.500	0.808	0.781	0.738	0.654	0.949	0.799	0.733	0.786	0.699	0.717	0.807	0.942	0.749	0.757
MEY	0.500	0.501	0.816	0.798	0.743	0.669	0.951	0.805	0.741	0.799	0.703	0.729	0.814	0.943	0.753	0.760
MET	0.808	0.816	0.498	0.899	0.875	0.825	0.884	0.846	0.913	0.929	0.898	0.908	0.934	0.904	0.906	0.902
CAR	0.781	0.798	0.899	0.499	0.627	0.736	0.959	0.889	0.650	0.625	0.660	0.667	0.605	0.966	0.626	0.624
ITO	0.738	0.743	0.875	0.627	0.494	0.691	0.955	0.870	0.681	0.679	0.690	0.683	0.666	0.959	0.629	0.610
DIF	0.654	0.669	0.825	0.736	0.691	0.498	0.909	0.760	0.722	0.764	0.710	0.714	0.783	0.900	0.716	0.729
NIS	0.949	0.951	0.884	0.959	0.955	0.909	0.498	0.914	0.966	0.973	0.963	0.965	0.978	0.644	0.973	0.973
ISO	0.799	0.805	0.846	0.889	0.870	0.760	0.914	0.500	0.847	0.895	0.812	0.832	0.907	0.927	0.883	0.887
GUA	0.733	0.741	0.913	0.650	0.681	0.722	0.966	0.847	0.499	0.554	0.526	0.506	0.597	0.965	0.565	0.614
DOB	0.786	0.799	0.929	0.625	0.679	0.764	0.973	0.895	0.554	0.501	0.571	0.583	0.523	0.972	0.581	0.608
ADE	0.699	0.703	0.898	0.660	0.690	0.710	0.963	0.812	0.526	0.571	0.492	0.514	0.603	0.963	0.558	0.618
BEN	0.717	0.729	0.908	0.667	0.683	0.714	0.965	0.832	0.506	0.583	0.514	0.497	0.614	0.963	0.570	0.610
DEB	0.807	0.814	0.934	0.605	0.666	0.783	0.978	0.907	0.597	0.523	0.603	0.614	0.494	0.979	0.577	0.594
AME	0.942	0.943	0.904	0.966	0.959	0.900	0.644	0.927	0.965	0.972	0.963	0.963	0.979	0.493	0.976	0.978
RIB	0.749	0.753	0.906	0.626	0.629	0.716	0.973	0.883	0.565	0.581	0.558	0.570	0.577	0.976	0.500	0.541
SUL	0.757	0.760	0.902	0.624	0.610	0.729	0.973	0.887	0.614	0.608	0.618	0.610	0.594	0.978	0.541	0.500

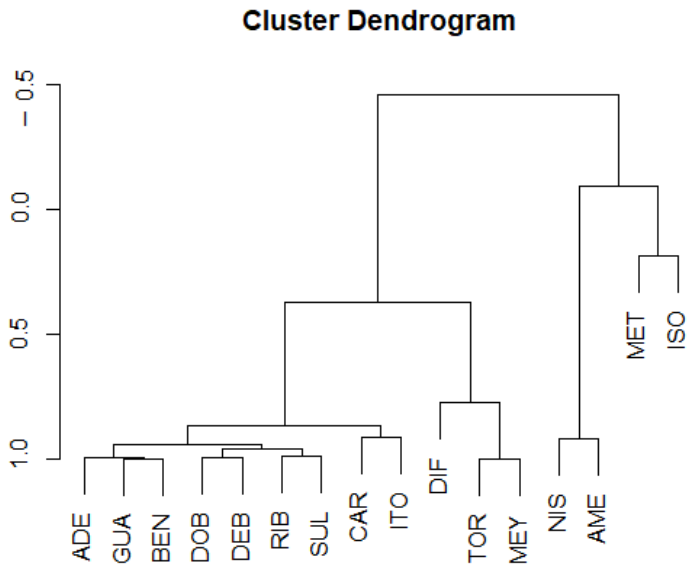
Figure 5.5: Distance matrix for the shortlist based on classification accuracy.



(a)



(b)



(c)

Figure 5.6: Cluster dendrograms based on different thresholds. (a) Shortlist threshold: 0.8. (b) Shortlist threshold: 0.75. (b) Shortlist threshold: 0.7.

Figure 5.6 shows the cluster dendrograms obtained by hierarchical clustering using correlation (average). The drug groups present in the shortlist with a 0.8 threshold remain unchanged in the shortlist with a 0.75 or 0.7 threshold. We did not know the true clustering structure of the data because there is none. There is no true answer as to which threshold to choose. All the clustering information is useful in future clinical trials or experiments to determine drugs that can be repurposed for new therapeutic uses. However, at this stage, we can look at some latent representations from each group and visually inspect their similarity.

First, we check the condensed representations and see if there is evidence for the clustering result. Figure 5.7 and Figure 5.8 show some of the extracted features projected from higher-dimensional latent space to two-dimensional space using UMAP [66]. Each data point in these figures represents a highly condensed representation of an image. These representations are organised under each drug and coloured by the drug group they are in. In Figure 5.7, we can clearly see separations between drug groups. For example, the drug group in orange colour are clustered to the lefthand side. Figure 5.8 shows distinctive groupings of data points. Blue dots are clustered at the top, away from other drug groups, while red dots occupy the bottom right corner. The separation suggests that the feature extraction was a success. Research has found that Methotrexate (AME) and Nisoldipine (NIS) are both repurposeable for treating treating rheumatoid arthritis [24]. While cellular biology is out of the scope of this thesis, that published research does support our findings. Next, we take a look at the real images of cells.

Figure 5.9 shows some processed windows of cells from different drug groups. The first two rows are from the blue drug group. We can see a similarity between these two rows of images, a piece of visual evidence that the comparison scheme was working in an intended way. Row three and four are from the yellow drug group and the red drug group, respectively. The reasoning behind this clustering is obvious. Compared with the first

two rows, row three contains significantly larger cells. In addition, the subcellular structures between rows three and four are quite different as the cells in row four have smaller nuclei. Row three cells also have spikes attached to their membranes.

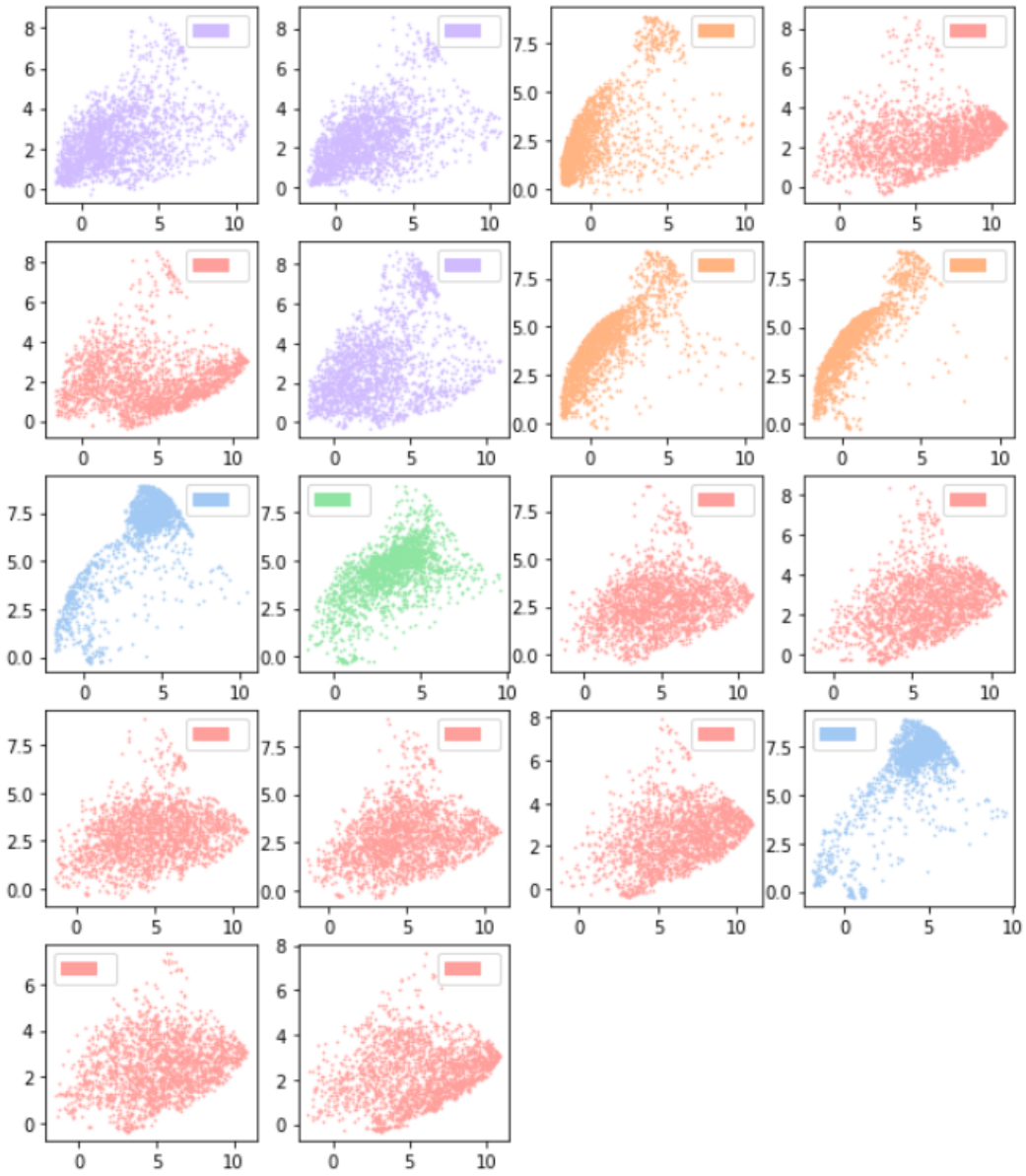


Figure 5.7: 2D projections of some extracted features. Each dot represents a two-dimensional representation of features. Features are coloured by drug group.

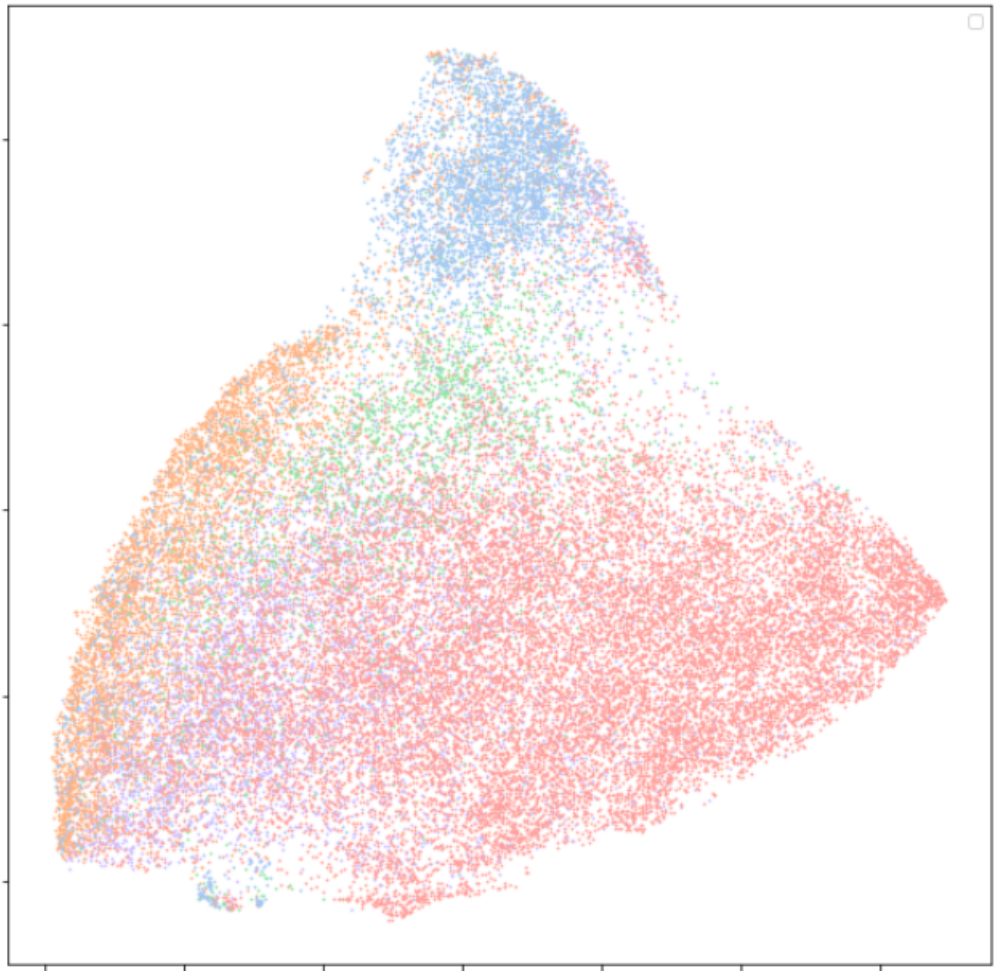


Figure 5.8: An overview of 2D projections of some extracted features. Each dot represents a two-dimensional representation of features. Features are coloured by drug group.

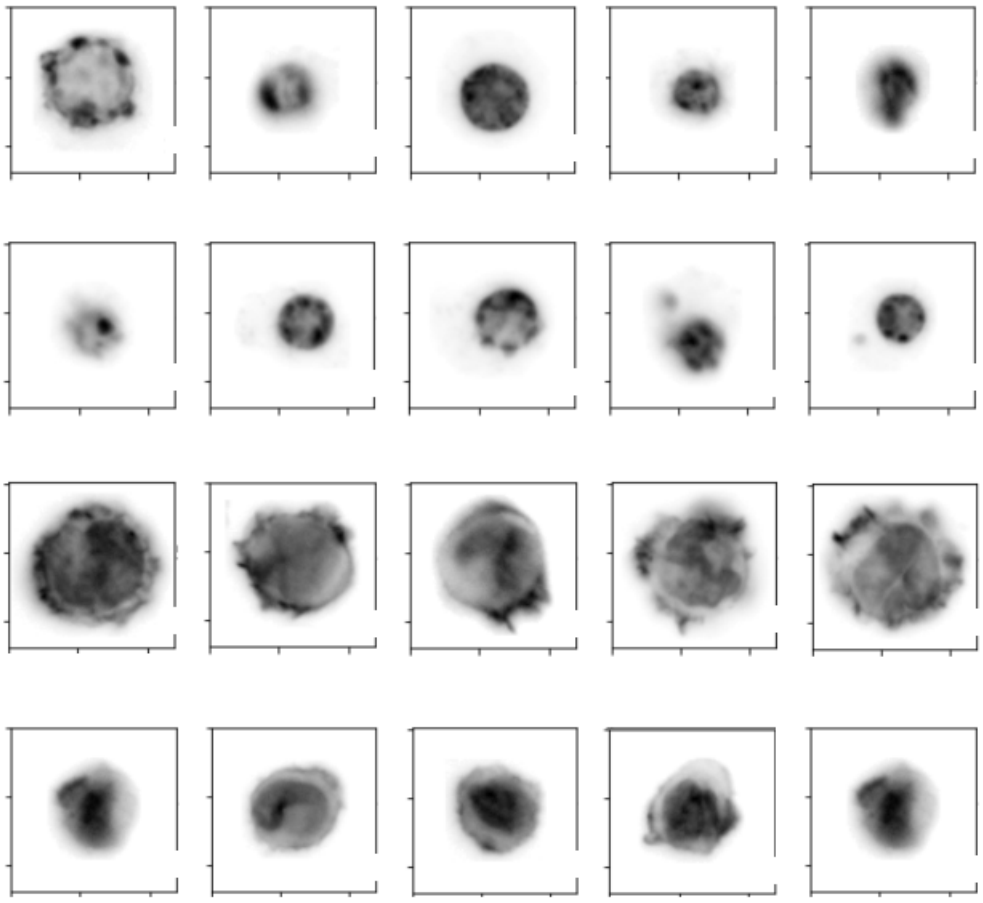


Figure 5.9: Some windows of cells from different drug groups.The first two rows are from the blue drug group, the third row is from the yellow drug group, the last row is from the red drug group.

5.5 Chapter Summary

This chapter described the design and the implementation of cluster analysis steps. Going in the clustering analysis, we had almost everything ready from the hit selection stage. For example, the condensed features were already ready. As a result, the only procedure in this stage was to

compare the features associated with each pair of shortlisted drugs. Using correlation of classification accuracy as a metric, we clustered the drugs for each shortlist and produced groups of drugs that induce similar phenotypic changes. The objectives set in this research have been fulfilled. Furthermore, we visually inspect the images as well as the extracted features of the images. The inspection firmly supports the clustering results.

Chapter 6

Conclusions

This thesis involved two major research areas, including statistics and machine learning. This thesis aimed to integrate the knowledge from both fields to develop an image analysis pipeline for high-content data. That goal was successfully achieved. Although the cell images were non-labelled, we used another labelled data set (MNIST) to demonstrate the efficiency and accuracy of the proposed pipeline. The results show that this pipeline is superior to other neural networks in terms of overall performance. Details of the comparison are included in Section 4.2.4. In particular, the proposed pipeline could achieve an unsupervised clustering accuracy of 99.6% (cross-validation with three digits unseen during the training phase) with just 4 minutes of training time. This was far less than the training time for other neural networks, e.g. DAC needed 30 minutes to achieve 78.1% accuracy. This outstanding performance arises from the close integration of mathematics and machine learning. The highlights of the proposed pipeline can be summarised below.

1. Simple neural network architecture;

2. Automatic tuning of hyperparameters using statistical modelling;

3. Does not require feature engineering or feature selection;

4. Does not require training labels or the number of clusters;

5. Does not require re-training for unseen clusters.

We applied the pipeline to a massive high-content data set of over 500 GB. By careful data management, we were able to compress the data set to 28 GB, around 5% of the original storage size. As a result, the condensed representations were much easier to maneuver and work with. Using the proposed pipeline, we produced three shortlists containing drugs that have the potential to be repurposed for new therapeutic uses. The proposed pipeline reduced high-content data processing time from days to just hours and at a higher accuracy. Details of the results are presented in Section 5. Our clustering results align with some published studies, described in Section 5.4. Ultimately, our work could help discovery new therapeutic uses for existing drugs.

Last but not least, we reviewed some statistical and machine learning techniques in Section 2, such as generalised linear models, generalised estimating equations, convolutional neural networks, and convolutional autoencoders. We looked at the conventional approaches and explored novel methods that are faster, more robust, and more accurate. We developed a novel comparison scheme that does not require any training yet is able to produce accurate similarity measures for unseen data.

There is so much more to explore in the combined field of mathematics and machine learning. For example, the upgrade from convolutional autoencoders to other more advanced dimensionality reduction tools can further improve overall performance. Instead of making AI programs more intelligent, we have learned that replacing AI with advanced mathematical algorithms would have saved more time. Model training in machine learning is a double-edged sword. On the one hand, it is effective and carefree; on the other hand, it becomes dependent on a predefined set of data. Sometimes, multiple training sessions are required for the neural network to adapt to new data — not ideal for many real-world scenar-

ios. Therefore it is always wise to ask if the problem really needs a neural network to begin with.

Bibliography

[1] AGGARWAL, C. C., HINNEBURG, A., AND KEIM, D. A. On the surprising behavior of distance metrics in high dimensional space. In *International Conference on Database Theory* (2001), Springer, pp. 420–434.

[2] AGRESTI, A. *Categorical Data Analysis*, 2nd ed. John Wiley and Sons, 2002, pp. 11–13.

[3] AGRESTI, A. *Statistical Methods for the Social Sciences*, 5th ed. Pearson, 2018, pp. 420–422.

[4] AKRAM, M. W., LI, G., JIN, Y., CHEN, X., ZHU, C., ZHAO, X., KHALIQ, A., FAHEEM, M., AND AHMAD, A. Cnn based automatic detection of photovoltaic cell defects in electroluminescence images. *Energy 189* (2019), 116319.

[5] ANDERSEN, E. B. Sufficiency and exponential families for discrete sample spaces. *Journal of the American Statistical Association 65*, 331 (1970), 1248–1255.

[6] BALAKIN, K. V., SAVCHUK, N. P., AND TETKO, I. V. In silico approaches to prediction of aqueous and dmso solubility of drug-like compounds: trends, problems and solutions. *Current Medicinal Chemistry 13*, 2 (2006), 223–241.

[7] BENGIO, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.

[8] BIANCHINI, M., AND SCARSELLI, F. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems 25*, 8 (2014), 1553–1565.

[9] BOUTROS, M., HEIGWER, F., AND LAUFER, C. Microscopy-based high-content screening. *Cell 163*, 6 (2015), 1314–1325.

[10] BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[11] CAICEDO, J. C., MCQUIN, C., GOODMAN, A., SINGH, S., AND CARPENTER, A. E. Weakly supervised learning of single-cell feature embeddings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 9309–9318.

[12] CHANG, J., AND WANG, L. Deep adaptive image clustering. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 5879–5887.

[13] CHEN, M., SHI, X., ZHANG, Y., WU, D., AND GUIZANI, M. Deep features learning for medical image analysis with convolutional autoencoder neural network. *IEEE Transactions on Big Data* (2017).

[14] CHOLLET, F., ET AL. Keras, 2015. `https://github.com/fchollet/keras`.

[15] COX, D. R. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological) 20*, 2 (1958), 215–232.

[16] CUI, J. Qic program and model selection in gee analyses. *The Stata Journal 7*, 2 (2007), 209–220.

[17] CUI, J., AND QIAN, G. Selection of working correlation structure and best model in GEE analyses of longitudinal data. *Communications in Statistics—Simulation and Computation 36*, 5 (2007), 987–996.

[18] CUMMING, G., AND FINCH, S. Inference by eye: confidence intervals and how to read pictures of data. *American Psychologist 60*, 2 (2005), 170.

[19] DAVIDIAN, M., AND CARROLL, R. J. Variance function estimation. *Journal of the American Statistical Association 82*, 400 (1987), 1079–1091.

[20] DAVIS, C. S. *Statistical Methods for the Analysis of Repeated Measurements*. Springer, 2002, pp. 174, 274–301.

[21] DOMINGOS, P. A few useful things to know about machine learning, 2012. Last accessed on 2021-07-1, `https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf`.

[22] DU, B., XIONG, W., WU, J., ZHANG, L., ZHANG, L., AND TAO, D. Stacked convolutional denoising auto-encoders for feature representation. *IEEE Transactions on Cybernetics 47*, 4 (2016), 1017–1027.

[23] EVERITT, B. S., AND SKRONDAL, A. *The Cambridge Dictionary of Statistics*, 4th ed. Cambridge University Press, 2010, pp. 419–421.

[24] FRAENKEL, L., BATHON, J. M., ENGLAND, B. R., ST. CLAIR, E. W., ARAYSSI, T., CARANDANG, K., DEANE, K. D., GENOVESE, M., HUSTON, K. K., KERR, G., ET AL. 2021 american college of rheumatology guideline for the treatment of rheumatoid arthritis. *Arthritis & Rheumatology* (2021).

[25] FREEDMAN, D. A. *Statistical Models: Theory and Practice*, revised ed. Cambridge University Press, 2009, pp. 22–26.

[26] FUKUSHIMA, K., AND MIYAKE, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition.

In *Competition and Cooperation in Neural Nets*. Springer, 1982, pp. 267–285.

[27] GÉRON, A. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. O'Reilly Media, 2019, pp. 4–15, 277–291.

[28] GIULIANO, K. A., CHEN, Y.-T., AND TAYLOR, D. L. High-content screening with siRNA optimizes a cell biological approach to drug discovery: defining the role of p53 activation in the cellular response to anticancer drugs. *Journal of Biomolecular Screening 9*, 7 (2004), 557–568.

[29] GIULIANO, K. A., DEBIASIO, R. L., DUNLAY, R. T., GOUGH, A., VOLOSKY, J. M., ZOCK, J., PAVLAKIS, G. N., AND TAYLOR, D. L. High-content screening: A new approach to easing key bottlenecks in the drug discovery process. *Journal of Biomolecular Screening 2*, 4 (1997), 249–259.

[30] GLOROT, X., BORDES, A., AND BENGIO, Y. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011), JMLR Workshop and Conference Proceedings, pp. 315–323.

[31] GOLDSTEIN, H., AND HEALY, M. J. The graphical presentation of a collection of means. *Journal of the Royal Statistical Society: Series A (Statistics in Society) 158*, 1 (1995), 175–177.

[32] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016, ch. 5-15. `http://www.deeplearningbook.org`.

[33] GOUGH, A. H., AND JOHNSTON, P. A. Requirements, features, and performance of high content screening platforms. *High Content Screening* (2007), 41–61.

[34] HALD, A. Sampling distributions under normality, 1876-1908. *A History of Parametric Statistical Inference from Bernoulli to Fisher, 1713-1935* (2007), 149–156.

[35] HALL, J., AND MARS, P. Limitations of artificial neural networks for traffic prediction in broadband networks. *IEE Proceedings-Communications 147*, 2 (2000), 114–118.

[36] HANEY, S. A. *High Content Screening: Science, Techniques and Applications*. John Wiley and Sons, 2008, pp. 1–15.

[37] 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.

[38] HEALEY, J. F. *The Essentials of Statistics: A Tool for Social Research*, 2nd ed. Cengage Learning, 2009, p. 177–205.

[39] HIN, L.-Y., AND WANG, Y.-G. Working-correlation-structure identification in generalized estimating equations. *Statistics in Medicine 28*, 4 (2009), 642–658.

[40] HINKELMANN, K. Neural networks, 2017. Last accessed on 2021-07-1, `http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf`.

[41] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[42] HOCHREITER, S., BENGIO, Y., FRASCONI, P., SCHMIDHUBER, J., ET AL. Gradient flow in recurrent nets: The difficulty of learning long-term dependencies, 2001.

[43] ILOUGA, P. E., WINKLER, D., KIRCHHOFF, C., SCHIERHOLZ, B., AND WÖLCKE, J. Investigation of 3 industry-wide applied storage conditions for compound libraries. *Journal of Biomolecular Screening 12*, 1 (2007), 21–32.

[44] INDYK, P., AND MOTWANI, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (1998), pp. 604–613.

[45] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning* (2015), PMLR, pp. 448–456.

[46] JOHNSON, J., DOUZE, M., AND JÉGOU, H. Billion-scale similarity search with gpus. *ArXiv preprint arXiv:1702.08734* (2017).

[47] KALCHBRENNER, N., GREFENSTETTE, E., AND BLUNSOM, P. A convolutional neural network for modelling sentences. *ArXiv preprint arXiv:1404.2188* (2014).

[48] KALUARACHCHI, T., REIS, A., AND NANAYAKKARA, S. A review of recent deep learning approaches in human-centered machine learning. *Sensors 21*, 7 (2021), 2514.

[49] KARPATHY, A. CS231n convolutional neural networks for visual recognition, 2015. Last accessed on 2021-07-1, `https://cs231n.github.io/neural-networks-1/`.

[50] KAUERMANN, G., AND CARROLL, R. J. A note on the efficiency of sandwich covariance matrix estimation. *Journal of the American Statistical Association 96*, 456 (2001), 1387–1396.

[51] KENNEY, J. F., AND KEEPING, E. S. *Mathematics of Statistics*, 3rd ed., vol. 1. Princeton, 1962, p. 252–285.

[52] KHAMPARIA, A., GUPTA, D., RODRIGUES, J. J., AND DE ALBUQUERQUE, V. H. C. Dcavn: Cervical cancer prediction and classification using deep convolutional and variational autoencoder network. *Multimedia Tools and Applications* (2020), 1–17.

[53] KLAMBAUER, G., UNTERTHINER, T., MAYR, A., AND HOCHREITER, S. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems* (2017), pp. 971–980.

[54] KUTNER, M. H., NACHTSHEIM, C. J., NETER, J., AND LI, W. *Applied Linear Statistical Models*, 5th ed. McGraw-Hill/Irwin, 2005, p. 50.

[55] LAW, M. T., TRABOULSEE, A. L., LI, D. K., CARRUTHERS, R. L., FREEDMAN, M. S., KOLIND, S. H., AND TAM, R. Machine learning in secondary progressive multiple sclerosis: An improved predictive model for short-term disability progression. *Multiple Sclerosis Journal–Experimental, Translational and Clinical 5*, 4 (2019), 2055217319885983.

[56] LECUN, Y., BOSER, B., DENKER, J. S., HENDERSON, D., HOWARD, R. E., HUBBARD, W., AND JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation 1*, 4 (1989), 541–551.

[57] LECUN, Y., AND CORTES, C. MNIST handwritten digit database, 2010. Last accessed on 2021-07-1, `http://yann.lecun.com/exdb/mnist/`.

[58] LEE, J. A., UHLIK, M. T., MOXHAM, C. M., TOMANDL, D., AND SALL, D. J. Modern phenotypic drug discovery is a viable, neoclassic pharma strategy. *Journal of Medicinal Chemistry 55*, 10 (2012), 4527–4538.

[59] LIANG, K.-Y., AND ZEGER, S. L. Longitudinal data analysis using generalized linear models. *Biometrika 73*, 1 (1986), 13–22.

[60] LOO, L.-H., WU, L. F., AND ALTSCHULER, S. J. Image-based multivariate profiling of drug responses from single cells. *Nature Methods 4*, 5 (2007), 445–453.

[61] MAHESH, B. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet] 9* (2020), 381–386.

[62] MASCI, J., MEIER, U., CIREŞAN, D., AND SCHMIDHUBER, J. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks* (2011), Springer, pp. 52–59.

[63] MASTERS, D., AND LUSCHI, C. Revisiting small batch training for deep neural networks. *ArXiv preprint arXiv:1804.07612* (2018).

[64] MCCULLAGH, P., AND NELDER, J. A. *Generalized Linear Models*, 2nd ed. Chapman and Hall, 1989, pp. 30–32.

[65] MCDONALD, J. H. *Handbook of Biological Statistics*, 3rd ed. Sparky House Publishing, 2014, p. 203.

[66] MCINNES, L., HEALY, J., AND MELVILLE, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[67] MIOTTO, R., LI, L., KIDD, B. A., AND DUDLEY, J. T. Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. *Scientific Reports 6*, 1 (2016), 1–10.

[68] MOFFAT, J. G., VINCENT, F., LEE, J. A., EDER, J., AND PRUNOTTO, M. Opportunities and challenges in phenotypic drug discovery: An industry perspective. *Nature Reviews Drug Discovery 16*, 8 (2017), 531–543.

[69] MOORE, D. S., NOTZ, W., AND FLIGNER, M. A. *The Basic Practice of Statistics*, 5th ed. W. H. Freeman New York, 2010, pp. 400–405.

[70] MYUNG, I. J. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology 47*, 1 (2003), 90–100.

[71] NELDER, J. A., AND WEDDERBURN, R. W. M. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General) 135*, 3 (1972), 370–384.

[72] NG, A. Machine learning and ai via brain simulations, 2013. Last accessed on 2021-07-1, `https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-`

[73] PAN, W. Akaike's information criterion in generalized estimating equations. *Biometrics 57*, 1 (2001), 120–125.

[74] PATEL, J. K., AND READ, C. B. *Handbook of the Normal Distribution*, 2nd ed. Marcel Dekker, 1996, pp. 19–24.

[75] PAWLOWSKI, N., CAICEDO, J. C., SINGH, S., CARPENTER, A. E., AND STORKEY, A. Automating morphological profiling with generic deep convolutional networks. *BioRxiv* (2016), 085118.

[76] PECK, R., OLSEN, C., AND DEVORE, J. L. *Introduction to Statistics and Data Analysis*, 4th ed. Cengage Learning, 2012, pp. 639–640.

[77] PERLMAN, Z. E., SLACK, M. D., FENG, Y., MITCHISON, T. J., WU, L. F., AND ALTSCHULER, S. J. Multidimensional drug profiling by automated microscopy. *Science 306*, 5699 (2004), 1194–1198.

[78] RATNER, B. Variable selection methods in regression: Ignorable problem, outing notable solution. *Journal of Targeting, Measurement and Analysis for Marketing 18*, 1 (2010), 65–75.

[79] ROTNITZKY, A., AND JEWELL, N. P. Hypothesis testing of regression parameters in semiparametric generalized linear models for cluster correlated data. *Biometrika 77*, 3 (1990), 485–497.

[80] RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature 323*, 6088 (1986), 533–536.

[81] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson Education, 2009, p. 494.

[82] SAINANI, K. The importance of accounting for correlated observations. *PM&R 2*, 9 (2010), 858–861.

[83] SARANGI, S., SAHIDULLAH, M., AND SAHA, G. Optimization of data-driven filterbank for automatic speaker verification. *Digital Signal Processing 104* (2020), 102795.

[84] SCHEEDER, C., HEIGWER, F., AND BOUTROS, M. Machine learning and image-based profiling in drug discovery. *Current Opinion in Systems Biology 10* (2018), 43–52.

[85] SCHULD, M., SINAYSKIY, I., AND PETRUCCIONE, F. An introduction to quantum machine learning. *Contemporary Physics 56*, 2 (2015), 172–185.

[86] SHORTEN, C., AND KHOSHGOFTAAR, T. M. A survey on image data augmentation for deep learning. *Journal of Big Data 6*, 1 (2019), 1–48.

[87] SHULTS, J., AND CHAGANTY, N. R. Analysis of serially correlated data using quasi-least squares. *Biometrics* (1998), 1622–1630.

[88] STEINBACH, M., ERTÖZ, L., AND KUMAR, V. The challenges of clustering high dimensional data. In *New Directions in Statistical Physics*. Springer, 2004, pp. 273–309.

[89] TEDJOPURNOMO, D. A., BAO, Z., ZHENG, B., CHOUDHURY, F., AND QIN, A. A survey on modern deep neural network for traffic prediction: Trends, methods and challenges. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[90] VAN ROSSUM, G., AND DRAKE, F. L. *Python 3 Reference Manual*. CreateSpace, 2009.

[91] WALD, A. Tests of statistical hypotheses concerning several parameters when the number of observations is large. *Transactions of the American Mathematical Society 54*, 3 (1943), 426–482.

[92] WANG, L. *Support Vector Machines: Theory and Applications*, vol. 177. Springer Science & Business Media, 2005.

[93] WANG, M. Generalized estimating equations in longitudinal data analysis: a review and recent developments. *Advances in Statistics 2014* (2014).

[94] WANG, Y.-G., AND HIN, L.-Y. Modeling strategies in longitudinal data analysis: Covariate, variance function and correlation structure selection. *Computational Statistics & Data Analysis 54*, 12 (2010), 3359–3370.

[95] WEDDERBURN, R. W. M. Quasi-likelihood functions, generalized linear models, and the Gauss — Newton method. *Biometrika 61*, 3 (1974), 439–447.

[96] ZEGER, S. L., AND LIANG, K.-Y. Quasi-likelihood. *Wiley StatsRef: Statistics Reference Online* (2014).

[97] ZEGER, S. L., LIANG, K.-Y., AND ALBERT, P. S. Models for longitudinal data: A generalized estimating equation approach. *Biometrics* (1988), 1049–1060.

[98] ZHANG, L., LU, L., NOGUES, I., SUMMERS, R. M., LIU, S., AND YAO, J. Deeppap: Deep convolutional networks for cervical cell classification. *IEEE Journal of Biomedical and Health Informatics 21*, 6 (2017), 1633–1643.