Evolutionary Deep Learning

Author:
Emmanuel DUFOURQ

Supervisor:
Prof. Bruce A. BASSETT

Thesis Presented for the Degree of
Doctor of Philosophy

in the

Department of Mathematics and Applied Mathematics
University of Cape Town
and
African Institute for Mathematical Sciences

June 5, 2019

Declaration of Authorship

I, Emmanuel DUFOURQ, declare that this thesis titled, "Evolutionary Deep Learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Signed by candidate

Date: 06/June/2019

Declaration of Publications

- **Publication 1** Dufourq, E., Bassett, B. A., Automated Classification of Text Sentiment, SAICSIT 2017, South Africa. Presented in chapter 4.1 in this thesis.
- **Publication 2** Dufourq, E., Bassett, B. A., Automated Problem Identification: Regression vs Classification via Evolutionary Deep Networks, SAICSIT 2017, South Africa. Presented in chapter 5 in this thesis.
- **Publication 3** Dufourq, E., Bassett, B. A., Text Compression for Sentiment Analysis via Evolutionary Algorithms, PRASA-ROBMECH 2017, South Africa. Presented in chapter 6 in this thesis.
- **Publication 4** Dufourq, E., Bassett, B. A., EDEN: Evolutionary Deep Networks for Efficient Machine Learning, PRASA-ROBMECH 2017, South Africa. Presented in chapter 4 in this thesis.

Abstract

Evolutionary Deep Learning

by Emmanuel DUFOURQ

The primary objective of this thesis is to investigate whether evolutionary concepts can improve the performance, speed and convenience of algorithms in various active areas of machine learning research.

Deep neural networks are exhibiting an explosion in the number of parameters that need to be trained, as well as the number of permutations of possible network architectures and hyper-parameters. There is little guidance on how to choose these and brute-force experimentation is prohibitively time consuming. We show that evolutionary algorithms can help tame this explosion of freedom, by developing an algorithm that robustly evolves near-optimal deep neural network architectures and hyper-parameters across a wide range of image and sentiment classification problems. We further develop an algorithm that automatically determines whether a given data science problem is of classification or regression type, successfully choosing the correct problem type with more than 95% accuracy. Together these algorithms show that a great deal of the current "art" in the design of deep learning networks - and in the job of the data scientist - can be automated.

Having discussed the general problem of optimising deep learning networks the thesis moves on to a specific application: the automated extraction of human sentiment from text and images of human faces. Our results reveal that our approach is able to outperform several public and/or commercial text sentiment analysis algorithms using an evolutionary algorithm that learned to encode and extend sentiment lexicons. A second analysis looked at using evolutionary algorithms to estimate text sentiment while simultaneously compressing text data. An extensive analysis of twelve sentiment datasets reveal that accurate compression is possible with 3.3% loss in classification accuracy even with 75% compression of text size, which is useful in environments where data volumes are a problem.

Finally, the thesis presents improvements to automated sentiment analysis of human faces to identify emotion, an area where there has been a tremendous amount of progress using convolutional neural networks. We provide a comprehensive critique of past work, highlight recommendations and list some open, unanswered questions in facial expression recognition

using convolutional neural networks. One serious challenge when implementing such networks for facial expression recognition is the large number of trainable parameters which results in long training times. We propose a novel method based on evolutionary algorithms, to reduce the number of trainable parameters whilst simultaneously retaining classification performance, and in some cases achieving superior performance.

We are robustly able to reduce the number of parameters on average by 95% with no loss in classification accuracy. Overall our analyses show that evolutionary algorithms are a valuable addition to machine learning in the deep learning era: automating, compressing and/or improving results significantly, depending on the desired goal.

Acknowledgements

Many thanks to my supervisor, Professor Bruce Bassett, for his invaluable guidance, encouragement, support and words of extreme wisdom throughout this degree. He has been an incredible mentor and friend.

A million thanks to my parents for their daily support, for listening to every single detail I had to say in each conversation, for providing me with guidance, and continuous love and encouragement from the first day. A special thanks to Lauren for her infinite kindness, love and support.

Many thanks to the staff and Cosmology group at the African Institute for Mathematical Sciences. Thanks for welcoming me and providing a great working environment. Thanks to Martin Kunz to granting access to the Baobab cluster at the University of Geneva.

I would also like to thank the following people who have helped me or impacted my research in a positive way throughout this degree: Shankar Agarwal, Etienne Vos, Ethan Roberts, Michelle Lochner, Dee Knights, Pierre-Yves Lablanche, Nadeem Oozeer, Kimeel Sooknunan, Louwrens Labuschagne, Blake Cuningham, Yabebal Tadesse, Kayode Olaleye, Matthew Nixon, Mark Heerden, Arun Aniyan, Ian Durbach, Bubacarr Bah, Zafiirah Hosenie, Alireza Sadr, Kai Stats, Arrykrishna Mootoovaloo, Gilad Amar, Diego Cardenas, Anne and Mark Pennels, James and Karen Barnes, and Rene January.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

Contents

List of Figures

List of Tables

To my grandparents.

Chapter 1

Introduction

Progress in machine learning [177] has skyrocketed over recent years. Machine learning has been applied to a vast number of domains such as product recommendations, social media analysis, search engines, face identification and emotion analysis. Machine learning is transforming academic research by enabling researchers to gain insights into problems which would take humans a great amount of time. This transformation is also noticeable in industry where machine learning is being used to automate challenging tasks. The number of research articles published on machine learning has also drastically increased especially with the undeniable attention that deep learning [138, 229, 85] has received.

Amongst the large number of machine learning algorithms which have been proposed over the years is one which has been inspired by biological evolution. Just as Leonardo da Vinci and the Wright brothers studied the flight of birds to enable their works on creating a flying machine [219, 102], researchers have turned to nature to develop evolutionary computational methods (which use natural selection to blend and evolve a population of candidate solutions to find an optimal or near-optimal solution) [15, 270, 69, 14] to enable them to solve optimisation problems.

1.1 Purpose of the Study

The primary objective of this thesis is to investigate how evolutionary machine learning can be adapted and applied to various active areas of research. Based on literature surveys, we explore and investigate the modifications necessary to address these areas which have not previously been researched. The focus was not to investigate on how to create new state-of-the-art approaches, but rather to evaluate and investigate novel approaches. We formulate the following six objectives for this thesis.

1. Does the problem require estimating a continuous value (regression) or a discrete class label (classification)? This is perhaps the most basic question faced when tackling a new supervised learning problem. Humans are able to study a dataset and decide whether it represents a classification or a regression problem, and consequently make decisions which will be applied to the execution of a neural network. The objective is to propose a machine learning algorithm that can automatically identify the problem type with the aim of moving towards algorithms that are more intelligent. The proposed method will be applied to various classification and regression datasets.

2. Deep neural networks continue to show improved performance with increasing depth, an encouraging trend that implies an explosion in the possible permutations of network architectures and hyper-parameters for which there is little intuitive guidance. Following from the previous objective, we propose to investigate how an evolutionary algorithm can be applied to optimise neural network architectures and associated hyper-parameters. The proposed method will be investigated by experimenting on various problem types such as natural language processing and computer vision problems.

3. For adult humans, interpreting the underlying emotions in text is usually performed unconsciously and with apparent ease. We are able to recognize emotions in emails, sentiments in our social media feed and appreciate the subtle nuances of conflicting views in novels. Nevertheless, even for humans text can be notoriously easy to misinterpret. For machines, on the other hand, sentiment analysis is highly non-trivial. The objective is to propose and investigate how an evolutionary algorithm can be applied to this problem domain. The proposed approach will be evaluated on a number of text sentiment datasets.

4. Following the previous objective, we pose the following question, can textual data be compressed intelligently without losing accuracy in evaluating sentiment? There are often a large number of redundant words which humans use when expressing their sentiment about something. The objective is to propose an evolutionary algorithm which can compress text in such as way that the classification accuracy is not significantly impacted. The proposed approach will be evaluated on a number of text sentiment datasets.

5. There have been a large number of studies focusing on using convolutional neural networks for facial expression recognition. We propose as an objective, to provide a critical analysis of works using convolutional neural networks for facial expression recognition. The analysis should enable researchers who are new to the field to rapidly gain knowledge as to different available approaches. Furthermore, the objective is to provide established researchers with unanswered questions which need addressing.

6. One challenge when implementing convolutional neural networks for facial expression recognition is the large number of trainable parameters. This can result in large training times and the inability to train deep networks on limited hardware. Following the previous objective, we propose to investigate how evolutionary machine learning can be incorporated with convolutional neural networks in order to achieve good predictive performance while simultaneously reducing the number of trainable parameters. The proposed approach will be evaluated on various facial expression recognition datasets.

1.2 Contributions

This thesis makes the following contributions:

1. Experimenters and researchers make a lot of decisions when implementing neural networks. For example, when creating deep neural networks, the number of parameters must be selected in advance and furthermore, a lot of these choices are made based upon pre-existing knowledge of the data such as the use of a categorical cross entropy loss function. Humans are able to study a dataset and decide whether it represents a classification or a regression problem, and consequently make decisions which will be applied to the execution of the neural network. There have not been any previous attempts to propose an algorithm that can automatically discriminate between classification and regression problems and to automatically decide on the loss function and number of units in the last layer of a neural network. We propose the Automated Problem Identification (API) algorithm, which uses an evolutionary algorithm interface to TensorFlow to manipulate a deep neural network to decide if a dataset represents a classification or a

regression problem. We test API on 16 different classification, regression and sentiment analysis datasets with up to 10,000 features and up to 17,000 unique target values. API achieves an average accuracy of 96.3% in identifying the problem type without hardcoding any insights about the general characteristics of regression or classification problems. These findings are revealed in chapter 4.1.

2. Following the previous contribution, we examined how to adapt an evolutionary algorithm to enable the automatic creation of deep neural networks. We propose Evolutionary DEep Networks (EDEN), a computationally efficient neuro-evolutionary algorithm which interfaces to any deep neural network platform, such as TensorFlow. We show that EDEN evolves simple yet successful architectures built from embedding, 1D and 2D convolutional, max pooling and fully connected layers along with their hyper-parameters. Evaluation of EDEN across seven image and sentiment classification datasets shows that it reliably finds good networks – and in three cases achieves state-of-the-art results – even on a single GPU, in just 6-24 hours. Our study provides a first attempt at applying neuro-evolution to the creation of 1D convolutional networks for sentiment analysis including the optimisation of the embedding layer. These findings are revealed in chapter 4.

3. The ability to identify sentiment in text, referred to as sentiment analysis, is one which is natural to adult humans. This task, however, is not one which a computer can perform by default. Identifying sentiments in an automated, algorithmic manner will be a useful capability for business and research in their search to understand what consumers think about their products or services and to understand human sociology. Here we propose two new genetic algorithms for the task of automated text sentiment analysis. The genetic algorithms learn whether words occurring in a text corpus are either sentiment or amplifier words, and their corresponding magnitude. Sentiment words, such as 'horrible', adds linearly to the final sentiment. Amplifier words in contrast, which are typically adjectives/adverbs such as 'very', multiply the sentiment of the following word. This increases, decreases or negates the sentiment of the following word. The sentiment of the full text is then the sum of these terms. This approach grows both a sentiment and amplifier dictionary which can be reused for other purposes and fed into other machine learning algorithms. We report the results

of multiple experiments conducted on large Amazon data sets. The results reveal that our proposed approach was able to outperform several public and/or commercial sentiment analysis algorithms. These findings are revealed in chapter 5.

4. Text reviews are often lengthy and contain a lot of extra words which do not contribute to the sentiment which is being expressed by the author. We propose a novel evolutionary compression algorithm, PARSEC (PARts-of-Speech for sEntiment Compression), which makes use of Parts-of-Speech tags to compress text in a way that sacrifices minimal classification accuracy when used in conjunction with sentiment analysis algorithms. An analysis of PARSEC with eight commercial and non-commercial sentiment analysis algorithms on twelve English sentiment data sets reveals that accurate compression is possible with (0%, 1.3%, 3.3%) loss in sentiment classification accuracy for (20%, 50%, 75%) data compression with PARSEC using LingPipe, the most accurate of the sentiment algorithms. Other sentiment analysis algorithms are more severely affected by compression. We conclude that significant compression of text data is possible for sentiment analysis depending on the accuracy demands of the specific application and the specific sentiment analysis algorithm used. These findings are revealed in chapter 6.

5. Humans are generally good at recognising emotions which are portrayed on another person's face. Can the same be said for machines? In recent years, there has been a tremendous amount of progress in the field of computer vision using deep learning methods, namely by convolutional neural networks. How good are these convolutional neural networks at recognising facial expressions? With the explosion of research outputs using convolutional neural networks for facial expression recognition in recent years, it is an appropriate time to review the state of the art in this field, provide a critical analysis of what has and has not been achieved, and synthesize recommendations for each step of the process needed for facial expression recognition. This work serves as a guide to those who are new to the field. This survey provides a critique of past work, highlights recommendations and lists some open, unanswered questions in facial expression recognition that deserve further investigation. The survey and analysis is presented in chapter 7.

6. The previous objective dealt with reviewing studies that use convolutional neural networks for facial expression recognition. Convolutional neural networks are known to result in a large number of neural network parameters. Can these parameters be reduced intelligently in such a way as to preserve the predictive ability of the convolutional neural network? We propose and investigate an evolutionary algorithm for facial expression recognition. The proposed algorithm extracts patches from an image and trains the convolutional neural network on the patches instead of the entire face. The objective was to minimise the number of parameters and maximise the classification accuracy. We demonstrate that the algorithm can reduce the number of parameters on average by 95% and simultaneously increase the classification accuracy. The method and findings are presented in chapter 8.

1.3 Thesis Layout

In this section we present the layout of the thesis and briefly describe the contents of each chapter.

Chapter 2 - Introduction to Machine Learning

The reader is introduced to machine learning. Supervised machine learning problems are typically either regression or classification problems. In this chapter we present the characteristics and discuss both classification and regression problems. We define the metrics which are used to evaluate such problems and introduce the reader to terminology used throughout the thesis. The chapter then discusses evolutionary algorithms and presents the reader with the fundamental knowledge required to understand the method. The various aspects of the method are described in detail. Finally, the chapter introduces deep neural networks.

Chapter 3 - Neuro-evolutionary Problem Identification

Researchers and experimenters typically make certain premeditated decisions based on whether they are addressing a classification or regression problem. In this chapter we present a novel approach which enables an algorithm to automatically make these decisions using evolutionary machine learning.

Chapter 4 - Neuro-evolutionary Architecture Optimisation

When using a neural network one has to make a lot of decisions with regards to the architecture of the network and the associated hyper-parameters. In this chapter we present an evolutionary approach which attempts to optimise neural network architectures and the hyper-parameters.

Chapter 5 - Automated Classification of Text Sentiment

The ability to identify sentiment in text, referred to as sentiment analysis, is one which is natural to adult humans. In this chapter we propose an evolutionary algorithm for text sentiment analysis. We report the results of multiple experiments conducted on large Amazon data sets.

Chapter 6 - Text Compression for Sentiment Analysis via Evolutionary Algorithms

Following the previous contribution, in this chapter, we propose a novel evolutionary compression algorithm which makes use of Parts-of-Speech tags to compress text in a way that sacrifices minimal classification accuracy when used in conjunction with sentiment analysis algorithms.

Chapter 7 - Facial Expression Recognition and Convolutional Neural Networks

Humans are generally good at recognising emotions which are portrayed on another person's face. With the explosion of research outputs using convolutional neural networks for facial expression recognition in recent studies, it is an appropriate time to review the state of the art in this field, provide a critical analysis of what has and has not been achieved, and synthesize recommendations for each step of the process needed for facial expression recognition.

Chapter 8 - Evolutionary Facial Expression Recognition

The previous chapter reviews studies that use convolutional neural networks for facial expression recognition. The review reveals that convolutional neural networks achieve state-of-the-art performance as opposed to other machine learning methods. In this chapter, we explore a novel idea which attempts to optimise the predictive performance of convolutional neural networks for facial expression recognition and simultaneously, reduce the number of trainable network parameters without compromising on the classification accuracy.

Chapter 9 - Conclusion

We conclude this thesis by providing a summary of the findings which were observed throughout the experiments conducted in the previous chapters. We highlight the objectives presented in the first chapter and discuss how these have been met. This chapter provides direction for future research.

Chapter 2

Introduction to Machine Learning

2.1 Introduction

This chapter introduces the reader to machine learning. The following section introduces the general idea and history of machine learning. Section 2.3 introduces various terminology and presents the reader with the primary steps to undertake when implementing a machine learning algorithm. An introduction to evolutionary algorithms is presented in section 2.4 which is the fundamental technique used throughout this thesis. The various aspects of the evolutionary algorithm are discussed in detail. Section 2.5 provides an introduction to deep learning and equips the reader with the necessary knowledge for the methods implemented in this thesis. Finally, section 2.6 concludes this chapter.

2.2 Introduction to Machine Learning

Machine learning algorithms deal with the creation of models which enable a computer program to recognise patterns, learn relationships between data, make predictions and mimic the brain's ability to learn [177]. Mitchell [177] stated that "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E". Certain tasks, or problems are far too challenging and would take too long for a human to solve. Machine learning on the other hand has been used to solve a vast number of problems. For example, a large number of data classification tasks were solved using machine learning [126], more specifically using genetic programming in [65]. The performance measures are discussed in the following subsections. The experience, E, can be supervised or unsupervised. In supervised machine learning the problems have input variables for which

the algorithm can be applied on to obtain output values. Unsupervised machine learning refers to problems for which there are input variables but there are no output variables which need predicting. Here the machine learning algorithm learns useful properties of the structure of this dataset [85].

Early works in machine learning date back to 1950's where Arthur Samuel [224] proposed and invented an algorithm which played the board game checkers. Since then several breakthroughs were made, notably when Deep-Blue [26] outperformed chess champion Garry Kasparov. LeCun [137] demonstrated that a machine learning algorithm, namely a convolutional neural network could learn how to recognise hand written digits in 1998. Neural networks gained significant attention in 2012 when they were used to win the ImageNet challenge where the task is to correctly identify images from a corpus of millions of images [130]. More recently, DeepMind produced AlphaGo, a machine learning algorithm which outperformed grandmaster Lee Sedol, a champion at the game of Go in 2016 [236]. Figure 2.1 highlights a number of selected advances in machine learning from the 1950's to the present.

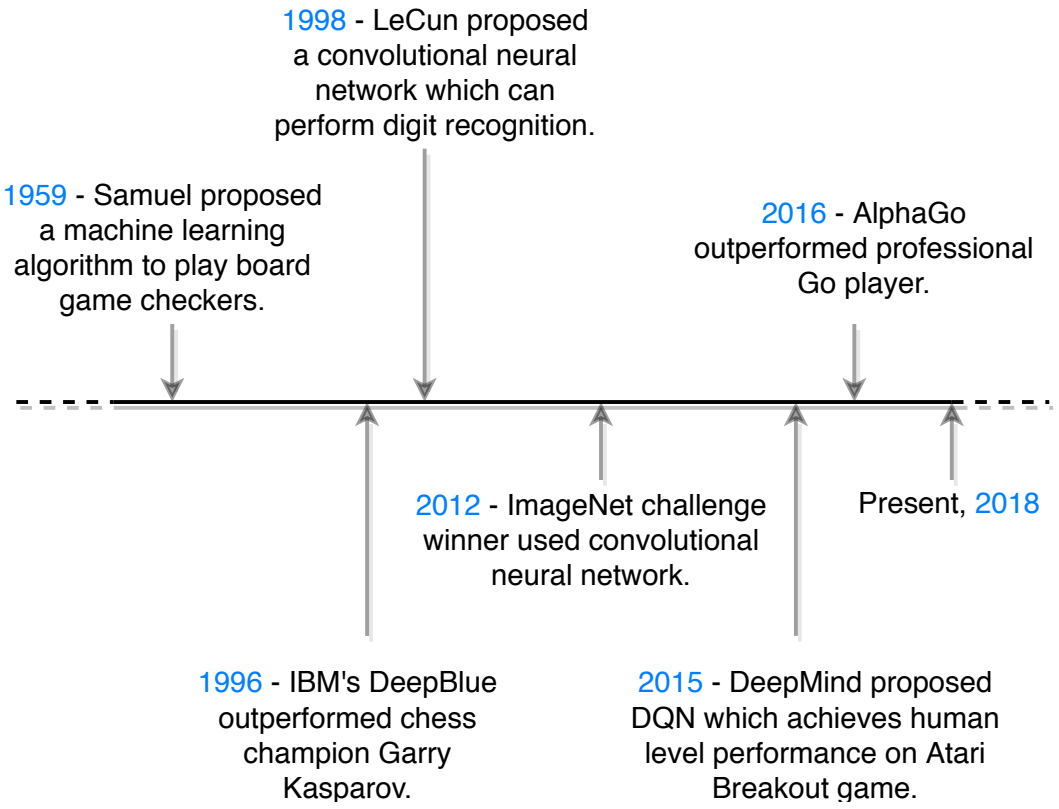

FIGURE 2.1: Timeline illustrating selected significant advances in machine learning from the 1950's to the present.

Applications of machine learning algorithms include, but are not limited to:

- sentiment analysis [193]
- assisting people with limited mobility [95]
- pneumonia detection [211]
- skin cancer classification [72]

Examples of machine learning algorithms include, but are not limited to:

- metaheuristics including evolutionary algorithms [22]
- neural networks [75]
- decision trees [127]
- support vector machines [43]

2.3 Applying Machine Learning

This section introduces various concepts and terminology associated with the implementation of machine learning algorithms. Figure 2.2 illustrates the steps which are performed in such implementations. These steps are explained in further details in the subsections below.

2.3.1 Dataset terminology

The first step in implementing a machine learning algorithm is to acquire a dataset. A *dataset* can be defined as a collection of records which encode some information about a particular problem. Each record, or row in a dataset is referred to as an *example* or *instance*. Thus, a dataset is made up of a number of examples. Each example in turn is made up of a number of *features* and often a *target*. The features, or attributes denote the characteristics of the examples and the target denotes the variable which must be predicted for each example [23]. Features can be discrete or continuous values. Thus, typically a dataset is made up of a number of examples and corresponding targets. There can be one or multiple features and, one or multiple targets. In classification tasks the target is also referred to as the *class*. A machine learning model is thus one which can map the input features to the targets [131]. In

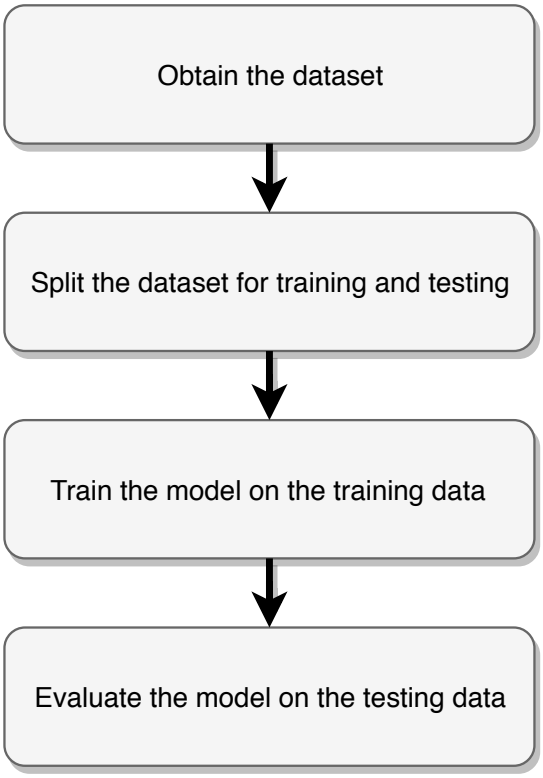

FIGURE 2.2: Primary steps involved when implementing a machine learning algorithm. The steps exclude any pre-processing that needs to be done to the dataset and hyper-parameter optimisation for the algorithm.

the case of a classification task the model is also referred to as the classifier. Figure 2.3 illustrates an example of a dataset which is made up of a number of examples. Each example has a number of features and a target.

2.3.2 Classification and Regression

In *classification* tasks the objective is to construct a model that can correctly predict the classes for as many examples as possible. A perfect classifier is one that correctly predicts all of the examples. The targets are discrete finite categories. Classification problems with two classes are referred to as binary classification, and if the problem has more than two then it is called multi-class classification. On the other hand, *regression* problems have continuous values in the targets. For regression, the objective is to construct a model that can predict values as close as possible to the targets in each example. A perfect model for a regression problem is one that predicts the exact values for all of the examples. Classification and regression problems are both supervised machine learning problems.

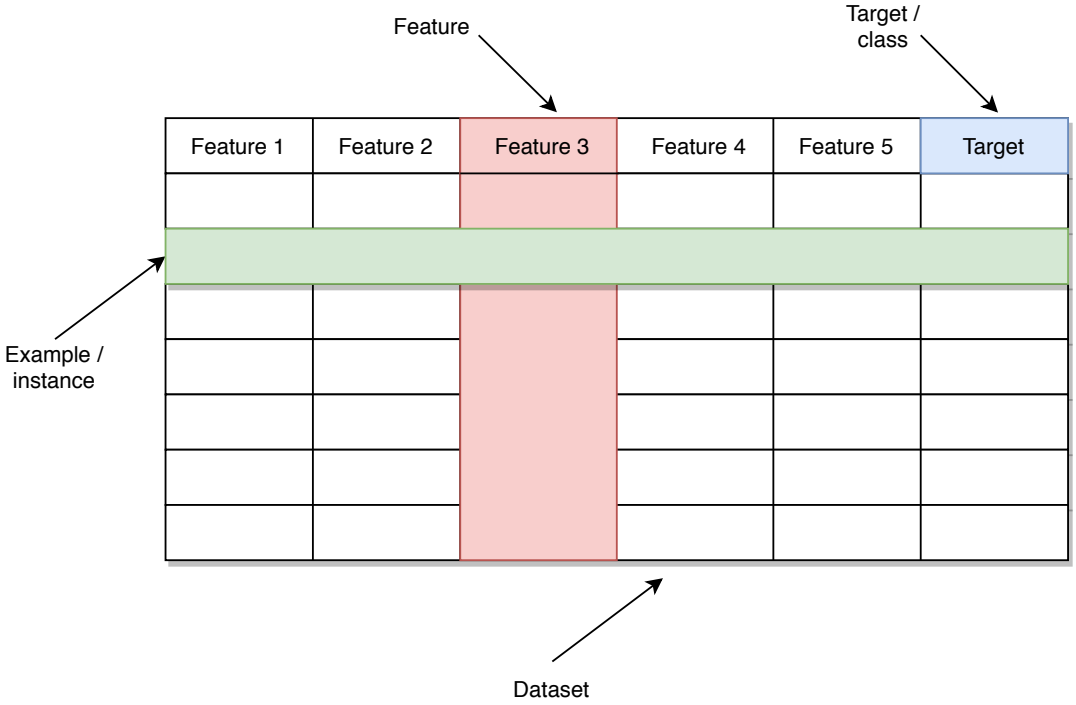

FIGURE 2.3: Highlighting the primary components related to the dataset terminology. The dataset is represented by the whole table. Each row in the dataset is an example, for which the examples are made up of a number of features and targets. In the figure there are five features and one target. Each feature is represented as a column, and the right most column is the target.

2.3.3 Data splitting

The dataset needs to be split so that a model can be evaluated and to determine its performance. Typically, the dataset will be split in such a way that there is training data for which the model learns from – this is the *training* phase. Furthermore, the dataset needs to be split in such a way that some of the data can be used to evaluate how the model is doing on data it has not seen before (i.e. an unbiased performance evaluation)[131] – this is the *testing* phase. The unseen data is referred to as the test data. The test data is used to evaluate the trained model. Training a model and evaluating its performance on the same training data is not a true representation of the model as the model can be fit perfectly to the training data and thus achieve perfect predictive ability on that data [280]. It is thus important that the training and testing data are disjoint sets [280]. The model is thus optimised on the training data and its performance is evaluated on the test data which was not used in the training process [280]. Two common methods for splitting a dataset are *k-fold cross-validation* and *holdout*.

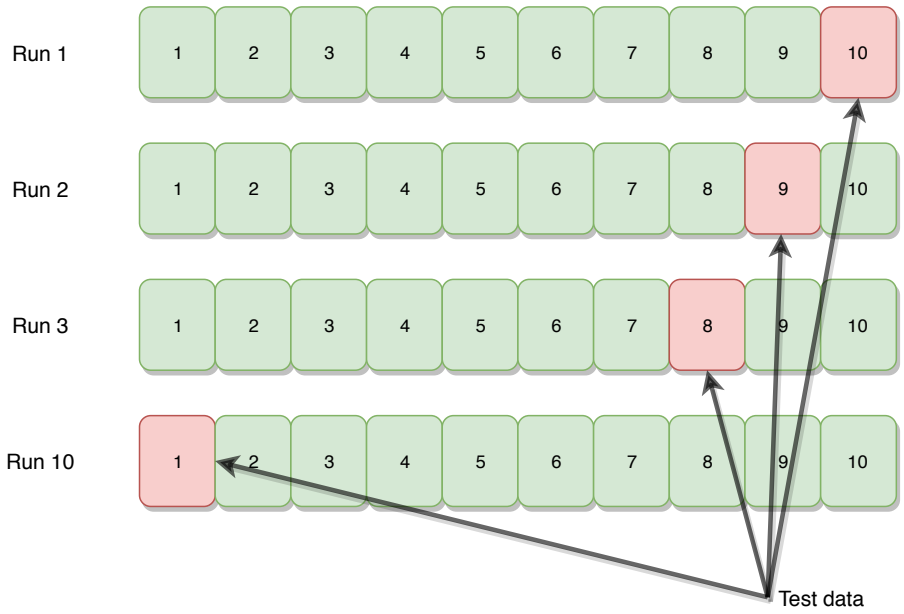

FIGURE 2.4: Illustrating 10-fold cross-validation. The dataset is split into 10 disjoint partitions and the algorithm is trained on 9 partitions and tested on the remaining one. The training data is coloured in green and the testing data in red. In the first run partition 10 is used as the test set. Then, the algorithm is executed again except this time the test set is another partition, in run 2 partition 9 is used as the test set. This is repeated 10 times until eventually the first partition is used for testing and the remaining ones are used for training.

In k-fold cross-validation the dataset is split in k disjoint partitions of approximately equal size and then k-1 partitions are used for training and the remaining partition k is used for testing [21]. Furthermore, each partition k is used exactly once for testing, and as a consequence each partition has a chance to be evaluated. Figure 2.4 illustrates how 10-fold cross-validation is used.

In holdout the dataset is split into two disjoint partitions, the training and the testing set. The training data is used for training and once this is complete the model is applied to the test data. The percentage of data used for training and testing varies amongst studies, however commonly used values are 2/3 and 1/3 for training and testing respectively [21]. Another variation of this approach is to split the full dataset into three disjoint partitions, the training, validation and test set. The rationale for doing this is that in the former approach (2 splits) the hyper-parameters of the model are being selected based on the test data. In the latter case (3 splits), the training data is used during the training process, and once complete the model is evaluated on the validation data. In this setting multiple models can be evaluated. The

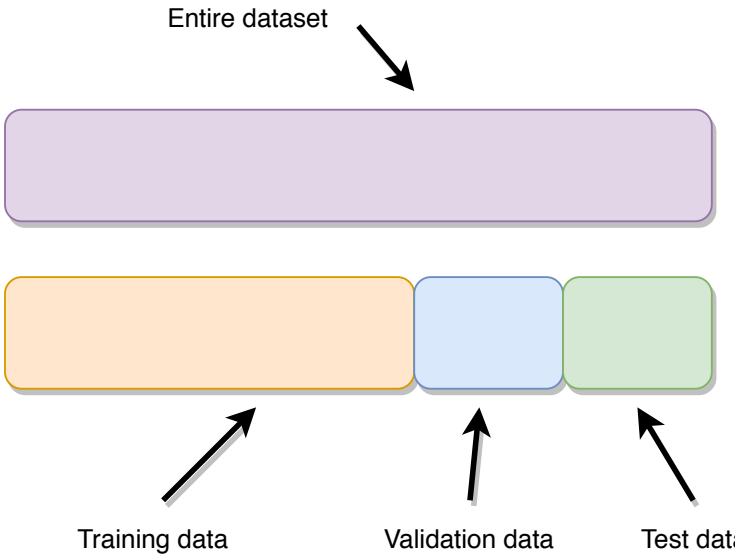

FIGURE 2.5: Illustrating how a dataset is split into training, validation and testing sets. Each set is disjoint. Models are trained on the training data, then evaluated on the validation data. Multiple models can be evaluated in this manner. The model which achieves the best performance on the validation data is then selected and applied to the test data.

model achieving the best performance on the validation data can then be applied to the test data. In this approach the hyper-parameters are fine-tuned to the validation data and not the test data. Other methods of splitting the dataset include bootstrapping and leave-one-out [280]. Figure 2.5 illustrates the holdout splitting.

2.3.4 Model Evaluation

The previous subsection mentioned that the model is applied and evaluated on the validation or testing data. In this section we introduce evaluation metrics. Evaluation metrics are problem dependent and vary amongst studies. The purpose of the metric is to define a function which returns a value that is used to evaluate the performance of the model, which ultimately answers the question: how good is the model on a particular dataset? Commonly used metrics include the accuracy, sensitivity, specificity, F1 score, log loss, mean absolute error, mean squared error metrics, precision and recall. It is common to use accuracy in classification problems, whereas the mean squared error is often used for regression problems. In this section we review the accuracy and mean squared error as it is used throughout the thesis. Additional information on the other metrics are detailed in [230].

Let y_i denote the target label for example i, $\bar{y}_i$ denote the model's predicted output for example i and n denote the number of examples.

The accuracy measure is defined as the number of correctly classified examples divided by the total number of examples. Accuracy can be computed for both binary and multi-class classification. The training and test accuracy are computed separately. The accuracy metric is defined in equations 2.1 and 2.2.

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} p_i}{n} \tag{2.1}$$

where

$$p_i = \begin{cases} 1 & \text{if } \bar{y}_i \text{ was correctly predicted} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The mean squared error is defined in equation 2.3. This function measures the distance between the model's prediction and the correct value. The result is squared to increase the penalty for predictions which are far from the correct values and also assists in dealing with negative and positive predictions and targets. The mean squared error is computed on the training and test data separately.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \bar{y}_i)^2 \tag{2.3}$$

2.4 Introduction to Evolutionary Algorithms

Evolutionary algorithms (EAs) are nature-inspired algorithms for which the processes performed in the algorithm takes ideas from the analogy of biological evolution [15, 270, 69, 14]. EAs are *population* based methods and are used to solve optimisation problems. The population consists of a number of *individuals* for which these individuals have particular representations (subsection 2.4.1 discusses this in detail). The individuals represent a solution to the optimisation problem. Biological processes are applied to a population of individuals; these processes *evolve* the population. The manner in which the population is represented and the way the population is evolved depends on the particular type of EA. There are various types of EAs such as *genetic algorithms* [84], *genetic programming* [128], *gene expression programming* [78] and *neuro-evolution* [234]. Neuro-evolution makes use of EAs to create neural networks [290]. EAs are stochastic and thus multiple executions of an

EA on a particular problem can result in various solutions. As a consequence the algorithm may not converge to the optimal solution on each execution. However, the stochastic nature does have a strength; multiple interesting solutions can be obtained as opposed to other machine learning methods that produce the same model upon each execution (e.g. decision trees [127]).

A genetic algorithm (GA) [84] is an evolutionary algorithm [69] inspired by 'survival of the fittest' in nature that can be used to solve optimisation problems. A GA evolves a population of *chromosomes* which are made up of several *genes*. Each gene is an input feature. The size of the population is a user-defined parameter. Each chromosome represents a candidate solution to the optimisation problem. Each chromosome is evaluated in order to determine how successful it is at solving the optimisation problem. The evaluation is obtained by computing the *fitness* of each chromosome. For a maximisation problem, a chromosome with a higher fitness is considered better, whereas a chromosome with a smaller fitness is considered weaker.

Algorithm 1 illustrates the pseudocode for a GA. An initial population of chromosomes is randomly created in step 2, and each chromosome is evaluated in step 3 to determine if a solution to the optimisation problem exists from the initial population. In step 5, the algorithm enters into a generational loop until the maximum number of generations is met, or until a solution to the optimisation problem is found. The maximum number of generations is a user-defined parameter. The various aspects are discussed in the sections which follow.

Algorithm 1: Genetic algorithm

input : generation_max: maximum number of GA generations
output: The best chromosome from the evolutionary process

```
1  begin
2      Create an initial population of chromosomes.
3      Evaluate the initial population.
4      generation ← 0.
5      while generation ≤ generation_max do
6          generation ← generation + 1.
7          Select the parents.
8          Perform the genetic operators.
9          Replace the current population with the new offspring created
             in step 8.
10         Evaluate the current population.
11     return The best chromosome.
```

2.4.1 Individual Representation

The representation of each individual in the population is dependent on the type of EA implemented. For genetic programming, the individuals are encoded using a tree structure (see figure 2.6 left) which are made up of functions and terminals [205]. In GAs, the individuals are referred to as chromosomes (see figure 2.6 right) and in turn, the chromosomes are made up of a number of genes. The chromosomes encode the solution to the problem for which the EA is being applied to. The experimenter decides on how to encode the genes and decides on the length of the genes. Each chromosome in the population represents a solution to the problem and thus, a GA searches and attempts to optimise the chromosomes in a solution space. The solution space consists of all the possible permutations of the possible values for which the genes encode.

2.4.2 Initial Population Generation

The first step in the algorithm is to initialise a population. This step is performed once the practitioner has decided upon a representation for each individual in the population. We can denote this population as generation zero. This initialisation is typically randomly performed so that the population can contain a variety of solutions which represent various candidate solutions to the optimisation problem. For genetic algorithms, the length of the chromosomes is either fixed or vary and this is dependent on the problem. Algorithm 2 presents the pseudocode to initialise a chromosome population of fixed size. The practitioner has to specify the population size in advance. When a chromosome is created, the genes which make up the chromosome are randomly selected from the available values. Once the initial population has been created, the next step is to evaluate each individual.

2.4.3 Fitness Evaluation

A *fitness function* [15] is defined to evaluate each individual in the population to compute its *fitness*. We can denote the fitness as a numerical value which represents the performance of the individual. The EA uses the fitness function to guide the population through the search space during the evolutionary process. The fitness obtained from the function should be able to distinguish between strong and weak individuals. The fitness function should enable the evolutionary process to distinguish between solutions which are

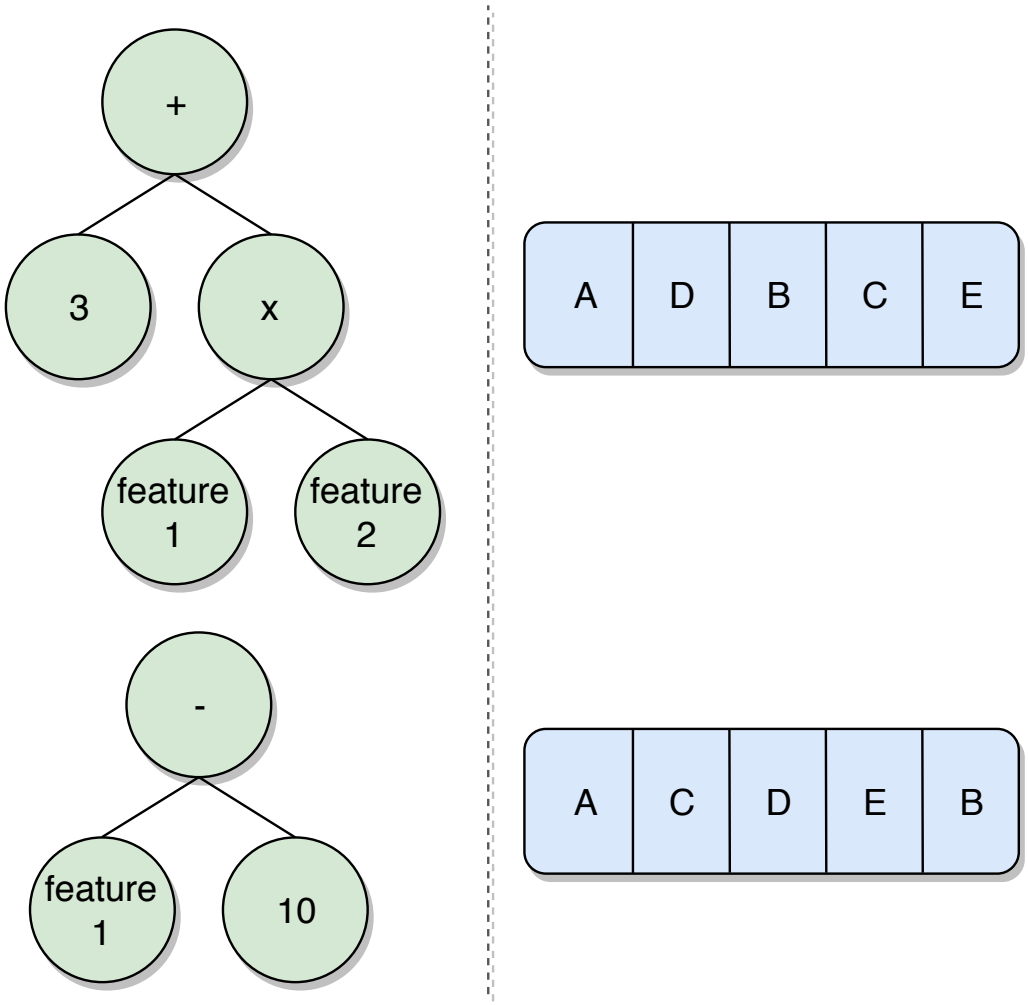

FIGURE 2.6: Illustrating two representations. On the left there are two parse trees, each parse tree is an indvidiual. This representation is used in genetic programming. In this example the parse trees encodes the following functions: $3 + feature_1 * feature_2$ and $feature_1 - 10$ where the features are input variables from some dataset. On the right there are two chromosomes, each chromosome is an individual. This representation is used in genetic algorithms. In this example the chromosome encodes a solution to the travelling salesman problem [11]. There are five genes and each gene encodes the order of the cities to visit. In the top chromosome, city A is visited first, then D, B, C, E and return back to city A. Different individual representations are used for the various types of evolutionary algorithms and problem domains.

far from an optimal solution, near an optimal solution and those which have reached an optimal solution.

Based on the problem which is being addressed the fitness should be defined to either be maximised or minimised. For example, in a classification problem one could define the function so as to maximise the classification performance. In a regression problem, one could define the fitness function

Algorithm 2: Creating an initial chromosome population.

```
input : population_size: the population size
input : size: the maximum length of a chromosome
output: The initial population
1 begin
2     for i ← 0 to population_size do
3         Initialise a chromosome with length equal to size
4         for each gene in the chromosome do
5             Randomly select a value for the gene from the available
              values
6         Add the chromosome to the initial population
```

such that the error between the correct values and the predictions are minimised.

The fitness function is applied to each individual once the initial population has been created, and is also applied whenever an individual is modified to compute its new fitness.

A fitness function can be single or multi-objective. A single objective fitness function evaluates the individuals based on a single metric, for example classification accuracy [23]. A multi-objective fitness function [300] uses several metrics to compute the fitness, for example the classification accuracy and the complexity of the chromosome. The user can define what the complexity represents. For example, assume that the number of genes in a chromosome denotes the complexity of the gene. Then a chromosome with 5 genes which achieves a classification performance of 60% should obtain a better fitness than another chromosome with 25 genes for which it achieves the same classification accuracy of 60%.

2.4.4 Parent Selection

Once a suitable fitness function has been defined the first step in the generational loop (step 7 in algorithm 1) can be executed, *i.e.* the parents can be selected. The *parent selection* method is a process that enables the EA to select certain individuals in the population which will act as parents for reproduction (discussed in the next section). A suitable parent selection method should be selected. For instance, one method could be to always select individuals in the population that have the best fitness. The result of such an approach is that the algorithm could become elitist and the population

could rapidly converge towards particular sub-optimal solutions. This approach does not allow for diversity amongst the different individuals and for weaker individuals to remain in the population. Weak individuals, through the evolutionary process can yield stronger solutions in future generations. Diversity in the population enables the EA to explore a variety of candidate solutions.

Common parent selection methods include fitness-proportionate, roulette wheel, rank and tournament selection (see [20] for comparisons). In this section we review the *tournament selection* method as it is frequently used.

The pseudocode for tournament selection is presented in algorithm 3 and is illustrated in figure 2.7. This selection method has one user-defined parameter, namely, the *tournament size*. Let k be the tournament size. Tournament selection randomly selects k chromosomes from the current GA population, and compares the fitness of each of the k chromosomes. The chromosome with the highest fitness is returned as the parent chromosome. If a tie occurs, then a random chromosome is selected to break the tie, or alternatively, some metric can be used. An execution of tournament selection returns a single parent. Thus if n parents are required, then the algorithm is executed n times.

The tournament size sets the selection pressure [175]. A large tournament size implies that a large number of chromosomes will be compared and will render the EA more elitist. If the tournament size is equal to the population size then the EA will most likely result in premature convergence towards a local optimum. Conversely, a small tournament size implies that few individuals are compared. Thus, the tournament size is an important hyper-parameter which must be fine-tuned to enable the EA to select appropriate parents for reproduction.

2.4.5 Genetic Operators

The previous section describes how parents are selected. Once the parents are selected then the *genetic operators* can be applied. The resulting individual from applying a genetic operator on a parent is referred to as an *offspring*. The parents are obtained from the population in the current generation, and the offspring are inserted into the population of the next generation. The initial population is randomly generated and thus most likely will not contain an optimal solution to some optimisation problem. The genetic operators are applied in step 8 in algorithm 1.

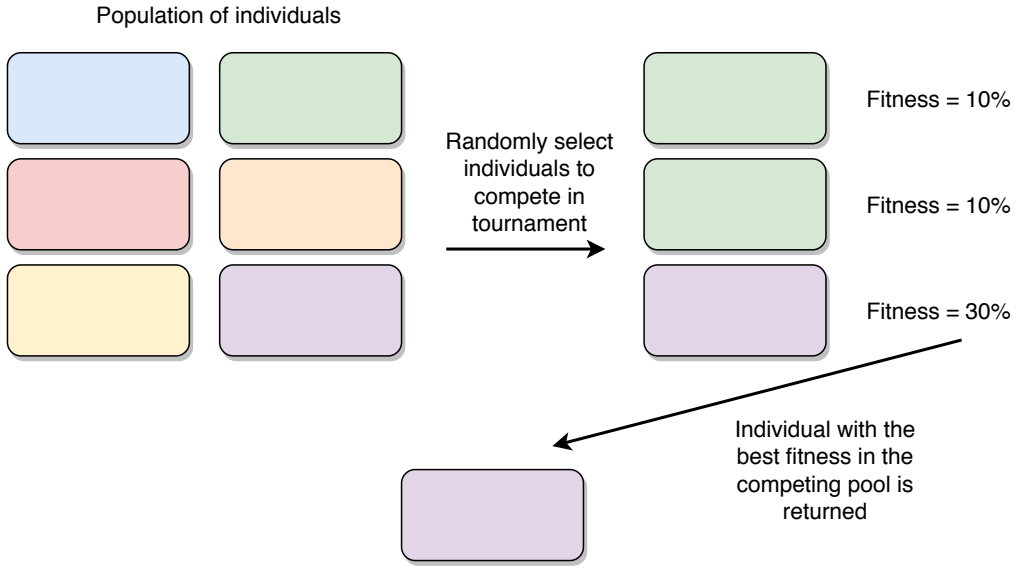

FIGURE 2.7: Illustrating tournament parent selection. There is a population of six individuals. In this example the tournament size is three and therefore three individuals are randomly selected from the population. Any individual can be randomly selected more than once. In this example the individual illustrated in green (with a fitness of 10% was selected twice). Finally, the individual in the pool with the highest fitness is returned as the parent. In this case, the individual with a fitness of 30% is returned.

Algorithm 3: Pseudocode for tournament selection.

```
   input : size: size of the tournament
   output: The best chromosome which will be used as a parent
1  begin
2      current_best ← null
3      for i ← 1 to size do
4          random_chromosome ← randomly select a chromosome from
             the population
5          Evaluate random_chromosome
6          if fitness of random_chromosome > fitness of current_best then
7              current_best ← random_chromosome
8      return current_best
```

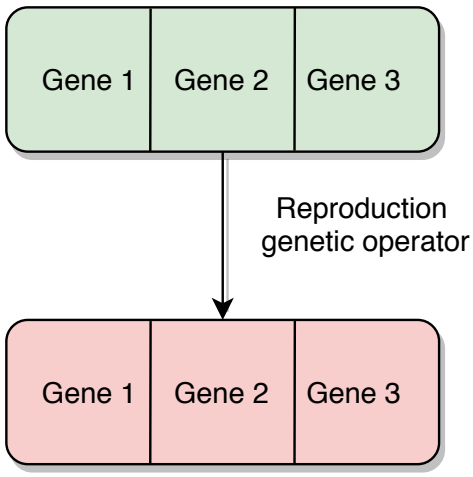

FIGURE 2.8: Illustrating the reproduction genetic operator. The chromosome on top (in green) is the parent chromosome. A copy of the parent is made and is the offspring (shown in red) of the reproduction genetic operator.

The role of the genetic operators is to modify the individuals so that the EA can traverse through the search space (combination of parameters and resulting fitness value) with the goal of finding the global optimal solution. The operators either enable the EA to explore or exploit the search space. Operators that enable exploring are those that make large changes to the individuals such that the offspring are in a different area of the search space to the parents. Conversely, operators that enable exploitation are those that produce offspring in a similar area of the search space to the parents. The combination of both exploration and exploitation is what enables the EA to get unstuck from local optimal solutions through exploration, but to also converge towards optimal solutions through exploitation. The user defines the amount of exploration and exploitation which the EA should conduct. The genetic operators depend on the problem domain. Thus for certain problems, contraints have to be put in place so that the modifications from the operators do not violate any rules of the problem domain.

In this section we review the three most commonly used genetic operators.

Reproduction

The reproduction operator is the simplest of the three. It uses the parent selection method to obtain one parent and then creates a copy of the individual. The copied individual is then inserted into the new population. Figure 2.8 illustrates the reproduction operator.

Mutation

The parent selection method is used to obtain a single parent and then a copy of it is created. A random element of the individual is selected and modified. In genetic programming, a single point is selected within the parse tree and an entire new sub-tree is generated at that point. In GAs a single gene can be randomly modified with a new one. This operator allows the EA to add genetic diversity to the individuals by randomly changing parts of the individuals. The mutation operator is illustrated in figure 2.9.

Crossover

Two parents are obtained using the parent selection method and copies of the parents are created. Genetic material between the parents are exchanged by randomly selecting part of parent 1 and parent 2, and then swapping them to create the offspring. When conducting this exchange the other parts of the individuals are usually not modified. Assume that two parents P_1 and P_2 are obtained. Now assume that a random part of P_1 is selected and denote that as S_1, and similarly S_2 is randomly selected from P_2. Then two offspring O_1 and O_2 can be created by duplicating all of the genetic material from P_1 and P_2 except for S_1 and S_2. The swap is performed by adding S_1 into O_2 and by adding S_2 into O_1. This operation allows the EA to exploit the local search space of the parents. Figure 2.10 illustrates the crossover operator.

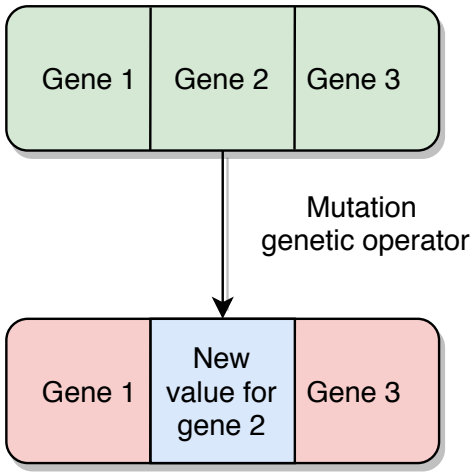

FIGURE 2.9: Illustrating the mutation genetic operator. The parent chromosome is shown on top (in green). This operator randomly selects some genetic material, in this case the second gene was randomly selected. The operator then changes the value at that position with a new one. The other genetic material is not affected. The offspring is shown at the bottom (in red) along with the new genetic material (in blue).

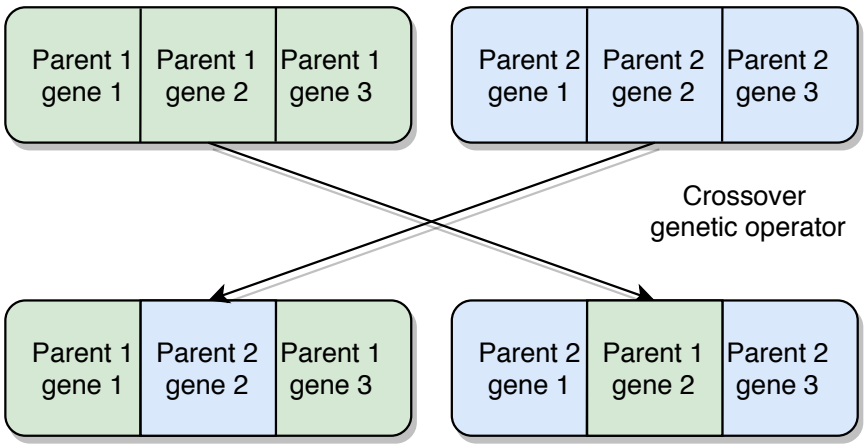

FIGURE 2.10: Illustrating the crossover genetic operator. The two parents are shown on top (first parent is shown on the left in green and the second parent is shown on the right in blue). This operator randomly selects some genetic material from each parent and swaps them whilst keeping the other genetic material constant. In this example the second gene from each parent was randomly selected and swapped to create two offspring shown on the bottom.

2.4.6 Generations and Termination

The EA iterates for a number of generations as is denoted in the while loop in step 5 from algorithm 1. Each generation has its own population of individuals. Assume that the population size is set to N. During each generation, g, parents are selected from the population of generation g, and then offspring are created and inserted into the population of generation $g+1$. Once the new population in generation $g+1$ is created (by creating N offspring), then the population from generation g is cleared. Thus, only a single population is retained in memory and evolved. In this manner, the populations iterate from one generation to another. The practitioner defines the termination criterion. For example, one could specify that the algorithm must iterate for G generations and then terminate. Alternatively, one could specify that if the fitness of the best individual across X generations does not improve then the EA must terminate. Figure 2.11 illustrates the generational loop.

2.5 Introduction to Deep Learning

Deep neural networks, or deep learning [138, 229, 85], have gained a lot of attention over recent years. There are a vast number of research articles that implement some variant of deep learning by either proposing new techniques or by applying it to various application domains. In the computer vision community deep learning has received a lot of attention.

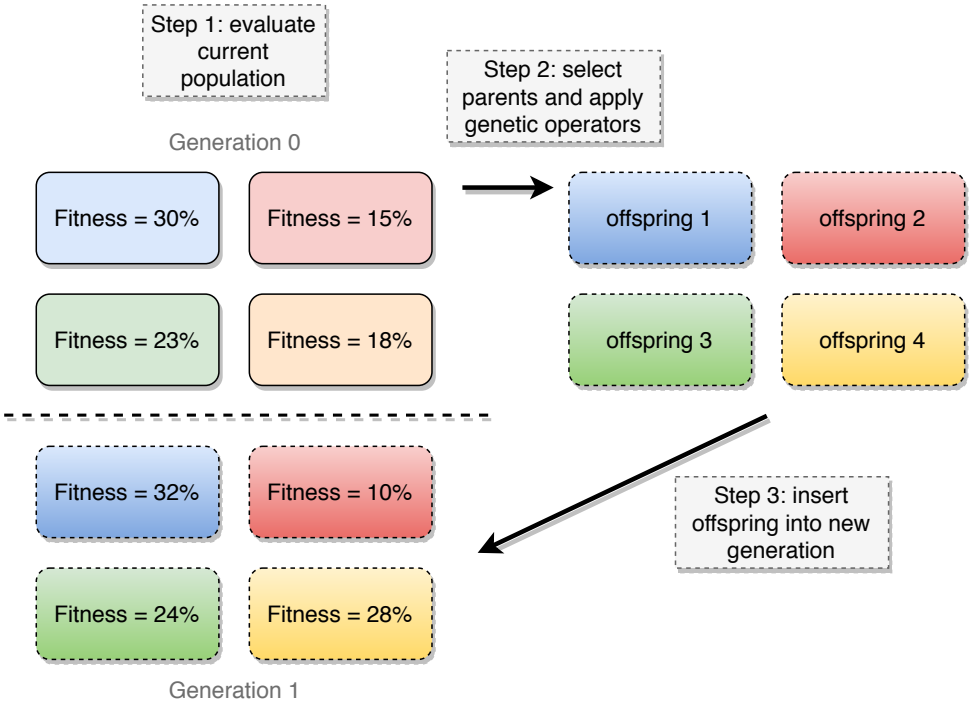

FIGURE 2.11: Illustrating the process of selecting parents from one generation, creating offspring and inserting the offspring into the new generation. In generation zero there are four individuals for which the first step is to evaluate each of them. Secondly, parents are selected using a parent selection method and then the genetic operators are applied to those parents to create offspring. The offspring are shown on the right (with dotted lines). Finally, the offspring are inserted into the new generation. The number of offspring in generation one is equal to the number of individuals in the previous generation. The new population is evaluated, and this process is repeated for a number of generations.

Applications of deep learning include:

- colouring grey scale images [34]
- real time pose estimation [262]
- automatic image annotation [182]
- self driving cars [104]
- video games [178]
- voice generation [189]

Neural networks are loosely inspired by neuroscience [85] in the sense that the neural networks are made up of weights connected to units which output a signal, which in turn resemble the dendrites connected to the cellular body which output through the axon [28, 76, 276]. Figure 2.12 illustrates

this analogy. Neural networks are not models of the brain, they instead represent a function approximation of some optimisation problem [85]. First, an introduction to the simplest unit, a perceptron, is presented. Then the remaining subsections discuss further details related to deep neural networks which are used throughout this thesis.

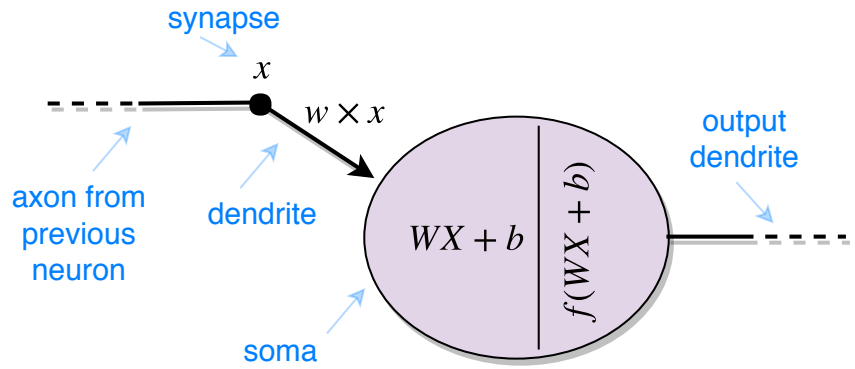

FIGURE 2.12: Illustrating the analogy which neural networks take from neuroscience. The input, x, is equivalent to the synapses, the multiplication of the weights and the input, $w \times x$, is equivalent to the dendrite, the application of the activation, $f(WX + b)$, is similar to the soma cellular body and the output of the function is similar to the output dendrite.

2.5.1 Perceptron

A perceptron maps some input vector **x** to a single output value. Perceptrons are binary classifiers and thus output either 0 or 1. To achieve this mapping, the perceptron uses some function $f(\mathbf{x})$. Equation 2.4 presents the function $f(\mathbf{x})$. The dot product is computed between the weights, **w**, (w_i denotes the i^{th} weight), and the input vector (x_i denotes the i^{th} vector component).

$$f(\mathbf{x}) = \begin{cases} 1 & if \ \sum w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Initially the weights are typically randomly initialised real values. Here the goal is to optimise the weights such that the network can learn which of the input features are important. Following this logic, larger weights enable the corresponding input features to have a greater contribution to the function. Similarly, smaller weights (or weights of value 0) enables the corresponding input to have minimal contribution. The perceptron is a single layer network.

In order to enable the perceptron to approximate a larger number of linear classifiers, a bias term can be added which allows the function to be shifted

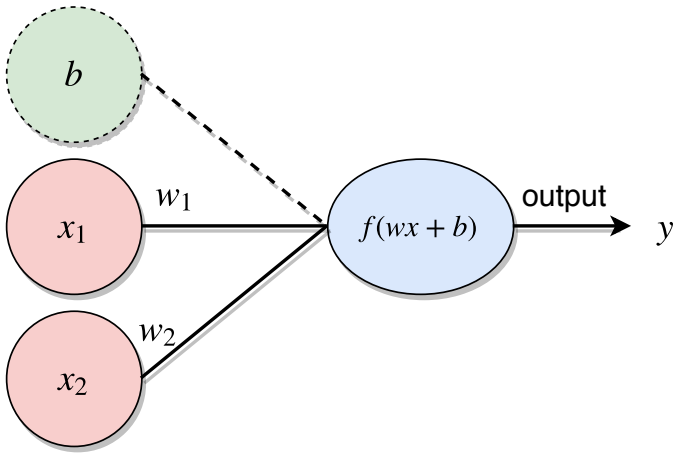

FIGURE 2.13: Illustrating a perceptron single layer network. There are two inputs, x_1 and x_2, and two weights w_1 and w_2. The perceptron is a linear classifier which computes the dot product between the input features and the weights and adds a bias term, b. The network produces an output y.

horizontally. Equation 2.5 presents the modified function $f(\mathbf{x}, b)$ to incorporate the bias term. Figure 2.13 illustrates an example of a perceptron along with the inputs, weights and bias.

$$f(\mathbf{x}) = \begin{cases} 1 & if \ \sum w_i x_i + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

2.5.2 Activation Functions

The function $f(\mathbf{x}, b)$ from the previous subsection was a simple thresholding function (referred to as a step function) which outputs either 1 or 0. The function enabled the perceptron to model linear classifiers. In general, the function which is applied to the inputs and weights is refered to as an *activation function*. One is not limited to only using the step function. Alternative functions include hyperbolic tangent, sigmoid, softmax and ReLU.

Equation 2.6 presents the ReLU activation function [183]. The function outputs a value of 0 when the input is less than zero, and the value x is output when the input is greater than zero.

$$f(x) = \max\{x, 0\} \tag{2.6}$$

The sigmoid activation function [232] is presented in equation 2.7. This function maps the input values to a bound, between 0 and 1, and can be interpreted as the probability of success.

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{2.7}$$

Equation 2.8 presents the softmax activation function [19]. This activation function is useful when dealing with multiclass classification problems. This is the case because the output of the values from the softmax function sum to 1, and additionally, each output is between 0 and 1. Each value is the probability associated with each class.

$$f(x) = \frac{e^{x_j}}{\sum_{i=1}^{D} e^{x_i}} \tag{2.8}$$

where

D = dimension of x (typically the number of classes)
j = 1, ..., D

Additional activations have been proposed over the years, these include leaky ReLU [160], PReLU [97] and ELU [38]. The choice of activation is important. For example, if the targets for some dataset are values which range from 1 to 5 then it does not make sense to use the hyperbolic tangent function as this outputs values between -1 and 1.

2.5.3 Multi-layer network

Subsection 2.5.1 discussed the perceptron, a single layer network. However, to enable the network to approximate more complex functions the network needs to contain more than one layer. This subsection introduces multilayer perceptrons (MLPs). The complexity arises from the fact that several functions can be connected in a chain. For instance, functions f_1 and f_2 can be connected as follows: $f(x) = f_2(f_1(x))$. Here the function $f(x)$ is more complex than the individual functions. Goodfellow *et al.* [85] describe this chain as the most commonly used structures of neural networks. In this example, function f_1 is referred to as layer 1 and f_2 is layer 2. Following this logic, any number of layers can be added in the chain, and hence the name 'deep neural networks' given that the length of the network can be large. The number of functions which are chained together is refered to as the depth of the neural network. Figure 2.14 illustrates a neural network with two layers, *i.e.*, the networks approximates the function $f(x) = f_2(f_1(x))$. The units are grouped vertically at each layer. The first group of units is the input layer, and the last group of units is the output layer. All other layers between the input and the

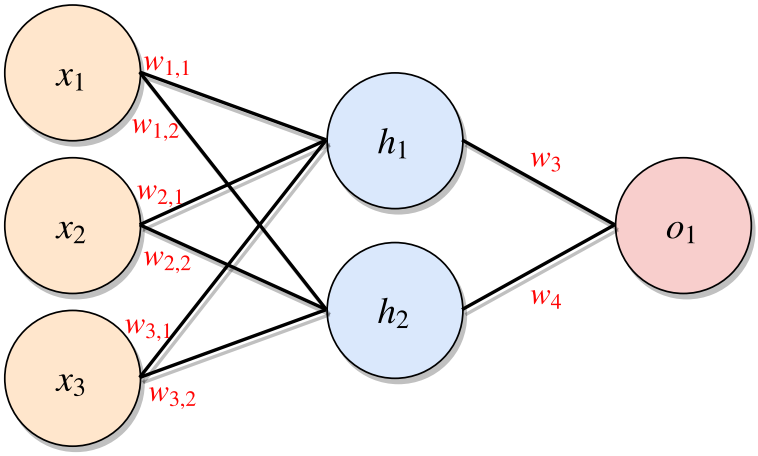

FIGURE 2.14: Illustrating a 2 layer neural network. There are three input units, two hidden units and one output unit. The units are stacked vertically. In the MLP, each unit is connected to every other unit in the previous layer. There are 6 weights in the first layer and 2 weights in the second.

output layer are refered to as hidden layers. In the figure there is one hidden layer.

From figure 2.14 there are three calculations which need computing. Assume that the sigmoid activation function is applied on each layer. First, the output value of h_1 must be computed. This is obtained by computing $(w_{1,1} \times x_1) + (w_{2,1} \times x_2) + (w_{3,1} \times x_3)$. Next the sigmoid activation function is applied to the previously computed value, let's denote this as h_1. Now the value of h_2 can be computed in a similar way. We compute $(w_{1,2} \times x_1) + (w_{2,2} \times x_2) + (w_{3,2} \times x_3)$ and apply the sigmoid activation function to that result, denote the result as h_2. Finally, to compute o_1 we need to compute $(h_1 \times w_3) + (h_2 \times w_4)$, and once again the sigmoid activation function is applied. This value is now the output of the neural network.

More generally, the previous layer acts as input to the next layer. The process which was described is refered to as a *forward pass* and the network is refered to as a feedforward network. Such a name is given because the information flows through the function and intermediate computations, and finally flows to the output $\mathbf{o}$ [85]. The output of an activation function is the input to the following layer. Figure 2.15 illustrates each of these computations graphically.

2.5.4 Optimisers

Here we review some commonly used optimisation algorithms for updating the weights of the neural network. Goodfellow *et al.* [85] state that there is no

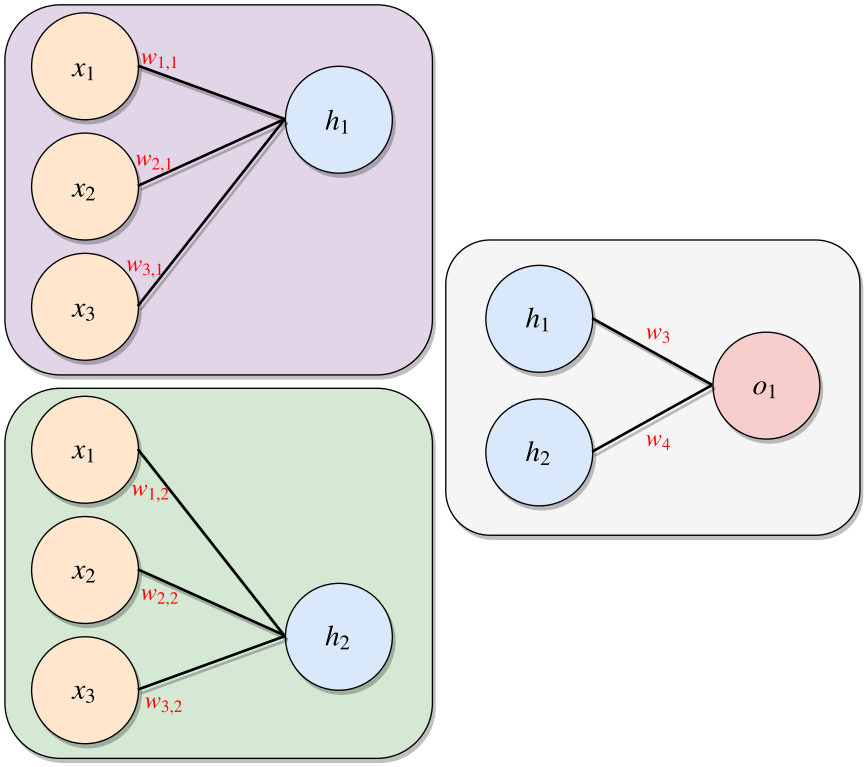

FIGURE 2.15: Illustrating the three separate computations required to perform a forward pass through the network in figure 2.14. Each block denotes a computation. The left two must be computed first. The output from those two are then used as input to the third block, and in turn, the output from this block is the output of the network.

best optimisation algorithm and that the choice is typically seletected based on the experimenters experience with the algorithm.

Gradient descent

Given some objective function, $J(\theta)$, and associated parameters, θ, we can apply the gradient descent algorithm to update the parameters and find solutions which minimise the objective function $J(\theta)$. The gradient of $J(\theta)$, at any θ, is the direction in which $J(\theta)$ rises the steepest. In order to minimise $J(\theta)$, one has to compute its gradient at the current value of θ, and take a step along the opposite direction, that is, move along the negative of the gradient of the objective function by computing $-\nabla_\theta J(\theta)$.

Gradient descent is an iterative algorithm and performs a number of steps. The *learning rate*, η, is a hyper-parameter which represents the size of each step. A large η can result in missing the local minimum, whereas a small η can result in very long optimisation times. This is illustrated in figure 2.16.

Equation 2.9 presents the parameter updates performed during a single step in the gradient descent algorithm. A termination criterion can be used to determine when to stop repeating the steps (for example, when the cost function $J(\theta)$ stops reducing over a number of steps). Figure 2.17 illustrates a network which will be used provide an example of the computations performed during a gradient descent step.

$$\theta = \theta - \eta \times \nabla_\theta J(\theta) \tag{2.9}$$

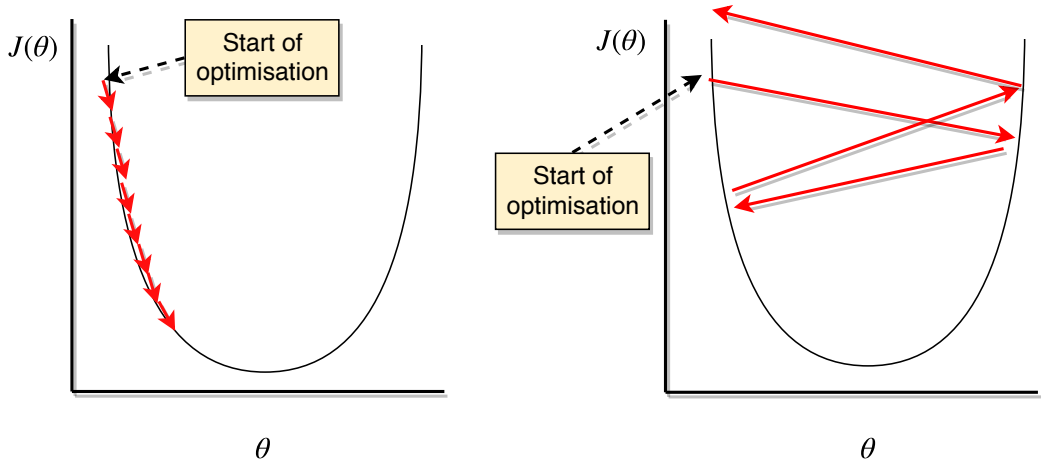

FIGURE 2.16: The effect of the learning rate, η. On the left, a small learning rate will result in many small gradient steps which can lead to long optimisation times. On the right, a large learning rate can result in missing the optimal solution.

Let $b = 1$ be the bias of the network in 2.17; the network can thus be expressed as $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$. Assume that the objective function is defined as $J(\theta) = (\hat{y}(\theta) - y)^2$, where $\hat{y}$ is the network's prediction, y is the correct value, θ_0, θ_1 and θ_2 are the network parameters θ. Following equation 2.9, we can perform the calculations in eqution 2.10 to compute the new values for the weights θ_1 and θ_2. In a similar way, we can perform the calculations in equation 2.11 to obtain the new value for the bias θ_0.

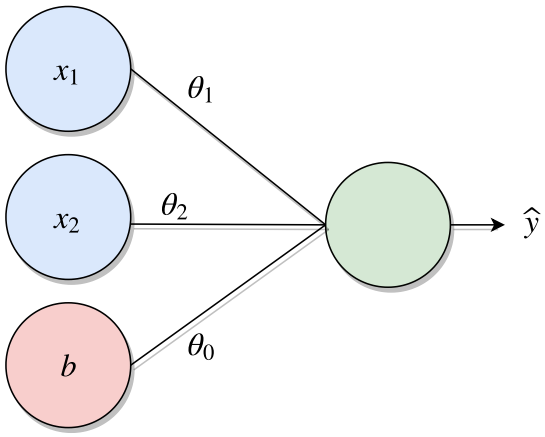

FIGURE 2.17: Network to illustrate gradient descent.

$$\begin{aligned}\theta_1 &= \theta_1 - \eta\frac{\partial J(\theta)}{\partial \theta_1} \\ &= \theta_1 - \eta(\frac{\partial J(\theta)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial \theta_1}) \\ &= \theta_1 - \eta(2(\hat{y} - y) \times x_1)\end{aligned} \tag{2.10}$$

$$\begin{aligned}\theta_2 &= \theta_2 - \eta\frac{\partial J(\theta)}{\partial \theta_2} \\ &= \theta_2 - \eta(\frac{\partial J(\theta)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial \theta_2}) \\ &= \theta_2 - \eta(2(\hat{y} - y) \times x_2\end{aligned}$$

$$\begin{aligned}\theta_0 &= \theta_0 - \eta\frac{\partial J(\theta)}{\partial \theta_0} \\ &= \theta_0 - \eta(\frac{\partial J(\theta)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial \theta_0}) \\ &= \theta_0 - \eta(2(\hat{y} - y) \times 1)\end{aligned} \tag{2.11}$$

Stochastic gradient descent

In *batch gradient descent*, the gradients ($-\nabla_\theta J(\theta)$) for each example in the entire training data are computed after which the parameters are updated. This renders the approach slow as the batch contains all training examples. To overcome this *stochastic gradient descent* (SGD) can be applied instead. The primary difference is that SGD executes the parameter update after each

training example in the training set, i.e. a batch size of 1. Both batch gradient and stochastic gradient descent are executed over a number of iterations, referred to as *epochs*. One epoch means that all of the training examples have completed one forward and backward pass (discussed in subsection 2.5.4). Another variation is *mini-batch gradient descent*. Here a random sample of size equal to a user-specified batch size is drawn from the training set. The random sample is used to compute the gradient step and update the parameters.

Alternative approaches

Modifications to gradient descent have been proposed over the years to enhance the performance in various ways. This subsection and the following one discusses some of these alternative approaches.

Gradient descent with *momentum* [206] takes analogy from Newton's law of motion in which it introduces a velocity term which enables gradient decent to take faster gradient steps towards the minimum. The velocity "is the direction and speed at which the parameters move through parameter space" [85]. Figure 2.18 illustrates the contour lines for some cost function for which we are trying to optimise the values of θ_1 and θ_2. The contour lines represent the values of the cost function at the various points of θ_1 and θ_2. The solid blue lines represent a path that gradient descent may take. There are a lot of oscillations which slow down the convergence towards the minimum (denoted as the center circle). Momentum attempts to smooth the steps of gradient descent by taking larger steps in the horizontal direction and smaller steps in the vertical. A path that momentum could take is illustrated in the dotted red line. In real-world applications of neural networks, there are more than just two parameters and thus the figure only serves as an example to illustrate the idea behind the approach. Additional variations include *RMSProp* [99] and *Adam* [123]. These share similar ideas of speeding up convergence with different subtleties to the algorithm to achieve this.

Backpropagation

Optimisers such as SGD or Adam make use of the gradient of the cost function with respect to each of the parameters of the network. However, in deep neural networks, computing these gradients can be computationally expensive especially since a lot of the terms in the calculation are duplicated. *Backpropagation* [223] provides a method for efficiently computing the gradients

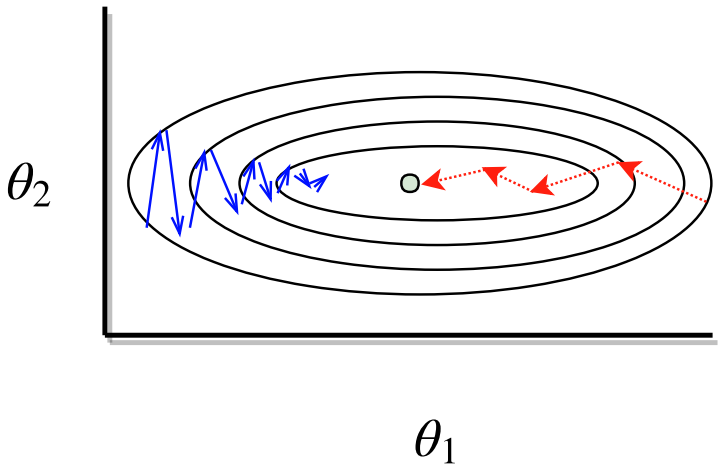

FIGURE 2.18: Illustrating the oscillations that take place during gradient descent in the solid blue line and the faster convergence which is achieved by momentum in the dotted red line. Two parameters θ_1 and θ_2 are shown. The contour lines represent the values of some cost function, $J(\theta_1, \theta_2)$, at the various settings of θ_1 and θ_2.

of the objective function with respect to each of the weights in the network. All of these gradients are required to update the weights with respect to the error the network makes.

Computing the gradient of the objective function with respect to layers close to the input require the calculation of derivatives of units deeper in the network (by using the chain rule). Consider the neural network in figure 2.19.

Now, to use an optimiser (such as gradient descent) we will need to compute the gradients. Let's assume that we wish to compute the gradients $\frac{\partial o}{\partial x_1}$ and $\frac{\partial o}{\partial z_1}$. Note that in the function $o = o(h_1, h_2)$, o is a function of h_1 and h_2. Also, h_1 and h_2 are functions of z_1 and z_2, and z_1 and z_2 are functions of x_1 and x_2. Using the chain rule, the gradients can be computed. First compute $\frac{\partial o}{\partial h_1}$ and $\frac{\partial o}{\partial h_2}$. Next, we can compute $\frac{\partial o}{\partial z_1}$ as shown in equation 2.12, the equation has two terms since o is a function of h_1 and h_2 which in turn are both functions of z_1.

The next derivative to compute is $\frac{\partial z_1}{\partial x_1}$ which is shown in equation 2.13. Notice in that equation that the first two terms have a repeated derivative $\frac{\partial z_1}{\partial x_1}$. We can take the derivative out, and place the two terms between brackets as shown in equation 2.14. Now notice that the term between the brackets has been computed before, in equation 2.12. So we can rewrite $\frac{\partial o}{\partial x_1}$ using equation 2.15. The equation can be further simplified by replacing the repeated expression in the second and third term in equation 2.15. Following this logic, backpropagation provides an efficient manner to avoid repeated calculations by reusing partial derivatives which were previous computed

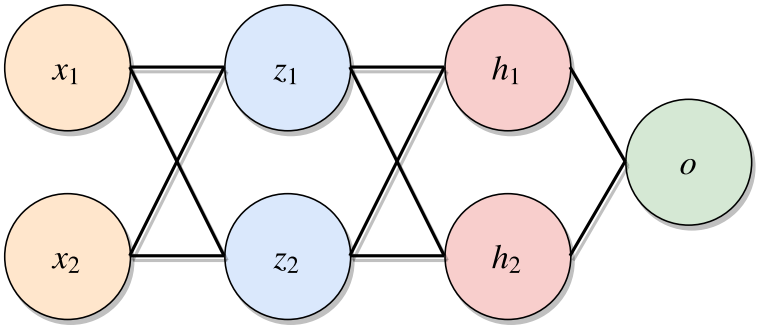

FIGURE 2.19: Illustrating a neural network with 3 layers. Backpropagation avoids the repeated computations of derivatives.

[85]. All of the partial derivatives are required for the weight update and backpropagation provides a recursive methodology.

$$\frac{\partial o}{\partial z_1} = \frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial z_1} + \frac{\partial o}{\partial h_2}\frac{\partial h_2}{\partial z_1} \tag{2.12}$$

$$\frac{\partial o}{\partial x_1} = \frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z1}{\partial x_1} + \frac{\partial o}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z1}{\partial x_1} + \frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial o}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_1} \tag{2.13}$$

$$\frac{\partial o}{\partial x_1} = (\frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial z_1} + \frac{\partial o}{\partial h_2}\frac{\partial h_2}{\partial z_1})\frac{\partial z_1}{\partial x_1} + \frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial o}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_1} \tag{2.14}$$

$$\frac{\partial o}{\partial x_1} = \frac{\partial o}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial o}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial o}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_1} \tag{2.15}$$

2.5.5 Deep neural network layers

Fully connected layers

Fully connected layers have a weight connection between each unit in layer *i* with each unit in the previous layer *i-1*. The dot product between the weights in layer *i* and the output from the units in layer $i-1$ are computed, and then typically an activation is applied. Figure 2.20 illustrates an example of a fully connected layer.

Convolutional layers

Convolutional layers [139] apply a convolution operation to extract features from an image (in the case of 2 dimensional convolution). The convolution

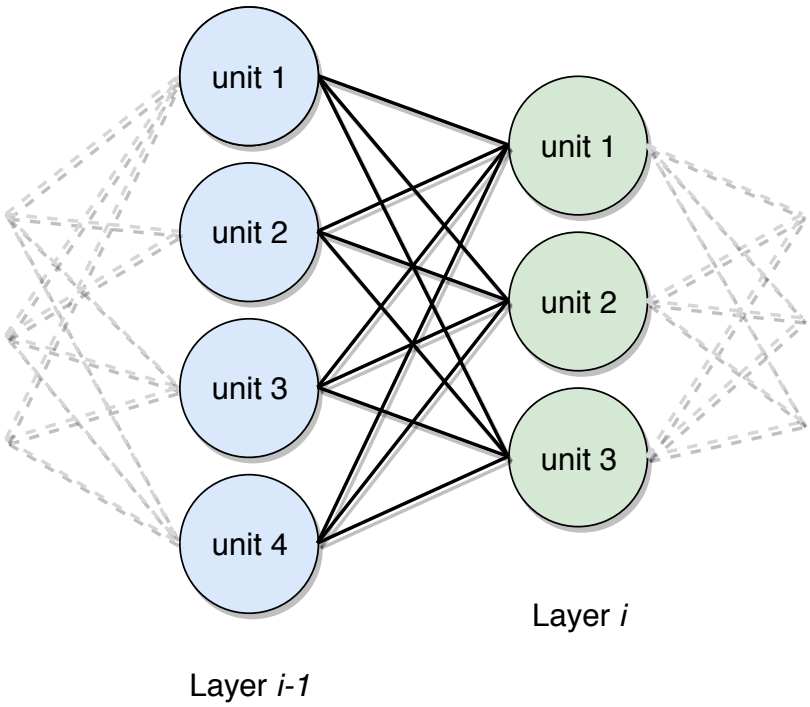

FIGURE 2.20: Illustrating a fully connected layer. Here each unit in layer i is connected to every other unit in layer *i-1*. There are 12 parameters since each of the four units connect to each of the other three units. The dotted lines denote connections to other layers.

operation is applied to two real valued functions and in turn returns another function, let us denote this as $F = x * w$. In this example the convolution is applied to functions x and w, and the resulting function is F.

In the case of convolutional layers in the context of deep learning, the function x is typically an input image, and w is referred to as a *filter* or kernel. A filter has a width and a height. The depth of a filter is the same as the depth of the input image. A colour image is represented by three colour channels, namely red, green and blue channels – thus the depth of a colour image is 3. The resulting function, F, is referred to as a *feature map* or activation map. For discrete convolutions, the filters convolve (or slide) around the input image horizontally and vertically. Given an image I and a filter K, each pixel (i, j) in the feature map, F, is computed using the expression in equation 2.16. This computation is executed each time the filter is moved, and this is the convolution operator, $*$, in the expression $F = x * w$. During the sliding of the filter, the expression in equation 2.16 is computed.

Figure 2.21 illustrates an example of applying the convolution operator to an input image. In this example the input image is 5x5 and the filter is of size 3x3. The bottom part of the figure illustrates how the value of 3 in position (1,1) in the feature map is computed. The filter is placed on the image and

the dot product of the corresponding values in the image is computed. The filter then slides one by one pixel to the right on the input image and the dot-product is computed again to get a value of 0. Once the filter can no longer move to the right it goes down by one pixel and repeats the process. This is repeated until all of the values in the feature map are computed. Each value in the filter is a weight, for which the neural network must optimise.

$$F(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.16}$$

The feature map has its own width, height and depth. The depth of the feature map is determined by the number of filters (a user-defined parameter). Thus, if an experimenter wants to create 10 filters, then the depth of the feature map is 10. The features are stacked on the depth dimension. The width and the height of the feature are affected by the *stride* that the filter uses. The stride denotes the number of pixels for which the filter is moved horizontally and vertically. A large stride will result in a smaller feature map (in terms of the width and the height). Typically, a stride value of 1 or 2 is used.

Figure 2.22 illustrates an example of an input image, a filter and the resulting feature map when the convolutional operator is applied. In the figure there is only one filter so the depth of the feature is one. The stride is set to one. The figure illustrates the new size of the feature. An input image of size 32x32 for which a filter of size 5x5 is applied to will result in a feature of size 28x28 (784 pixels). For each of those 784 pixels, the dot product of the filter with the corresponding region of the image is computed.

Convolutional layers can be one or two dimensional. One dimensional convolutional layers can be applied to natural language processing. In such a case the text is often converted into a sequence of numbers and thus the filter is represented as a vector. All of the remaining details are the same.

Pooling layers

Pooling layers are typically applied after convolutional layers. Their purpose is to reduce the spatial size of the previous layer. As a consequence, the number of parameters are reduced. The pooling layer is applied separately to each depth of the previous layer. The pooling layer takes on its own width, height and stride, and is applied to the previous layer in a similar way as the filter in the convolutional layer. The operation that is executed is however different. Pooling typically provides a summary statistic of the neighbouring

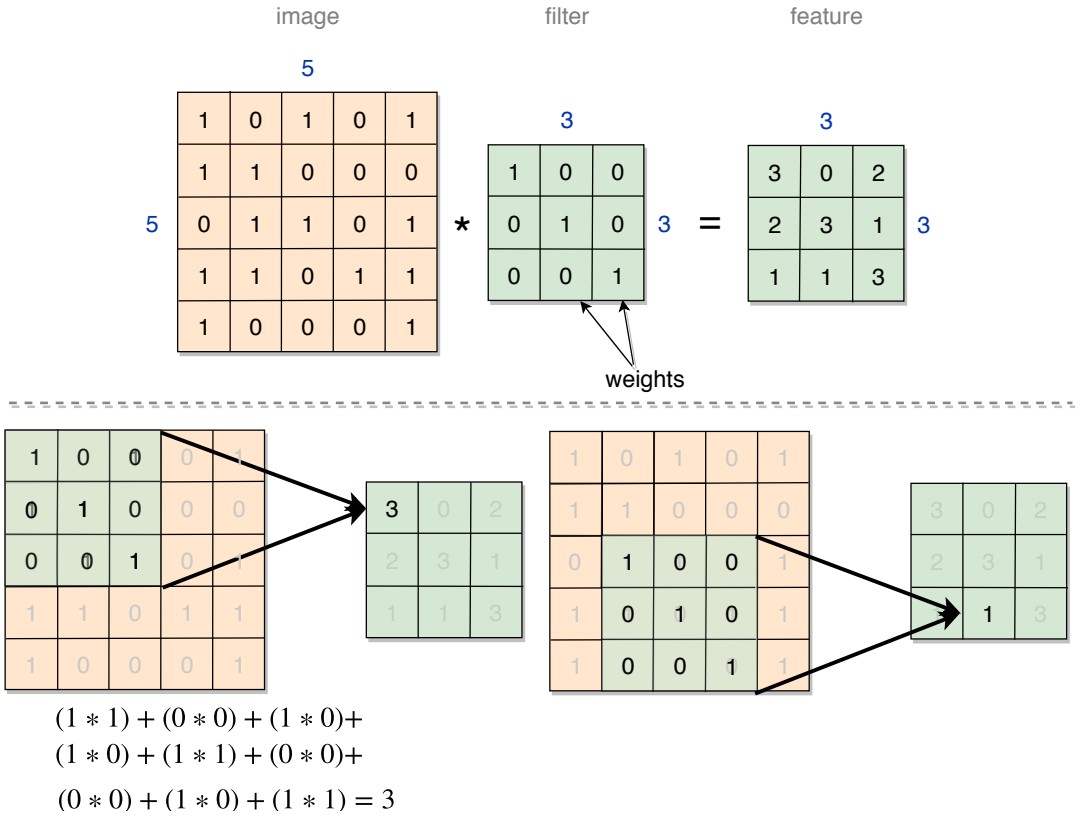

FIGURE 2.21: Illustrating, on top, an image, a filter and the resulting feature. The bottom half of the figure illustrates how the values in the feature are obtained. The sum of the element-wise multiplication between the filter and the corresponding region of the image are computed. In the left example, the computed value is 3. The filter is then shifted by one pixel, and the computation is repeated. This is performed until every value in the feature map is computed. The figure illustrates a second example where the value of 1 is computed in the feature map.

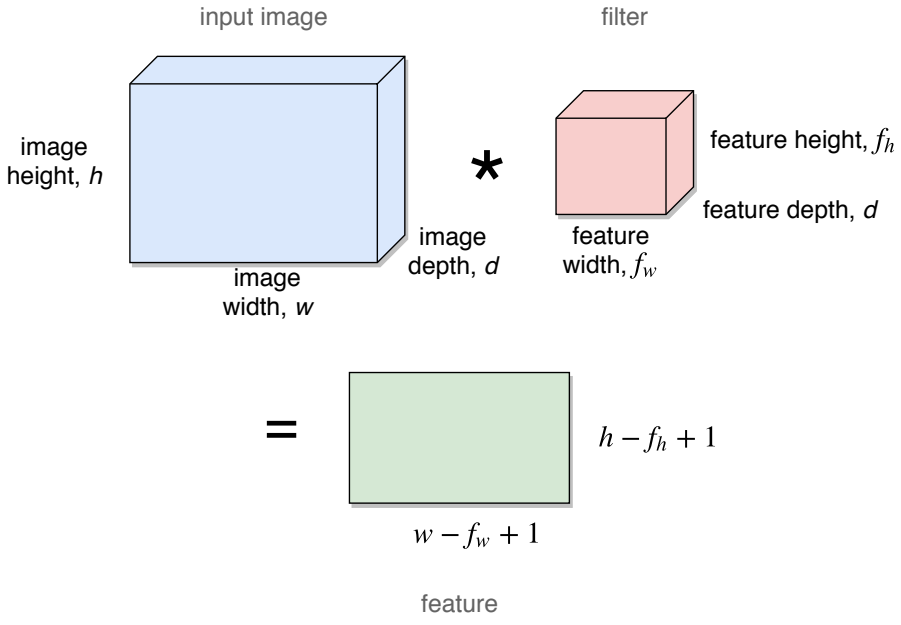

FIGURE 2.22: Illustrating the width and the height of the input image, the filter and the feature map.

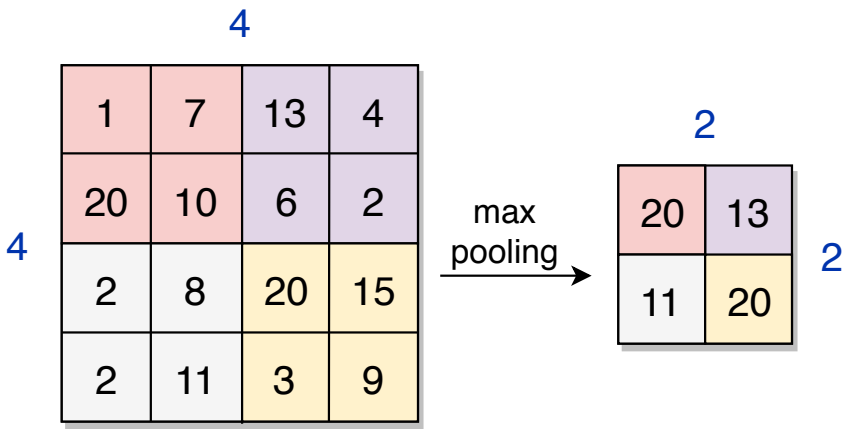

FIGURE 2.23: Illustrating the spatial reduction achieved by performing max pooling. Here the width and the height of the filter is 2x2, and the stride is set to 2. Max pooling computes the maximum value in each of the four regions.

pixels. *Max pooling* [303] is a commonly used layer, typically with a filter width and height of 2x2. Here the maximum value is returned. Figure 2.23 illustrates an example of the resulting feature from the max pooling layer. The max pooling layer has no parameters which are optimised during the optimisation of the neural network. Average pooling is an alternative to max pooling.

Dropout

Dropout is a regularisation technique which assists in preventing over-fitting when training a neural network [243]. A keep probably p is typically associated with dropout which is a user-defined hyper-parameter. During each epoch, each unit within the network is ignored with a probabilty of p. These ignored units are not taken into consideration when the network performs a forward or backward pass, and thus enables the network to optimise the weights despite the reduced number of units.

2.5.6 Software

Over the years, various software packages have been developed to enable researchers and practitioners to implement neural networks. While it is beyond the scope of this thesis to evaluate each of the available software, we present a list of commonly used ones.

- Caffe [111], primarly for C++ users
- DeepLearning4J [257], primarily for Java users

- Keras [35], primarily for Python and R users
- PyTorch [199], primarily for Python users
- Tensorflow [1], primarily for Python and C users

2.5.7 Network Architecture

Another consideration when implementing a neural network is the design of the *architecture* [85]. The architecture refers to the number of layers in a network, the number of units in each layer, and how these layers are connected to each other. Since each layer has its own hyper-parameters, in this thesis, the term architecture refers not only to the structure of the network but also to all of the associated hyper-parameters. Typically, the layers are connected to each other in a chain like manner, layer *i* is connected to layer *i+1* and so on. Goodfellow *et al.* [85] denotes the first layer as $\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)T}\mathbf{x} + b^{(1)})$ and the second layer is defined as $\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)T}\mathbf{h}^{(1)} + b^{(2)})$, where $\mathbf{W}$ are the weights, $\mathbf{x}$ is the input, b is the bias and g is the activation function. The second layer makes use of the output from the first layer. Goodfellow *et al.* state that the validation error is used to guide an experimenter in finding the most suitable network and that this is achieved through experimentation [85].

2.6 Conclusion

This chapter provided an introduction to machine learning. The reader was presented with various concepts and terminology related to machine learning. The concept of classification and regression was introduced. Methods to split the data into training and validation sets were described and furthermore, approaches for evaluating the performance of a machine learning algorithm was discussed. Next, the reader was presented with an introduction to evolutionary algorithms. Despite the fact that such algorithms are computationally expensive and take long to run, their stochastic nature allows for a variety of solutions to be evolved as opposed to a method which produces identical solutions. Finally, we introduced deep learning. The perceptron was introduced and then more complicated multi-layer networks were discussed. Various activation functions were presented. The chapter then discussed various types of layers. In addition to the layers discussed, there are other ones which we have not explored, such as long short-term memory

units [101] widely used as layers in recurrent neural networks. We thus limited the discussion to concepts which are used throughout this thesis. The following chapters describe how evolutionary algorithms can be adapted to solve a variety of machine learning problems.

Chapter 3

Neuro-evolutionary Problem Identification

3.1 Introduction

In this chapter, a GA harnessed to a dynamic and flexible deep learning framework is proposed for the automated identification of problems. We call this the *Automated Problem Identification* (API) algorithm and show that it can successfully determine if a dataset is a classification or a regression one; and furthermore, recommend whether to use the categorical cross entropy or mean-squared error loss function. To achieve this we use neuro-evolution. *API* will recommend which layers (e.g. convolutional or fully connected) – from a known set – to use, either as the final architecture or as the input to further optimisation.

Section 3.2 provides the reader with a more detailed formulation of problem identification and discusses certain related work. Section 3.3 discusses the proposed approach and this is followed by details on the proposed algorithm in section 3.4. The experimental setup is presented in section 3.5 and the findings are revealed in section 3.6. Finally, section 3.7 concludes this chapter.

3.2 Problem Identification

Regression or classification? This is perhaps the most basic question faced when tackling a new supervised learning problem. Typically, a significant amount of human insight and preparation is required prior to executing machine learning algorithms. For example, when creating deep neural networks, the number of parameters must be selected in advance and furthermore, a lot of these choices are made based upon pre-existing knowledge of

the data such as the use of a categorical cross entropy loss function. Humans are able to study a dataset and decide whether it represents a classification or a regression problem, and consequently make decisions which will be applied to the execution of the neural network.

As the performance of machine learning algorithms has skyrocketed over recent years the often unspoken relationship between the human data scientist and the machines they run has evolved significantly. A great deal of work has been put into new state-of-the-art methods, and researchers are constantly optimising the various aspects of machine learning algorithms. Such efforts include proposing algorithms for optimising hyper-parameters and network architectures [217] and the latest trends show increasing emphasis on algorithms that require less human intervention. Consider the automatic statistician project [1] which aims at removing the data scientist from the process of understanding data by using Bayesian model selection.

Real *et al*. [217] propose an evolutionary algorithm for optimising image classification neural networks which requires no human intervention in creating the networks. Similarly, Zoph and Le [305] use recurrent neural networks along with reinforcement learning in order to achieve a similar goal. It is clear from these research efforts that this is a trend that will continue, driven both by potential industrial profits to compensate for shortages of expensive data scientists and by the general goal of artificial general intelligence.

Nevertheless, for most current machine learning algorithms, there is a considerable amount of human intervention which must be performed prior to the final execution of the algorithm. For example, setting the number of parameters, pre-processing the data, deciding on the loss function and interpreting the results, to name a few. Another example, and perhaps the first of the steps in the data science process, is problem identification: *"does a supervised set of data correspond to a classification or regression problem?"* Understanding which type of the two problems a given dataset represents is a step in the direction of automated machine learning research and is the subject of this chapter.

Classification problems typically represent a set of problems whereby the goal is to create a predictive model that can discriminate between various known classes. CIFAR-10 and MNIST are examples of classification datasets where the goals are to identify the correct label (airplane, automobile, bird, cat etc... and digits respectively) for each image. For regression problems,

[1] https://www.automaticstatistician.com/index/

the predictive output is continuous (as opposed to discrete in the case of classification). An example of a regression dataset is the Boston housing price regression dataset for which the goal is to predict the median value of the houses.

In the context of deep learning [138], when presented with a dataset, typically one will verify whether the data represents a classification or a regression problem, and then will decide on the loss function and network layers accordingly. For the CIFAR-10 image dataset, one might consider using convolutional, dropout and fully connected layers (based on the layers used in the VGG network [237]); and for the Boston housing price dataset one might use fully connected and dropout layers. Furthermore, a decision should be made with regards to which loss function to use. For CIFAR-10 one might use categorical cross entropy (see 3.3.2), and use the mean squared error loss function for the Boston housing case. As researchers in machine learning, in most cases, these decisions can be made with relative ease. For a machine, on the other hand, this decision is non-trivial and current machine learning algorithms do not automatically decide if a given dataset is a classification or a regression problem; nor do they recommend a loss function.

An application of automatic problem identification is one that allows an algorithm to perform predictive modelling in an environment where various dataset types are supplied and little human intervention is provided.

3.3 Proposed *API* Approach

In the following subsections we define our proposed approach. We provide details on how the proposed approach is adapted in terms of the GA details. Figure 3.1 illustrates an example of a network which was produced by a chromosome when the CIFAR-10 dataset was input into *API*. The resulting architecture is very similar to one that a human might use for the problem.

3.3.1 *API* chromosome

In this section and the following subsections, we describe the *API* chromosome along with a description about each of the genes within the chromosome. In this chapter, the word layer refers to the layers in deep neural network architectures. Each chromosome is made up of four genes, namely, the neural network loss function, the number of units in the last layer of the neural network, the activation function used in the last layer and the

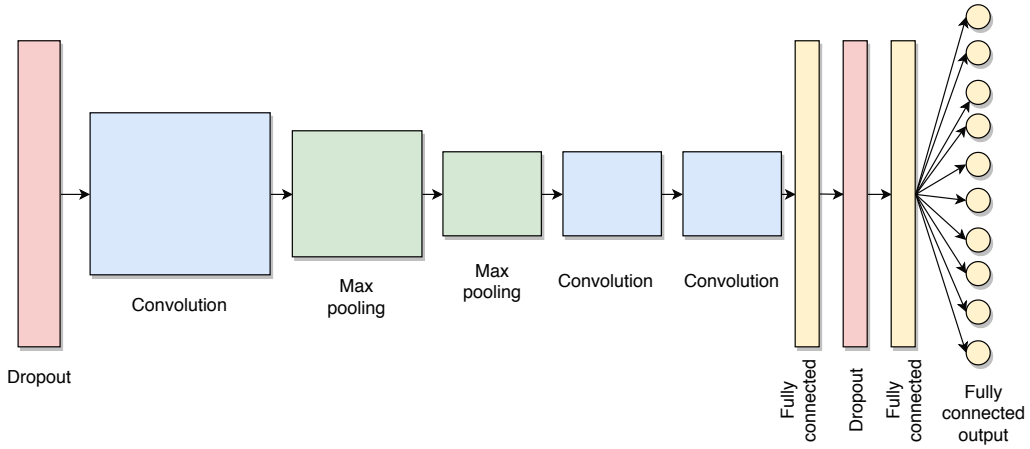

FIGURE 3.1: Each chromosome contains four genes of which one gene represents a network architecture. The figure illustrates an example of a network architecture generated by an *API* chromosome (which was obtained at the end of an execution of the *API* algorithm). The input dataset was CIFAR-10 – an image classification dataset. The chromosome, presented in figure 3.2, recommended that the last layer should have 10 units and that these should use the softmax activation function. Furthermore, the chromosome recommended using the categorical cross entropy loss function, and consequently, correctly determined that the dataset was a classification problem.

configuration of the layers (configurations are explained in section 3.3.5). A chromosome thus encodes a neural network architecture along with the loss function.

Figure 3.2 illustrates an example of an *API* chromosome. In the example, the chromosome encodes the following neural network architecture, a fully connected layer, followed by dropout and two fully connected layers. Furthermore, the chromosome will apply the categorical cross entropy loss function (during the training of the neural network) and the last layer has 10 units of which the activation function is softmax. The following subsections provide additional details about the four genes.

Categorical cross entropy	10	Softmax	[2, 0, 3, 3, 0, 0, 1, 2, 1, 1]

FIGURE 3.2: Example of an *API* chromosome which encodes the categorical cross entropy error loss function, 10 units in the last layer of the network which has the softmax activation function. The architecture of the network is denoted as [2, 0, 3, 3, 0, 0, 1, 2, 1, 1] for which a convolutional layer is mapped to 0, a fully connected layer to 1, dropout to 2 and max pooling to 3.

We chose to use GAs since the number of genes can easily be modified in

order to encode additional complexity. GAs have previously been successful in optimising complex search spaces. It was deemed appropriate to use GAs since *API* searches through a space of network architectures in addition to other parameters. We can thus increase the complexity of the chromosomes by including more parameters. We explore this in the following chapter.

3.3.2 Loss function

This gene represents the loss function that will be used when training the network and it takes on two possible values: Mean Squared Error (MSE) and Categorical Cross Entropy (CCE) loss. Let y_i denote the target label for sample i, $\bar{y}_i$ denote the model's predicted output for sample i and n denote the number of training samples. The mean-squared error used in this study is presented in equation 3.1.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2 \tag{3.1}$$

For example, let $y = [2.2\ 5.5\ 0.2]$ and assume that some network N_1 predicts $\bar{y} = [2.1\ 5.0\ 0.2]$ then $MSE_{N_1} \approx 0.09$. Similarly, assume that another network N_2 predicts $\bar{y} = [0.0\ 2.5\ 3.2]$ then $MSE_{N_2} \approx 7.61$. Network N_1 is preferred since $0.09 < 7.61$ (a smaller value is better). When using the MSE, the objective of an optimisation algorithm will be to minimise the MSE value to reduce the distance between the correct values and the model's predicted values. The CCE used in this study is presented in equation 3.2.

$$CCE = -\sum_{i=1}^{N} y_i \ln \bar{y}_i \tag{3.2}$$

When using this loss function the objective is to minimise the CCE in such a way to make the network predictions as similar to the labels as possible. In this case, the target labels will be represented as a vector (often one-hot encoded vectors, discussed in section 3.3.6) and the network predictions will also be in a vector of the same length. For example, let $y_i = [0\ 1\ 0]$ and assume that some network N_1 predicts $\bar{y}_i = [0.1\ 0.8\ 0.1]$ then CCE for sample i is $-\ln(0.8) \approx 0.22$. Similarly, assume that another network N_2 predicts $\bar{y}_i = [0.7\ 0.1\ 0.2]$ then for sample i, CCE $= -\ln(0.1) \approx 2.30$. Network N_1 is preferred since $0.22 < 2.30$ (a smaller value is better).

3.3.3 Number of units in last layer

The second gene denotes the number of units in the last layer in the network. There are two possible values for this gene: one or U, where U denotes the number of unique values in Y (Y represents the target values for a dataset). Formally, $U = |S|$, where $S = \{y_i\}_{i \in \{1,\ldots,N\}}$ and N is the number of samples. For example, assume that for some dataset $Y = (0, 1, 0, 2, 3, 2, 3, 0, 2, 3, 1, 1)$ then $U = 4$ since there are 4 unique values in the targets. In the case of classification or regression it would be incorrect to have values other than one or U in the last layer, for example using 5 units in a 7 class classification problem.

3.3.4 Last layer function

This gene takes on four possible values and denotes which activation function will be used in the last layer in the network. The possible values are: {linear, relu, sigmoid, softmax}. Here 'relu' refers to rectified linear units. Given some input x to a layer, the equations for each of the activation functions are presented in equations 3.3 to 3.6 respectively.

$$f(x) = x \tag{3.3}$$

$$f(x) = \max(x, 0) \tag{3.4}$$

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{3.5}$$

$$f(x) = \frac{e^x}{\sum_{i=1}^{D} e^{x_i}} \tag{3.6}$$

where

$D =$ dimension of x

3.3.5 Configuration of layers

Each chromosome has a gene which corresponds to the architecture of the network which we define as the configuration. The configuration represents the exact sequence of the network layers and is stored in a list. The first

element in the configuration represents the first layer and the last element represents the last layer. There are four possible values which each element in the configuration can take, namely: convolution, fully connected, dropout [243] and max pooling. Here, convolution refers to two-dimensional convolution. We add dropout to the list of possible configuration values even though dropout is not a layer.

The size of the configurations is randomly selected between 5 and 15. The configurations are initialised randomly during the initial population generation and modified during the mutation operator; these are explained in sections 3.4.1 and 3.4.2 respectively. Each of the layers are mapped to an integer value, i.e. convolution is mapped to 0, fully connected to 1, dropout to 2 and max pooling to 3. Each chromosome has exactly one configuration.

We provide the following example to illustrate the configurations. Let the configuration for a chromosome be: [2, 0, 3, 3, 0, 0, 1, 2, 1, 1]; figure 3.1 illustrates this network. The network is comprised of several convolution and max pooling layers followed by fully connected and dropout layers.

3.3.6 Chromosome fitness evaluation

In this study, we designed a fitness function to discriminate between classification and regression problems. When the proposed system commences, it splits the dataset into two subsets, the features, X and the labels Y. The labels are then converted into their corresponding one-hot encoded values – a process for encoding single categorical variables into a vector. The length of the vectors corresponds to the number of unique values. For example, a classification problem with 3 class values will result in one-hot encoded vectors of length 3. For example, if a label has a value of 0 and the unique Y values are {0, 1, 2}, then the one-hot encoded value of '0' is [1 0 0], '1' is [0 1 0] and '2' is [0 0 1]. The system retains both Y and the one-hot encoded Y values. The dataset is split such that 50% of the data is in the training set and the remaining in the validation set.

Each chromosome is evaluated as follows. The chromosome's loss function is used to train the neural network on the training data. If the chromosome's loss function was categorical cross entropy, then the one-hot encoded Y values are used during training. However, if the loss function was mean squared error, then the Y values are used during training.

The validation loss is recorded during the optimisation of the neural network across the epochs. Let the validation loss be $V_0, V_1, ..., V_e$ where e denotes the total number of epochs and V_i denote the validation loss for epoch i. We define the change in validation as follows, $\delta_{val} = average(V_1, V_2, ..., V_e) - V_0$. Finally, we define the ratio in validation drop, R, as $R = \frac{\delta_{val}}{V_0}$.

For each chromosome, after the optimisation of the neural network has taken place, we compute R, and if $R > 0$ then this implies that the network has not done any learning since the validation loss increased. Furthermore, if $R = 0$ then once again the network has not managed to learn anything since the validation loss has remained constant over the epochs. Finally, if $R < 0$ we conclude that given the drop in validation loss, that the network has managed to learn.

The model then predicts the output values on the validation data. The predictions and the validation target values are compared using mean squared error. We chose to use the mean squared error to be consistent with the comparisons. In the case whereby the network has not learnt anything we penalise the chromosome with a fitness of infinity. However, in the case whereby the network has learnt, i.e. $R < 0$, then we assign the computed validation mean squared error as the fitness of the chromosome. The objective of the *API* algorithm is thus to minimise the fitness of each chromosome by rewarding networks that learn and have a small mean squared error on the validation set. The lower the fitness value the better a chromosome performed.

Figure 3.3 illustrates a plot which explains the fitness of a chromosome. The plot is separated in two where $R = 0$. From the plot, it is observed that when $R \geq 0$ then the fitness is set to a very large value. When $R < 0$ then the value of the fitness corresponds to the mean squared error whereby a smaller value is better. For the sake of the example, a straight line was drawn for $R < 0$ to illustrate that a smaller mean squared error results in a better chromosome fitness.

Since the chromosomes are randomly generated, it is possible that they represent invalid networks for particular features and labels on a given dataset. For example, assume that, for some chromosome, the number of units in the last layer is 1 and the categorical cross entropy loss function is used. Given the previous description in this subsection, the one-hot encoded Y values should be used during training. However, in the example, the number of outputs is 1 and thus a one-hot encoded vector cannot be compared to a single value. To illustrate with another example, consider a chromosome

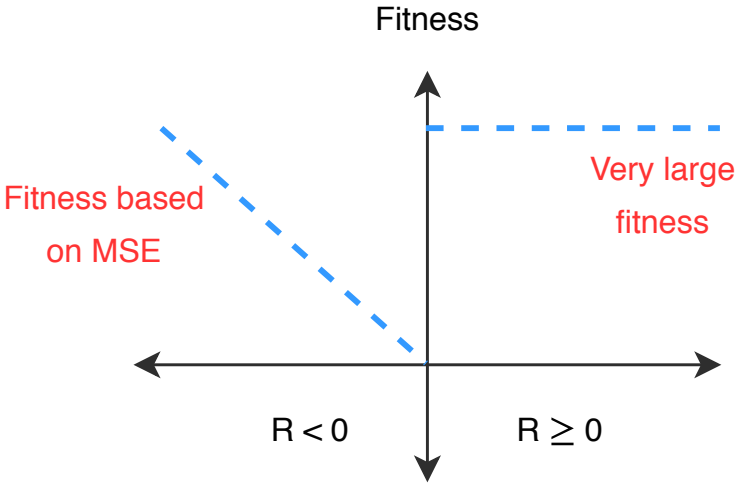

FIGURE 3.3: During the training of each neural network on the validation data we record the validation loss. We then determine whether or not the network has learnt. If the network has not learnt ($R \geq 0$) then we penalise the chromosome with a very large fitness. However, if the network was able to learn ($R < 0$) then we assign the mean squared error as the fitness value.

that tries to use convolutional layers on a feature based regression dataset – this is of course not feasible. Invalid chromosomes such as this are penalised with a fitness of infinity. Section 3.4.3 describes how the chromosome makes the discrimination between regression and classification.

3.4 The *API* Algorithm

The following subsections explain how each aspect of the GA has been altered to determine if a given dataset is a classification or regression problem. Furthermore, the algorithm recommends the following upon termination: the loss function which should be used in order to enable the training of a neural network, the number of units and type of activation function in the last layer and finally, a simple network architecture is also recommended. The *API* algorithm performs optimisation in two phases, namely in optimising the GA population, and since each chromosome represents a neural network, optimisation is performed when training the networks.

3.4.1 Initial population generation

Each chromosome has a fixed length of 4 genes (discussed in section 3.3.1). During the creation of a chromosome, each gene is randomly created based

on the values each gene can take on. The pseudocode for creating a chromosome is presented in algorithm 4. The initial fitness of each chromosome is set to infinity.

Algorithm 4: Creating a chromosome.

```
1 begin
2     Initialise an empty chromosome.
3     Set the loss function to either categorical cross entropy or mean
        squared error.
4     Set the number of units in the last layer to either one or U.
5     Set the activation function in the last layer to either linear,
        sigmoid, softmax or relu.
6     Create a random configuration.
```

3.4.2 Genetic operators

Mutation and crossover were implemented in this study. Their implementation details for this study are described below.

Mutation

The mutation operator randomly selects a gene and alters the value within the gene with a new random value. The values assocaited with each gene are discussed in sections 3.3.2 to 3.3.5. The operator changes a single gene upon execution.

Figure 3.4 illustrates the application of the mutation operator on a parent chromosome, and the resulting offspring is illustrated. From the example, the fourth gene was selected for mutation which implies that the configuration of the network is regenerated. In the figure, the parent chromosome configuration was changed from [1, 2, 1, 1] to [1, 1, 1, 1, 1] in the offspring.

Crossover

The crossover method we implement randomly selects a position p in the range $[0, n]$ — where n denotes the length of the chromosome — within the parent chromosomes; the same position p must be selected within the two parents. Two offspring are created, and all the genes except those at position p are copied across to the corresponding offspring without modification. The genes in position p are swapped, i.e., the gene in position p from $parent_1$ is

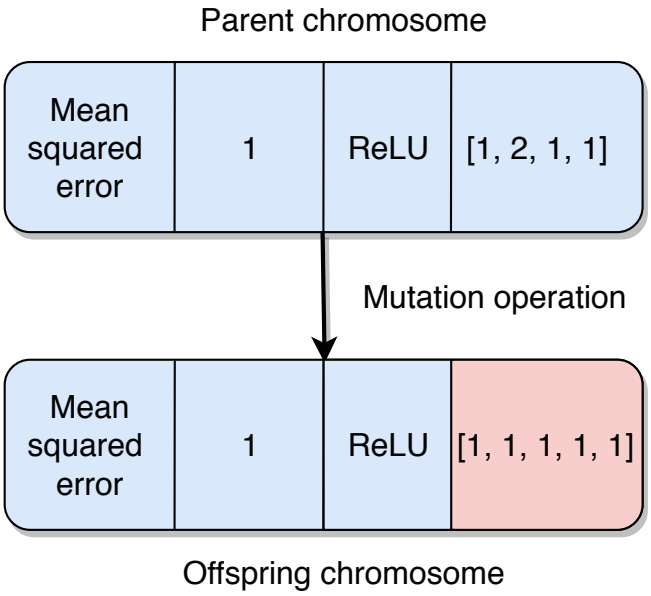

FIGURE 3.4: Example of the mutation operator being applied to a parent chromosome. The fourth gene was selected for mutation and consequently a new configuration was generated for the offspring.

inserted into position p in $offspring_2$, and similarly, the gene in position p from $parent_2$ is inserted into position p in $offspring_1$.

An example of the application of the crossover operator is presented in figure 3.5. The figure shows two parent chromosomes. The crossover point was the third gene from each parent, i.e. the last activation function was swapped between the parents. The offspring show the result of the crossover.

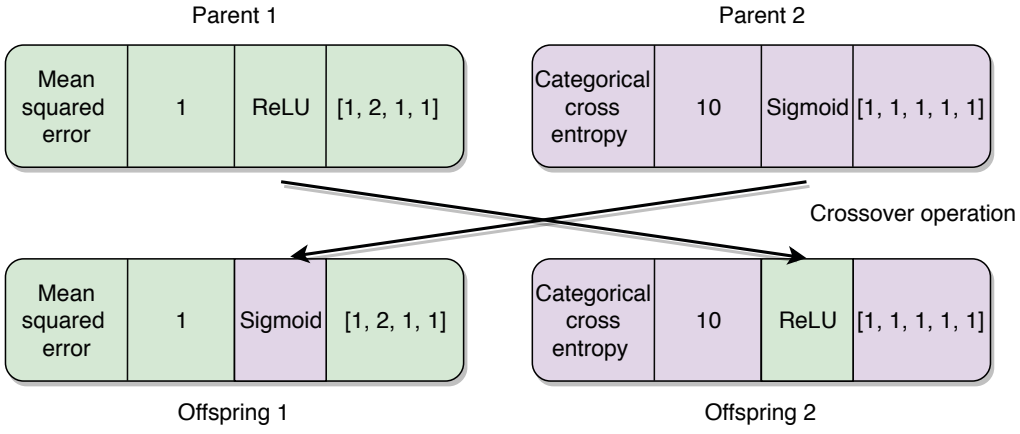

FIGURE 3.5: Example of the crossover operator being applied two parent chromosomes. The third gene from both of the parents were selected for crossover. As a result, the last activation functions were swapped between the parents.

3.4.3 Algorithm termination and final decision

At the end of the generational loop, the best chromosome is output. The loss function in the best chromosome is then used to decide if the dataset was a classification or a regression problem. If the loss function was categorical cross entropy, then the problem was labelled as classification. However, if the loss function was mean squared error then the problem was labelled as regression.

3.5 Experimental Setup

This section describes the experimental set up which was used to evaluate the performance of API. The algorithm was programmed in Python 3.6.0 and TensorFlow 1.1.0 [1]. We made use of certain functions provided by TensorFlow – such as *tf.nn.conv2d* [2] and *tf.nn.dropout* [3]– to interface the neural networks found in each chromosomes with TensorFlow. *API* was evaluated on a machine with an Intel Core i7-6700K CPU and 16GB RAM.

3.5.1 Datasets

Table 3.1 presents the 16 datasets which were used in this study along with their characteristics and type. All of the datasets were obtained from the UCI machine learning repository [147] except for CIFAR-10 and CIFAR-100 which were obtained from [129], MNIST from [137], CrowdFlower [4], Aloi [5] and IMDB [6] were obtained externally. In this study, CrowdFlower represents the 'emotion in text' dataset. The assumptions are that there are no missing values in each dataset and that categorical features are converted to corresponding numerical features using a one-hot encoding approach. Of course, it would be possible to implement an imputation method [169] to overcome datasets with missing values, however, this was not part of the scope of this study. The algorithm standardises each feature. Where possible, we used 1000 samples for training and 1000 for validation. Boston housing, for example, did not have that many samples. In this case, we simply equally split

[2]https://www.tensorflow.org/api_docs/python/tf/nn/conv2d
[3]https://www.tensorflow.org/api_docs/python/tf/nn/dropout
[4]https://www.crowdflower.com/data-for-everyone/
[5]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/multiclass.html
[6]https://keras.io/datasets/

TABLE 3.1: The 16 datasets used in this study. We used datasets from four problem domains with various characteristics and are denoted as follows: 'D' represents data classification, 'R' for regression, 'IC' for image classification and 'SA' for sentiment analysis. The types 'D', 'IC' and 'SA' denote classification problems. The unique targets refers to the unique number of outputs in the target values for each dataset. For example, for CIFAR-10 has 10 unique target classes, whereas Relative CT Slice has 2001 unique target values. For CrowdFlower and IMDB we used a bags of words approach in order to generate word embeddings.

Dataset	Features	Unique Targets	Type
Aloi	128	1000	D
Isolet5	617	26	D
Letter Recognition	16	26	D
Sensorless Drive	48	11	D
Year Prediction	90	89	D
Boston Housing	13	506	R
CCPP	4	4837	R
Concrete Comp	15	1030	R
Forest Fires	29	17380	R
Physiocochemical	9	15903	R
Relative CT Slice	384	2001	R
CIFAR-10	3072	10	IC
CIFAR-100	3072	100	IC
MNIST	784	10	IC
CrowdFlower	1000	13	SA
IMDB	10000	2	SA

the dataset into two sets. We distinguish between data and image classification problems because in the former the data is typically resented by one-dimensional vectors; whereas, image classification datasets are commonly represented as three-dimensional arrays. *API* can adapt to the various input shapes without human intervention.

3.5.2 Experimental parameters

The GA and neural network parameters used in this study are presented in tables 3.2 and 3.3 respectively. These parameters were obtained by preliminary runs of the algorithm. The purpose of this study was to evolve chromosomes that could determine whether a given dataset was classification or regression in addition to several other outputs. Certain variables had to remain fixed in order to evolve the chromosomes. Each parameter in table 3.3 was set to a fixed value.

TABLE 3.2: The GA parameters used in this study. Preliminary experiments revealed that we did not need to use a large population size or a large number of generations to evolve accurate solutions.

Parameter	**Value**
Crossover rate	70%
Mutation rate	30%
Number of generations	10
Tournament size	5
Population size	50

3.6 Results and Discussion

The results obtained by *API* are presented and discussed in this section. The Aloi dataset was included in the experiments because one could hypothesise that if a dataset has a large number of targets then it is a regression dataset. For this reason, we included Aloi as it has a much larger number of classes in comparison to the other classification datasets. The accuracy results achieved by *API* on the 16 datasets across the 20 runs are presented in figure 3.6. On each run the entire API evolutionary process was executed. When discriminating between regression or classification problems, *API* obtained an average accuracy of 96.3%, the lowest accuracy was 90% which was obtained on 3 datasets and the highest accuracy was 100% which was achieved on 7 datasets.

Table 3.4 presents the number of times, out of 20 runs, that *API* incorrectly classified each dataset. There were 3 datasets for which *API* incorrectly classified two runs, this represents an accuracy of 90%. There was no dataset for which the performance across the runs was less than 90%.

In the case of the two misclassifications for the CIFAR-10 dataset, the fitness for chromosomes having the mean squared error and the categorical cross entropy loss function were very close. It happened to be that, for that particular run, the former had a slightly lower fitness. In the second case, the population was rapidly dominated by chromosomes having the mean squared error loss function as the generational loop progressed. A similar observation was made for the other incorrectly classified runs. Two possible ways of overcoming this issue would be to re-introduce genetic diversity into the population by randomly initialising a number of chromosomes across the generations. This would thus allow chromosomes containing both types of loss functions to be present in the population. Alternatively, increasing the tournament size could allow for weaker chromosomes to remain in the population which could in turn preserve the balance between the chromosomes

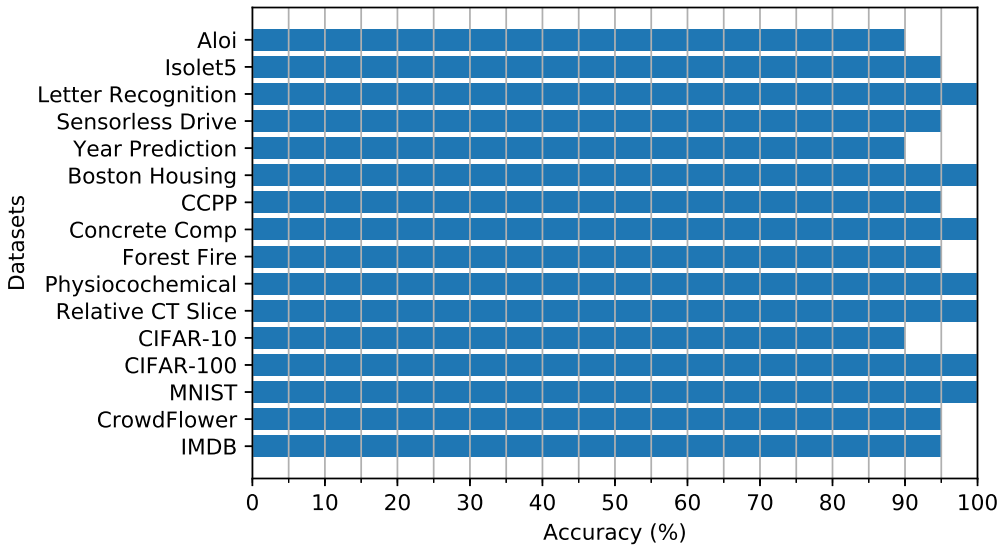

FIGURE 3.6: Accuracy (%) results obtained by *API* on the various datasets. The accuracy denotes *API*'s ability to correctly determine if each dataset was a classification or a regression problem. For each dataset, 20 runs of the algorithm were executed. The lowest accuracy was 90% and *API* achieved 100% accuracy on 7 datasets. The average accuracy across the datasets was 96.3%.

containing both loss functions.

Next, we present for each dataset a random chromosome that was evolved. These chromosomes were randomly selected from each of the 20 runs. The networks varied in size from 5 to 15 layers, however, in most cases the networks were deep. The architecture generated for the image classification problems are more complex than the ones generated for the other problems. In particular, the evolved chromosome for the CIFAR-10 dataset was of interest because the configuration resembles an architecture that a human might generated when creating a deep neural network for image classification. For instance, consider AlexNet [129], which is made up of a series of convolutional and max pooling layers towards the start of the network, and ends with three fully connected layers. In a similar way, the chromosome's architecture which is presented below has a similar structure of convolutions and max pooling layers followed by fully connected and dropout layers.

In the following list of results, the dataset name is provided along with the problem type. For the last activation function, 'MSE' denotes mean squared error, and 'CCE' denotes categorical cross entropy. For the configurations, convolution is mapped to 0, fully connected to 1, dropout to 2 and max pooling to 3. In each example the chromosome was able to correctly classify the dataset.

TABLE 3.3: The neural network parameters used in this study. When training a neural network contained in a chromosome each of the parameters listed in this table were applied.

Parameter	**Value**
Number of epochs	5
Weight initialisation - mean	0.0
Weight initialisation - standard deviation	0.01
Number of units in all layers except last	100
Activation functions in all layers except last	relu
Number of filters in each convolution layer	10
Convolution filter size	2x2
Convolution strides	1
Max pooling size	2x2
Max pooling stride	1
Dropout keep probability	0.8
Learning rate	0.001
Optimiser	Adam [123]
Batch size	2048

TABLE 3.4: The number of runs for which the algorithm incorrectly classified each dataset. The objective of *API* was to discriminate between regression and classification datasets. For each dataset we performed 20 *API* runs. A perfect accuracy of 100% was achieved on 7 datasets. For the types, 'D' represents data classification, 'R' for regression, 'IC' for image classification and 'SA' for sentiment analysis.

Dataset	**Type**	**Number of Incorrectly Classified Runs**
Aloi	D	2
Isolet5	D	1
Letter Recognition	D	0
Sensorless Drive	D	1
Year Prediction	D	2
Boston Housing	R	0
CCPP	R	1
Concrete Comp	R	0
Forest Fire	R	1
Physiocochemical	R	0
Relative CT Slice	R	0
CIFAR-10	IC	2
CIFAR-100	IC	0
MNIST	IC	0
CrowdFlower	SA	1
IMDB	SA	1

- **Dataset:** Aloi – Classification
  Chromosome: Units: 1000, Loss: CCE, Activation: linear, Configuration: [2, 1, 1, 2, 1]

- **Dataset:** Isolet5 – Classification
  Chromosome: Units: 1, Loss: MSE, Activation: softmax, Configuration: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

- **Dataset:** Letter Recognition – Classification
  Chromosome: Units: 26, Loss: CCE, Activation: sigmoid, Configuration: [1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1]

- **Dataset:** Sensorless Drive – Classification
  Chromosome: Units: 11, Loss: CCE, Activation: relu, Configuration: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

- **Dataset:** Year Prediction – Classification
  Chromosome: Units: 64, Loss: CCE, Activation: sotmax, Configuration: [1, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1]

- **Dataset:** Boston Housing – Regression
  Chromosome: Units: 1, Loss: MSE, Activation: softmax, Configuration: [2, 1, 1, 2, 1, 1, 1, 1, 2, 1]

- **Dataset:** CCPP – Regression
  Chromosome: Units: 1, Loss: MSE, Activation: softmax, Configuration: [1, 2, 2, 2, 1, 1, 1, 1, 2, 2, 1]

- **Dataset:** Concrete Comp – Regression
  Chromosome: Units: 1, Loss: MSE, Activation: softmax, Configuration: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

- **Dataset:** Forest Fire –Regression
  Chromosome: Units: 1, Loss: MSE, Activation: softmax, Configuration: [1, 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 2, 1]

- **Dataset:** Pysiocochemical – Regression
  Chromosome: Units: 1, Loss: MSE, Activation: softmax, Configuration: [1, 1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1]

- **Dataset:** Relative CT Slice – Regression
  Chromosome: Units: 1, Loss: MSE, Activation: softmax, Configuration: [1, 2, 1, 1, 2, 1, 1]

- **Dataset:** CIFAR-10 – Image classification
  Chromosome: Units: 10, Loss: CCE, Activation: linear, Configuration: [3, 3, 0, 0, 2, 3, 3, 0, 0, 0, 1]

- **Dataset:** CIFAR-100 – Image classification
  Chromosome: Units: 100, Loss: CCE , Activation: sigmoid, Configuration: [2, 0, 3, 3, 0, 0, 1, 2, 1, 1, 1]

- **Dataset:** MNIST – Image classification
  Chromosome: Units: 10, Loss: CCE, Activation: relu, Configuration: [2, 0, 2, 0, 3, 0, 1]

- **Dataset:** CrowdFlower – Sentiment analysis
  Chromosome: Units: 13, Loss: CCE, Activation: sigmoid, Configuration: [1, 2, 1, 1, 1, 2, 2, 1, 2, 2, 1, 1, 1, 2, 1]

- **Dataset:** IMDB – Sentiment analysis
  Chromosome: Units: 2, Loss: CCE, Activation: softmax, Configuration: [2, 1, 2, 1, 2, 1]

Some of the other chromosomes in the other runs for CIFAR-10 evolved similar architectures, but this was not always the case. For example, in one

particular run, the evolved architecture was: [0, 0, 0, 2, 2, 2, 2, 0, 0, 1]. In this case, the architecture was primarily made up of dropout and convolutional layers – there was only one fully connected layer. For certain runs, the evolved architectures were made up of deep networks containing only fully connected layers. For example, from the list of results above, consider the chromosome presented for the Sensorless Drive dataset; the architecture was [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1].

The number of epochs used throughout the optimisation of the neural networks was small. It would thus be of interest to extend this study in order to investigate the architectures which would be generated by using a larger number of epochs. One drawback of *API* is the computational effort required to obtain the results. On average, a single experiment took 49 minutes to run. It would be of interest to further decrease the population size to determine to which extent it can be reduced whilst retaining its current accuracy in discriminating between classification and regression problems.

3.7 Conclusion

In this chapter, we presented the Automated Problem Identification (*API*) algorithm, a genetic algorithm coupled to deep networks to automatically determining whether a dataset represents a regression or classification problem. While great effort has been put into improving and proposing new machine learning algorithms, typically the practitioner must still decide on the loss function, neural network architecture, number of units in each layer and select appropriate activation functions prior to the execution of the neural network. We propose *API* with the goal of moving towards general artificial intelligence and automated machine learning that requires little to no human intervention.

API was applied 20 times on 16 different datasets drawn from various problem domains and data characteristics. We find that *API* correctly identified the problem type with an average accuracy of 96.3% running only on a single CPU. Furthermore, *API* was able to recommend whether to use mean squared error or categorical cross entropy, a suitable number of units in the last layer together with the activation function, and furthermore, recommend a network architecture. Despite not being the primary focus of this study, the proposed algorithm generated interesting and relevant deep architectures.

The following chapter discusses the next phase of this research which is to develop an algorithm which can optimise the entire pipeline for creating

deep neural networks; whereby, the goal is simply to provide the algorithm with a dataset (without specifying if the problem is a classification or regression problem) and in return, get a deep neural network which can produce competitive results. This would completely remove the human from the pipeline. It would be of interest to determine if the evolved networks could outperform those created by humans. It is clear, with the efforts of various researchers that the machine learning community should steer towards algorithms which are completely automated requiring no human intervention.

Chapter 4

Neuro-evolutionary Architecture Optimisation

4.1 Introduction

Deep neural networks are powerful but unintuitive beasts whose wrangling requires experience along with significant trial and error to achieve good performance. The performance of such networks continue to improve as the depth is increased, e.g. [244]. Along with the rising influence of deep learning in all fields means it is becoming even more important to develop methods to automatically design optimal or near-optimal network architectures and hyper-parameters. Chapter addressed the problem of enabling an algorithm to determine the problem type; this chapter focuses on enabling an algorithm to optimise suitable network architectures to be trained on problems. Deciding on the exact nature and order of the layers, choice of activation functions, number of units in fully connected layers, number of filters in convolutional layers and other variables in creating deep neural networks is non-trivial. Is there a way to be competitive with only a small amount of computing power, such as a single GPU?

One solution, which we pursue here, is to evolve good neural networks through the use of EAs. Such neuro-evolutionary algorithms [234] are not new, spanning nearly three decades, see e.g. [297], [12], [105], beginning with a study that evolved the weights of the neural network [176].

We introduce related work and the rationale in section 4.2. We then discuss the proposed method and details related to the evolutionary algorithm in section 4.3. The experimental set up is presented in section 4.4. In section 4.5 we present the findings and conclude this chapter in section 4.6.

4.2 Related Work and Rationale

Here we briefly summarise recent related work on neuro-evolutionary algorithms, which, by contrast to this study, have used very significant computing resources. Real et al. [217] proposed a neuro-evolutionary approach to optimise neural networks for image classification problems using a parallel system executed on 250 computers and achieved considerable success on the CIFAR image problems. Zoph and Le [305] instead use recurrent neural networks along with reinforcement learning to learn good architectures. Eight hundred networks were trained on 800 GPUs. Such *et al.* [245] apply neuro-evolutionary to reinforcement learning tasks.

Miikkulainen *et al.* propose CoDeepNEAT [173] in which a population of modules and blueprints are evolved. The blueprints are made up of several nodes which point to particular modules representing neural networks. Thus their proposed approach allows for the evolution of repetitive structures by enabling the blueprints to reuse evolved modules. Desell [52] proposed EXACT, a neuro-evolutionary algorithm for deployment on a distributed cluster which they executed across 4500 volunteered computers and evolved 120,000 networks to tackle the MNIST dataset. Their approach did not use pooling layers and was limited to two dimensional input and filters.

In this chapter we propose *Evolutionary DEep Networks* (EDEN), a neuro-evolutionary algorithm that combines the strengths of GAs and deep neural networks to explore the search space of neural network architectures, their associated hyper-parameters and the number of epochs to be applied. In our study, we explore additional features – such as the optimisation of embedding layers – and increase the complexity on the existing research. With *EDEN* we are interested in addressing two questions: can we evolve generally good architectures and hyper-parameters for a broad range of problems (not just image classification)? Can this be successfully achieved on a single GPU, as opposed to the very large clusters used in previous studies?

We interface *EDEN* to TensorFlow [1] and thus new layers, functions and other features can easily be incorporated and controlled by *EDEN* as these represent function calls to the respective TensorFlow functions. Additionally, *EDEN* is not limited to TensorFlow, other modern deep neural network platforms can be interfaced. Figure 4.1 illustrates an example of a neural network architecture encoded by an *EDEN* chromosome.

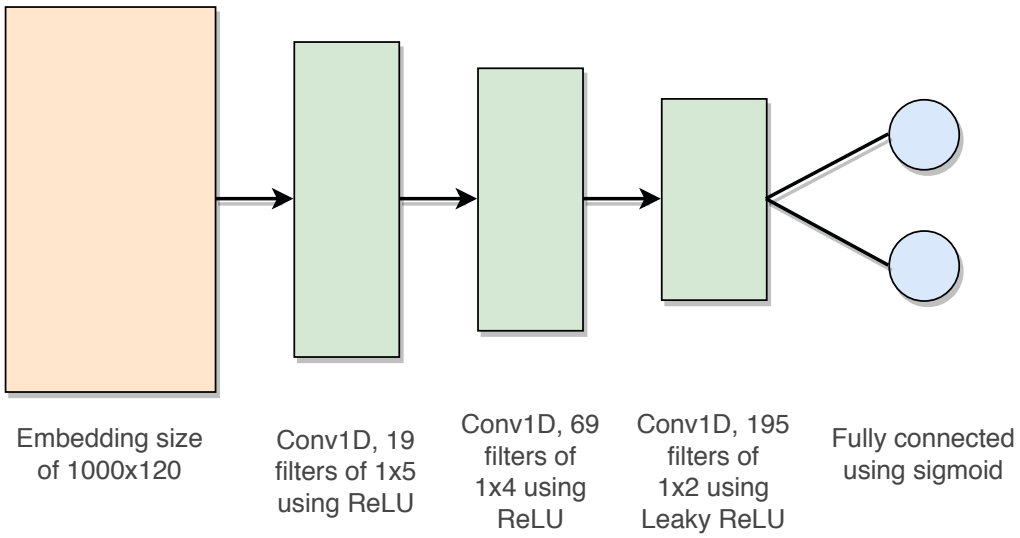

FIGURE 4.1: Each *EDEN* chromosome contains two genes, encoding the learning rate and a neural network. The figure illustrates an example of a neural network evolved using *EDEN* for a sentiment analysis task. *EDEN* created an embedding layer with an output dimension of 120, followed by three 1D convolutional layers. *EDEN* evolved the number of filters, each filters' dimension along with each corresponding activation function. For the last layer, the selected activation function which *EDEN* determined was the sigmoid function. The learning rate for this chromosome is 0.0023.

The associated video[1] illustrates the evolution of the chromosomes during the execution of *EDEN* on the MNIST image classification problem. The population converges to an efficient solution made up primarily of two-dimensional convolutional layers, similar to the state of the art. This convergence is illustrated in the video.

4.3 Proposed Approach

In this study we use the traditional GA. We additionally increment the number of neural network epochs along with the number of generations to explore the best value for the number of epochs. Algorithm 5 presents the GA used.

We choose to use GAs since the complexity of the chromosomes can be increased or decreased based on the number of genes which are encoded. GAs provide a further key advantage over other optimisation algorithms: they fluently handle complex combinations of discrete (e.g. layer type) and continuous (e.g. learning rate) search spaces, making them ideal for neuro-evolutionary studies; e.g. [217, 63].

[1] https://vimeo.com/234510097

Algorithm 5: Modified genetic algorithm used in this study

```
   input: epochs: number of neural network epochs
   input: population_size: population size
   input: generation_max: maximum number of GA generations
1  begin
2      generation ← 0.
3      epochs ← epochs.
4      population_size ← population_size.
5      Create an initial population of chromosomes.
6      Evaluate the initial population.
7      while generation ≤ generation_max do
8          if generation ≠ 0 then
9              epochs ← (epochs +1).
10             population_size ← (population_size −10).
11         Select the parents.
12         Create offspring using the genetic operators.
13         Replace the current population with the new offspring created
             in step 12.
14         Evaluate the current population.
15         generation ← generation + 1.
16     return The best chromosome.
```

4.3.1 Proposed Chromosome

Each *EDEN* chromosome is made up of two genes, and these genes constitute the required components to optimise a single neural network on some given input classification dataset. The two genes encode the learning rate and the network architecture. The learning rate within the chromosome denotes the actual value which is applied during the training of the neural network. The architecture represents the exact order of the neural network layers and operations. Figure 4.2 illustrates an example of an *EDEN* chromosome.

4.3.2 Network Layers

The following layers and operations were made available to *EDEN*: two-dimension convolution [139], one-dimension convolution [122], fully connected, dropout [243], one-, and two-dimension max pooling [303] and embedding [161]. Inappropriate choices (such as using a 2-D convolution for a text sentiment problem) are penalised as described in [63].

For the sentiment analysis tasks, instead of using pre-trained vectors such as Word2Vec [174], or setting a pre-determined embedding dimension size, we decided to allow *EDEN* to learn the dimension of the word embeddings as

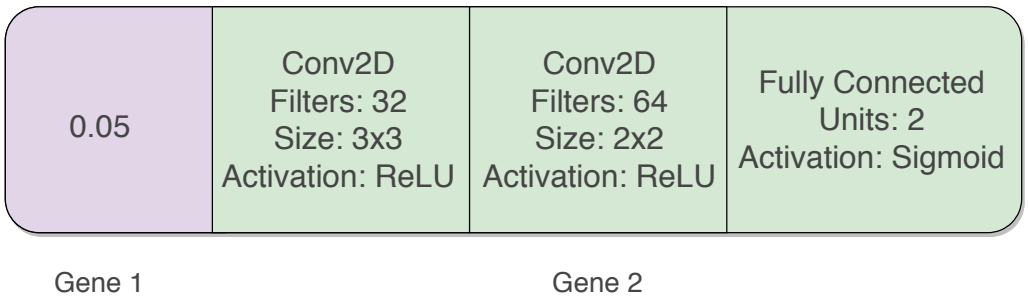

FIGURE 4.2: An example of an *EDEN* chromosome. There are two genes. The first gene (purple) encodes the learning rate and the second gene (green) encodes the neural network architecture along with the hyper-parameters for the various layers.

part of the optimisation. Figure 4.3 illustrates how the embedding operates. We created a dictionary by mapping each unique word to their frequency count in the training data. We took the top 1000 most frequent words and used this to encode the text into vectors of integers. Enabling *EDEN* to optimise both the vocabulary size and the embedding would result in significant computation time and hence this was not included in this study.

4.3.3 Activation Functions

When a layer is randomly generated an activation function is also randomly selected. Convolutional layers can choose between the following functions: {linear, leaky relu, prelu, relu}. Fully connected layers choose from: {linear, sigmoid, softmax, relu}. The last fully connected layer in the network can use any of: {linear, sigmoid, softmax}. These functions were selected as they are commonly used in literature. It is however possible to include a larger number of activations functions.

4.3.4 Initial Population Generation

The initial population generation method used in this study was inspired by the ramped-half-and-half method proposed by Koza [128] which enables the creation of candidate solutions of various sizes. In a similar manner, we implemented an initial population generation method that would create neural network architectures of various sizes to increase the amount of diversity in the initial population (as opposed to a population that is skewed towards a particular size).

Algorithm 6 presents the initial population generation method used in this study and algorithm 7 outlines the pseudocode on how a chromosome is

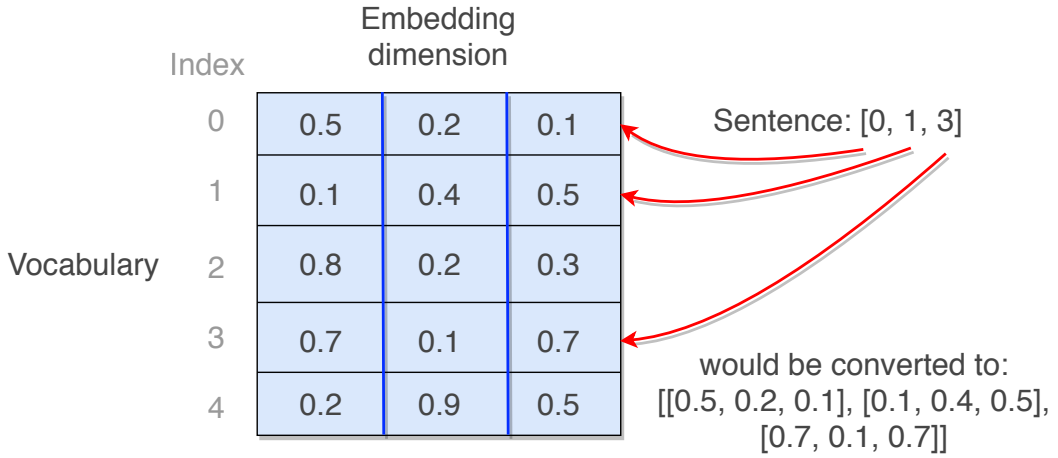

FIGURE 4.3: An example of the embedding operation. In this example assume that a vocabulary has been created which contains 5 words indexed 0 to 4. Assume that a sentence needs to be converted in the embedding space and the sentence is mapped using the vocabulary to corresponding integer values *[0, 1, 3]*. The embedding layer is presented in the figure. In this example the vocabulary size is 5 and the embedding dimension is 3 (user-defined). Thus, this layer maps the vocabulary indexed words into a vector of length 3. The example illustrates how the sentence *[0, 1, 3]* is converted into *[[0.5, 0.2, 0.1], [0.1, 0.4, 0.5], [0.7, 0.1, 0.7]]* by replacing the indices with the corresponding embedding

.

randomly generated. In certain cases, invalid architectures can be generated, these invalid architectures are discarded and a new one is generated.

Given our computational limitations, we had to limit the search space by setting bounds on certain variables. Real *et al.* [217] did not implement these limitations, however it is worth noting that in their study they used 250 machines. The *keep* probability (probability that a weight is not removed during dropout) for dropout was randomly generated between 0 and 1 as these are the only acceptable values.

The bounds for each variable are listed below. These were empirically determined through trial experiments.

- number of filters in 1D and 2D convolution: [10, 100]
- filter size for 1D and 2D convolution: [1, 6]
- kernel size for 1D and 2D max pooling: [1, 6]
- number of units in fully connected layers: [10, 100]
- embedding layer output size: [100, 300]

Algorithm 6: Creating initial population of chromosomes of various architecture sizes

```
   input: population_size: population size
1  begin
2      for i ← 0 to population_size do
3          Generate chromosome with size = (⌊i/10⌋ + 1)
4          Determine number of parameters
5          Evaluate chromosome's validation accuracy
6          Add chromosome to initial population
```

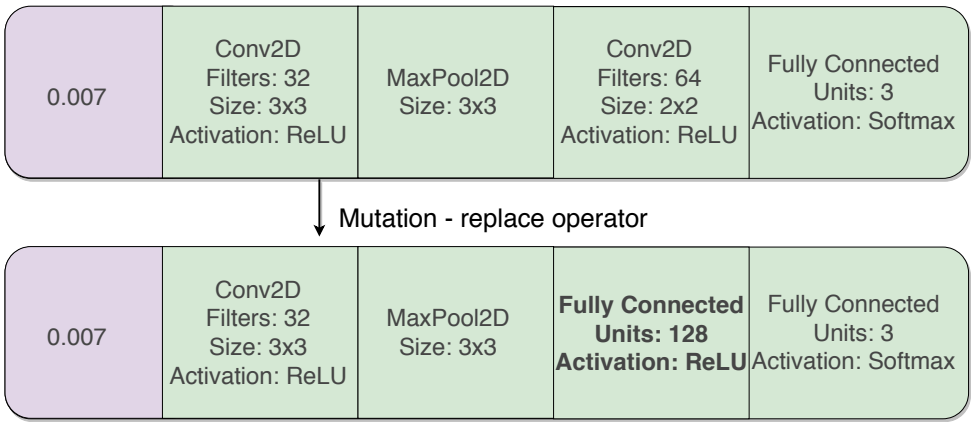

FIGURE 4.4: Illustrating the replacement mutation operator. The chromosome on top is the parent chromosome and the one on the bottom is the offspring. The last convolutional layer in the parent was replaced with a fully connected layer in the offspring (highlighted in bold).

4.3.5 Mutation

The recombination genetic operator was not included in our study, similar to Real *et al.* [217]. For each execution of the mutation operator a single parent is obtained using tournament selection. The mutation operator is applied to the parent to generate $offspring_1$. The mutation operator is then applied to $offspring_1$ and consequently creates $offspring_2$. The fitness of $offspring_1$, $offspring_2$ and the original parent chromosome are compared. The chromosome with the lowest fitness (the proposed method is formulated as a minimisation problem) is returned and placed into the new population. Preliminary experiments revealed that performing mutation once on a parent prohibited the algorithm from sufficiently exploring the search space. It is for this reason that we repeat the mutation operator to generate two variations of offspring.

The details about how the mutation operator changes a chromosome are as follows. For a given chromosome, the operator randomly changes either

Algorithm 7: Creating an *EDEN* chromosome.

```
   input: chromosome_size: maximum number of genes in chromosome
 1 begin
 2     Initialise an empty chromosome.
 3     layer_type ← 'cnn'
 4     for i ← 0 to chromosome_size − 1 do
 5         if i = 0 then
 6             dropout_allowed ← false
 7         else
 8             dropout_allowed ← true
 9         new_layer ← CreateLayer(dropout_allowed, layer_type)
10         Append newlayer to chromosome
11         if newlayer is fully connected then
12             layer_type ← 'non-cnn'
13     Randomly create fully connected layer and append to
        chromosome
14     return chromosome.
15 Function CreateLayer (dropout, type)
16     if type = 'cnn' then
17         if dropout = true then
18             Randomly create convolution, fully connected or dropout
                operation
19         else
20             Randomly create convolution layer
21     else
22         if dropout = true then
23             Randomly create fully connected layer or dropout
                operation
24         else
25             Randomly create fully connected layer
```

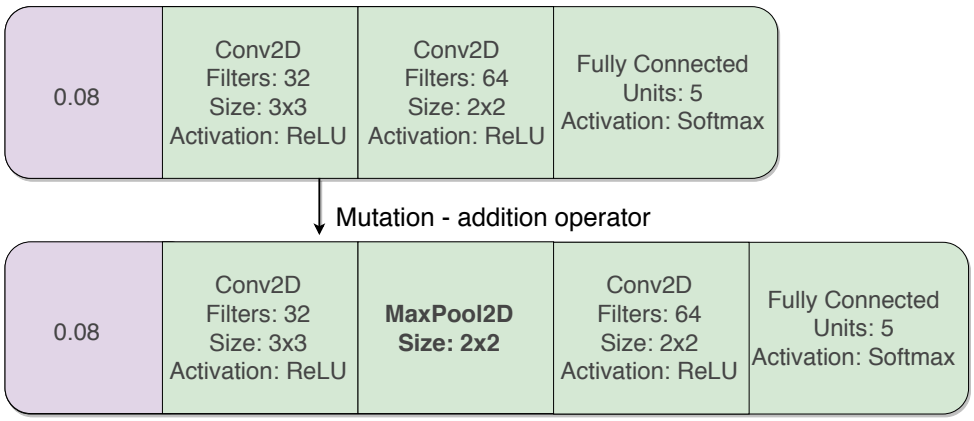

FIGURE 4.5: Illustrating the addition mutation operator. The chromosome on top is the parent chromosome and the one on the bottom is the offspring. The operator added a 2D max pooling layer after the first convolutional layer in the parent (highlighted in bold).

the chromosome's learning rate or the neural network layers. In the case that the learning rate is selected, then a new value for the learning rate is randomly generated as was discussed in section 4.3.1. In the case where the neural network layers are selected, then the operator either adds a new layer, deletes a layer or replaces one. Figures 4.4, 4.5 and 4.6 illustrate the replace, addition and deletion mutation operators.

The choice is made based on the size of the architecture. If the size of the chromosome's architecture has reached the maximum size (predetermined by the experimenter), then a layer can either be deleted or replaced. However, if the size is less than the maximum allowed size then a layer can either be added, deleted or replaced. A constraint was put in place so that the mutation operator cannot remove the first or last layers (which would resulting in an invalid network).

Deletion is performed by randomly selecting any layer (excluding the first or last layers) and removing it from the network architecture in the chromosome. *Replacement* is performed by randomly selecting a layer within the architecture and removing it. An entirely new layer is generated and inserted in the same position as the one which was removed. *Addition* generates a new layer and adds it anywhere in the architecture.

It is possible that the randomness within the mutation results in invalid neural network architectures. After each application of the mutation operator, a check is performed to assess the validity of the resulting architecture. If mutation generates an invalid architecture, then the mutation operator is applied again until a valid one is generated.

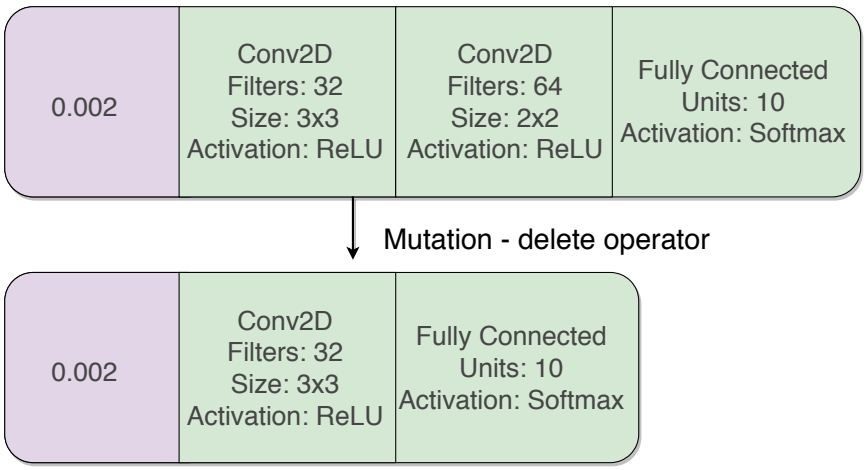

FIGURE 4.6: Illustrating the delete mutation operator. The chromosome on top is the parent chromosome and the one on the bottom is the offspring. The last convolutional layer from the parent was deleted.

The number of neural network parameters is computed for each offspring. These parameters represent the number of trainable weights in the neural network. Larger values denote more complex models, and small numbers consequently denote less complex ones.

4.3.6 Chromosome Evaluation

A fitness function is required to steer *EDEN* towards an optimal solution. For this study, we chose a fitness function that makes use of the error on the validation set (the dataset was split into training, validation and testing subsets) as well as the number of trainable parameters. The relative importance of these two is controlled by α, a complexity parameter. This fitness function rewards less complex and more accurate models compared to more complex, less accurate ones. Furthermore, it helps to break ties when two chromosomes have the same validation error. We fix $\alpha = 1$ (such that there is an equal weighting between the network error and complexity), but this can be changed depending on the relative importance of performance versus the need for small networks in the problem at hand.

Once the mutation operator generates an offspring, the architecture and hyper-parameters which are encoded in the chromosome are used to train a neural network using TensorFlow. The training data is used in the training of the network. The categorical cross entropy loss function is used during training. Once the network is done training then the neural network is evaluated on the validation data using the fitness function. The fitness obtained from

the function is then stored as the chromosome's fitness. The fitness function used in this study is presented in equation 4.1.

$$\text{fitness}(\text{Net}) = \text{val}_{\text{error}}(\text{Net}) + \alpha \left(1 - \frac{1}{\text{N}_\text{p}(\text{Net})}\right) \tag{4.1}$$

where

Net = the neural network being evaluated
$\text{val}_{\text{error}}$ = the validation error
N_p = the number of trainable parameters
α = complexity parameter (default $\alpha = 1$)

4.4 Experimental Setup

For each dataset, we executed *EDEN* 5 times and averaged the results (similar to [217]). *EDEN* was evaluated on a single machine with a MSI GeForce GTX1070 and 16GB of CPU RAM. During the evolutionary process an experiment used between 4GB to 7GB CPU RAM based on the dataset, and the GPU utilisation varied from 50 to 99 percent during the training of the neural networks. The algorithm was developed in Python 3.6.1, TensorFlow 1.2.1 and Keras 2.0.6 [35]. The operating system was Ubuntu 16.04 LTS.

4.4.1 Datasets

Table 4.1 presents the datasets for which *EDEN* was evaluated on. IMDB [161] and Electronics [112] are sentiment analysis datasets. The other datasets – namely, MNIST [137], CIFAR-10 [129], Fashion-MNIST [285] and the two EMNIST datasets [40] – were image classification problems. For each dataset, *EDEN* was trained on the training data, the validation set was used to evaluate the performance of the chromosomes and the test set was used when reporting the results. Each dataset was obtained pre-labelled with the ground truth. The training and testing split is presented in the table. The datasets did not contain any missing values, and the class values were converted into their respective one-hot encoded values.

4.4.2 Parameters

Table 4.2 present the GA and neural network parameters used throughout this study. Preliminary runs were conducted by searching a range of values

TABLE 4.1: The 7 datasets used in this study. The number of training and testing samples are provided along with the number of classes for each dataset. The IMDB and Elec datasets are sentiment analysis problems, and the remaining datasets are image classification problems.

Dataset	Training	Testing	Classes
CIFAR-10	50,000	10,000	10
Elec	25,000	25,000	2
EMNIST - Balanced	112,800	18,800	47
EMNIST - Digits	240,000	40,000	10
Fashion-MNIST	60,000	10,000	10
IMDB	25,000	25,000	2
MNIST	60,000	10,000	10

TABLE 4.2: The GA and neural network parameters used in this study. We conducted additional experiments to select these parameters. The number of generations was not set to a high value to avoid extreme runtimes. The neural network parameters were used by *EDEN* during the training of the neural networks.

Parameter	Value
Number of generations	10
Initial population size	100
Tournament size	7
Number of epochs	starting value of 3, incremented by 1 every generation
Weight initialisation - mean & standard deviation	0.00, 0.01
Batch size	1024
Optimiser	Adam [123]

for each of the hyper-parameters to obtain these final values. The purpose of *EDEN* was to, amongst other things, evolve the neural network's hyper-parameters and thus the parameters presented in the table were the only values which were input into *EDEN*.

4.5 Results and Discussion

Table 4.3 presents the average test accuracy and number of trainable parameters. The best results (for the population size which we used) were obtained when using the Adam optimiser. *EDEN* was initially configured to include the optimiser function in the chromosome, but the results revealed that this did not improve performance. It is possible that a larger population size would have yielded interesting results by searching for the most optimal network optimiser.

TABLE 4.3: The average best *EDEN* test accuracy (%) after 10 generations and 13 epochs of training (standard deviation shown in parentheses). The average number of trainable *EDEN* parameters and the previous state-of-the-art results and reference are also shown along with the average learning rates (denoted LR) which were evolved for the best chromosomes.

Dataset	Test Accuracy	LR	Params	State of the art (%)
CIFAR-10	74.5 (3.1)	0.0024	172,767	**97.14** [79]
Elec	87.2 (0.5)	0.0040	26,625	**93.17** [272]
EMNIST-Bal.	**88.3** (0.8)	0.0019	168,843	78.02 [40]
EMNIST-Digits	**99.3** (0.1)	0.0027	3,001,576	95.90 [40]
Fashion-MNIST	**90.6** (0.5)	0.0059	4,624,447	89.7 [285]
IMDB	85.8 (0.6)	0.0053	319,185	**93.34** [272]
MNIST	98.4 (0.3)	0.0031	1,857,601	**99.79** [273]

Dataset	Train time
IMDB	9
Elec	7
EMNIST-balanced	18
EMNIST-digits	24
CIFAR-10	12
Fashion-MNIST	6

TABLE 4.4: Average training times, in hours, for a single *EDEN* experiment.

EDEN achieved new state-of-the-art results on the EMNIST-balanced, EMNIST-digits and Fashion-MNIST datasets. For the two sentiment analysis tasks (Elec and IMDB) *EDEN* evolved neural networks which produced good – but sub-state-of-the-art – accuracy despite *EDEN*'s ability to optimise the embedding layer. In future work, we will determine the effect of also allowing *EDEN* to optimise the vocabulary size. The average evolved learning rate ranged between 0.00186 and 0.0059. The average execution times, in hours, for a single *EDEN* experiment of ten generations are shown in table 4.4. In addition, we enforce the constraint that no networks receive more than 13 epochs of training. As a result *EDEN* took, on average, 12 hours for the MNIST dataset (accuracy 98.4%); significantly less than the 2 month execution time of EXACT which achieved a similar accuracy of 98.32% [52].

EDEN did not, however, produce competitive results on CIFAR-10, obtaining an average test accuracy of 74.5% after 13 (80.5% after 100 epochs) training epochs of the final network evolved after 12 hours, compared to the current 97.14% state-of-the-art [79]. This is primarily due to the 7-layer depth constraint we imposed due to running *EDEN* on a single GPU. As a result the best model that *EDEN* evolved had only 172,767 parameters, only 0.7% of the

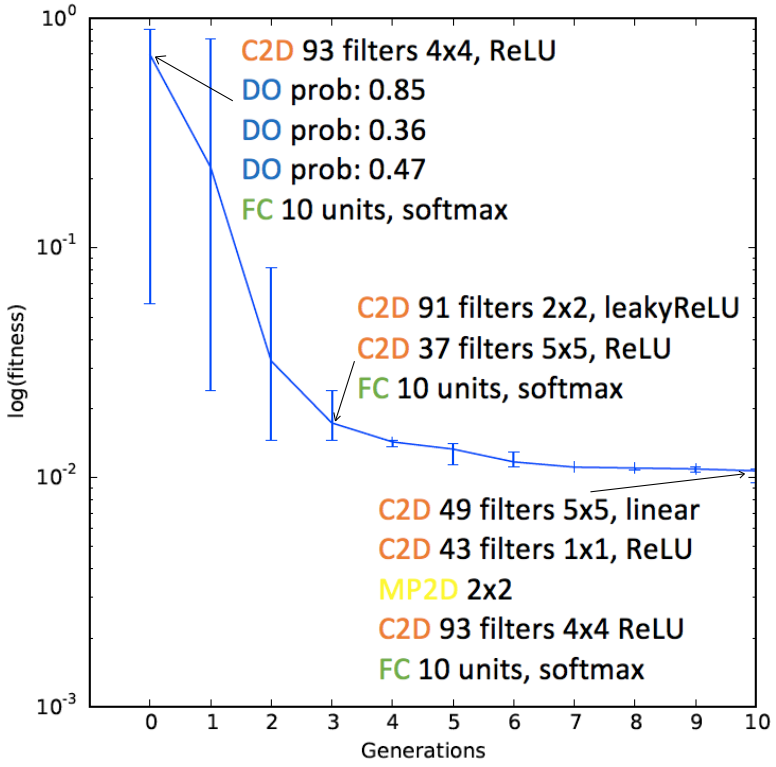

FIGURE 4.7: Illustrating the change in mean fitness over the GA generations for the MNIST data. Error bars show the 5% and 95% percentile values in fitness across the population. Initially there is significant variance in the fitness which reduces as the solutions improve and the population converges. We also show the best three networks from the initial, mid-point and final generations, along with their associated hyper-parameters. Here C2D, MP2D, DO, FC represent 2D convolution, 2D max pooling, dropout and fully connected layers respectively.

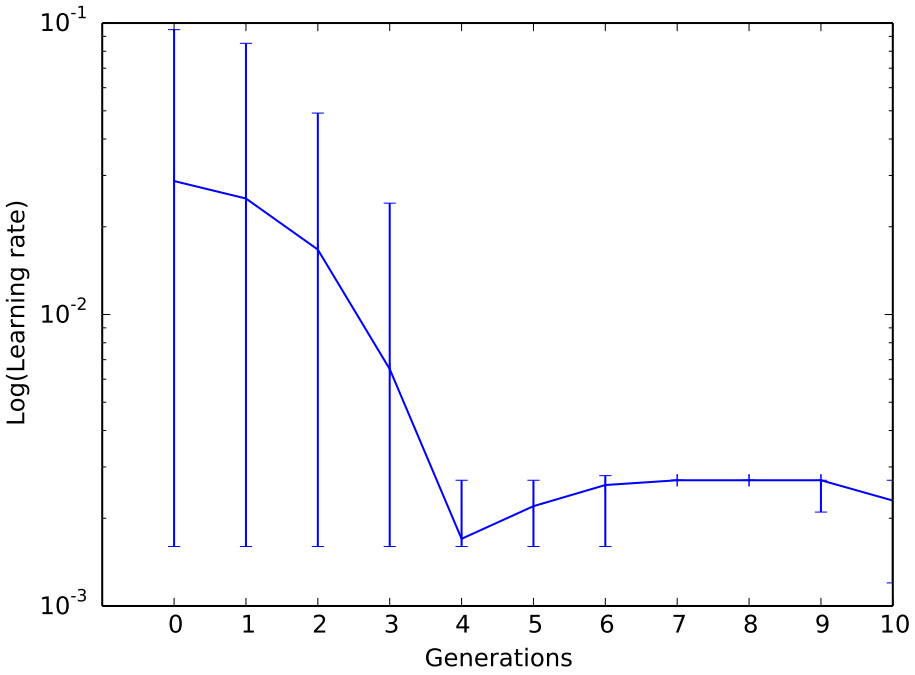

FIGURE 4.8: Change in mean learning rate over the GA generations. Error bars show the 5% and 95% percentile value in terms of the learning rate variance in the population. Initially the chromosomes are random so there is a lot of variance in the learning rate. This changes as the population converges towards better solutions.

26.2 million used in the state-of-the-art [79]. Figures 4.7 and 4.8 illustrate the change in fitness and learning rate over the evolutionary process. Both figures show the convergence of the population. The fitness rapidly decreases from the random initial population to generation 3, after which the fitness decreases at a slower rate.

4.6 Conclusion

Determining optimal or efficient deep neural network architectures and hyper-parameters is a challenging task. Researchers and practitioners who are new to the creation of deep neural networks can benefit from algorithms which automatically create architectures and determine hyper-parameters. In our study, we propose *EDEN*, a neuro-evolutionary algorithm that interfaces with TensorFlow – or any other deep neural network platform – to automatically create architectures and optimise hyper-parameters. Here *EDEN* was evaluated on classification problems, but can easily be applied to regression problems.

EDEN is designed to evolve efficient deep networks and for each dataset was executed on a single GPU running for 24 hours or less. The findings reveal that competitive results can be obtained using significantly less computational power than has been deployed in other neuro-evolutionary studies. Evaluated on image classification and sentiment analysis problems, *EDEN* achieves state-of-the-art results in three of seven datasets. This study is also a first attempt at applying neuro-evolution to the creation of 1D convolutional networks for sentiment analysis, optimising an embedding layer for sentiment analysis. In future work, we intend on extending *EDEN* to evolve generative adversarial networks architectures, as well as exploring parallel implementations. Furthermore, modifications such as using simulated annealing [133] to optimise the learning rate would be interesting as opposed to randomly generating values.

Chapter 5

Automated Classification of Text Sentiment

5.1 Introduction

The amount of data collected from users around the world and stored for posterity has skyrocketed over the past decade as websites such as Twitter, Amazon and Facebook have facilitated the publication and aggregation of micro opinion pieces that allow individuals to record their sentiments towards things, people and events. This data is clearly of value to researchers, organizations and companies to understand sentiment both as individuals and on average, and as well as to identify trends. The automated detection of emotions and attitudes towards a particular subject, event or entity is what we will call sentiment analysis [193, 148].

In section 5.2 the rationale and related work is presented. Section 5.3 presents the *classification-value* pair, a pair of values which allows the GA to perform sentiment analysis. The proposed algorithm is described in section 5.4. The experimental setup is presented in section 5.5 and the results are revealed in section 5.6. Section 5.7 discusses an extension to the proposed approach which attempts to incorporate context into the algorithm. Finally, section 5.8 concludes this chapter.

5.2 Related Work and Rationale

The ability to identify sentiment in text, referred to as *sentiment analysis*, is one which is natural to adult humans. This task is, however, not one which a computer can perform by default. Identifying sentiments in an automated, algorithmic manner will be a useful capability for business and research in

their search to understand what consumers think about their products or services and to understand human sociology.

Sentiment analysis has been applied to many problem domains; for instance, determining sentiments of consumers towards products, or mining social media to gain an understanding of the public's opinion on matters such as corruption [148, 193, 216].

For adult humans, interpreting the underlying emotions in text is usually performed unconsciously and with apparent ease. We are able to recognize emotions in emails, sentiments in our social media feed and appreciate the subtle nuances of conflicting views in novels. Nevertheless, even for humans text can be notoriously easy to misinterpret. For machines, on the other hand, sentiment analysis is highly non-trivial. From the year 2000 onwards, a number of researchers have begun contributing towards the field of sentiment analysis [193]. This area of research is highly active, increasingly so, due to the vast amount of digital information available, and the amount of sentiments expressed online. With the rapid increase in computational power available in the recent years and the extreme amount of data available online, it is clear that developing novel sentiment analysis methods will be beneficial to organisations in order to enable them to understand what the public feels about their products and services.

GAs have been used before in sentiment analysis studies, though not primarily for actual sentiment determination but rather for feature selection and reduction, e.g. [3, 115, 45]. In the work of Abbasi [3] *et al.* the chromosomes had length equal to the total number of features, and the genes were encoded with a *0* or a *1* depending on whether or not that particular feature was to be used or not. The GA optimized which features to use from the original set, and a support vector machine (SVM) classifier [98] (for which the aim in a classification problem is to obtain a decision boundary which best separates the training examples within the classes) was then applied to that feature set in order to train and predict the reviews. Genes were encoded in a similar manner for feature selection with the ultimate goal of reducing the number of features in the study of Kalaivani and Shunmuganathan [116].

GAs were also used to optimize features in several other studies, such as that of Paramesha and Ravishankar [197] who used a GA in order to allocate weights to features. Govindarajan [88] proposed an ensemble approach using Naive Bayes and a GA. Smith [238] proposed the use of GA to reduce the number of features as did Acampora and Cosma [5].

Carvalho *et al.* [29] present a novel GA approach whereby a fixed chromosome is split in two parts, a positive and a negative part. A set of 25 positive and 25 negative words were seeded into the algorithm. Their approach attempts to find which of those words should be added into the respective parts of the GA chromosomes in order to maximize the accuracy of classifying Twitter tweets. A chromosome is then evaluated using a distance measure based on the words in the tweets in relation to the words in the chromosome. Thus, for example, if a particular chromosome is evaluated on some tweet, and the words in the tweet are considered to be nearer (based on the distance measure) to the positive words in the chromosome than to the negative, then the tweet is classified as positive.

Das and Bandyopadhyay [50] make use of a GA for subjectivity detection. Even though this area of research does not deal with sentiments, the research is aligned. Ten features were chosen and a number of predetermined values were assigned to each feature. An example of two features used were parts-of-speech and SentiWordNet values. The former takes up to 45 possible parts-of-speech values; and latter 2 values, positive or negative. The aim behind the research was to optimize the best set of features.

By contrast, the rationale behind the present study is not to propose a new GA feature selection method; instead, the focus is to propose a GA that determines the sentiment of reviews without making use of a feature set. Furthermore, our approach treats each individual piece of text with a sentiment as a mathematical formula made up of unknown variables corresponding to each word in the text. Thus, the goal is to use a GA to simultaneously solve for the unknown variables as a step towards correctly predicting the total sentiment of a piece of text.

5.3 Classification-Value Pair

In our study, each word is assigned both a 'classification' and a 'value' that we call a classification-value pair in the form '*classification:value*'. Classifications take on one of two types, namely either *sentiment* or *amplifier*. Intuitively this captures the difference between words that carry sentiment directly (e.g. 'horrible', 'sad', 'wonderful') and adjectives/adverbs that modify the sentiment of the following word (e.g. 'very', 'not', 'little'). In addition to this classification every word is given exactly one value associated with that classification, taken from this list:

- Sentiment $\in$ {*-1.0*, *0.0*, *1.0*}

- Amplifier $\in \{0.5, 1.0, 1.5\}$

For this classification-value pair, a sentiment value of -1, 0 and 1 represents a negative, neutral and positive sentiment respectively. The three values for the amplifier represent different intensification values, i.e. a value of 1.5 is a larger amplification than a value of 0.5. These values were selected by conducting various preliminary runs on a range of positive and negative values.

A word is referred to as an unknown word if its classification-value is not known. Examples of three classification-value pairs are: *sentiment:1.0*, *amplifier:0.5*, and *sentiment:-1.0*.

The goal of this study is to optimize and determine the classification-value pairs for certain unknown words within a data set, given that, a number of words already have known classification-value pairs. The words which already have a known classification-value pair are stored in a dictionary. Words in a dictionary do not have to be optimized and their classification-value pairs are never altered.

In this study we use two dictionaries, one for sentiment words and one for amplifier words. Known sentiment words were added into the sentiment dictionary, and similarly, known amplifier words were added into the amplifier dictionary. This was done to provide seeds as to guide the GA to converge to the correct solutions. Furthermore, the proposed algorithm can extend these dictionaries in order to create a sentiment lexicon. Details regarding which dictionaries were used are provided in section 5.6.

5.4 Proposed Methods for Optimizing Classification - Value Pairs

This section describes the use of machine learning in order to optimize the classification-value pairs for the unknown words in the sentences of a data set. We propose *Genetic Algorithm for Sentiment Analysis* (*GASA*). Each aspect of the GA is explained in terms of how it has been adapted for *GASA* in the following subsections.

5.4.1 *GASA* chromosome representation

Each gene within a chromosome is made up of the classification-value pair for an unknown word (not in the sentiment or amplifier dictionary). The

length of the chromosome is equal to the number of unknown words in the training corpus. The classification for each unknown word corresponds to a gene in the chromosome, and thus the classification of unknown words: $word_1, word_2, word_3, \ldots, word_n$ is mapped to: $gene_1, gene_2, gene_3, \ldots, gene_n$ — where n represents the number of unknown words in a training corpus. This mapping is never changed.

In order to illustrate the chromosome representation, suppose there are three unknown words: $word_1, word_2, word_3$. Figure 5.1 illustrates an example of a candidate chromosome of length 3. The illustrated chromosome corresponds to the following classification:

- $word_1$ in gene position 1 is classified as a *sentiment* word with a value of 1.0
- $word_2$ in gene position 2 is classified as an *amplifier* word with a value of 0.5
- $word_3$ in gene position 3 is classified as a *sentiment* word with a value of 0.0

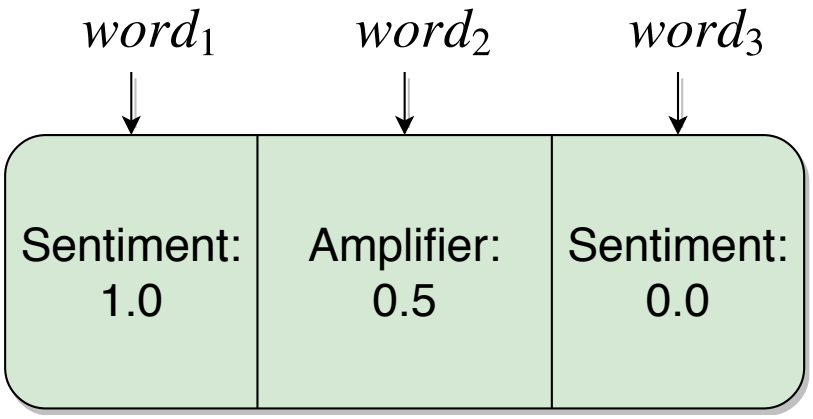

FIGURE 5.1: Example of a *GASA* chromosome. The chromosome has three genes, one for each word. Each word is mapped to a sentiment or amplifier type. The first word is classified as a sentiment with a value of 1 (positive sentiment), the second to an amplifier of 0.5 and the third to a sentiment of 0 (neutral sentiment).

5.4.2 *GASA* initial population generation

Prior to creating the initial population, the unknown words have to be input into the GA. The initial population size is set to the same value as the user-defined population size. Each chromosome has a fixed length which is

set to the number of unknown training words. The pseudocode for creating a chromosome is presented in algorithm 8. The genes which make up the chromosome are created by randomly selecting either a sentiment or an amplifier classification and assigned a value randomly as described in section 5.3.

Algorithm 8: Creating a chromosome.

```
   input: size: the number of unknown words
1  begin
2      Initialise the length of the chromosome to size
3      for each gene in the chromosome do
4          Randomly select a classification type.
5          Randomly select a corresponding value for the classification
             type previously obtained in step 4.
```

5.4.3 *GASA* chromosome evaluation

Every chromosome is evaluated on each example in the data set. In this study, an example corresponds to text. Assume that chromosome c is being evaluated. Chromosome c is applied to every example in the data set, and each word within the examples is examined in order to obtain its classification-value pair. Assume that chromosome c is evaluating example i, whereby the text for example i is made up of the following words: $w_1, w_2, w_3, \ldots, w_n$, and n denotes the length of example i.

If a word w_j from example i is in the sentiment dictionary, then w_j is classified as a sentiment word, and its corresponding value is retrieved from the sentiment dictionary. Similarly, if w_j is in the amplifier dictionary, then w_j is classified as an amplifier word, and its corresponding value is obtained from the amplifier dictionary. If however, w_j is unknown, then its classification-value pair is retrieved from the corresponding gene in chromosome c (since the genes encode words and their classification-value pairs).

Once the classification-value pair for every word in an example of data has been obtained, these classification-value pairs are converted into a mathematical expression in order to obtain the sentiment for the example. The mathematical expression is evaluated sequentially from left to right. Algorithm 9 presents the pseudocode to evaluate expressions. Amplifier words boost the sentiment words, whereas the sentiment words accumulate each other. If the final word is an amplifier, then that value is simply added onto

the result. A positive output denotes a positive sentiment, and a negative output denotes a negative sentiment.

The fitness of a chromosome is determined as the total number of examples for which the sentiment output by the chromosome is equal to the correct sentiment from the data set. Assume that some data set has sentences s_1, s_2, and s_3, and these have correct sentiments of *positive, negative, positive* respectively. If some chromosome evaluates each sentence to: *negative, negative, negative*, then the fitness of that chromosome is one, since, it only correctly classified the second sentence.

Algorithm 9: Pseudocode for arithmetically evaluating a sentence.

```
   input : sentence: the sentence to be evaluated
   output: The sentiment for the evaluated sentence
 1 begin
 2     sentiment_count ← 0
 3     amplifier_count ← 0
 4     for each word in the sentence do
 5         if word is an amplifier then
 6             amplifier_count ← amplifier_count + word's amplifier value
 7         else
 8             if amplifier_count is non-zero then
 9                 sentiment_count ← sentiment_count + amplifier_count ×
                    word's sentiment value
10             else
11                 sentiment_count ← sentiment_count + word's sentiment
                    value
12     if amplifier_count is non-zero then
13         sentiment_count ← sentiment_count + amplifier_count
14     return sentiment_count
```

5.4.4 *GASA* genetic operators

For this study we use mutation and crossover. Their implementation details are described below.

5.4.5 *GASA* mutation

The mutation genetic operator makes use of a single parent chromosome. The classification-value for a single gene in the parent is modified to a new one. Figure 5.2 illustrates the application of the mutation operator on a parent

chromosome, and the resulting offspring is illustrated. The second gene in the parent was changed from a classification of "amplifier" with a value of 0.5 to a classification of "sentiment" with a value of 1.0.

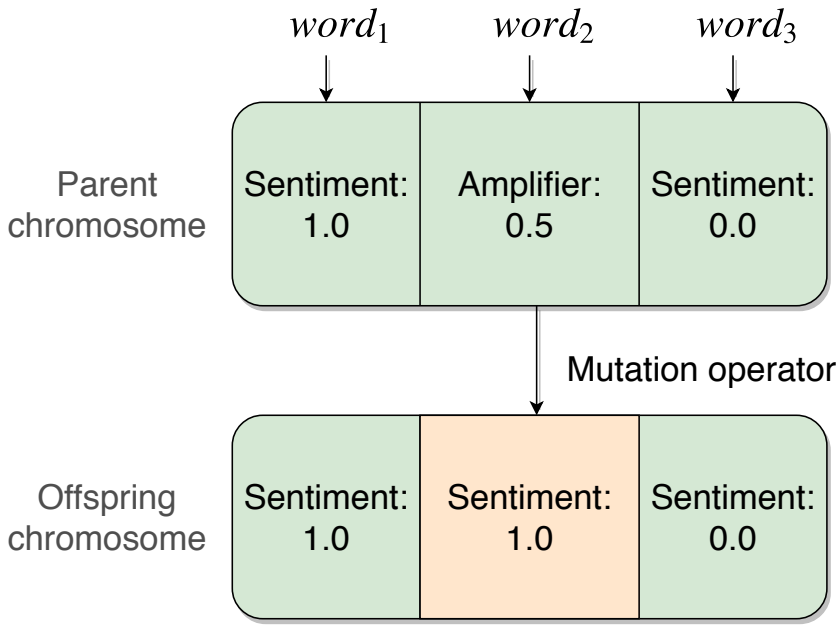

FIGURE 5.2: Example of *GASA* mutation. The second gene was selected for mutation and was changed from an amplifier with a value 0.5 to a sentiment with a value of 1.0. The other genes remain unchanged.

5.4.6 *GASA* crossover

The crossover method we implement randomly selects a position p in the range $[0, n]$ — where n denotes the length of the chromosome — within the parent chromosomes; the same position p must be selected within the two parents. Two offspring are created, and all the genes except those at position p are copied across to the corresponding offspring without modification. The genes at position p are swapped, i.e., the gene in position p from $parent_1$ is inserted into position p in $child_2$, and similarly, the gene in position p from $parent_2$ is inserted into position p in $child_1$.

Figure 5.3 illustrates the application of the proposed crossover operator on two parent chromosomes; the resulting offspring are also illustrated. In this case, the value of p was 2, implying that the second gene was swapped amongst the parent chromosomes.

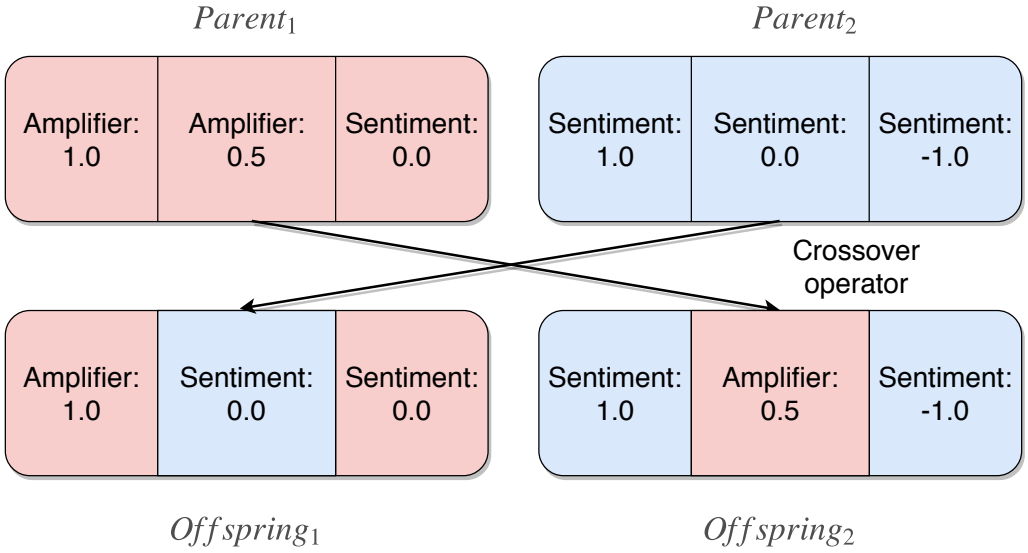

FIGURE 5.3: Example of *GASA* crossover. The second gene was swapped between the parents, i.e. the amplifier in the second gene from parent 1 was swapped with the sentiment in the second gene from parent 2. The result of the crossover is observed in the offspring chromosomes. All of the other genes remain unchanged.

5.5 Experimental Setup

This section describes the experimental set up which was used in order to evaluate the performance of *GASA*. *GASA* was programmed in Java and the experiments were conducted at the University of Geneva on the Baobab cluster.

5.5.1 Data sets

Based on the literature surveyed, there is no consistency in terms of the number of data sets used in previous studies. Furthermore, the total number of reviews also varies from one study to another. For example, Che *et al.* [30] used a data set containing only 878 reviews, whereas the data set used in the study of Wang *et al.* [274] had 108,891 reviews — the number of reviews largely differ between these two. Pang *et al.* [194] and Govindarajan [88] used 2,000 reviews, whereas Acampora and Cosma [5] used 95,084. Paramesha and Ravishankar [197] used 2,243. Carvalho *et al.* [29] used two data sets which had 359 and 1,908 reviews. There is no study which guides researchers to a set of recommended benchmark data sets for sentiment analysis.

For this study, eight data sets were constructed from several Amazon data sets. The Amazon data sets were obtained from Leskovec and Krevl [141] and McAuley *et al.* [168]. Each example in the Amazon data sets is made up of a short summary, a review and a star rating which were provided by a

user. The star rating was used to label the datasets using a value of 2.5 as a threshold between negative and positive classes. An example of an instance of data is as follows:

- **summary:** "Pittsburgh - Home of the OLDIES"
- **review:** "I have all of the doo wop DVD's and this one is as good or better than the 1st ones. Remember once these performers are gone, we'll never get to see them again. Rhino did an excellent job and if you like or love doo wop and Rock n Roll you'll LOVE this DVD !"

Four Amazon data sets were randomly selected, namely: Cell Phones and Accessories (Cellphone), Office Products (Office Prod), Grocery and Gourmet Food (Foods) and Video Games. From these four large data sets, eight data sets (four summary and four review data sets) were created for this study. The summary data sets were created by randomly selecting 1000 positive and 1000 negative summaries from a problem domain. Similarly, the review data sets were created by randomly selecting 1000 positive and 1000 negative reviews from a problem domain. For example, the created Cellphone review data set had reviews selected from the Amazon Cellphone data set only. Similarly, the created Foods summary data set had summaries selected from the Amazon Foods data set only.

These data sets were created since they represent different problem domains and contain a similar number of instances as compared to that presented in [194, 192] and allow for a large number of experiments to be performed. The eight data sets used in this study are listed in table 5.1. Stop words and other irrelevant words were not removed from the data. For these to have no contribution to the overall sentiment, GASA must classify them as either 'sentiment' or 'amplifier' with a value of 0.

5.5.2 Experimental parameters

The parameters used for the *GASA* experiments are presented in table 5.2. These parameters were obtained by exploring a large range of values on preliminary runs. The following section presents the results obtained by *GASA* on several experiments.

TABLE 5.1: Data sets used in this study. Four review and four summary data sets were created. Each data set had 2,000 examples.

Data set	**Number of positive/ negative instances**
Cellphone	1000/1000 reviews
Office Prod	1000/1000 reviews
Foods	1000/1000 reviews
Video Games	1000/1000 reviews
Cellphone	1000/1000 summaries
Office Prod	1000/1000 summaries
Foods	1000/1000 summaries
Video Games	1000/1000 summaries

TABLE 5.2: *GASA* parameters used in the study.

GASA Parameter	**Value**
Population Size	200
Parent selection method	Tournament
Tournament size	7
Maximum number of generations	500
Crossover rate	60%
Mutation rate	40%

5.6 Results and Discussion

This section is split into three subsections. The purpose of the algorithm discussed the first subsection was to determine whether a given word was a sentiment or an amplifier word. The second subsection describes experiments whereby the goal was to determine the value of the sentiment word. And finally, the third subsection presents the results when *GASA* was compared to other sentiment analysis algorithms. The results from the first two experiments demonstrate *GASA*'s ability to generate a sentiment lexicon whereas the third experiment illustrates how *GASA* performs as a sentiment analysis algorithm.

The amplifier dictionary was seeded with two words: "not" and "never"; each had a value of "-1", representing negation. The sentiment dictionary was obtained from [103] which we refer to as "HuLiu-6786". This dictionary contains 6,786 known sentiment words labelled as either positive or negative.

5.6.1 Predicting 'sentiment' or 'amplifier'

Several experiments were conducted whereby the classification problem was converted into a 2-class problem, namely sentiment and amplifier classes.

Each word in the HuLiu-6786 dictionary was considered as a sentiment word. Ten-fold cross-validation was used on the HuLiu-6786 dictionary whereby during each experiment, 9 folds from the dictionary were used for training, and the remaining fold from the dictionary was used for testing. *GASA* had to predict whether each word in the test fold was a sentiment or an amplifier. The training and testing data was made up of dictionary words and not reviews or summary data.

GASA's fitness function did not take into consideration (during the evolutionary process) whether or not a training dictionary word was correctly classified as a sentiment or not. Thus, during the evolutionary process for these experiments, *GASA* did not directly optimize the chromosomes in order to correctly distinguish between the two classes. Instead, *GASA*'s goal was to correctly classify the overall sentiment of as many instances (reviews or summary data) as possible.

For these experiments, two data sets were created from the Cellphone, Office Products, Foods and Video Games data sets. Namely, all of the summary data combined and all of the review data combined. Thus, the two combined data sets had 8,000 instances each.

The results for these experiments are presented in tables 5.3 and 5.4. Each row in the table represents a particular word frequency condition; this is followed by the corresponding average number of dictionary words in the training data which met the word frequency condition and the average test accuracy across the 10-folds. Note that this test accuracy is in terms of the 2-class problem of distinguishing between a sentiment or amplifier word as described above.

The word frequency condition is read as follows: a value of '> 10' means that the experiment only took into consideration the training dictionary words which occurred at least 10 times in the review/summary data. In table 3, there were an average of 446 training dictionary words which had a frequency of 10. Similarly, a value of '> 250' means that the experiment only took into consideration the training dictionary words which occurred at least 250 times in the review/summary data. There was an average of 23 words in the training data which had a frequency of 250. The condition '> 0' implied that a dictionary word had to occur at least once in the review/summary data.

The purpose of using the word frequency condition was to determine the effect on the number of times a word was present in the training data and *GASA*'s ability to correctly classify the words in the test set which also had

TABLE 5.3: Test accuracy (%) results on the two class problem (sentiment and amplifier) on all of the review data combined into a single data set. Ten-fold cross-validation was used.

Word Frequency	**Total number of dictionary words**	**Accuracy (%)**
> 0	1917	51.4
> 10	446	54.8
> 15	334	55.5
> 20	260	56.4
> 100	56	64.5
> 250	23	72.0

TABLE 5.4: Test accuracy (%) results on the two class problem (sentiment and amplifier) on all of the summary data combined into a single data set. Ten-fold cross-validation was used.

Word Frequency	**Total number of dictionary words**	**Accuracy (%)**
> 0	626	55.1
> 10	76	68.4
> 15	56	67.9
> 20	41	75.6

such a frequency.

When all of the words are taken into consideration, i.e. a frequency value '> 0' , *GASA* achieved an accuracy of 51.39% and 55.11% on the review and summary data respectively. The accuracy improved when the word frequency condition was increased. In terms of the review data, the accuracy went from 51.39% to 72.00% when the word frequency was increased from 'greater than 0' to 'greater than 250'.

For the combined summary data, when the words had a frequency of at least 20 the accuracy was 75.61% as opposed to an accuracy of 55.11% for a frequency condition greater than 0. The combined review data set had more words than the combined summary data set because the summaries are short text. For this reason, the conditions were stopped at 20 for the combined summary data. Words from the dictionary which occur with a small frequency are more challenging for *GASA* to correctly classify as a sentiment or amplifier since they occur infrequently in the data. Nonetheless, the findings reveal that *GASA* is able to extend a sentiment and amplifier lexicon provided that the words occur with a large frequency in the training data.

TABLE 5.5: Test accuracy (%) results on the two class problem (positive and negative sentiment) on all of the review data combined into a single data set. Ten-fold cross-validation was used.

Word Frequency	Total number of dictionary words	Accuracy (%)
> 0	1917	36.6
> 10	446	44.4
> 15	334	46.7
> 20	260	49.2
> 100	56	69.6
> 250	23	82.6

5.6.2 Predicting the value of the sentiment

A set of experiments was conducted in order to determine how effective *GASA* would be at classifying the sentiment value of a set of words instead of sentences. In order to achieve this, the HuLiu-6786 dictionary was used, and a certain percentage of the words in the dictionary were considered as unknown. The problem was converted into a 2-class classification problem, namely positive and negative sentiment values. Thus, in terms of the classification-value pair, only the "value" aspect was taken into consideration.

The HuLiu-6786 dictionary contained more sentiment words than were present in the data sets, and thus only the dictionary words found in the training data sets were considered — this set was named *S*.

Ten-fold cross-validation was used in the following manner: 10 folds were randomly created from *S*, 9 folds were seeded into *GASA* and the algorithm was executed as defined in section 5.4. At the end of the GA generational loop the algorithm had to predict the sentiment value of the words in the test fold as either "positive" or "negative". The predictions were then compared against the correct values in order to determine the accuracy. Similarly to the experiment described in subsection 5.6.1, *GASA* did not directly optimize the chromosomes in order to correctly distinguish between the two classes. *GASA*'s objective was to correctly classify the overall sentiment of as many instances (reviews or summary data) as possible.

For these experiments, the same data sets which were described in subsection 5.6.1 were used. The results for these experiments are presented in tables 5.5 and 5.6. Subsection 5.6.1 describes how to interpret the tables. Dictionary words in the combined summary data set which had a frequency value of at least 20 resulted in an accuracy of 95.12%. Sentiment words had to appear a greater number of times in the combined review data set in order to achieve a

TABLE 5.6: Test accuracy (%) results on the two class problem (positive and negative sentiment) on all of the summary data combined into a single data set. Ten-fold cross-validation was used.

Word Frequency	Total number of dictionary words	Accuracy (%)
> 0	626	53.0
> 10	76	86.8
> 15	56	92.9
> 20	41	95.1

higher accuracy; more precisely, words which had a frequency of at least 250 times resulted in an accuracy of 82.61%. These results reveal that *GASA* is able to extend a sentiment lexicon provided that the words occur frequently.

5.6.3 Comparison of *GASA* with commercial Sentiment Tools

How good is *GASA*? To check we compared *GASA* seeded with the HuLiu-6786 dictionary to other sentiment analysis methods including AlchemyAPI [106], MeaningCloud [170], NLTK [188], Lexalytics [142], LingPipe [8], Stanford sentiment analysis [239, 162], SentiStrength [260, 259] and Dandelion API [241]. AlchemyAPI, MeaningCloud, Lexalytics and Dandelion are commercial APIs. LingPipe and SentiStrength have both commercial and non-commercial licences. The results of the comparison are presented in table 5.7.

In terms of the summary data, the top 3 ranking methods in order of performance were LingPipe, AlchemyAPI and *GASA* with an average test accuracy of 77.75%, 72.88% and 69.92% respectively. When comparing *GASA* to the four commercial APIs, AlchemyAPI achieved the best accuracy, while *GASA* outperformed Dandelion, Lexalytics and MeaningCloud.

In terms of the review data sets, the top three performing methods were LingPipe, AlchemyAPI and SentiStrength. *GASA* was outperformed by two commercial API, namely AlchemyAPI and Dandelion. *GASA* achieved higher test accuracy when compared to two commercial APIs, namely, Lexalytics and MeaningCloud.

Subsections 5.7.3 and 5.7.3 presents some of the data and correct and incorrect classifications made my *GASA*.

TABLE 5.7: Test accuracy (%) illustrating a comparison between other commercial and non commercial sentiment analysis methods and GASA. The 70/30 holdout method was used, and all of the data sets had a size of 2000. Types "S" and "R" denote summary and review data sets respectively. Watson refers to Alchemy API, MC to MeaningCloud, LEX to Lexalytics, LP to LingPipe, SS to SentiStrength, DL to Dandelion and SD to Stanford sentiment analysis.

Data set	**Type**	**Watson**	**MC**	**NLTK**	**LEX**	**LP**	**SS**	**DL**	**SD**	**GASA**
Cellphone	S	75.67	60.00	60.67	37.50	80.17	68.00	55.17	54.83	74.33
Office Prod	S	71.50	59.83	59.83	39.83	80.00	69.83	53.50	55.83	70.67
Foods	S	68.83	52.67	57.67	40.00	74.50	64.50	50.17	51.00	63.33
Video Games	S	75.50	59.83	57.67	38.33	76.33	66.17	55.00	51.33	71.33
Average Summary		*72.88*	*58.08*	*58.96*	*38.92*	*77.75*	*67.13*	*53.46*	*53.25*	*69.92*
Cellphone	R	70.17	62.33	64.17	58.50	81.50	67.67	65.00	54.17	69.17
Office Prod	R	71.50	64.83	64.83	60.17	81.50	72.00	68.33	53.67	68.67
Foods	R	71.17	62.83	63.83	64.00	78.83	70.67	69.67	52.83	64.83
Video Games	R	69.17	62.00	71.83	53.00	79.17	68.17	67.00	58.17	66.00
Average Review		*70.50*	*63.00*	*66.17*	*58.92*	*80.25*	*69.63*	*67.50*	*54.71*	*67.17*

5.7 Extending GASA (CA-GASA)

When determining the classification for an unknown word w, the *GASA* algorithm does not take into consideration the words before and after w, i.e. it is context independent. This ignores the fact that many words have different meanings — with different sentiments. How can we begin to allow for multiple, context-dependent sentiments? We propose to allocate a context-dependent classification to an unknown word w; an approach we call *Context Aware GASA (CA-GASA)*. In order to achieve this, several modifications to *GASA* are required. The primary modification lies within the representation of the chromosomes.

Each gene contains two principle parts, the context classification and the context-free classification. When a word in an instance of data is evaluated the classification-value pair is obtained from either the context classification or context-free classification. Two lists of words are used in order to make this decision, namely $list_{next}$ and $list_{previous}$. When a *CA-GASA* chromosome is evaluated on an instance of data i on a word w, the context classification-value pair is allocated if the word w is surrounded by the words in $list_{next}$ and $list_{previous}$. If this is not the case, then the context-free classification-value pair is allocated. This process is further discussed below. Figure 5.4 illustrates an example of a *CA-GASA* chromosome.

In order to enable multiple classification-value pairs to be associated with a word, a new gene encoding is used. For a word w, each gene has the following properties:

- The word w.
- The context rule which is defined as follows:
  - The maximum possible size of the next context words, denoted as $next_{size}$.
  - The maximum possible size of the previous context words, denoted as $previous_{size}$.
  - The list of the next context words, denoted as $list_{next}$.
  - The list of the previous context words, denoted as $list_{previous}$.
  - The number of words to look ahead and compare with $list_{next}$, denoted as $number_{ahead}$.
  - The number of words to look behind and compare with $list_{previous}$, denoted as $number_{behind}$.

  - The context classification-value pair.

- The context-free classification-value pair.

5.7.1 CA-GASA chromosome evaluation

Assume that a *CA-GASA* chromosome c is being applied to an instance of data. A word w in the sentence is evaluated as follows. Starting from the word w, look at the next $number_{ahead}$ words from w and add them to the $list_x$. Once again, starting from the word w, look at the previous $number_{behind}$ words from w and add them to the $list_y$.

Let the size of $list_x$ be denoted as $size_x$, and let the size of $list_y$ be denoted as $size_y$. Let the number of words in the intersection between $list_x$ and $list_{next}$ be denoted by a, and let the number of words in the intersection between $list_y$ and $list_{previous}$ be denoted by b.

If $\frac{a+b}{size_x+size_y} \geq 0.5$, then the word w is classified by the context classification value. Otherwise, the word w is classified by the context-free classification value.

The *CA-GASA* chromosome in figure 5.4 has one next context word, "ship", and one previous context word, "book". Assume the sentence "the ship sunk" is being evaluated, and the classification-value for the word "sunk" is being determined. In this case $list_x$ is empty, and $list_y$ = {ship, the}. Consequently, $size_x = 0$ and $size_y = 2$. The intersection between $list_x$ and $list_{next}$ is empty, and the intersection between $list_y$ and $list_{previous}$ is {ship}, and thus, $a = 0$ and $b = 1$. The word "sunk" is classified by the context classification-value since $\frac{0+1}{0+2} \geq 0.5$, i.e. "sunk" is classified as a sentiment word with a value of -1.0 (since the context classification-value pair in the figure is a sentiment with value of -1.0).

5.7.2 CA-GASA Results

From table 5.8, when HuLiu-6786 was seeded into the proposed methods, it is observed that *GASA* outperformed *CA-GASA* on 2 summary data sets, and tied in the other data sets, whereas *CA-GASA* outperformed *GASA* on 3 of the review data sets. One drawback of *CA-GASA* is that the search space is significantly larger than *GASA* and as a result the training time is much longer. Given this drawback, *CA-GASA* was not tested against the other sentiment analysis algorithms.

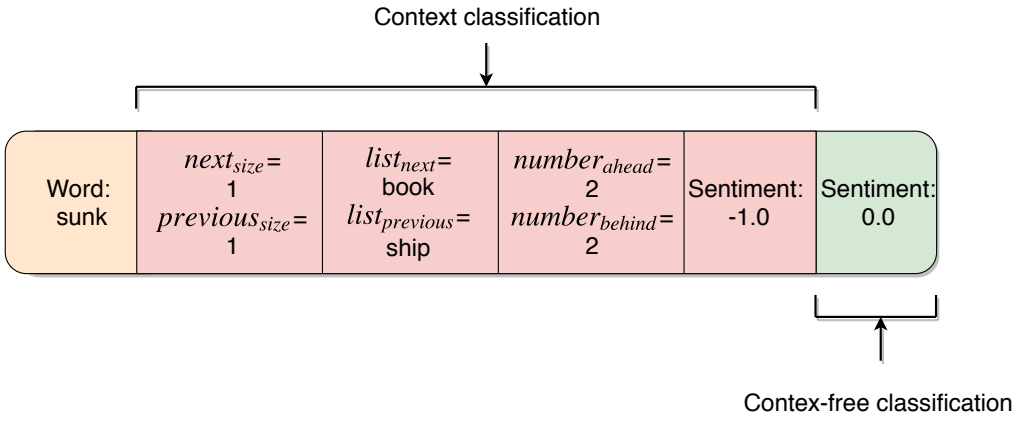

FIGURE 5.4: Example of a *CA-GASA* chromosome. The genes corresponding to context classification are denoted in red. The context-free classification gene (in green) is applied when the context classification has not found a match. In this example the chromosome will predict a negative sentiment if the word 'ship' is found before the word 'sunk'. In the case where none of the words in the context list are found then the context-free classification is returned, in this case, a neutral sentiment.

TABLE 5.8: Test accuracy (%) on the summary and review data showing a comparison between GASA and CA-GASA. Ten-fold cross-validation was used, and all of the data sets had a size of 2000. Both GASA and CA-GASA were seeded with the HuLiu-6786 sentiment dictionary.

	Summary Data		**Review Data**	
Data set	**GASA**	**CA-GASA**	**GASA**	**CA-GASA**
Cellphone	73.6	71.8	69.1	70.2
Office Prod	71.8	70.7	71.0	68.2
Foods	65.8	65.8	67.2	67.4
Video Games	69.4	69.4	65.5	67.0

In order to address the large training time, it would be of interest to determine if an approach could be proposed in order to find which words in some data set have more than one meaning, and to create the context classification for those words only. This would reduce the complexity of *CA-GASA* whilst retaining the ability to perform word disambiguation. Unknown words which only express one sentiment regardless of the context could be represented using *GASA*, and words which have more than one sentiment could use the *CA-GASA* representation.

5.7.3 Text Correctly Classified by *GASA*

This section presents a random sample of the reviews which were correctly classified by *GASA*. The reviews were obtained from the Video Game data set. The reviews were lemmatized (reducing words to their base form, for

example: 'walking' becomes 'walk') using Stanford CoreNLP [162]. The sentence "we thought this movie was quite entertaining" is lemmatized as follows: "we think this movie be quite entertaining". The results are compared to the following four sentiment analysis entities: NLTK, Alchemy API, SentiStrength and MeaningCloud. The reviews are as follows:

Text: *"have purchase the original when release several year ago, and thoroughly enjoy it, I be excite to see a new version out for 2006. it live up to the challenge I expect from this game. it be difficult to find truly challenging puzzle game, and you will not be disappoint with this. my only disappoint come after solve it as the original provide a video of the programmer, although this version do offer the opportunity to replay and select from several final outcome."*

Result: Correctly classified by *GASA* as positive.

Text: *"have not finish it yet, but it sure be a lot of fun! yes, there be/will be even better game. and yes, maybe GTA V be better (it have at least three time more budget so no surprise there). but Watch Dogs be truly a great game! some have unrealistically high expectation or be real hater and should just stick to buy gta every five year instead of buy each game and then give they bad review over and over... before write another obvious review: yes, WE all KNOW that "gta5" be NOT "watch dog" "call of duty" be NOT "candy crush" and "Far Cry 3" be NOT "Tetris""*

Result: Correctly classified by *GASA* as positive.

Text: *"I recommend this game. very little communication loss or problem. if problem occur they be very good as about replace what you lose. the game be fun to play."*

Result: Correctly classified by *GASA* as positive.

Text: *"I buy this game because I remember how Madden use to be. I have hear the gameplay be break (it be, horribly), but I think, hey, if I get use to the stupid game glitch at least it will look good. no. wow be I wrong. everything about this game be atrocious . the only thing that look crisp in this game be the score, which incidentally be the only reason I know it be display in hd. every time a player score a touchdown he literally run through the stand and disappear into the mesh. I could go on, but just think that I pay more than $ 5 for this game be make I extremely angry right no."*

Result: Correctly classified by *GASA* as negative.

Text: *"I play game on a desktop pc in my office, and on a laptop in the living room. I also frequently rebuild my machine and upgrade hardware. I be interested in the online community that will follow this game, however I will not support such*

strict measure in copy protection. I have cancel my pre-order and may purchase later once this limited activation process be remove."

Result: Correctly classified by *GASA* as negative.

5.7.4 Text Incorrectly Classified by *GASA*

This section presents a random sample of the reviews which were incorrectly classified by *GASA*. Refer to the previous subsection for details regarding the data, lemmatization and the algorithms used for comparison. The reviews are as follows:

Text: "*I like final Fantasy 13 but it just do not draw I in that well and the story be slightly confusing. with that be say I absolutely love final Fantasy 13-2! the story be weird because of time travel, but I think it be awesome. they seem to have fix everything about the first one. I also like the fact that this game have multiple ending and new game plus! buy it!*"

Result: The correct classification for the review is positive. *GASA* classified it as negative. Three of the algorithms correctly classified the text as positive, and one algorithm classified it as negative.

Text: "*outstanding shooter game. it be set in world war two and you better not run out in the open. good graphic and story.*"

Result: The correct classification for the review is positive. *GASA* classified it as negative. Only one algorithm correctly classified the text. Two algorithms classified it as neutral and the remaining algorithm classified it as negative.

Text:"*they sit a little bit loose on the controller's peg, so there be some extra play. not shabby, not stellar. put some more life into a controller that you think you be go to lose.*"

Result: The correct classification for the review is positive. *GASA* classified it as negative. All of the algorithms incorrectly classified the text as negataive.

Text: "*this headset rock, that be all you need to know. unless you need more, here be some pro's and con's. pro's 1. affordable price2. long cable3. lightweight design4. adjustable microphone5. Chat Boost / Independent Game and Chat Sound6. stereo ExpanderCon's 1. slightly complicated set - up2. cheap ear Cushions The hiss sound people complain about be negligible, it disappear after just a few minute of use and be otherwise drown out by game. amazing quality and durability at a affordable price, highly recommend. UPDATE : 2/18/11 the microphone serve I very well for only 17 day before the Right Earphone cease to function. a day later, the right earphone*

completely fall off. I get off recommend the product initially, but the fact that the headset can stop work so quickly be terrible. I be return these for a refund and be go to try another headset. I no longer recommend these headphone."

Result: The correct classification for the review is negative. *GASA* classified it as positive. Three of the algorithms correctly classified the text as negative whereas the other classified it as neutral.

Text:"*Brunswick pro Bowling be not worth much. game be slow, unnatural, and poor scripting. I can not believe someone can not make a better program then the Wii bowling which be do pretty good. do not waste you money. I want sport game that actually feel and act like real sport. how hard can that be?*"

Result: The correct classification for the review is negative. *GASA* classified it as positive. Three of the algorithms correctly classified the text as negative whereas the other classified it as positive.

5.8 Conclusion

Being able to determine the sentiment of text is a useful ability to businesses and other entities in order to gain an understanding of people's opinions on their products and services. This study proposed a GA approach in order to classify the sentiment of sentences by optimizing unknown words as either a sentiment or an amplifier word. A way to represent the sentiments and amplifiers through the use of simple mathematical expressions in order to evaluate the final sentiment of sentences is proposed. The experimental results revealed that *GASA* was able to outperform certain commercial APIs.

One advantage of *GASA* is that the algorithm can grow a sentiment dictionary (by using the classification-value pairs for the words which were optimised during training) which can be reused or further improved upon. The experiments suggested that if a particular word appeared a large number of times in the training data set, then the proposed method is likely to correctly classify its sentiment.

We also proposed *CA-GASA*, and the rationale behind this modification was to provide the ability to allocate a sentiment based on the context for which the words are in. This method requires additional work in order to reduce the complex search space. It would also be of interest to investigate if an ensemble of *GASA* chromosomes could outperform the accuracy of a single one. In the next chapter we extend our idea by investigating the effect of sentence compression in the context of sentiment analysis.

Chapter 6

Text Compression for Sentiment Analysis via Evolutionary Algorithms

6.1 Introduction and Rationale

Can textual data be compressed intelligently without losing accuracy in evaluating sentiment? To illustrate the idea, consider the sentence “I went home yesterday, opened the door and was immediately faced with a terrible shock". The key sentiment-encoding phrase is “terrible shock". We thus want to propose an algorithm that can learn to evolve rules that can effectively do this compression. The remaining words in the sentence do not contain any useful information in determining the sentiment expressed, and thus the rules can discard those words. Reducing the amount of text needed to determine the sentiment and reduce the processing overhead when sending text to remote servers for analysis.

In this chapter, we propose a novel EA which is tasked with compressing text, whereby the objective is to retain the correct sentiment classification of the compressed text. To achieve this, the EA makes use of *Parts of Speech* (POS) to determine which POS can be removed from the text without hindering the classification performance.

Feng *et al.* [77] proposed a Chinese text compression method with the objectives of preserving the sentiment of opinions expressed in the text and ensuring that the compressed sentence is grammatically correct. The authors proposed a score function with three primary functions: a word significance, a linguistic and an opinion scoring function. The score function was applied to each candidate sentence and the result was divided by the number of

words in the compressed sentence. The compressed sentence with the highest value after these operations was deemed to be the final compressed sentence. One Chinese data set of compressed sentences was manually created to evaluate their proposed approach. The study revealed that their proposed approach was able to outperform a traditional sentence compression method when the sentences were evaluated - in terms of the opinion and grammar - by a human. A compression rate of 47% was achieved.

Che *et al.* [31] also proposed a text compression method for sentiment analysis. In their study, they make use of 10 features (which are determined for every word) to perform the compression. The features are grouped into basic, sentiment related, semantic and syntactic features. The basic features include the words before and after the word for which the features are being constructed for; they also include the POS tags for those words. There are two sentiment features, namely a binary field denoting if the word is a perception word and if it is a polarity word. These words were obtained from an existing lexicon. The semantic features include prefix and suffix characters and also Brown word features which helps identify that words which are written differently represent the same concept. They also include word embedding which makes use of Word2Vec [174] to determine if two words have a similar meaning. The syntactic feature makes use of a single feature which indicates the relationship between the words in the sentence. The sentence compression is conducted prior to the sentiment analysis. The results reveal that the F-score was improved from 64.70% (no compression) to 67.49% (with compression).

With the limited amount of existing work in this field, we propose and investigate *PARts-of-Speech for sEntiment Compression* (*PARSEC*), which makes use of an EA to achieve text compression with minimal loss in sentiment accuracy. Section 6.2 introduces and presents examples of POS. Next, section 6.3 discusses the proposed *PARSEC* method. The experimental setup is presented in section 6.4 and section 6.5 reveals the results. Experiments on fixed compression rates were conducted and these are discussed in section 6.6. Finally, section 6.7 concludes this chapter.

6.2 Parts-of-Speech

POS, in languages, are defined as the primary groups of words that are grammatically similar. Common POS include, but are not limited to: adjectives, verbs and nouns.

TABLE 6.1: Common POS along with their tag, description and examples.

POS	Description	Examples
DT	Determiner	a, the
JJ	Adjective	wonderful, massive
NN	Noun, singular	house, phone
NNS	Noun, plural	cities, boxes
RB	Adverb	very, rather
TO	To	to
VB	Verb	explain, be
VBD	Verb, past tense	involved, skipped

In this study, the Stanford Log-linear Part-Of-Speech Tagger [263] was used to convert the original data sets into their corresponding POS tags. Thus, each word in a dataset was converted into a POS tag, and the length of each sentence in the original dataset was identical to that of the POS tagged dataset. For example, if the following sentence was found in the original dataset: "this is a great product", it would be converted into "DT VBZ DT JJ NN" (which corresponds to the POS equivalent) in the POS tagged dataset.

We used the Penn Treebank tag set [163, 256]. This set contains 36 tags and an additional 12 tags to denote certain characters such as comma, semi-colon and other characters. Tabe 6.1 presents common POS tags, descriptions and examples.

6.3 Proposed Compression Method

6.3.1 Compressor

This study proposes a new type of individual for EAs – referred to as a *compressor* – which reduces the length of sentences when applied to a dataset. The compressor is made up of several rules, and in turn, each rule is made up of a sequence of POS tags and a decision. The decision is applied whenever its sequence of POS tags matches the POS tags in the data. The decisions represent the indices of the words which will be removed in the matched sequence. Figure 6.1 illustrates an example of a compressor.

In the following explanation, let X denote some original dataset for which *PARSEC* is being applied to, and let Y denote the equivalent POS dataset. Furthermore, let $X_{i,j}$ denote sentence i and word j in X, and $Y_{i,j}$ denote sentence i and word j in Y.

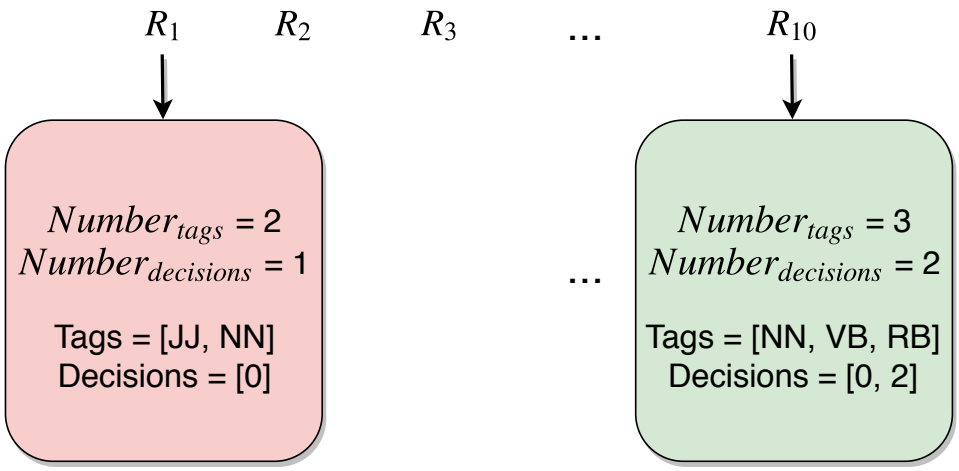

FIGURE 6.1: An example of a PARSEC compressor with 10 rules. The first and last rules are shown. Each rule encodes a number of POS tags and decisions. The tags denote the POS tags for which each rule will try and match. Finally, the decisions encode the word index which will be deleted. Rule 1 will delete any word which is at index 0 each time the rule matches. Rule 10 will delete words at index 0 and 2.

The compressor in figure 6.1 has 10 rules denoted as R_1, R_2, R_3, ..., R_{10}. The first rule, R_1, has 2 tags which are represented by the sequence "JJ, NN". This rule has one decision, and that is, to delete the word at index 0. This implies that when this particular compressor is applied, rule R_1 will find POS tags in Y corresponding to the sequence "JJ NN".

Let $Y_{l,m}$ and $Y_{l,m+1}$ denote two consecutive words in Y which have the sequence "JJ NN". The decision in R_1 is to delete index 0, thus R_1 will delete $Y_{l,m}$ and consequently it will delete $X_{l,m}$. Thus, the size of X and Y has been reduced by one. Each time R_1 finds the exact sequence "JJ NN", the POS tag and word corresponding to "JJ" (index 0) in Y and X are deleted.

Similarly for R_{10}, let $Y_{l,m}$, $Y_{l,m+1}$ and $Y_{l,m+2}$ denote three consecutive words in Y which have the sequence "NN VB RB". The decision in R_{10} is to delete index 0 and 2, thus R_{10} will delete $Y_{l,m}$, $Y_{l,m+2}$, $X_{l,m}$ and $X_{l,m+2}$ from Y and X respectively for each occurrence of the POS tags "NN VB RB".

When a rule is being processed, if an end of sentence punctuation mark ("?", "!", ".") occurs, then the rule evaluation immediately stops. This mechanism was incorporated so that the sequence of tags – which is encapsulated by a rule – is applied to a single sentence and is not applied across two separate sentences. For example, consider the two following sentences "This book was really funny. Great, tomorrow is Friday." which corresponds to: "DT NN VBD RB JJ. JJ, NN VBZ NNP." Assume that some rule has the following tags "JJ JJ" and the decision is to delete both indices. If the punctuation marks are not taken into consideration, then the rule will match with

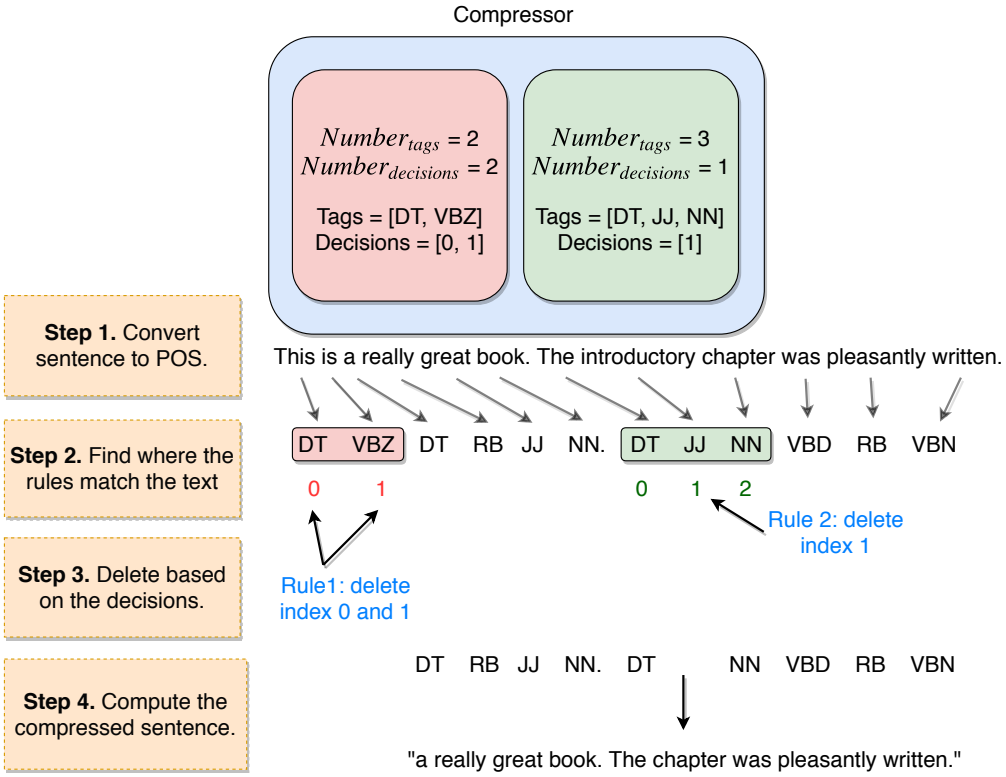

FIGURE 6.2: Illustrating how PARSEC compresses text. In the example the compressor has two rules. The original sentence is displayed as "This is a really great book. The introductory chapter was pleasantly written." The first rule matches with DT and VBZ. The second rule matches with DT, JJ and NN. The first rule states that both indices 0 and 1 must be deleted. The second rule states that only words at index 1 must be deleted. The figure presents the compressed POS sequence along with the compressed sentence.

the words "funny" and "great". This will result in "This book was really . , tomorrow is Friday." Consequently, the sentiments in these two sentences are no longer positive. However, if punctuation is taken into consideration, then the rule will not delete any words since the full stop separates the two sentences.

In this study, we define the term 'wildcard', which represents the notion of 'any POS' tag and is denoted with a '*' symbol. Thus, when a rule is being evaluated, the wildcard POS can match with any POS in the POS dataset. For example, if a rule has the following tags: "JJ * NNP", then any POS sequence that has 'JJ', followed by any POS, followed by 'NNP' will cause the rule to match with that POS sequence.

Algorithm 10 presents the generalised pseudocode to match the tags in a rule with the POS tags in the POS dataset. The algorithm also presents how to delete words from the original and POS dataset once a rule matches with a sequence. Figure 6.2 illustrates how the compression takes place.

Algorithm 10: Pseudocode to apply a compressor to a sentence and compress the text.

```
   input: compressor the compressor to evaluate
   input: original_sentence the original sentence
   input: POS_sentence the POS sentence
   input: length_sentence the length of the sentence to evaluate
 1 begin
 2     for each rule in compressor do
 3         tag_in_rule ← the tags inside rule
 4         rule_decisions ← the decisions inside rule
 5         for i ← 0 to length_sentence do
 6             match ← true
 7             for j ← 0 to length(tag_in_rule) do
 8                 if tag_in_rule [j] not equal POS_sentence [i + j] then
 9                     match ← false
10                     break loop and go to line 6, set i ← i + 1
11             if match equal true then
12                 for k ← 0 to length(rule_decisions) do
13                     flag original_sentence [i + rule_decisions[k]]
14         compressed_sentence ← delete all words in original_sentence
            that have been flagged.
15 return compressed_sentence
```

6.3.2 *PARSEC* initial population generation

The pseudocode for creating an initial population of compressors is presented in algorithm 11. Several individuals are created based on a predefined user parameter, namely the population size. Several rules have to be created for each compressor. There must be at least one rule, and a user-defined parameter, namely $\max_{rules}$, is implemented to restrict the total number of rules. When creating the rules, the algorithm iterates until it has reached $\max_{rules}$, and a probability value of 0.5 was set to determine if a rule was to be created or not, as can be seen on line 9 in algorithm 11. Each rule consists of two primary parts, namely the tags and the decisions, the pseudocode for the creation of those respective parts are presented in algorithms 12 and 13. Algorithm 12 shows that wildcards are not added to the start or end of the compressor tag rule sequence.

Algorithm 11: Pseudocode to create the initial population of compressors.

```
   input: population_size the number of compressors to create
1  begin
2      for i ← 0 to population_size do
3          CreateCompressor()
4          Evaluate compressor.
5          Add compressor to the initial population.
6      Function CreateCompressor ()
7          new_compressor ← create a blank compressor
8          for j ← 0 to max_tags do
9              if rand() < 0.5 then
10                 CreateRule() and add rule to new_compressor
11         if Compressor has no rule then
12             CreateRule() and add rule to new_compressor
13         return new_compressor
14     Function CreateRule ()
15         new_rule ← create a blank rule
16         Create the tags using algorithm 12 and add the rules to
             new_rule.
17         Create the decisions using algorithm 13 and add the rules to
             new_rule.
18 return The initial population.
```

Algorithm 12: Pseudocode to create POS tags.

```
   input: minimum_tags the minimum number of allowed pos tags
   input: maximum_tags the maximum number of allowed pos tags
 1 begin
 2     pos_list ← { }
 3     number_tags ← random[minimum_tags, maximum_tags ]
 4     for i ← 0 to number_tags do
 5         if i = 0 then
 6             Create a tag (excluding wildcard) and add it to pos_list
 7         else if i = number_tags −1 then
 8             Create a tag (excluding wildcard) and add it to pos_list
 9         else
10             Create a tag (including wildcard) and add it to pos_list
11 return pos_list
```

6.3.3 *PARSEC* fitness function

The pseudocode for the fitness evaluation of a compressor is presented in algorithm 15. Each example (an example can be made up of several sentences) in the uncompressed dataset is considered in turn. The sentiment for each example in the uncompressed dataset is computed using algorithm 14. The compressor is applied to the uncompressed sentence and a compressed example is created. The sentiment of the compressed example is computed using algorithm 14. The compressed sentiment is then compared to the uncompressed sentiment.

Algorithm 13: Pseudocode to create decisions.

```
   input: number_tags the number of tags
 1 begin
 2     for i ← 0 to number_tags do
 3         if random[0,1] < 0.5 then
 4             Add i to decision_list
 5 return decision_list.
```

If these are the same then the compressed example has had no effect on the sentiment when compared to the uncompressed one. Conversely, if the values are different, then there are two possibilities. Either the compressed sentiment is equal to the correct sentiment as labelled in the training data (in which case the compressor is rewarded) or it is not (in which case the compressor is penalised). Thus, the fitness function takes into consideration

the number of examples for which the compressor produces the correct sentiment prediction.

Algorithm 14: Pseudocode for baseline evaluation.

```
   input : sentence the sentence to be evaluated
   input : dictionary the dictionary of sentiment words
   input : negation_words the list of negation words
   output: The sentiment for the evaluated sentence
 1 begin
 2     sentiment_score ← 0
 3     negation ← false
 4     for each word in the sentence do
 5         if word is in dictionary and negation is true then
 6             sentiment_score ← sentiment_score + (-1 × word's sentiment
                value)
 7         if word is in dictionary and negation is false then
 8             sentiment_score ← sentiment_score + word's sentiment value
 9         if word is in negation_words then
10             negation ← swap polarity
11         else
12             negation ← false
13     return sentiment_score
```

A multi-objective fitness function was created to combine the fitness, the average reduction in terms of the length of the sentences before and after compression, and the size of the compressor in terms of the number of rules. Thus, the objective was to maximize the fitness and average reduction in sentences, and to minimize the size of the compressor.

6.4 Experimental Setup

6.4.1 Datasets and Setup

Twelve data sets were created by randomly selecting reviews from corresponding Amazon review datasets [141]. For each created data set, 1000 positive and 1000 negative reviews were randomly selected from the larger corresponding Amazon data set. The larger Amazon data sets were obtained from [141, 168]. The data sets were: Amazon Instant Video (AIV), Apps for Android, Automotive, Baby, Beauty, Digital Music, Health and Personal Care, Musical Instruments (MI), Patio Lawn and Garden, Pet Supplies, Tools and

Algorithm 15: Pseudocode to compute fitness of a compressor.

```
input : compressor the compressor to evaluated
input : sentence a list of sentences for which compressor will be
        evaluated on
input : original_lengths a list of lengths for each sentence in the
        original data set
input : original_sentiments a list of sententiment values determined
        by applying algorithm 14 on the original data set
input : correct_sentiments a list of correct sententiment values for the
        original data set
input : number_rules the number of rules that compressor has
output: The sentiment for the evaluated sentence
1  begin
2      for i ← 0 to number_of_sentences do
3          compressed_sentence ← apply compression to sentences [i].
4          compressed_sentiment ← apply algorithm 14.
5          if compressed_sentiment is not equal to original_sentiments [i]
            then
6              if compressed_sentiment is equal to correct_sentiments [i] then
7                  raw_fitness ← raw_fitness + 1
8              else
9                  raw_fitness ← raw_fitness − 1
10         average_change ← average_change + (Length(sentences [i]) -
            Length(compressed_sentence))
11     raw_fitness ← raw_fitness + [0.5 × (average_change/
        number_of_sentences)] − (0.1 × number_rules)
12 return raw_fitness
```

TABLE 6.2: The lower (LCB) and upper (UCB) compression bound constrain the algorithm by ensuring that the compression rate for each compressor is between the two values respectively. The table presents the LCB and UCB values (%) along with the minimum and maximum number of rules used in the experiments. These set compression rates were enforced and the performance of *PARSEC* was measured on several datasets. The values for the minimum and maximum number of rules were determined by additional trial runs.

LCB	**UCB**	$compressionRules_{min}$	$compressionRules_{max}$
10	13	5	50
15	18	10	70
20	13	20	90
25	28	40	120
30	33	90	150
50	53	350	500

Home Improvement, and Toys and Games. Each example in the Amazon data sets is made up of a short summary, a review and a star rating which were provided by a user. The star rating was used to label the datasets using 2.5 as a threshold between negative and positive classes.

We propose a set of experiments whereby the compression rate is a user parameter and *PARSEC* must generate compressors that can compress the original datasets to reach the specified compression rate.

To achieve this, a lower and upper user-defined compression bound were defined. The lower compression bound (LCB) was implemented to ensure that the compressors did not result in a compression ratio less than the specified value. Similarly, the upper compression bound (UCB) ensured that the compressors did not result in a compression ratio greater than the specified value. Thus, every compressor in the EA population had to have a compression rate between the LCB and UCB. Table 6.2 presents the LCB and UCB used. It was observed that a greater number of rules were needed to achieve higher compression rates. For example, in the case of a compression requirement between 50 to 53 percent, a small number of rules would result in a compression rate smaller than the required range. This is the so because a small number of rules will remove a smaller amount of text.

Furthermore, two additional user-defined parameters were implemented to ensure that the total number of rules within each compressor remained within a certain bound. The two parameters were named $compressionRules_{min}$ and $compressionRules_{max}$. The $compressionRules_{max}$ parameter prevents compressors from having an extremely large number of rules. Both parameters aid in restricting the search space in terms of the

number of rules to create and maintain for each compressor during the evolutionary process. The initial population generation had to respect the LCB and UCB, and respect the minimum and maximum number of rules. The mutation operator also had to respect these constraints.

Crossover and mutation enabled *PARSEC* to exploit and explore the search space. Crossover was implemented by randomly selecting a rule from two parent chromosomes and exchanging them. Mutation was implementing by randomly selecting a rule within a parent chromosome and creating a new rule in its place.

6.4.2 Sentiment analysis algorithms

Several sentiment analysis algorithms were applied to the original and compressed datasets to determine the difference in accuracy and to investigate the performance of *PARSEC*. The following algorithms were used in this study: SentiStrength [260, 259], MeaningCloud [170], Vivek [184], Stanford [239], uClassify [264], Sentiment140 [83], Intellexer [68] and LingPipe [8].

6.4.3 Execution of experiments

We first applied the sentiment analysis algorithms described in section 6.4.2 to the original datasets and recorded the test accuracy. Then, we ran *PARSEC* using the baseline evaluation to create compressor rules. Once the compressors were created, we applied the compressors to the original datasets to generate the compressed data. We applied the sentiment analysis algorithms on the compressed data and recorded the test accuracy. In the following section we report on the change in accuracy between the compressed and original data. *PARSEC* evolved a population of 250 compressors over 100 generations, the crossover rate was 60% and mutation rate was 40%. *PARSEC* was developed in Java 1.8.

6.5 Results and Discussion

Figure 6.3 shows the average change in test accuracy across the different sentiment analysis algorithms based on the compression rates. The general trend is a decrease in performance with the largest decrease in performance observed by Sentiment140. LingPipe and SentiStrength were the only two methods that achieved a positive change in performance (on certain datasets)

TABLE 6.3: Average change in test accuracy (%) across all the data sets when PARSEC models were created for 75% compression rate.

Algorithm	**Change in Test Accuracy**	**Original Accuracy**	**Compressed Accuracy**
LingPipe	-3.2	79.7	76.4
MeaningCloud	-4.2	62.5	58.2
SentiStrength	-6.9	67.8	60.8
uClassify	-6.4	80.7	74.3

on compression rates of 10% and 15% respectively. When experimenting with *PARSEC* we posed the following question. For a predetermined drop in accuracy, how much compression is achievable by *PARSEC*? For a threshold of -1% change in accuracy, Vivek and LingPipe were able to achieve up to 30% compression. For this same threshold five methods could achieve up to 20% compression rate. Regardless of the compression rate used, both Stanford and Sentiment140 did not obtain any result less than -1%. These two algorithms produced the weakest performance with *PARSEC*.

For a threshold of -2%, five algorithms could achieve up to 30% compression rate and LingPipe was able to achieve up to 50% compression. When the threshold was set to -3% four methods could achieve up to 50% compression.

We additionally wanted to determine the change in test accuracy when a much larger compression rate was used. Table 6.3 presents the results when a compression rate of 75% was used. Only certain algorithms were tested due to our limited access to the sentiment analysis algorithms. These results reveal that a large compression rate is achievable without a great loss in test accuracy: LingPipe obtained a loss of -3.3% accuracy.

An example of *PARSEC* compression is as follows. The original sentence (which contains 56 words): *"this was a wonderful movie involving fascinating people who were such great actors and actresses. the events were interesting and I never got tired of watching the episodes. I was so sad to see that they will not go to keep going with the life of all these inhabitants of Lark rise and candleford. Mrs. Hamlin"*. The compressed sentence (22 words): *"this was wonderful movie involving fascinating people who actresses. got tired. so sad to see not keep going rise candleford. Mrs. Hamlin"*.

Figure 6.4 highlights are four rules which were randomly selected from a *PARSEC* model to illustrate some of the evolved rules. This model had a total of 38 rules. The POS tags are presented in order of appearance between square brackets. The indices start from zero. Rule 1 denotes that if the POS "NNP" is followed by "JJ", then delete both words. Rule 4 denotes that if

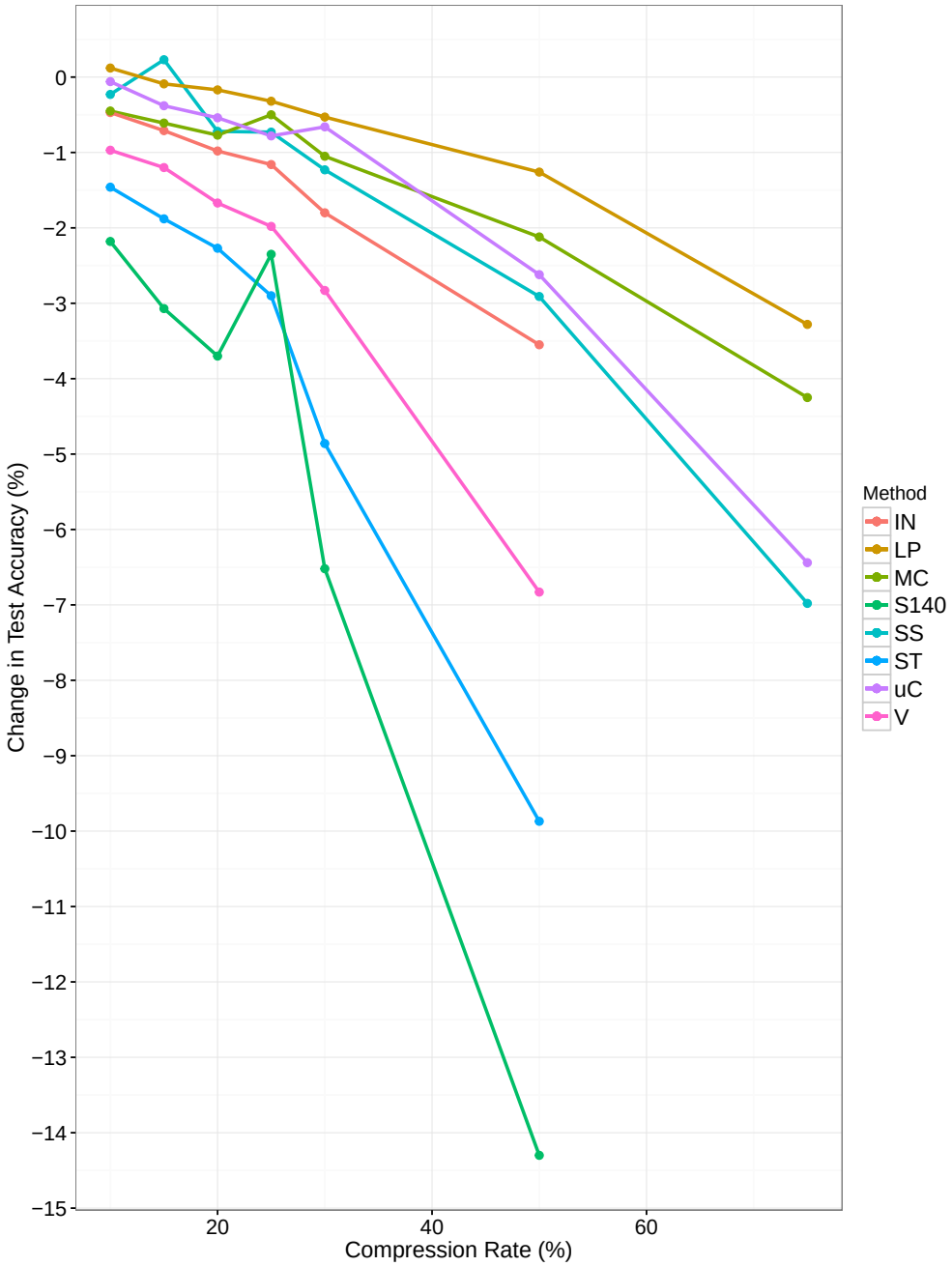

FIGURE 6.3: Change in average test accuracy across the different sentiment analysis algorithms for different text compression rates enforced by PARSEC, averaged across all the data sets. A higher positive value indicates a better result. The various sentiment analysis algorithms are denoted as follows: *IN*, *LP*, *MC*, *S140*, *SS*, *ST*, *uC* and *V* refer to Intellexer, LingPipe, MeaningCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. The accuracy obtained by LingPipe outperformed all the other methods. Furthermore, LingPipe obtained an average positive change in accuracy of 0.1% for 10% compression. At a compression rate of 50%, LingPipe obtained an change in accuracy of -1.2% averaged over all the datasets, illustrating that it works well with PARSEC.

"IN" is followed by any POS and then followed by "VB", then delete the word at index 1.

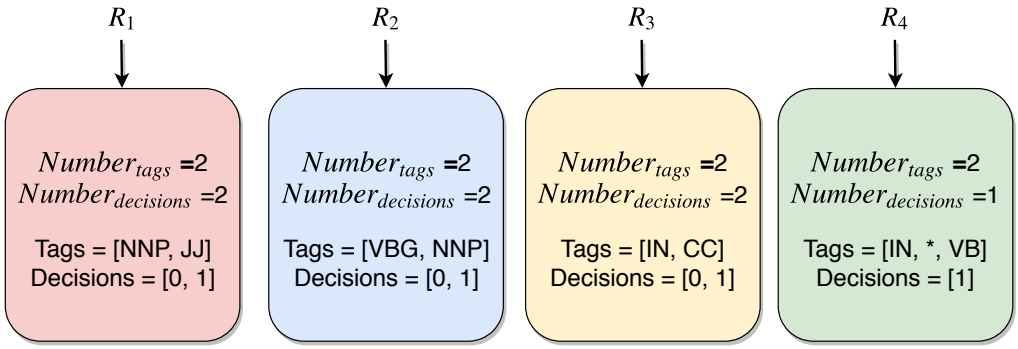

FIGURE 6.4: Illustrating four rules which were extracted from a PARSEC compressor.

6.6 Detailed results for fixed compression rate experiments

Tables 6.4 to 6.9 present the difference in test accuracy results for the various compression rates. The various sentiment analysis algorithms are abbreviated as follows: *IN, LP, MC, S140, SS, ST, uC* and *V* refer to Intellexer, LingPipe, MeainingCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. For a compression rate of 10%, the best and worst change in accuracy were obtained by LingPipe and Sentiment140 respectively. Based on all the findings for 10 per cent compression, there were 21 cases across all the methods and data sets whereby the change in accuracy was greater than zero. For a compression rate of 15% the number of cases was 17 whereby LingPipe obtained the best average change in accuracy with a value of 0.0%.

The best change in accuracy for 20% compression rate was obtained by uClassify with a value of 0.7 on two data sets. There were six data sets for which uClassify reduced the data up to 20% and for which the test accuracy on the compressed data was better than on the original data. Additionally, there were 14 cases in total across all the algorithms whereby the change in accuracy was greater than zero for 20% compression.

The findings revealed that for the compression rate of 25% there were 14 cases for which the change was greater than zero, and the best change in average accuracy was obtained once again by LingPipe with a value of -0.3%. There was a reduction performance in terms of the number of cases having a change greater than zero when a compression of 30% was used;

notably 9 cases. uClassify however, on 5 datasets, had a test accuracy greater than zero. Finally, only two algorithms were able to obtain improvements in terms of the changes for a compression rate of 50%, namely uClassify and LingPipe, with changes in 3 and 2 data sets respectively.

6.7 Conclusion

This study empirically demonstrates that machine learning can reduce the amount of data needed to determine the sentiments within sentences with little loss in accuracy. To achieve this, we proposed an evolutionary algorithm, *PARSEC*, which evolves a population of chromosomes which encode rules for compression based on the removal of POS.

We studied the test accuracies achieved by eight sentiment analyis algorithms when set compression rate thresholds were enforced. Two algorithms showed marginal improvements in accuracy on several data sets for a compression rate of 15%. Additionally, there were 9 data sets whereby some of the algorithms were able to yield an improvement in accuracy for a compression rate of 30%. At a threshold of -1% sentiment accuracy loss, two methods were able to achieve up to 30% compression, five methods up to 20% compression, and six methods up to 10% compression. The best performing algorithm, LingPipe, showed only modest losses in accuracy even at high compression: 3.3% for a four-fold data compression.

It would be of interest to incorporate the *PARSEC* algorithm directly into a sentiment analysis algorithm instead of via a simple dictionary approach. One could achieve this by replacing the fitness function used in the evolutionary phase with a more sophisticated sentiment analysis algorithm. This would allow the emergence of an algorithm where the compression and sentiment analysis are mutually optimised for best performance.

	IN	**LP**	**MC**	**S140**	**SS**	**ST**	**uC**	**V**
AIV	-0.9	**0.0**	**0.4**	-2.1	-0.3	-3.0	-1.9	-0.9
MI	**0.1**	**0.3**	-0.7	-1.9	-0.4	**0.5**	-0.4	-1.4
Digital Music	-1.0	**0.2**	-0.3	-2.5	-0.1	-1.9	-0.4	-2.0
Baby	**0.0**	**0.4**	-0.2	-2.7	**0.0**	-1.2	**0.3**	**0.2**
Patio & Garden	-0.1	**0.4**	-0.2	-2.1	-0.2	-0.7	**0.9**	-0.9
Automotive	-1.1	**0.4**	-0.5	-2.1	**0.0**	-1.1	**0.7**	-0.9
Pet Supplies	-1.2	**0.3**	-0.4	-2.9	-0.4	**0.3**	**0.0**	-1.6
Apps for Android	-1.4	**0.0**	-0.5	-2.5	-0.3	-3.5	**0.0**	-1.6
Beauty	-0.5	**0.0**	**0.2**	-1.2	**0.0**	-0.5	-0.7	-0.4
Tools & Home	**0.8**	-0.1	-1.0	-2.3	-0.6	-1.7	**0.5**	**0.0**
Toys & Games	-0.5	**0.0**	-1.5	-1.8	**0.2**	-3.1	-0.1	-1.4
Health & Personal	**0.4**	**0.1**	-0.4	-1.6	-0.3	-1.4	**0.5**	-0.4
Average	-0.4	**0.1**	-0.4	-2.1	-0.2	-1.4	**0.0**	-0.9

TABLE 6.4: Percentage difference between the compressed and original test accuracy when a compression rate between 10% and 13% was imposed. The column titles *IN*, *LP*, *MC*, *S140*, *SS*, *ST*, *uC* and *V* refer to Intellexer, LingPipe, MeaningCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. Percentage differences greater than or equal to zero are highlighted in bold.

	IN	**LP**	**MC**	**S140**	**SS**	**ST**	**uC**	**V**
AIV	-1.1	-1.1	-0.2	-2.7	**0.1**	-3.2	-2.0	-0.6
MI	**1.1**	**0.5**	-0.8	-2.9	-0.8	**0.0**	-1.3	-0.8
Digital Music	-1.6	**0.4**	-0.7	-3.6	-0.5	-2.3	-1.2	-2.1
Baby	**0.0**	-0.9	-0.5	-2.4	**0.2**	-2.0	**0.6**	-0.1
Patio & Garden	-0.3	-0.4	-0.3	-3.5	-0.6	-1.2	**0.5**	-1.0
Automotive	-1.4	**0.6**	-1.0	-2.8	-0.3	-2.2	**0.9**	-0.8
Pet Supplies	-1.1	**0.6**	-0.2	-3.8	-0.6	-1.3	**0.2**	-2.9
Apps for Android	-1.7	-0.3	-0.9	-4.2	-0.5	-3.4	-1.0	-3.2
Beauty	-0.9	-0.3	-0.7	-3.5	-0.2	-1.3	-1.8	-1.2
Tools & Home	-0.2	**0.0**	**0.0**	-3.0	-0.3	-1.7	1.2	**0.7**
Toys & Games	-1.3	-0.2	-1.2	-1.6	-0.1	-1.1	-0.8	-1.8
Health & Personal	**0.1**	**0.0**	-0.6	-2.8	-0.5	-2.6	**0.2**	-0.4
Average	-0.7	**0.0**	-0.6	-3.0	-0.3	-1.8	-0.3	-1.2

TABLE 6.5: Percentage difference between the compressed and original test accuracy when a compression rate between 15% and 18% was imposed. The column titles *IN*, *LP*, *MC*, *S140*, *SS*, *ST*, *uC* and *V* refer to Intellexer, LingPipe, MeaningCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. Percentage differences greater than or equal to zero are highlighted in bold.

	IN	**LP**	**MC**	**S140**	**SS**	**ST**	**uC**	**V**
AIV	-1.7	-1.2	**0.3**	-4.0	-0.4	-3.1	-1.8	-1.8
MI	**0.2**	**1.1**	-1.1	-2.5	-0.6	**0.4**	-1.2	-1.4
Digital Music	-2.2	-0.4	-0.2	-3.5	-1.2	-3.1	-1.7	-2.0
Baby	-0.7	-0.4	-0.4	-2.7	-0.8	-2.2	**0.5**	-1.1
Patio & Garden	-0.9	-1.1	-1.2	-3.7	-0.5	-2.2	**0.5**	-0.9
Automotive	-1.6	**0.2**	-0.3	-3.5	-0.9	-2.8	**0.3**	-1.7
Pet Supplies	-1.0	**0.4**	-1.3	-3.5	-0.4	-2.1	**0.7**	-2.2
Apps for Android	-1.7	-0.1	-1.6	-4.8	-0.4	-4.5	-1.7	-3.3
Beauty	-1.1	**0.2**	-0.6	-4.0	-1.0	-1.1	-1.2	-1.2
Tools & Home	**0.6**	-0.1	-1.2	-4.4	-0.7	-1.9	**0.0**	-1.5
Toys & Games	-1.5	**0.0**	-0.4	-3.3	-0.2	-2.8	-1.5	-1.9
Health & Personal	**0.0**	-0.4	-1.0	-4.2	-1.2	-1.7	**0.7**	-0.9
Average	-0.9	-0.1	-0.7	-3.7	-0.7	-2.2	-0.5	-1.6

TABLE 6.6: Percentage difference between the compressed and original test accuracy when a compression rate between 20% and 23% was imposed. The column titles *IN*, *LP*, *MC*, *S140*, *SS*, *ST*, *uC* and *V* refer to Intellexer, LingPipe, MeaningCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. Percentage differences greater than or equal to zero are highlighted in bold.

	IN	**LP**	**MC**	**S140**	**SS**	**ST**	**uC**	**V**
AIV	-2.4	-1.6	-0.6	-3.7	-0.2	-6.0	-2.5	-2.0
MI	-0.6	**1.0**	-0.1	-4.8	-1.0	-1.6	-0.1	-1.8
Digital Music	-1.9	-0.1	-0.5	**25.1**	-0.5	-2.4	-2.9	-3.5
Baby	**0.1**	-0.9	**0.5**	-4.4	-0.5	-2.6	-0.4	-0.6
Patio & Garden	-0.9	-1.4	**0.1**	-4.7	-0.9	-1.5	**0.6**	-1.6
Automotive	-2.0	**0.1**	**0.1**	-3.3	-1.5	-4.2	**0.5**	-0.4
Pet Supplies	-0.8	**0.0**	-0.7	-5.3	-0.6	-1.7	**0.1**	-2.9
Apps for Android	-2.4	-0.4	-1.4	-6.7	-0.9	-3.4	-0.9	-3.4
Beauty	-1.1	**0.2**	-0.7	-5.5	-0.6	-2.1	-1.8	-2.4
Tools & Home	-0.8	-1.0	-1.6	-5.2	-1.1	-3.6	**0.0**	-0.3
Toys & Games	-1.2	-0.5	-1.1	-5.7	-0.2	-2.7	-1.0	-3.6
Health & Personal	**0.3**	**0.9**	**0.0**	-3.9	-0.3	-2.8	-0.9	-1.1
Average	-1.1	-0.3	-0.5	-2.3	-0.7	-2.9	-0.7	-1.9

TABLE 6.7: Percentage difference between the compressed and original test accuracy when a compression rate between 25% and 28% was imposed. The column titles *IN*, *LP*, *MC*, *S140*, *SS*, *ST*, *uC* and *V* refer to Intellexer, LingPipe, MeaningCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. Percentage differences greater than or equal to zero are highlighted in bold.

	IN	**LP**	**MC**	**S140**	**SS**	**ST**	**uC**	**V**
AIV	-2.3	-2.1	-0.1	-8.2	-1.3	-7.0	-2.3	-2.5
MI	-0.5	**0.6**	-1.1	-6.9	-1.6	-2.4	-1.9	-2.8
Digital Music	-3.3	-1.1	-1.0	-6.2	-1.7	-3.6	-2.8	-4.9
Baby	-0.8	-0.7	**0.2**	-6.1	-1.1	-5.3	**0.7**	-1.3
Patio & Garden	-1.8	-1.2	-1.7	-6.6	-1.5	-4.3	-0.3	-2.2
Automotive	-1.2	-0.4	-0.9	-4.8	-1.5	-4.7	**0.8**	-2.6
Pet Supplies	-2.2	-0.9	-1.1	-5.8	-1.0	-4.4	**1.6**	-2.8
Apps for Android	-3.0	**0.0**	-2.1	-7.7	-1.1	-6.4	-2.1	-5.3
Beauty	-2.4	-0.2	-1.1	-7.7	-1.2	-4.6	-1.9	-3.2
Tools & Home	-0.9	**0.0**	-1.3	-5.9	-1.5	-5.0	**0.4**	-1.1
Toys & Games	-2.2	**0.0**	-1.6	-6.0	-0.2	-5.0	-0.7	-3.0
Health & Personal	-0.6	-0.3	-0.6	-6.1	-0.7	-5.3	**0.7**	-2.0
Average	-1.8	-0.5	-1.0	-6.5	-1.2	-4.8	-0.6	-2.8

TABLE 6.8: Percentage difference between the compressed and original test accuracy when a compression rate between 30% and 33% was imposed. The column titles *IN*, *LP*, *MC*, *S140*, *SS*, *ST*, *uC* and *V* refer to Intellexer, LingPipe, MeaningCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. Percentage differences greater than or equal to zero are highlighted in bold.

	IN	**LP**	**MC**	**S140**	**SS**	**ST**	**uC**	**V**
AIV	-5.8	-4.2	-1.8	-15.6	-3.4	-13.2	-7.8	-8.2
MI	-1.9	**0.8**	-0.9	-15.1	-2.6	-6.9	-3.9	-7.6
Digital Music	-5.5	-1.3	-1.9	-15.3	-3.5	-9.9	-5.5	-9.0
Baby	-1.4	-1.4	-1.7	-12.4	-2.4	-11.2	-1.5	-5.6
Patio & Garden	-3.8	-1.8	-2.1	-13.9	-2.5	-10.1	-0.5	-4.7
Automotive	-3.7	-2.4	-1.9	-12.6	-2.8	-7.9	**0.1**	-4.6
Pet Supplies	-1.8	-1.6	-2.2	-14.9	-2.5	-8.0	**0.6**	-6.2
Apps for Android	-6.2	-1.6	-4.8	-17.1	-4.4	-12.4	-4.3	-11.0
Beauty	-3.0	-0.1	-1.8	-15.3	-3.0	-8.7	-4.1	-7.8
Tools & Home	-2.2	**0.1**	-2.2	-14.3	-3.6	-9.5	**1.0**	-4.9
Toys & Games	-4.8	-1.2	-2.0	-12.7	-1.8	-10.1	-4.3	-7.2
Health & Personal	-2.2	-0.2	-2.0	-12.1	-2.2	-10.1	-1.1	-5.0
Average	-3.5	-1.2	-2.1	-14.3	-2.9	-9.8	-2.6	-6.8

TABLE 6.9: Percentage difference between the compressed and original test accuracy when a compression rate between 50% and 53% was imposed. The column titles *IN*, *LP*, *MC*, *S140*, *SS*, *ST*, *uC* and *V* refer to Intellexer, LingPipe, MeaningCloud, Sentiment140, SentiStrength, Stanford, uClassify and Vivek respectively. Percentage differences greater than or equal to zero are highlighted in bold.

Chapter 7

Facial Expression Recognition and Convolutional Neural Networks

7.1 Introduction

It is a well-studied phenomenon that the majority of human communication is non-verbal in many situations, for example see [171]. Facial expressions play an important role both in such non-verbal communication and in conveying emotion. The human face provides a lot of information [108]. As a result *Facial Expression Recognition* (FER) [261] is an important task for researchers interested in building algorithms or robots that interact effectively with humans. In the past few years CNNs have, as with other computer vision tasks, become the leading approach to FER.

With the explosion of research outputs using CNN for FER in recent years, it is an appropriate time to review the state of the art in this field, provide a critical analysis of what has and has not been achieved, and synthesize recommendations for each step of the process needed for FER. This chapter serves as a guide to those who are new to the field. It provides a critique of past work, highlights recommendations and list some open, unanswered questions in FER that deserve further investigation. In particular this will inform our decisions in the next chapter where we present an evolutionary approach to CNNs for FER.

This chapter focuses on CNN methods published until early 2018 which have been applied to image FER. This study surveyed different types of articles available including conference proceeding publications, journal articles and pre-prints. The articles surveyed were not limited to any particular database. Several aspects were considered when reviewing each paper. The primary aspects were focused on extracting information regarding the implementation of the CNN as well as the rationale behind each decision made

in the entire process (i.e. from pre-processing the data to evaluating and reporting results).

This chapter is structured in such a way that each primary aspect of implementing a CNN for FER is discussed separately. Thus, researchers with knowledge in certain areas can focus on those that they are most interested in, and furthermore, researchers who are new to the field can gain knowledge on each of the necessary implementation steps. The outline of the chapter is as follows. Section 7.2 presents the rationale and discusses related literature reviews. Section 7.3 discusses the available FER datasets. Section 7.4 describes the pre-processing steps that can be applied to boost the classification accuracy of the CNNs. CNN architecture selection, design and optimisation is vital in the success of implementing CNNs. This study reviews this in the context of the existing FER works in section 7.5. A review on hyper-parameter selection and optimisation is presented in section 7.6. Section 7.7 presents the results on commonly used datasets obtained from the studies surveyed. Finally, section 7.8 concludes this study. The extended review is provided in Appendix A.

7.2 Rationale

Interest in facial expressions date back to works in the eighteenth century where Darwin discussed the universality of emotions across mammals [49]. Ekman and Friesen proposed that the expressions are classified into six classes, namely, happiness, sadness, anger, surprise, disgust and fear [70]. Different classification have been established across the years [190]. Early works date back to 1973 where Kanade [117] proposed an automatic facial identification recognition system and in 1978 Suwa *et al.* [250] proposed an automatic FER system. Figure 7.1 illustrates a subject that is expressing surprise and is an example of the type of images that a system would have to classify. There has since been a lot of work in this field of research given the progress in computer vision research, see [44] for a review of works from 2003 to 2012. Kumari *et al.* [132] describes certain traditional approaches which have been applied to FER. This area of research has been applied to a wide number of application areas which include pain detection [153], monitoring stress in space flight [60], assisting with autism [39], human computer interaction [4], empathic robots to encourage medicine intake [222], an alternative to CAPTCHA human verification [67] and E-learning [246].

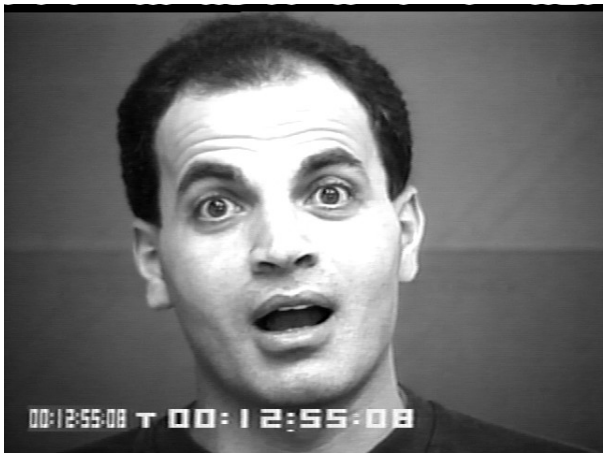

FIGURE 7.1: A subject expressing surprise. Here the goal of a FER system would be to read in the image and output 'surprise' as prediction. The image is from the CK+ dataset.

There are existing literature reviews on the topic of machine learning and FER. This study provides a more comprehensive and detailed critical analysis of the literature – specifically in terms of CNNs. Table 7.1 provides the reader with rapid insights on some of the high-level insights which are discussed in this study. Corneanu *et al.* [42] provides a useful taxonomy of FER and computer vision research areas such as face localisation, feature extraction, classification and multimodal fusion – these are discussed in the context of RGB, thermal and 3D images. In terms of the classification areas not a lot of CNN studies were analysed. Pramerdorfer and Kampel [209] compared 6 CNN studies of depths 5 to 11 against VGG [237], Inception [251], ResNet [96] and an ensemble on a single dataset. The best model was VGG achieving 72.7% accuracy with 1.8 million parameters as opposed to ResNet which obtained 72.4% with 5.3 million parameters. Pantic and Rothkrantz [195] discussed some neural network studies but did not focus on CNNs, similarly, [227, 282] did not describe CNN related works. Martinez and Valstar [164] reviewed a number of studies and described challenges and opportunities in FER research such as medical, marketing and audience monitoring as well as human computer interaction studies. They did not review studies that implemented CNNs. Ghayoumi [80] discussed a few CNN studies. Zhang [299] discussed a few deep belief networks and CNNs. Ko [124] describe a number of long short term memory CNN approaches which have been applied to FER. Latha and Priya [135] discuss 17 CNN studies which have been applied to FER. The existing reviews provide valuable insights into the application of CNNs to FER but an even further critical analysis with a larger number of studies is still to be conducted and is the rationale behind this work. Figure

7.2 presents a flowchart which illustrates the primary aspects which are to be considered when implementing a CNN for FER. The flowchart presents the aspects which are reviewed in this chapter and appendix A.

7.3 Dataset

Several datasets for the task of FER have been made available over the years. The development of such datasets is a tedious task. Certain datasets were created in laboratory environments whereby subjects were asked to pose certain facial expressions and photographs were taken. In other cases, datasets are created by collecting images from the Internet using search engines. Individuals were then asked to assign an emotion to these images. Each dataset has its own problems and advantages and this section reviews and provides an analysis the most commonly used ones. A large list of facial datasets is presented in [220]. Figure 7.3 illustrates an individual conveying the six most common classes of expressions found across the datasets. Table 7.2 lists the top 5 most commonly used datasets from the studies surveyed.

The Extended Cohn-Kanade dataset (CK+) [154] was used in 35% of the articles surveyed and is an extension of the Cohn-Kanade dataset (CK). Even though CK+ is an improved version of CK, it was found that authors still made use of the older version – it was used in 8% of the studies. The CK dataset contains 486 sequence images of 8 posed expressions obtained from 97 subjects. The resolution is 640x490. The enhanced dataset contains 123 subjects and has images of spontaneous smiles and additional posed expressions. A subset of the images are typically used since the dataset is made up of image sequences. In most cases the last three frames of the sequence images were used as those frames represented the subject's peak pose of the expressions [203, 191, 100]. The first frame was used to obtain images for the neutral expression [278, 288].

The Japanese Female Facial Expression (JAFFE) dataset [158] was smaller compared to CK+, it contained 213 images of individuals posing 7 expressions and was obtained from 10 subjects. The resolution of the images is 256x256. This dataset was used in 23% of the studies. The Facial Expression Recognition Challenge 2013 (FER'13) dataset [86] was used in 32% of the studies and contains 35,887 images of subjects displaying 7 expressions. The images are smaller than other widely used datasets with a resolution of 48x48 pixels. The dataset was created using the Google search API. The Karolinska Directed Emotional Faces (KDEF) dataset [155] contains 4900 images of 70

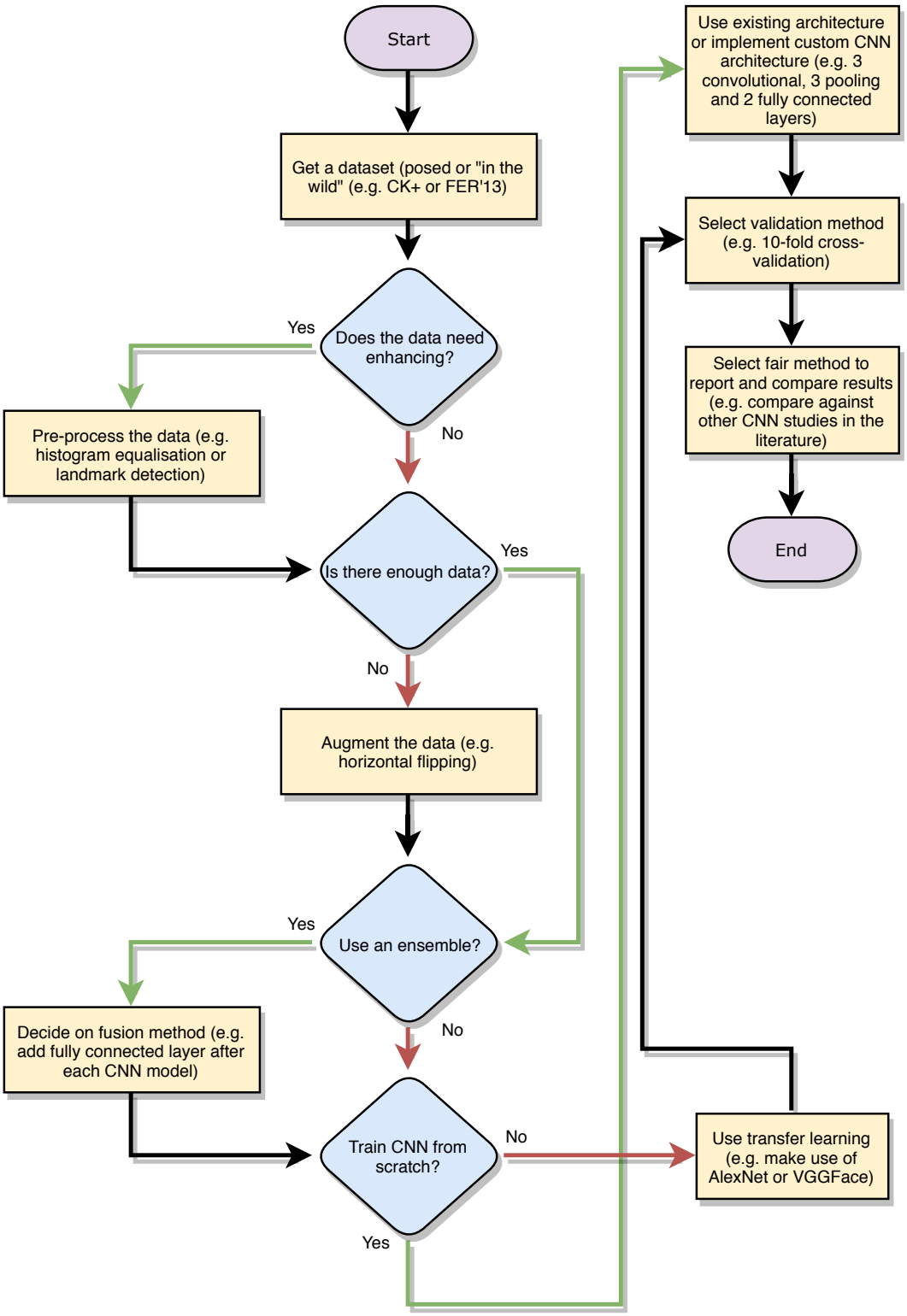

FIGURE 7.2: Flowchart illustrating the primary steps involved when implementing a CNN for FER. The flowchart also illustrates the aspects which are reviewed in this thesis (both in this chapter and in Appendix A).

TABLE 7.1: The table presents the primary aspects that needs consideration when implementing a CNN for FER. This table serves as a high-level summary of the insights. The thesis discusses these in greater detail.

Primary aspects	**Recommendation & insights**
Datasets	For purposes of benchmarking against other studies use CK+, JAFFE and FER'13. Supplement with "in the wild" datasets to overcome issues with posed datasets.
Pre-processing	Recommend both pre-processing (in particular histogram equalisation) and detection of facial landmarks. For the latter the Viola Jones algorithm is most commonly used
Data augmentation	Horizontal flipping can be used to increase amount of data by 2 folds. Combination of cropping (4 corners and center) and then flipping can increase data by 10 folds.
Transfer learning	If size of dataset is small then apply transfer learning, e.g. implement AlexNet. Alternatively, extract features from the last convolutional (or first fully connected) layer and input those features into a SVM (or other algorithm).
Ensembles	The simplest implementation is to combine each CNN by connecting each of them to a single fully connected layer followed by an output layer.
Implementing custom architecture	A good starting point is to implement 3 convolutional, 3 max pooling and 2 fully connected layers. The ReLU activation function can be used in all layers and add a softmax activation to the output layer.
Using existing architectures	Recommend starting with AlexNet and VGG-Face. Using an existing architecture implies that less time is spent determining the optimal depth and associated hyper-parameters.
Network and hyper-parameter optimisation	Recommend using an optimisation algorithm to find the best choice of network architecture and hyper-parameters (e.g. *EDEN* in Chapter 4). This is currently a very under-explored area in the literature.
Validation techniques	Run a large number of executions of the CNN instead of a single execution. Conduct cross-dataset experiments. Accuracy metric should be used to benchmark with majority of studies.
Reporting results	A standard on how to compare and report results is currently lacking. In section A.6.9 we propose such a standard to be followed.

TABLE 7.2: Characteristics of the most commonly used FER datasets. *Posed* refers to whether or not the dataset has images of subjects posing the expressions. *Lab* denotes whether or not the dataset has images of subjects in a laboratory setting. *"In the wild"* refers to datasets that have images of people expressing realistic expressions, typically images are obtained from the Internet. *Usage* denotes the percentage that each dataset was reported in the studies surveyed.

	CK+	**FER'13**	**JAFFE**	**SFEW**	**KDEF**
Posed	Y	N	Y	N	Y
Lab	Y	N	Y	N	Y
Subjects	123	-	10	-	70
Ethnic Diversity	Y	Y	N	Y	-
Gender (M/F)	31/69	-	0/100	-	50/50
Age	18-50	-	-	1-70	18-37
Images	593	35,887	213	700	4,900
Year	2010	2013	1998	2011	1998
"In the wild"	N	Y	N	Y	N
Resolution	640x490	48x48	256x256	720x576	562x762
Usage	35%	32%	23%	11%	8%
Classes	Anger (45), disgust (59), fear (25), happy (69), sadness (28), surprise (83), contempt (18), neutral (-)	Anger (4953), disgust (547), fear (5121), happy (8989), sadness (6077), surprise (4002), neutral (6198)	Anger (30), disgust (19), fear (32), happy (31), sadness (31), surprise (30), neutral (30)	Anger (104), disgust (81), fear (90), happy (112), sadness (92), surprise (86), neutral (98)	Anger (700), disgust (700), fear (700), happy (700), sadness (700), surprise (700), contempt (700), neutral (700)

FIGURE 7.3: Illustrating the six common expressions found in the majority of the FER datasets. The top row has images of an individual expressing surprise, sadness and happiness. On the bottom row the individual is expressing disgust, anger and fear. The images are from the KDEF dataset.

subjects each displaying 7 expressions. The subjects were selected based on certain criterion, for example, they could not have facial hair or eyeglasses. The ages varied between 20 to 30 years, and there was an even gender balance. The resolution of the images is 562x762 pixels. This dataset was used in 8% of the studies. The Static Facial Expressions in the Wild (SFEW) dataset [54] was used in 11% of the studies. It contains 700 images obtained from 95 subjects which were extracted from the Acted Facial Expressions in the Wild (AFEW) dataset – a video dataset. The ages of the subjects varied from 1 to 70.

Other datasets which were used include: MMI Facial Expression Database (MMI) [196] used in 7% of studies, Toronto Face Dataset (Toronto) [248] used in 7%, Radboud Faces Database (RAFD) [134] used in 7%, Emotion Recognition in the Wild (EmotiW) [57, 55] used in 7%, Binghamton University 3D Facial Expression (BU-3DFE) [291] used in 3% of the studies which is a 3D dataset thus containing expressions posed at various angles, Bosphorus [228] is also a posed 3D dataset and was used in 1%, CMU Multi-PIE [89] which used multiple illumination conditions used in 3%, Facial Expression Recognition and Analysis Challenge (GEMEP-FERA2011) [268] contains images from different angles and illumination intensity - it was used in

2%, MPLab GENKI Database (GENKI) [265] used in 2%, Multimedia Understanding Group (MUG) Facial Expression [6] used in 1%, Natural Visible and Infrared facial Expression (USTC-NVIE) [275] was proposed to introduce an infrared dataset and was used in 1% of studies, Oulu-CASIA NIR&VIS [253] contains images sequences of subjects under three different illumination conditions (from bright to dark) and was used in 1% of the studies, iCV Multi-Emotion Facial Expression Dataset (iCV-MEFED) [157] contains posed laboratory images of much higher resolution compared to most datasets (5184x3456 pixels) and was used in 1% of the studies. Lawrence *et al.* [136] used the ORL Database of Face released in 1994.

Limitations in reproducibility of results and in comparisons are inevitable in the case where authors collect their own images and do not release the data. Furthermore, no comparison can be made in the case where authors do not provide details about the dataset used [165]. Certain studies used datasets which were collected from the Internet. Baumgartner *et al.* [17] collected their own dataset to overcome the issue that most datasets are only available for academic use. The dataset contains 280,000 images which was obtained from Flickr [1] by searching for images based on a list of keywords for 7 expressions. Lucey *et al.* [154] collected a dataset of 5 expressions and [51] created a balanced dataset of 2,156 images containing 7 expressions. Song *et al.* [240] collected images from the Internet which were manually labelled but further details are not available. Mollahosseini *et al.* [180] describe their process of creating an "in the wild" dataset. They used 2 people to label the images which were collected from Google, Bing and Yahoo for which 1250 emotion keywords were used in 6 languages. The images which had the most agreement between the 2 taggers were for the 'happy' emotion. Peng *et al.* [201] created a similar dataset of 2000 images using Google and Baidu using 6 keywords. They also show that their CNN outperforms traditional methods on this dataset.

A significant limitation in some of the datasets used from the studies surveyed is that they are of posed expressions in a laboratory setting and consequently do not contain representative images of facial expressions which humans convey spontaneously – an example is presented in figure 7.4. The figure also illustrates an image which is more representative of human emotions, datasets containing non-posed emotion are referred to as 'in the wild' datasets.

[1] www.flickr.com

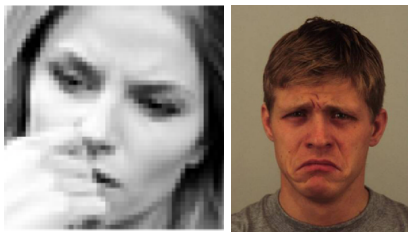

FIGURE 7.4: Illustrating the difference between images from "in the wild" datasets and laboratory posed datasets. The former is far more realistic than the latter. Left: spontaneous emotion which appears realistic. Right: posed emotion in a laboratory

Variations in head pose, illumination, ethnicity, age, occlusion from eyeglasses, beard and moustache are factors which render FER challenging are not included in most of the datasets. CK+ and JAFFE for instance were both created in laboratory setting and subjects were asked to pose. Furthermore, the subjects in KDEF were amateur actors who were asked to rehearse the expressions. The dataset however has an equal gender balance. JAFFE is limited by the gender and ethnicity imbalance. CK+ addresses ethnic diversity however the dataset is still imbalanced with 81% images are of Euro-Americans, 13% Afro-Americans and 6% images of other ethnic groups. To deal with the limitations of posed laboratory settings, authors have released datasets of images of spontaneous non-laboratory facial expressions – these are referred to as "in the wild" datasets. FER'13, SFEW and EmotiW'17 are examples of such datasets. From the trends in FER competitions (e.g. [56]), it is evident that effort is being concentrated on FER in the wild, yet a large number of studies are using older datasets.

On average, based on the literature surveyed, two datasets (a rounded-up value) were used in the research articles. As a result, one can immediately ask if it is sufficient to report on the novelty of a new research idea by conducting experiments on two datasets. From the no free lunch theorem it is clear that if a method performs well on one dataset it is not necessarily true that the method will perform just as well on another dataset. It would thus be scientifically more correct to use a larger number of datasets. Why is this not the case? Acquiring access to most datasets requires obtaining permission from the author of the dataset and signing an end user agreement form. It is possible that researchers are not able to gain access to certain datasets because of this. JAFFE is easier to obtain compared to other datasets as a signed end user agreement is not required, this dataset was used in 23 % of the studies.

7.4 Pre-processing

Pre-processing is a vital step when dealing with image data. This can be described as some method to transform the original input image using some image manipulation function. The function renders a new image with the ultimate goal being that the new image will help the predictive performance of the classifier. Applying a CNN directly to images which have not been pre-processed in some way can yield weaker recognition performance. For example, consider a dataset containing images of people for which the lighting conditions across the face are not consistent. It would be more suitable to attempt to correct the variation in illumination in such a way to have the same amount of lighting exposure across the entire face so that the CNN can learn useful features across the entire face. This section describes and analyses the different approaches that authors have used in their studies.

The use of resizing was reported in a large number of papers. Here values ranged from 48x48 to 256x256 pixels. In most studies the authors resized their images to 48x48 or 64x64 pixels. Studies which used the former resolution include [2, 149, 293, 179, 207, 267, 278] and studies which used the latter include [286, 156, 32, 289]. Large resolutions which were used include 128x128 [181], 224x244 [207, 215] and 256x256 [33]. Despite the fact that most used 48x48 pixels, there does not seem to be a lot of consistency within the studies and there has not been a comparison to guide future research in selecting an optimal resolution. In certain studies whereby the authors use a pre-defined CNN architecture they resize the images to match the input for which the architecture expects. Pilla *et al.* [203] used Viola Jones [271] to crop images to 156x176 then resize (using bicubic interpolation) to 227x227 to match the input requirements for AlexNet [130]. Viola Jones is a commonly used method to perform object detection and is typically used for face detection.

Authors make use of face detection algorithms to either extract features, align the faces or crop the images. To align faces the studies primarily make use of the location of the eyes and re-align them so both are horizontal. Once the landmarks and face has been detected the image can be cropped so that it contains only relevant facial pixels around the eyes and mouth which eliminates irrelevant facial parts and background noise [203].

Cropping is often applied, for examples, see [283, 292]. The Viola Jones

face detection algorithm was implemented the most frequently, typically authors made use of the Open Source Computer Vision Library (OpenCV) implementation [2] to achieve this. OpenCV is a widely used library for image processing. Studies which used this approach include [288] which cropped the images to 39x39 pixels, [113] also did the same and cropped to 48x48 and [294] cropped the images to 128x128. DLIB [3] is a C++ library which performs facial landmark detection and was used by [235] and [215] to extract and crop faces. If faces were not detected by DLIB in [215] then the Viola Jones approach was applied as an alternative. IntraFace [287] (a software package for face image processing) was also used to extract landmarks, detect and align the faces. Mollahosseini and Mahoor [179] and Kim *et al.* [120] used this approach and resized the images to 48x48 pixels, [59] used Viola Jones and IntraFace to extract faces and [186] used Viola Jones to detect faces and IntraFace was used to remove false positives. A similar approach for removing false positives was used in [92] whereby an ensemble of regression trees was used and a CNN model was used to remove the false positives. The same ensemble of decision trees for face detection was used in [91]. Luxard FaceSDK [4] was used in [214]. Kim *et al.* [121] used Zhu-Ramanan [304] and Viola Jones for face detection and Surace *et al.* [247] used Google Vision API [5] for face detection.

Studies made use of equalization techniques to overcome the issues in variation of lighting conditions. In a controlled laboratory environment, there is typically less drastic variation in lighting conditions since the equipment used remains static, however in more complex settings and "in the wild" datasets this issue has to be dealt with to enable the CNN to achieve superior performance. Six of the studies made use of some form of equalisation. Yu and Zhang [293] implemented histogram equalisation [187] (a method for adjusting image intensities to enhance the image contrast) and then normalised the pixels, their approach was evaluated on the SFEW dataset. The authors of [233] conducted experiments which compare the performance of their CNN with and without histogram equalisation. Their findings reveal that superior results can be achieved using histogram equalisation. Kim *et al.* [121] investigated the use of a CNN ensemble with different pre-processing applied to the input images of the CNNs. The three methods used were either to use the original image, illumination normalisation using the Illumination

[2] https://opencv.org
[3] http://dlib.net
[4] https://www.luxand.com
[5] https://cloud.google.com/vision/

Normalization techniques for robust Face recognition (INFace) toolbox [6] and contrast enhancement using MATLAB. A similar approach was conducted in [120]; an ensemble of CNNs was used for which certain CNNs used either illumination normalisation, contrast enhancement or histogram equalisation. Shin *et al.* [235] compared histogram equalisation (implemented using OpenCV), isotropic diffusion based normalisation, DCT-based normalisation and difference of Gaussian (all three implementing using INFace toolbox). The best average performance was achieved using histogram equalisation when compared on FER'13, SFEW, CK+, KDEF and JAFFE. Normalisation and standardisation of the input pixels were reported in studies, for examples see [242, 73, 255, 225].

There were five studies for which there were no claims of any pre-processing being performed to the input images, see [16, 25, 37, 18, 146]. It is unclear whether these studies did not include any pre-processing steps or if the authors did not report any details about this. From the study of [204] it is clear that a much larger improvement in classification accuracy can be achieved when using pre-processing as opposed to not using it. They compare test accuracy results obtained on 32x32, 64x64 and 128x128 pixel images when using no pre-processing and when using cropping, global contrast normalisation, local normalisation, histogram equalisation, salt and paper and speckle noise, and a combination of cropping along with noise. Each of the aforementioned pre-processing techniques outperformed the experiments without pre-processing on all three resolutions. The best overall average accuracy was obtained when using cropping and adding noise.

From the studies surveyed it was observed that Viola Jones face detection was conducted the most often, Sannikov *et al.* [226] states that it is a rapid method which requires little computational effort and achieves good recognition accuracy. Kim *et al.* [120] found that IntraFace was able to achieve up to 85% accuracy in detecting faces in the FER'13 dataset. It would be of interest to compare the different methods used in literature, and additionally, to benchmark other algorithms which have not been used in the studies surveyed to find the most suitable one across various FER datasets. Cropping is usually conducted once the face has been detected and is applied to remove background information and to extract only the facial region. Restricting the CNN inputs to only the face region can reduce processing time and the number of trainable neural network parameters in the first layer given that the input image is smaller than the original one. It would be of interest

[6] http://luks.fe.uni-lj.si/sl/osebje/vitomir/face_tools/INFace/

to investigate further into how much cropping is suitable and to determine the optimal bounding box around a face which yields the best performance across several datasets.

Resizing was applied once the faces had been detected and cropped. In the studies for which the authors used existing CNN models resizing was often done so that the images match the input requirements of the architecture. The studies of [179] and [149] mention that reducing resolution does not significantly impact the prediction performance but does decrease training time. It would be of interest to verify this claim by conducting experiments on several FER datasets. From the literature, it was observed that histogram equalisation resulted in increased overall performance and is primarily used to cater for the variations in illumination across the face.

Luo *et al.* [156] conducted a unique set of experiments. The authors propose a novel input image transformation from RGB colour channels into 3 different channels. Here the new channels are referred to as AGE; angle-, gradient- and edge-values. The performance of a CNN was compared using RGB and AGE input features on the SFEW dataset – the results reveal that the CNN performed better using AGE features. It would be of interest to devise an optimisation algorithm to determine the most optimal input transformation as an alternative to RGB or greyscale images. Typically authors did not make use of RGB images, instead, converted the images into greyscale images, for examples see [252, 66, 293, 221, 27].

It was observed that certain studies did not use, or did not mention the use of pre-processing techniques. Furthermore, it was observed that in certain studies there were details which were lacking, for example studies mention that faces were extracted but it is unclear as to how this was performed. Lopes *et al.* [152] demonstrate the benefits of applying pre-processing as opposed to not applying it. Finally, it was noticed that a great number of studies do not report on the time taken to conduct pre-processing on a test image.

7.5 Network Architectures

Constructing an optimal CNN architecture and the selection of the hyper-parameters is a challenging task. There are no problem domain independent rules on how to determine these and they have mostly been achieved through trial and error by running many different settings and evaluating the performance on validation data [119]. Fewer parameters are obtained when constructing a shallow CNN, however, as a consequence in most cases

TABLE 7.3: Illustrating the number of times that each existing architecture was reported in the studies surveyed.

Architecture	**Number of times reported**
AlexNet [130]	11
VGG-16 [237]	5
VGG-Face [198]	4
Variants of Inception [251]	2
Variants of ResNet [96]	2
Xception [36]	1

the performance is not as good when compared to a deep model which has a larger number of trainable parameters. There is a trade-off between performance and training time as well as computational requirements. Furthermore, the choice of hyper-parameters will largely affect the performance of a CNN, for example, consider the effects of a large learning rate as opposed to a smaller one. Which activation functions should be used? How will the performance of a CNN for FER be affected by the choice of loss function? Early CNNs for FER – published in 1997 – include [136] which implemented 2 convolutional layers, 2 pooling and 1 fully connected layer. This section reports on the analysis of the CNN architectures and hyper-parameters which have been reported in the literature.

From the studies surveyed, it was found that 26 studies did not implement their own CNN, instead the authors of these studies used existing architectures from the literature. This does not necessarily imply that the authors used the exact pre-trained weights which were trained for each model, but rather that they used the overall structure of the layers and then either fine-tuned, applied transfer learning or trained the model from scratch. Immediately a question can be posed, why were those particular architectures selected by the authors as opposed to another one from the literature?

Table 7.3 presents the architectures along with the number of times each one was reported in the literature. VGG-Face was used in [231, 215, 277, 200]. AlexNet was used in [203, 167, 166, 302, 33, 186, 191, 91, 146, 267, 81]. VGG-16 was used in [207, 210, 277, 294, 2]. Variants of Inception were used in [283, 92]. ResNet32 and ResNet101 were used in [208] for which the latter achieved better performance. Peng *et al.* [200] used both ResNet and VGG-Face which were pre-trained on the CK+ dataset and fine tuned on another FER dataset. Both models have a softmax layer and the maximum probability across both models was used as the final prediction. Xception was used by Abbas and Chalup [2].

The observations reveal that only a few authors provided their rationale for their choice of architecture. Rationale for using VGG-based models were described in 4 studies. The authors of [207] state their rationale for using VGG-16 is because it is the most commonly used model for object recognition. Schwan *et al.* [231] used VGG-Face because it was freely available and achieves good results. VGG-Face was used in [92] because it is the state-of-the-art in face recognition. Additionally, the authors used ResNet152 because it also is state-of-the-art on the ImageNet dataset. VGG-Face was used in [215] because it was already pre-trained on face recognition and was deemed appropriate for FER. Chen *et al.* [33] used AlexNet since other studies in the literature have used the pre-trained model and it achieved good performance. Good speed and accuracy are listed as reasons for using AlexNet in [186]. Faster R-CNN was explored in [144] and it would be of interest to further explore this on additional datasets. The most commonly used architectures were AlexNet (very few authors justified this choice as opposed to using another architecture), followed by VGG-16 and VGG-Face.

Next we discuss the studies which did not use an existing architecture from the literature, but instead proposed their own custom one. Five studies reported that their implementations were inspired from another existing study in the literature. The implementation in [13] was based on the Xception model; the authors used depth-wise separable convolutional layers [114] which was used to reduce the number of trainable parameters as opposed to canonical convolutional layers. Burkert *et al.* [25] implemented a CNN which was inspired by GoogleNet [251], the authors mention the success of this architecture in existing literature and represents the rationale behind their architecture selection. Their architecture is composed of several parallel feature extraction blocks, and these blocks contain parallel convolutions and max pooling operations which are followed by a concatenation operation. The authors of [179] state the success of the Inception architecture in literature as rationale for basing their implementation on it. Sang *et al.* [225] and Barsoum *et al.* [16] implemented architectures that were inspired by VGG. The authors of [225] point out the simplicity in the design of the architecture and its success in the 2014 ImageNet challenge. They reconstruct the original model and reduce the number of trainable parameters by reducing the number of filters in the convolutional layers in an attempt to reduce overfitting. Barsoum *et al.* [16] mention that various architectures were investigated but eventually implement a network inspired by VGG, the authors however do not provide a rationale for this decision. Ali *et al.* [71] based their network

from the architecture implemented in [90] for which the author attempted to optimise the number of filters, depth, pooling layers and dropout probabilities within a certain range of values. The authors of [210] used VGG-16 and VGG-Face along with two architectures whereby one was much deeper than the other. The deeper custom model (made up of convolutional, max pooling, batch normalisation, dropout and several fully connected layers) outperformed the other models. Alizadeh and Fazel [9] implemented three custom architectures of increasing depth. The deepest architecture did not yield the best results. The architecture (ranked 2nd in terms of depth) with 4 convolutional, 4 max pooling along with batch normalisation on all layers obtained the best result. The authors of [81] split the FER13 data into three groups, neutral class vs. all of the classes combined, individual classes except for the neutral class and the all of the individual classes. The authors compared GoogLeNet and AlexNet on these groups. In the first group GoogLeNet performed the best, in the second group the performance obtained at the last iteration was similar for both architectures and in the last group AlexNet did the best. AlexNet, VGG-19 and GoogLeNet were compared in [267]. The networks were trained on KDEF for 40 iterations and evaluated on JAFFE. The best result was obtained using VGG-19.

Table 7.4 presents the average architecture that were implemented by the authors of the studies examined. The table only reports on the architecture of custom implementations and not on studies which used existing architectures. It was observed that most authors had a max pooling layer for each convolutional layer. Models inspired by VGG for example had more convolutional layers than pooling layers. Despite the fact that few authors provide rationale for their choices it was interesting to see most implementations used a kernel size of 5x5 in their first convolutional layer. This table can serve as an initial guide on the architecture that one can implement when constructing a CNN for FER.

Table 7.5 presents the activation functions which were used in the studies that implemented their own custom CNN architecture. ReLU was used the most often in both the convolutional and fully connected layers with values of 36 and 20 respectively. Hyperbolic tangent was used in the convolutional layers in three studies and two studies used it in the fully connected layers. Sannikov *et al.* [226] mention that in their preliminary findings ReLU lead to worse results than hyperbolic tangent. There was only one study [7] that used LeakyReLU in the convolutional layers to overcome the dying ReLU problem (whereby the neurons output zero for any input). Sigmoid

TABLE 7.4: Average, standard deviation, minimum and maximum of the CNN architectures used in the studies for which the authors implemented their own architecture. This table only reports on the studies which implemented a single architecture. Only one value for the kernel size is presented since the same value is used for both the width and height of the kernel. The first kernel denotes the kernel size used in the first convolutional layer. On average, authors implemented 3 convolutional layers consisting of 42 filters (in the first layer) of size 5x5, 3 pooling layers along with 2 fully connected layers at the end (values rounded to nearest integer).

	Dense	**Convolutional**			**Pooling**
	Layers	Layers	Filters (first layer)	First Kernel Size	Layers
Mean	2.2	3.2	42.3	5.1	2.7
S. Dev	0.9	1.5	26.0	1.6	0.6
Min	0.0	1.0	4.0	3.0	1.0
Max	4.0	10.0	100.0	11.0	4.0

TABLE 7.5: The table presents the activation functions (refer to section 2.5.2 for details on activation functions) that were reported in the studies that implemented their own CNN architecture. The number of times that each function was used in the convolutional and fully connected layers are shown.

Activation function	**Convolutional layers**	**Fully connected layers**
ReLU	36	20
LeakyReLU	1	0
PReLU	2	1

was only used in the convolutional layers in a single study. PReLU was used in the convolutional layers in two studies [172, 301] and in the fully connected layers in a single study [301]. Only a few authors provided a rationale for their choice in activation function, nor did they attempt to optimise their CNN's performance by experimenting with various functions. Three studies reported their decision to use ReLU was to overcome the vanishing gradient problem [113, 140, 279]. Lopes *et al.* [152, 222, 149, 156] mention that ReLU learns fast and was their rationale for using it. For the output layer the softmax function was used in almost every study. Tang [255] compared the softmax to L2-SVM on a single dataset and the results reveals that L2-SVM results in better performance. It would be of interest to further investigate this approach and validate its performance on multiple FER datasets. Although it is true that the objective of FER studies is not to propose new or investigate activation functions, authors should attempt to optimise such details to boost the overall classification performance.

Dropout was used in either the convolutional or fully connected layers in 29 of the studies which implemented their own architecture. Dropout

was used more frequently in the fully connected layers than in the convolutional layers. Certain authors stated that dropout was used as a regularisation technique. Additionally, other regularisation techniques were used such as batch normalisation (BN) [107], local response normalisation (LRN) [130] and L1/L2 regularisation – these techniques were found in 10 studies. [172, 277, 13] used BN after each convolutional layer, Luo *et al.* [156] used it after the first 2 out of the 4 convolutional layers and Ucar [266] used it after the first convolutional and after the last convolutional layer. [172, 156] mention that their rationale for using BN was to reduce the training time. L1 and L2 normalisation was used on each layer in [37]. LRN was used after the convolutional and pooling layers in [25] and after some of the convolutional and pooling layers in [222, 247]. Few studies investigate the effect of regularisation techniques in the context of FER. Hinz *et al.* [100] investigated 8 variations of dropout, BN and max pooling dropout [281] on the CK dataset using 10-fold cross-validation. A fixed architecture of 3 convolutional and 2 fully connected layers followed by a softmax output was used in these experiments. The findings revealed that BN helped achieved faster convergence. The best result was achieved when using max pooling dropout after each convolutional layer, dropout after each fully connected layer and BN after each layer.

7.6 Network and Hyper-Parameter Optimisation

From the studies surveyed, it was observed that there was not a lot of discussion on network architecture optimisation. It is unclear whether authors did not attempt to optimise their network architecture or whether they omitted these details. Twelve studies stated that some form of optimisation was conducted. Alizadeh and Fazel [9] attempted to determine the most suitable network by implementing three architectures. The first had 2 convolutional and 1 fully connected layer, the second had 4 convolutional and 2 fully connected layers, and finally the authors tried up to 6 convolutional layers. They state that the deepest model did not result in an improvement in performance, and that the best architecture was the network with 4 convolutional layers. Raghuvanshi and Choksi [210] tried to optimise the number of fully connected and convolutional layers as well as the number of filters in the convolutional layers. Their findings reveal that increasing the number of filters in the initial layers did not lead to an improvement in performance, however increasing the number of filters at deep layers lead to improvements. They

also observed that using 4 fully connected layers outperformed using a single one. Sonmez and Cangelosi [252] provided very few details except that the authors tried different depths of 3, 5 and 7 – a depth of 5 resulted in the best performance. Jung *et al.* [113] tried to optimise the number of neurons in two fully connected layers, the authors do not mention how this was done or what values were tested. Shan *et al.* [233] attempted to determine the optimal number of filters in the first two convolutional layers by evaluating on JAFFE and CK+. The values used for the first was 4, 6 and 8, and for the second they used 10, 12 and 14. The findings show that 6 and 12 filters in the first two convolutional layers respectively yielded the best performance. Sang *et al.* [225] compared various network depths (5, 7, 9 and 11) by increasing the number of convolutional layers. The best result was obtained from a depth of 9 which resulted in an increase from 70.2% (depth 5) accuracy to 71.9% (depth 9) and then decreased for a depth of 11. Li *et al.* [145] mentioned that they optimised the number of convolutional layers and filters but do not provide any details of the optimisation. Kim *et al.* [120] tried 1 or 2 convolutional, and 2 or 3 fully connected layers both with and without max pooling. The best result was achieved with 2 convolutional and 2 fully connected layers along with max pooling. They mention that increasing the number of neurons in the fully connected layers from 500 to 1000 improved performance and that increasing from 2 to 3 fully connected layers did not yield improvements. Spiers [242] optimised the number of neurons in the fully connected layers by trial and error but no further details are provided as to what values were used, the best values were 384 and 192 for the first and second fully connected layer respectively. Sannikov *et al.* [226] mention that hundreds of experiments were conducted to determine the best architectures which had the least number of trainable parameters and for which the predictive error was below a specified threshold. Ali *et al.* [71] state that several architectures were tested but provide no further details. Valero [267] investigated different max pooling kernel sizes of 2x2, 4x4 and 10x10 – the best was 2x2. They determined the effect of increasing the number of neurons in the fully connected layers from 3 to 2048 for which the results reveal that there is no need for more than 128 neurons. They further compared using 2, 3, 5 and 10 fully connected layers, the results revealed that there is no need for more than 5 after which the performance decreased.

Based on the studies discussed it is clear that not a lot of research has been done in optimising the CNN architectures for FER tasks. Furthermore, authors simply mention that they experimented with this but provide no discussion on which values were tested and what their methodology was. It is true that running multiple CNNs with various architectures and training these from scratch is a computationally expensive task, but such an exploration could be valuable to help guide future studies by providing insights into suitable architectures.

The methodologies that authors have used to optimise the hyper-parameters associated with training CNNs are discussed next. The findings reveal that few authors attempted to optimise the hyper-parameters. Ten studies mention that some form of optimisation was conducted. Pons and Masip [207] performed a grid search on certain hyper-parameters then trained and validated the CNNs to determine the most optimal values. Raghuvanshi and Choksi [210] attempted to optimise the learning and dropout rate but they do not discuss their approach. Alizadeh and Fazel [9] also do not provide their methodology but the authors mention that different values for the learning rate and regularisation were performed on 30 epochs. Sonmez and Cangelosi [252] mention that different values were explored for the batch size and number of epochs, their optimal values were 25 and 100 respectively. Luo *et al.* [156] tried different batch sizes with values of 64, 128, 256 and 512 – the best was 256. Shan *et al.* [233] optimised the learning rate by trying values ranging from 0.1 to 0.9 on the JAFFE and CK+ datasets. The best values were 0.5 and 0.7 respectively. Mavani *et al.* [166] mention that they optimised the learning rate, regularisation, dropout and decay rate but provide no evidence or details on their methodology. Similarly, [71, 7] mention that different hyper-parameters were explored but the approach was not discussed. Valero [267] explore different learning rates and their effects when applied to different optimisers, 7 optimisers were investigated along with 6 values for the learning rate ranging from 0.1 to 0.001. The findings reveal that Nesterov [185], Momentum, RMSProp and Adam all achieve the highest accuracy on different learning rates. Xiang and Zhu [284] randomly sampled values and evaluated the performance on the validation images. Surace *et al.* [247] mention that preliminary experiments revealed that RMSProp outperformed Adam and Adagrad [62].

From the studies surveyed it is apparent that authors omit the details of their methodology and the values investigated when attempting to optimise

the hyper-parameters. Those who did discuss this examined only a few parameters on a small number of datasets. Hyper-parameter optimisation is an essential part of training CNN models to achieve high accuracies and more details should be provided in future work.

The average of the number of epochs and batch size was computed for the studies which reported on these values. The average number of epochs was 790 (median value of 100), minimum of 20 and maximum of 10,000. The average batch size was 107 (median value of 100), with a minimum of 25 and maximum of 256. It was observed that a large number of studies did not report on these values. The most commonly reported optimiser was stochastic gradient descent which was used in 20 studies. There were a number of studies which did not report on their choice of optimiser. In terms of the loss function, the cross-entropy function was most commonly reported. Pons and Masip [208] proposed an interesting loss function whereby two tasks were learned simultaneously (FER and action unit (AU) recognition). Typically two loss functions would be used (softmax for FER and sigmoid for AU); instead, the authors proposed a single sigmoid cross-entropy function and showed that it outperformed the former approach. Using ResNet101, the authors report (on the SFEW dataset) 40.3% on the former approach and 45.9% on their new proposed loss function. There were a number of studies that did not report on their choice of loss function.

The weight initialisation method used by authors was also analysed – few authors reported their approach. Jung *et al.* [113] used a value of 0.05 and 0.01 to initialise the weights for the convolutional and fully connected layers respectively. Shan *et al.* [233] randomly initialised the values between -1 and 1. The authors of [152, 247] used the Xavier initialisation approach [82]. The authors of [59, 118, 119] initialised their values from a Gaussian distribution. [221, 218] used an auto-encoder to obtain the initial weight values.

7.7 Results from Literature

Tables 7.6 to 7.11 presents the results obtained from the studies surveyed for each of the most popular datasets. The table also presents the validation method along with the data augmentation used.

TABLE 7.6: Classification accuracy results (%) on the CK+ dataset. The results are presented in ascending order for each validation method. The validation values are represented as follows 3-CV (3-fold stratified cross-validation), 10-CV (10-fold cross-validation), Hold (holdout) and LODO (leave-one-dataset-out), LOSU (leave-one-subject-out) and LOSO (leave-one-sample-out). The data augmentation techniques are defined as follows C (crop), F (flip), R (rotation), S (shift), O (other) and Z (zoom).

Result	Validation	Augmentation	Study
90.00	3-CV	-	[27]
86.54	10-CV	-	[113]
92.06	10-CV	-	[286]
92.22	10-CV	-	[150]
96.40	10-CV	F, R, S, O	[118]
96.95	10-CV	-	[302]
98.30	10-CV	-	[302]
98.60	10-CV	F, R, S, O	[24]
99.10	10-CV	F, R, O, Z	[7]
99.60	10-CV	-	[25]
80.30	Hold	-	[233]
80.41	Hold	F, R	[18]
83.00	Hold	O	[145]
94.40	Hold	-	[191]
97.00	Hold	-	[283]
97.60	Hold	-	[292]
98.52	Hold	C, F	[203]
88.50	LODO	R, S, O	[294]
99.38	LOSU	O	[252]
96.02	LOSO	-	[167]

TABLE 7.7: Classification accuracy results (%) on the JAFFE dataset. The results are presented in ascending order for each validation method. The validation values are represented as follows 10-CV (10-fold cross-validation), Hold (holdout) and LODO (leave-one-dataset-out). The data augmentation techniques are defined as follows C (crop), F (flip), R (rotation), S (shift), O (other) and Z (zoom).

Result	Validation	Augmentation	Study
88.70	10-CV	-	[151]
94.10	10-CV	-	[94]
98.12	10-CV	-	[167]
76.74	Hold	-	[233]
86.38	Hold	C	[46]
87.70	Hold	-	[74]
91.40	Hold	R, S, Z	[73]
96.10	Hold	-	[266]
44.32	LODO	O, R, S	[294]

TABLE 7.8: Classification accuracy results (%) on the FER'13 dataset. The results are presented in ascending order for each validation method. The validation values are represented as follows 10-CV (10-fold cross-validation) and Hold (holdout). The data augmentation techniques are defined as follows C (crop), F (flip), R (rotation), S (shift), O (other) and Z (zoom).

Result	Validation	Augmentation	Study
98.60	10-CV	F, O, R, S	[24]
48.00	Hold	F, S	[210]
60.7	Hold	-	[284]
65.03	Hold	C, F	[149]
67.21	Hold	F, R, S, Z	[53]
69.80	Hold	-	[231]
70.74	Hold	C	[110]
71.20	Hold	-	[255]
71.80	Hold	-	[301]
71.90	Hold	C, F, O R	[225]
73.73	Hold	C, F	[120]
96.00	Hold	F	[140]

TABLE 7.9: Classification accuracy results (%) on the EmotiW'17 dataset. The results are presented in ascending order for each validation method. Hold represents the holdout validation method. The data augmentation techniques are defined as follows C (crop), F (flip), R (rotation), S (shift), O (other) and Z (zoom).

Result	Validation	Augmentation	Study
72.83	Hold	F, R, S, Z	[2]
78.53	Hold	F, R, Z	[215]
79.78	Hold	-	[277]
80.60	Hold	-	[92]
80.90	Hold	C, F, O	[254]

TABLE 7.10: Classification accuracy results (%) on the SFEW dataset. The results are presented in ascending order for each validation method. The validation values are represented as follows 2-CV (2-fold cross-validation), 10-CV (10-fold cross-validation) and Hold (holdout). The data augmentation techniques are defined as follows C (crop), F (flip), S (shift) and O (other)

Result	Validation	Augmentation	Study
26.14	2-CV	-	[150]
47.70	5-CV	F	[179]
42.90	Hold	-	[207]
54.30	Hold	C, F	[172]
55.15	Hold	F, O	[59]
61.29	Hold	O	[293]

TABLE 7.11: Classification accuracy results (%) on the KDEF dataset. The results are presented in ascending order for each validation method. The validation values are represented as follows 10-CV (10-fold cross-validation), Hold (holdout) and LODO (leave-one-dataset-out). Data agumentation was not applied by these authors.

Result	Validation	Augmentation	Study
88.27	10-CV	-	[302]
72.55	LODO	-	[294]
46.00	Hold	-	[71]
92.50	Hold	-	[221]
96.93	Hold	-	[222]

7.8 Conclusion

This chapter surveyed works on FER and CNN. Each major step when designing and implementing a CNN for FER has been discussed. Additional information is provided in Appendix A. This work serves as a guide to those who are new to the field and need to make a lot of fundamental decisions. Furthermore, the critical analysis of the existing literature has resulted in a list of several aspects which have not yet been explored and for which both novel and established researchers can address. It was observed that there is a significant lack of standards amongst published works and this survey aims at bringing this to the attention of researchers and that efforts are put in place to adhere to the guidelines provided so as to establish standards across future work. This will result in works which contain a greater amount of detail and allow for fair comparisons. There is no doubt that a great deal of work is yet to be done in the field of FER. CNNs have enabled researchers to achieve superior classification accuracy as opposed to the earlier traditional methods. Examples of traditional methods for extracting features include local binary pattern [125], gabor filters [159] and histogram of gradients [47] for which a SVM or nearest neighbour [10] classify would use. CNNs however can perform both feature extraction and classification.

With the distribution of new "in the wild" datasets, and researchers focusing on additional factors such as context within the images, the field will skyrocket with progress and systems which implement FER will become more robust. CNNs can result in a lot of trainable parameters which take a long time to train and furthermore require advanced hardware. With the correct effort, future CNNs will be able to achieve state-of-the-art performance with fewer computational requirements which can be used in useful areas such as medical applications. The next chapter proposes and investigates an algorithm to reduce the number of neural network weights associated with

models trained for FER.

As a final remark, authors are encouraged to no longer discuss traditional methods as a rationale for implementing a CNN. It is clear from the studies surveyed that CNNs outperform older traditional methods. It was also observed that a large number of studies cite the success of AlexNet as a motivation for implementing a CNN. Once again it is clear that AlexNet and other more recent CNNs achieve superior performance on image recognition tasks and authors are encouraged to use alternative rationales to drive their research.

Chapter 8

Evolutionary Facial Expression Recognition

8.1 Introduction

In the previous chapter we reviewed studies which applied CNNs to FER. It is clear from the results presented in that chapter that CNNs achieve state-of-the-art performance on FER tasks as opposed to other machine learning methods. Training CNNs often result in a large number of trainable parameters. This in turn implies that long training times are to be expected on limited hardware.

In this chapter, we explore a novel idea which attempts to optimise the predictive performance of CNNs for FER and simultaneously, reduce the number of neural network trainable parameters without compromising on the classification accuracy. We ask the following, can we achieve similar predictive performance when training a model on small image patches (extracted from an image) as opposed to the entire image of the face? Are certain facial features more discriminative for FER and can an evolutionary algorithm discover these areas? Can an evolutionary algorithm optimise the size of small image patches to reduce the input image size and consequently the number of trainable CNN parameters?

Section 8.2 introduces the proposed EA for FER. Section 8.3 presents the experimental setup and the results are discussed in section 8.4. Finally, section 8.5 concludes this chapter.

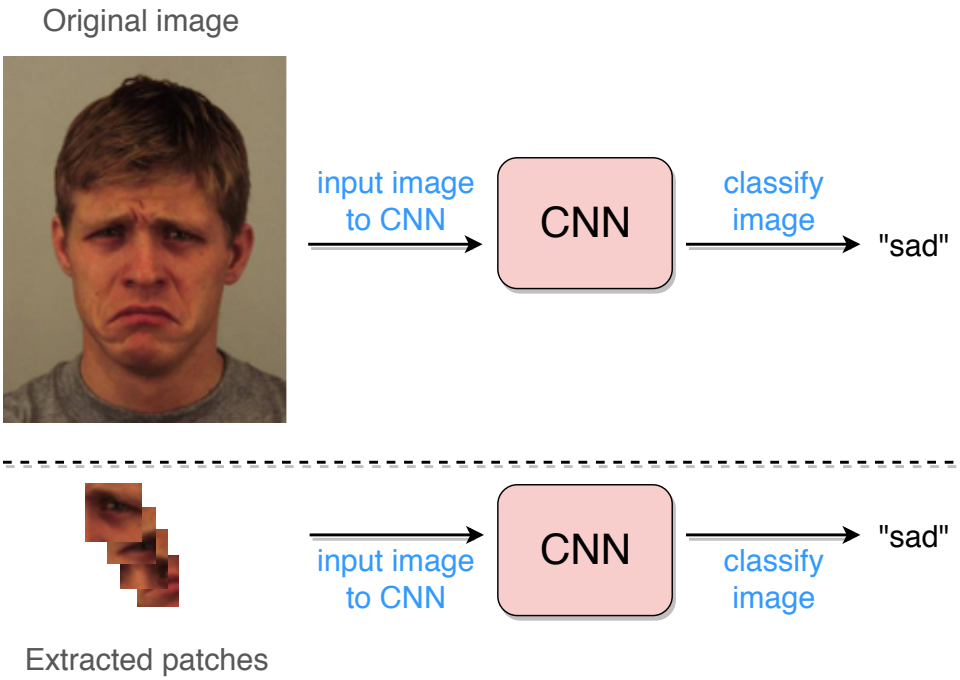

FIGURE 8.1: Illustrating the primary idea behind *EvoFER*. Instead of inputting an entire face image into a CNN to get the prediction, we propose to input patches to achieve similar predictive performance. The extracted imaged are stacked to form a new image.

8.2 Proposed Approach

Based on the literature there have been no prior attempts at using an evolutionary algorithm to extract patches from images with the objective of reducing the number of trainable CNN parameters and to retain (or improve) classification performance. We propose *Evolutionary Facial Expression Recognition* (EvoFER) in this section. *EvoFER* extracts a number of patches from the original image and stacks the patches to form a new image. Figure 8.1 illustrates the original image being input into a CNN and an example of four extracted patches being inputted into the same CNN. The resolution of the original image is 281×381 whereas the patches are each 50×50. Thus, the number of trainable CNN parameters using the original image is greater than the number of parameters when using the patches.

The objective function for *EvoFER* is multi-objective. Firstly, *EvoFER* attempts to reduce the number of trainable parameters as opposed to the number of parameters which would be used when training on the full image. The second objective is to achieve the highest possible classification accuracy using the patches as opposed to using the full image. Further details about the

objective function are presented in subsection 8.2.5. The following subsections describe *EvoFER*.

8.2.1 Chromosome

EvoFER chromosomes contain 2 fixed genes and then allows for a number of pair of genes to be added. The first two genes denote values α and β which represent the width and height of the patches to be extracted. The remainder of the chromosome denotes (x, y) pairs. Each x and y pair are the coordinates of the top left point of the corresponding patch to be extracted from the original image. The coordinates x and y are relative to the location of the nose. The nose is used as a reference point and was obtained using OpenCV and DLIB [1]. The patch is extracted by obtaining the coordinate (x, y) and using α and β to obtain the entire patch. A chromosome thus encodes the location and size of the patches to be extracted from the original images. A chromosome must have at least one (x, y) pair. User-defined parameters specify the maximum number of (x, y) pairs allowed in each chromosome. The patches are stacked on top of each other vertically which in turn creates a new image. The order of the patches is not important. Figure 8.2 illustrates an example of a chromosome with two patches. The width of the patches to be extracted is 10 and the height is 20. The location of the top left corner of each patch to extract relative to the coordinates of the nose are (10, 30) and (-10, -5). The figure illustrates the exacted patches.

8.2.2 Initial Population Generation

We use the standard initial population generation algorithm. We propose algorithm 16 to generate the *EvoFER* chromosomes. This algorithm is executed a number of times based on the number of chromosomes to create. A random value for α and β is assigned to each chromosome based on a corresponding bound which is pre-defined by the experimenter. The values of α and β are not modified during the evolutionary process.

Upon the creation of a chromosome, the number of patches is randomly assigned based on a user specified bound. For each patch to be created, a random value for x and y is created. These values are randomly generated based on the image width and height as is illustrated from lines 11 to 14 in algorithm 16. This was done so that the patches remain as closely as possible within in the bounds of the image. Initially, the evolutionary algorithm can

[1] http://dlib.net/imaging.html

Algorithm 16: Creating an EvoFER chromosome.

input: min_alpha: minimum size for alpha
input: min_beta: minimum size for beta
input: max_alpha: maximum size for alpha
input: max_beta: maximum size for beta
input: min_patches: minimum number of patches allowed
input: max_patches: maximum number of patches allowed

```
1  begin
2      Initialise an empty chromosome.
3      alpha ← random[min_alpha, max_alpha ]
4      beta ← random[min_beta, max_beta ]
5      patches ← random[min_patches, max_patches ]
6      for i ← 0 to patches do
7          X ← GenerateX(alpha)
8          Y ← GenerateY(beta)
9          Append (X, Y) to chromosome
10     return chromosome.
11     Function GenerateX (alpha)
12         return random[−(image_width/2 − alpha),
               image_width/2 − alpha]
13     Function GenerateY (beta)
14         return random[−(image_height/2 − beta),
               image_height/2 − beta]
```

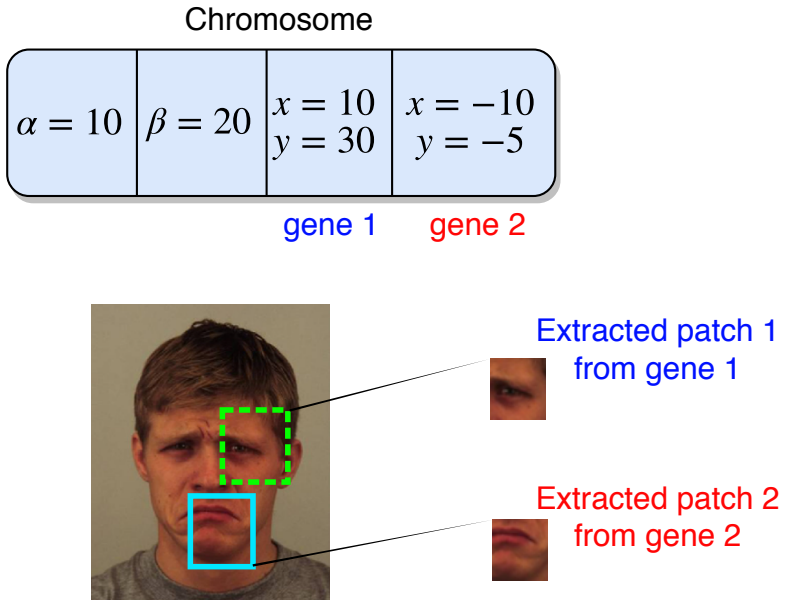

FIGURE 8.2: Illustrating a *EvoFER* chromosome which extracts two patches. The width of the patches to extract is 10 and the height is 20. The figure illustrates how the chromosome is applied to an image to extract a patch. The green dotted line is the first patch and the solid blue line is the second patch. The genes encode the location of the top left corner of the patches to extract. These coordinates are relative to the coordinates of the nose which is used as an anchor point. The location of the nose is obtained using OpenCV and DLIB.

create patches which contain redundant pixels (for example the background, hair, clothing). We do not bias the algorithm towards initialising on patches of interest, such as the mouth or eyes.

8.2.3 Mutation

We propose two mutation operators to traverse through the search space. The first is a variation of the standard mutation operator and the second is tailored to the problem domain. When the mutation function is applied one of four things can be executed depending on the number of patches in the chromosomes, namely *adding*, *changing*, *removing* or *shifting*. The shift operation is a novel method which we describe below. Changing and shifting is always allowed, however adding and removing is only allowed under the following conditions:

- if the number of patches is less than the maximum allowed then a patch can be added
- if the number of patches is greater than the minimum allowed then a patch can be removed

Adding a new patch simply generates a value for x and y in a similar way to the functions presented in algorithm 16. The new patch is appended to the chromosome. Removing a patch consists of randomly selecting a (x, y) pair to be removed. Changing a patch randomly selects one and then replaces the (x, y) pair with new values.

Algorithm 17 presents the pseudocode for applying shift mutation to a chromosome. A patch is randomly selected from the chromosome and then the x and y values are extracted from the patch. The pair is then perturbed by values ranging between $[-alpha, alpha]$ and $[-beta, beta]$. This enables the algorithm to shift the patch in a random direction. The standard mutation operator allows the algorithm to take a large jump in the pixel space, whereas shift mutation restricts the jump. Figure 8.3 shows an example of the shift mutation and standard mutation operators being applied to a *EvoFER* chromosome.

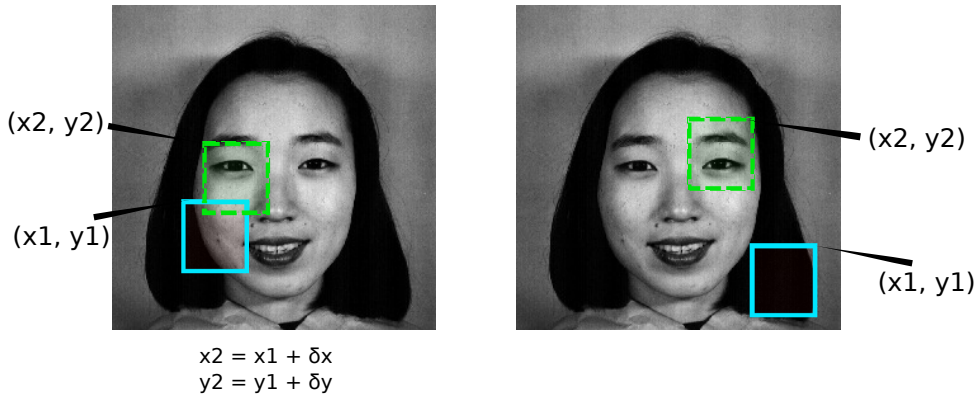

FIGURE 8.3: Illustrating shift mutation on the left and standard mutation on the right. In shift mutation the patch is shifted, using values of δ, within the associated bounds of $[-alpha, alpha]$ and $[-beta, beta]$. For the standard mutation the patch can jump anywhere.

8.2.4 Crossover

The standard crossover operator is applied to the (x, y) pairs between two parent chromosomes. Specifically, let (x_1, y_1) and (x_2, y_2) be a pair of coordinates within parents P_1 and P_2. Then offspring C_1 is created by copying all of the genes in P_1 however, x_1 is replaced with x_2 and similarly, y_1 is replaced with y_2. Chromosome C_2 is created by copying all of the genes in P_2 however, x_2 is replaced with x_1 and similarly, y_2 is replaced with y_1. The offspring are evaluated and the one with the highest fitness is returned.

Algorithm 17: Applying shift mutation.

input: alpha: patch width
input: beta: patch height
input: chromosome: the chromosome which will be mutated

```
1  begin
2      mutation_patch ← randomly select a patch from chromosome
3      x_i ← x value from mutation_patch
4      y_i ← y value from mutation_patch
5      delta_x ← GenerateDeltaX(alpha)
6      delta_y ← GenerateDeltaY(beta)
7      x_i ← x_i + delta_x
8      y_i ← y_i + delta_y
9      return chromosome.
10     Function GenerateDeltaX (alpha)
11         return random[−alpha, alpha]
12     Function GenerateDeltaY (beta)
13         return random[−beta, beta]
```

Figure 8.4 shows an example of the crossover operator being applied to two *EvoFER* chromosomes. The second patch from each parent chromosome is swapped to create the two offspring.

8.2.5 Chromosome Evaluation

Before executing the evolutionary process, we run a CNN model M on the image dataset and record the validation accuracy and the number of neural network trainable parameters. Once the model is trained we apply the model to the test images and record the test performance – we denote this as the baseline model.

Each chromosome C_i is evaluated on every image X_j in the dataset. When determining the fitness for C_i, the corresponding image patches E_k are extracted from X_j based on the number of (x, y) pairs in C_i. All the extracted patches E_k are stacked upon each other which produce a new image N_j. That is, each chromosome will be applied to each X_j which creates a corresponding image N_j. The new images N_j are input into the CNN model M. When computing the fitness score, we make use of the baseline validation accuracy and number of trainable CNN parameters. This is done to assign a fitness score that compares relative performance of chromosomes to the baseline CNN with the ultimate goal of improving the chromosome accuracy and reducing the number of parameters.

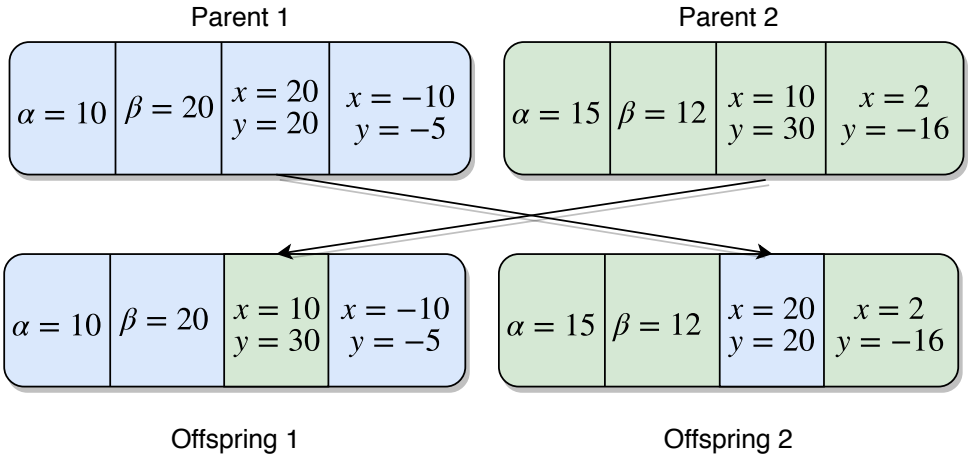

FIGURE 8.4: Illustrating the crossover operator swapping the second patch between both parent chromosomes. All other genes within the parents are not swapped when creating the offspring. The crossover operator can swap any of the patches between the parents.

We propose the fitness function presented in equation 8.1. The function considers both the effect of the validation accuracy and the number of trainable parameters between the network produced on images N and the images X. The objective is to maximise the fitness. When the validation accuracy of the chromosome is larger than the baseline then $\frac{S_c}{S_b}$ is a larger number. Similarly, when the number of parameters obtained by the network as a result of the chromosome is smaller than the number of parameters obtained by the baseline then $\frac{P_b - P_c}{P_b}$ is a larger number. We allow a parameter W_S to fine-tune the weight allocated to the validation accuracy. This way, the experimenter can decide on the important of the validation as opposed to the number of parameters in advance. The value of W_S was determined through trial runs.

$$\text{Fitness (chromosome)} = e^{\left\{W_S\left(\frac{S_c}{S_b}\right) + \left(\frac{P_b - P_c}{P_b}\right)\right\}} \tag{8.1}$$

where

W_S = the weight of the validation accuracy
S_c = the validation accuracy for the chromosome
S_b = the validation accuracy for the baseline network
P_c = the number of trainable parameters for the chromosome network
P_b = the number of trainable parameters for the baseline network

We provide an example of the fitness computation of a baseline and chromosome. Suppose that a CNN is trained on a FER dataset and the validation accuracy averaged over R runs is computed to be 0.65. Assume that

the number of trainable parameters for the network is 12,189,447. Now assume that the proposed algorithm is executed and that the validation accuracy of a chromosome is 0.31 and that the number of trainable parameters is 123,063 (a smaller value is obtained since the input images consist of smaller patches than the original images). Then we have $\frac{0.31}{0.65} =\approx 0.48$ and $\frac{12{,}189{,}447-123{,}063}{12{,}189{,}447} \approx 0.99$. Let $W_S = 5$. The final calculation for the fitness of the chromosome is $e^{(5*0.48+0.99)} \approx 29.67$. Since the objective is to maximise the fitness then a larger value denotes a better chromosome.

For each chromosome, we execute a CNN and allow it to train on the transformed images (made up of extracted patches). We use Keras and Tensorflow to achieve the training of the CNN. The pipeline of contrusting patches and training the CNN is illustrated in figure 8.5.

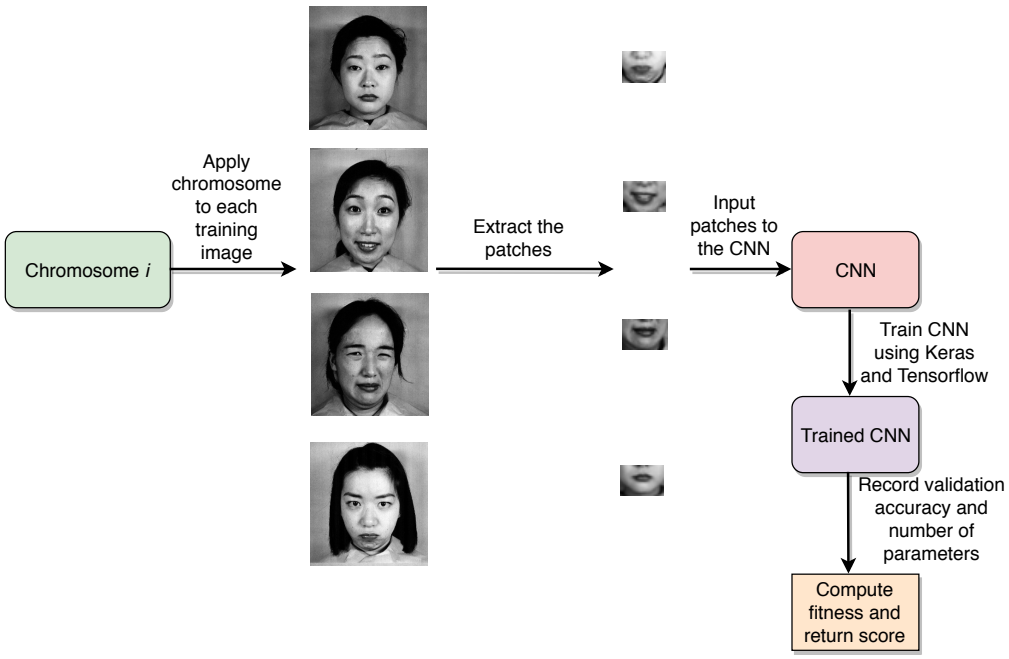

FIGURE 8.5: Illustrating the *EvoFER* pipeline. Each chromosome is applied to the training data which in turns generates a new set of training images made up of patches. The patches along with the associated target classes are input into the CNN. The CNN, inspired by the findings from chapter 7, is trained using Keras and Tensorflow. Once the training is complete, the validation accuracy and number of parameters are used in the computation of the fitness. The process is repeated for each chromosome in the population.

8.3 Experimental Setup

In this section we describe the experimental setup which was used to evaluate the performance of *EvoFER*. The rationale behind the decisions made to

guide the experiments were based on the review discussed in the previous chapter.

8.3.1 Datasets

To evaluate the performance of *EvoFER* we conducted a number of experiments on the datasets listed below. Details regarding these datasets are available in the previous chapter in section 7.3. These datasets were selected since they are commonly used in the literature and represent different characteristics (gender, age, ethnic diversity and image resolution).

- JAFFE [158]
- KDEF [155]
- MUG [6]
- RAFD [134]

8.3.2 Pre-processing

Pre-processing was shown to improve classification performance as was discussed in section 7.4. We implemented histogram equalisation on each of the images. Additionally, we converted each image into greyscale as the additional information available in colour images does not impact the performance of FER. To achieve this, we made use of OpenCV as it is commonly used in literature as discussed in section 7.4.

8.3.3 Data Augmentation

The application of CNNs often require large datasets to enable good predictive performance; this is discussed in greater detail in section A.1 and provides a rationale for our decisions. We implemented a number of augmentation techniques. For each training image we retain the original image and supplement the dataset with augmented images using the following techniques:

- horizontal flipping
- blurring
- noise

- rotate by -5 and -10 degrees
- rotate by 5 and 10 degrees

These augmentation techniques were only applied to the training images. The test images were kept in their original form. For each training image we generated an additional 9 images by applying two levels of blurring and noise. These were achieved by using *imgaug*[2] – a software package for image augmentation. Table 8.1 presents the number of images which were used for training for each dataset after we applied the augmentation techniques. Figure 8.6 presents the augmentation techniques applied to an image.

TABLE 8.1: Number of training images used for each dataset after the images were augmented. The images were augmented using rotation, blurring, random noise and horizontal flipping.

Dataset	Training images after augmentation
JAFFE	1,910
KDEF	6,860
RAFD	10,152
MUG	2,800

8.3.4 Network Architecture

Figure 8.7 presents the CNN architecture which we propose to use for the *EvoFER* experiments. The architecture was inspired by our findings from chapter 7. More specifically, the reviewed works used on average 3 convolutional layers with max pooling in between them, and two fully connected layers at the end of the network. The ReLU activation function was used the most frequently in all layers except for the last where softmax was used. We thus used the literature to guide our decisions, and furthermore various modifications of the architecture were explored using preliminary runs on the JAFFE dataset by varying the depth, activation function and parameters.

The proposed architecture consists of three convolutional and max pooling layers, along with two fully connected layers. Dropout was applied after each of the layers except for the last one. The ReLU activation function was used for all the layers except for the last fully connected layer whereby the softmax function was used. It would also be possible to optimise the network

[2]https://github.com/aleju/imgaug

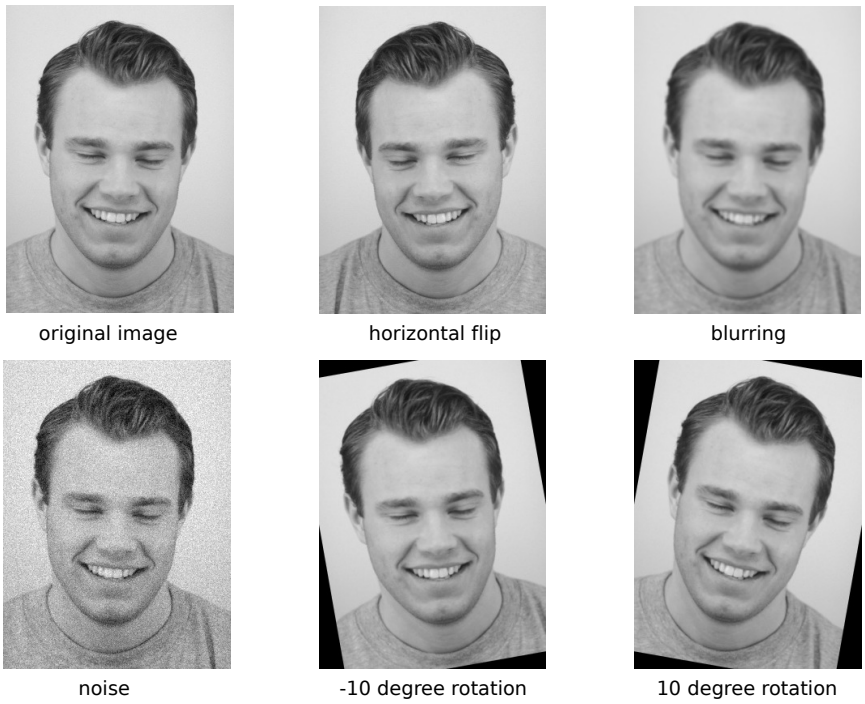

FIGURE 8.6: Illustrating the different augmentation techniques which were used to increase the number of training images.

using our previous approach, *EDEN*, as described in chapter 4, however, we chose to use the findings from the literature guide our decisions. Our findings in section A.3 reveal that using an ensemble will result in superior classification performance, however, we wanted to examine the effect of using patches as input to enhance the performance.

We run two sets of experiments. In the first, we run *EvoFER* using the literature inspired network and use the patches as input to the CNN. In the second set of experiments, we run the same literature inspired network and use the full image as input to the CNN. We denote experiments conducted on the full images as the *baseline*.

8.3.5 Training and testing

In section A.5 we discuss that it is more correct to evaluate the performance of FER algorithms when the train and test split are subject independent. That is, the subjects that are used for training do not occur in the test data. We implement this approach in our experiments. To split the training data into a training and validation set, we randomly set 30% of the data for validation and the remaining images are used for training. The images not used for training and validation are used for testing. We repeat the execution of *EvoFER* ten times and average our results. This is conducted for both the

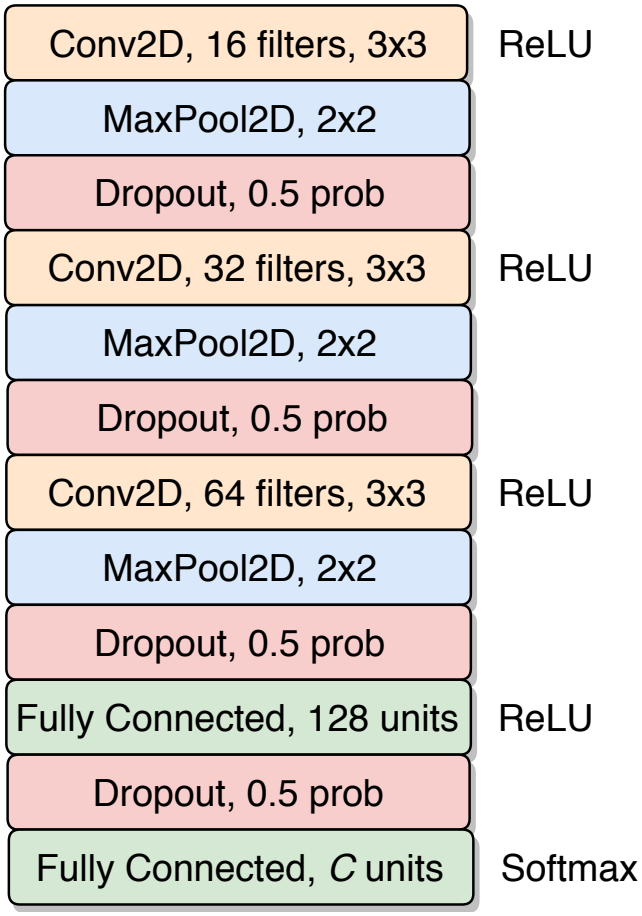

FIGURE 8.7: The CNN architecture which is proposed for the experiments. This network was inspired by the findings from chapter 7. 'Conv2D' denotes 2D convolution and 'MaxPool2D' denotes 2D max pooling. The associated parameters are listed. In the case of convolutional layers we provide the number of filters and the kernel size. For max pooling we provide the kernel size. For dropout we provide the keep probability. For fully connected layers we provide the number of units. The last fully connected layer takes on a value of 'C' for the number of units as this is dependant on the number of emotion classes in each dataset.

evaluation of the baseline and for the evolutionary process. The choice of optimiser was selected based on our previous work from chapters 4.1 and 4. We selected the adam optimiser and a batch size of 8.

8.3.6 *EvoFER* Parameters

Table 8.2 provides the details of the parameters associated with the evolutionary algorithm. The parameters associated with the training of the CNN are presented in table 8.3. Finally, table 8.4 presents the parameters associated with the initialisation of the chromosomes during the initial population generation and mutation operation. These parameters were chosen by performing a search on a number of values which obtained from the literature

review in chapter 7 and by performing a random hyper-parameter search.

TABLE 8.2: Parameters associated with the evolutionary algorithm.

Parameter	Value
Population size	100
Tournament size	7
Crossover percentage	50
Mutation percentage	50
Generations	15
Fitness function, W_S	5

TABLE 8.3: Parameters associated with the CNN.

Parameter	Value
Number of epochs	10
Optimiser	Adam
Batch size	8

TABLE 8.4: Additional parameters used in the initial population generation and mutation operator.

Parameter	Value
min_alpha	30
max_alpha	50
min_beta	30
max_beta	50
min_patches	1
max_patches	4

8.4 Results and Discussion

Table 8.5 presents the average test classification accuracy results. The baseline accuracy (literature inspired CNN model with original images) and *EvoFER* accuracy (the same literature inspired CNN model with extracted patches from best chromosome) are presented. The CNN architecture and hyper-parameters were the same for both results. The findings reveal that on all of the datasets *EvoFER* was able to outperform the same CNN model which had been trained on the original images. The smallest improvement in accuracy was observed in the KDEF dataset with an improvement of 1.3%. The largest improvement in classification accuracy was obtained on the MUG dataset with a value of 20.6%. This indicates that given a fixed architecture,

TABLE 8.5: Test classification accuracy (%) for the CNN network on the original images and performance when using *EvoFER*. The standard deviation is presented in parentheses. In both cases the same CNN architecture and hyper-parameters are used based on the findings from chapter 7. *EvoFER* inputs patches extracted from the original image. The baseline is the same CNN, however, the full image is used as input. The findings reveal that the performance is superior when *EvoFER* is used.

Dataset	Baseline Accuracy	EvoFER Accuracy	Difference
KDEF	61.9 (1.6)	63.2 (2.9)	1.3
JAFFE	60.0 (4.7)	75.5 (4.4)	15.5
RAFD	75.4 (3.4)	80.4 (2.3)	5.0
MUG	45.8 (3.8)	66.4 (5.0)	20.6

an EA can be applied in such a way to extract patches from an image and to train the CNN on those patches and achieve competitive performance.

Why did *EvoFER* achieve superior performance to the baseline method? It is hypothesised that *EvoFER* is able to obtain better predictive performance given that it inputs extracted patches to the CNN and consequently the number of neural network parameters are reduced. The average number of neural network trainable parameters obtained in the experiments are presented in table 8.6. For each dataset the number of neural network parameters for *EvoFER* is significantly less than for the baseline CNN. The baseline CNN inputs larger images of the faces compared to *EvoFER* which inputs smaller extracted patches. The percentage difference achieved by *EvoFER* for KDEF, JAFFE, RAFD and MUG are 88%, 97%, 97% and 98% respectively. The largest difference in parameters was obtained on the JAFFE dataset with a reduction of over 7 million parameters. It is observed on the JAFFE dataset that the standard deviation for the number of parameters is zero, which indicates that in each execution of the algorithm, *EvoFER* extracted the same number of patches of the same size. Despite the large standard deviation on the KDEF dataset the number of parameters is still distant from the baseline parameters.

Figures 8.8 to 8.11 illustrate the patches which were extracted from the best chromosome in generation 0 and the last generation for the various datasets. Each row presents the patches which were extracted from a random test example.

Figure 8.8 illustrates the patches for the JAFFE dataset. In both cases the best chromosome from the initial and final population extracted two patches. For generation 0, both patches were around the left eye region. This of course is sub-optimal as that does not provide the CNN with enough information to discriminate between the various emotion classes. The best chromosome

TABLE 8.6: The average number of trainable neural network parameters when using the original image and *EvoFER* extracted patches. The standard deviation is presented in parentheses. The percentage difference in parameters is shown in the last column. These findings reveal that *EvoFER* can significantly reduce the number of trainable neural network parameters.

Dataset	Baseline Parameters	EvoFER Parameters	Difference	%
KDEF	1,218,944	145,741 (15,782)	1,073,203	88.0
JAFFE	7,397,127	155,543 (0)	7,241,584	97.9
RAFD	2,152,520	45,528 (6,476)	2,106,992	97.9
MUG	2,574,279	47,027 (6,345)	2,527,252	98.2

extracts patches around the left eye region and includes the eyebrow. The second patch extracts pixels around the center and right region of the mouth.

Figure 8.9 illustrates the patches for the KDEF dataset. In generation 0 the chromosome extracts four patches. The first and last patch are redundant as the first extracts pixels around the hair and the other around the neck. This does not help discriminate between expressions. The second and third patch are better since they extract pixels between the eyes and near the left corner of the mouth. In the case of the best chromosome in generation 15, two patches are extracted. Combined, they extract pixels near the left eye and half of the mouth.

The extracted patches for the MUG dataset are presented in figure 8.10. The best chromosome from generation 0 extracts two patches, one near the left ear and the other contains pixels around the nose and mouth. The first patch does not assist in distinguishing between expressions. The best chromosome from the final generation extracts three patches. The second and the third patch are similar since they both extract pixels around the left eye. The first patch extracts pixels around the center of the mouth.

Finally, figure 8.11 presents the patches extracts for the RAFD dataset. For generation 0, the best chromosome extracts four patches of which the first, second and last contain redundant information. Those patches cannot help the CNN distinguish between the various expression classes. The third patch extracts pixels near the right eye and eyebrow. For the last generation, the best chromosome extracts two patches. The first contains pixels around the mouth and chin, and the second contains pixels capturing the mouth, nose and eyes (no information about the eyebrows).

From the figures it is observed that the best predictive performance is achieved when pixels around the eyes and mouth are obtained. The best chromosome from the each of the last generations extracted two patches on 3 of the 4 datasets. We can deduce that those areas are the most salient when

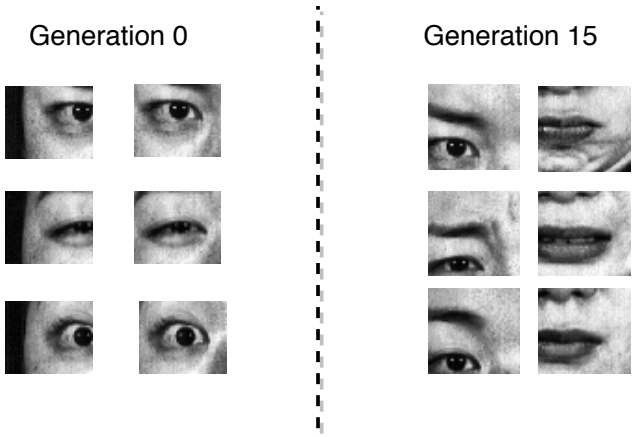

FIGURE 8.8: Illustrating the patches extracted from the best chromosome in generation 0 and in the last evolutionary generation. Each row presents the patches which were extracted for randomly selected test examples in the JAFFE dataset. The best chromosome from generation 0 extracted two patches. The patches were both around the left eye region. The best chromosome from the last generation extracted two patches, one around the left eye region and the other around the mouth.

distinguishing between expressions on the datasets examined. We did not have sufficient computing resources to compare the results to state-of-the-art CNNs which use much deeper network with a larger number of parameters. The findings do reveal that on a fixed architecture which was inspired by the literature, *EvoFER* is able to enhance the performance by using patches as input.

What is the processing time for *EvoFER* given that the nose has to be located prior to the application of the chromosome? We examined the time it took to perform the necessary pre-processing steps for *EvoFER* and in the case of the baseline CNN. These findings are presented in table 8.7. The time, in seconds, is presented for the two methods to pre-process and predict the expression for 10 images. In the case of KDEF, RAFD and MUG, *EvoFER* took approximately twice as long as the baseline method. This is because *EvoFER* has the extra overhead of needing to locate the nose to establish a reference point for the chromosome. *EvoFER* took less time than the baseline on the JAFFE dataset. It can be hypothesised that this is the case since the resolution of the images in the JAFFE dataset is much smaller than the other datasets. It would be of interest to reduce the resolution of images in the other datasets to determine if this could result in faster execution times for *EvoFER*. Despite the fact that *EvoFER* has additional overhead, the processing time is not sufficiently large and could thus still be implemented in a real-world setting.

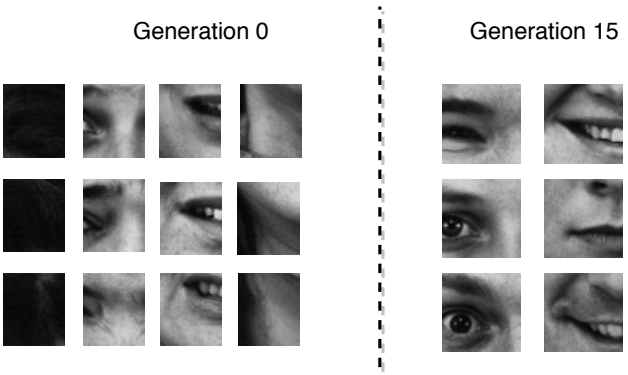

FIGURE 8.9: Illustrating the patches extracted from the best chromosome in generation 0 and in the last evolutionary generation. Each row presents the patches which were extracted for randomly selected test examples in the KDEF dataset. The best chromosome from generation 0 extracted four patches. The patches extracted pixels around the hair, between the eyes, the left corner of the mouth and around the neck. The best chromosome from the last generation extracted two patches. The first was around the left eye and the second was around the left center of the mouth.

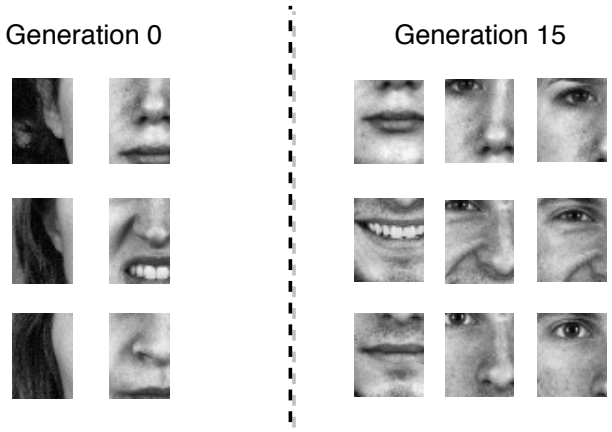

FIGURE 8.10: Illustrating the patches extracted from the best chromosome in generation 0 and in the last evolutionary generation. Each row presents the patches which were extracted for randomly selected test examples in the MUG dataset. The best chromosome from generation 0 extracted two patches. The first patch was around the left ear and hair and the second around the nose and mouth. The best chromosome from the last generation extracted three patches around the mouth, nose and left eye.

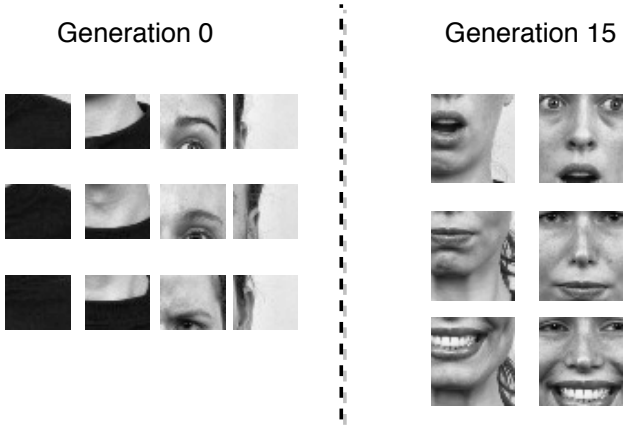

FIGURE 8.11: Illustrating the patches extracted from the best chromosome in generation 0 and in the last evolutionary generation. Each row presents the patches which were extracted for randomly selected test examples in the RAFD dataset. The best chromosome from generation 0 extracted four patches. Two patches were extracted around the t-shirt, the third around the right eye brow and eye, and the last around the right ear. The best chromosome from the last generation extracted two patches. The first was around the mouth and the second contained the mouth, nose and eyes.

TABLE 8.7: Average time taken (seconds) to process 10 images using *EvoFER* and the baseline. The results are averaged over 10 executions. The standard deviation is shown in parentheses. The baseline method applies the pre-processing steps described in section 8.3.2. *EvoFER* applies the same pre-processing steps and additionally locates the nose to enable the chromosome to extract patches. In the case of KDEF, RAFD and MUG, the baseline CNN is roughly twice as fast than *EvoFER*. *EvoFER* is faster than the baseline on the JAFFE dataset.

Dataset	**EvoFER**	**Baseline**
JAFFE	0.27 (0.01)	0.29 (0.01)
KDEF	0.42 (0.01)	0.18 (0.01)
RAFD	0.28 (0.01)	0.14 (0.01)
MUG	0.23 (0.01)	0.13 (0.01)

8.5 Conclusion

This chapter proposes a novel evolutionary algorithm to extract patches from images with the goal of transforming images into a more compact one whilst retaining the predictive performance (or increasing it) and to reduce the number of trainable parameters. We propose a chromosome representation which allows the algorithm to encode locations relative to the nose. The chromosomes also encode the width and height of the patches to be extracted at each location. We introduce a fitness function which combines the relative performance of each chromosome to a baseline execution. The baseline execution consists of running a CNN on the original images. The multi-objective fitness function attempts to optimise the validation accuracy and number of trainable parameters. To enable us to interface each chromosome to the training of a CNN we use Keras and Tensorflow.

We evaluated *EvoFER* and the findings revealed that it can achieve superior performance to the exact CNN architecture trained on the entire image. Furthermore, the findings revealed that *EvoFER* can reduce the number of trainable parameters, on average, up to 95%. *EvoFER* was able to optimise the search for optimal patch locations and size to enable the CNN to distinguish between the various expressions (without any insights hardcoded into the algorithm). Initially *EvoFER* could select patches which did not contain parts of the face at all. In several cases there was patches which were located near people's hair. Through the evolutionary process, *EvoFER* extracted patches generally around the eyes and mouth which enabled the CNN to achieve better predictive performance.

In future work we will explore the combination of *EDEN* (see chapter 4) and *EvoFER* so that the optimal network architecture is evolved along with the patches to be extracted. Furthermore, additional experiments across various fixed architectures will be conducted to futher evaluate *EvoFER*.

Chapter 9

Conclusion

This chapter provides a summary of the findings which were achieved in this thesis. Furthermore, additional future work for each of the objectives is presented. The overall objective addressed in this thesis was to evaluate how EAs could be adapted to solve various modern machine learning problems. Six objectives were addressed, and these were introduced in greater detail in section 1.1. A brief summary of the objectives is listed below. The remainder of this chapter discusses how each of the objectives have been attained.

1. **Objective 1:** Automatic identification of classification and regression problems.
2. **Objective 2:** Evolutionary algorithm to evolve deep neural network architectures.
3. **Objective 3:** Using an evolutionary algorithm to perform text sentiment analysis.
4. **Objective 4:** Compressing text for sentiment analysis.
5. **Objective 5:** Review of studies using convolutional neural networks for facial expression recognition.
6. **Objective 6:** Using an evolutionary algorithm to reduce the number of neural network parameters for facial expression recognition.

Objective 1 - Automatic Problem Identification

Through the application of neuro-evolution, *API*, as discussed in chapter 4.1, was proposed for which the algorithm can identify whether a dataset represents a classification or a regression problem. Furthermore, *API* evolves a neural network architecture and automatically selects a suitable loss function. *API* was tested on 16 different classification, regression and sentiment

analysis datasets with up to 10,000 features and up to 17,000 unique target values. *API* achieved an average accuracy of 96.3% in identifying the problem type without hardcoding any insights about the general characteristics of regression or classification problems. This study is an initial attempt at removing the practitioner from a section of the machine learning process. The practitioner would have to decide on the problem type and make decisions about the number of units in the last layer and loss function, *API* overcomes this issue.

Objective 2 - Evolving Neural Network Architectures

Goodfellow *et al.* state that "the ideal network for a task must be found via experimentation guided by monitoring the validation set error" [85]. Following the previous objective, a neuro-evolutionary method was proposed to further remove the human practitioner from the neural network architecture design. In chapter 4, *EDEN* was proposed with the objective of evolving the architecture and associated hyper-parameters for various problems. The findings reveal that EDEN evolves simple yet successful architectures built from embedding, 1D and 2D convolutional, max pooling and fully connected layers along with their hyper-parameters such as the number of filters, filter size, number of fully connected units and the embedding dimension. *EDEN* was evaluated on seven image and sentiment classification datasets, and the results demonstrate that the proposed method can produce competitive results. The study was a novel attempt at applying neuro-evolution to the creation of 1D convolutional networks for sentiment analysis. To achieve this, *EDEN* evolved the embedding dimension in the embedding layer. It would be useful to extend *EDEN* so that it can evolve more complex neural networks, such a recurrent neural networks [249] and generative adversarial neural networks [87].

Objective 3 - Evolutionary Text Sentiment Analysis

This objective dealt with proposing an evolutionary algorithm for sentiment analysis for which the focus was not to use the algorithm as a feature selection method, but rather, to enable it to predict sentiments for textual data. In chapter 5, *GASA* was proposed for which the genes in the chromosome

encoded either a sentiment or an amplifier along with corresponding values for the words. The algorithm had to optimise the classification-value pair. *GASA* was compared against eight commercial and non-commercial sentiment analysis algorithms to evaluate its performance. The approach was tested on four review datasets and the results revealed that *GASA* was able to outperform certain commercial algorithms. One advantage of *GASA* is that the algorithm can grow a sentiment dictionary which can be reused or further improved upon. The experiments suggested that if a particular word appeared a large number of times in the training data set, then the proposed method is likely to correctly classify its sentiment, and thus enable the construction of a sentiment lexicon. *GASA* performed well when trained on short sentences. A limitation of *GASA* is the long computational time and that the genetic operators manipulate a small section of the chromosome thus rendering the traversal of the search space slow. An extension of *GASA* is to encode *n-grams* for which the gene encodes more than one word, namely, n, words.

Objective 4 - Evolutionary Text Compression for Sentiment Analysis

When investigating the previous objective, it was observed that the length of certain text reviews was long. Thus, an objective was formulated to propose a method that can compress text in such a way that the overall sentiment which is being expressed is not affected. Chapter 6 proposed *PARSEC*, an evolutionary algorithm which makes use of parts-of-speech to compress textual data. The algorithm learns which sequence of parts-of-speech can be removed from a sentence whilst attempting to preserve the sentiment. *PARSEC* was applied to eight commercial and non-commercial sentiment analysis algorithms to investigate the effect of the compression. This investigation was conducted on twelve sentiment datasets. The findings reveal that a compression rate of 50% was possible with a loss of -1.2% in classification accuracy. Furthermore, with a loss of -3.2% in classification accuracy, a compression rate of 75% was attained. The results indicate that evolving chromosomes which encode a sequence of parts-of-speech – for which the chromosome deletes words – can successfully be applied as a text compression method. The associated loss in accuracy depends on the amount of compression the experimenter wants. In this approach, a simple heuristic rule was applied

during the evolutionary process to evaluate the sentiment of the compressed sentences. This was done so that the fitness evaluation was rapid. It would be of interest to incorporate a pre-trained neural network based sentiment classifier into *PARSEC* and investigate the resulting performance.

Objective 5 - Convolutional Neural Networks and Facial Expression Recognition

This objective consisted of reviewing studies which used convolutional neural networks for the task of image facial expression recognition. A comprehensive review is presented in chapter 7 and appendix A. The review studied works published until early 2018. The review was split into several sections whereby each section discussed a primary aspect which a practitioner should consider when implementing such a neural network. The review serves as a guide for those who are new to the field and want to learn the various aspects which need consideration. To achieve this, a number of recommendations are presented to the reader. Furthermore, the review also presents ideas for future research in image facial expression recognition. It would be of interest to extend this work by reviewing studies on video facial expression recognition as this is a natural extension of the image based recognition.

Objective 6 - Evolutionary Facial Expression Recognition

Following from the previous objective, chapter 8 proposed a novel evolutionary algorithm to enable a neural network to reduce the number of trainable parameters without compromising on the predictive performance of a facial expression classifier. The chapter introduced *EvoFER*, a multi-objective evolutionary algorithm which extracts patches from an image and uses those patches to train a convolutional neural network. The approach was evaluated on four facial expression datasets and the findings revealed that *EvoFER* was able to reduce the number of neural network parameters by up to 95% on average. Furthermore, the predictive performance achieved on the extracted patches was superior to using the entire face image. It would be of interest to extend the approach to video facial expression recognition, for example a long short-term memory network could be trained and the patches used as input.

These various results show that EAs can play a powerful role in enhancing deep learning by removing the need for human dabbling in the design of the architectures and significantly reducing the complexity of the resulting networks. It will be interesting to see how far this trend will continue in the future, and to what extent EAs will be used alongside future breakthroughs in machine learning.

Appendix A

Additional Material on Facial Expression Recognition and Convolutional Neural Networks

Section A.1 describes data augmentation techniques which can be applied to generate additional images which in turn can increase the classification accuracy given that CNNs often require large datasets. Section A.2 highlights computational hardware requirements for training the CNNs. Ensembles achieves greater classification accuracy than a single CNN. These studies are reviewed in section A.3. Transfer learning has shown to achieve good results and helps one save time by not training a CNN from scratch, this is discussed in section A.4. Correctly reporting on results and conducting fair comparisons is an important consideration, this is reviewed in section A.5. Section A.6 combines the findings and critical analysis of the literature. The section provides the reader with recommendations on how to implement a CNN for FER. The section also provides ideas for future research.

A.1 Data Augmentation

Deep learning models require a large amount of representative training data to perform well. Training CNNs from scratch (i.e. random initialisation of weights) on small data sets can result in models that perform poorly. This issue is certainly also the case for FER. To overcome this issue, data augmentation techniques can be implemented to create additional training data. Augmenting the data also assists in dealing with the issue of overfitting [225].

The most commonly used techniques from the studies surveyed are listed in table A.1. Horizontal flipping (mirroring) was the most wisely used technique. This is a simple technique that allows one to double the amount of

Augmentation Technique	Studies
Horizontal flipping	30
Rotation	13
Cropping	12
Zooming	6

TABLE A.1: Commonly used data augmentation techniques. Horizontal flipping was reported the most frequently. This method does not have an associated parameter thus it is easy to implement and provides a rapid way to increase the amount of data by two folds.

training data using a simple flipping operation. [267] implemented both horizontal and vertical flipping. Assuming that test images are aligned such that the eyes are at the top and the mouth is at the bottom of the image, then applying a vertical flip on the training data will not augment the data in a representative manner. Examples of studies that used horizontal flipping include [298, 9, 51, 18, 113]. Figure A.1 illustrates an image which has been flipped horizontally and rotated.

FIGURE A.1: Illustrating the original image (top) along with two images which were augmented using horizontal flipping (bottom left) and 15 degree anti-clockwise rotation (bottom right). The images are from the CK+ dataset.

Rotation and cropping were also widely used. Rotation requires more attention to detail during implementation given that the rotation angles must be specified. Assuming that face alignment is performed on training and test images, then augmenting the data with a large rotational angle will generate training images which will not be as representative as the aligned test images. Most studies did not report on the angles which were used and there was variation in the values from the studies which did report the angles, for example see [2, 118, 119]. Two studies rotated by 45 degrees [225, 53], [7]

used a value of 30, and [215, 298] used 10 degrees – rotations were all done both clockwise and anticlockwise. Smaller angles (1, 2, 3 degrees) were used in [18], the authors do not investigate whether such small angles provide enough variation in the image to be considered as an additional training example. A much larger rotation was used in [267] which rotated with a value of 90 degrees. The optimal rotational angle that should be used for data augmentation remains unknown from the literature surveyed.

Cropping was used nearly as often as rotation and various approaches were defined to achieve this. Randomly cropping the image to a specified size ranging from 42x42, 48x48 and 56x56 were reported. Here the original image is larger, and the authors either randomly crop patches or preform location specific predefined crops of patches from the original image. The most commonly used approach was to crop five patches of fixed size from the original image, i.e. the four corners and the center of the image were cropped. Once again the cropped size was smaller than the original image, for instance, in [121], the original image was 48x48 and the cropped patches were of size 42x42. Extracting five crops from the image was performed by [235, 120]. [240] extracted 5 crops of 88x88 from 96x96 images, [156] extracted a 56x56 crop from 64x64 images. In some cases additional augmentation was performed on the cropped images, for example [121] cropped 5 patches and then applied horizontal flipping to each crop, consequently the authors augment their data 10 folds. This was also conducted in [172]. [46] extracted every possible 48x48 patch from the 64x64 original images resulting in 16 crops per image. Single crops are extracted from the original images in [225, 110]. Additional studies that conducted cropping include [203, 149, 254, 51]. It was observed that in certain studies cropping was applied to the train and test images, for example, [110] cropped the 4 corners of the test images and flipped them. These additional images were used by the CNN and the class probabilities were averaged for the final prediction.

There is no study which attempts to determine which cropping technique is most optimal for data augmentation in the field of FER. Cropping reduces the amount of pixels in the training images which consequently reduces the number of input pixels to the CNN. This ultimately reduces the number trainable parameters in the model.

Zooming was used as a data augmentation technique in six studies. There was variation in the implementation and no analysis on the optimal approach was been investigated. [215] zoomed the images by 10%, [53] zoomed by a

factor of 1.2, [7] zoomed on the four corners of the input image, [267] randomly selected patches of the image and then zoomed in, [73] zoomed in and out of the image but provided no details about the zoom factor, and finally [2] provides no details on their implementation. It is challenging to conclude as to which implementation is the best given that there is no consistency in the implementations. It would be of interest to further investigate this augmentation technique and to gain insight as to why it is not used as frequently as the other approaches.

Augmentation by collecting extra data was applied in [37, 254]. In the former 1000 additional images were collected for each class. In the latter, Tan *et al.* collected 2000 additional images for each class using Baidu and Google search engines. Such an augmentation approach is useful; however, it is a tedious one. To deal with the time complexity involved in labelling new images one can make use of services such as Amazon Turk [1] to take advantage of crowdsourcing the labelling task.

Only a few studies reported the software which was used to perform data augmentation – OpenCV was used in five studies. There are no further implementation details reported for data augmentation. From the articles surveyed in this paper, 47% of the studies used data augmentation techniques. This observation is surprising as it is well known that deep learning models perform better with additional data. In certain studies, there was no mention whether data augmentation was implemented. It is possible, however, that data augmentation was implemented without the authors mentioning it. In certain studies the authors report that no augmentation was used at all.

A.2 Programming and Hardware

This section discusses the choice of programming framework which were used to implement the CNNs as well as the hardware used and training times. Eight different frameworks were used, the platforms were Tensorflow [1], Caffe [111], MatConvNet [269], Keras [35], TFLearn [48], Torch [41], Lasagne [58] and Theano [258]. Out of the studies reviewed, 38 reported their choice of framework. Caffe was used the most frequently, it was reported in 20 studies. The next framework which was used most frequently was Tensorflow which was reported in 5 studies. The remaining frameworks were used in 3 or less studies. Details are presented in table A.2. It is beyond the

[1] https://www.mturk.com

Framework	Number of studies
Caffe	20
Tensorflow	5
Keras	3
MatConvNet	3
Lasagne	2
Theano	2
Torch	2
TFLearn	1

TABLE A.2: The different frameworks which were used in the studies surveyed along with the number of times each one was used.

scope of this literature review to comment on which framework is the most suitable one in terms of performance.

Training a CNN on a GPU is considerably faster than on a CPU. Throughout the studies surveyed, 17 studies reported the use of a GPU in their experiments. A total of 13 different GPUs were reported for which the Titan X was used in three studies and was the most commonly used GPU. GPUs with larger memory are an advantage as they allow for large CNN architectures to be stored in memory. The use of GPUs as an online service, such as Amazon AWS was reported in [267]. [283] reported that experiments were conducted on CPUs. In terms of CPU RAM, five studies reported the amount of RAM used, the values varied from 2GB [226] to 65GB [204]. Seven studies report training or test time, [242] trained their model in 10 minutes and [226] required up to 40 hours to train theirs. The authors of [240] report that a test image can be evaluated in 50 milliseconds, [301] reports 18 and 43 milliseconds on their two models and [207] needs up to 70 seconds to process a test image.

Here the main findings are that authors do not report sufficient detail with regards to the software and platform used as well as the hardware which was used. There is also a lack of reporting in terms of the training and test time for image. Consider the application of FER in robotics, here the goal would be to create a FER model that can process images as rapidly as possible in real time. Thus, reporting on the evaluation time to process a test image is valuable information to compare between different studies.

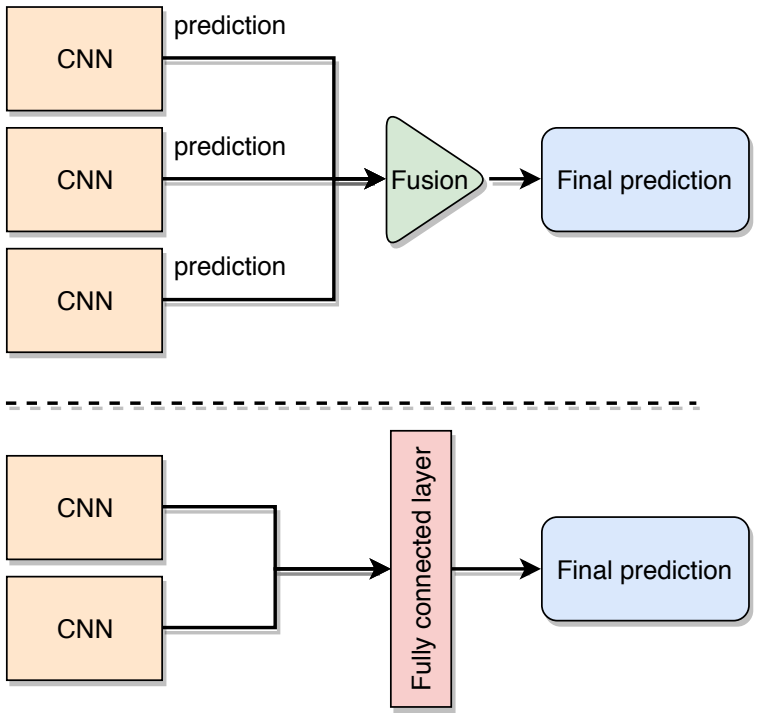

FIGURE A.2: Illustrating the difference between the two primary ensemble methods which were observed in the studies surveyed. The figure on the top shows the structure of *individual fusion*. In this approach the predictions of each CNN are combined to form a final prediction. The bottom figure shows the structure of the *concatenation fusion*. In this approach the individual CNNs are combined into a fully connected layer which in turn produces the final prediction.

A.3 Ensembles

This section reviews studies which have used ensembles of CNNs with or without additional ML approaches applied to FER tasks. Out of the studies reviewed there were 22 of them which implemented ensembles. The studies were clustered based on the overall system structure proposed and two primary groups were found. These groups differ primarily on how the authors implement the fusion of the individual CNNs. In the first type, which is denoted as *individual fusion*, the studies obtain the predictions from each individual CNN and apply some fusion mechanism to combine the predictions into a single prediction which is used as the output of the ensemble. An example is a simple majority vote across each CNN.

The other primary type, which is denoted as *concatenation fusion*, is where a single concatenation layer is applied to combine each CNN. This layer is then used to establish the final prediction which serves as the output of the ensemble. Figure A.2 illustrates these two approaches. Other variations were observed and will be discussed.

Studies which used the *individual fusion* approach are now discussed. [301] used three CNNs with a similar architecture however, each CNN took in images at different input scales. The images were scaled by a factor of 1.0, 2.0 and 2.14. Each CNN was trained separately on the FER'13 dataset and an average feature for each class is computed from the last fully connected layer in each CNN; softmax is then applied to compute the final prediction. The individual CNNs are compared to the ensemble and the findings reveal that the ensemble outperformed the individual networks. [267] also made use of three CNNs in their ensemble. In this study each CNN was the AlexNet network.

The authors of [7] implemented three CNNs, two of which are made up of two primary branches. One branch uses a CNN and scale invariant feature transforms (SIFT). The other uses a CNN and dense-SIFT. SIFT and dense-SIFT are hand crafted features. Features from both branches are combined into a fully connected layer. The authors execute their proposed method as three models, CNN only, CNN with SIFT and CNN with dense-SIFT. Each of the three primary models use the softmax function in the last year. These probabilities are fused together by averaging the predictions and the largest one is used as the final prediction. The proposed ensemble is evaluated on FER'13 and CK+. Once again the ensembles outperform the individual networks. [254] proposed an ensemble which contained two main components. The first contained two CNNs. The first was a custom CNN architecture which was pre-trained on a face recognition dataset and this CNN was applied to crops of aligned faces. The second CNN was applied to crops of non-aligned faces; this was ResNet34 which was pre-trained on a FER dataset. The second main component of the ensemble was ResNet101 (pre-trained on ImageNet) and was tasked with predicting the overall image emotion. The authors explored combinations of weights assigned to each model when fusing the predictions. The highest weight was assigned to the CNN which took in aligned faces as input. The authors of [279] use a probability based fusion function and then choose the highest corresponding class as the final prediction. Each CNN outputs a probability for each of the classes. The individual CNNS were trained and evaluated on a validation dataset. A number of the CNNs were selected to be in the ensemble based on their individual validation performance. The authors compared majority, weighted majority and probability based fusion on the FER'13 dataset and their findings indicated that the probability based fusion was the most suitable method. They additionally compared different ensemble sizes from 1 to 100 – they observed that

approximately 35 members formed the most suitable ensemble on FER'13, JAFFE, CK+ and EmotiW'15 datasets.

Studies which implemented the *concatination fusion* approach are now discussed. [222] used two CNNs which was followed by a concatenation layer. The first CNN used the top half of the input image (thus learning features around the eyes and eyebrows) and the second used the bottom half of the input image (thus learning the useful features around the mouth). The proposed approach was evaluated on the KDEF dataset and the authors compared the ensemble to a single CNN with the identical architecture. A fair comparison cannot be concluded since the ensemble was run with less epochs than the single CNN model. [149] trained three CNNs separately – made up of 8, 9 and 10 layers – inspired by VGG. The three CNNs were combined using a concatenation layer and then the softmax operation was applied. The ensemble was evaluated on FER'13 and the authors demonstrated that it outperformed the individual CNNs. [286] also implemented three CNNs and combined them using a concatenation layer. In comparison to [149], the authors of [286] implemented identical CNNs. Their approach was evaluated on JAFEE and CK+. Three CNNs were connected to two fully connected layers in [73]. Here each CNN uses a different kernel in the first layer, the sizes were 5x5, 7x7 and 9x9. The ensemble was applied to the JAFEE dataset. [94] made use of two parallel CNNs made up of convolutional and max-pooling layers. The first convolutional layer in one of the CNNs was pre-trained as a convolutional auto-encoder after which the weights were frozen during the optimisation of the ensemble. The two CNNS were concatenated and this was followed by a logistic regression function. Their approach was evaluated on JAFEE. Two pre-trained CNNs were used to form an ensemble which was combined using a concatenation layer and followed by the softmax operator in [2]. The first CNN was the Xception model [36] which was pre-trained on FER'13 and the second was VGG-16 pre-trained on ImageNet. The ensemble was evaluated on Group Affect 2. In [289] the authors attempt to solve age estimation and FER in a single ensemble made up of two CNN components. The features from each CNN are concatenated together and their approach was evaluated on the Lifespan and FACES datasets. [91] extracted 256 features from AlexNet and 136 facial landmark features from the input images. Here the ensemble was made up of those two branches and then the features were concatenated and input into another fully connected layer. This ensemble was evaluated on the iCV-MEFED dataset.

The other studies which implemented ensembles used different approaches to the ones previous discussed. The following discussion now proceeds to describe studies whereby the ensemble is made up of CNN and non-CNN features which were input into another machine learning algorithm and then fused. [215] implemented an ensemble which was made up of three primary components. The first two were used applied when a face was detected in the input image. Here a CNN and hand-crafted approach were used – the CNN was VGG-Face. Both of these were used as input to a random forest. The third component was applied when no face was detected in the input image – the authors used VGG-16 pre-trained on ImageNet in this component. Each component of the ensemble used random forest to perform the individual predictions and a weight was assigned to each of them. The weights were obtained based on the validation accuracy. The authors evaluated their ensemble on the Group Affect 2 dataset and their findings show that the ensemble outperformed the individual components.

[277] used a similar approach whereby a component of the ensemble was applied when faces were detected and another applied when the faces were not detected. In this study 8 CNNs were used when faces were detected. These include VGG-Face, VGG-16 and custom CNN architectures. In the second component one CNN and two non-CNNs were applied to the entire image. Features were extracted from each of the members in the ensemble and input into a recurrent neural network and SVM. Their ensemble was evaluated on the Group Affect 2 dataset. In the study of [278] a number of CNNs were randomly generated by randomly initialising the architecture and parameters. Each CNN was applied (without being optimised) on the training data and the output of each CNN was used to train one echo state network. The ensemble predicts on new examples by enabling each CNN to predict and the majority vote is used to obtain the final prediction. The authors analyse the performance when creating ensembles of varying sizes from 5 to 80 members. Their approach was evaluated on JAFFE and CK. This approach benefits from the fact that a large ensemble can be created without training each individual CNN. Hand-crafted features, namely the centralised binary patterns are extracted from input features alongside a CNN pre-trained on face images in the study of [151]. Both sets of features are used as input into a SVM. The ensemble was evaluated on JAFEE and CK+.

The following discussion now proceeds to describe two studies which use unique approaches which cannot be clustered with the previous ones. [207] examined an ensemble of 72 custom CNN architectures and another

ensemble of 64 VGG-16 networks. Instead of using a conventional fusion approach, the authors propose to use a CNN to decide on the final prediction by considering the probability output for each of the ensemble members on each of the images. Their approach was evaluated on SFEW 2 and the authors compared various fusion strategies and ultimately demonstrate that their CNN fusion method outperformed traditional methods. [121] created 12 groups of 36 CNNs whereby the CNNs in each group had a different architecture, used different pre-processing on the input images and used a different seed to initialise the weights. The authors compared different overall structural approaches made up of various levels on how to combine each group into a hierarchy. Using a single level, they demonstrated that using an exponential weighted average fusion approach was better than more common approaches such a majority voting and averaging. Their approach was evaluated on the EmotiW'15 dataset.

A key idea with building an ensemble is ensuring diversity within the members of the ensemble. The authors of [120] attempt to deal with this issue by creating an ensemble of 9 CNNs for which each network is made up from a combination of different input image normalisation techniques and different seeds. Each CNN has an identical architecture, namely, 3 convolutional and 3 max-pooling layers followed by 2 fully connected layers. The ensemble is evaluated on the FER'13 dataset and the authors compared average and majority voting fusion methods. Their final solution used the average of the probabilities from each CNN since similar performance was obtained on both fusion approaches. Different weight initialisations were explored by [293]. In their study 6 CNNs were combined to form an ensemble. Each CNN was trained independently and the authors assigned weights to each member of the ensemble to fuse the predictions and obtain a final prediction. These weights were optimised separately in an additional step. The ensemble was evaluated on the FER'13 and SFEW datasets. The authors show that the ensemble outperforms the individual CNNs.

A number of authors implemented an ensemble and then compared the performance of the individual CNNs to the ensemble. In these experiments the findings reveal that the ensembles outperform the single CNNs. This result was confirmed by each study that conducted such a comparison. It is well established in machine learning research that an ensemble will usually outperform its individual members and thus less emphasis should be placed on this in future work.

The proposed ensembles were most commonly evaluated on the FER'13

and JAFFE datasets, and on average, evaluated on a single dataset. It would be of interest to evaluate new proposed ensembles on a larger number of datasets and to apply them on non-posed datasets and non-laboratory settings to determine their performance. It was observed that there was a lack of detail with regards to the fusion method when combining the individual ensemble members. In [209] the authors mention that their ensemble of 8 CNNs was able to outperform the current state-of-the-art on the FER'13 at the time of publication but the details of the exact members of the ensemble are not provided nor the fusion method. Few studies reported the rationale behind using the number of members in their ensemble. Most studies used less than 10 members in their ensemble. It would be of interest to examine the effect of the ensemble size across various architectures and datasets (including both posed and "in the wild" datasets). Despite the obvious improvement in performance which an ensemble provides it does simultaneously result in larger training times given that CNNs already take relatively long to train. It was observed that few studies address the primary issue around incorporating diversity amongst the members of the ensembles. Certain studies allowed for some differences in architecture and input pre-processing. It would be of interest to further explore this in the case of FER. It was also observed that there have been few attempts at proposing or exploring novel fusion methods. Using a concatenation layer to combine features from various CNNs overcomes the issue of optimising a fusion mechanism. It would be of interest to compare whether a concatenation approach yields similar performance to an optimised fusion method.

A.4 Transfer Learning

CNNs which have been trained on some image dataset can be applied to another dataset, in such a case the knowledge which been acquired by training the original dataset is reused – this process is referred to as transfer learning [85]. This section surveys the studies which have used transfer learning in the context of CNNs and FER. From the studies surveyed, 30 of them reported the use of transfer learning. When applying transfer learning with an existing CNN architecture, the input for the architecture can expect, for instance, 3 channel inputs. To deal with this it was observed that authors would create copies of their greyscale input images to obtain the correct number of

channels, for example see [294, 203]. AlexNet [130] and VGG [237] based architectures were used the most frequently. AlexNet was reported in 6 studies and architectures similar to VGG-16 were reported in 6 studies. Other reported architectures include ResNet variants [96], Inception modules and GoogleNet [251]. The implementations varied from one study to another.

It was observed that authors used transfer learning in one of two ways, either by loading existing weights from an architecture which was trained on some other problem, or to initialise the weights of an architecture and then pre-train it on some dataset and fine tune on a FER dataset (two training phases in this case).

For the former case, authors loaded the existing weights and then fine-tuned the weights on a FER dataset. In other studies the convolutional layer weights were frozen and the fully connected layers were fine tuned on a FER dataset, for example [66]. The last layer would often be replaced with a new one which would correspond to the number of classes in the FER dataset. This was done because the initial architectures were often trained on ImageNet which would have 1000 classes, and consequently, 1000 units in the last fully connected layer. Here authors would randomly initialise the weights in the new fully connected layers and fine-tune these weights on a FER dataset, for example see [293]. [93] used VGG-16 which was pre-trained on ImageNet and then fine-tuned the model on the AffectNet dataset for 80 epochs. Pre-trained models on ImageNet were also used in [166, 186, 302, 81].

In the latter case, authors would use either an existing architecture or a custom architecture and then pre-train it on some dataset and then fine-tune it on a FER dataset. [17] pre-trained a 24-layer CNN on a celebrity identification dataset. The authors collected 280,000 images from Flickr to create a fine-tuning dataset which expressed 7 emotions. In [296], the authors pre-trained a CNN on a face identification dataset and then fine-tuned the model on FER'13 and SFEW by replacing the last fully connected layer with 7 units. Similarly, [289] pre-trained a CNN on the Morph Album dataset (face identification) and fine-tuned the model on the Lifespan and Faces emotion datasets. Instead of pre-training on a face identification dataset, the authors of [293] and [7] pre-trained their models on an FER dataset. [293] pre-trained on the entire FER'13 dataset. The convolutional layers were frozen, and the fully connected layers were fine-tuned on the SFEW training dataset. [7] pre-trained on FER'13 and fine-tuned their CNN on CK+.

There were certain studies that would make use of a forward pass through

a CNN that was pre-trained on a dataset (FER or non-FER) and then extract the features at a given layer. These features would then be used as input into a classifier which would be optimised for the FER task. Here, typically the SVM classifier was optimised and fine-tuned on a FER dataset. [167] used AlexNet which was pre-trained on ImageNet and the authors obtained 9216 features from the third pooling layer. These features were input into a SVM for which a grid search was used to obtain the best parameters. The method was evaluated on JAFFE and CK+ (using the last frame). [191] also extracted features from AlexNet. In this study the authors extracted 9216 features (from the last pooling layer) and 4096 features (from the first fully connected layer). These features were input into a SVM and the method was evaluated on CK+. [215] used VGG-Face [198] which was already pre-trained on a face recognition task. Here 512 features from a pooling layer (before the first fully connected layer) and 4096 features from the second fully connected layer were extracted. Combined, the authors input these features into several classifiers and found that random forest yielded the best results. The approach was evaluated on the EmotiW'17 dataset. The authors of [288] used a custom CNN architecture which was made of 4 convolutional and max pooling layers followed by two fully connected layers. The model was pre-trained on MSRA-CFW – a face recognition dataset – and 120 features were extracted from the first fully connected layer and input into SVM (the authors used LIBSVM [2] as implementation software). The advantage of this approach is that the CNN can extract salient features which are useful for FER tasks and in turn have the freedom to input these features into a suitable classifier which can be further optimised.

Six studies compared their results with and without transfer learning. [289] pre-trained a CNN on the Morph Album dataset and fine-tuned the model on the Lifespan and Faces datasets. The same network which was initialised with random Gaussian weights had a weaker performance than the pre-trained model. [121] pre-trained a CNN on the FER'13 and Toronto datasets. Various kernel sizes in the convolutional layers and number of units in the full connected layers were compared. The equivalent networks which were randomly initialised had weaker performance than the pre-trained ones. It is observed that the pre-trained models achieve a higher validation accuracy in the first few epochs when compared to the models which were randomly initialised. A similar observation is made in the work of [2]. [221] used a convolutional auto-encoder to initialise the weights of the

[2]https://www.csie.ntu.edu.tw/ cjlin/libsvm/

convolutional layers in their CNN model. The KDEF dataset was used for pre-training. [207] conducted a similar comparison to [121] whereby various convolutional kernel sizes and number of units in the fully connected layers were compared. The authors compared the usefulness of pre-training on a combination of datasets. On average, the pre-trained models outperformed the equivalent CNN without pre-training. It was however observed that this was not the case when the CNN was pre-trained on the MMI dataset. The weaker performance could be a result of only using 80 images from MMI to pre-train the CNN. Here the insight is that pre-training on a small dataset does not provide much of an advantage over random weight initialisation. [294] show that transfer learning outperformed random weight initialised models on 4 out of 5 datasets across 10 runs.

The choice of dataset for pre-training could impact performance. Pre-training on a dataset with images of faces would be ideal since the convolutional layers learn facial features which are useful for FER. Pre-training on datasets which contain additional classes which are not related to faces could result in a weaker positive impact – here the convolutional layers could be learning features which help discriminate between the classes instead of learning features useful for faces.

From the studies surveyed several observations were made and recommendations can be suggested to address certain shortcomings. There is a lack of detail in terms of method which was used to initialise the weights. For instance, some studies froze the convolutional layers and then trained the weights in the fully connected layers from scratch. Authors should mention how the weights in these layers were initialised. Details regarding exactly which layers are frozen, which are fine-tuned and which are randomly initialised should be provided. It was observed that in certain studies the authors did not mention which dataset was used to fine-tune certain layers. In the case whereby existing architectures are used, certain studies do not specifically mention if they are loading the existing pre-trained weights, if they are training the model from scratch, or if certain layers only are being fine-tuned. An attempt was made to reason why AlexNet was used frequently. Thorough conclusions cannot be drawn without a fair comparison between the different architectures. However, it was observed that the weights for AlexNet are publicly available without any license restrictions. It is clear that using transfer learning leads to improved results and thus authors are encouraged to implement it. It also reduces the overhead of having to train a model from random weight initialisations which can take long

especially on weaker hardware, or in the case of multiple experiments (i.e. when multiple architectures are being compared or if k-fold cross-validation is used).

A.5 Validation and Reporting of Results

This section analyses the ways in which authors report their findings and how they compare their results to other methods. The purpose was to determine the trends in which authors conduct the validation of their methods, and to determine whether the comparisons that authors make to other studies in the literature are fair and scientifically correct.

On average, most studies used a single metric to report the performance of their CNN. The most commonly used metric was the accuracy measure and was reported in 83 studies. Accuracy is reported in a wide number of machine learning papers outside of the field of FER and is widely accepted as a conventional metric to use. For certain datasets one has be cautious about the accuracy paradox when using this metric. There is sufficient overall class balance in most FER datasets for this to not be a concern when reporting with accuracy, for example, the average class distribution for CK+ and FER'13 is 16% and 14% respectively (both containing 7 classes). The next metric which was used the most was F-score, it uses precision and recall which is more suitable for imbalanced datasets – this was reported in three studies. [167] reported on both accuracy and F-score, [149] reported on both but the findings are not compared to literature. Precision and recall was only reported in [149]. [33] used the average true positive rate across 5-folds and compared their result to a single study which also used the same metric and dataset. [294] discuss micro- and macro-average accuracy whereby the latter is used to deal with class imbalance, they report their results using micro-averaged accuracy. Studies that make use of metrics other than accuracy provide little insight into the information that these metrics are revealing and thus it can be concluded that these metrics do not provide insightful information beyond which can be obtained from the accuracy metric. Implementing a different metric in comparison to the most commonly used one – being the accuracy metric – introduces complexities when comparing results and conducting a fair comparison.

Reporting results using a confusion matrix was performed in 42 studies. A confusion matrix can provide useful insight into which facial expressions are easier and more complex to correctly classify. It can furthermore be used

to get an understanding on which expression could benefit from additional images by either collecting more images or using data augmentation techniques. Across all the studies which used a confusion matrix, happiness was the expression which was correctly classified the most often and fear was the most misclassified expression. Fear was misclassified as either anger, disgust or sadness. Fear is the minority class in the CK+ dataset but this is not the case for FER'13. It would be of interest to determine whether this expression is misclassified due to the expression class distribution or because there is little visual saliency between fear and the other negative expressions within both the posed and "in the wild" datasets. There were no consistent misclassified expressions for each individual dataset and there are not enough results for each dataset to make concrete conclusions.

The different approaches authors took to validate their experiments were compared. The most commonly used approach was holdout, whereby the dataset was split into either training and test sets or, training, validation and test sets – this approach was reported into 42 studies. K-fold cross-validation was used in many studies, here authors used different values of k whereby 19 studies reported the use of 10-fold cross-validation. Using different values of k than what most existing studies can result in unfair comparisons. [172] used 8-folds whereby 6 were used for training and the remaining 2 were used for validation and testing. [32] used 4-folds and compared their results to three studies which also used 4-folds. [294] gathered 7 datasets for which 6 were used for training and the last one was used as the validation set. [94] evaluated their approach on the JAFFE dataset and to validate their method they randomly extract a single image for each expression from each subject and was descried as a leave-one-out approach. [252] used a leave-one-subject-out approach whereby a single subject was used for testing and the remaining subjects were used for training – this was applied to the CK+ dataset. A similar validation method was used in [167] on the CK+ dataset. The result obtained by [252] was greater than [167]. It was observed that in a number of studies it was not possible to determine what validation technique was used, and thus, any comparison to such studies is not possible.

There were few studies for which the authors mentioned the number of times they executed their validation approach by conducting multiple runs. The primary concern is that reporting a result based on a single execution of a proposed approach is not statistically significant when compared to a study that has repeated experiments with the same experimental settings. Furthermore, reporting a single run of holdout is less statistically correct as

opposed to a 10-fold cross-validation for which the 10 runs are averaged and authors can report on the standard deviation in the results. A discussion on the statistical significance of results falls outside of the scope of this study, however the findings raise an immediate question with regards to the correctness in the way authors have been reporting their results. [203] used holdout, repeated their experiments 10 times and reported on the best results along with the average performance across the 10 runs. The authors however do not report on the standard deviation of the results. [181] also reported on the best and average performance, however the authors reported their findings across 50 runs. [252] reported their results across 5 runs and also did not report on the standard deviation. [152] ran their experiments up to 10 times and reported on the standard deviation when reporting their results on the BU-3DFE and JAFFE datasets, however this was not done when reporting on their cross-database experiments. [94] conducted two experiments, the one used the leave-one-out approach and the second used 10-fold cross-validation. Both experiments were repeated 15 times and the average accuracy along with the standard deviations are presented. [294] report their minimum, maximum and average result along with the standard deviation across 10 runs. On the JAFFE dataset, the minimum result was 40.4% and the maximum was 50.2%. Assume that the authors had only conducted a single experiment which happened to be run which obtained the best accuracy. It would not be fair to report and compare that result with other studies as the best result is not representative of the overall average performance of 44%. Thus, one can put into question the fairness and statistical correctness of the results from studies which either do not mention how many runs they conducted, or those that conducted only a single run.

A new trend of experiments was observed. In these experiments, CNNs are trained on a given dataset and then evaluated on another dataset. This has often been referred to as cross-dataset, or cross-database validation and was reported in two studies from 2016 [179, 146]. [294] states that reporting results on a single dataset is not sufficiently representative of the model's ability to generalise and thus cross-dataset experiments are more appropriate. [172] trained a CNN on CK+ and evaluated on MMI then evaluated the performance by training on MMI and evaluating on CK+. [231] trained on FER'13 and evaluated on AFEW. [18] trained three networks, one on RAFD, the other on CK+ and the third on a mixture of the datasets. The authors evaluated each of the three CNNs on RAFD, CK+ and JAFFE. [152] trained

a CNN on CK+ and evaluated it on BU-3DFE and then on JAFFE, the experiments were averaged from 8 runs. [179] trained their CNNs on 6 datasets and evaluated the performance on another dataset, this was repeated so that each dataset was used as the test set. The authors compared their results to 4 studies which also conducted cross-dataset experiments. [294] also trained their models on 6 datasets and evaluated on the remaining dataset, a comparison against other cross-dataset methods was also conducted. [166] trained a CNN on CFEE dataset and evaluated on RAFD. [151] trained a CNN on CK+ and evaluated on JAFFE. [302] compared the performance when training and testing on CK+ and RAFD. [24] evaluated their approach by training and testing on the combination of CK+, NovaEmotions and FER'13. On average the studies evaluated their cross-dataset experiments on 2 datasets and few studies compared these results to other literature papers. It would be of interest to compare the performance of cross-dataset experiments on pose and "in the wild" datasets to determine whether posed datasets can successfully be used when evaluating on non-posed ones.

On average, authors of studies compared their results to less than 2 results found in the literature with a large number of studies not conducting any comparison to the literature. There were three studies for which the authors compared their results to more than 5 other literature studies on a single dataset. [172] compared their findings on the SFEW dataset to 9 studies, 7 studies for MMI and 8 studies for CK+. [59] compared against 5 to 11 other studies and [266] compared against 5 to 9 studies. Furthermore, in addition to comparing to other studies found in the literature, few authors compared their proposed methods to other CNN studies. On the other hand, 44 studies compared their results to their own implementation of another type of CNN, and 11 studies compared their results to non-CNN implementations. Comparisons were made to local binary patterns [202] which were used as input features in [32, 240]. [214] compared their results to a decision tree, multi-layer perceptron and SVM classifiers. SVMs were also used for comparisons in [213]. A comparison was made against k-nearest neighbours in [233]. Other non-CNN approaches were used in [288, 266, 27]. [51, 208] compared their approach to Microsoft Emotion API[3].

Several papers which are submitted to FER competitions and are then published do not compare their results to literature as these competitions often release a new dataset for which participants must specifically report their results on. These papers are often compared to a baseline paper which

[3]https://azure.microsoft.com/en-us/services/cognitive-services/emotion/

would have been provided by the organisers of the competition. [255] submitted a CNN solution to ICML'13, [121, 186] submitted to EmotiW'15, [91] to DCER&HPE'17 and [215, 2] to EmotiW'17. In such submissions it would be of interest to compare various CNN architectures to the submitted solution so that a number of results can be obtained which will enable future research to benchmark against. [225, 149] compared their results to the Kaggle leader board for the FER'13 challenge. Immediately one can put into question whether it is fair and useful to compare results to benchmark results on leader boards. It could be of interest to enable online competitions in a fair setting, such as through Kaggle, which could allow for a larger audience of participants to achieve new state-of-the-art results. Tables 8 to 7.6 presents the results obtained for each of the most popular datasets. The table also presents the validation method along with the data augmentation used.

A.6 Recommendations and Future Work

This section presents recommendations for future implementations of CNN for FER based on the studies surveyed. These should enable rapid implementation of a CNN and provide details on how to conduct experiments, report on results and various other important aspects to be considered. This section also presents ideas for future investigations which have not yet been explored in addition to those mentioned in the previous sections.

A.6.1 Datasets

The choice of dataset is important when conducting comparisons with other studies. There were a few studies that used private datasets for which other authors most likely will not be able to compare to and thus any conclusions drawn on the private datasets cannot be verified or benchmarked. The most commonly used dataset was CK+, and authors used the first frame (neutral class) and last three frames (most expressive in terms of emotions) since this is an image sequence dataset. JAFFE and FER'13 were also widely used. In terms of comparing against existing studies authors should benchmark their future works with at least those three datasets. However CK+ and JAFFE have limitations, and thus authors are strongly encourage authors to benchmark on "in the wild" datasets. Authors are urged to not solely report on posed laboratory datasets.

The development and release of new "in the wild" datasets would be beneficial. There is a need for them as laboratory posed datasets do not reflect how humans express spontaneous facial expressions. Furthermore, laboratory datasets have constant light settings which is not realistic. New datasets should consider catering for gender, ethnicity and age balance [282] as well as variations in lighting conditions (which can affect performance [282]). JAFFE for instance is limited in terms of variations in gender and ethnicity. Variations in age can greatly impact the performance of the classifier as facial features such as wrinkles and nasolabial folds [289] will have an impact. New datasets should also contain images of individuals with eyeglasses, facial hair and other features that can obscure the face – this of course renders the task more complex but simultaneously more realistic. Should a new dataset be released, then the authors are encouraged to evaluate multiple CNN architectures on the new dataset. This could include using several custom architectures and to use existing state-of-the-art CNNs to provide an initial benchmark.

Most existing datasets have few emotion classes which is not entirely truthful to what humans convey. As humans, we express various combinations of the six basic emotions which are referred to as compound emotions [61], for example, happily surprised and angrily disgusted. Authors of datasets are encouraged to create more complex datasets with a larger variety of emotions. Additionally, efforts should be placed on the construction of 3D image datasets since it is unrealistic to only be able to conduct FER on subjects that are facing forward. Few studies reported results on BU-3DFE, see [152, 294, 298, 143, 109] for studies which applied CNNs to this dataset. [299] also mentions some of the observations which were presented here. Section 7.3 highlights various studies which discuss how they constructed new datasets – this can serve as a guide for authors wishing to create new ones. The primary drawback is that constructing such new datasets is extremely time and resource consuming.

A.6.2 Pre-processing

It is clear that including pre-processing has benefits. These include reducing the number of trainable neural network parameters, reducing the processing time and achieving higher classification accuracy. It is recommended that future works implement pre-processing, and clearly report on which algorithm was used and how. Histogram equalisation should be applied to deal with

variations in illumination. Viola Jones can be used for face detection but it is advised to use another algorithm to boost the performance or reduce the number of false positives. Authors should report on the pre-processing time. Section 7.4 lists several experiments for future research. By conducting these experiments authors can then apply certain methods which are supported by scientific evidence instead of using techniques without clearly stating the rationale behind the methods selected and to avoid a lot of inconsistency within the literature. One challenge which will be faced in attempting to determine the most optimal pre-processing techniques is that there are lot of factors to consider, for example, the variations across datasets and the CNN architecture. Training a large number of architectures across a large number of datasets and for which various pre-processing techniques are examined will be a computationally expensive experiment.

A.6.3 Data Augmentation

In terms of data augmentation, future works should implement some form of augmentation to increase the number of training samples as this is essential in training deep learning models. At the very least, horizontal flipping should be implemented as this doubles the amount of images available. OpenCV can be used to achieve this; it requires little computational effort and there are no parameters associated with this technique making it the easiest to implement. Rotation was also used commonly and can greatly augment the data especially if this is done both in the clockwise and anticlockwise direction. The challenge with rotation is selecting the angles to use. The most frequently used values were 45 and 10 and by using both of these values the dataset is augmented by 5 folds. Cropping was also commonly used. Assume that an original image is 48x48, then authors commonly extracted five 42x42 patches from the top left, top right, bottom left, bottom right and centre of the original image. One advantage of applying cropping is that it reduces the input image size and consequently the number of neural network trainable parameters. By applying this approach the dataset is augmented by 5 folds, furthermore, these cropped images can be flipped and rotated for additional augmentation. Zooming was used in a number of studies but it is unclear as to what the optimal zoom factor is.

Given that horizontal flipping, rotation, cropping and zooming are the most commonly used techniques, it would be of interest to conduct an analysis on how much benefit each of these augmentation techniques actually

provide using statistical tests across a number of datasets. It would also be of interest to determine the most suitable parameters associated with these, such as rotational angle, crop size and zoom factor across various datasets. It is unclear why certain authors used certain techniques over another, for example, why would horizontal flipping and zooming be chosen instead of horizontal flipping and rotation? There is a lack of justification in most studies. Authors are encouraged to share insights when selecting particular techniques over others (as well as their justifications about their choice of associated parameters) and to provide details regarding which software was used as it was noticed that this was often omitted.

A.6.4 Hardware and Programming

It is known that training deep learning models is a time-consuming task especially when the number of trainable parameters is large and there is a lot of training data. It was noticed that there are some authors which have trained their models on CPUs. Ideally a GPU should be used as this will result in faster training times and large models can be trained; one drawback is that GPUs are expensive. The most commonly reported GPU was the Titan X. To overcome the need of training a model from scratch authors can use transfer learning and fine tune a smaller set of weights – this is discussed in the following subsection. It was noticed that some authors are using online services such as Amazon AWS to train their models, this can be beneficial as opposed to purchasing expensive hardware.

Authors are encouraged to report on their choice of deep learning framework as it was observed that a large number of studies omitted such details. The most commonly used framework was Caffe. To the best of our knowledge we have not found any recent academic article which benchmarked frameworks, but we have observed that there are blog posts and other online documentation of such benchmarks. Authors are encouraged to report on the number of trainable parameters in their CNNs as well as to report on the training and testing times. Reporting on test times can be of value especially when executing (and benchmarking) CNNs for FER on robots (such as in [13]) or mobile devices with more limited hardware. Real time execution of CNNs for FER is still a challenge to address [299] and few studies attempt to deal with this.

A.6.5 Ensembles

Studies that implement ensemble methods should clearly report on the overall structure of their approach and describe in detail how the fusion of the individual members of the ensemble take place. Researchers should focus efforts on an analysis of the effect of the ensemble size and attempt to develop a method to optimise the best ensemble size. One way to achieve this could be to implement a similarity index between networks in the ensemble to remove redundant members. For example, there is little use in having two or more members which output similar predictions across the datasets. It might be of interest to develop a similarity index amongst network architectures to optimise diversity between the networks and the effect that such optimisation would have in increasing the overall ensemble. Furthermore, it would be of interest to implement a boosting type of ensemble whereby examples in the dataset that are correctly classifies by many members of the ensemble receive a smaller weight, and examples which are incorrectly classified by the members receive a higher weight. Authors either implement a fusion of the individual predictions or add a concatenation layer to combine multiple CNNs. It would be useful to examine these two approaches in greater detail and to conduct a thorough comparison to set the standard for future research. Given that ensembles outperform single CNNs it is recommended that future research attempts to incorporate an ensemble instead of focusing on single CNN methods. Any such comparisons or future investigations should be conducted across multiple datasets – on average researchers concluded their findings on a single dataset and it is thus recommended that a larger number of datasets is used, especially "in the wild" datasets.

A.6.6 Transfer learning

From the studies surveyed it was clear that applying transfer learning improved the classification accuracy as opposed to training from scratch. AlexNet was used the most frequently and it is thus recommended using that architecture and weights as a starting point. Authors should report in detail how transfer learning was used in their study. Studies should mention which architecture was used and if the weights were loaded. Details should be provided about which layers are frozen, which are fine-tuned and which are randomly initialised (and how random initialisation is conducted). The

datasets which were used for pre-training and fine-tuning should be mentioned. Any changes which are made to loaded architectures should be mentioned. Pre-training on a face identification dataset and fine-tuning on a FER dataset will yield good results as opposed to training all the weights from scratch. In section 7.3, a large list of datasets which could be of benefit in order to pre-train a CNN is presented. This also helps to save time and is recommended for authors that have limited hardware. It would be of interest to investigate the effect of freezing various layers in the network and to determine its effect on the results. It would also be of interest to load an existing dataset and to develop an optimisation algorithm which can determine the optimal architecture given the frozen layers. Furthermore, it would be useful to determine for a given frozen CNN architecture, what is the best classifier to be used along with the features extracted from the CNN, i.e. extracting features from a CNN and using these features as input to various optimisation algorithms. [215] address this, however it would be useful to further explore this across various algorithms (SVMs and random forests were investigated) and FER datasets.

A.6.7 Architecture and hyper-parameter selection

There were a number of studies that used existing CNN architectures as opposed to implementing their own from scratch. In most cases the authors do not justify their choice. Thus, it is recommended that authors provide some rationale for their selection, for instance if certain preliminary experiments were conducted to rule out the possibility of using another model. Simply using a model without comparing the possibility of using another implies that the results reported may not be the most optimal results which could have been obtained had another architecture been used. It was observed that AlexNet was used the most often, this was followed by VGG-16 and VGG-Face. Given that authors frequently used these architectures, it would be of interest to compare these across multiple datasets to determine which of the three is the most optimal architecture and establish the best of the three as a benchmark for future research. This comparison could also include an experiment on how long it takes to process an image and the hardware requirements required to run the model. This will help in determining the feasibility of implementing such models on external devices such as robots or smart TVs. [13] used depth-wise separable convolutional layers as mentioned in section 7.5. It would be interesting to further evaluate this approach

on multiple FER datasets since the goal was to reduce the number of trainable network weights.

From the studies surveyed, the average CNN architecture consisted of 3 convolutional layers with 42 filters (in the first layer) of size 5x5, 3 pooling layers along with 2 fully connected layers. This can serve as a starting point for future research, however, it is recommended that the architecture undergoes a thorough optimisation. It was found that in a lot of studies a significant amount of information was omitted in terms of the network architecture. Authors are encouraged to provide the number of trainable neural network parameters to facilitate comparisons with future work in terms of the efficiency of the models.

In terms of the choice of activation functions for the convolutional and fully connected layers, the ReLU function was used the most frequently since it overcomes the vanishing gradient problem and trains fast. Authors are thus recommended to use this function in future works in their initial investigations. There have been studies which propose new activation functions so as to improve the overall performance of deep learning architectures and authors are encouraged to keep up to date on these to boost the performance of their CNNs, for example see [38] and [212]. Authors are encouraged to report on all of the details regarding their choices of activation functions, a number of studies omitted these details and reproducibility becomes challenging in these circumstances.

The experiments conducted in [100] provided useful insights on the effects of regularisation on the CK dataset. From the findings, it is recommended to apply dropout after fully connected layers, BN after each layer and max pooling dropout after each convolutional layer. It would be interesting to further explore the findings achieved by the authors on a larger number of datasets to generalise the findings.

Little work has been done in exploring useful CNN architectures for FER across a wide variety of datasets. Authors often do not provide details as to how they optimised their architectures and thus it is unclear as to whether or not they actually did this. Those who have attempted to optimise their architectures often did not explore a large number of different settings, and provide little details on their methodology. It is possible that authors want to omit performing such optimisations given the large computational effort required in training a CNN from scratch. It would be valuable to further study this by proposing an optimisation strategy across various datasets.

Future work could then base their architectures on the results of such studies. One approach would be to use an optimisation algorithm to explore the search space of possible architectures as was conducted in [64]. In addition to architecture optimisation, hyper-parameter optimisation is an important step when attempting to train CNNs to achieve high accuracies. It was observed that authors do not provide a lot of details and that only a few hyper-parameters were optimised.

In future work, authors are recommended to explore a wide variety of values across various datasets to determine the best combination of architecture and hyper-parameters. It is also recommended that authors provide details of any optimisation that they have used and which values were explored as this will add value to the body of knowledge and help guide researchers. There were no studies which conducted an in-depth investigation on the effect of weight initialisation in the context of FER datasets. For completeness, it would be interesting to examine this in addition to the other considerations previously discussed when implementing CNNs. Authors are encouraged to report on the initialisation method that was used as a large number of studies have omitted such details, furthermore, all hyper-parameters should be clearly stated (e.g. learning rate, batch size etc.).

In terms of the loss function and optimiser it is recommended that the cross-entropy loss function and stochastic gradient descent are used as starting points based on the literature surveyed. Once again many authors did not provide these details to the reader. There was a lot of variance in terms of the number of epochs used thus recommending an optimal value is challenging, and the same can be said for the batch size.

A.6.8 Validation and reporting

The accuracy metric was used the most frequently and can easily be compared amongst existing and future research. Certain authors have made use of additional measures but sufficient evidence that justifies the need for excluding accuracy was not found. Little rationale was provided when a different metric was used. Authors should be cautious when using a different metric as consequently any comparisons to existing research is no longer feasible. Authors are recommended to present a confusion matrix when reporting their results to provide an insight into which expressions were challenging to correctly classify as this can be used to further improve the proposed methods. It was observed that the confusion matrices were not normalised in

certain studies and this makes the interpretation of the results slightly challenging, it is thus recommended that the results are normalised so that insightful information can rapidly be obtained at a glance.

To ensure that comparisons of future work are fair, authors should consider validating their results using more commonly used approaches to those used in literature. Holdout and 10-fold cross-validation were used the most frequently and it would justify reporting results using one of those. It would however, be more appropriate to report on multiple runs of holdout instead of one so as to get a distribution of the performance of the method instead of a single execution (which could obtain a good result by chance). Reporting on 10-fold cross-validation would be more suitable than a single holdout run. Often the CNN models and weights are not made publicly available and thus reproducing and comparing results can be challenging. Authors should thus conduct more than one experiment and should report on the number of runs conducted. In addition, authors should report on the standard deviation of their results when conducting multiple runs.

Reviewers should be more strict with comparisons. Comparisons with existing studies should be conducted in a fair manner, i.e. authors should compare similar validation methods together otherwise the comparison can be misleading into thinking that a proposed method is better than it actually is. Additionally, when conducting cross-dataset comparisons the same training and testing datasets should be compared – it was observed that this was not always the case. It was observed that certain studies combined multiple datasets together, although this provides a larger number of training instances and testing examples (which enables better statistical rigour), it simultaneously renders comparisons challenging because it is unclear on what the test data actually is especially if the authors provide little detail on how they constructed the splits.

Authors are encouraged to report results using cross-dataset experiments so that the generalisation of the method across datasets can be determined. Here it would be useful to report on as many datasets as possible and to compare against other studies which have done the same. It would be of particular interest to evaluate the performance on "in the wild" datasets as opposed to only using laboratory posed datasets. Cross-dataset results in existing studies achieve weaker accuracy as opposed to training and testing on the same dataset, but by reporting on cross-dataset results it can encourage future research to place efforts to boost the performance in such experiments.

Future studies should compare their findings to CNN methods and not compare against non-CNN methods as it has been shown in numerous studies that the traditional hand-crafted methods do not perform as well as CNNs for FER. A large number of papers published in 2017 describe traditional hand-crafted feature based methods for solving FER, it is thus encouraged that future work no longer uses this as a rationale for proposing a CNN based method. Authors are also encouraged to compare future results to the large number of CNN studies published from 2017.

A.6.9 Fairness in reporting results

This section discusses the matter of fairness in validation as much work has yet to be done in ensuring that future work reports results in a fair manner. There has been a lot of variation in the way authors validation their findings and few attempts have been made to set a benchmark standard. Lopes *et al.* [152] describe that a fair comparison is one whereby the same subjects should not be found in the training and testing sets. In their work they split the images into 8 groups containing a number of subjects each, and that the same subject is not found in more than one group. From all the studies reviewed, it was found that their work contained the most emphasis on fairness. [295] compare two experiments, one which allowed the data to contain the same subjects (not the same images) in the training and testing sets, and the second which ensured that the subjects used in training were not used for testing. This was compared on the JAFFE and CK datasets using various classifiers (non-CNN) and the findings reveal a weaker performance in the latter case. It is important to note that the images in the CK dataset are images extracted from a video sequence and consequently two images expressing any emotion within a short time difference are almost identical.

The following experimental design is recommended such that future work can be compared in a fair manner and to ensure a standard that enables uniformity. The experiments are named as follows *subject independent*, *subject dependent*, *dataset independent* and *dataset dependent*. In subject independent experiments, authors can report their results by ensuring that subjects in the training and testing sets are independent of each other. In subject dependent experiments, authors can use the same subjects for training and testing but should ensure that the same images are not used in both sets. These two experiments should be conducted on a single dataset and can be repeated on additional datasets – this is similar to the approach described by [295]. Since

a great deal of CNN research has been applied to existing datasets and a vast amount of experimental results exist, authors are encouraged to report on cross-dataset experiments. This will test the generalisability in an even more challenging environment than the subject independent test. In dataset independent experiments, authors should report their results by ensuring that datasets in the training and testing sets are independent of each other. Finally, in the dataset dependent experiments, authors can use the same datasets in the training and testing phases but should ensure that the same images are not used in both sets. These four experiments represent every combination of results which were found in the literature and will ensure that future comparisons are fair to the extent that the same data is being compared.

Extracting images from a video sequence can result in multiple images which are nearly identical especially if the frames are extracted at a small time interval from each other. As discussed in section A.6.1, it would be encouraged that "in the wild" datasets are created, however should there be a need to create a dataset for particular subjects, then the time intervals between which the images are obtained should be large so as to avoid the issue previously described. The rationale here is that it is unlikely that, for a particular expression, images obtained at large time intervals are nearly as identical to images obtained at a small time interval difference (e.g. frames extracted from a video sequence).

Authors of new datasets are encouraged to provide pre-defined splits, for example training, validation and test sets and a k-fold cross-validation split so that uniformity can be established when reporting on new datasets. This ensures that authors do not split the data in various unconventional manners which render comparisons complex.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] A. Abbas and S. K. Chalup. "Group Emotion Recognition in the Wild by Combining Deep Neural Networks for Facial Expression Classification and Scene-context Analysis". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. Glasgow, UK: ACM, 2017, pp. 561–568. ISBN: 978-1-4503-5543-8.

[3] A. Abbasi, H. Chen, and A. Salem. "Sentiment Analysis in Multiple Languages: Feature Selection for Opinion Classification in Web Forums". In: *ACM Transactions on Information Systems* 26.3 (June 2008), 12:1–12:34. ISSN: 1046-8188.

[4] F. Abdat, C. Maaoui, and A. Pruski. "Human-Computer Interaction Using Emotion Recognition from Facial Expression". In: *2011 UKSim 5th European Symposium on Computer Modeling and Simulation*. 2011, pp. 196–201.

[5] G. Acampora and G. Cosma. "A hybrid computational intelligence approach for efficiently evaluating customer sentiments in E-commerce reviews". In: *Intelligent Agents (IA), 2014 IEEE Symposium on*. 2014, pp. 73–80.

[6] N. Aifanti, C. Papachristou, and A. Delopoulos. "The MUG facial expression database". In: *11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10*. 2010, pp. 1–4.

[7] M. Al-Shabi, W. P. Cheah, and T. Connie. "Facial Expression Recognition Using a Hybrid CNN-SIFT Aggregator". In: *CoRR* abs/1608.02833 (2016). arXiv: 1608.02833.

[8] Alias-i. *LingPipe 4.1.0*. http://alias-i.com/lingpipe. 2008.

[9] S. Alizadeh and A. Fazel. "Convolutional Neural Networks for Facial Expression Recognition". In: *arXiv preprint arXiv:1704.06756* (2017).

[10] N. S. Altman. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression". In: *The American Statistician* 46.3 (1992), pp. 175–185. ISSN: 00031305. URL: http://www.jstor.org/stable/2685209.

[11] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 0691129932, 9780691129938.

[12] J. Arifovic and R. Gençay. "Using genetic algorithms to select architecture of a feedforward artificial neural network". In: *Physica A: Statistical Mechanics and its Applications* 289.3 (2001), pp. 574 –594. ISSN: 0378-4371.

[13] O. Arriaga, M. Valdenegro-Toro, and P. Plöger. "Real-time Convolutional Neural Networks for Emotion and Gender Classification". In: *arXiv preprint arXiv:1710.07557* (2017).

[14] D. Ashlock. *Evolutionary Computation for Modeling and Optimization*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 1441919694, 9781441919694.

[15] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY, USA: Oxford University Press, Inc., 1996. ISBN: 0-19-509971-0.

[16] E. Barsoum, C. Zhang, C. C. Ferrer, and Z. Zhang. "Training Deep Networks for Facial Expression Recognition with Crowd-sourced Label Distribution". In: *Proceedings of the 18th ACM International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction 2016. Tokyo, Japan: ACM, 2016, pp. 279–283. ISBN: 978-1-4503-4556-9.

[17] T. Baumgartner and J. Culpepper. "Deep Architectures for Face Attributes". In: *Computer Vision – ACCV 2016 Workshops: ACCV 2016 International Workshops, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part II*. Ed. by C.-S. Chen, J. Lu, and K.-K. Ma. Cham: Springer International Publishing, 2017, pp. 334–344. ISBN: 978-3-319-54427-4.

[18] S. Bazrafkan, T. Nedelcu, P. Filipczuk, and P. Corcoran. "Deep learning for facial expression recognition: A step closer to a smartphone that knows your moods". In: *2017 IEEE International Conference on Consumer Electronics (ICCE)*. 2017, pp. 217–220.

[19] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[20] T. Blickle and L. Thiele. "A Comparison of Selection Schemes Used in Evolutionary Algorithms". In: *Evolutionary Computation* 4.4 (1996), pp. 361–394. ISSN: 1063-6560.

[21] T. Borovicka, M. Jirina Jr, P. Kordik, and M. Jirina. "Selecting representative data sets". In: *Advances in data mining knowledge discovery and applications*. InTech, 2012.

[22] I. BoussaïD, J. Lepagnot, and P. Siarry. "A survey on optimization metaheuristics". In: *Information Sciences* 237 (2013), pp. 82–117.

[23] M. Bramer. *Principles of data mining*. Vol. 180. Springer, 2007.

[24] R. Breuer. *A Deep Learning Perspective on the Origin of Facial Expressions*. dissertation. 2017.

[25] P. Burkert, F. Trier, M. Z. Afzal, A. Dengel, and M. Liwicki. "Dexpression: Deep convolutional neural network for expression recognition". In: *arXiv preprint arXiv:1509.05371* (2015).

[26] M. Campbell, A. Hoane, and F. hsiung Hsu. "Deep Blue". In: *Artificial Intelligence* 134.1 (2002), pp. 57 –83. ISSN: 0004-3702.

[27] J. P. Canário and L. Oliveira. "Recognition of Facial Expressions Based on Deep Conspicuous Net". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 20th Iberoamerican Congress, CIARP 2015, Montevideo, Uruguay, November 9-12, 2015, Proceedings*. Ed. by A. Pardo and J. Kittler. Cham: Springer International Publishing, 2015, pp. 255–262. ISBN: 978-3-319-25751-8.

[28] R. Carter. *The Brain Book: An Illustrated Guide to Its Structure, Functions, and Disorders*. Dorling Kindersley Ltd, 2014.

[29] J. Carvalho, A. Prado, and A. Plastino. "A Statistical and Evolutionary Approach to Sentiment Analysis". In: *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 02*. WI-IAT '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 110–117. ISBN: 978-1-4799-4143-8.

[30] W. Che, Y. Zhao, H. Guo, Z. Su, and T. Liu. "Sentence Compression for Aspect-Based Sentiment Analysis". In: *Audio, Speech, and Language Processing, IEEE/ACM Transactions on* 23.12 (2015), pp. 2111–2124. ISSN: 2329-9290.

[31] W. Che, Y. Zhao, H. Guo, Z. Su, and T. Liu. "Sentence Compression for Aspect-based Sentiment Analysis". In: *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 23.12 (Dec. 2015), pp. 2111–2124. ISSN: 2329-9290.

[32] J. Chen, Q. Ou, Z. Chi, and H. Fu. "Smile Detection in the Wild with Deep Convolutional Neural Networks". In: *Mach. Vision Appl.* 28.1-2 (Feb. 2017), pp. 173–183. ISSN: 0932-8092.

[33] M. Chen, L. Zhang, and J. P. Allebach. "Learning deep features for image emotion classification". In: *2015 IEEE International Conference on Image Processing (ICIP)*. 2015, pp. 4491–4495.

[34] Z. Cheng, Q. Yang, and B Sheng. "Deep Colorization". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 415–423. ISBN: 978-1-4673-8391-2.

[35] F. Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[36] F. Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357.

[37] N. Churamani, M. Kerzel, E. Strahl, P. Barros, and S. Wermter. "Teaching emotion expressions to a human companion robot using deep neural architectures". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 627–634.

[38] D. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *CoRR* abs/1511.07289 (2015).

[39] J. Cockburn, M. Bartlett, J. Tanaka, J. Movellan, M. Pierce, and R. Schultz. "Smilemaze: A tutoring system in real-time facial expression perception and production in children with autism spectrum disorder". In: *ECAG 2008 workshop facial and bodily expressions for control and adaptation of games*. Amsterdam. 2008, p. 3.

[40] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. "EMNIST: an extension of MNIST to handwritten letters". In: *arXiv preprint arXiv: 1702.05373* (2017).

[41] R. Collobert, K. Kavukcuoglu, and C. Farabet. "Torch7: A Matlab-like Environment for Machine Learning". In: *BigLearn, NIPS Workshop*. 2011.

[42] C. A. Corneanu, M. O. Simón, J. F. Cohn, and S. E. Guerrero. "Survey on RGB, 3D, Thermal, and Multimodal Approaches for Facial Expression Recognition: History, Trends, and Affect-Related Applications". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.8 (2016), pp. 1548–1568. ISSN: 0162-8828.

[43] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. New York, NY, USA: Cambridge University Press, 2000. ISBN: 0-521-78019-5.

[44] C. D. Căleanu. "Face expression recognition: A brief overview of the last decade". In: *2013 IEEE 8th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. 2013, pp. 157–161.

[45] R. S. C. da Rocha, L. Forero, H. de Mello, M. Kohler, and M. Vellasco. "Polarity classification on web-based reviews using Support Vector Machine". In: *2016 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*. 2016, pp. 1–6.

[46] P. R. Dachapally. "Facial Emotion Detection Using Convolutional Neural Networks and Representational Autoencoder Units". In: abs/1706.01509 (2017). arXiv: 1706.01509.

[47] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*. CVPR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893. ISBN: 0-7695-2372-2.

[48] A. Damien et al. *TFLearn*. https://github.com/tflearn/tflearn. 2016.

[49] C. Darwin. *The Expression of the Emotions in Man and Animals*. John Murray, 1872.

[50] A Das and S Bandyopadhyay. "Subjectivity Detection using Genetic Algorithm". In: *1st Workshop on Computational Approaches to Subjectivity and Sentiment Analysis (WASSA10)*. 2010.

[51] A. Dehghan, E. G. Ortiz, G. Shu, and S. Z. Masood. "DAGER: Deep Age, Gender and Emotion Recognition Using Convolutional Neural Network". In: *CoRR* abs/1702.04280 (2017). arXiv: 1702.04280.

[52] T. Desell. "Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing". In: *arXiv preprint arXiv:1703.05422* (2017).

[53] T. Devries, K. Biswaranjan, and G. W. Taylor. "Multi-task Learning of Facial Landmarks and Expression". In: *2014 Canadian Conference on Computer and Robot Vision*. 2014, pp. 98–103.

[54] A. Dhall, R. Goecke, S. Lucey, and T. Gedeon. "Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 2011, pp. 2106–2112.

[55] A. Dhall, J. Joshi, K. Sikka, R. Goecke, and N. Sebe. "The more the merrier: Analysing the affect of a group of people in images". In: *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. Vol. 1. 2015, pp. 1–8.

[56] A. Dhall, R. Goecke, S. Ghosh, J. Joshi, J. Hoey, and T. Gedeon. "From Individual to Group-level Emotion Recognition: EmotiW 5.0". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction 2017. Glasgow, UK: ACM, 2017, pp. 524–528. ISBN: 978-1-4503-5543-8.

[57] A. Dhall, O. Ramana Murthy, R. Goecke, J. Joshi, and T. Gedeon. "Video and Image Based Emotion Recognition Challenges in the Wild: EmotiW 2015". In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction '15. Seattle, Washington, USA: ACM, 2015, pp. 423–426. ISBN: 978-1-4503-3912-4.

[58] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. de Almeida, B. McFee, H. Weideman, G. Takács, P. de Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, and J. Degrave. *Lasagne: First release.* Aug. 2015. URL: http://dx.doi.org/10.5281/zenodo.27878.

[59] H. Ding, S. K. Zhou, and R. Chellappa. "FaceNet2ExpNet: Regularizing a Deep Face Recognition Net for Expression Recognition". In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*. 2017, pp. 118–126.

[60] D. F. Dinges, S. Venkataraman, E. L. McGlinchey, and D. N. Metaxas. "Monitoring of facial stress during space flight: Optical computer recognition combining discriminative and generative methods". In: *Acta Astronautica* 60.4 (2007). Benefits of human presence in space - historical, scientific, medical, cultural and political aspects. A selection of papers presented at the 15th IAA Humans in Space Symposium, Graz, Austria, 2005, pp. 341 –350. ISSN: 0094-5765.

[61] S. Du, Y. Tao, and A. M. Martinez. "Compound facial expressions of emotion". In: *Proceedings of the National Academy of Sciences* 111.15 (2014), E1454–E1462.

[62] J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[63] E. Dufourq and B. A. Bassett. "Automated Problem Identification: Regression vs Classification via Evolutionary Deep Networks". In: *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*. SAICSIT '17. ACM, 2017. ISBN: 978-1-4503-5250-5.

[64] E. Dufourq and B. A. Bassett. "EDEN: Evolutionary deep networks for efficient machine learning". In: *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*. 2017, pp. 110–115.

[65] E. Dufourq. *Data Classification Using Genetic Programming*. 2015.

[66] D. Duncan, G. Shine, and C. English. "Facial Emotion Recognition in Real Time". In: ().

[67] U. Dwivedi, K. Ahuja, R. Islam, F. A. Barbhuiya, S. Nagar, and K. Dey. "EyamKayo: Interactive Gaze and Facial Expression Captcha". In: *Proceedings of the 22Nd International Conference on Intelligent User Interfaces Companion*. IUI '17 Companion. Limassol, Cyprus: ACM, 2017, pp. 53–56. ISBN: 978-1-4503-4893-5.

[68] EffectiveSoft. *Intellexer*. http://www.intellexer.com/. 2000.

[69] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003. ISBN: 3540401849.

[70] P. Ekman and W. V. Friesen. "Constants across cultures in the face and emotion." In: *Journal of personality and social psychology* 17.2 (1971), p. 124.

[71] A. El Ali, T. Wallbaum, M. Wasmann, W. Heuten, and S. C. Boll. "Face2Emoji: Using Facial Emotional Expressions to Filter Emojis". In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. CHI EA '17. Denver, Colorado, USA: ACM, 2017, pp. 1577–1584. ISBN: 978-1-4503-4656-6.

[72] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. "Dermatologist-level classification of skin cancer with deep neural networks". In: *Nature* 542.7639 (2017), p. 115.

[73] B. Fasel. "Head-pose invariant facial expression recognition using convolutional neural networks". In: *Proceedings. Fourth IEEE International Conference on Multimodal Interfaces*. 2002, pp. 529–534.

[74] B. Fasel. "Robust face analysis using convolutional neural networks". In: *Object recognition supported by user interaction for service robots*. Vol. 2. 2002, 40–43 vol.2.

[75] L. V. Fausett et al. *Fundamentals of neural networks: architectures, algorithms, and applications*. Vol. 3. Prentice-Hall Englewood Cliffs, 1994.

[76] D. L. Felten, M. K. O'Banion, and M. E. Maida. *Netter's atlas of neuroscience*. Elsevier Health Sciences, 2015.

[77] S. Feng, D. Wang, G. Yu, B. Li, and K.-F. Wong. "A Chinese Sentence Compression Method for Opinion Mining". In: *Information Retrieval Technology: 6th Asia Information Retrieval Societies Conference, AIRS 2010, Taipei, Taiwan, December 1-3, 2010. Proceedings*. Ed. by P.-J. Cheng, M.-Y. Kan, W. Lam, and P. Nakov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 320–329. ISBN: 978-3-642-17187-1.

[78] C. Ferreira. "Gene Expression Programming: a New Adaptive Algorithm for Solving Problems". In: *Complex Systems* 13.2 (), pp. 87–129.

[79] X. Gastaldi. "Shake-Shake regularization". In: *arXiv preprint 1705.07485* (2017).

[80] M. Ghayoumi. "A Quick Review of Deep Learning in Facial Expression". In: *Journal of Communication and Computer* 14 (2017), pp. 34–38.

[81] P. Giannopoulos, I. Perikos, and I. Hatzilygeroudis. "Deep Learning Approaches for Facial Emotion Recognition: A Case Study on FER-2013". In: *Advances in Hybridization of Intelligent Methods: Models, Systems and Applications*. Ed. by I. Hatzilygeroudis and V. Palade. Cham: Springer International Publishing, 2018, pp. 1–16. ISBN: 978-3-319-66790-4.

[82] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256.

[83] A. Go, R. Bhayani, and L. Huang. "Twitter sentiment classification using distant supervision". In: *Technical report, Stanford* (2009).

[84] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.

[85] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[86] I. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio. "Challenges in representation learning: A report on three machine learning contests". In: *Neural Networks* 64 (2015). Special Issue on "Deep Learning of Representations", pp. 59 –63. ISSN: 0893-6080.

[87] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[88] M Govindarajan. "Sentiment Analysis of Movie Reviews using Hybrid Method of Naive Bayes and Genetic Algorithm". In: *International Journal of Advanced Computer Research* 3.4 (2013).

[89] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker. "Multi-PIE". In: *2008 8th IEEE International Conference on Automatic Face Gesture Recognition*. 2008, pp. 1–8.

[90] A. Gudi. "Recognizing Semantic Features in Faces using Deep Learning". In: *CoRR* abs/1512.00743 (2015). arXiv: 1512.00743.

[91] J. Guo, S. Zhou, J. Wu, J. Wan, X. Zhu, Z. Lei, and S. Z. Li. "Multi-modality Network with Visual and Geometrical Information for Micro Emotion Recognition". In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*. 2017, pp. 814–819.

[92] X. Guo, L. F. Polanía, and K. E. Barner. "Group-level Emotion Recognition Using Deep Models on Image Scene, Faces, and Skeletons". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction 2017. Glasgow, UK: ACM, 2017, pp. 603–608. ISBN: 978-1-4503-5543-8.

[93] A. Gurnani, V. Gajjar, V. Mavani, and Y. Khandhediya. "VEGAC: Visual Saliency-based Age, Gender, and Facial Expression Classification Using Convolutional Neural Networks". In: *arXiv preprint arXiv:1803.05719* (2018).

[94] D. Hamester, P. Barros, and S. Wermter. "Face expression recognition with a 2-channel Convolutional Neural Network". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8.

[95] K. Hashimoto, F. Saito, T. Yamamoto, and K. Ikeda. "A field study of the human support robot in the home environment". In: *Advanced Robotics and its Social Impacts (ARSO), 2013 IEEE Workshop on*. IEEE. 2013, pp. 143–150.

[96] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[97] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

[98] M. A. Hearst. "Support Vector Machines". In: *IEEE Intelligent Systems* 13.4 (July 1998), pp. 18–28. ISSN: 1541-1672.

[99] G. Hinton, N. Srivastava, and K. Swersky. *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent*. 2012.

[100] T. Hinz, P. Barros, and S. Wermter. "The Effects of Regularization on Learning Facial Expressions with Convolutional Neural Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II*. Ed. by A. E. Villa, P. Masulli, and A. J. Pons Rivero. Cham: Springer International Publishing, 2016, pp. 80–87. ISBN: 978-3-319-44781-0.

[101] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667.

[102] F. Howard. *Wilbur and Orville: a biography of the Wright brothers*. Dover Publications, 1987.

[103] M. Hu and B. Liu. "Mining and Summarizing Customer Reviews". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. Seattle, WA, USA: ACM, 2004, pp. 168–177. ISBN: 1-58113-888-1.

[104] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, et al. "An empirical evaluation of deep learning on highway driving". In: *arXiv preprint arXiv:1504.01716* (2015).

[105] M. A. J. Idrissi, H. Ramchoun, Y. Ghanou, and M. Ettaouil. "Genetic algorithm for neural network architecture optimization". In: *2016 3rd International Conference on Logistics Operations Management (GOL)*. 2016, pp. 1–4.

[106] International Business Machines Watson. *AlchemyAPI Service*. http://www.alchemyapi.com/. 2005.

[107] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. International Conference on Machine Learning'15. Lille, France: JMLR.org, 2015, pp. 448–456.

[108] R. Jack and P. Schyns. "The Human Face as a Dynamic Tool for Social Communication". In: *Current Biology* 25.14 (2015), R621 –R634. ISSN: 0960-9822.

[109] A. Jan, H. Ding, H. Meng, L. Chen, and H. Li. "Accurate Facial Parts Localization and Deep Learning for 3D Facial Expression Recognition". In: *arXiv preprint arXiv:1803.05846* (2018).

[110] J. Jeon, J.-C. Park, Y. Jo, C. Nam, K.-H. Bae, Y. Hwang, and D.-S. Kim. "A Real-time Facial Expression Recognizer Using Deep Neural Network". In: *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*. IMCOM '16. Danang, Viet Nam: ACM, 2016, 94:1–94:4. ISBN: 978-1-4503-4142-4.

[111] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[112] R. Johnson and T. Zhang. "Effective use of word order for text categorization with convolutional neural networks". In: *arXiv preprint arXiv:1412.1058* (2014).

[113] H. Jung, S. Lee, S. Park, B. Kim, J. Kim, I. Lee, and C. Ahn. "Development of deep learning-based facial expression recognition system". In: *2015 21st Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV)*. 2015, pp. 1–4.

[114] L. Kaiser, A. N. Gomez, and F. Chollet. "Depthwise Separable Convolutions for Neural Machine Translation". In: *CoRR* abs/1706.03059 (2017). arXiv: 1706.03059.

[115] P. Kalaivani and K. L. Shunmuganathan. "An improved K-nearest-neighbor algorithm using genetic algorithm for sentiment classification". In: *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*. 2014, pp. 1647–1651.

[116] P. Kalaivani and K. L. Shunmuganathan. "Feature Reduction Based on Genetic Algorithm and Hybrid Model for Opinion Mining". In: *Scientific Programming* 2015 (Jan. 2015), 12:12–12:12. ISSN: 1058-9244.

[117] T. Kanade. *Picture Processing System by Computer Complex and Recognition of Human Faces*. 1973.

[118] P. Khorrami, T. L. Paine, and T. S. Huang. "Do Deep Neural Networks Learn Facial Action Units When Doing Expression Recognition?" In: *Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. ICCVW '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 19–27. ISBN: 978-1-4673-9711-7.

[119] P. R. Khorrami. "How deep learning can help emotion recognition". dissertation. University of Illinois, 2017.

[120] B. K. Kim, S. Y. Dong, J. Roh, G. Kim, and S. Y. Lee. "Fusing Aligned and Non-aligned Face Information for Automatic Affect Recognition in the Wild: A Deep Learning Approach". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2016, pp. 1499–1508.

[121] B.-K. Kim, H. Lee, J. Roh, and S.-Y. Lee. "Hierarchical Committee of Deep CNNs with Exponentially-Weighted Decision Fusion for Static Facial Expression Recognition". In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction '15. Seattle, Washington, USA: ACM, 2015, pp. 427–434. ISBN: 978-1-4503-3912-4.

[122] Y. Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[123] D. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[124] B. C. Ko. "A Brief Review of Facial Emotion Recognition Based on Visual Information". In: *Sensors* 18.2 (2018), p. 401.

[125] P. P. Koltsov. "Comparative study of texture detection and classification algorithms". In: *Computational Mathematics and Mathematical Physics* 51.8 (2011), p. 1460.

[126] S. B. Kotsiantis. "Supervised Machine Learning: A Review of Classification Techniques". In: *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2007, pp. 3–24. ISBN: 978-1-58603-780-2. URL: http://dl.acm.org/citation.cfm?id=1566770.1566773.

[127] S. B. Kotsiantis. "Decision trees: a recent overview". In: *Artificial Intelligence Review* 39.4 (2013), pp. 261–283.

[128] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992. ISBN: 0-262-11170-5.

[129] A. Krizhevsky. "Learning multiple layers of features from tiny images". In: (2009).

[130] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[131] J. C. Krzysztof, W Pedrycz, R. Swiniarski, and L. Kurgan. "Data mining: A knowledge discovery approach". In: *Springer* (2007).

[132] J. Kumari, R. Rajesh, and K. Pooja. "Facial Expression Recognition: A Survey". In: *Procedia Computer Science* 58 (2015). Second International Symposium on Computer Vision and the Internet (VisionNet'15), pp. 486 –491. ISSN: 1877-0509.

[133] P. J. M. Laarhoven and E. H. L. Aarts, eds. *Simulated Annealing: Theory and Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1987. ISBN: 9-027-72513-6.

[134] O. Langner, R. Dotsch, G. Bijlstra, D. H. Wigboldus, S. T. Hawk, and A. Van Knippenberg. "Presentation and validation of the Radboud Faces Database". In: *Cognition and emotion* 24.8 (2010), pp. 1377–1388.

[135] C. P. Latha and M. Priya. "A Review on Deep Learning Algorithms for Speech and Facial Emotion Recognition". In: *APTIKOM Journal on Computer Science and Information Technologies* 1.3 (2016), pp. 88–104.

[136] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. "Face recognition: a convolutional neural-network approach". In: *IEEE Transactions on Neural Networks* 8.1 (1997), pp. 98–113. ISSN: 1045-9227.

[137] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. ISSN: 0018-9219.

[138] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[139] Y. LeCun et al. "Generalization and network design strategies". In: *Connectionism in perspective* (1989), pp. 143–155.

[140] I. Lee, H. Jung, C. H. Ahn, J. Seo, J. Kim, and O. Kwon. "Real-time personalized facial expression recognition system based on deep learning". In: *2016 IEEE International Conference on Consumer Electronics (ICCE)*. 2016, pp. 267–268.

[141] J. Leskovec and A. Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data. June 2014.

[142] Lexalytics. https://www.lexalytics.com/. 2003.

[143] H. Li, J. Sun, Z. Xu, and L. Chen. "Multimodal 2D+3D Facial Expression Recognition With Deep Fusion Convolutional Neural Network". In: *IEEE Transactions on Multimedia* 19.12 (2017), pp. 2816–2831. ISSN: 1520-9210.

[144] J. Li, D. Zhang, J. Zhang, J. Zhang, T. Li, Y. Xia, Q. Yan, and L. Xun. "Facial Expression Recognition with Faster R-CNN". In: *Procedia Computer Science* 107 (2017). Advances in Information and Communication Technology: Proceedings of 7th International Congress of Information and Communication Technology (ICICT2017), pp. 135 –140. ISSN: 1877-0509.

[145] W. Li, M. Li, Z. Su, and Z. Zhu. "A deep-learning approach to facial expression recognition with candid images". In: *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*. 2015, pp. 279–282.

[146] W. Li, F. Abtahi, C. Tsangouri, and Z. Zhu. "Towards an "In-the-Wild" Emotion Dataset Using a Game-Based Framework". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2016 IEEE Conference on*. IEEE. 2016, pp. 1526–1534.

[147] M. Lichman. *UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml. 2013.

[148] B. Liu. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press, 2015.

[149] K. Liu, M. Zhang, and Z. Pan. "Facial Expression Recognition with CNN Ensemble". In: *2016 International Conference on Cyberworlds (CW)*. 2016, pp. 163–166.

[150] M. Liu, S. Li, S. Shan, and X. Chen. "AU-aware Deep Networks for facial expression recognition". In: *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. 2013, pp. 1–6.

[151] Y. Liu and Y. Chen. "Recognition of facial expression based on CNN-CBP features". In: *2017 29th Chinese Control And Decision Conference (CCDC)*. 2017, pp. 2139–2145.

[152] A. T. Lopes, E. de Aguiar, A. F. D. Souza, and T. Oliveira-Santos. "Facial expression recognition with Convolutional Neural Networks: Coping with few data and the training sample order". In: *Pattern Recognition* 61 (2017), pp. 610 –628. ISSN: 0031-3203.

[153] P. Lucey, J. Cohn, S. Lucey, I. Matthews, S. Sridharan, and K. M. Prkachin. "Automatically detecting pain using facial actions". In: *2009 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops*. 2009, pp. 1–8.

[154] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews. "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. 2010, pp. 94–101.

[155] D. Lundqvist, F. A., and A. Ohman. *The Karolinska Directed Emotional Faces – KDEF*. 1998.

[156] Z. Luo, J. Chen, T. Takiguchi, and Y. Ariki. "Facial Expression Recognition with deep age". In: *2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. 2017, pp. 657–662.

[157] I. Lusi, J. C.S. J. Junior, J. Gorbova, X. Baró, S. Escalera, H. Demirel, J. Allik, C. Ozcinar, and G. Anbarjafari. "Joint Challenge on Dominant and Complementary Emotion Recognition Using Micro Emotion Features and Head-Pose Estimation: Databases". In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*. 2017, pp. 809–813.

[158] M. Lyons, S. Akamatsu, M. Kamachi, and J. Gyoba. "Coding facial expressions with Gabor wavelets". In: *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*. 1998, pp. 200–205.

[159] M. J. Lyons, J. Budynek, and S. Akamatsu. "Automatic Classification of Single Facial Images". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 21.12 (Dec. 1999), pp. 1357–1362. ISSN: 0162-8828. DOI: 10.1109/34.817413.

[160] A. L. Maas, A. Y. Hannun, and A. Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[161] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 142–150. ISBN: 978-1-932432-87-9.

[162] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Association for Computational Linguistics (ACL) System Demonstrations*. 2014, pp. 55–60.

[163] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics* 19.2 (June 1993), pp. 313–330. ISSN: 0891-2017.

[164] B. Martinez and M. F. Valstar. "Advances, Challenges, and Opportunities in Automatic Facial Expression Recognition". In: *Advances in Face Detection and Facial Image Analysis*. Ed. by M. Kawulok, M. E. Celebi, and B. Smolka. Cham: Springer International Publishing, 2016, pp. 63–100. ISBN: 978-3-319-25958-1.

[165] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda. "Subject Independent Facial Expression Recognition with Robust Face Detection Using a Convolutional Neural Network". In: *Neural Networks* 16.5-6 (June 2003), pp. 555–559. ISSN: 0893-6080.

[166] V. Mavani, S. Raman, and K. P. Miyapuram. "Facial expression recognition using visual saliency and deep learning". In: *arXiv preprint arXiv:1708.08016* (2017).

[167] V. Mayya, R. M. Pai, and M. M. Pai. "Automatic Facial Expression Recognition Using DCNN". In: *Procedia Computer Science* 93 (2016). Proceedings of the 6th International Conference on Advances in Computing and Communications, pp. 453 –461. ISSN: 1877-0509.

[168] J. McAuley, R. Pandey, and J. Leskovec. "Inferring Networks of Substitutable and Complementary Products". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining. KDD '15. Sydney, NSW, Australia: ACM, 2015, pp. 785–794. ISBN: 978-1-4503-3664-2.

[169] P. E. McKnight, K. M. McKnight, S. Sidani, and A. J. Figueredo. *Missing data: A gentle introduction*. Guilford Press, 2007.

[170] MeaningCloud. *MeaningCloud is a brand by MeaningCloud LLC, a s | ngular company*. https://www.meaningcloud.com/. 2015.

[171] A. Mehrabian. *Nonverbal communication*. Routledge, 2017.

[172] Z. Meng, P. Liu, J. Cai, S. Han, and Y. Tong. "Identity-Aware Convolutional Neural Network for Facial Expression Recognition". In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*. 2017, pp. 558–565.

[173] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat. "Evolving Deep Neural Networks". In: *arXiv preprint arXiv:1703.00548* (2017).

[174] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[175] B. L. Miller and D. E. Goldberg. "Genetic Algorithms, Selection Schemes, and the Varying Effects of Noise". In: *Evolutionary Computation* 4.2 (June 1996), pp. 113–131. ISSN: 1063-6560.

[176] G. F. Miller, P. M. Todd, and S. U. Hegde. "Designing Neural Networks Using Genetic Algorithms". In: *Proceedings of the 3rd International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 379–384. ISBN: 1-55860-066-3.

[177] T. M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077, 9780070428072.

[178] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[179] A. Mollahosseini, D. Chan, and M. H. Mahoor. "Going deeper in facial expression recognition using deep neural networks". In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2016, pp. 1–10.

[180] A. Mollahosseini, B. Hassani, M. J. Salvador, H. Abdollahi, D. Chan, and M. H. Mahoor. "Facial expression recognition from world wild web". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2016 IEEE Conference on*. IEEE. 2016, pp. 1509–1516.

[181] N. Mousavi, H. Siqueira, P. Barros, B. Fernandes, and S. Wermter. "Understanding how deep neural networks learn face expressions". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016, pp. 227–234.

[182] V. N. Murthy, S. Maji, and R. Manmatha. "Automatic Image Annotation Using Deep Learning Representations". In: *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ICMR '15. Shanghai, China: ACM, 2015, pp. 603–606. ISBN: 978-1-4503-3274-3.

[183] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7.

[184] V. Narayanan, I. Arora, and A. Bhatia. "Fast and Accurate Sentiment Classification Using an Enhanced Naive Bayes Model". In: *Proceedings of the 14th International Conference on Intelligent Data Engineering and Automated Learning — IDEAL 2013 - Volume 8206*. IDEAL 2013. Hefei, China: Springer-Verlag New York, Inc., 2013, pp. 194–201. ISBN: 978-3-642-41277-6.

[185] Y. Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)". In: *Doklady AN USSR*. Vol. 269. 1983, pp. 543–547.

[186] H.-W. Ng, V. D. Nguyen, V. Vonikakis, and S. Winkler. "Deep Learning for Emotion Recognition on Small Datasets Using Transfer Learning". In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction '15. Seattle, Washington, USA: ACM, 2015, pp. 443–449. ISBN: 978-1-4503-3912-4.

[187] C. R. Nithyananda, A. C. Ramachandra, and Preethi. "Review on Histogram Equalization based Image Enhancement Techniques". In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. 2016, pp. 2512–2517.

[188] NLTK. http://text-processing.com/. 2010.

[189] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016).

[190] A. Ortony and T. J. Turner. "What's basic about basic emotions?" In: *Psychological review* 97.3 (1990), p. 315.

[191] S. Ouellet. "Real-time emotion recognition for gaming using deep convolutional network features". In: *arXiv preprint arXiv:1408.3750* (2014).

[192] B. Pang and L. Lee. "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts". In: *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. 2004.

[193] B. Pang and L. Lee. "Opinion Mining and Sentiment Analysis". In: *Foundations and Trends in Information Retrieval* 2.1-2 (Jan. 2008), pp. 1–135. ISSN: 1554-0669.

[194] B. Pang, L. Lee, and S. Vaithyanathan. "Thumbs Up?: Sentiment Classification Using Machine Learning Techniques". In: *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*. EMNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 79–86.

[195] M. Pantic and L. J. M. Rothkrantz. "Automatic analysis of facial expressions: the state of the art". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.12 (2000), pp. 1424–1445. ISSN: 0162-8828.

[196] M. Pantic, M. Valstar, R. Rademaker, and L. Maat. "Web-based database for facial expression analysis". In: *2005 IEEE International Conference on Multimedia and Expo*. 2005, 5 pp.–.

[197] K Paramesha and C Ravishankar K. "Optimization of Cross Domain Sentiment Analysis Using Sentiwordnet". In: *International Journal in Foundations of Computer Science & Technology* 3.5 (2013).

[198] O. M. Parkhi, A. Vedaldi, and A. Zisserman. "Deep Face Recognition". In: *British Machine Vision Conference*. 2015.

[199] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic differentiation in PyTorch". In: (2017).

[200] X. Peng, L. Li, X. Feng, and J. Fan. "Spontaneous facial expression recognition by heterogeneous convolutional networks". In: *2017 International Conference on the Frontiers and Advances in Data Science (FADS)*. 2017, pp. 70–73.

[201] X. Peng, Z. Xia, L. Li, and X. Feng. "Towards Facial Expression Recognition in the Wild: A New Database and Deep Recognition System". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2016, pp. 1544–1550.

[202] M. Pietikäinen and G. Zhao. "Two decades of local binary patterns: A survey". In: *Advances in Independent Component Analysis and Learning Machines*. Ed. by E. Bingham, S. Kaski, J. Laaksonen, and J. Lampinen. Academic Press, 2015, pp. 175 –210. ISBN: 978-0-12-802806-3.

[203] V. Pilla Jr, A. Zanellato, C. Bortolini, H. R. Gamba, G. B. Borba, and H. Medeiros. "Facial Expression Classification Using Convolutional Neural Network and Support Vector Machine". In: (2016).

[204] D. A. Pitaloka, A. Wulandari, T. Basaruddin, and D. Y. Liliana. "Enhancing CNN with Preprocessing Stage in Automatic Emotion Recognition". In: *Procedia Computer Science* 116 (2017). Discovery and innovation of computer science technology in artificial intelligence era: The 2nd International Conference on Computer Science and Computational Intelligence (ICCSCI 2017), pp. 523 –529. ISSN: 1877-0509.

[205] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008. ISBN: 1409200736, 9781409200734.

[206] B. Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1 –17. ISSN: 0041-5553.

[207] G. Pons and D. Masip. "Supervised Committee of Convolutional Neural Networks in Automated Facial Expression Analysis". In: *IEEE Transactions on Affective Computing* PP.99 (2017), pp. 1–1. ISSN: 1949-3045.

[208] G. Pons and D. Masip. "Multi-task, multi-label and multi-domain learning with residual convolutional networks for emotion recognition". In: *arXiv preprint arXiv:1802.06664* (2018).

[209] C. Pramerdorfer and M. Kampel. "Facial Expression Recognition using Convolutional Neural Networks: State of the Art". In: *arXiv preprint arXiv:1612.02903* (2016).

[210] A. Raghuvanshi and V. Choksi. "Facial Expression Recognition with Convolutional Neural Networks". In: (2016).

[211] P. Rajpurkar, J. Irvin, K. Zhu, B. Yang, H. Mehta, T. Duan, D. Ding, A. Bagul, C. Langlotz, K. Shpanskaya, et al. "Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning". In: *arXiv preprint arXiv:1711.05225* (2017).

[212] P. Ramachandran, B. Zoph, and Q. V. Le. "Searching for Activation Functions". In: *CoRR* abs/1710.05941 (2017).

[213] M. Ranzato, J. Susskind, V. Mnih, and G. Hinton. "On deep generative models with applications to recognition". In: *CVPR 2011*. 2011, pp. 2857–2864.

[214] T. A. Rashid. "Convolutional Neural Networks based Method for Improving Facial Expression Recognition". In: *Intelligent Systems Technologies and Applications 2016*. Ed. by J. M. Corchado Rodriguez, S. Mitra, S. M. Thampi, and E.-S. El-Alfy. Cham: Springer International Publishing, 2016, pp. 73–84. ISBN: 978-3-319-47952-1.

[215] A. Rassadin, A. Gruzdev, and A. Savchenko. "Group-level Emotion Recognition Using Transfer Learning from Face Identification". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction 2017. Glasgow, UK: ACM, 2017, pp. 544–548. ISBN: 978-1-4503-5543-8.

[216] K. Ravi and V. Ravi. "A Survey on Opinion Mining and Sentiment Analysis". In: *Knowledge-Based Systems* 89.C (Nov. 2015), pp. 14–46. ISSN: 0950-7051.

[217] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. "Large-scale evolution of image classifiers". In: *arXiv preprint arXiv:1703.01041* (2017).

[218] S. Rifai, Y. Bengio, A. Courville, P. Vincent, and M. Mirza. "Disentangling Factors of Variation for Facial Expression Recognition". In: *Computer Vision – ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI*. Ed. by A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid. Berlin,

Heidelberg: Springer Berlin Heidelberg, 2012, pp. 808–822. ISBN: 978-3-642-33783-3.

[219] F. Romei. *Leonardo Da Vinci*. The Oliver Press, 2008.

[220] S. Roychowdhury and M. Emmons. "A Survey of the Trends in Facial and Expression Recognition Databases and Methods". In: *International Journal of Computer Science and Engineering Survey (IJCSES)* 6.5 (2015).

[221] A. Ruiz-Garcia, M. Elshaw, A. Altahhan, and V. Palade. "Stacked deep convolutional auto-encoders for emotion recognition from facial expressions". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 1586–1593.

[222] A. Ruiz-Garcia, M. Elshaw, A. Altahhan, and V. Palade. "Deep Learning for Emotion Recognition in Faces". In: *Artificial Neural Networks and Machine Learning – ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II*. Ed. by A. E. Villa, P. Masulli, and A. J. Pons Rivero. Cham: Springer International Publishing, 2016, pp. 38–46. ISBN: 978-3-319-44781-0.

[223] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Neurocomputing: Foundations of Research". In: ed. by J. A. Anderson and E. Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6.

[224] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM J. Res. Dev.* 3.3 (July 1959), pp. 210–229. ISSN: 0018-8646.

[225] D. V. Sang, N. V. Dat, and D. P. Thuan. "Facial Expression Recognition Using Deep Convolutional Neural Networks". In: *International Conference on Knowledge and Systems Engineering (KSE)* (2017).

[226] K. A. Sannikov, A. A. Bashlikov, and A. A. Druki. "Two-level algorithm of facial expressions classification on complex background". In: *2017 International Siberian Conference on Control and Communications (SIBCON)*. 2017, pp. 1–5.

[227] E. Sariyanidi, H. Gunes, and A. Cavallaro. "Automatic Analysis of Facial Affect: A Survey of Registration, Representation, and Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.6 (2015), pp. 1113–1133. ISSN: 0162-8828.

[228] A. Savran, N. Alyüz, H. Dibeklioğlu, O. Çeliktutan, B. Gökberk, B. Sankur, and L. Akarun. "Biometrics and Identity Management". In: ed. by B. Schouten, N. C. Juul, A. Drygajlo, and M. Tistarelli. Berlin, Heidelberg: Springer-Verlag, 2008. Chap. Bosphorus Database for 3D Face Analysis, pp. 47–56. ISBN: 978-3-540-89990-7.

[229] J. Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85 –117. ISSN: 0893-6080.

[230] H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*. Vol. 39. Cambridge University Press, 2008.

[231] J. Schwan, E. Ghaleb, E. Hortal, and S. Asteriadis. "High-performance and lightweight real-time deep face emotion recognition". In: *2017 12th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP)*. 2017, pp. 76–79.

[232] D. H. v. Seggern. *CRC Standard Curves and Surfaces with Mathematica, Second Edition (Chapman & Hall/Crc Applied Mathematics and Nonlinear Science)*. Chapman & Hall/CRC, 2006. ISBN: 1584885998.

[233] K. Shan, J. Guo, W. You, D. Lu, and R. Bie. "Automatic facial expression recognition based on a deep convolutional-neural-network structure". In: *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*. 2017, pp. 123–128.

[234] G. I. Sher. *Handbook of Neuroevolution Through Erlang*. Springer Publishing Company, Incorporated, 2012. ISBN: 1461444624, 9781461444626.

[235] M. Shin, M. Kim, and D. S. Kwon. "Baseline CNN structure analysis for facial expression recognition". In: *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 2016, pp. 724–729.

[236] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[237] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[238] P. Smith. "Using genetic algorithms in word-vector optimisation". In: *Computational Intelligence (UKCI), 2010 UK Workshop on*. 2010, pp. 1–5.

[239] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Citeseer. 2013, pp. 1631–1642.

[240] I. Song, H. J. Kim, and P. B. Jeon. "Deep learning for real-time robust facial expression recognition on a smartphone". In: *2014 IEEE International Conference on Consumer Electronics (ICCE)*. 2014, pp. 564–567.

[241] SpazioDati S.r.l. *Dandelion API*. http://dandelion.eu/. 2013.

[242] D. Spiers. *Facial emotion detection using deep learning*. dissertation. 2016.

[243] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[244] R. K. Srivastava, K. Greff, and J. Schmidhuber. "Training Very Deep Networks". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 2377–2385.

[245] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning". In: *arXiv preprint arXiv:1712.06567* (2017).

[246] A. Sun, Y.-J. Li, Y.-M. Huang, and Q. Li. "Using Facial Expression to Detect Emotion in E-learning System: A Deep Learning Method". In: *Emerging Technologies for Education: Second International Symposium, SETE 2017, Held in Conjunction with ICWL 2017, Cape Town, South Africa, September 20–22, 2017, Revised Selected Papers*. Ed. by T.-C. Huang, R. Lau, Y.-M. Huang, M. Spaniol, and C.-H. Yuen. Cham: Springer International Publishing, 2017, pp. 446–455. ISBN: 978-3-319-71084-6.

[247] L. Surace, M. Patacchiola, E. Battini Sönmez, W. Spataro, and A. Cangelosi. "Emotion Recognition in the Wild Using Deep Neural Networks and Bayesian Classifiers". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction 2017. Glasgow, UK: ACM, 2017, pp. 593–597. ISBN: 978-1-4503-5543-8.

[248] J. M. Susskind, A. K. Anderson, and G. E. Hinton. "The toronto face database". In: *Department of Computer Science, University of Toronto, Toronto, ON, Canada, Tech. Rep* 3 (2010).

[249] I. Sutskever. "Training recurrent neural networks". PhD thesis. University of Toronto Toronto, Ontario, Canada, 2013.

[250] M. Suwa, N. Sugie, and K. Fujimora. "A preliminary note on pattern recognition of human emotional expression". In: *4th International Joint Conference on Pattern Recognition*. 1978, pp. 408–410.

[251] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9.

[252] E. B. Sönmez and A. Cangelosi. *Convolutional neural networks with balanced batches for facial expressions recognition*. 2017.

[253] M. Taini, G. Zhao, S. Z. Li, and M. Pietikainen. "Facial expression recognition from near-infrared video sequences". In: *2008 19th International Conference on Pattern Recognition*. 2008, pp. 1–4.

[254] L. Tan, K. Zhang, K. Wang, X. Zeng, X. Peng, and Y. Qiao. "Group Emotion Recognition with Individual Facial Emotion CNNs and Global Image Based CNNs". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction 2017. Glasgow, UK: ACM, 2017, pp. 549–552. ISBN: 978-1-4503-5543-8.

[255] Y. Tang. "Deep learning using linear support vector machines". In: *arXiv preprint arXiv:1306.0239* (2013).

[256] A. Taylor, M. Marcus, and B. Santorini. "Treebanks: Building and Using Parsed Corpora". In: ed. by A. Abeillé. Dordrecht: Springer Netherlands, 2003. Chap. The Penn Treebank: An Overview, pp. 5–22. ISBN: 978-94-010-0201-1.

[257] E. D. D. Team. *Deeplearning4j: Open-source distributed deep learning for the JVM*. URL: http://deeplearning4j.org.

[258] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: http://arxiv.org/abs/1605.02688.

[259] M. Thelwall, K. Buckley, and G. Paltoglou. "Sentiment Strength Detection for the Social Web". In: *Journal of the Association for Information Science and Technology* 63.1 (Jan. 2012), pp. 163–173. ISSN: 1532-2882.

[260] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. "Sentiment in Short Strength Detection Informal Text". In: *Journal of the Association for Information Science and Technology* 61.12 (Dec. 2010), pp. 2544–2558. ISSN: 1532-2882.

[261] Y. Tian, T. Kanade, and J. F. Cohn. "Facial expression recognition". In: *Handbook of face recognition*. Springer, 2011, pp. 487–519.

[262] A. Toshev and C. Szegedy. "DeepPose: Human Pose Estimation via Deep Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1653–1660.

[263] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. "Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network". In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*. NAACL '03. Edmonton, Canada: Association for Computational Linguistics, 2003, pp. 173–180.

[264] uClassify. *uClassify*. https://www.uclassify.com/. 2008.

[265] http://mplab.ucsd.edu. *The MPLab GENKI Database*. 2009.

[266] A. Uçar. "Deep Convolutional Neural Networks for facial expression recognition". In: *2017 IEEE International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*. 2017, pp. 371–375.

[267] H. G. Valero. *Automatic Facial Expression Recognition*. dissertation. 2016.

[268] M. F. Valstar, B. Jiang, M. Mehu, M. Pantic, and K. Scherer. "The first facial expression recognition and analysis challenge". In: *Face and Gesture 2011*. 2011, pp. 921–926.

[269] A. Vedaldi and K. Lenc. "MatConvNet: Convolutional Neural Networks for MATLAB". In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM '15. Brisbane, Australia: ACM, 2015, pp. 689–692. ISBN: 978-1-4503-3459-4.

[270] P. A. Vikhar. "Evolutionary algorithms: A critical review and its future prospects". In: *Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 2016 International Conference on*. IEEE. 2016, pp. 261–265.

[271] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, I–511–I–518 vol.1.

[272] D. Vishwanath and S. Gupta. "Adding CNNs to the Mix: Stacking models for sentiment classification". In: *2016 IEEE Annual India Conference (INDICON)*. 2016, pp. 1–4.

[273] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. "Regularization of Neural Networks Using Dropconnect". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. International Conference on Machine Learning'13. Atlanta, GA, USA: JMLR.org, 2013, pp. III–1058–III–1066.

[274] H. Wang, Y. Lu, and C. Zhai. "Latent Aspect Rating Analysis on Review Text Data: A Rating Regression Approach". In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '10. Washington, DC, USA: ACM, 2010, pp. 783–792. ISBN: 978-1-4503-0055-1.

[275] S. Wang, Z. Liu, S. Lv, Y. Lv, G. Wu, P. Peng, F. Chen, and X. Wang. "A Natural Visible and Infrared Facial Expression Database for Expression Recognition and Emotion Inference". In: *IEEE Transactions on Multimedia* 12.7 (2010), pp. 682–691. ISSN: 1520-9210.

[276] A. Waugh and A. Grant. *Ross & Wilson Anatomy and physiology in health and illness E-book*. Elsevier Health Sciences, 2014.

[277] Q. Wei, Y. Zhao, Q. Xu, L. Li, J. He, L. Yu, and B. Sun. "A New Deep-learning Framework for Group Emotion Recognition". In: *Proceedings of the 19th ACM International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction 2017. Glasgow, UK: ACM, 2017, pp. 587–592. ISBN: 978-1-4503-5543-8.

[278] G. Wen, H. Li, and D. Li. "An ensemble convolutional echo state networks for facial expression recognition". In: *2015 International Conference on Affective Computing and Intelligent Interaction (ACII)*. 2015, pp. 873–878.

[279] G. Wen, Z. Hou, H. Li, D. Li, L. Jiang, and E. Xun. "Ensemble of Deep Neural Networks with Probability-Based Fusion for Facial Expression Recognition". In: *Cognitive Computation* 9.5 (2017), pp. 597–610. ISSN: 1866-9964.

[280] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations*. 1999.

[281] H. Wu and X. Gu. "Towards Dropout Training for Convolutional Neural Networks". In: *Neural Netw.* 71.C (Nov. 2015), pp. 1–10. ISSN: 0893-6080.

[282] T. Wu, S. Fu, and G. Yang. "Survey of the Facial Expression Recognition Research". In: *Advances in Brain Inspired Cognitive Systems*. Ed. by H. Zhang, A. Hussain, D. Liu, and Z. Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 392–402. ISBN: 978-3-642-31561-9.

[283] X.-L. Xia, C. Xu, and B. Nan. "Facial Expression Recognition Based on TensorFlow Platform". In: *ITM Web of Conferences*. Vol. 12. EDP Sciences. 2017, p. 01005.

[284] J. Xiang and G. Zhu. "Joint Face Detection and Facial Expression Recognition with MTCNN". In: *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*. 2017, pp. 424–427.

[285] H. Xiao, R. Kashif, and V. Roland. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[286] S. Xie and H. Hu. "Facial expression recognition with FRR-CNN". In: *Electronics Letters* 53.4 (2017), pp. 235–237. ISSN: 0013-5194.

[287] X. Xiong and F. D. la Torre. "Supervised Descent Method and Its Applications to Face Alignment". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 532–539.

[288] M. Xu, W. Cheng, Q. Zhao, L. Ma, and F. Xu. "Facial expression recognition based on transfer learning from deep convolutional networks". In: *2015 11th International Conference on Natural Computation (ICNC)*. 2015, pp. 702–708.

[289] H.-F. Yang, B.-Y. Lin, K.-Y. Chang, and C.-S. Chen. "Joint Estimation of Age and Expression by Combining Scattering and Convolutional Networks". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 14.1 (Jan. 2018), 9:1–9:18. ISSN: 1551-6857.

[290] X. Yao. "Evolving artificial neural networks". In: *Proceedings of the IEEE* 87.9 (1999), pp. 1423–1447. ISSN: 0018-9219.

[291] L. Yin, X. Wei, Y. Sun, J. Wang, and M. J. Rosato. "A 3D facial expression database for facial behavior research". In: *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*. 2006, pp. 211–216.

[292] G. Yolcu, I. Oztel, S. Kazan, C. Oz, K. Palaniappan, T. E. Lever, and F. Bunyak. "Deep learning-based facial expression recognition for monitoring neurological disorders". In: *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. Vol. 00. 2017, pp. 1652–1657.

[293] Z. Yu and C. Zhang. "Image Based Static Facial Expression Recognition with Multiple Deep Network Learning". In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. International Conference on Multimodal Interaction '15. Seattle, Washington, USA: ACM, 2015, pp. 435–442. ISBN: 978-1-4503-3912-4.

[294] M. V. Zavarez, R. F. Berriel, and T. Oliveira-Santos. "Cross-Database Facial Expression Recognition Based on Fine-Tuned Deep Convolutional Network". In: *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. 2017, pp. 405–412.

[295] T. H. Zavaschi, A. S. Britto, L. E. Oliveira, and A. L. Koerich. "Fusion of feature sets and classifiers for facial expression recognition". In: *Expert Systems with Applications* 40.2 (2013), pp. 646 –655. ISSN: 0957-4174.

[296] Y. Zhai, J. Liu, J. Zeng, V. Piuri, F. Scotti, Z. Ying, Y. Xu, and J. Gan. "Deep Convolutional Neural Network for Facial Expression Recognition". In: *Image and Graphics: 9th International Conference, ICIG 2017, Shanghai, China, September 13-15, 2017, Revised Selected Papers, Part I*. Ed. by Y. Zhao, X. Kong, and D. Taubman. Cham: Springer International Publishing, 2017, pp. 211–223. ISBN: 978-3-319-71607-7.

[297] B. T. Zhang and H. Mühlenbein. "Balancing Accuracy and Parsimony in Genetic Programming". In: *Evolutionary Computation* 3.1 (1995), pp. 17–38. ISSN: 1063-6560.

[298] T. Zhang, W. Zheng, Z. Cui, Y. Zong, J. Yan, and K. Yan. "A Deep Neural Network-Driven Feature Learning Method for Multi-view Facial Expression Recognition". In: *IEEE Transactions on Multimedia* 18.12 (2016), pp. 2528–2536. ISSN: 1520-9210.

[299] T. Zhang. "Facial Expression Recognition Based on Deep Learning: A Survey". In: *Advances in Intelligent Systems and Interactive Applications:*

Proceedings of the 2nd International Conference on Intelligent and Interactive Systems and Applications (IISA2017). Ed. by F. Xhafa, S. Patnaik, and A. Y. Zomaya. Cham: Springer International Publishing, 2018, pp. 345–352. ISBN: 978-3-319-69096-4.

[300] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang. "Multiobjective evolutionary algorithms: A survey of the state of the art". In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 32 –49. ISSN: 2210-6502.

[301] S. Zhou, Y. Liang, J. Wan, and S. Z. Li. "Facial Expression Recognition Based on Multi-scale CNNs". In: *Biometric Recognition: 11th Chinese Conference, CCBR 2016, Chengdu, China, October 14-16, 2016, Proceedings*. Ed. by Z. You, J. Zhou, Y. Wang, Z. Sun, S. Shan, W. Zheng, J. Feng, and Q. Zhao. Cham: Springer International Publishing, 2016, pp. 503–510.

[302] Y. Zhou and B. E. Shi. "Action unit selective feature maps in deep networks for facial expression recognition". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2031–2038.

[303] Y. T. Zhou and R. Chellappa. "Computation of optical flow using a neural network". In: *IEEE 1988 International Conference on Neural Networks*. 1988, 71–78 vol.2.

[304] X. Zhu and D. Ramanan. "Face detection, pose estimation, and landmark localization in the wild". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 2879–2886. DOI: 10.1109/CVPR.2012.6248014.

[305] B. Zoph and Q. V. Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016).