VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science
Te Kura Matai Pukaha, Purorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Evolving Network Structures For Deep Learning in Text Classification

Sean Stevenson

Supervisors: Xiaoying Gao, Bing Xue

Submitted in partial fulfilment of the requirements for Bachelor of Engineering with Honours, Majoring in Software Engineering.

Abstract

The aim of this project is to evolve convolutional neural network structures for text classification. Convolutional neural networks are inherently limited by requiring a set of well tuned architecture and parameters to be performant. The process to manually tune these settings is cumbersome and complex. As a solution, we propose a method to evolve a convolutional neural network using Evolutionary Computation methods to lessen this limitation. More specifically, a Particle Swarm Optimisation (PSO) based solution has been developed to automatically evolve the network structures. As part of the PSO solution, two encoding schemes have been developed to evolve either the hyper-parameters, or the architecture as well as the hyper-parameters. These two encoding schemes and subsequent methods surrounding their implementation have been shown to evolve well-performing convolutional neural networks that have promising results compared to state-of-art techniques.

Acknowledgments

To my supervisors Xiaoying Sharon Gao and Bing Xue for all the support they have provided. To Mengjie Zhang for practically being a supervisor during our weekly meetings. To Ying Bi for also helping out at the honours meetings. To Bin Wang and Trevor Londt for providing their support in the early stages of the project. To Ramesh Rayudu for being a supportive and helpful Course Coordinator.

To Yi Sian, Inti, Andrew, Peter and William for being excellent students to work and discuss the projects with.

To Celine and Leyton for having daily discussions about the trials and tribulations this year entailed.

Contents

Figures

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) are a technique within the field of deep learning that perform feature extraction and classification from an input to derive meaning. Whilst most commonly used for analysing the properties within visual imagery, more recently CNNs have been applied to the field of text classification [1, 2]. Text classification is the practice of categorising documents into predefined topics, with applications such as establishing the sentiment of a text and filtering text by established topics for news purposes. CNNs have been shown to outperform classical machine learning methods regarding various text classification problems [1, 2] as established by prior research. The incredible performance of CNNs for text classification is founded in their ability to use a high-dimensional vector representing text, much like how an image is represented to achieve similar performance. The sequence of the text is preserved in this vector format from which the CNN can discover patterns to discern within the text to discern meaning.

However, when using a CNN for text classification there is a major limitation as a consequence of the complexity of the method. The limitation is in the difficulty constructing an optimal architecture for a given problem, such as text classification. A CNN architecture must first be constructed, then it requires a set of suitable hyper-parameters to sufficiently model and train the architecture [3], from which a high accuracy can be achieved. Currently a manual approach is the default method for constructing the architecture and choosing these hyper-parameters. Deciding on the hyper-parameters for a CNN requires domain experts aware of appropriate hyper-parameter settings such as filter window sizes, pooling strategies and many more hyper-parameter interactions to create a model because of the complexity of the system. Optimal hyper-parameter values are critical for achieving a high-classification accuracy. The classification performance of a CNN is sensitive to all hyper-parameters, in addition to it being difficult to discern what hyper-parameter changes result in a difference in ability to classify a text. Developing an exact model with suitable hyper-parameters for the particular issue at hand is a time consuming, error prone and laborious process.

1.1 Proposed Solution

The proposed solution developed through the project work entails an Evolutionary Computation (EC) solution to automatically evolve the hyper-parameters and architecture of a CNN specifically for text classification. The scope of the project permits evolving the hyper-parameters and the architecture. As such, the proposed solution is two-fold. The first part involves an optimisation of the hyper-parameters, where the architecture of a CNN is enforced and the hyper-parameter of each layer are evolved to find a performant set. The

second part involves an optimisation of the architecture, where the architecture and corresponding hyper-parameters are both evolved.

For the two aspects, we propose two novel encoding schemes to encode either the hyper-parameters or the architecture, in order to perform optimisation. These two aspects have been designed and implemented, thus achieving automatic evolution of the hyper-parameters and the architecture of a CNN. This solves the issues mentioned present with CNNs currently. In addition to this, two methods have been developed which further enhance PSO and optimise the efficiency and speed of the method.

To perform the optimisation for this project, Particle Swarm Optimisation (PSO) is used in this project as the proposed solution to perform hyper-parameter optimisation. PSO is an algorithm that uses swarm intelligence principles to improve a candidate solution by optimising an objective. We chose to use PSO as the EC method for this project for the inherent effectiveness of PSO as an optimisation strategy [4].

Using PSO allows for searching multi-dimensional search-space to tune the various hyper-parameters of a CNN architecture, without human intervention, to achieve effective text classification. An evaluation of the system has been performed on popular test datasets, using the baseline classifiers and other state-of-art method results for the datasets to be used as comparisons for the proposed solution.

1.2 Objectives

1. Develop a baseline CNN text classifier using a rudimentary combination of hyper-parameters and architecture.

2. Develop a method to automatically evolve the CNN hyper-parameters using PSO with a new hyperparameter representation.

3. Develop a method to automatically evolve the CNN architecture using PSO with a new architecture representation.

1.3 Contributions

1. A novel encoding scheme for representing CNN hyper-parameters to aid the PSO process. This novel contribution is the first of its kind, and achieves promising results compared to state-of-art methods.

2. A novel variable-length encoding scheme for representing CNN architecture and hyper-parameters to aid the PSO process. This is a significant contribution as it is the first of its kind, as well as being a variable-length encoding scheme, which addresses a major limitation of PSO.

3. Two methods that enhance the performance of PSO when used in tandem with the architecture encoding scheme. This work can be adapted for other indirect encoding schemes thus are applicable beyond the scope of this project work. The PSO-RV method achieves promising results compared to state-of-art methods.

Overall, to the best of the author's knowledge, the project work is the first of its kind to perform evolutionary optimisation on CNNs for text classification, using PSO. This is significant as the project work proves these optimisation methods are worth investigating in future research.

1.4 Organisation

The remainder of this report is organised as follows. A background on the concepts and research literature is provided in Chapter 2. Chapter 3 describes the design of the methods behind the project. Chapter 4 discusses the implementation details from the design section. Chapter 5 sums up the evaluation and analysis of the project, and Chapter 6 summarises the conclusions of this report and the future work.

Chapter 2

Literature Review

2.1 Machine Learning

Machine learning (ML) is the study of computer systems that automatically learn and improve through the analysis of provided example data [5]. ML is a subset of artificial intelligence (AI) based on the premise that a system can learn how to form an accurate decision for a given problem, without explicit human intervention using data [6]. The primary objective is for a ML system to derive patterns in data to build an analytical model that performs these decisions. The performance of an ML system is optimised by the quality and quantity of the example data provided; this helps develop a performant system as clearer patterns can be established to tune the calculation for a particular output adjusted accordingly. The two categories of ML that apply to this project are convolutional neural networks and evolutionary learning. The following sections will elaborate on these.

2.2 Text Classification

Text classification is a machine learning task to label a document from a selection of given labelled categories, based on key features within the text [5]. It is a supervised ML method, meaning the text classifier trains on a labelled dataset to establish a set of features from a piece of data that correspond to a certain label [6]. Text classification is a key application of text mining - the analysis of text to develop an informative understanding of the text presented [5]. Text classification techniques build systems to receive a set of texts as input to develop an understanding of the particular structure, from the words used, the particular sequence of those words, to many more determining features. Text classification has a wide variety of useful applications. Categorising a set of news articles into defined topics for a large news organisation is one [7]. This can extend into organising the contents of libraries, social feeds and scientific literature [5]. Another application is the semantic analysis of a product review to establish the rating. Text classification is a powerful and useful method regarding text based applications.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a form of feed forward deep neural-networks. They were originally designed to perform image processing, achieving excellent classification accuracy [8]. CNNs utilize a series of mathematical operations to learn differences in input data and classify data based on these differences. The first operation performed by a CNN is a convolution. A convolution is a feature extraction operation performed on inputs

that aims to preserve the spatial structure of the input. This operation involves sliding small filter maps over the input, using matrix multiplication to map a NxN matrix to a smaller matrix where key features are present. These maps can capture varying sections of input, dependent on the size of the map. Secondly, the filtered outputs have a downsampling operation applied over to extract out the most relevant features. This operation is referred to as pooling and isolates out high-level features by taking the max, or average values of portions of the feature maps. Finally, the output from this pooling operation is passed to typically one fully connected layer to perform the final classification. This dense section of the network can have any number of layers and neurons per layer. Each neuron has an activation that is used to establish what neurons should fire within the layer. The resultant combination of different neurons activating governs the final output of the network.

2.4 Neuroevolution & Hyper-parameter Optimisation

Neuroevolution is a form of evolution specific to neural networks (NN), with the aim of evolving various characteristics of NNs by applying evolutionary algorithms to optimise performance [9]. The characteristics of NNs that can be evolved include the hyper-parameters of the networks, in addition to their overall structure. Neuroevolution concepts were first formally proposed in 1999 by Yao [10] where he reviews and discusses the effect of EAs being applied to NNs to enhance their learning and performance. The work serves as an introduction to various methodologies to evolve NNs, in addition to proposing a framework for neuroevolution being comprised of three aspects: evolving the weights, structures or learning rules [10]. Additional research by Stanley and Miikkulainen [11] contributes to neuroevolution concepts. Their work proposes a taxonomy that aims to capture the complexity of evolution seen in natural organisms. They formalise five dimensions based on genetic principles that underpin embryonic development, which are used to draw comparisons between various aspects of encoding schemes used in neuroevolution. The works of Stanley and Miikkulainen, and Yao, inform modern neuroevolution research.

As practice of AI, neuroevolution mimics natural evolution through artificially learning optimal network structures. The structures are represented as a population of solutions, where structures of solutions that are strongly performing in an evolution have their structure mimicked and evolved. The evaluation criteria for neuroevolution is an assessment on how performant a network solution is for the given task [9]. Neuroevolution is the premise behind the project, we will utilise the practice of neuroevolution to evolve the network structure of a CNN for text classification.

The practice of neuroevolution for this project is hyper-parameter and architecture optimisation. Hyper-parameter optimisation (HPO) is the automation of hyper-parameter configurations to optimise the performance of a system [12]. All machine learning systems have a set of hyper-parameters that represent how a structure adapts to input and forms an output. A simple analogy for hyper-parameters is a telescope, in that a telescope has the zoom and focus as parameters, where rotating the lens and dials aids in enhancing the clarity of the image, therefore optimising the parameters. Hyper-parameter and architecture optimisation are beneficial to the field of machine learning as it reduces the human effort necessary for building a ML model [12]. Utilising an automatic approach eliminates the need for human intervention in the process. Optimisation can also explore solutions of a system in far greater detail than a human would ever be able to. As the solution to the project, HPO and architecture optimisation will be used to optimise the structure of a CNN for text classification, using an evolutionary computing approach.

2.5 Evolutionary Computation

Evolutionary computation (EC) is a subfield of AI pertaining to algorithms that perform global optimisation with a stochastic evolutionary process akin to biological principles of evolution as discussed in [13]. EC is utilised when the solution pool is too large for classical optimisation algorithms to compute a reasonable solution. For an EC solution, a population of candidate solutions is generated, where each candidate represents partial or potential solutions, that are continually evolved using processes based on evolutionary concepts - natural selection, or genetic inheritance, mutation - and then evaluated. The goal of the evaluation is to find optimal solutions to include in the population of the next generation, and sub-optimal solutions to eliminate from the population like in natural selection, where new solutions are bred from previous well performing solutions to replace them, or mutated to alter existing performant solutions. EC is typically categorised into evolutionary algorithms such as genetic algorithms and genetic programming, swarm intelligence such as particle swarm optimisation, ant colony optimisation and artificial bee colony algorithm, and other techniques such as EMO and memetic algorithms [14, 15]. EC is critical for the project as we will use an EC method to perform neuroevolution to optimise the hyper-parameters.

2.6 Particle Swarm Optimisiation

The steps above detail a traditional EC approach. The EC method this project will employ is the swarm intelligence algorithm, Particle Swarm Optimisation (PSO) proposed by Kennedy and Eberhart in 1995 [16]. Further work by Shi in 1998 [17] enhanced the original algorithm by proposing the addition of a new parameter, inertia weight, which showed to greatly increase the chance of finding the global optimum, and is the version of PSO used in this project. The inspiration behind the method is sourced from swarm intelligence in nature, where a swarm is a collection of many individuals seeking a common goal. PSO operates iteratively by finding the most fit candidate solution of the swarm, and update the swarm accordingly. In PSO, an individual operator is referred to as a particle, an encoded candidate solution. A number of candidate solutions then form a population pool labelled as a swarm. Each candidate solution has a position and velocity in D dimensional search space, as represented by $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ and $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$ respectively. The optimal position of a particle is denoted as their personal optimal position or $PBEST$, with the optimal position of the swarm denoted as the global optimal position or $GBEST$. After a random initialisation of the population, each iteration searches for an more optimal position, and updates the position and velocity of the particles according to the following equations:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}$$

$$v_{id}^{t+1} = w * v_{id}^{t} + c_1 * r_{1i} * (p_{id} - x_{id}^{t}) + c_2 * r_{2i} * (p_{gd} - x_{id}^{t})$$

t denotes the iteration. d denotes the dth dimension. c_1 and c_2 denotes the "cognitive" and "social" constants. They control whether a particle follows its personal best or the swarm's (global) best position. w denotes the inertia component. r_{1i} and r_{2i} denote random values uniformly distributed $(0 \leq r_{1i}, r_{2i} \leq 1)$. p_{gd} and p_{gd} denote personal best position and global best position respectively. These equations update each particle per generation, till an optimal solution is found. This can be applied to hyper-parameter optimisation where each candidate solution is encoded as a set of D hyper-parameters, each iteration of PSO aiming to discover a more optimal configuration of hyper-parameters than the last.

Particle swarm optimisation was selected as the method for optimisation for a few reasons. Firstly, it is a relatively straight-forward optimisation technique, unlike GP and GA methods where more processing is required to establish the nodes for the algorithm, the tree construction and the cross-over and mutation operators. PSO has another advantage over GP and GA in that convergence will be quicker, as the particles are constantly moving in the of the direction of the vector sum of the global best solution, their personal best solution and their current velocity, which allows for more variety in the exploration of the search space [4]. Another justification for the selection of PSO is due to the particles encoding a solution within D dimensional space, which allows for a more precise approach than GP rather than a random generative approach, as the PSO evolutions are more controlled and thus more reliable as no mutation or crossover operators are applied. A final reason is PSO has had little research done with the algorithm for evolving CNN architectures, especially regarding CNNs for text classification. Exploring this research space allows for more impactful research.

2.7 Related Work

There is limited research on evolving CNN for text classification. To the best of the author's knowledge, there are no publications on evolving CNNs for text classification with PSO as the method of evolution. However, there are many publications that deal with evolving deep neural networks using PSO, and were deemed sufficiently similar to justify their inclusion in the background survey. The following section is split into sections covering the main topics pertaining to the project.

2.7.1 Convolutional Neural Networks for Text Classification

One of the first proposed methods of using CNNs for text classification was by Kim in 2014 [2]. CNNs for text classification work by the spatial orientation of the input being preserved. In text, this equates to the structure of sentences where the words appear next to each other and their sequences. Filter maps can be applied to extract patterns in the text to discern the sentiment for example, where negative words will appear more often than positive. Varying the size of the filter maps allows contextual understanding to be developed too, through N amount of words appearing in succession, denoting a happy or sad phrase. A caveat of using text with a CNN is the text must be converted an acceptable format, as CNNs are designed to operate using numerical data. To achieve this, the technique of word embeddings is utilised. Word embeddings are a technique to represent the vocabulary of a document through dense vectors that have to be learned for a particular corpus. The vector space used to represent the words aims to group similar words by learning their distribution. The distribution of a word is learned by capturing similarities and their usage in context to the entirety of the vocabulary; similar words are represented in a similar way to capture their meaning. Thus the position of a word within the vector space is dependent on every other word in the vocabulary. Each of these dense vectors is mapped to a corresponding location in an embedding matrix. Words are initially encoded as a sparse vector that has matrix multiplication applied to it to retrieve the corresponding dense vector from the mapped location. The dimensionality of the output of this layer is known as the embedding dimension. This weight matrix forms what is known as the embedding layer. The embedding layer is theoretically similarly to a fully-connected layer which outputs a dense vector per word from the input document. This dense vector is the input layer for a text-based CNN. Yoon proposes in his research that using pre-trained word embeddings when training a CNN for text classification leads to a strongly performing classifier, as opposed to a random initialisation

[2]. He also concludes that even with a simple structure, performant results are achievable when using pre-trained embeddings.

Conneau et al. [18] uses character-level CNNs for text classification. Character-level means the input of the CNN is not a singular sentence, or a body of text, it is instead all the characters in the text. They are extracted and used as the input sequence. This means their work can operate on many languages as it is the individual character, not the semantic of words that is being interpreted. They manually designed an architecture for their usage, and achieved promising results relative to traditional NLP methods.

Zhang et al. [1] are as far as the author is aware, propose the deepest CNN for text classification at 29 layers deep. Their work represents a translation of principles discovered in CNNs for image-classification that deep and narrow structures can lead to strong classification accuracy [19]. They too use character-level input sequences, as well as having an extremely dense architecture with many convolutional plus pooling layers. Their work considers manually designed architectures at depths of 9, 17 and 29, and for each depth they test 3 different pooling strategies. They achieve strong performance, outperforming all other methods on the datasets tested.

Both of these works use the datasets selected for the evaluation of the project work, thus the work serves also as a comparative measure of performance.

Lee and Dernoncourt [20] propose both an recurrent neural network (RNN) and CNN for text classification, using short-text input sequences. Short-text in this scenario means defined sequences of text but not sentences or whole documents, their primary testing used 3 words. The designs for their architectures involving many convolutional layers extracting features from a series of short-texts, then a max-pooling layer is applied for down-sampling. This leads to a shallow and wide architecture. Their models achieve state-of-art results across the datasets tested.

Rios and Kavuluru [21] perform biomedical text classification using CNNs. They are the few academic researchers to use sentence input sequences. They base their model off of Yoon's work [2] by applying various filter window sizes over the input text to then have a singular pooling layer to perform down-sampling. In addition to the model, they also performed similar tests to Yoon's on a variety of text embedding initialisations. Across the medical classification datasets used, their work achieved promising results relative to prior methods. Their research represents the promise evolving CNNs for text classification can have, if applied to fields like medical research, the practical impact could be quite beneficial.

All aforementioned research on CNNs for text classification all share a distinct commonality in that their architectures are manually designed, as opposed to an evolved approach for architecture design. Their individual implementations differ in terms of what research they cite as justification for the the design of their architectures, but have yet to explore evolving CNNs for text classification. This is a key limitation within the research field that is addressed by the project work.

2.7.2 PSO for Hyper-Parameter Optimisation

Yamasaki et al. [22] used PSO to perform evolution on the parameter settings of an image based CNN. Their contribution includes being one of the first to optimise hyper-parameter settings for CNN with a fixed architecture. They state their form of PSO uses only 2-4% computational load compared to standard PSO, with this being their primary contribution. This works by using Spearman's ranking correlation which is calculated to determine when the accuracy fails to improve a significant amount after a certain number of epochs. Secondly,

they employ a volatility approach by looking at the stability of the network structure. When their measure becomes constant over time, they can compare the performance difference between particles. This is a huge advantage as they noted when the correlation ranking was over 0.8, the volatility stabilised and was used to determine the amount of epochs. This is how they accounted for such computational improvement by having a small number (5-10) of epochs. They optimise the kernel sizes, padding, number of feature maps, and types of pooling where the project solution aims to enhance an increased number of hyper-parameters. One hyper-parameter they omit is stride size, as when the stride size becomes too great the convolutions will not function well as they will not capture key detail. Mine will be different, as when this hyper-parameter becomes too large the system should tend away from the higher ranges and thus not be an optimal position in the swarm. Another feature of their work is they do not cap the randomness of the parameter distribution at the start of PSO. This is a disadvantage that should be avoided. Having uncontrolled parameters and not setting reasonable bounds from known well performing ranges and is a disadvantage to the optimisation of the system.

Ye [23] has the hyper-parameters encoded as a vector of m-dimensional real-number values. The parameters used are learning rate, dropout rate, momentum, and weight decay, in addition to the number of neurons per hidden layer. Whilst real numbers allow for efficient processing, the structure of this scheme is limited by the length of the encoding strategy, it is practical albeit not scalable. A disadvantage of Ye's proposed method is using time-varying social and cognitive constants, and inertia. The time-varying mechanism seeks to reduce the explorative nature of PSO by focusing on the global best solution to avoid local minima. However, this assumes the initial global best solution is the one to be sought out, potentially missing a better solution. One advantage that offsets the aforementioned issue, is ensemble learning. Due to PSO being a population-based optimisation method, Ye harnesses the top particle solutions to form an ensemble model for the task at hand.

Lorenzo et al. [24] discuss using PSO on Deep Neural Networks (DNNs) to perform hyper-parameter optimisation. This practically equates to a CNN for all intents and purposes. Their contributions include the use of two metrics of evaluation to govern whether the PSO algorithm should finish early. Firstly, if the position of the previous GBEST from the new GBEST is less than a minimum threshold, end optimisation. Secondly, if the fitness value of the most optimal particle only increased below a minimum threshold, end optimisation. This is an advantage as it prevents two optimal solutions continually pivoting for the best position. The second criteria ceases the optimisation after it establishes the fitness of the value is stagnating, ceasing to improve. Both of these metrics reduce the computational cost of utilising PSO, similar to the prior paper mentioned. We sought to utilise metrics proposed in this paper for the third objective, and improve upon them by analysing more than just the change in GBEST solution.

2.7.3 PSO for DNN/CNN Architecture Optimisation

Wang et al. [25] performs PSO with a novel variable-length encoding approach to CNN architecture. Wang proposes a structure based on IP addresses, where each component separated by a period of the IP address represents a layer. Each component is an 8 bit value ranging from 0 to 255. This can be represented by one dimension, but this method goes further to fragment large IP component values into a number of smaller values representing a new dimension to achieve convergence quicker through exploring more dimensions, and allow the encoding of more hyper-parameters. This is an improvement to the PSO field, as

this IP address vector representation can encode any deep neural network layer, alongside being extended to an increased byte length. In addition to this, having a disabled layer - a layer that performs no operations - allows for variable-length encoding which has long been a limitation of using PSO. However, the encoding scheme is still limited by a preset maximum length, it can only decrease in depth, not gain it. Wang also proposes a form of accelerant coefficient change, by allowing dimensions to be more explorative or exploitative due to the variable value. Traditionally these are set constants for all particles regardless of the dimension. I hypothesis this method may not be optimal - altering these constants - as analysis must be done to find how to best vary these constants with respect to the dimension values. Wang's code was initially utilised as a way of achieving the first objective. His work was initially aimed at replacing the baseline classifiers for the initial comparison for the project work. The usage of his work is detailed in the next chapter.

Fielding and Zhang [26] propose a multitude of methods to enhance PSO for evolving the architecture of a CNN for image classification. Their method shows excellent results relative to the dataset used. The basis for their architecture is the architecture of VGG-16, a performant CNN architecture, and modifying it slightly [19]. The form of PSO they use is enhanced by adaptive acceleration coefficients, their first contribution. Instead of fixed values the importance of the global and personal components is reversed in later generations, with various cosine and crossover functions used to calculate these new values. This allows different search behaviours to be exploited and used within their methods. Their second contribution is a form of parameter sharing through weight inheritance and a table lookup. The former is achieved by having architectures inherit weights from previously trained particles. The latter is achieved by establishing a table that is populated by trained blocks within the CNN individuals. It is used when a block that has been previously trained reappears in a different particle. If so, the previously trained weights are inherited. Whilst their system is performant, an issue may occur when applying the methods to a different solution space, where the parameter settings for their methods will need to be recalculated to be performant. My solution furthers research like this by exploring new ways PSO can be enhanced to aid training.

Albeahdili, Han and Islam [27] discuss in their work an optimisation to the traditional back-propogation algorithm that uses stochastic gradient descent, by introducing PSO as an initial training scheme for CNN architectures. Their proposed approach works by performing PSO on a population of CNN architecture weights to optimise them. When the error stagnates, retrain the particles otherwise apply GA to the population to stimulate the particles by applying genetic operators to them, effectively regenerating the population. This means the diversity in the search space is always high, and can steer PSO away from local optimums which in conjunction with premature convergence that is known to occur. In addition to this when a generation has finished the GA-PSO aspect of training, SGD is applied to accelerate the training further. This enhanced algorithm is very promising research as it is not always a singular method that proves successful, but a combination of methods to balance out the drawbacks of others. A limitation of this approach however is due to the introduction of GA, even though high diversity is maintained, for such a stringent training process reproducing the population probably means the training time for this algorithm is relatively long. This method is also hampered by not evolving the architecture, moreso the weights for a given architecture which means the method has an artificial cap it can reach.

Junior and Yen [28] too use PSO to optimise CNN architectures. They propose a variety of methods to optimise the use of PSO and the training of the architecture. The first is calcu-

lating the difference between the convolutional and pooling layers to establish how much one particle should change and be influenced by the PBEST and/or GBEST. The second is their velocity calculation, where a sum of differences is taken between the current particle, PBEST and GBEST to establish the change in velocity. In addition to this, they also have a mechanism that allows unlimited depth to be added to the CNN, in the form of if by a random number threshold a layer is selected to be compared with the current particle, they layer can be added or removed, removing the artificial limit imposed by PSO. It is interesting to know they train their particles for a singular epoch as the evaluation criteria when performing PSO. Their approach takes the difference between layers to calculate the velocity. However, they do not consider the ratio of the difference in layers, they merely determine the difference between layers as a numeric sum, and if the layers between particles do not match, a value is resolved to be in-between both which can lead to inaccuracies. In addition to this, the velocities of layers are not even calculated for particles with no difference in layers, as well as fully-connected layers remain untouched for the initial calculation. My ratio-based velocity strategy instead deals with the ratio between the hyper-parameters of the layers, as well as considering the fully connected layers as layers that can be evolved, unlike this limited approach.

2.7.4 PSO for Feature Selection

Tran et al. [29] proposes a variable-length PSO encoding scheme for Feature Selection. The paper introduces VLPSO a variable-length encoding scheme that operates by a length changing mechanism, where if GBEST has not changed after a predefined number of iterations, the current swarm is updated and the lengths of the particles are varied. The length changing mechanism works in tandem with a new initialisation method called population division. This method sections particles into divisions with varying lengths to facilitate a diverse population. When the stagnant GBEST iterations value is reached, the length of the particle with the best performing division becomes the longest length of all particles, and the rest of the particles are resized. This method is less applicable to the current project as a division based approach to CNN architecture is not feasible. A benefit of this method reduces a premature convergence as the particles can have their length altered to avoid local optima, to find potentially better regions in the search space. By reducing the length of particles, computational cost is reduced also. Similar to research by Wang et al. [25] the encoding scheme in this paper is important work within the field of PSO, as developing a variable-length scheme solves a key limitation of PSO.

2.7.5 EC for DNN/CNN architecture optimisation

Knippenberg et al. [30] propose a neuro-evolution based approach to evolving CNNs. The initial conception of their idea was sourced from *Large-scale evolution of image classifiers*, which acts as a baseline of using genetic programming to evolve a CNN. They saw issues with how the selection process within the population was conducted, it incurred a large computational cost, so have sought to improve it. Their method involves evolving a set of convolutional autoencoders, to find the best one to train a population of CNNs. The pairing of the most effective CNN and autoencoder creates the most optimal overall network. They designed their own set of mutations for the encoders to aid the initial evolution, as well as used a Pareto front approach to determine the most performant autoencoder. Autoencoders were used in an effort to reduce the input sample sizes. Their research showed the approach could reduce training times by a factor of days, with in some instances twenty percent more networks being generated throughout the process [30]. This approach is how-

ever flawed by the inherent fact you are training a set of autoencoders before you even train your networks, meaning the training time reduction gained may not be offset by this initial evolutionary process. This EC approach is still significant as any methodology that reduces training time is valuable.

Zhu et al. [31] developed a novel solution based on an EC method similar to PSO, Artificial Bee Colony (ABC). They perform hyper-parameter optimisation by using ABC in a three phase approach. Firstly, employed bees use a local memory factor, similar to the "cognitive constant" in PSO, to search a new area of the search space. Secondly, onlooker bees receive the updated search information from the previous step. Thirdly, scout bees find a new area to search if their local memory of a previous position has been dropped. They too use only five epochs to train each CNN in the evolutionary procedure, and a reduced training set count. This approach also extends to evolve batch size, learning rate and rule. Their overall encoding is fascinating as it deals with layers in a "block" approach, encoding the layers in a vector within the larger encoding. The rule they developed is if a pooling layer is to exist, it always follows a convolutional layer. The approach is further simplified by limiting the pooling type to be only max, and a fixed stride size. I evolve the stride size, but fix the pooling type based on prior research proving max is the best pooling operator for text classification [32]. However, the inherent complexity of their encoding scheme lead their research to not be complete due to the computational cost.

Yanan et al. [33] proposes an EC method using GA to evolve performant CNN architectures for image classification. The primary contributions are a new encoding scheme that extends the depths of CNNs by introducing skip connections as a layer that can be evolved, as well as running population evaluation across GPUs asynchronously. Firstly, the mutation operator as part of the CNN-GA strategy can add a skip or pooling layer, remove a layer or randomise the hyper-parameter values of a block. Secondly, the crossover operator allows for variable length individuals, meaning there is a higher ceiling on the potential evolutions per individual that can occur. As part of the algorithm an environmental selection is employed, to filter the top individuals for the next generation. One benefit of the algorithm is it requires no pre or post-processing, a truly manual system for the task at hand. One aspect that is highlighted is the reduced probability of a pooling layer being evolved. Down-sampling affects the total information the system has to process, thus limiting the addition of this layer to the population may increase the performance. In contrast, there is a higher probability of mutating a skip layer, as this deepens the CNN to aid in the performance. Relative to state-of-the-art algorithms on selected data-sets, the proposed algorithm outperforms almost all competitors.

2.8 Summary of limitations

1. There is a lack of academic research on PSO for automatically evolving CNNs, especially for text classification.
2. Most methods change a small number of hyper-parameters, thus not evolving the CNN to a great degree.
3. A limitation exists with the use of the "social" and "cognitive" constants to control training time as they reduce the explorative or exploitative nature of PSO.
4. There are no optimal formulas to calculate the difference between two encoded CNN particles using PSO, which is critical to determining how close a particle solution is to

the GBEST or PBEST position.

5. The encoding scheme for CNN architectures using PSO is critical to evolving a performant architecture. However, there are no optimal encoding schemes proposed in academia yet.

6. All classifiers for text classification are manually designed, and do not use any evolutionary computing techniques.

Chapter 3

Design

3.1 Choosing Tools and Technologies

Keras was chosen as the neural-network library for the project. Keras is comprised of many deep-learning modules written in Python for academic usage. Keras has enabled the implementation of the first two objectives seemlessly due to the ease of use of the library. Tensorflow or PyTorch are libraries that have the same functionality required for this project. These two options are promising for the future development of the project as their modules extend beyond Keras', allowing for in-depth deep-learning development through lower level modules. However, Keras has a lower learning curve and is suitable for the initial objectives. Likewise, PyTorch and Tensorflow require a more delicate setup when it comes to package and OS versioning, where Keras did not.

PySwarms is the library used for the PSO implementation for the first objective of the project. It is a robust, frequently updated PSO library that has allowed for the initial solution to be developed [34]. PySwarms allows for various usages and implementations of the PSO method. This library was chosen over two other options, a personal implementation and pyswarm. A personal implementation in the initial stages of the project poses challenges to the development as an understanding needed to be developed in how to implement PSO, detracting from the other objectives which is not ideal. However, a personal implementation was developed for the latter objectives. PySwarms is easier to use than Pyswarm, in addition to having a greater breadth of documentation and is regularly updated. For future development, we will require an extension of this library through implementation of our own methods, or, an entirely independent implementation of PSO.

3.2 Text Preprocessing

Text classification has an inherent issue with the data used to train and classify. Much like a blurry image, text can contain a variety of uninformative characters and unexpected formatting which increases the difficulty of accurately classifying a document. We have opted to preprocess our data for the following reasons. Firstly, it cleanses the data of content such as punctuation, numbers and extra white space as these aspects of the text have little predictive power in addition to being more information the classifier has to process. Secondly, as a consequence of cleaning the data the vectors generated for the text representations are shorter and more communicative of the overall sentiment as non-predictive elements have been removed. Various publications do not preprocess data and perform classification on the data as it is [18]. However, the project aims are to develop methods of evolving the hyperparameters and architecture of a CNN for text classification, thus it is in our best interests

to feed the CNN important information. An additional final step limits the total corpus to the most frequent words. Another approach would be to replace the lower frequency words with part-of-speech tagging (POS tagging); converting them to their base particular part of speech; nouns, verbs, adjectives, etc. This would reduce the overall dimensionality of the input by changing the representation to a simpler value. However, the word embeddings mentioned below solve this issue of losing information of removing words by incorporating sentiment information which is more indicative of the context and content of the text than simply POS tagging.

3.3 Word Embeddings

The project also requires a form of embeddings for Text Classification. Pre-trained embeddings were shown to increase classification performance [2]. As such, the embeddings utilised for the project are sourced from GloVe, developed by Pennington et al [35]. GloVe embeddings are developed by global and local statistics of the co-occurence words in the contexts it appears in the document, to derive the semantic relationship. If words co-occur they are deemed to be semantically linked. Another option for embeddings which is widely used in literature is word2vec. However, Pennington et al [35] concluded that their model of word embeddings, GloVe, consistently outperformed classification tasks that used word2vec.

3.4 Baseline Classifiers

The premise for the first objective was to generate a set of CNNs for text classification to be used for analysis to inform the design of the PSO solution. The initial design for this section was not my own. As previously stated in the background, we utilised Wang's work for a baseline classification result [25]. This is because he had created a promising PSO implementation for classification that would have been an extensive baseline to evaluate the project work against. However, his work is based upon image classification. This posed a significant problem as a conversion was required to train on, and classify text. As elaborated upon in the implementation, this approach was attempted but it was deemed baseline classifiers that did not evolve were better. Classifiers without evolution were a better option as developing eight different architectures as classifiers enabled an investigation into what exactly the system should aim to evolve in terms of hyper-parameters. Analysing the results of the different architectures developed an understanding of what is required for the final solution.

The design for the implementation of the baseline classifiers are simple three till ten layered architectures. Each architecture has at least an embedding layer, a convolutional layer, a pooling layer and a fully connected output layer. The number of layers was counted as the layers between the starting embedding and the final fully connected output layer, which had the required amount of output nodes per number of classes of dataset. The layers also used were a max-pooling strategy, and global max-pooling strategy. These were chosen based off prior research [32], and based off initial testing showing the training per architecture suffered from a degree of over-fitting thus dropout was included.

A number of hyper-parameter settings were also used per each layer as permitted. The value ranges for the hyper-parameters were based on a publication by Zhang [32] and randomly deviated from here. This was a critical design decision as an exhaustive approach to the hyper-parameter setups would be time inefficient. Establishing randomised values means more meaning can be derived as to what aspects govern the classification of a text, due to there being potentially multiple optimal solutions. As PSO operates to find an opti-

mal solution within the search space, having randomised values aids this search by covering a larger spread of the potential values which allows for more discovery per evolution. Furthermore, this best mimics the output of the PSO system which will look unlike traditional architecture, as the optimal values may not follow a pattern or correlate in any regard.

3.5 Design of PSO-based Classifiers for Text Classification

Particle Swarm Optimisation is the choice of EC method to optimise CNN architectures for text classification. The majority of the project work was invested into developing and analysing the classifiers, developing new encoding schemes, and the PSO implementation. PySwarms is the aforementioned library used for the PSO solution for the following hyper-parameter encoding. A personal implementation of PSO is used for the following objective relating to optimising the architecture. A standard algorithm for PSO is being utilised for the project. This involves using a GBEST topology for both implementations. In addition to this, both implementations also share the same form of fitness function, the error value from training each particle as an individual architecture. The first encoding design for PSO regards a scheme used to perform hyper-parameter optimisation, with an enforced architecture. The second encoding design for PSO regards a scheme used to perform structure optimisation, modifying and comparing various architectures to evolve a well-performing system. Both encoding schemes cover the second and third objectives of the project.

3.6 Hyper-parameter Encoding

The second objective for the project involved the development of an encoding scheme for optimising the hyper-parameters of a CNN. The design of the hyper-parameter encoding is limited to six layers with a hyper-parameter representation encompassing the number of filters, filter sizes and pooling window size listed below.

```
[nf_1, sf_1, sp_1, nf_2, sf_2, sp_2, nf_3, sf_3, sp_3]
```

nf_X denotes the amount of filters in layer x. sf_X denotes the size of filter window. sp_X denotes the size of pooling window. This encoding scheme is used for a CNN architecture with a convolutional layer and pooling layer pair repeated three times to form the six layer architecture.

The design for this architecture uses six layers as this depth was the most performant of the baseline classifiers when evaluated on the IMDB dataset. The classifier with six layers was also evaluated to have one of the highest classification accuracies from the Yelp dataset. PSO is incorporated into this representation by having a nine-dimensional particle structure, using the ranges listed above as the bounds for the dimension ranges. The particle class consists of nine hyper-parameter values that are encoded for a six-layered CNN structure. Whilst academic work has shown that deep and narrow architectures are performant for text classification [18], the classification accuracy using the baseline classifiers on both datasets indicated that promising classification performance can be achieved using a shallow structure. This is another reason behind evolving a six layered architecture.

The design for this encoding scheme stems from the simplicity and easy translation it offers when decoding an individual particle. Whilst other implementations use a bit-string implementations [25], [29], for this objective it was deemed excessive to design such a scheme.

As we are evolving a CNN, a set of convolutional layers and pooling layers were an appropriate fixed structure to evolve. The reasoning for this design was the performance of this straightforward architecture could govern the performance utilising key CNN components, and highlight issues where additional layers such as a regularisation layer may be needed. An enhancement to this design could have involved evolving a global pooling layer in contrast to a max pooling layer within the encoding scheme. The reason behind this is for text classification Zhang et al. concluded that 1-max pooling is more effective than max pooling in general scenarios [32]. However, max pooling was favoured due to the increased hyper-parameters associated with the layer that can be evolved. Opting for max-pooling allows for a more performant CNN to be evolved through greater dimensionality.

The ranges of these hyper-parameters are shown in the Table 3.1 below. Unlike Yamasaki et al. [22], whose research did not include a restriction to the layer hyper-parameters, enforcing a restriction is a benefit to the solution to limit the number of unsuccessful architectures evolved. Ranges used are sourced from literature recommending suitable values for the layers [32, 2, 1, 25].

Hyper-Parameter	Lower bound	Upper bound
Num. of filters	32	1024
Size of filter	2	5
Size of pooling	2	4

Table 3.1: Hyper-parameter ranges for PSO evolution

3.7 Architecture Encoding

The third objective for the project involved the development of an encoding scheme for a CNN architecture. This is a continuation of the previous work done for the hyper-parameter optimisation where the overall optimisation has been extended to include the architecture as a solution representation. A key difference is now the layers in conjunction with their hyper-parameters are being evolved. Another key difference from the previous encoding scheme is in addition to evolving the individual layers, a new implementation of PSO is used. PySwarms, the framework used to run the PSO algorithm for the second objective has been replaced with a personal implementation. PySwarms could have been extended to be used for this work on the third objective too, but the investment of effort put into achieving this was deemed better spent on developing the encoding scheme and subsequent methods associated it.

In addition to this new encoding scheme, two methods were devised to enhance the optimisation of PSO, detailed in sections 3.8 and 3.9.

3.7.1 Encoding Scheme

The basis of the encoding scheme is to encode a layer and the corresponding hyper-parameters for that layer in the most effective way possible, a way that will allow for the best performance from PSO. The design developed for this objective is an eighteen dimensional continuous encoding. Each three integers of the encoding represented a new layer or block, allowing for up to two hyper-parameters per layer. There are six layers per individual. Below is a representation of the encoding scheme where X is equal to six, for a six layered CNN.

A key design decision for this encoding scheme was to enable a variable-length encoding. This means providing the ability for an architecture to evolve to be shorter or longer depending on the problem at hand. As such, the scheme introduces the idea of a disabled layer. A disabled layer is not a layer used within the final CNN architecture, instead it is a representation of a layer slot that is not used, thus disabled. These can be evolved to included or excluded from the particle dependent on the solution space favouring a deeper or shallower network.

```
[lt_1, hp1_1, hp2_1 | lt_2, hp1_2, hp2_2 |,| lt_X, hp1_X, hp2_X]
```

lt_X denotes the type of network layer used x. $hp1\_X$ denotes the first hyper-parameter for the layer $hp2\_X$ denotes the second hyper-parameter for the layer. The encoding scheme is initialised to have a maximum of six layers as based on the results from the baseline classifiers, and results from the hyper-parameter encoding. The primary reasoning is due to the initial PSO and baseline results being performant relative to benchmarks in various papers [18], [36], thus I retained the six layered architecture.

Encoding schemes for PSO in academic research are typically constructed from bit-strings [29, 25]. The encoding scheme for this project uses continuous values, for the reasons stated below. Using continuous values allows for a direct translation between the value decoded from the particle, to the hyper-parameter setting that is used in the CNN. The secondary reasoning is this allows an easier conversion to a ratio, or probability, for how likely a layer should be to occur within the automatic CNN evolution by having clearly defined ranges that require no translation. If I had implemented a bit-string encoding I would require a system to decode the particle, reducing the time to work on methods to enhance the overall system.

The ranges of these layer-types are shown in Table 3.2 below. These values correspond to the lt_X values described in the encoding scheme above. The ranges for these values were arrived at by initial testing the types of evolved architectures that the algorithm computed being performant for classification.

Layer	Lower bound	Upper bound
Convolutional	1	<= 10
Pooling	> 10	<= 20
Fully Connected	> 20	<= 30
Disabled	> 30	<= 35

Table 3.2: Layer type ranges for architecture encoding

There is an equal ratio (a range of 10) for a layer to be a convolutional, pooling or fully-connected layer. This is in line with CNN development where a convolutional layer is followed by a pooling layer, with the potential for a fully-connected layer to be included also [33, 25, 18]. There is a point highlighted in [33] that discusses the reduced probability of pooling layers to increase the amount of information the system has, however such work as [32] describes the importance of down-sampling for text classification thus the ratio of a pooling layer existing in the architecture was kept the same as the convolutional and fully-connected layers. In addition to this, the values were also inspired by literature [25]. There is a decrease of 50% to the ratio that a layer can evolve to be disabled, this is due to the particle encoding already being shallow, thus the ratio is reduced to allow for a disabled layer to be evolved, but not at the same likelihood as other layers. All convolutional layers for this encoding scheme use ReLu activation function. ReLu is a well performing activation function, as it converges quickly, avoids weight related issues, and outperforms other activation

functions [37].

In addition to the previous point about the ratios of each layer type, every particle encoding has a convolutional layer initialised as the first layer, preceding the embedding layer. This is to reduce the number of invalid architectures within the initial population pool. If this step was not implemented, some particles may fail to be trained and thus would need to be pruned from the population pool.

```
[3.4, hp1_1, hp2_1 | 17.8, hp1_2, hp2_2 | 31, hp1_X, hp2_X ... ]
```

The above encoding is an example particle that represents a convolutional layer, pooling and a disabled layer for the first three layers of the architecture.

The ranges of these hyper-parameters are shown in Table 3.3 below. These values correspond to $hp1\_X$ and $hp2\_X$ values described in the encoding scheme above. Ranges are again sourced from literature recommending suitable values for the layers [32, 2, 1, 25]. However, I have reduced the ranges of filter amounts for the convolutional layer. This was done to reduce the time taken to evolve the architectures as there was a noticeable decrease in time taken to train each particle after this change. To further support this change, there was no noticeable decrease in performance from this change too. A side effect of this encoding strategy is for the pooling and fully connected layers, only one hyper-parameter value is used. This is shown below in the table where the pooling and fully connected rows only list one hyper-parameter, but the convolutional row lists two. In terms of the encoding, this means after the first value that establishes the layer type, only one out of the preceding two values may be used as hyper-parameter values if the layer type is not convolutional. The reasoning behind this is because it allows the encoding scheme to be extensible if required, as well as the fully connected layer can only use one value for the amount of neurons within the layer. This too applies to the pooling layer, as using the second hyper-parameter as the stride size may lose critical feature information thus is not used in the encoding.

Layer Type	Hyper-Parameter	Lower bound	Upper bound
Convolutional	Num. of filters	1	256
	Stride Size	1	5
Pooling	Size of window	1	5
Fully Connected	Num. of units	1	768

Table 3.3: Hyper-parameter ranges for architecture encoding

In addition to this referenced literature [32, 2, 1, 25], there is a key change of enabling all the lower bounds to start at one. For a convolutional layer this means there may be only one filter. For a pooling layer this means there is may be no reduction in the dimensionality after a convolutional layer. These changes were made to allow for a more seamless evolution between the layers, allowing greater transition from a pooling layer to a convolutional layer and vice versa, as this means there is overlap between each layer's hyper-parameter ranges.

```
[3.4, 173, 2 | 17.8, 4, 4  | 31, 2, 2  | ... ]
```

Shown above is an extended view of the prior example that shows the corresponding layers and the hyper-parameters. The encoding translates to the first layer being a convolutional layer, with one hundred and seventy three filter maps, using a stride size of two. The second layer is a pooling layer, with down-sampling window size of 4. The additional '4' value is not used as the pooling layer only uses one hyper-parameter. The third layer is a disabled layer, and thus neither of the hyper-parameter values are used.

3.8 GBEST Inheritance Strategy

In conjunction with the new encoding scheme, the development of additional methods to enhance the performance of the architecture encoding scheme were developed. These methods are aimed at aiding the evolution process of the population per generation. Zhang and Fielding [26] proposed a method that allows weight-sharing between the particles to aid training. The project work is a development from this method by inheriting values from other particles.

The first method developed focuses on using elitism to enhance the solution developed from the PSO algorithm. The premise is when one block of the encoding scheme at a particular position moves from one layer type to another, if within the GBEST particle there is the same layer type at the same position, inherit the hyper-parameter values of that GBEST block. This is where elitism is used, as the GBEST hyper-parameters values replace the previous values. The development of this method stemmed from analysis of the evolutionary process showing the blocks would tend towards the upper and lower bounds of the range for that layer type, but stagnate and not continue to evolve when the layer type changed. A constraint upon this design is how this limits the diversity of the population as the solution space around the GBEST may not be explored due to the forced change. However, the stochastic nature of PSO should enable a local search to take place. An additional design feature could have been to add a random factor to this strategy being employed, if there was a chance it did or did not occur per evolution. This has the benefit of allowing some solutions to perform a global search. It was discovered however that this did little to stem the issue previously highlighted thus this was not implemented.

Taking the below particle as the current GBEST position found,

```
[3.4, 173, 2 | 17.8, 4, 1 | 31, 2, 2 ... ]
```

When the particle below has a change in the first block from a pooling to a convolutional layer, as that exact position of the GBEST position is also a convolutional layer, it inherits the hyper-parameters.

```
[11, 2, 3 ... ] -> [9, hp1_N, hp2_N ... ] = [9, 173, 2 ... ]
```

3.9 Ratio-based Velocity Strategy

A further development of the prior GBEST Inheritance strategy is a Ratio-based Evolution Strategy. This is a change in the prior strategy by adapting it to develop a method more inline with PSO principles of exploring and exploiting the solution space of a problem, where the GBEST Inheritance strategy was aggressive and cut down the available solutions that could be explored. Further to this, the prior strategy shows a reduction in the classification performance, especially on the Yelp dataset from the baseline classifiers. The premise of this method is to develop a method that can calculate the difference between two encoded CNN particles using PSO, as this is a current limitation of PSO. If you are comparing the GBEST particle's position and another particle within the population, how is a convolutional layer as the third layer of the GBEST particle directly comparable to a pooling layer of another particle. This difference does not factor in the effect these layers have on the overall CNN, and thus a different approach must be developed, in contrast to simply performing a simple dimensional position comparison.

The idea of the method is to alter the velocity calculation, specifically the social component and cognitive component to enable a smoother update to the position of each particle.

As the formula for velocity contains a social and cognitive component, and these are influenced by the PBEST and GBEST positions, if the layer types at each position do not match up, there will be a significant difference when comparing a particle's position to the best position that skews the velocity formula.

For example, the social component of the PSO velocity formula involves finding the difference between the GBEST position and the particles' position; if at a particular position the layer types differ, the component will try to calculate the difference between the down-sampling size and the amount of filters in a convolutional layer. A ratio-based approach is thus required to allow for smoother evolutions, allowing greater exploration of the search space. This method like the GBEST Inheritance Strategy only applies when one block of the encoding scheme at a particular position differs from the layer type at the same position as a GBEST block. When this occurs, as the velocity for each dimension of a particle is being updated, each layer type of the blocks within the particle are checked if they differ from the PBEST and GBEST solutions.

If they do, firstly take each hyper-parameter value of the layer in the best solution, and divide them by their max range, to calculate the ratio of the value relative to the range. This ratio is then multiplied by the range of the hyper-parameters of the layer currently being evolved. The output is thus a value that can be used in the social and cognitive component formulas as a translation from the difference in layers.

Taking the below particle as the current GBEST position found,

```
GBEST: [3.4, 173, 2 ... ]
```

Previously when the velocity was calculated the social component would perform the calculation as shown below,

```
Current Particle: [13, 2, 3 ... ] -> 173 - 2 = -171
```

However, after the ratio-based strategy is employed, the resultant value is much more suited for the ranges of that block being updated.

```
[13, 2, 3 ... ] -> 173/256 * 5 - 2 = 1
```

Due to opting for a real number encoding scheme, this is another reason why this strategy is suited as this reduces the difference in encoding different layers by having to evolve particles with differing hyper-parameter ranges. This strategy circumvents that issue.

Chapter 4

Implementation

The following section explains the implementation of my solution. Firstly, the input data is preprocessed and the embedding layer is generated. Then a method of classification is utilised; either a baseline classifier or the PSO based solution. These methods are then trained on this input data. Once training is completed, the classifier is then applied to a set of testing data to establish a classification accuracy, thus an indication of the fitness of the classifier.

4.1 Text Preprocessing

The text preprocessing performed includes stripping of extra whitespace and punctuation from the documents. This leaves the documents in an easily tokenisable form with less noise. In addition to this, only the top fifteen thousand most frequent words were selected from each corpus to only include the most relevant words per classification task. Otherwise, meaningless words that do not help learn the overall pattern would need to be learned by the CNN classifier. This is the initial step of my solution.

4.2 Word Embeddings

The text vectors are generated from the cleaned data. The vectorised documents to be used as input for the CNN have a max number of 400 words enforced. This has been done to limit the sparsity of vectors which perform poorly when used as input for a CNN. A sparse vector is when the majority of the vector has a value of 0. The average length of the documents is 176 thus 400 was chosen by convention based on similar text classification systems [2]. As previously mentioned GloVe embeddings are used as the embedding matrix for the CNN classifiers. GloVe provides a selection of embedding vector lengths to employ; I selected their variation with an embedding dimensionality of 50. This is due to the publication concluding there were diminishing returns on vectors of greater length [35]. Furthermore, these word embeddings are trained on the corpus whilst the model is training, to develop further contextual awareness between the words used in the texts. This is the second pre-classification step of my implementation.

4.3 Baseline Classifiers

The implementation of the baseline classifier architectures uses convolutional layers, a pooling layer with a max pooling strategy, a flatten layer to account for the dimensionality re-

duction required and a dense layer. Eight architectures using a variety of the aforementioned layers with various settings formed the construction of the classifiers. Keras was integral for the implementation of these aforementioned classifiers. All of the structures developed had their inner structures based upon options present in the 'Layers' module provided through Keras. This implementation was chosen as it made the development of the classifiers straightforward and interpretable. As these classifiers would go on to inform the design of the PSO representation, they needed to be clear.

4.4 Hyper-paramter Optimisation PSO Implementation

The second objective implementation of PSO is relatively straightforward.

Algorithm 1 Integration of PSO

1: Process input documents by vectorising words and generating embeddings
2: Initialise a population of particles with randomly encoded values for a six-layered architecture
3: **while** Max generations not reached **do**
4: **for** Each particle i **do**
5: Decode particle to CNN structure and train classifier
6: Evaluate the fitness of trained structure from the particle $f(X_i)$
7: **if** $f(X_i) < f(pbest)$ **then**
8: $pbest \leftarrow X_i$
9: **end if**
10: **if** $f(X_i) < f(gbest)$ **then**
11: $gbest \leftarrow X_i$
12: **end if**
13: Update velocity, then update position of particle using PSO equations
14: **end for**
15: **end while**
16: Construct CNN using $gbest$ and train classifier
17: Evaluated trained CNN

Algorithm 1 shows that firstly, the method initialises a pool of particle solutions representing hyper-parameters for a six layer CNN architecture. Secondly, these solutions form CNN architectures that are trained on a portion of the training dataset per each generation. The output of that step is a loss value on the remaining portion of the training set. This is a fitness measure of a particle on a small subset of unseen data: how performant the position found is depends on the loss value returned. The lower the loss, the more performant that particle solution is. A separate portion of the training data is used to reduced overfitting. This form of evaluation is used in all PSO algorithms for the project work. Each particle solution then has their positions and velocities updated, using the equations listed in 2.6. This above process continues iteratively until the max generation count is reached. When this termination criteria is satisfied, use the optimal position found and build an architecture from it.

An example particle solution representing the most optimal position found is:

```
[920, 3, 3, 965, 4, 3, 861, 3, 3]
```

As discussed in the design section earlier, this comprises the hyper-parameters selected to

specify the 6 layers - the filter amounts, windows sizes and pooling window sizes per their respective layers.

```
Conv. Layer 1: Number of filters = 920, Size of filter window = 3
Max Pooling Layer 1: Size of pooling window = 3
Conv.  Layer 2: Number of filters = 965, Size of filter window = 4
Max Pooling Layer 2: Size of pooling window = 3
Conv. Layer 3: Number of filters = 861, Size of filter window = 3
Max Pooling Layer 3: Size of pooling window = 3
```

These values are then used to construct an architecture in the below form,

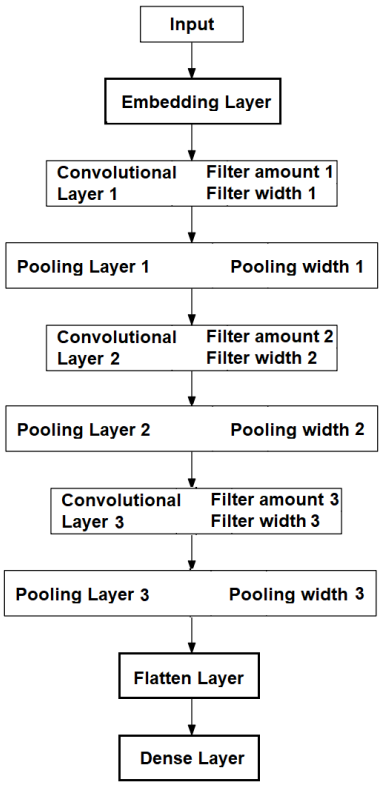

Figure 4.1: Structure of evolved CNN using hyper-parameter encoding

With this current implementation we have validated the design of the hyper-parameter representation. However, an issue of diversity exists in the current representation. As a consequence of evolving only a few key hyper-parameters, the performance of the architecture is limited. The rationale behind this decision was based on the analysis of the baseline classifiers as these all performed similarly with a major difference being the value of hyper-parameters. These previously mentioned issues can be remedied by a more complex encoding scheme, which is the next section of the report - an encoding scheme for the architecture as well as the hyper-parameters. This will allow a more diverse CNN architecture to evolve, where the current solution will only ever generate the same architecture with variations of the hyper-parameters. The baseline classifiers showed that varying depths, in addition to varying layer types will be the most beneficial to achieving optimal performance. Thus, this knowledge informs the next section of research.

4.5 Architecture Encoding

The third objective for the project involves the development of an variable-length encoding scheme for a CNN architecture. This has been achieved through a continuous value encoding scheme comprised of blocks that correspond to a layer and the respective hyper-parameters. The architectures that are evolved include convolutional layers, max-pooling layers, fully-connected layers. A flatten layer is added at the end of every evolved structure to account for the dimensionality reduction required before the final fully-connected layer. As stated in section 3.7.1, all architectures are initialised with a convolutional layer in the first position also, to reduce the amount of invalid architectures within the population. This implementation was also done using Keras, but with a personal implementation of PSO. The general PSO algorithm explained in 4.4 is the same PSO algorithm used in the following section. However, an architecture encoding is used in lieu of a hyper-parameter encoding. The methods described in 4.5.3 and 4.5.4 are implemented in line 13 of Algorithm 1 in 4.4.

4.5.1 Architecture Encoding Scheme

An example encoding scheme is shown below, which when decoded corresponds to an architecture of a CNN with a depth of four. The ellipsis at the end is omitted to save space, for representation purposes all layers following are disabled.

```
[1.3, 230, 3 | 14, 2.3, 5 | 27, 400, 3 | 5, 67, 3 ... ]
```

The structure shown below is one of the many possible for this encoding scheme. In addition to this structure, each layer also has a set of hyper-parameters as per the encoding. The first convolutional layer has 230 filters with a stride size of 3, the max-pooling layer has a window size of 2, the dense layer has 400 units and the final convolutional layer has 67 filters with a stride size of 3.

This particle above can be show as the following architecture below:

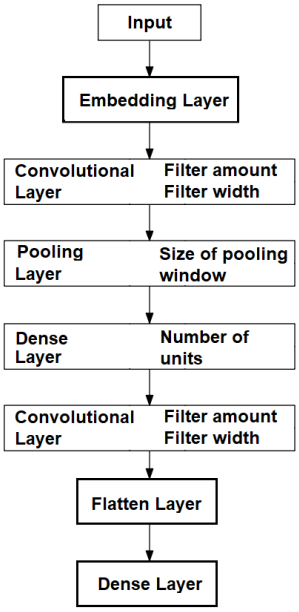

Figure 4.2: Structure of evolved CNN using architecture encoding

This CNN architecture is what the particle above would translate to when decoded. For example the third dense layer would have 400 units, and the pooling layer before it would

have a pooling window size of 2. This also shows how only convolutional layers use both hyper-parameters in the encoding scheme. This architecture encoding scheme addresses a major limitation of PSO, by being of variable-length. Much like how the second layer above is a pooling layer, this can also be a disabled layer, and thus reduce the depth of the overall architecture.

4.5.2 PSO Optimisation Methods

With a personal implementation of PSO in conjunction with a new encoding scheme, various mechanisms are established. The first method relates to the generation of the particles. Due to the encoding scheme being continuous, specific boundaries must be set when generating the population. For example, an architecture would not work with a pooling layer with a pooling window size of 178, if the value was taken from the range of the convolutional layer range from 1 to 256. Therefore, the method generates a random value to represent the layer type of each block. Layer type being what layer that block is. The hyper-parameters specific to the layer type are then generated, to be within the preset range values. This block generation continues until the whole particle is encoded. In addition to this, the first layer of each particle is always a convolutional layer. This is manually added before other layers are generated. This reduces the amount of invalid particles within the population.

The second additional method is a pooling check. This was developed to identify and cleanse particle solutions that are unfit. They are deemed unfit if they are initialised with too many pooling layers or evolved to have too many. The issue with allowing these particles in the population is two-fold. Firstly, an increased number of pooling layers vastly reduces the information in the system, and secondly this behaviour can reduce the dimensionality of the output from the convolutional layers enough to introduce errors into the system. This functionality allows the termination of particles within the population that would be unfit for the task before they have had a chance to be evaluated, thus eliminating influence on the global population. This method works by iterating over each particle and if there are 3 or more pooling layers, replace the first and final with a convolutional layer. This occurs when the population is initialised and seldom when a particle evolves to have an illegal number of pooling layers.

Below is the full Population Initialisation algorithm.

Algorithm 2 Population Initialisation

```
Initialise total dimensions and boundaries for these dimensions
Initialise empty population array
while Max population size not reached do
    initialise empty particle P
    append Convolutional Layer as first layer
    for Each remaining dimension P do
        Generate value, lt, for layer type of P[n]
        Based on lt, retrieve the min and max bounds for the layer type
        for Each hyper-parameter hp do
            P[n + hp] ← Generate random value between min and max
        end for
    end for
    Perform pooling check to reduce pooling layers
end while
Perform PSO algorithm
```

The third method regards how the particle is decoded. As the particle blocks are encoded with the layer-type first, this is first translated from the value to the range boundary it occupies. Then dependent on the layer-type, a layer is then added with the following two values within the block used as the hyper-parameters for said layer. In the case of a disabled-block, no layer is added. For example, if the layer type was within range to be a convolutional layer, the following two values are the amount of filters and the stride size.

4.5.3 GBEST Inheritance Strategy Implementation

To aid the evolution of the particles finding an optimal position, the GBEST Inheritance Strategy was developed.

Algorithm 3 GBEST Inheritance Velocity Strategy

```
1: Perform PSO algorithm to update velocities
2: while Max generations not reached do
3:     for Each particle P do
4:         Update velocity using PBEST and GBEST values
5:         for Each particle layer block position d do
6:             Update layer type position
7:             if P[d] and GBEST[d] are now within same layer range then
8:                 Save GBEST[d] hyper-parameter values
9:                 P[d + 1] ← GBEST[d + 1]
10:                P[d + 2] ← GBEST[d + 2]
11:            end if
12:        end for
13:    end for
14: end while
15: Continue PSO algorithm
```

The below algorithm is applied when the layer-type of a block is the same as the GBEST layer-type at that position. An example being if the GBEST layer-type at that position is a convolutional layer, save the amount of filters and the stride size. These saved values are then applied to the particle that is being updated, so that block now matches the GBEST particle.

This method accelerates the search for the optimum position within the search space by allowing more particles to "skip" to the regions of the search space that have been proven to be promising.

4.5.4 Ratio-based Velocity Strategy Implementation

This algorithm builds upon the ideas of the previous method to aid the evolution of PSO, but in a fashion better suited for the PSO algorithm. It is better suited as this strategy still maintains the diversity of the population by not immediately adopting the GBEST hyper-parameters.

As discussed before, the use of continuous values is suited towards a ratio based scheme as the values within the particle can easily be computed as a ratio representing how close a hyper-parameter is to the upper range boundary. All that is required is knowledge of each layer range, as then the hyper-parameter range values can be used to establish a ratio between the PBEST and GBEST layers.

Algorithm 4 Ratio-based Velocity Strategy

Perform PSO algorithm to update velocities
while Max generations not reached **do**
  for Each particle i **do**
    Update velocity using PBEST and GBEST values
    for Each particle layer block position d **do**
      Calculate social component, *socialComponent*
      if $i[d]$ and $GBEST[d]$ not within same layer range **then**
        for Each hyper-parameter hp, in i **do**
          $ratio \leftarrow GBEST[d+hp]$ / Upper bound of $GBEST[d+hp]$ layer range
          $convertedValue \leftarrow ratio \times$ Upper bound of $i[d+hp]$ layer range
          $socialComponent \leftarrow c_1 \times r1 \times (convertedValue - i[d+hp])$
          Return *socialComponent*
        end for
      else
        $socialComponent \leftarrow c_1 \times r1 \times (GBEST[d+hp] - i[d+hp])$
        Return *socialComponent*
      end if
      Calculate cognitive component, *cognitiveComponent*
      if $i[d]$ and $PBEST[d]$ not within same layer range **then**
        for Each hyper-parameter hp, in i **do**
          $ratio \leftarrow PBEST[d+hp]$ / Upper bound of $PBEST[d+hp]$ layer range
          $convertedValue \leftarrow ratio \times$ Upper bound of $i[d+hp]$ layer range
          $cognitiveComponent \leftarrow c_2 \times r2 \times (convertedValue - i[d+hp])$
          Return *cognitiveComponent*
        end for
      else
        $cognitiveComponent \leftarrow c_2 \times r2 \times (PBEST[d+hp] - i[d+hp])$
        Return *cognitiveComponent*
      end if
      Update velocity of current layer block using computed *XComponent* values
    end for
  end for
end while
Continue PSO algorithm

This strategy applies in the stage of PSO where the velocities for all particles are updated. The algorithm begins by iterating over each particle block, and computing the layer type. If the layer type of the currently updating particle, and the particle it is being compared are not the same, calculate the ratio between the two. This calculation occurs for each hyper-parameter. Per hyper-parameter, get the upper bound of the hyper-parameter of the particle you are comparing the original to. The value of the compared particle's hyper-parameter is then divided by this upper bound, establishing a ratio. This ratio is then multiplied by the upper bound of the original particle's hyper-parameter range. Hence, the final value is then used in the PSO velocity formula to replace the PBEST or GBEST position, as shown below

$$v_{id}^{t+1} = w * v_{id}^{t} + c_1 * r_{1i} * (cognitiveComponent - x_{id}^{t}) + c_2 * r_{2i} * (socialComponent - x_{id}^{t})$$

As shown in Algorithm 4, this strategy is used firstly when calculating the social component, which is dependent on the GBEST particle layer structure, then again when calculating the cognitive component, which is dependent on the PBEST particle layer structure.

Chapter 5

Evaluation

The evaluation of the solution will be two-fold: an evaluation on the success of the evolving network architectures based upon popular test datasets and the time taken to train these architectures. Assessing the performance difference between the project solution and baseline classifiers, using the same datasets, will determine if the project is a success. These baseline classifiers are used to form a comparison with manually tuned architectures rather than a PSO implementation, as there are no existing PSO implementations for evolving CNNs for text classification. An evaluation of the classification accuracy on the same datasets determines if the PSO solutions to evolve the architecture are fit for purpose by improving on previous results. The time taken to train will be evaluated but it is more important to display an increase in performance, though a reduction in training time is still a valuable result.

5.1 Experiment Design

5.1.1 Datasets

The following two datasets are used to evaluate the project solution. The baseline classifiers, hyper-parameter encoding and architecture-encoding of PSO are evaluated against these datasets.

Dataset	Classes	Train Samples	Test Samples
IMDB Polarity	2	25,000	25,000
Yelp Review Full	5	650,000	50,000

Table 5.1: Datasets

The **Large Movie Review** or **IMDB Polarity** dataset contains 50,000 reviews from IMDB. The task of the dataset is binary classification; determining if a given review is positive or negative. It was constructed by categorising 10 star reviews by polarity. Negative reviews were those with 4 stars and below, and positive reviews were those with 7 stars and above. There are equal numbers of positive and negative reviews, totalling 25,000 instances per category. There are 25,000 training instances and 25,000 test instances.

The **Yelp Review Full** dataset is sourced from the Yelp Dataset Challenge in 2015. The task of the dataset is multi-class classification; determining if a given review received one to five stars. This dataset contains 130,000 training instances per each review categorisation with 10,000 testing instances per category too, and this amounts to a dataset with 650,000 training instances and 50,000 testing instances.

However, due to the extent of the Yelp dataset, when training each particle a subset of 25% of the total size was used, meaning each particle only trained on 162500 instances. This

was deemed necessary to accelerate training. In addition to this, initial testing showed this reduction in the training set size did not have a noticable effect on the final accuracy. Both of these datasets have been used in the literature [1, 36, 38, 39, 40, 18, 41], and will inform the evaluation of the PSO solution in conjunction with the baseline classifiers.

5.1.2 Setup

All experiments were performed on a range of hardware. An Nvidia GTX970 GPU was used for the baseline classifiers. The rest of the experiments were ran on Victoria University of Wellington CUDA servers, using the Nvidia RTX2080 and NVidia GTX1080 GPUs installed on those servers. Each result is an average of the classification accuracy across 10 runs. The training parameters used for PSO are a population of 20, which are all evolved for 20 generations. PSO parameters are specified by convention in academic practice [32], $c_1 = c_2 = 1.49618$ and $w = 0.7298$.

As an early stopping criteria to limit the PSO process, whenever the GBEST error loss value reamins the same for 5 generations, if on the next generation it ceased to improve still, the PSO training procedure would terminate and the most performant particle at that time, the current GBEST, would be used as the final particle solution. All significance results are calculated using the Wilcoxon signed-rank test, where a + indicates a method has significantly differently ranked results to be deemed a valid method, and = indicates the two compared methods share a similar distribution of results.

It should be noted that the classification accuracy for Yelp Review Full does not factor in the difference in predicted class compared to the actual class. Yelp Review Full is a multi-class classification problem. If the classifier predicts 2 stars or 5 stars when the true class was 1 star, the prediction in each case would be considered to be a false prediction and not count towards the total number of correct predictions. A formula that did compute a classification accuracy that factors in the distance between the correct classification and the predicted classification would be more communicative of the actual performance of the classifier as this would consider how close a prediction was. However, for evaluating the performance of this dataset, alternative solutions proposed in academia do not use this form of evaluation. As such, we have decided to use the same evaluation, the total number of correctly classified instances divided by the total number of classified instances, as this is what is used in academia thus allows for a fair performance comparison.

Additionally, for the project methods involving PSO, training accuracy results have been included to provide additional information and context behind the project methods.

5.1.3 Academic Benchmark Results

To inform the suitability of the solution and how effective it is, relative research benchmarks are used for comparison. Below is a table containing the most state-of-art results for the datasets used in the evaluation.

Regarding the Yelp Review Full dataset, CL refers to the work by Zhang et al. [1]. S&W refers to the work by Le et al. [41], and it should be noted this approach does not use a CNN. Finally, VD29 refers to the work by Conneau et al. [18], referencing their developed architecture that has a depth of 29.

Regarding the IMDB Polarity dataset, NBSVM-bi refers to the work by Wang et al. [39]. This work too does not use a CNN. ULMFiT refers to the work by Le et al. [40]. The result they achieved is currently 6th globally. Finally, CNN&LSTM refers to the work by Yenter

Benchmark Classification Accuracy		
Dataset	Method	Value
IMDB Polarity	NBSVM-bi [39]	91.22
	ULMFiT [40]	**95.00**
	CNN&LSTM [38]	89.50
Yelp Review Full	CL [1]	62.05
	VD29 [18]	64.72
	S&W [41]	**64.9**

Table 5.2: Benchmark Accuracies for both datasets

et al. [38]. As shown in Table 5.2, the best result achieved on the IMDB Polarity data is ULMFiT. The best result result achieved on the Yelp Review Full dataset is S&W.

5.2 Method Results

Below are the results of the classification accuracy of the baseline classifiers from the first objective. Each result is an average across 10 runs per CNN depth.

5.2.1 Results of Baseline Classifiers

Baseline Classification Accuracy				
CNN Depth	IMDB Polarity	Best	Yelp Review Full	Best
3	86.49 $\pm$ 0.29	86.84	59.05 $\pm$ 0.43	61.14
4	85.53 $\pm$ 0.60	85.52	59.84 $\pm$ 0.21	62.17
5	85.48 $\pm$ 1.08	86.69	60.97$\pm$ 0.47	62.39
6	**86.93** $\pm$ 0.58	89.08	61.75 $\pm$ 0.36	62.42
7	85.96 $\pm$ 0.73	88.64	61.37 $\pm$ 0.40	62.66
8	85.93 $\pm$ 0.49	87.68	**62.56** $\pm$ 0.41	63.43
9	84.49 $\pm$ 1.52	86.73	61.19 $\pm$ 0.73	62.32
10	84.91 $\pm$ 1.37	88.18	60.87 $\pm$ 0.47	61.44

Table 5.3: Baseline Results: Accuracy

According to the results shown in Table 5.3, the baseline classifiers achieved a consistent accuracy across both datasets, in addition to being very similar. The ranges for accuracy on IMDB do not vary greatly. However, it can be noted that there is a trend towards the six-layered architecture being the most performant. The results on the IMDB dataset are comparable to published results by Yenter et al. [38] where they achieved an accuracy of 89.50. Whilst the results are on average 3% lower than this benchmark, they incoporated an LSTM into their architecture which does not allow for a fair comparison. A similar case happens for the Yelp Review Full dataset, the accuracies are relatively stable with a trend towards the middle ranges of CNN depth. The accuracy on the Yelp Review Full dataset is promising against current state-of-art CNN structures for text classification [18, 1]. One aspect to note is how the accuracy across all of the classifiers shows little variance from the shallowest to the deepest architectures. The first reason this may occur is due to the random initialisation of the baseline classifiers, they were not optimally tuned for the classification tasks and thus have performance that is not indicative of their depth. Another reason may be due to these experiments only being run 10 times, a claim could be made about the lack

of statistical significance. However, the standard deviations for the accuracies on a majority of the architectures is sub one-percent, thus this shows the results are reasonably stable. It can also be noted that the nine and ten layer architectures experienced a performance drop, possibly due to over-training of the architecture. The evaluation of these classifiers informed the development of the encoding schemes throughout objective two and three, with the encoding scheme using a six-layered architecture as the basis.

5.2.2 Hyper-Parameter Encoding Scheme

The following section details the results regarding the second objective. PSO-HO refers to the developed hyper-parameter encoding scheme, as part of the second objective. The significance test is performed on the results from this method compared to the 6 layered baseline classifer result, on both datasets.

5.2.3 Results of PSO-HO Testing Performance

PSO-HO Testing Accuracy						
Method	IMDB Polarity	Best	Significance	Yelp Review Full	Best	Significance
6	**86.93** $\pm$ 0.58	89.08	N/A	**61.75** $\pm$ 0.36	62.42	N/A
PSO-HO	86.33 $\pm$ 2.06	88.08	+	61.20 $\pm$ 1.48	62.53	+

Table 5.4: Testing accuracy of the hyper-parameter encoding, comparing PSO-HO to the 6 layered encoding

The results shown in Table 5.4 indicate that the hyper-parameter optimisation is performant relative to the baseline performance, for both IMDB and Yelp. For IMDB the averages between the baseline and PSO-HO is only 0.60%. This is a promising result and lends credence to the fact that automatically evolving the hyper-parameters is able to be performant. However, the standard deviation of PSO-HO is not promising for the stability of the method as this may just be random variance from the seeds used, meaning the method is not performing the same regardless of the seeding. The significance measure for the IMDB dataset indicates that this method is significantly worse, which is not an optimal result. However, the average accuracy between the two methods across both datasets remains relatively the same.

For Yelp, the PSO-HO method is shown to be performant. This is due to the accuracy from PSO-HO only being 0.55% less than the baseline. It is critical to note that the best performing architecture evolved for Yelp using PSO-HO surpassed the average, and the best accuracy of the baseline. This too shows the method can have promising results. However, this method too is also significantly worse than the baseline values.

The PSO-HO method is not as performant as the baseline classifier on average, however on the Yelp Review Full

5.2.4 Results of PSO-HO Training Performance

The training accuracy across both datasets using PSO-HO outperforms the baseline classifier with 6 layers. Compared to the test accuracy, this is the opposite result. We believe this is for a few reasons. The first is due to the inherent tendency for an evolutionary method to overtrain, as it is continually enhancing the evaluation and is only evaluated against the test set when the final evaluation is performed. This promotes overfitting.

PSO-HO Training Accuracy				
Method	IMDB Polarity	Best	Yelp Review Full	Best
6	97.05 $\pm$ 0.64	97.46	67.85 $\pm$ 0.95	68.49
PSO-HO	**98.01** $\pm$ 1.38	99.35	**70.29** $\pm$ 2.01	74.74

Table 5.5: Training accuracy of the hyper-parameter encoding, comparing PSO-HO to the 6 layered encoding

The second reason is there is no formal regularisation used within the training scheme for PSO-HO. Due to being an evolutionary method, it is expected to develop a more performant than a randomly initiated, manually designed, baseline classifier. Whilst the performance of each trained CNN is evaluated on a set of unseen instances to reduce the effects of overfitting, it appears that both classifiers suffer from overfitting.

5.2.5 Architecture Encoding Scheme

The following section details the results regarding the third objective. The methods developed for the third objective are the architecture encoding scheme, as well as the GBEST Inheritance Strategy and Ratio-Based Velocity Strategy. The significance test is performed on the results from the two methods PSO-GI and PSO-RV, compared to PSO-HO, on both datasets.

PSO-GI and PSO-RV Testing Accuracy						
Method	IMDB Polarity	Best	Significance	Yelp Review Full	Best	Significance
PSO-HO	86.33 $\pm$ 2.06	88.08	N/A	**61.20** $\pm$ 1.48	62.53	N/A
PSO-GI	85.97 $\pm$ 1.03	87.82	+	54.18 $\pm$ 1.30	55.70	+
PSO-RV	**86.68** $\pm$ 0.80	87.68	+	58.17 $\pm$ 0.61	59.42	+

Table 5.6: Testing accuracy of the architecture encoding, comparing PSO-GI and PSO-RV to PSO-HO

PSO-GI and PSO-RV Training Accuracy				
Method	IMDB Polarity	Best	Yelp Review Full	Best
PSO-HO	98.01 $\pm$ 1.38	99.35	70.29 $\pm$ 2.01	74.74
PSO-GI	94.76 $\pm$ 14.99	100.00	**94.69** $\pm$ 3.97	98.17
PSO-RV	**99.23** $\pm$ 0.99	100.00	86.55 $\pm$ 3.40	90.15

Table 5.7: Training accuracy of the architecture encoding, comparing PSO-GI and PSO-RV to PSO-HO

5.2.6 Results of GBEST Inheritance Strategy with Architecture Encoding

PSO-GI refers to the developed GBEST Inheritance Strategy, as part of the third objective. The results shown in Table 5.5 indicate that the PSO-GI method is the weakest out of all methods developed for the project. This is reflected in both datasets, especially on the Yelp dataset where there was a performance drop. The validity of the method is called into question from these results, as this shows evolving the CNN architecture using PSO may not be as performant as a hyper-parameter optimisation. PSO-GI is also significantly worse than PSO-HO.

We attribute this loss of performance to the loss of diversity within the population. As the PSO-GI method allows for particles to inherit blocks from the GBEST particle solution, areas of the solution space are skipped over and never explored. In these regions, there may be a local optimum that would enhance the performance of the system.

It should be noted that PSO-GI outperformed the best result from PSO-RV on the IMDB dataset. We attribute this to the particular seeding of the data, where a local minimum was found early and explored. Due to the PSO-GI method allowing for a local search near the current GBEST position, this is a valid reason why the best performance of PSO-GI outperforms PSO-RV on the IMDB dataset. This result speaks to the solution space of IMDB also being relatively simplistic. Due to being a binary classification, the class boundaries may be quite large. However, when compared to Yelp, a multi-class classification problem, PSO-GI is the weakest method. This indicates for a more complex search space such as Yelp, the reliance on a local search decreases performance, and a global search still needs to be performed.

The training performance of PSO-GI is the highest amongst all the methods on the Yelp Review Full dataset, and second on IMDB Polarity. This confirms that the reduction in test performance can be attributed to overfitting. The PSO-GI method enables a local search of the most well performing GBEST particle at any evolution. However, this decreases the diversity, and thus will overfit to the few local minimum regions discovered. In conjunction with this, it also has an unusually high standard deviation. Upon inspection, this is due to an outlier. One experiment run had a training accuracy of 54%, but a testing accuracy of 83%.

5.2.7 Results of Ratio-based Velocity Strategy with Architecture Encoding

PSO-RV refers to the developed Ratio-based Velocity Strategy, as part of the third objective. The results shown in Table 5.5 indicate that for both datasets, PSO-RV outperforms PSO-GI, and shows promise compared to PSO-HO especially on the IMDB dataset, where PSO-RV is the most well performing method. It outperforms PSO-HO by 0.35% on the IMDB dataset. PSO-RV is also significantly worse than PSO-HO.

A significant metric associated with the PSO-RV results is the standard deviation. Across both datasets, the standard deviation is below 0.8. Relative to the other results, this value is the lowest. This indicates the PSO-RV method is more stable than the other methods, as well as the accuracy provided is more likely to represent the actual accuracy if ran for 30 experiment runs.

We attribute the success of this method to the more common sense approach to using PSO. The output of the PSO-RV method within the code is a value that better represents the position of the particles in space, rather than their raw hyper-parameter value which is used in the PSO-GI method to update the hyper-parameters of each layer. This allows for lesser changes in the velocity, promoting exploration. PSO-RV effectively acts as a measure of the difference between two particles, and once this difference is established, the PSO formulas can be calculated and return more sensible velocity values.

However, this method did lead to an improved result on the Yelp dataset. PSO-RV saw a 4% increase in accuracy from PSO-GI, but still trails PSO-HO by 3.03%. A reason for this is most likely the shallowness of the architecture limits the potential accuracy. The networks evolved for the Yelp dataset using the PSO-RV method were wide and shallow, often evolving to be only constructed from convolutional layers. This behaviour indicates that the Yelp dataset required a deeper architecture with a larger filter map range than what was achievable from the encoding scheme. This is a promising conclusion as further research involved with evolving deeper structures may prove beneficial.

The training performance of PSO-RV is the highest amongst all the methods on the IMDB Polarity dataset, and second on Yelp Review Full. This is an interesting result, due to how well it performs compared to PSO-GI on the test set.

It is promising to see a lower training accuracy on the Yelp Review Full dataset, compared to PSO-GI. This shows that the developed of PSO-RV after PSO-GI was well founded, the method avoided a local search and the particle velocities are less sporadic, thus enhancing the performance. It also shows that a more sensible approach to comparing particles shows potential to be quite performant. However, this still indicates that overfitting occurred.

5.2.8 Training Time

The following section includes the training times for the various methods. These values are given in seconds taken to training a full architecture. For simplicity's sake, only one result was included from the baseline classifiers as the 6 layered architecture has been the focus of the development for this project.

Average Training Time (seconds)		
Depth	IMDB Polarity	Yelp Review Full
6	49.2	2236.0
PSO-HO	12423	48600
PSO-GI	10106	68534
PSO-RV	18661	64292

Table 5.8: Preliminary Results: Time

According to the results shown in Table 5.2, PSO-HO takes the least time to optimise the CNN architecture. An obvious conclusion is because it does not need to evolve various architectures, the architecture enforced may inherently not take much time to train. This is in contrast to the PSO-RV method which can evolve architectures that make take drastically more time to train. Out of all the project methods, PSO-GI is the quickest to train on Yelp and PSO-HO is the quickest to train on IMDB. Compared to the baseline classifiers the results in Table 5.2.8 are expected, as the evolutionary process is quite computationally intensive.

The reason for PSO-GI ending training quicker on the IMDB Polarity dataset, than the other methods is due to the early stopping criteria, we propose. As shown by the training results, a local optimum is reached and over-fitting occurs, thus the error rate does not increase after this point. Due to the use of elitism in this method, this behaviour is not unusual. Contrasted with the better performing method PSO-RV, where this behaviour was not as prevalent due to using a ratio-based change.

In terms of the project objectives, a primary motivator behind them was the time factor taken to manually tune an architecture. From these results, if the optimisation is run on a GPU overnight the optimisation will have finished, without any human intervention. Thus, a secondary goal for the objectives has been achieved.

5.3 Analysis

Table 5.5 represents the final table of all results regarding the project work. The solution developed for this project consists of these three methods, PSO-HO, PSO-GI and PSO-RV. The most performant method developed for the Yelp dataset is PSO-HO, and for the IMDB dataset it is PSO-RV.

The image below depicts the final results of each method in Figure 5.1.

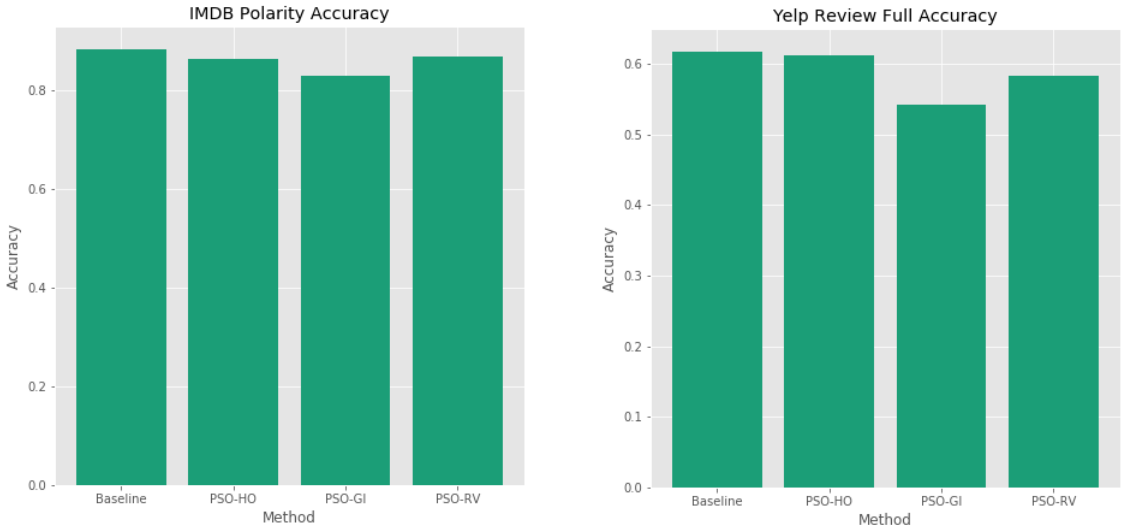

Figure 5.1: Bar chart of project methods averages on both datasets

The first conclusion we can draw is PSO-HO and PSO-RV both have promising performance compared to the baseline classifiers. The relative accuracy of the first two, and last method, indicate the methods developed are performant. They are all based on a novel encoding scheme for a CNN architecture, automatically evolve to be more performant and achieve promising results. An aspect that must be considered when judging the performance of both methods on the Yelp Review Full dataset is how only a subset of 25% of the training data was used in the PSO process. This was due for time considerations, however it may have increased the performance on any of the project methods.

A summary of the methods developed as the project solution is shown in a box plot below in Figure 5.2.

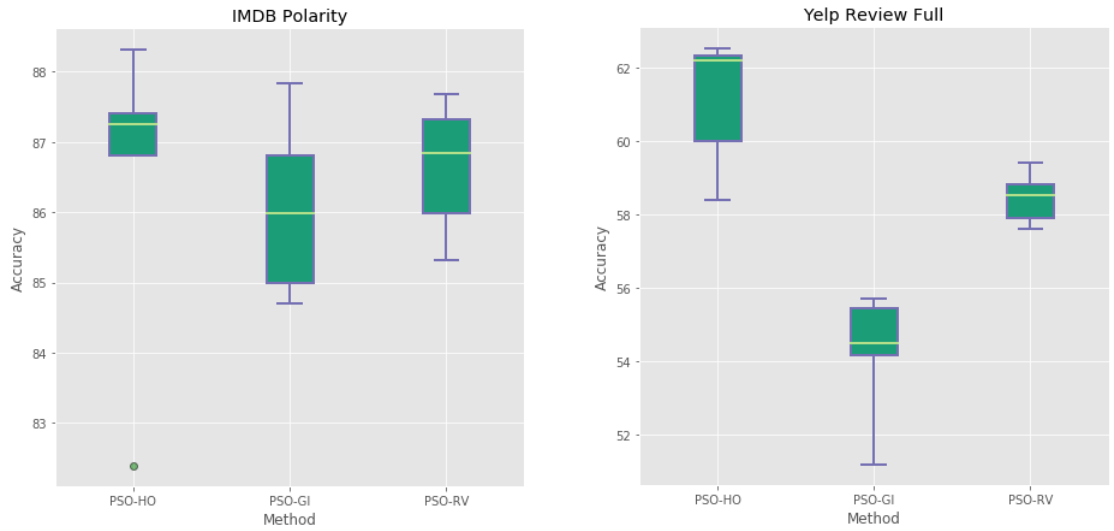

Figure 5.2: Box and Whiskers plot of project methods on both datasets

This plot clearly supports the claim that IMDB is an easier problem for the CNN structures to classify. Because IMDB is a binary classification problem, the solution space is more easily separable. For Yelp however, a 5 multi-class classification problem, this is a far greater issue thus is an inherently harder task to achieve a high classification rate.

Considering the methods developed, in both plots PSO-HO outperforms the other methods on average, bested only by PSO-RV on the IMDB dataset. This can be attributed to the reasoning stated above. The single outlier in PSO-HO can be attributed to the stochastic nature of the methods finding a local minimum and never discovering a more optimal position in the solution space. Aside from this, PSO-HO has the most promising results. However, regarding the motivation of the project, PSO-HO is the least automatic as it requires a pre-existing architecture.

PSO-GI has the most variable performance, as shown by the plot the upper and lower bounds were either tightly clustered to body of the box, or were far enough way to have an extended tail. This further supports the claim of the variability of the method. PSO-RV can be shown in both plots to have the most tightly grouped box indicating that 50% of the results are quite similar.

A proper claim can not be made that PSO-RV is wholly a better performing method than PSO-HO on IMDB polarity. As shown in the box and whiskers plot, there is a greater distribution of the middle 50% of results for IMDB polarity of PSO-RV compared to PSO-HO.

An aspect of the project that has not been considered is the type of architectures evolved, shallow architectures versus deep architectures when regarding the PSO-RV and PSO-GI methods. For most researchers, a 6 layered architecture, or less, would be considered more so a shallow architecture than a deep one. This is important as the authors behind VGG16 showed within their research that deeper networks with less filters could achieve better results than a shallow architecture [19]. This point could be a reason why the evolved architectures using the Yelp Review Full dataset are less performant than the baseline classifiers; due to the imposed limit of the architecture encoding being 6 layers, allowing for deeper architectures may have proven to be more performant.

Relative to the manually-designed architectures, these results do not match any of the academic work regarding research done using the IMDB dataset. This is due to a few reasons. Firstly, a lot of state-of-art results either do not use CNNs [40, 39] or they utilise an LSTM in conjunction with a CNN [38]. This last result is less than 4% better forming than PSO-RV. This is a promising realisation and supports the validity of PSO-RV being a performant method. This is a more comparative indicator of performance than the other IMDB results.

However, all results on the Yelp dataset show promising performance, especially PSO-HO which achieved an accuracy less than 4% off the state-of-art results. Compared to VD29, a CNN architecture with 29 layers, achieving accuracy within 4% of that shows immense promise for the project solution. All that is required is an extension to the depth the architecture can evolve, avoiding the shallow and wide architecture problem, and the solution may excel on this dataset. In addition to this, it is promising that such a shallow network achieved comparable results.

Considering the project objectives, each of these methods meet the requirements. The baseline classifier, hyper-parameter optimised classifier, and the architecture optimised classifier are all novel works that show promising results relative to state-of-art methods.

Chapter 6

Conclusions

This chapter addresses the future work that the project would benefit from or be extended by, in addition to conclusionary statements regarding the report and the final results of the project.

6.1 Conclusions

For this project, various novel encoding schemes and PSO enhancement methods were designed, implemented and evaluated to automatically evolve CNNs for text classification.

The aim of this project was develop a system to automatically evolve CNNs for text classification. CNNs have shown promising results within the field of text classification, thus the investigation into enhancing CNN usage is a worthy effort. The motivation stems from the inherent difficulty to perform such a task, in addition to the lack of research in the area. This project investigated evolving two aspects of deep convolutional neural network structures for text classification using PSO. Particle Swarm Optimisation was chosen as the method to evolve these CNN structures, due to the benefits listed in section 2.4.1. To begin with, the first objective is achieved through attaining baseline performance of two datasets, using CNNs where no evolutionary optimisation was used to enhance their performance. The second objective was met by developing a system to evolve the hyper-parameters of a fixed CNN architecture. The third objective was met by developing a system to evolve architecture as well as the hyper-parameters, in conjunction to two methods that enhance the usage of PSO. The encoding scheme developed for the third objective is also variable-length, which addresses a major limitation of PSO. The above developed methods show promising results relative to state-of-art-methods. These objectives were all successfully achieved. The aim can therefore be said to have been met through the above objectives.

In summary, the contributions made through the course of this project are:

1. A novel encoding scheme for representing CNN hyper-parameters to aid the PSO process. This novel contribution is the first of its kind, and achieves promising results compared to state-of-art methods.

2. A novel variable-length encoding scheme for representing CNN architecture and hyper-parameters to aid the PSO process. This is a significant contribution as it is the first of its kind, as well as being a variable-length encoding scheme, which addresses a major limitation of PSO.

3. Two methods that enhance the performance of PSO when used in tandem with the architecture encoding scheme. This work can be adapted for other indirect encod-

ing schemes thus are applicable beyond the scope of this project work. The PSO-RV method achieves promising results compared to state-of-art methods.

In addition to these contributions, to the best of the author's knowledge, the project work is the first of its kind to perform evolutionary optimisation on CNNs for text classification, using PSO. This is significant as the project work proves these optimisation methods are worth investigating in future research.

6.2 Future Work

The results obtained by the project were promising, they provided an insight into a research area that has little prior academic work done surrounding it. This addresses a key limitation identified in the academic work. However, there is a significant amount of work that can be done to further the research.

The encoding schemes developed are at the forefront of this project, and can be enhanced to be more performant. In regard to the hyper-parameter encoding, enabling more variables or differing types would prove useful to enhancing hand-made architectures. In regard to the architecture encoding, there is a great amount that can be explored. Considering different activation functions, different layers, more hyper-parameters and the ability to increase the depth to avoid shallow but wide networks that often occured during the PSO optimisation.

Further work can also be performed to the methods aimed at enhancing PSO. The GBEST Inheritance method has a lot of potential to be implemented with a percentage chance of occurring, or instead of adopting the GBEST layer block hyper-parameters, it moves to a random position within a certain range of that block. Considering the Ratio-Based Velocity method, a differing encoding scheme may be better suited to this approach, one that has less difference in the layer and hyper-parameter range values. This could be expressed using a binary-string, or a continuous encoding that uses values from 0 to 1.

Another research direction to consider is how to best evaluate two particles with differing layer types at the same positions. Simply put, the traditional PSO formula takes the difference between the GBEST or PBEST particle, and each particle within the population, when updating the velocity. If a comparison occurs at the same position between a convolutional layer, and a pooling layer, currently there is no further information gathered or used to understand how effective and necessary that layer is at that particular position. Thus, a direct comparison only considers how performant the overall architecture is, and by proxy the layers at each position. The aim of the Ratio-Based Velocity method was to address the limitation found. However, the Ratio-Based Velocity method was still limited by the encoding scheme representation, and inherent structure of the PSO formulas. Hence, further research can be done regarding this issue.

A direction of future research could also be the incorporation of other deep learning structures, such as an LSTM which was shown to have excellent performance in [38]. Whether it is evolving a CNN-LSTM structure in tandem, or just LSTM are both avenues that could be explored. This is especially true for incoporating regularisation strategies within evolved CNNs. As shown by the training performances of each method, overfitting was prevalent in most methods, and is another potential area for future work.

Bibliography

[1] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 649–657. [Online]. Available: http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf

[2] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: http://arxiv.org/abs/1408.5882

[3] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017. [Online]. Available: http://arxiv.org/abs/1703.00548

[4] J. Kennedy and W. M. Spears, "Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, May 1998, pp. 78–83.

[5] A. Hotho, A. Nurnberger, and G. Paass, "A brief survey of text mining," *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, vol. 20, pp. 19–62, 01 2005.

[6] E. Alpaydin, *Introduction to Machine Learning*, 3rd ed., ser. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press, 2014.

[7] C. C. Aggarwal and C. Zhai, *A Survey of Text Classification Algorithms*. Boston, MA: Springer US, 2012, pp. 163–222. [Online]. Available: https://doi.org/10.1007/978-1-4614-3223-4_6

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[9] K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, 01 2019.

[10] Xin Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[11] K. O. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artif. Life*, vol. 9, no. 2, pp. 93–130, Apr. 2003. [Online]. Available: http://dx.doi.org/10.1162/106454603322221487

[12] M. Feurer and F. Hutter, "Hyperparameter optimization," in *AutoML: Methods, Sytems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer, Dec. 2018, ch. 1, pp. 3–37.

[13] A. E. Eiben and J. Smith, "From evolutionary computation to the evolution of things," *Nature*, vol. 521, pp. 476–482, May 28 2015. [Online]. Available: https://search-proquest-com.helicon.vuw.ac.nz/docview/1685003412?accountid=14782

[14] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[15] Y. Shi, *Critical developments and applications of swarm intelligence*. Hershey, PA : Engineering Science Reference, 2018.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, Nov 1995, pp. 1942–1948 vol.4.

[17] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, May 1998, pp. 69–73.

[18] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 1107–1116. [Online]. Available: https://www.aclweb.org/anthology/E17-1104

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[20] J. Y. Lee and F. Dernoncourt, "Sequential short-text classification with recurrent and convolutional neural networks," *CoRR*, vol. abs/1603.03827, 2016. [Online]. Available: http://arxiv.org/abs/1603.03827

[21] A. Rios and R. Kavuluru, "Convolutional neural networks for biomedical text classification: Application in indexing biomedical articles," in *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, ser. BCB '15. New York, NY, USA: ACM, 2015, pp. 258–267. [Online]. Available: http://doi.acm.org/10.1145/2808719.2808746

[22] T. Yamasaki, T. Honma, and K. Aizawa, "Efficient optimization of convolutional neural networks using particle swarm optimization," in *2017 IEEE Third International Conference on Multimedia Big Data (BigMM)*, April 2017, pp. 70–73.

[23] F. Ye, "Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data," *PLOS ONE*, vol. 12, no. 12, pp. 1–36, 12 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0188746

[24] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, "Particle swarm optimization for hyper-parameter selection in deep neural networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 481–488. [Online]. Available: http://doi.acm.org/10.1145/3071178.3071208

[25] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, July 2018, pp. 1–8.

[26] B. Fielding and L. Zhang, "Evolving image classification architectures with enhanced particle swarm optimisation," *IEEE Access*, vol. 6, pp. 68 560–68 575, 2018.

[27] H. M., T. Han, and N. E., "Hybrid algorithm for the optimization of training convolutional neural network," *International Journal of Advanced Computer Science and Applications*, vol. 6, 10 2015.

[28] F. E. Fernandes Junior and G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm and Evolutionary Computation*, vol. 49, pp. 62–74, 06 2019.

[29] B. Tran, B. Xue, and M. Zhang, "Variable-length particle swarm optimization for feature selection on high-dimensional classification," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 473–487, June 2019.

[30] M. van Knippenberg, V. Menkovski, and S. Consoli, "Evolutionary construction of convolutional neural networks," *CoRR*, vol. abs/1903.01895, 2019. [Online]. Available: http://arxiv.org/abs/1903.01895

[31] W. Zhu, W. Yeh, J. Chen, D. Chen, A. Li, and Y. Lin, "Evolutionary convolutional neural networks using abc," in *Proceedings of the 2019 11th International Conference on Machine Learning and Computing*, ser. ICMLC '19. New York, NY, USA: ACM, 2019, pp. 156–162. [Online]. Available: http://doi.acm.org.helicon.vuw.ac.nz/10.1145/3318299.3318301

[32] Y. Zhang and B. C. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *CoRR*, vol. abs/1510.03820, 2015. [Online]. Available: http://arxiv.org/abs/1510.03820

[33] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically designing cnn architectures using genetic algorithm for image classification," *arXiv preprint arXiv:1808.03818*, 2018.

[34] L. J. V. Miranda, "PySwarms, a research-toolkit for Particle Swarm Optimization in Python," *Journal of Open Source Software*, vol. 3, 2018. [Online]. Available: https://doi.org/10.21105/joss.00433

[35] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[36] 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*, ser. HLT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 142–150. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002472.2002491

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[38] A. Yenter and A. Verma, "Deep cnn-lstm with combined kernels from multiple branches for imdb review sentiment analysis," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, Oct 2017, pp. 540–546.

[39] S. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ser. ACL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 90–94. [Online]. Available: http://dl.acm.org/citation.cfm?id=2390665.2390688

[40] J. Howard and S. Ruder, "Fine-tuned language models for text classification," *CoRR*, vol. abs/1801.06146, 2018. [Online]. Available: http://arxiv.org/abs/1801.06146

[41] H. T. Le, C. Cerisara, and A. Denis, "Do convolutional networks need to be deep for text classification ?" *CoRR*, vol. abs/1707.04108, 2017. [Online]. Available: http://arxiv.org/abs/1707.04108