AUTOMATED DESIGN OF THE DEEP NEURAL NETWORK PIPELINE

by
Mia Gerber

Submitted in fulfillment of the requirements for the degree
Master of Science (Computer Science)
in the
Department of Computer Science
Faculty of Engineering, Built Environment and Information Technology
UNIVERSITY OF PRETORIA
December 2021

ABSTRACT

AUTOMATED DESIGN OF THE DEEP NEURAL NETWORK PIPELINE

by

Mia Gerber

Supervisor: Prof Nelishia Pillay

Department: Department of Computer Science

University: University of Pretoria

Degree: MSc Computer Science

Keywords: automated design, deep neural network pipeline, transfer learning, sentiment analysis, spam detection, image segmentation, image classification

Deep neural networks have been shown to be very effective for image processing and text processing. However the big challenge is designing the deep neural network pipeline, as it is time consuming and requires machine learning expertise. More and more non-experts are using deep neural networks in their day-to-day lives, but do not have the expertise to parameter tune and construct optimal deep neural network pipelines. AutoML has mainly focused on neural architecture design and parameter tuning, but little attention has been given to optimal design of the deep neural network pipeline and all of its constituent parts. In this work a single point hyper heuristic (SPHH) was used to automate

the design of the deep neural network pipeline. The SPHH constructed a deep neural network pipeline design by selecting techniques to use at the various stages of the pipeline, namely: the preprocessing stage, the feature engineering stage, the augmentation stage as well as selecting a deep neural network architecture and relevant hyper-parameters. This work also investigated transfer learning by using a design that was created for one dataset as a starting point for the design process for a different dataset and the effect thereof was evaluated. The reusability of the designs themselves were also tested. The SPHH designed pipelines for both the image processing and text processing domain. The image processing domain covered maize disease detection and oral lesion detection specifically and text processing used sentiment analysis and spam detection, with multiple datasets being used for all the aforementioned tasks. The pipeline designs created by means of automated design were compared to manually derived pipelines from the literature for the given datasets. This research showed that automated design of a deep neural network pipeline using a single point hyper-heuristic is effective. Deep neural network pipelines designed by the SPHH are either better than or just as good as manually derived pipeline designs in terms of performance and application time. The results showed that the pipeline designs created by the SPHH are not reusable as they do not provide comparable performance to the results achieved when specifically creating a design for a dataset. Transfer learning using the designed pipelines is found to produce results comparable to or better than the results achieved when using the SPHH without transfer learning. Transfer learning is only effective when the correct target and source are chosen, for some target datasets negative transfer occurs when using certain datasets as the transfer learning source. Future work will include applying the automated design approach to more domains and making designs reusable. The transfer learning process will also be automated in future work to ensure posi-

tive transfer occurs. The last recommendation for future work is to construct a pipeline for unsupervised deep neural network techniques instead of supervised deep neural network techniques.

PLAGIARISM DECLARATION

I, Mia Gerber (Student Number: 15016502), declare that:

1. The research reported in this thesis, except where otherwise indicated or acknowledged, is my original work;
2. This thesis has not been submitted in full or in part for any degree or examination to any other university;
3. This thesis does not contain other persons data, pictures, graphs or other information, unless specifically acknowledged as sourced from other persons;
4. This thesis does not contain other persons writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
   (a) their words have been re-written but the general information attributed to them has been referenced;
   (b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced
5. Where I have reproduced a publication of which I am author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself along and have fully referenced such publications.
6. This thesis does not contain text, graphics and tables copied and pasted from the internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Student Name: Student Signature:

Date:

SUPERVISOR DECLARATION

I confirm that this work was done under my supervision and it is the candidate's original work. As the candidate's supervisor, I have approved this thesis for submission.

Supervisor Name: Supervisor Signature:

Date:

PUBLICATIONS

The following publications are associated with the research presented in this thesis:

1. Gerber,M., Pillay,N. Automated Design of the Deep Neural Network Pipeline, Expert Systems with Applications, Elsevier *(under review)*
2. Gerber,M., Pillay,N.,Khammissa, R. A Comparative Study of Supervised and Unsupervised Neural Networks for Oral Lesion Detection, Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (SSCI 2021), December 2021.
3. Gerber, M.,Pillay, N., Holan, K., Whitham, S. A.,Berger,D. K. Automated Hyper-Parameter Tuning of a Mask R-CNN for Quantifying Common Rust Severity in Maize,2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1-7, doi: 10.1109/IJCNN52387.2021.9534417.
4. Pillay N., Gerber M., Holan K., Whitham S.A., Berger D.K. (2021) Quantifying the Severity of Common Rust in Maize Using Mask R-CNN. In: Rutkowski L., Scherer R., Korytkowski M., Pedrycz W., Tadeusiewicz R., Zurada J.M. (eds) Artificial Intelligence and Soft Computing. ICAISC 2021. Lecture Notes in Computer Science, vol 12854., pp. 202-213, Springer, Cham. https://doi.org/10.1007/978-3-030-87986-$0_1$8. Student Name:

Student Signature:

Date:

ACKNOWLEDGEMENTS

I would like to offer my sincerest gratitude to Prof Nelishia Pillay for her mentorship and support in the development of this thesis.

The work presented in this thesis is supported by the National Research Foundation of South Africa (Grant Numbers 46712). Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

LIST OF ABBREVIATIONS

NN	Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
SPHH	Single Point Hyper Heuristic
DNNP	Deep Neural Network Pipeline
DS	Design String
LLPH	Low Level Perturbative Heuristic
EA	Evolutionary Algorithm
GP	Genetic Programming

TABLE OF CONTENTS

CHAPTER 1

Introduction

This chapter provides an overview of what this study aims to investigate by presenting the objectives and contributions. This chapter also presents the layout of the remainder of the thesis, providing a short summary of each chapter.

1.1 Purpose of study

Deep neural networks have led to significant advances in the fields of text and image processing [1] [2]. One of the challenges associated with applying deep neural networks is the high design time associated with the deep neural network pipeline and the expertise needed for this process [3]. Thus, there is a need to automate the design of this pipeline. While there has been a fair amount of research into automating the design of specific neural network aspects like the architecture for example NAS [4] and the weights for example Auto-Net [3], there has not been previous research into automating the design of the entire pipeline. There has been previous work on automating the pipeline for data classification resulting in the development of TPOT [5], however this has not been done for the deep neural network pipeline.

The first purpose of this study is to automate the design of the deep neural network pipeline by means of a single point selection perturbative hyper-heuristic (SPHH). Various techniques, including evolutionary algorithms [6] and hyper-heuristics [7] have been successfully employed for the automated design of machine learning techniques. An advantage of using selection hyper-heuristics for automated design is that in certain instances the same performance can be achieved with less computational effort than needed by evolutionary algorithms. The SPHH selects the preprocessing techniques, augmentation techniques, feature engineering techniques, the neural network architecture and the neural network parameters to construct an end-to-end deep neural network pipeline. The SPHH is used to design deep neural network pipelines for datasets in both the text processing and image processing domain, the reason being as deep neural network pipelines are frequently used for both text processing and image processing. The SPHH automated design technique is validated using sentiment analysis, spam detection, maize disease detection and oral lesion detection datasets.

The second purpose of this study is to evaluate the effect of transfer learning on the automated design of the deep neural network pipeline. Transfer learning is a concept that has seen some initial research in the automated design field [8] but has not been evaluated at all for the automated design of deep neural

network pipelines. Transfer learning generally is used to boost the performance of a system by applying knowledge obtained during prior executions to a current execution. The effect of transfer learning on the SPHH is analysed by using the deep neural network pipeline designs created as the means of transferring knowledge.

This study also evaluates whether the deep neural network pipelines designed by the SPHH are reusable. Reusability testing is done by applying a deep neural network pipeline designed for one dataset as-is for a different dataset. For a design to be reusable, it must produce results that are comparable to results that a deep neural network pipeline that is specially designed from scratch would produce.

The work presented here helps to advance the field of automated design, specifically pertaining to deep neural networks, and in doing so contributes to making machine learning more accessible to non-experts and more effective for experts.

1.2 Objectives of study

The work presented starts with a thorough review of the literature in the fields of deep neural networks, text processing, image processing and automated design. The literature survey informs the construction of a selection perturbative hyper-heuristic (SPHH) which is used as the mechanism to automate the design of the deep neural network pipeline. The SPHH is applied to both the text processing domain specifically sentiment analysis, spam detection as well as the image processing domain specifically maize disease detection and oral lesion detection. Experimentation performed using the SPHH is done in order to fulfill the objectives of this study, which can be summarized as follows:

1. To automate the design of the deep neural network pipeline using a selection perturbative hyper-heuristic for both text processing and image processing.
2. To investigate the effects of transfer learning on the automated design of a deep neural network pipeline.

1.3 Contributions

The main contribution of this work is automating the design of the deep neural network pipeline for both text processing and image processing. In addition, the other contributions made by this study are outlined below:

1. Is the first study to use a selection perturbative hyper-heuristic for automated design in the deep neural network domain.
2. Is the first study to automate the design of an end-to-end deep neural network pipeline.
3. Is the first study to analyse the effect of transfer learning when transferring the deep neural network pipeline designs.

1.4 Layout of thesis

This section gives an overview and description of each of the chapters that follows Chapter 1.

1.4.1 Chapter 2 - Deep neural networks

This chapter presents fundamental deep neural network concepts as well as a brief history of neural networks. Several popular deep neural network architectures are presented. The neural network architectures included are those that are incorporated into the automated design technique described in later chapters.

1.4.2 Chapter 3 - Text processing with deep neural networks

This chapter focuses on the application of deep neural networks to the text processing domain. Basic text processing concepts are discussed and the two relevant domains this study uses, namely sentiment analysis and spam detection are introduced. Techniques that have previously been applied to sentiment analysis and spam detection are discussed. The deep neural network pipeline for text processing and the various stages it comprises is also outlined.

1.4.3 Chapter 4 - Image processing with deep neural networks

This chapter discusses deep neural networks as applied to the image processing domain. Fundamental concepts and terminology within image processing are explained including the two relevant domains this study uses, namely segmentation and classification. For both segmentation and classification the various techniques that have been previously applied to these domains are concisely discussed. The deep neural network pipeline for image processing and the various stages it comprises is also described.

1.4.4 Chapter 5 - Critical analysis

This chapter presents a critical analysis of the work discussed in Chapters 2, 3 and 4. The critical analysis shows which aspects from the literature this study uses and the gaps in research this study addresses. From the critical analysis three areas of focus are defined: automated design, hyper-heuristics and transfer learning. Each focus area is then discussed in turn. First basic automated design concepts and terminology are presented briefly before discussing how automated design has been used in the deep neural network field. Attention is given specifically to hyper-parameter optimization and methods that design pipelines as they are most relevant to this work. Secondly the use of hyper-heuristics to perform automated design is presented by describing hyper-heuristics generally then discussing literature wherein hyper-heuristics have been used for automated design. Lastly general terminology relating to transfer learning is presented as well as

the motivations for performing transfer learning. Transfer learning has not been used for the automated design of a deep neural network pipeline, so the use of transfer learning within automated design is described by referring to literature wherein transfer learning is done for various other automated design techniques.

1.4.5 Chapter 6 - Research methodology

This chapter describes the research methodology that was followed in order to achieve the objectives described. This work makes use of a proof by demonstration methodology, this methodology and the experimentation that is done is described in detail. This chapter also provides an overview of the datasets, performance measures, statistical measures and technical specifications for the hardware that were used as part of the experimentation process.

1.4.6 Chapter 7 - Single point hyper-heuristic

This chapter describes the single point selection perturbative hyper-heuristic (SPHH) employed to design the deep neural network pipeline. This chapter presents both the structure of the design that the SPHH creates as well as the structure of the SPHH itself. This chapter also lists the hyper-parameter values and various design decisions and explains how they were derived.

1.4.7 Chapter 8 - Results and discussion

This chapter presents the results of the three experiments performed (as described in Chapter 7) both in terms of text processing and image processing. For Experiment 1 the designs from the SPHH are compared to the results of manually derived pipelines from the literature. For Experiment 2 the results of using transfer learning with the SPHH are compared to the results of not using transfer learning with the SPHH. For Experiment 3 the results from reusing pipeline designs is compared to the results of designing a deep neural network pipeline from scratch for each dataset. The results are discussed in the context of this study's objectives.

1.4.8 Chapter 9 - Conclusions and future work

Finally this chapter summarizes the results from Chapter 8, linking them to the research objectives. Recommendations for future work are also presented.

CHAPTER 2

Deep neural networks

This chapter presents fundamental deep neural network concepts as well as a brief history of neural networks. In Section 2.1 basic deep neural network concepts and terminology are discussed and Section 2.2 presents a brief history of neural networks up to the present day. Section 2.3, Section 2.4 and Section 2.5 each present a popular deep neural network architecture in turn, namely the Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM). The neural network architectures included are those that are focused on in this thesis. Finally Section 2.6 summarizes the chapter.

2.1 Introduction to Neural Networks

In order to discuss deep neural networks and the state of the literature surrounding them, it is necessary to first understand what a neural network is. A neural network can be defined as an interconnected system of artificial neurons organized in a layered structure where a layer can consist of multiple neurons. A single neuron receives one or more inputs, processes the input, and then outputs the result of processing. Each input is associated with a weight, which determines the contribution of that specific input towards calculating the output. Once the weighted inputs are summed, they are processed using an activation function.

The activation function is used to transform the input into an output. Reasons for transformation include: scaling a value within a certain range and modelling non-linear relationships. Activation functions need to be computationally inexpensive as they are calculated many times during the training of a neural network. The choice of which activation function to use is influenced by the type of neural network architecture being used, the position of a given neuron in that architecture as well as the characteristics of the input data [9]. The result of the activation function is what the neuron outputs. The output of a single neuron can be used as input to one or more neurons. If the neuron resides in the output layer, the output of that neuron is the output of the neural network as a whole. Figure 2.1 illustrates the working of a basic neural network neuron.

A basic neural network architecture consists of a first layer that is dedicated to receiving input, this layer has a number of neurons equal to the number of inputs being received. The first layer is followed by a number of hidden or non-linear layers. A hidden or non-linear layer need not have the same number of neurons as the first (input) layer. The last layer in the architecture is a layer that

is dedicated to outputting the result of the computation done by the preceding layers.

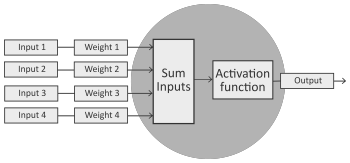

Fig. 2.1. A basic neural network neuron

The values for the neuron weights are iteratively adjusted by means of an optimization algorithm (referred to as an optimizer); this process is called training. The training process consists of multiple epochs; the weight values are updated at the end of an epoch. At the start of training the neuron weight values are either initialized randomly or set to weight values from a previous round of training. The training process completes when weight values are found that results in the neural network outputting a high accuracy value and low loss value. The accuracy for a neural network is a function of how often the neural network delivers correct predictions, the loss value is the prediction error for the neural network. Calculation of the loss value is done by means of a loss function; the loss function used depends on the task at hand. An example of a loss function is Mean Squared Error (MSE) which uses the value predicted by the neural network as well as the correct value and calculates the mean of the squared differences. The MSE loss function is suited towards neural networks that produce real valued outputs for regression tasks as opposed to simple binary outputs for classification. In the case of a neural network that is solving a classification task, if the classes are binary a loss function such as Binary Crossentropy (BCE) would be used, if there are multiple classes Categorical Crossentropy (CCE) would be used as a loss function. The loss function chosen is based on the problem domain and neural network architecture.

The optimization algorithm uses the loss function to update the weights for all neurons in the neural network. For example, if gradient descent is used as an optimizer the first order derivative of the loss function is used to calculate new values for the weights that minimizes the loss function, if the loss function cannot be minimized any further training completes. There are many optimization algorithms to choose from and other popular choices include: Stochastic Gradient Descent (SGD), Adam [10], AdaGrad [11], AdaDelta [12] and more. SGD is a very basic optimizer and most other optimizer algorithms at their core make use of the fundamentals of SGD. SGD trains a neural network by updating the weights of the neural network by means of gradient descent. Gradient descent attempts to find a minimum for a function by incrementally updating the function parameters. In the context of neural network optimization, we are trying

to minimize the loss and incrementally update the weight values. The first step to gradient descent is calculating the gradient of the loss function value after getting a prediction from the neural network for a given input. The magnitude and direction of the error of the neural network prediction is referred to as the gradient. The gradient is then multiplied by a parameter called the learning rate which provides us with a step size. The learning rate is a hyper-parameter that is set at the start of training. The step size given by the learning rate is subtracted from the current weight value to give us the new weight value. The stochastic aspect of SGD refers to when gradient descent is performed, a dataset instance is chosen at random, instead of in a predetermined order. The choice of optimizer is heavily dependent on both the problem domain that the neural network is being applied to, as well as the overall architecture of the neural network [13].

The optimizer uses a hyper-parameter called the learning rate which controls how large a change in weight value the optimizer can make from one epoch to another. If the learning rate is a large value, the weight values change dramatically between epochs, if the learning rate is a small value, the weight values change less dramatically between epochs. Having a large learning rate could result in faster training, but it could also result in optimum weight values being missed or a local optimum being found instead of a true global optimum. Conversely a smaller learning rate can slow down the training process, meaning it can take longer for optimum weight values to be found, but because weight values are being updated in smaller increments, the chances of missing an optimum set of weight values or finding a local optimum is lower. Techniques that dynamically adjust the learning rate values throughout the training process are commonly employed to exploit the positive effects and dampen the negative effects of both a large and small learning rate value [14]. A momentum parameter is also commonly used as a means of avoiding local minima during the training process. The momentum parameter is real valued and forms part of the weight value update process by adding a fraction (the exact fraction added is determined by the momentum parameter's value) of the current weight value to the next weight value. The momentum parameter has the effect of strengthening the weight update process in a certain direction i.e. if a certain weight value has been steadily increasing, the momentum parameter ensures that it continues to increase. This inertia-like effect of the momentum parameter often results in larger weight value changes and for that reason the learning rate parameter often has a smaller value when used in conjunction with a momentum parameter.

2.2 History of Neural Networks

The basic building block of a neural network, the neuron, was first posited in the 1950s [15] as a perceptron. The perceptron was missing the networked aspect of neural networks, in other words, being connected to other neurons. Work was eventually done to combine multiple perceptrons in a networked fashion and this provided a significant increase in computational power over a single perceptron.

Networked perceptrons still had the significant problem of only having a single layer, namely the output layer [16]. The introduction of at first only a single hidden layer and then later on multiple hidden layers between the input and output layer resulted in neural network research experiencing a resurgence of interest from the computational intelligence community.

The last decade in particular has seen a massive rise in neural network research. The reason for the recent boom in neural network study is often attributed to increased computational power as well as larger and more complex datasets being available to researchers [17]. The latter reason - large complex datasets - can be seen as having created a need for a machine learning method that can effectively and accurately process complex data. Deep neural networks were developed as a solution to this need.

Neural networks can be classified as being either shallow or deep. The key to differentiating a shallow neural network from a deep-neural network lies in the number of hidden layers it contains. While there is no standard threshold, the general consensus amongst researchers is that any neural network with more than two hidden layers is considered a deep neural network [18]. Figure 2.2 shows an example of a shallow neural network that has only a single hidden layer, Figure 2.3 shows an example of a deep neural network that has three hidden layers.

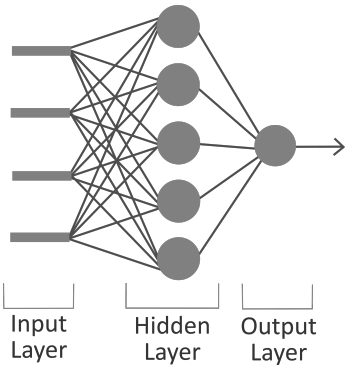

Fig. 2.2. A shallow neural network architecture

It would be incorrect to say that deep neural networks are simply better than shallow neural networks, as shallow neural networks do have a place in neural network study [19] [20]. The reason for choosing a shallow neural network over a deeper neural network for certain problems relates to the additional hidden layers required by a deep neural network architecture adding considerable complexity. Additional hidden layers means that there are more weights that need to be adjusted and more parameters to be tuned, which altogether requires more computing resources and leads to longer training times [9]. The added complex-

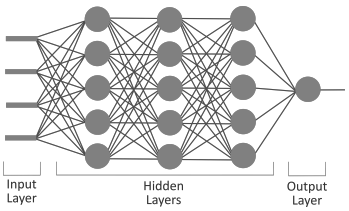

Fig. 2.3. A deep neural network architecture

ity of a deep neural network does however pay off in that it has been shown that as hidden layers are added and the complexity of computation increases, neural networks can be employed to solve more complex problems. Where a shallow neural network might only be capable of doing rudimentary classification, a deep neural network can extract features and learn relationships from data such as videos or high-definition images. Simply put deep neural networks are able to solve more complex problems far better than shallow neural networks [21]. The majority of the problem domains to which neural networks are being applied nowadays do tend to be more complex, for example natural language processing (NLP) and image processing, and in these instances deep neural networks have been shown to outperform shallow neural networks [22] [23].

Although the neural networks illustrated by Figure 2.2 and Figure 2.3 differ in the number of layers they have, they can both be used as an example of a multi-layer perceptron (MLP) architecture. The MLP architecture is characterized by having each neuron in a given layer being fully connected to all neurons in the preceding and subsequent layer; this is known as being fully connected. If the layers in a neural network architecture are not fully-connected we refer to this as a feed forward neural network architecture, an example of a feed-forward neural network architecture is given in Figure 2.4.

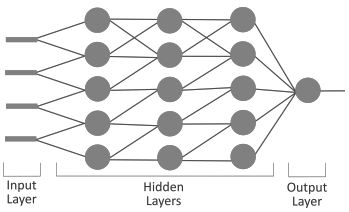

Fig. 2.4. Feed-forward neural network architecture

The difference between the MLP and the feed forward neural network architecture is that of propagation. Propagation within a neural network can occur in two directions: forward propagation and backpropagation. Forward propagation refers to the process where output values are calculated using the input received

by the neural network in combination with its weights. Backward propagation refers to the process of updating the weight values of the neural network using the loss values in conjunction with the optimizer. The feed forward neural network only makes use of forward propagation whereas the MLP makes use of both forward propagation as well as the backprop algorithm. The absence of backward propagation in feed forward neural networks implies that the weight values are kept static, which reduces the complexity of the problems that a feed forward neural network can be applied to.

Both MLP and feed forward neural networks are basic neural network architectures and the current state of deep neural network research has advanced beyond only adding hidden layers. A plethora of deep neural network architectures have been developed that attempt to address the different weaknesses presented by more basic neural network architectures. These more complex architectures are further discussed in Section 2.3, Section 2.4, and Section 2.5 respectively.

2.3 Convolution Neural Network

The architecture that led to one of the largest advances in deep neural network research, is the convolutional neural network (CNN). Many state-of-the-art deep neural networks incorporate some elements of convolutional neural networks.

CNNs build on the basic deep neural network architecture by adding the following additional layers: a convolutional layer, a rectified linear unit (ReLU) layer, a pooling layer and a fully connected layer. Figure 2.5 illustrates a deep convolutional neural network. The convolutional layer and ReLU layer are often combined into one, but for ease of explanation and illustration they are separated. The neurons in the convolutional layer are not fully connected to the preceding layer, rather a neuron in the convolutional layer is connected to neighbouring neurons in the same layer.

The convolutional layer receives input data either directly from the input layer or from preceding layers; the input data is referred to as a feature map. The convolutional layer processes only a portion of the feature map at a time, as determined by the stride size. For example when using an image, if the stride is 8 by 8 pixels, meaning a stride consists of 8 pixels, a total of 64 pixels are processed at a time. A dot product is computed using the input and a kernel. The kernel is composed of parameters that are learned during training. The application of the kernel to the input data is done in an overlapping manner, in other words, the area of the feature map is used as input to the dot product calculation more than once with different neighbouring areas. An example of the process performed by the kernel and feature map is given in Figure 2.6.

The type of convolution being performed in Figure 2.6 is full or simple convolution. During full convolution the width and height of the kernel are the

same, the kernel passes over the image once and the output of the convolution process is a single value. The other kind of convolution that can be performed is separable convolution. Separable convolution can be performed in one of two dimensions: space or depth. The first, spatially separable convolution, refers to the width and height of the kernel and separates the kernel (and subsequently the image) on one of these two measures. For example, if we spatially separate the 2x2 kernel in Figure 2.6 we would have two kernels, one that is 2x1 and another that is 1x2. The result of the 2x1 and 1x2 kernel when applied in succession is identical to the 2x2 kernel.

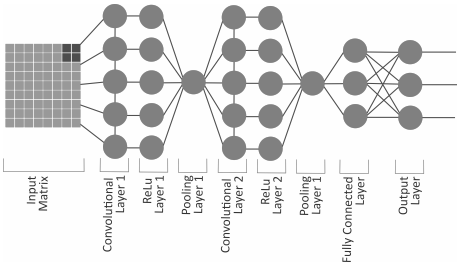

Fig. 2.5. A CNN with the convolutional layer, ReLU layer and pooling layer repeated twice.

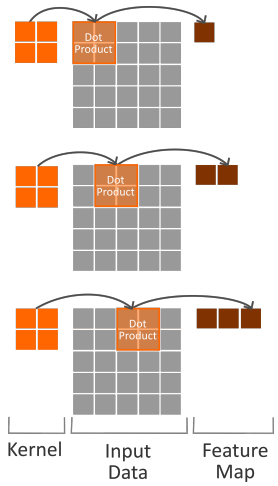

Fig. 2.6. Convolution using a 2x2 sized kernel on a 5x5 feature map with a stride of 1

The second type of separable convolution is depth-wise separable convolution, where depth refers to the number of channels. Digital image representation

is further discussed in Chapter 4 which provides more clarity regarding channels in images. For the sake of this discussion it is enough to know that each individual element of the input data consists of a tuple of values, not just a single value, and each element of the tuple is a different channel. Depthwise separable convolution uses the same number of kernels as channels in the image. Assuming a color image with three channels and using the same example kernel from Figure 2.6, depthwise separable convolution performs three iterations - one on each channel - using a 2x1 shaped kernel. The three iterations each produce a separate output, in order to recombine these three separate outputs pointwise convolution is used. The kernel for pointwise convolution is always 1x1 (regardless of the number of channels or previous kernel size used) and operates across the outputs from all channels, to produce a single consolidated output.

The feature map from the convolutional layer is input to the ReLU layer. The ReLU layer acts as the activation function and increases the non-linearity in the feature map by outputting any negative values as zero while positive values are kept constant. The ReLU layer allows the neural network to train faster and learn more complex relationships within the input data [24]. The reason why ReLU is preferred over other activation functions such as sigmoid or *tanh* is because those activation functions tend to fall victim to behaviour known as saturation. Saturation means that the activation functions become less sensitive to small changes and more frequently output values at the end of the scale i.e 1 or 0 for *tanh* and 0 or -1 for sigmoid. Saturation is undesirable as it makes training less effective.

The ReLU layer provides input to the pooling layer where the feature map is downsized. The downsizing is done similarly to the convolutional layer where a stride is used to consider only a certain portion of the feature map at a time. In the case of the pooling layer, instead of computing a product, a function is applied that summarizes that portion. The exact function used can differ, but taking the average value or taking the maximum value is commonly used.

The output from the last pooling layer is input to the fully connected layer. The fully connected layer is functionally similar to the last output layer of a standard neural network. As implied by the name, the neurons in the fully connected layer are fully connected to the output layer [25]. A convolutional neural network can have multiple convolutional, ReLU and pooling layers. Having discussed the overall architecture of the CNN, the application of an optimizer to this architecture can be discussed.

In the introduction of this chapter it was mentioned that there are several different optimizers that can be used to train a neural network, for CNN architectures the Adam optimizer [10] as well as RMSProp [26] have been shown to perform well across different problem domains [27] [28] [29]. Adam extends the basic stochastic gradient descent optimization algorithm by having a different learn-

ing rate for each weight and allowing the learning rate to be dynamically adjusted throughout the learning process. The adjustment of the learning rate is based on two parameters that are calculated at each epoch. The first parameter is the exponential moving average of the gradient plus the moving average of the gradient squared. The second parameter is the moving average of the gradient plus the gradient. These two parameters are controlled by hyper-parameter beta one and beta two. The beta one and beta two parameters are decreased over time, which results in the learning rate getting smaller over time, this helps the Adam optimizer to focus on exploitation of the weight value search space near the end of the training process. RMSProp like Adam uses a dynamic learning rate that is set per weight. RMSProp only makes use of the first parameter described for Adam and excludes the second, meaning that only the beta one needs to have a value set for it. In RMSProp beta one is referred to as the decay rate.

The choice of loss function used for a CNN is very much dependent on the problem domain to which the CNN is being applied. Popular choices include binary cross entropy and categorical cross entropy which can be used for binary and multiclass classification problems respectively. CNNs are often employed for more complex image processing tasks such as image segmentation and object identification in which case multiple loss functions can be combined. For example calculating both the binary and sigmoid loss of the prediction. Alternatively a single loss function can be used but the loss can be calculated on multiple aspects of the neural network's prediction. For example if a CNN is identifying objects in an image as well as classifying them, the loss is calculated with respect to whether the objects are identified at all as well as the loss with respect to whether the identified objects have the correct class assigned, the popular CNN based Mask R-CNN algorithm [30] employs this strategy.

CNNs outperform non-convolutional deep neural networks specifically in the image processing domain. The reason for this is because of the feature map created by the convolutional layer which represents images far better than other basic neural networks can. The CNN also offers a measure of scale and translation invariance which makes it more robust and generalizable [31]. Image processing was the original intended use for CNNs, but recent application to the domain of text classification has shown great promise as well [32] [33]. The use of a CNN in conjunction with an unsupervised feature engineering method was proven in [33] to improve on the state of the art for several text classification tasks. Although CNNs have led to massive research advancements in the deep neural network field, the added complexity presents some difficulties. One of the biggest challenges with CNNs are that they require a large number of parameters to be tuned [34] which leads to lengthy training times.

2.4 Recurrent Neural Network

Another variation on the deep neural network architecture is a Recurrent Neural Network (RNN). The architecture of an RNN allows for information to be retained, specifically information of a sequential nature. The neurons in an RNN allow information that is currently being received to be compared with information it had previously received, before the weight update takes place. The current input as well as the past information for a given neuron have separate weight values, and both weight values are updated through backpropagation [35]. The acronym RNN can also refer to a Recursive Neural Network, however there are differences between a recurrent and recursive neural network architecture. For a Recurrent Neural Network, input is given to the input layer and the weights of the different hidden layers are applied as it passes through the specified layer. For a Recursive Neural Network input is processed hierarchically and the architecture of the neural network itself won't be fixed, it is parsed and adjusted during the training process, making backpropagation significantly more complex [36]. Recursive neural networks have not been as widely adopted and studied because of their complexity. Within this work the acronym RNN refers to a Recurrent Neural Network.

A high level illustration of an RNN and how it incorporates past inputs and current inputs is given in Figure 2.7. In Figure 2.7 we can see that the first hidden layer receives its own input as well as input from the second and third hidden layer and the output layer. A RNN neuron multiplies the current input and the previous input by their respective weights. The two input values are then passed to an activation function to be combined, *tanh* is commonly used, this is illustrated in Figure 2.8. A hidden layer in an RNN has knowledge of data that originated from processing that only occurred after it had already provided output, in other words, at a later point in time. When looking at an RNN from the perspective of time and being able to access data at different time steps, the application of RNNs to sequence sensitive data starts to make sense. An RNN is able to learn the sequences and relationships of data through backpropagation and weighting of information in a way that a standard neural network such as that depicted in Figure 2.2 would not be able to.

Similar to the CNN, optimizers that have been applied to the RNN architecture include Adam and RMSProp [37]. The AdaGrad optimizer [12] is also a popular choice considering that the RNN neural network has significantly more weights that need to be updated than a typical deep neural network. AdaGrad is another improvement upon the basic stochastic gradient descent optimizer. AdaGrad like Adam and RMSProp have a learning rate for each individual parameter, however an advantage that AdaGrad has over Adam and RMSProp is that it does not have extra parameters that have to be assigned values in order to calculate the learning rate, the learning rate is tuned automatically. AdaGrad simply keeps track of the sum of the gradient squared for a given weight and then multiplies that by the learning rate for that weight. AdaGrad is generally

quite lightweight and fast to converge, but can suffer from a problem where the learning rate becomes too small to effect significant weight changes. An extension of AdaGrad called AdaDelta attempts to address this problem by setting an upper limit on the sum of the gradient squared. In terms of the loss function, the loss function used for an RNN architecture depends on the problem domain.

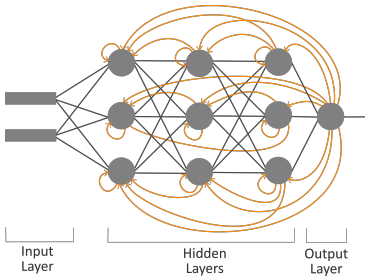

Fig. 2.7. A high level illustration of the working of an RNN

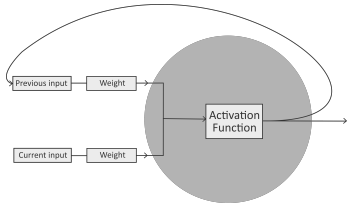

Fig. 2.8. An illustration of the basic RNN computational unit

The RNN is prone to an issue called the vanishing gradient problem [38]. The vanishing gradient problem arises from the backpropagation aspect of the RNN. Backpropagation is multiplicative, meaning that the values are not input to a prior neuron as is, instead it is multiplied by the weights of each neuron that it is backpropagated to. In the case of a hidden layer that is twelve layers deep, by the time that the value of that hidden layer's neurons is backpropagated to the first hidden layer, the value would have been multiplied eleven times and gotten so small, that it would have effectively vanished. The inverse problem, gradient explosion, where values get unreasonably large, can also affect the performance of the RNN architecture [39].

RNNs are particularly good at recognizing patterns across time which is quite different to CNN architectures which are better equipped to learn patterns across space [40]. Put differently, a CNN performs better when tasked with learning the local features from words or phrases, while a RNN is better able to learn the

long-term dependencies of texts rather than local features [41]. The combination of CNNs and RNNs is meant to exploit the positives of both while lessening the effects of the negatives. Recurrent convolutional neural networks (RCNNs) have shown competitive results in both the text classification [41] and image processing domains [42].

2.5 Long Short-Term Memory

As mentioned in the previous section, a basic RNN is prone to backpropagation becoming ineffective due to gradient vanishing/explosion. Variations on the basic RNN architecture such as long short-term memory (LSTM) address this problem. The fundamental difference between an RNN neuron and an LSTM neuron, is that the RNN has no internal cell state, whereas an LSTM neuron enforces a cell state by means of gates.

Within an LSTM neuron the current input and previous input are multiplied by their respective weights and the value pair is then passed to four different gates: a forget gate, an input gate, an input activation function and output gate. The forget gate takes the value pair and passes it to a sigmoid function. The input gate passes the value pair into a sigmoid function and the input activation function passes the value pair to a *tanh* function. The result of the input gate and input activation gate are then multiplied. The output gate takes the value pair and passes it to a sigmoid function. The all important cell state is calculated by taking the current value of the cell state and multiplying it by the output from the forget gate, the result is added to the output from the input gate and input activation gate multiplication operation. Lastly, the cell state is input to a *tanh* function and the result is multiplied by the result from the output gate.

Figure 2.9 illustrates the structure of an LSTM neuron.

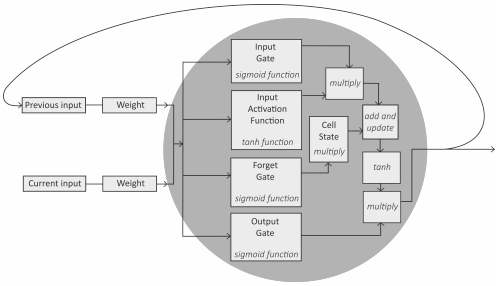

Fig. 2.9. An illustration of the basic LSTM computational unit

The structure of the LSTM neuron circumvents the vanishing gradient problem by way of the cell state which is able to retain state over a long period of

time. The gates through which all input to the neuron must pass allows for more control over how the cell state gets updated as well as what the neuron outputs, which ultimately mitigates against the vanishing gradient problem. The Adam optimizer is a popular choice for LSTM neural network architectures [43] [44] but traditional SGD has also been used quite successfully [45]. The exact loss function chosen within an LSTM architecture depends on the type of classification problem as well as the problem domain to which the architecture is being applied. The LSTM has been shown to produce competitive results in the text classification domain [46] however the added complexity of the additional gates means that the LSTM can take longer to train than a basic RNN [47].

2.6 Summary

This chapter provides basic theoretical knowledge regarding deep neural networks as well as a brief history of deep neural networks from the genesis to modern day usage. Section 2.1 provided an overview of the structure and function of neural networks and introduced some important fundamental concepts. Section 2.2 gave some more insight into where deep neural networks originated as well as why deep neural networks have become as popular as they are today. Section 2.3 discussed the structure of a convolutional neural network architecture, explaining the reasoning behind the various design choices as well as discussing some applications of this deep neural network. Section 2.4 discussed the various components and uses of a recurrent neural network, this was followed by Section 2.5 which presented the LSTM which is a specialization of the RNN. The deep neural network architectures discussed in this chapter are those that are focused on in this thesis, which is why it is crucial to have a working understanding of them.

CHAPTER 3

Text processing with deep neural networks

This chapter focuses on the application of deep neural networks to text processing. Basic text processing concepts are discussed and the two relevant problem domains this study uses, sentiment analysis and spam detection are introduced. Section 3.1 introduces the deep neural network pipeline for text processing. Section 3.2 discusses the first stage of the deep neural network pipeline which is the preprocessing stage. Section 3.3 moves on to explain the feature engineering stage of the deep neural network pipeline, which is the second stage. Section 3.4 presents the classification stage by analysing some popular deep neural network architectures for sentiment analysis and spam detection. Section 3.5 presents some examples of deep neural network pipelines for text processing from the literature. The summary in Section 3.6 concludes this chapter.

3.1 Introduction to text processing

Text processing is defined as the task of assigning individual text-based data instances within a collection of data to a given class (where a single instance can refer to anything from a single word to an entire document) on the basis of one or more features extracted from that instance [48]. Text processing is often used as an umbrella term to refer to several more specialized problems. In this research two specialized text processing problems are considered: sentiment analysis and spam detection.

The first problem, sentiment analysis, extracts features within texts with the aim of determining the opinion that the author holds of a specific entity. An example of sentiment analysis is knowing which way public opinion regarding a political candidate sways on a social media network like Twitter. Sentiment analysis was first described and attempted in 1931 [49] by hand. Sentiment analysis continued to evolve to make use of statistical methods [50] and machine learning techniques [51]. Presently machine learning has become one of the dominant means of doing sentiment analysis [52], this is further discussed in more detail in Section 3.4 below. The classes that are used in sentiment analysis problems can be either binary (i.e. a textual data instance is determined to have either positive or negative sentiment [53]) or multi-class [54] (i.e. positive, neutral or negative sentiment). Sentiment analysis can also use a continuous range of values between -1 and 1 where -1.0 represents a very negative opinion and 1.0 represents a very positive opinion [55]. The term “sentiment analysis” is often used interchangeably with the term “opinion mining” and generally the two are understood to mean the same [52], this thesis uses the term “sentiment analysis” throughout.

The second task, spam detection, differentiates between genuine benevolent texts and unsolicited, unwanted or malicious texts. The need for preventing spam first arose in the 1990s with the widespread adoption and use of emails and short-messaging service (SMS) communications [56]. The advent of Web 3.0 has meant that there are many more platforms that serve as potential targets, for example: Twitter, Facebook and Amazon [56]. Spam detection should take into consideration the platform from which the text originates. A spam email looks very different to that of a spam comment left on a Youtube video. The former most likely has more content than the latter, additionally an email comes with very different metadata to that of a comment. Spam detection can be performed using a variety of techniques ranging from simple blacklisting of words or senders to complex machine learning techniques [57] this is discussed further in Section 3.4 below. Spam detection is commonly defined as a binary classification task where a textual data instance is either classified as spam or ham [58]. Ham is the class given to benevolent and genuine texts. An example of binary spam detection is seen in email clients, where genuine emails arrive in the user's inbox (these are the "ham" emails) and the spam emails are immediately discarded and put into a separate folder. Spam detection can also be a multiclass classification problem [59] where data instances are classified as either "hard spam", "easy spam", "hard ham" or "easy ham". Where "easy" refers to a textual data instance that can easily be classified and "hard" refers to data instances that are more difficult to classify.

Text processing can be abstracted to consist of three high level stages which is referred to as the deep neural network pipeline. These stages in order are preprocessing, feature engineering and classification [60]. The preprocessing stage consists of cleaning and preparing the raw texts for feature engineering. The feature engineering stage refers to the process of selecting and extracting features from the preprocessed texts that can be input to the technique used in the subsequent step. The last stage is the classification stage where the feature representation produced by the feature engineering stage is used to classify the texts by means of a deep neural network. Figure 3.1 provides a visualization of the deep neural network pipeline for text processing.

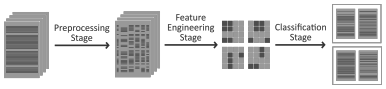

Fig. 3.1. The deep neural network pipeline for text processing

Each stage in the deep neural network pipeline can consist of one or more specific techniques. Section 3.2, 3.3 and 3.4 discusses the different techniques that have been used for the different stages in the deep neural network pipeline.

3.2 Deep neural network pipeline - preprocessing stage techniques

The preprocessing stage is the first stage in the deep neural network pipeline. Preprocessing means getting rid of any superfluous information in the text and removing any noise that could lessen the efficacy of the stages that follow. Preprocessing techniques should be applied on the basis of whether the text does indeed require that preprocessing technique to be applied.

The origin of the texts used for sentiment analysis are places like social media or review websites meaning these texts can be of low writing quality. The texts may be repetitive, overly wordy, use slang terms or have spelling errors, all of which adds noise. For this reason, the primary purpose of the preprocessing stage in sentiment analysis is to clean the texts and remove noise. For spam detection the preprocessing stage can serve a slightly different purpose than for sentiment analysis. In spam detection textual anomalies such as spelling errors or poor grammar serve as indicators that a text might be spam. Additionally those creating the spam texts might intentionally misspell words to bypass existing spam detection methods [61]. For example spelling "credit card" as "(redit (ard" instead, in this scenario correcting the spelling of all words would be a poor choice of the preprocessing method. Context-sensitive spelling correction methods are being researched [61] but given their infancy, fall outside of the scope of this work.

Preprocessing techniques that have seen frequent use are discussed below:

- Applying spelling correction to the text according to a dictionary that matches the language of the text [62]. If a word occurs multiple times in a text but it is spelled differently in a few instances, it can be seen as a different word entirely. The meaning of a word does not change when the spelling is incorrect, which means that misspelled words add unnecessary noise. Spelling correction cannot be applied without consideration for the text's origin as misspelled words might be misspelled on purpose because they are a colloquialism for example "nah" instead of "no".
- Stemming is used to transform words to their specific stem by cutting off the extra characters from the end of the word. For example, the words "games", "gamer" and "gamest" are all transformed to the single stem "game". Stemming is a good preprocessing technique to use for texts that are wordy. Stemming removes the noise of having many words that mean the same thing by increasing the frequency of the occurrence of a single word. Stemming is used in both sentiment analysis [63] [64] as well as spam detection [65].
- Removal of URLs and URI-type objects. URLs link to other places on the web whereas URIs are a reference to a specific piece of media like an image or journal article. An example of a URL is 'http://example.com' while an example of a URI is 'urn:oasis:names:specification:docbook:dtd:xml:4.1.2'.

This preprocessing technique is often used in texts that originate from social media like Twitter or Facebook. It is important to note that if the URLs or URIs contain information that is crucial to classification, then they cannot be removed, if they are not crucial they add noise and should rather be removed. Both sentiment analysis [64] and spam detection [66] make use of datasets that include texts with embedded URL and URI strings which require removal during preprocessing.

- Conversion of hashtags into normal words. Hashtags are a single string of characters preceded by a hash ('#'). Hashtags are used to summarize the content of a text, to identify a text as being related to a specific topic, or to add additional content to the text [67]. The majority of the time hashtags contain important and meaningful content. Because of the leading pound sign, hashtags are seen as unique words. For example the hashtag '#coronavirus' and the word 'coronavirus' are not seen as equal within the text, this can add noise which is why the leading hash is removed. It is more common for sentiment analysis studies to make use of hashtag conversion as a preprocessing technique [63] [68] [69].
- Removal of punctuation is often done to combine multiple sentences into a single sentence [70]. The reason for removing punctuation only becomes apparent at the feature engineering stage, when the frequency of word occurrences within sentences is used to encode feature representations. Both sentiment analysis [63] [64] and spam detection [66] have employed this preprocessing technique.
- Removal of stop words [70]. Stop words are words that occur often in texts but are neutral in terms of the meaning they add to a text. Stop words can include words like: "the", "a", "was" and so forth. Stop words are essential for humans to read texts but do not contribute to the overall meaning. Removing stop words gets rid of noise and intensifies words that are meaningful. Stop word removal should be used carefully as it can potentially alter the meaning of a text. For example if stop words are removed from the sentence: "I am not a fan of this product" the resulting sentence is "I am fan product" which is nonsensical. Stop word removal is used for both sentiment analysis [63] [64] and spam detection [65] [71].
- Lemmatization converts similar words to a single word that encompasses the same concept. The difference between stemming and lemmatization is that stemming converts similar words to their stem and lemmatization converts words to their lemma. The lemma of a word takes the morphological origin of a word into account [72]. Consider the words "games", "gamer" and "gamest" which all have the stem "game". If we add the word "gamification" to the list of words, when stemming is applied to "gamification" it is transformed to the stem "gamify". However if lemmatization is applied, all four words are converted to the lemma "game". Sentiment analysis [63] [64] and spam detection [65] make use of this preprocessing technique.
- Conversion of text to lowercase. This preprocessing technique is used to decrease variance within a dataset. Sentiment analysis [68] [69] as well as spam detection [66] [71] studies have used this preprocessing technique.

The preprocessing stage carries significant importance as it has been proven that the techniques used at this stage have an impact on the stages that follow [73]. The preprocessing stage is generally made up of several techniques which get applied to the text one after the other. The order in which the techniques are applied is as important as the techniques themselves [73].

3.3 Deep neural network pipeline - feature engineering stage techniques

The processed texts from the preprocessing stage are input to the feature engineering stage and feature engineering is then performed. Feature engineering is the process of encoding a text into a feature representation by selecting, combining, removing and transforming certain features within the text. Feature engineering makes use of two processes: feature selection and feature extraction. Feature selection selects and discards already present features within the text whereas feature extraction combines and transforms individual features to create entirely new features [74]. A single word can be considered a feature, but a specific combination of words or even whole sentences may also be considered features.

Feature engineering produces a feature representation which is input to a classifier. There are multiple reasons why the feature representation is given to the classifier instead of the raw texts. Firstly, feature engineering is used to ensure that the texts are in the format that the classifier expects it to be, for example when using a convolutional neural network we need to ensure that the input matrix matches the size expected by the input layer and kernel. Feature engineering can also be used to downsize the text that is being worked with [74], when using a smaller feature representation, it means that the deep neural network that the feature representation is input to trains faster than when using a larger feature representation. The last reason for performing feature engineering is to extract the most prominent and important features of the text. If the correct features of a text are transferred to the feature representation, it results in an improvement over simply using the text as is. When using the incorrect feature engineering technique, the incorrect features are extracted from the text (or correct features are missed) leading to poor performance of the classifier [75]. When applying a well chosen feature engineering technique it has been shown to provide a significant boost to deep neural networks that use the produced feature representation both in the case of sentiment analysis [76] as well as spam detection [77] [78].

The simplest feature engineering method is one hot encoding, used for both sentiment analysis [79] and spam detection [66]. One hot encoding creates a vector representation for each word within the text. The vector representation has elements that are either 0 or 1 and each position in the vector corresponds to a position within the text. If the word for which we are creating the vector appears in a specific position in the text, the same position in the vector has a

1. Conversely if the word for which we are creating the vector does not appear in a specific position within the text, the same position in the vector has a 0. A vector representation is created for each and every word within the text, including punctuation if present. The resulting collection of vectors is the feature representation. An illustration of the one hot encoding of the sentence: "I like learning about deep learning" is given in Figure 3.2. In Figure 3.2 the rows represent the set of all words that occur in the text, the columns each represent a word in the text as a binary string vector.

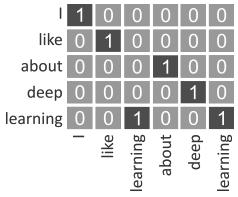

Fig. 3.2. Example of a sentence that is one-hot encoded.

One hot encoding performs feature selection and considers each word to be a feature. The recent trend within feature engineering is to make use of more complex neural network based methods such as Word2Vec [80] and FastText [81]. Word2Vec is discussed first, as FastText is an extension of the Word2Vec technique. Unlike one hot encoding, Word2Vec does not simply encode the presence and absence of words within a text. Word2Vec uses cosine similarity to group similar words together, thus a word vector represents a collection of words that are most similar to the word for which we are creating the vector. The collection of similar words is called the context. Each unique word in the text has a word vector generated for it. Word2Vec extracts a feature representation that models the relationships between the words within the text. Word2Vec makes use of a two layer neural network and has two variations in the architecture, the first being the continuous bag of words (CBOW) architecture and the second the skipgram architecture. In the CBOW architecture the neural network is given a context from the text as a whole and predicts the single word that would fit that context. In the skipgram architecture the neural network is given a single word and predicts the context within the text as a whole that the single word would occur in. In both the CBOW and skipgram architectures if the prediction is incorrect the vector for the single word is adjusted. The choice of whether to use CBOW or skipgram depends on the text to which the technique is being applied.

FastText extends the Word2Vec model by encoding the individual words into their n-gram representation before training the neural network. An n-gram is created by sampling a number of contiguous characters from a word. The number of characters to be sampled is greater than one and smaller than or equal to

n. The samples are unique, for example if n is set to 3, the n-gram for the word 'feature' contains: 'fe', 'ea', 'fea', 'at', 'eat', 'tu', 'atu', 'ur', 'tur', 're' and 'ure'. The use of n-grams for the representation allows longer words to be broken up into their prefixes and suffixes and shorter words to be better explicated. FastText supports CBOW and skipgram.

Neural network based feature engineering methods have resulted in the significant advancement of feature engineering research, however they are expensive to use both in terms of computing resources as well as time. Alternative feature engineering techniques that are less time and resource intensive are term frequency - inverse document frequency (TF-IDF) [82] and Global Vectors (GLoVe) [83]. TF-IDF calculates how frequently a word occurs in a single textual instance (term frequency) and then multiplies that value by the inverse of how frequently that word occurs across multiple texts (inverse document frequency). TF-IDF is used for both sentiment analysis and spam detection with competitive results being produced [84]. TF-IDF is simple to compute but it is unable to capture word context in the way Word2Vec and FastText do. The GloVe feature engineering technique like TF-IDF makes use of word frequency vectors for each word. The GloVe technique calculates the dot product of two word vectors and compares it to the log of the amount of times the two words occur near one another. The concept of how "near" two words are to one another is controlled by a hyper-parameter. For example if the hyperparameter is set to be three, the two words for which we are comparing word vectors, have to be within three words of one another. The two word vectors are adjusted based on how different the dot product is from the log result.

3.4 Deep neural network pipeline - classification stage techniques

The feature representation resulting from the feature engineering stage is input to the classification stage. Many different techniques can be applied at the classification stage, this work focuses on text processing within the context of deep neural networks and as such the discussion is limited to the use of different deep neural network techniques.

In Chapter 2 four distinct deep neural network models were discussed: a basic deep neural network (DNN), a convolutional neural network (CNN), a recurrent neural network (RNN) as well as a long short-term memory (LSTM) network. These four architectures are discussed in turn for both sentiment analysis and spam detection. The inclusion of these deep neural networks is supported by comparative studies that list the aforementioned four deep neural network architectures for sentiment analysis [85] [86] and spam detection [87][88] respectively. The explicit focus on deep neural networks is further supported by studies done on different text processing problems showing deep neural network techniques to be competitive with and in some cases superior to other supervised and un-

supervised techniques [89] [90].

The basic DNN architecture has seen recent use in both sentiment analysis [63] [91] and spam detection [66] studies. The reason for the inclusion of the basic DNN despite the existence of more complex deep neural network architectures is due to the fact that it is faster to train and requires far less hyper-parameter tuning than the other neural network architectures listed. Some studies have shown the basic DNN to outperform more complex architectures on text processing datasets that have a high level of variance [92].

A more popular architecture for both sentiment analysis [93] and spam detection [65] is a CNN. The CNN has been shown to exhibit longer training times than a DNN for sentiment analysis tasks but this is to the benefit of better performance, in other words the CNN obtains more accurate results than the DNN. The CNN is the state of the art for certain sentiment analysis [86] and spam detection [94] datasets over other deep neural network architectures such as RNN. The CNN is thought to perform well on text based data as a result of the feature extraction process inherent to the architecture. The process of convolution in a CNN captures n-grams in the case of text based data and this extra level of feature engineering allows the CNN to make use of features that might not have been extracted or apparent in the feature engineering stage. The CNN does have limitations, one of which is that although it excels at extracting features, it is unable to accurately model long-term dependencies [95]. In other words, a CNN cannot identify temporal relationships between texts because it has no way of recalling data seen in a past epoch at a future training epoch. For datasets that do not rely on such long term dependencies for classification, the CNN is a good architecture to use, however if relationship modelling is required, the memory components of both the RNN and LSTM architectures are better suited.

The application of a basic RNN is popular for both sentiment analysis [64] and spam detection [96]. The popularity of RNN networks can be owed to the fact that they are able to model long term dependencies. RNN has outperformed both CNN and DNN architectures for certain sentiment analysis [97] and spam detection [98] datasets. RNN architectures can present the problem of significantly higher training times than both DNN and CNN architectures. The LSTM architecture which is an extension of the RNN architecture has also been widely applied to both sentiment analysis [63] and spam detection [94] tasks. The LSTM architecture is also able to model relationships between words and texts that are not in the immediate context of one another. Recent comparative studies specifically in the sentiment analysis domain have concluded that LSTM architectures seem to be outperforming other deep neural network architectures [99]. The LSTM architecture requires significantly longer training times than both the DNN and CNN architectures.

From the above discussion it is clear that these four deep neural network ar-

chitectures have seen frequent use and different aspects of the four deep neural network architectures make them well suited to different kinds of tasks and datasets. Comparative studies have explicitly shown that the different deep neural network architectures have both strengths and weaknesses when applied to the classification of sentiment [100] as well as spam detection [87].

3.5 Deep neural network pipelines for text processing in the literature

Deep neural network pipelines have been applied to text processing for both sentiment analysis and spam detection in the literature. This section discusses a few of those instances. The pipelines discussed in this section have been selected for inclusion since the datasets that they were applied to, are the same datasets that this thesis makes use of for experimentation later on.

The work in [101] attempted to predict movie ratings by making use of a deep neural network pipeline. The preprocessing stage consisted of case conversion, tokenisation, stop word removal, spelling correction, stemming and lemmatization. The feature engineering stage using the Word2Vec technique in a skipgram configuration and the resulting feature representation was then input to an LSTM. The deep neural network pipeline described was applied to the ACL IMDB movie reviews dataset [102]. The work in [103] focused on increasing the performance of a LSTM in a deep neural network pipeline for sentiment analysis by trialling different feature engineering methods. This work showed that a deep neural network pipeline with no preprocessing techniques and a basic word embedding as the feature engineering technique provided optimal performance for the Amazon product reviews dataset [104]. The work in [105] aimed to develop a deep neural network pipeline that would be able to predict the sentiment of tweets pertaining to the 2020 Coronavirus pandemic. The deep neural network pipeline contained a preprocessing stage with unspecified techniques and a feature engineering stage that used one-hot encoding. The neural network component was a CNN. The deep neural network pipeline as a whole was applied to the Coronavirus tweets dataset [55]. The comparative study in [106] sought to determine whether an LSTM or CNN would perform most effectively in a deep neural network pipeline for sentiment analysis, specifically for tweets. The preprocessing stage made use of stop word removal, lemmatization and stemming. The feature engineering stage was not specified. The best performing deep neural network was a LSTM as applied to the Sentiment 140 dataset [107]. The study in [108] constructed a deep neural network pipeline to perform sentiment analysis on a combination of restaurant, hotel and business reviews. The preprocessing stage consisted of case conversion, punctuation removal, tokenisation, lemmatisation and stop word removal. The feature engineering technique chosen was GloVe and the resulting feature representation was given to a CNN for classification. The Yelp reviews dataset [109] was used to test the deep neural network pipeline performance.

The work done in [110] focused on spam detection for emails, specifically for datasets where the number of emails classified as ham (benevolent) and the number of emails classified as spam (malevolent) is balanced. A deep neural network pipeline is constructed where the preprocessing stage consists of case conversion, tokenization, stop word removal and stemming. The feature engineering stage made use of the TF-IDF technique and the neural network architecture chosen was a CNN. The Enron emails dataset [111] was used as it contains a balance of spam and ham emails. The work done in [112] similarly focused on spam detection but for shorter texts, in other words SMS messages. A deep neural network pipeline was again constructed. The author of [112] specifies that a preprocessing stage was included but does not mention the techniques used, the feature engineering stage used basic word embedding and the chosen deep neural network architecture was an RNN. The aforementioned deep neural network pipeline was designed for the SMS Spam dataset [113]. The work in [114] sought to do a general study of information filtering which includes spam detection. A deep neural network pipeline was used as the solution. No techniques were used during the preprocessing stage, the feature engineering stage used the GloVe technique and an LSTM was chosen as the deep neural network architecture. The Spam Asssassin dataset [115] was the dataset for which the pipeline in [114] was designed. The study in [66] constructed a general spam detection system that incorporates a deep neural network pipeline. The preprocessing stage was quite lengthy consisting of case conversion, removal of URI like strings, conversion to ASCII, removal of punctuation, removal of date and time strings, removal of whitespace, removal of stop words, stemming, lemmatization, spelling correction and duplicate word removal. The feature engineering stage used a custom unspecified word embedding technique, the deep neural network architecture chosen was a basic DNN. The YouTube comments spam dataset [116] was the dataset used in the construction of the aforementioned deep neural network pipeline.

The performance of the pipelines discussed above will be compared to the performance of the deep neural pipelines that are created by means of automated design in Chapter 8.

3.6 Summary

This chapter analysed deep neural networks in the context of text processing, specifically sentiment analysis and spam detection. In Section 3.1 sentiment analysis and spam detection were defined and some basic text processing concepts were discussed. Section 3.1 also introduced the concept of a deep neural network pipeline for text processing. Section 3.2 focused on the preprocessing stage of the deep neural network pipeline. Section 3.3 focused on the feature engineering stage and its respective techniques. Section 3.4 discussed some deep neural network architectures in the context of sentiment analysis and spam detection

as part of the classification stage. Finally Section 3.5 presents some instances of the deep neural network pipeline being used for text processing.

CHAPTER 4

Image processing with deep neural networks

This chapter discusses deep neural networks as applied to the image processing domain. Section 4.1 explicates some fundamental image processing concepts, also introducing image processing subdomains and the deep neural network pipeline. Section 4.2, Section 4.3 and Section 4.4 discuss the augmentation, preprocessing and processing stages of the deep neural network pipeline respectively. Section 4.5 presents some deep neural network pipelines for image processing from the literature and Section 4.6 summarizes the chapter.

4.1 Introduction to image processing

Digital images are defined as two dimensional matrices, where each individual element of the matrix represents a single pixel. The format in which the pixel is represented depends on the type of image. A digital image can be either binary, grayscale or color. Binary images have either a 1 or 0 as an element of the matrix, where 0 indicates a white pixel and 1 indicates black. Grayscale images have an integer value in the range 0 to 255 as an element of the matrix, where 0 represents white and 255 represents black. Color images have a tuple as an element of the matrix, where the tuple's format depends on the color space. A popular color space is RGB where the tuple has three components: the red, green and blue component (hence RGB). Another popular color space is HSV, where each tuple has three components: the hue, the saturation and the value. The matrix representation is what is input to the various techniques used for image processing. Figure 4.1 illustrates examples of each one of the three image types.

Fig. 4.1. Example of a binary, grayscale and color image.

Image processing refers to the general task of taking raw image data and giving it to some technique which processes the image to produce some output. The output that results from image processing is dependent on which specific problem we are solving. The term "image processing" encompasses image classification, object detection, semantic segmentation and instance segmentation.

The aim of image classification is to assign one or more classes to a given image. The aim of object detection is to detect one or more objects in an image and draw a bounding box around the detected objects. Semantic segmentation and instance segmentation similarly to object detection detect one or more objects in an image, but instead of simply drawing a bounding box, each pixel in the image is assigned a class, in effect drawing a polygon mask over the detected objects. The difference between semantic segmentation and instance segmentation is that semantic segmentation does not differentiate between individual object instances; it simply labels pixels as one of the predetermined classes, whereas instance segmentation is instance aware. Instance segmentation assigns a class label to pixels and additionally indicates which object instance a pixel belongs to. Figure 4.2 below provides a visual comparison of the above-mentioned four subdomains.

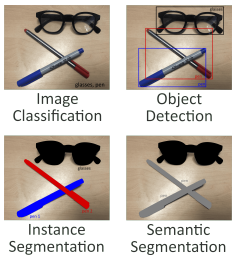

Fig. 4.2. A visual comparison of image classification, object detection, semantic segmentation and instance segmentation.

Image processing can be modelled as a deep neural network pipeline. The deep neural network pipeline for image processing consists of three stages: augmentation, preprocessing and processing. The processing stage differs depending on which subdomain of image processing the deep neural network pipeline is being applied to. If image classification is being performed it serves as the classification stage, if instance segmentation is being performed it serves as the segmentation stage. Figure 4.3 provides a visualization of the deep neural network pipeline for image processing.

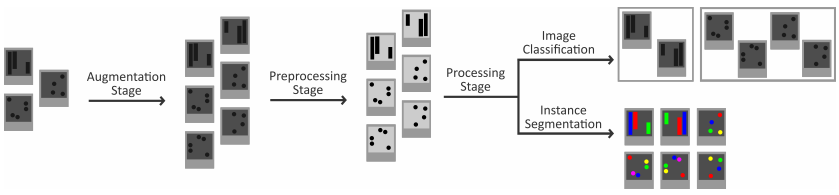

Fig. 4.3. The deep neural network pipeline as applied to image processing

Sections 4.2, 4.3 and 4.4 discuss the different techniques that have been used for the different stages in the neural network pipeline.

4.2 Deep neural network pipeline - augmentation stage techniques

The augmentation stage is the first stage in the deep neural network pipeline. An augmentation technique makes a duplicate of the original image, then makes small changes to the duplicate, producing an entirely new image we call the augmented image. Labels/annotations that have been given to the original image can be copied over to the augmented image if the augmented image is semantically similar to the original image. Augmentation can be performed in a way that allows for labels and annotations to be adjusted as the augmented image changes.

Augmentation techniques grow the dataset without needing to add more original images. A dataset of reasonable size is needed to produce adequate results when using deep neural networks [117]. The generalization ability of a deep neural network is also known to increase when given more images to learn from [118]. Generalization refers to the ability of an already trained deep neural network to deliver accurate predictions for data that was not involved in the training process. The opposite of generalization is overfitting where a deep neural network produces great results for the data used during the training process, but very poor results for other unseen data. Data augmentation attempts to increase the performance and generalization ability of a deep neural network. For image processing specifically, data augmentation has been shown to be very effective for both classification and segmentation [119]. Image augmentation is also used to solve the problem of class imbalance. Class imbalance refers to a situation where the dataset is dominated by instances of a specific class. Image augmentation can be applied only to the dataset instances of the classes that are under-represented, resulting in a more balanced dataset overall.

There are several popular data augmentation techniques that have seen frequent use, they are each listed and discussed below:

1. Vertical and/or horizontal flipping of an image [120]. This augmentation technique has been shown to be effective on benchmark datasets such as ImageNet and is computationally inexpensive.
2. Rotation of an image [120]. The number of degrees by which an image is rotated can vary, but a maximum rotation size might need to be set to prevent the augmentation from destroying essential information. For example, if we are building a classifier that can classify an image of a thumb as either being either a thumbs-up or thumbs-down, we cannot allow an image to be rotated 180 degrees as this changes the meaning of the image. In this case a maximum rotation limit of 45 degrees can be used.
3. Cropping an image to be smaller than the original image [120]. This technique requires some careful analysis of the dataset before being used, as the

amount of cropping done cannot be such that it removes valuable information from the image.

4. Changing the position of the centre point of the image while preserving image size i.e., translation [120]. Translation can be performed on either the vertical or horizontal axis and is useful especially for datasets where the object of interest is expected to be in the center of the image. For example, if we are developing an object detector that detects groceries in a self-checkout area, example images should show groceries that are nicely centered in the detection square as well as groceries that are on the edges of the detection square.
5. Changing the color space of an image [120]. Digital images that are full color commonly make use of either RGB or HSV representations. Creating an augmentation that either only uses a single color channel, or removes a color channel produces a new dataset instance that appears as an entirely "different" instance to the classifier, even though to humans the image looks the same.
6. Adding noise to an image [121] is a powerful augmentation technique which addresses one of the well known weaknesses of CNNs, which is that they tend to deliver bad results for images that are of poor quality. By including dataset instances that are of poorer quality, the classifier can learn to deal better with poor quality images during training.
7. Sharpening the image [121] by means of a kernel filter. The size of the kernel filter being used can be seen as a hyper-parameter that needs to be chosen. The larger the size of the kernel filter, the higher the degree of augmentation. Sharpening specifically could serve to enhance important areas of the image that could be vital for image classification and detection tasks.

4.3 Deep neural network pipeline - preprocessing stage techniques

The preprocessing stage follows the augmentation stage and applies preprocessing techniques to both the original and augmented images. Image preprocessing alters images in a way that improves the performance of the deep neural network. During preprocessing information in the image that is not beneficial is removed and information that is beneficial is enhanced. In other words, image quality is improved. There are many image preprocessing techniques available, popular techniques are discussed below:

1. Mean normalization [122] calculates the mean pixel value across the entire dataset, then preprocesses each image in the dataset by subtracting the mean pixel value from each pixel in the image. If images have multiple channels (ie RGB or HSV) the mean pixel value is calculated per channel. Data normalization is beneficial, since when values are scaled to lie in a certain range, it can result in a more controlled and faster training process. The effectiveness depends on the variation across the instances in the dataset. If all images

in the dataset are already quite close to their normalized values, this image preprocessing technique has a less pronounced effect.

2. The application of a gaussian blur to images [123] is done to remove unwanted noise. Each pixel in the image is set to the weighted average of the pixels surrounding it. The number of pixels considered is determined by the kernel size of the filter and the weighting of pixel values is given by a gaussian distribution. The resulting image is smoother than the original. When the gaussian filter is applied it is assumed that only unwanted noise such as small artifacts and imperfections in the image are removed, however care must be taken as the gaussian filter if used improperly can destroy features in the images.
3. Moving an image from the RGB to HSV color space [123] can be done as a preprocessing step since HSV allows for separation of the color itself from the intensity of the color, while RGB does not. Additionally, for certain datasets the RGB representation can be noisier than the HSV representation.
4. Segmenting an image using K-means clustering [123] is one of the more computationally expensive preprocessing techniques. This technique is used to isolate important foreground information from background information. K-means clustering is performed with a cluster size of 2 and once pixels belonging to the background are identified, they have their pixel value set to be black.
5. Enhancing the contrast of images can be applied to both grayscale and color images [124], in both cases the pixel values of the images are changed in order to enhance image intensity. Many contrast enhancement techniques are available including histogram equalization, contrast limited adaptive histogram equalization and more. The techniques differ in the mapping function that is used to calculate the new pixel values. Contrast enhancement is done to make features in the image that could be important to the classifier stand out more. Contrast enhancement can also enhance unwanted noise which makes it easier to identify and remove the noise.

When performing image processing more than one preprocessing technique is used. The selection of preprocessing techniques and the order of their application is determined by the dataset being used. Often the effectiveness of the preprocessing stage as a whole can only be determined after the entire deep neural network pipeline has been run.

4.4 Deep neural network pipeline - processing stage techniques

The processing stage takes the images from the preprocessing stage and uses them as input. Image classification, image detection, semantic segmentation and instance segmentation are discussed in Section 4.4.1, Section 4.4.2, Section 4.4.3 and Section 4.4.4 respectively.

4.4.1 Image Classification

Convolutional neural networks have become the state of the art technique for image classification. The relevant CNN architectures and their strengths and weaknesses are discussed.

VGG16 [125] is a popular choice for image classification, VGG refers to the Visual Geometry Group (the group of researchers who created the network) and the 16 indicates the number of layers that this architecture has. The VGG neural network makes use of a small kernel size of 3x3 and has more fully-connected layers than the majority of other CNN architectures. The use of a smaller filter size implies more weight values that need to be trained, which allows for greater precision, but at the cost of a larger architecture with a longer training time. The depth of VGG16 contributes to this architecture being slow to train.

The ResNet [126] collection of deep neural network architectures was developed in an attempt to reduce training time as well as architecture size. ResNet makes use of residual learning. Residual learning adjusts the normal structure of deep neural network layers to instead of directly mapping layers to one another, make use of a skip connection. The skip connection allows the feature map output of a previous layer to be directly fed to layers later in the model, allowing for fine tuning. Figure 4.4 illustrates how skip connections work. Residual learning combats the vanishing gradient problem and lowers the high training error that can arise in deep neural network architectures. The ResNet neural network architecture is faster to train as the use of skip connections does not add a substantial number of extra weight parameters that need to be trained.

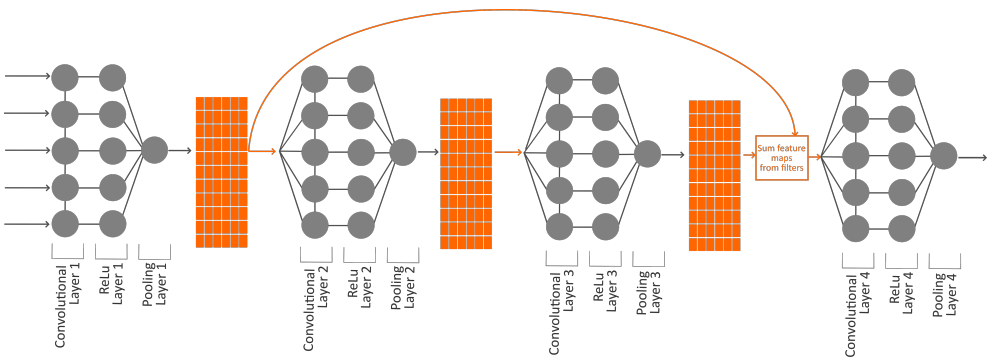

Fig. 4.4. Skip connections in a ResNet deep neural network architecture.

Improvements to the ResNet skip connection implementation were proposed in [127]. The improvements include using batch normalization at the beginning and directly thereafter the ReLU activation. The improvements were done with

the intent of preventing overfitting and resulted in a second version of the ResNet architecture.

The DenseNet [128] architecture also addresses the problem of a large number of parameters within a neural network causing slow training times. DenseNet adds direct connections between every layer in the deep neural network architecture. With DenseNet because every layer has direct access to all other layers, there is no need to rely on information (the output feature maps) being propagated, instead they can be accessed directly. The DenseNet deep neural network architecture concatenates all the output feature maps from connected layers, with the incoming feature maps; this differs from the ResNet architecture, which sums incoming feature maps. Concatenation is less computationally expensive than summation. Figure 4.5 provides an illustration of the DenseNet deep neural network architecture.

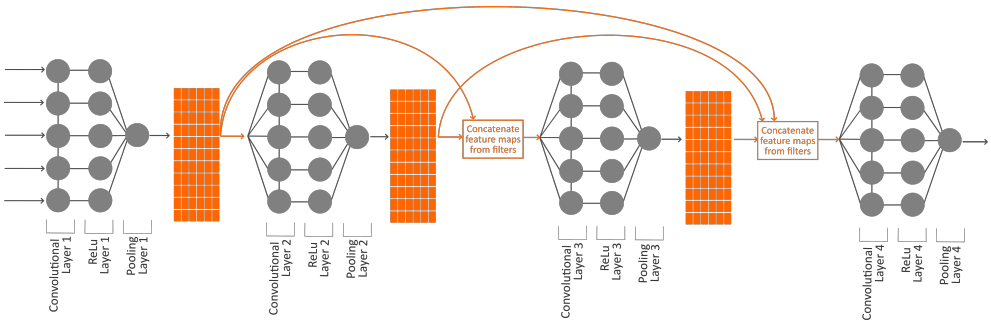

Fig. 4.5. The fully connected layers of a DenseNet deep neural network architecture.

Inception [129] is another neural network architecture that lessens training time, model complexity and size. The Inception deep neural network architecture consists of 27 layers where 9 of those layers are specialized inception layers. The inception layer processes the input feature map four times using different kernel sizes:

1. First, the inception layer feeds the input to a 1x1 convolution component; the result is recorded as Output 1.
2. Second, the input from the previous layer is fed to a 3x3 max pooling component, the output from the 3x3 max pooling component is then subsequently fed to a 1x1 convolution component and the result thereof is recorded as Output 2.
3. Third, the inception layer feeds the input to a 1x1 convolution component, the output from the 1x1 convolution component is then subsequently fed to a 3x3 convolution component and the result is recorded as Output 3.

4. Lastly, the inception layer feeds the input to a 1x1 convolution component, the output from the 1x1 convolution component is then subsequently fed to a 5x5 convolution component and the result is recorded as Output 4.

Output 1, Output 2, Output 3 and Output 4 are combined into a single output vector via concatenation as the final output from the inception layer. Figure 4.6 illustrates how the different components of the inception layer work together.

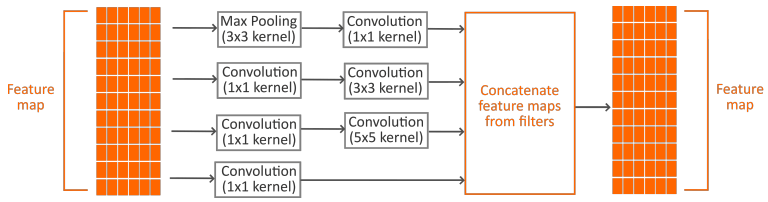

Fig. 4.6. A visualization of the inception layer and its components

The inception layer uses multiple kernel sizes to solve the problem of selecting an appropriate kernel size. The correct kernel size is dependent on the visual distribution and variance of objects in the images. A large kernel size like 5x5 works well if the objects in the image are large and equally distributed across the pixels in the image, whereas a smaller kernel size like 1x1 works well if the objects in the image are small and only cover a small percentage of the total pixels in the image. The kernel size needs to be chosen prior to starting training. Inception allows the neural network to select the most appropriate kernel size by having that output become dominant at training time via weight updates.

The Inception deep neural network architecture is currently in its third version, referred to as Inception V3 [130]. Inception V3 changes the inception layer to be more performant in terms of speed by using factorization and separable convolution. The convolution components in Inception V1 that made use of larger kernel sizes (i.e., 3x3 and 5x5) are decomposed into multiple smaller convolutional components to decrease computational complexity and training time. In Inception V3 the inception layer still serves the same purpose as in Inception V1. There are three new variations of the inception layer structure that came about as a result of the Inception V3 improvements, these are listed in Figure 4.7. The use of an RMSProp optimizer and the addition of a regularization component also form part of the V3 updates to the Inception architecture.

A combination of the Inception and ResNet architectures named Inception-ResNet [131] adds a skip connection to the inception layer, which necessitates some rearranging of the inception layer. A final 1x1 size convolution component is added which allows all the incoming feature maps to be equal in size to the incoming feature map from the skip connection. The InceptionResnet inception layer is illustrated in Figure 4.8 The combination InceptionResNet has

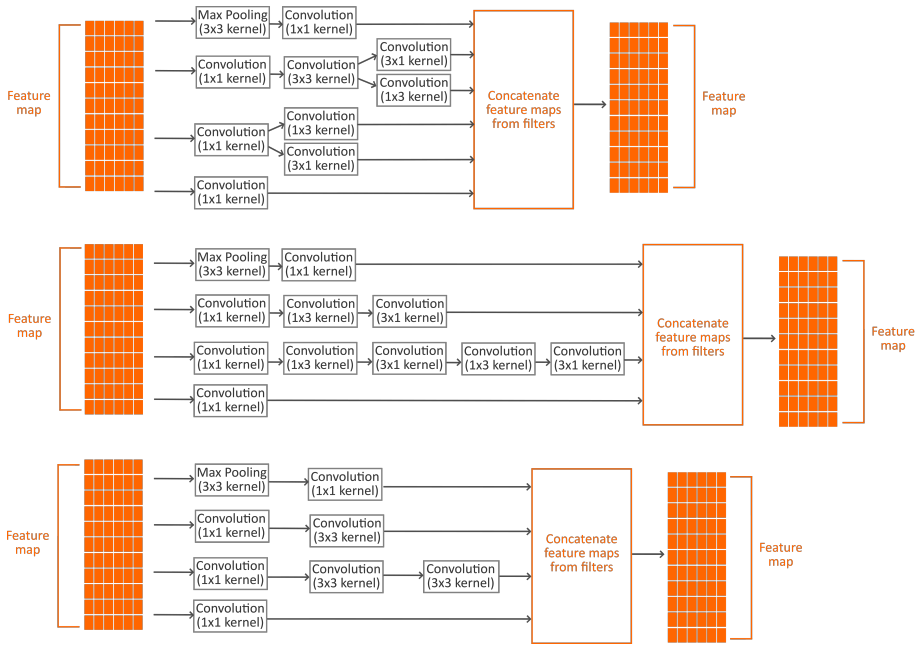

Fig. 4.7. The three updated inception layer structures as per Inception V3.

been shown to reach higher accuracies at earlier epochs than both ResNet and Inception respectively.

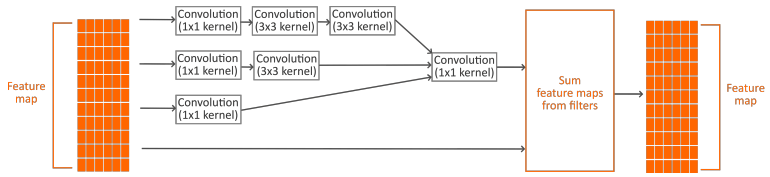

Fig. 4.8. The inception layer of the Inception ResNet architecture.

In the Inception architecture a depthwise separable convolution is performed over the feature map as a whole, the neural network known as Xception ("extreme inception") [132] partitions the feature map into its respective channels before performing a depthwise separable convolution on the individual channels. To recombine the channels, a pointwise convolution is used. The process of performing a depthwise separable convolution on different channels is illustrated in Figure 4.9

ResNet and Inception are more lightweight than architectures like VGG, but they still fall short in cases where computational resources are constrained such

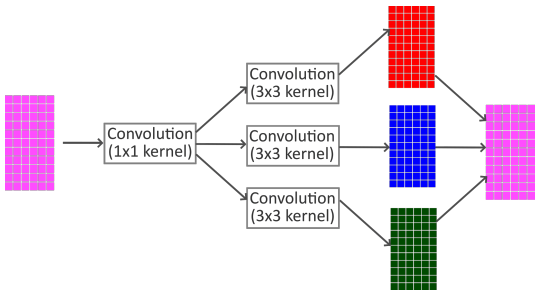

Fig. 4.9. Separate processing of image channels by Xception.

as on handheld devices. The MobileNet [133] deep neural network architecture was developed with resource constrained environments in mind. MobileNet reduces computational complexity by not making use of depthwise separable convolution similar to Xception. The difference between MobileNet and Xception lies in the fact that MobileNet is smaller than Xception and makes use of the ReLU6 activation function, instead of ReLU. ReLU6 is optimized for computational tasks that do not require extreme precision, but favours low computational complexity and speed.

The MobileNet deep neural network architecture as a whole favours speed over accuracy. The first version of the MobileNet has a convolutional block that looks similar to Xception, with the difference lying in the pointwise convolution (1x1 convolution) happening at the end as opposed to the start. The second version of MobileNet adds pointwise convolution to the start and end of the convolutional block as well as a skip connection. A comparison between the first and second version of the MobileNet deep neural network architecture is given in Figure 4.10

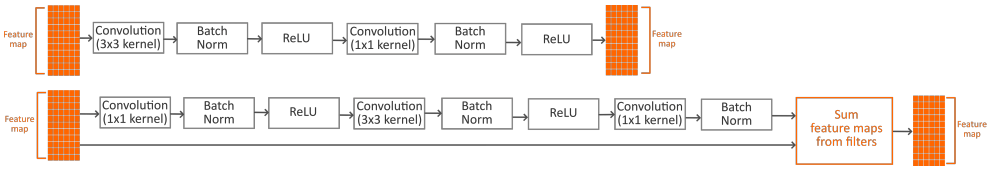

Fig. 4.10. Difference in architecture between the MobileNet version 1 (the first architecture shown in the diagram above) and MobileNet version 2 (the second architecture shown in the diagram above).

4.4.2 Image Detection

One of the most well known image detection techniques is Regions with CNN features (R-CNN). The basic R-CNN algorithm takes an input image and segments the given image into several smaller regions. The regions are chosen according to a region proposal method; the original work by Girshick et al. [134] makes use of selective search, but other techniques such as an objectness score calculation and multi-scale combinatorial grouping can also be used. Once regions have been extracted from the image, they are resized to be 227 x 227 pixels and each region is given to a CNN consisting of five convolutional layers and two fully connected layers. The features output by the CNN are given to an SVM to be classified. If we are trying to detect a human in a photograph, the SVM will have been trained to be able to detect the presence and absence of a human. Regions that do not contain a human are classified as "Not human" and regions that contain a human are classified as "Human". The position of the human in the original image can thus be inferred.

Fast R-CNN [135] improves upon the basic R-CNN algorithm. Fast R-CNN achieved faster runtimes than the R-CNN by removing the SVM component. Fast R-CNN refactors the CNN to do the feature extraction and deliver class predictions for the regions it is given by adding two fully connected layers. The CNN is given the image and ground truth regions as input. The CNN compares the regions detected to the ground truth regions in order to calculate the loss both in terms of the class as well as the region location and size. The detected regions are as per the original R-CNN extracted using an external region proposal method which is the largest bottleneck.

Faster R-CNN [136] follows Fast R-CNN and incorporates the region proposal step into the CNN by making use of a region proposal network (RPN). The RPN is a CNN that shares layers with the rest of the Faster R-CNN. When the RPN is given an input image it produces a number of rectangular region proposals. The number of proposals produced is a hyper-parameter that can be set. Each proposal is accompanied by an objectness score.

A different image detection algorithm, You Only Look Once (YOLO), attempts to match the performance of Faster R-CNN while improving the runtime. Several versions of the YOLO algorithm exist. The first version of the YOLO algorithm [137] divides the input image into a grid where the number of quadrants is a hyper-parameter. Each pixel in the image is assigned a class, where the classes depend on the objects that are being detected and one of the classes is always "background". If a pixel is in the center of a quadrant, the class assigned to that pixel represents the class of the entire quadrant. A deep neural network architecture is needed to train YOLO. The deep neural network architecture for the first version consisted of 26 layers with no normalization layers and two fully connected layers. The loss function used is composed of four parts: The error in height and width of the object detection, the error in the coordinates of the ob-

ject detected, whether the correct class was assigned and whether the confidence of the detection exceeds the minimum confidence score. The second version of YOLO increased the number of layers in the deep neural network architecture to 30 and used anchor boxes. Anchor boxes are coordinates that specify a region wherein some object was contained, but the task of exactly detecting and classifying the object in that region still had to be done. Both the first and second versions of YOLO struggles to detect small objects if the number of quadrants in the grid is incorrect or if objects straddle across two quadrants. This problem was solved in the third version of YOLO. In YOLO V3 three grids are used where the size of the quadrants specified is small, medium and large (scaled to the input image). The deep neural network architecture was also changed to be 106 layers deep.

4.4.3 Semantic Segmentation

A common method for semantic segmentation is a Fully Convolutional Neural network (FCN) [138]. An FCN removes the fully connected component that features in most CNN architectures, eliminating the need to specify feature map input sizes. The fully connected component is replaced by a pointwise (1x1 kernel size) convolution, making the output of the neural network a feature map. For semantic segmentation the output feature map is a mask, highlighting the object of interest in the original image. In order to get the feature map to be the size of the original image, the image is resized by means of deconvolution, also referred to as up-sampling. The convolution process in the FCN architecture is called encoding and the deconvolution process is called decoding. The FCN deep neural network architecture forms the basis for U-Net [139], so called for the shape that the deep neural network architecture takes with the addition of shortcut connections (similar to skip connections) as seen in Figure 4.11

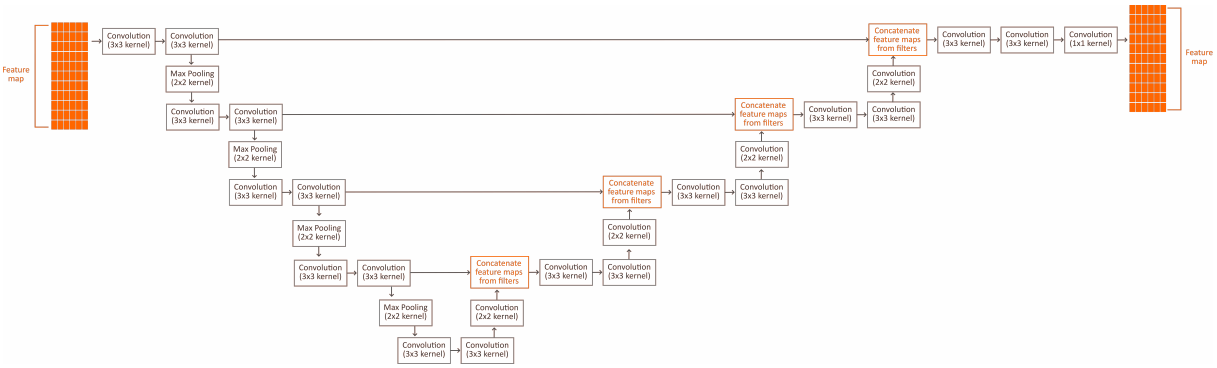

Fig. 4.11. The U-net deep neural network architecture.

U-net encodes the image by means of convolution and produces increasingly smaller feature maps, once a feature map of size 32x32 is created the decoding process begins and the feature map is resized to the size of the input image by means of deconvolution. The feature maps provided by the shortcut connections are concatenated with the feature map at the respective destination before deconvolution. The shortcut connections prevent important features from getting lost during the encoding process. U-net performs semantic segmentation but when used in conjunction with an appropriate post-processing script for certain domains this technique can be used for instance segmentation as well.

4.4.4 Instance Segmentation

The Mask R-CNN [140] deep neural network is a popular choice for instance segmentation. Faster R-CNN is the predecessor to the Mask R-CNN, the largest difference between these two deep neural network architectures being that Faster R-CNN uses bounding boxes to detect objects whereas Mask R-CNN uses a polygon mask that overlays onto the detected object at a pixel level. Figure 4.12 provides an illustration of the difference between Faster R-CNN and Mask R-CNN.

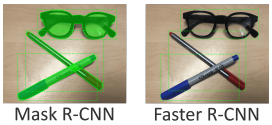

Fig. 4.12. Output comparison of Faster R-CNN and Mask R-CNN.

Faster R-CNN gets a single output from the RPN section specifying the detected bounding boxes. Mask R-CNN gets two outputs from the RPN, the first output specifies the bounding box predictions and class predictions, the second output is input to two additional convolutional layers to produce the mask detections. Mask R-CNN has been found to train quite slowly and as a result of the rather complex architecture. Mask R-CNN also has a large number of hyperparameters that need to be tuned.

Poly-YOLO [141] builds on the existing YOLO object detection algorithm in order to perform instance segmentation. Poly-YOLO adds an additional output layer that specifies a set of vertices that encircles an object of interest. Poly-YOLO outputs a set of coordinates which when traced form a polygon around the instance that has been segmented.

Annotations for instance segmentation tasks can be incredibly time consuming to produce. Various tools such as Fiji [142] exist to help ease the burden of annotation. Supervised learning techniques such as Mask R-CNN and Poly YOLO

requires training data that has masks (and the class for each mask) annotated on the images with a high degree of accuracy.

4.5 Deep neural network pipelines for image processing in the literature

Deep neural network pipelines have been applied to both the image classification and image segmentation problems. The pipelines discussed in this section have been selected for inclusion since the datasets that they were applied to, are the same datasets that this thesis makes use of during experimentation.

The research conducted in [143] performed image segmentation by means of a Mask R-CNN which had its hyper-parameters tuned using a genetic algorithm, in other words, by means of automated design. Cropping was used as a data augmentation technique and CLaHe contrast enhancement was used as a preprocessing technique. The work in [143] set a new state of the art for common rust quantification. The research in [144] compared the performance of two deep neural network pipelines for image classification, namely a ResNet-50 with rotation as augmentation technique and a VGG16 with rotation as the augmentation technique and no processing techniques. The two pipelines were applied to the Karnataka Oral Lesion Dataset [145] and UP Oral Images Dataset [146] and in both cases the ResNet-50 pipeline outperformed the VGG16 pipeline thereby becoming the state of the art deep neural network technique for oral lesion detection.

4.6 Summary

This chapter opens with an overview of image processing and some basic terminology in Section 4.1 before looking at the various techniques that comprise the three stages of the deep neural network pipeline for image processing. Section 4.2 presents the augmentation stage, Section 4.3 presents the preprocessing stage and finally Section 4.4 discusses at length the techniques that can be used for the processing stage for the four image processing subdomains. Section 4.5 briefly presents some examples of deep neural network pipeline for image processing from the literature. At the conclusion of this chapter the importance and structure of the deep neural network pipeline for image processing should be clear.

CHAPTER 5

Critical analysis and related work

5.1 Introduction

This chapter presents a critical analysis in Section 5.2 that builds on the work presented in Chapter 2, Chapter 3 and Chapter 4. The critical analysis will highlight the research recommendations as put forth by the literature and emphasize the need for research focusing on automated design, hyper-heuristics as well as transfer learning.

Section 5.3 will present an overview of automated design, first defining the field generally before discussing related work pertaining to deep neural networks. Section 5.4 will give an overview of hyper-heuristics, first defining the general concepts before moving on to discuss hyper-heuristics in the context of automated design. Section 5.5 will focus on transfer learning, providing general terminology and discussing related work in the field specifically for automated design as well as hyper-heuristics. Lastly Section 5.6 summarizes this chapter.

5.2 Critical analysis

Within the text processing domain, both sentiment analysis and spam detection necessitate continued research effort because new spamming techniques as well as new communication platforms are continually being created [147]. The fast changing nature of both spam detection and sentiment analysis means that researchers are continually needing to adjust the techniques that are used. When doing spam detection researchers are confronted by the problem of spam drift. Spam drift refers to the phenomenon where the features of spam texts gradually change over time because the techniques used by spammers evolve, this means classifiers that previously worked well are unable to correctly classify new data [147]. Investigation into hyperparameter tuning of classifiers has been suggested as a potential solution to spam drift as hyperparameter tuning is a largely ignored area of research within spam detection [147]. Spam detection is frequently done in time-sensitive scenarios where lengthy manual processes cannot be performed, for this reason the creation and execution of the spam detection pipeline has to be automated as far as possible to allow for faster and more accurate classification [147].

For the sentiment analysis problem domain, the issue of cross-domain sentiment analysis is similar to the issue of spam drift [148]. The subdomains within

sentiment analysis are determined by the origin of the text, an example of a sentiment analysis subdomain is product reviews which can originate from places like Amazon or Ebay. Other subdomains within sentiment analysis include social media data, movie reviews and hotel reviews. Moving between domains is not trivial because words that have positive connotations in one domain, might have negative connotations in another. For example the sentence "I stayed here for a long time" could have a positive connotation when reviewing a hotel, but a negative connotation in a Tweet regarding renewing your driver's license. Cross domain sentiment analysis is a non-trivial problem to solve in the sentiment analysis domain [149] [148]. From the literature it is clear that for both the spam detection and sentiment analysis the task of creating an effective text processing pipeline is non-trivial and automating this task would be beneficial both to researchers and the text processing field as a whole.

Image processing is a research area that has had significant research effort expended in the past few years. This research effort has resulted in several new state of the art deep neural network architectures being developed such as Mask R-CNN [140], U-Net [139] and YOLO [137]. The choice of a deep neural network architecture is however non-trivial and selecting an optimal architecture and hyper-parameter values that will be effective for a given task is a time-consuming problem that researchers currently solve by means of manual trial and error [150]. The work by Khan et al. [151] emphasizes that hyper-parameter optimization is a complex problem for deep convolutional neural networks particularly as they tend to have more hyper-parameters than other machine learning techniques. Khan et al. [151] recommend research be done into using a directed search process such as a hyper-heuristic for hyper-parameter optimization of deep convolutional neural networks in the image processing domain. The other stages of the deep neural network pipeline for image processing, namely the augmentation stage and the preprocessing stage, are equally as important as the chosen deep neural network architecture. The review paper in [152] lists different configurations of preprocessing and data augmentation techniques used in image segmentation pipelines and concludes that the preprocessing and augmentation techniques chosen significantly influence the overall success of the deep neural network pipeline. Pandey et al. [153] explains that there is no standard image processing pipeline that can be created because different image sets require different techniques to be applied at different stages, meaning that researchers are at pains to first analyse and then design a pipeline for each new dataset based on the features of that dataset.

From the work presented above it is clear that there is a need for automation to be applied to the process of designing deep neural network pipelines for both the text processing and image processing domains. The literature shows that automated design approaches have been applied mainly to image classification, which is why Talbi [154] suggests that less explored applications in computer vision such as segmentation as well as natural language processing should be in-

cluded in future studies. The need for automated design is further emphasized by Hutter et al. [155] and Feurer et al. [156] specifically suggesting that automated design techniques that design arbitrary-sized pipelines should be investigated and developed. Arbitrary-sized means that the number of stages, number of techniques per stage as well as the order of the techniques should not be predetermined, but rather optimized. Feurer et al. [156] recognises that the selection of the preprocessing and feature engineering methods for a deep neural network pipeline is a non-trivial problem requiring intelligent selection. Feurer et al. [156] go on to state that limiting the selection to only a single preprocessing or single feature engineering method might potentially result in a sub-par pipeline. The survey paper in [157] recommends that designing an easy-to-use and complete automated design pipeline is a promising research direction, provided that it includes decoupled functions and full training with hyper-parameter optimization.

A research recommendation that is common within automated design is that more emphasis should be given to reusability of designs as well as transfer learning. The comprehensive survey of state-of-the-art on automated design in [157] does not discuss any transfer learning methods for automated design, indicating a gap in research. The most recent survey of transfer learning in [158] explicitly suggests that transfer learning needs to be applied to a wider range of applications, including those that are more complex. The work done in [155] recognises that a problem in hyper-parameter optimization specifically is overfitting and lack of generalization. The work in [157] recommends that automated design techniques should aim for lifelong learning, in other words, techniques should be able to efficiently learn new data and remember old knowledge. The design of partially reusable sub-trees has been shown to be possible and effective in genetic programming [159]. On the whole however, the literature shows that insufficient investigation has been done into the analysis of design reusability and transfer learning in the automated design space [6].

Based on the critical analysis of the literature above, there are three areas that necessitate further research which this thesis will focus on. The first area is automated design of the deep neural network pipeline which is both a research recommendation and need that is presented multiple times across the literature that has been surveyed. The second area is hyper-heuristics, specifically the use of hyper-heuristics for automated design. Using a hyper-heuristic to perform automated design has not seen much research, but from the literature presented is a promising research focus which deserves further exploration. The third and last area is transfer learning in the context of automated design. Transfer learning has been shown to be effective in optimizing search processes in multiple domains. Transfer learning for automated design of the deep neural network pipeline by means of a hyper-heuristic has never been done and therefore warrants investigation. Section 5.3, Section 5.4 and Section 5.5 presents related work for the three aforementioned research areas: Automated design, hyper-heuristics and transfer learning.

5.3 Automated design

Automated design (AutoDes) is defined as the automation of the creation of the design of a machine learning technique or search algorithm [6]. This definition highlights the three key components of AutoDes: an algorithm, a design and an automated design technique.

Deep neural networks cannot be applied in isolation; preprocessing, feature engineering and augmentation techniques are included in a pipeline-like manner, with each serving a different purpose but all affecting the eventual performance of the deep neural network. In this thesis, a design will be created by means of automated design for an end-to-end deep neural network pipeline. Designing a deep neural network pipeline is non-trivial and when considering the No Free Lunch theorem [160], the task of designing a deep neural network pipeline will need to be performed for each new problem domain and potentially datasets. The task of automating the design of a deep neural network pipeline has received the least amount of research attention in the AutoDes space [157].

Two AutoDes techniques that produce pipeline designs are Auto-WEKA and Auto-Sklearn; both of these techniques can be used to build classification and regression pipelines. Auto-WEKA and Auto-Sklearn are limited in that they offer no neural network based techniques, only including techniques such as K-nearest neighbours and support vector machines. The first AutoDes technique that automates the design of a processing pipeline and includes a neural network technique is TPOT-NN (Tree-based Pipeline Optimization Tool) [161]. TPOT-NN uses a GP to design a processing pipeline that includes neural-network estimators that can be combined to form a basic MLP network, this however falls short of being a true deep neural network pipeline.

Several design decisions at different levels of granularity are made during the creation of a deep neural network pipeline. Design decisions are first made per stage of the pipeline i.e. which preprocessing technique or which feature engineering technique to use. Once the techniques per stage of the deep neural network pipeline have been finalized, design decisions are made regarding the hyper-parameter values for the chosen techniques. The process of making these design decisions is called hyper-parameter optimization (HPO).

A hyper-parameter is a value that dictates some aspect of the neural network architecture or training schedule. The learning rate is an example of a hyper-parameter that controls the size of the increments in which the neural network adjusts its weights. A side effect of increasing neural network complexity, as seen in deep neural network architectures, is the addition of more and more hyper-parameters. Hyper-parameters have a direct impact on the quality of the neural network's performance and therefore the selection of their values is a non-trivial process. Hyper-parameter value selection becomes an optimization problem unto itself and therefore the use of optimization algorithms to adjust neural

network parameters has seen more research attention in the AutoDes space in recent years [162]. Popular HPO techniques include random search, constructing a tree of Parzen estimators (TPE), bayesian optimization and grid search which can be used to find optimal values for real-valued, discrete, and conditional dimension hyper-parameters. Each HPO method has different advantages and disadvantages.The ideal HPO method would strike a balance between being computationally inexpensive, providing optimum values and being easy to implement [163]. The Vega [164] technique is able to design partial pipelines for image processing and allow for the optimization of the hyper-parameters in the data augmentation stage, which is applied prior to a neural network technique.

AutoDes produces designs, where a design is a set of values for a set of design decisions. In the context of this thesis, AutoDes does not perform text processing or image processing, it produces a design for a deep neural network pipeline which will be able to perform text processing or image processing. However it is important to note that the quality of the deep neural network pipeline design is directly correlated to the quality of the text processing or image processing performed by that deep neural network pipeline.

Designs can be disposable or reusable. Disposable designs are used only once and created for a specific domain, task or dataset and are not intended to be used to produce solutions for domains, tasks or datasets outside of which was seen during the automated design process. Reusable designs are designs that can be reused on domains, tasks or datasets outside of which was seen during the automated design process and still produce acceptable results. A reusable design achieves results that are on par with results achieved by a disposable design.

Evolutionary algorithms [6], genetic algorithms [165], grammatical evolution [165], bayesian and gradient based approaches [166] have all been used to automate the design of an algorithm. The introduction of more complex problem domains, resources intensive algorithms and an increase in non-machine learning experts making use of machine learning, have all contributed to the need for automated design [167]. The goal of automated design is to provide designs of equal or greater quality than what could be designed by a human in less time.

5.4 Hyper-heuristics

A hyper-heuristic can be defined as an algorithm that either generates or searches for heuristics for a given optimization problem, where the heuristics that the algorithm finds are used to either create a new solution or improve an existing solution to a given optimization problem [7]. This definition specifies two spaces: the heuristic space and the solution space. The hyper-heuristic performs search in the heuristic space, where the heuristic space consists of a collection of low-level heuristics. The low-level heuristics will either be operators that make incremental changes to a candidate solution or the low-level heuristics will rep-

resent components of the candidate solution itself. The low-level heuristics are mapped to the solution space, indirectly moving the search through the space, where the solution space consists of a collection of candidate solutions to the optimization problem that is being solved. Hyper-heuristics perform selection or generation, where selection hyper-heuristics select low-level heuristics in the heuristic space and generation hyper-heuristics create new low-level heuristics. Hyper-heuristics are perturbative or constructive in nature. Perturbative hyper-heuristics make incremental changes to an existing solution by means of low-level heuristics and constructive hyper-heuristics create an entirely new solution by means of low-level heuristics. Hyper-heuristics can thus be classified as being selection-perturbative, selection-constructive, generation-perturbative or generation-constructive.

The two classes of hyper-heuristics that are most commonly used for automated design are selection-perturbative hyper-heuristics and generation-perturbative hyper-heuristics. When using a hyper-heuristic to perform automated design, one of two scenarios can arise. The first scenario is where the low-level heuristics that are being selected are the actual values for the design decisions. The second scenario is one where a design is created with random values assigned to the design decisions and the low-level heuristics selected will consist of operators that are selected to make changes to the design decisions. In the second scenario another space called the design space is added. The heuristic space still consists of low-level heuristics, but instead of these low-level heuristics directly affecting the solution, it affects the design which resides in the design space. The design produced is evaluated to produce a solution (in the solution space) to the optimization problem that is being solved.

The use of hyper-heuristics for AutoDes has recently been proven to be effective [7]. The work in [168] made use of a genetic algorithm to design a convolutional neural network for text classification and the work in [169] made use of genetic programming to evolve a convolutional neural network for the purpose of image classification. Hyper-parameter optimization has also been done by means of hyper-heuristics, the work in [170] used a genetic algorithm to search for optimal CNN parameters. The work in [171] used different hyper-heuristic methods to search for optimal SVM hyper-parameters, specifically applied to a sentiment analysis task.

The use of an evolutionary algorithm for automated design must be considered carefully as evolutionary algorithms can become computationally expensive when applied to complex domains due to the repeated calculation of fitness values [172]. Using an evolutionary algorithm could ultimately lead to increased runtimes which undermines the goal of automated design to decrease design time. Single point hyper-heuristics offer a more lightweight but still effective alternative to evolutionary algorithms. Using a hyper-heuristic to design a deep neural network pipeline has at the time of writing not been attempted yet, furthermore

a selection perturbative hyper-heuristic has not been used for automated design of such a pipeline.

5.5 Transfer learning

5.5.1 Overview

A formal definition of transfer learning is given in [173] which states that given a source domain and a source task and a target domain and target task, transfer learning helps to improve the learning of the target function using the knowledge in the source domain and source task. The target function is what is used to perform the task, for example if we are trying to solve a classification problem the target function is what will be used to classify dataset instances in the domain. In transfer learning two crucial concepts to understand is that of the transfer learning source and a transfer learning target. Knowledge is extracted from the source and then transferred and applied to the target. Transfer learning works on the principle that starting the optimization process from an initial solution that is known to perform well for a different source domain or source dataset, could result in better performance of the optimization process.

Transfer learning is however non-trivial, it cannot be assumed that transferring knowledge from any source to any target is beneficial. Sometimes transfer learning results in what is known as negative transfer, this refers to a case where the optimization process is slowed or somehow has its performance worsened directly as a result of the knowledge that was transferred [174]. Some factors that contribute to negative transfer include domain divergence, a badly designed transfer algorithm and issues with data quality in either the source or target [175].

Negative transfer can be avoided by carefully scrutinizing the transfer learning to be performed. A framework for transfer learning is given by [174] and poses three questions in order to produce a transfer learning approach. The first question “What to transfer?”, refers to the knowledge that is extracted from the source and transferred to the target. For example, consider a neural network that has been trained to classify images of dogs into different breeds. If a neural network is created to do the same classification for cats, transfer learning can be used to leverage knowledge from the neural network that was trained to classify dogs. The transferred knowledge can be the architecture of the neural network, the hyper-parameters, the training schedule and more. The second question is “When to transfer?”. This question refers to the fact that optimization is a process that occurs for a period of time and transfer learning can be implemented at any point in that time period. Returning to the example of the neural network that classifies dog breeds, transfer of knowledge to the neural network that is being trained to classify cat breeds can occur at the very beginning before training starts or knowledge can be transferred only at later training epochs. The third and last question “How to transfer?”, refers to the means by which knowledge

is delivered to the target from the source. This question is easily answered in certain scenarios, for example if a target neural network simply has its hyper-parameters set to be the hyper-parameter values of a source neural network. This question becomes more complex to answer if knowledge is transferred from a source domain or system that differs from the target domain or system. For example, if weight values are transferred from a neural network with 5 layers, to a neural network with only 4 layers, a mapping function needs to be used to determine which weight values to discard.

Quantifying the success of transfer learning is important, as it allows the transfer learning process to be adjusted if it is suboptimal. Torrey et al. [176] provide a set criteria for quantifying the effectiveness of transfer learning. The first criterion is that when using transfer learning for a given algorithm, the results produced should not be worse than when using the same algorithm without any transfer learning. The second criterion specifies that when running the algorithm with transfer learning, the runtime to achieve comparable results should be less than when running the algorithm without transfer learning. The last criterion is that when applying transfer learning, the results produced by the algorithm should be better than when running the algorithm without any transfer learning.

5.5.2 Transfer learning for automated design

Transfer learning in automated design is a research area that has not garnered too much exploration at the time of writing this thesis. Applying transfer learning to a design that has been created automatically by means of some technique can be complex due to the existence of multiple concurrent search spaces. When using transfer learning in conjunction with automated design the questions raised by [174] are more difficult to answer and require more forethought and experimentation.

Transfer learning for automated design has seen some success in neural architecture search, the work in [177] automated the design of the architecture of a deep neural network using a reinforcement learning based architecture search method. In this work several neural networks are generated and trained on a specific task, it is then empirically determined what parts of the architecture generally stays the same and then once that is known, those learnings are applied to new tasks and if the accuracy isn't acceptable some further refinement of the architecture is done.

The work in [8] was the first to attempt transfer learning for automated design of genetic programming (GP). In [8] the design of a GP is automated by means of grammatical evolution (GE), transfer learning is done by transferring designs of the GP from one run of the GE to another.

Transfer learning for automated hyper parameter optimization has seen some research in [178] where sequential model-based optimization (SMBO) is used to

optimize the hyper-parameters for different classifiers such as an SVM, perceptron, logistic regression as well as an ensemble of classifiers. Surrogate based ranking was used to transfer knowledge from previous experiments.

At the time of writing automated design of a deep neural network pipeline has not been attempted, meaning that transfer learning applied to the automated design of a deep neural network pipeline is also yet to be attempted.

5.5.3 Transfer learning in hyper-heuristics

Transfer learning has been used for hyper-heuristics outside of the context of automated design, however it is still a novel area of research and the majority of the work done has focused on GP based methods and evolutionary algorithms as opposed to single point search.

The work in [179] applied transfer learning to a GP that is used to solve the uncertain capacitated arc routing problem (UCARP). Subtrees are transferred in different configurations from a source GP to a new target GP. The source and target problems share the same UCARP instance but different vehicle numbers. Transfer learning for GP does not always use subtrees as the mechanism of knowledge transfer. The work in [180] made use of a GP to perform constructive induction (CI) for both classification and regression. The goal of CI is to generate feature representations. The work in [180] uses a novel measure to determine whether source and target datasets are compatible and therefore eligible for transfer learning. The transfer learning in [180] is done by transferring the learned feature representations between runs of the GP.

Evolutionary algorithms (EAs) can be employed by a hyper-heuristic to explore the heuristic space, the work in [181] made use of an EA to match a photo of a person to a sketch of the same person. The photos and sketches are converted into 16 regions, where each region has two feature representations. The different regions are weighted differently to determine their contribution towards classifying the image as a whole. Each chromosome in the EA represents a weighting mechanism for each region which will result in each region being assigned a weight value. Transfer learning in [181] is done by transferring the 20 best individuals from a previous run of the EA (the source) to the next run of the EA (the target). The work in [182] also applied transfer learning to a hyper-heuristic making use of an EA by transferring individuals between runs, specifically when the source problem is low-order and the target problem is higher order.

Hyper-heuristic algorithms can result in a high-level of complexity and subsequently higher runtimes, making transfer learning a very promising research direction for advancing the field as a whole.

5.6 Summary

This chapter presented a critical analysis in Section 5.2 of the literature pertaining to deep neural networks as applied to text processing and image processing. From the critical analysis a consistent need for automation of the design of the deep neural network pipeline can be identified. Upon further investigation the automated design of the deep neural network pipeline consists of three distinct research areas, namely automated design, hyper-heuristics and transfer learning. Section 5.3 presented an overview of automated design and important related work pertaining specifically to the creation of pipelines and hyper-parameter tuning. Section 5.4 contained a brief discussion of hyper-heuristics generally before moving on to discuss and in the context of automated design. Finally Section 5.5 discussed transfer learning as relating to both automated design and hyper-heuristics.

CHAPTER 6

Research methodology

6.1 Introduction

This chapter describes the research methodology that was followed in order to achieve the objectives described in Chapter 1. The work makes use of a proof by demonstration methodology, which is discussed in Section 6.2. Section 6.3 provides a list of the domains and respective datasets that are used during experimentation. Section 6.4 outlines the experiments that are performed and Section 6.5 presents the performance measures that are used to evaluate the results of experimentation. Section 6.6 discusses the way in which statistical comparison is done for the various experiments. The technical specifications for the hardware and software that was used as part of the experimentation process are specified in Section 6.7. The chapter concludes with a summary in Section 6.8.

6.2 Proof by demonstration research methodology

6.2.1 Overview

The proof by demonstration research methodology is defined as building a system (also referred to as an artefact) and then letting that system (and the results of its execution) stand as an example for a more general class of solutions [183]. The proof by demonstration methodology requires that a hypothesis, or reason for conducting research, be established before constructing the system. There must be a clear link between the constructed system and the hypothesis, additionally the system must be iteratively refined until no further improvements can be achieved.

6.2.2 Application to Objectives

The first objective of this study is to automate the design of the deep neural network pipeline using a selection perturbative hyper-heuristic for both text processing and image processing. The second objective of this study is to investigate the effects of transfer learning on the automated design of a deep neural network pipeline. In order to achieve both objectives a selection perturbative hyper-heuristic (SPHH) is created. The goal of the SPHH is to produce a design for an end-to-end deep neural network pipeline for either a text processing or image processing dataset. The text processing and image processing domains use a generic deep neural network pipeline structure, however the exact techniques

that can be used (and from which the SPHH selects) differ between the two domains.

In fulfillment of Objective 1, the SPHH designs a deep neural network pipeline for each dataset and the results are empirically compared to results achieved by the manually designed deep neural network pipelines. In fulfillment of Objective 2, the SPHH designs a deep neural network pipeline for each dataset, however in this case the design process incorporates the designs from previous executions of the SPHH, in other words transfer learning is used. The results obtained from experimentation performed using transfer learning is then empirically compared to the results of experimentation performed when not using transfer learning.

6.2.3 Proof by demonstration methodology for selection perturbative hyper-heuristic (SPHH)

To create an SPHH in accordance with the two aforementioned research objectives, the proof by demonstration research methodology is employed as follows:

1. Design and implement an SPHH that designs a deep neural network pipeline.
2. Test the SPHH using both text and image processing datasets (described in Section 6.4)
3. Run the pipeline that was designed by the SPHH on the text processing or image processing dataset for which the pipeline was designed.
   (a) The SPHH incorporates stochastic elements in the search process and the deep neural network pipeline designs themselves also use stochasticity. To account for this the SPHH is run 30 times for each dataset with a different random seed being used for each run. The average results over those 30 runs is used to evaluate the results of the SPHH.
4. Compare the performance of the deep neural network pipeline that was designed by the SPHH to the performance of manually designed deep neural network pipelines using the measures outlined in Section 6.5.
5. If the results from the deep neural network pipeline that was designed by the SPHH are determined to be worse than the results from the manually designed deep neural network pipelines, the SPHH must be refined by making one or more of the below changes:
   (a) Add or remove techniques from the pool of techniques from which the SPHH is selecting for one of the stages of the deep neural network pipeline.
   (b) Adjust the value ranges for the hyper-parameters that are being optimized by the SPHH.
   (c) Evaluate the deep neural network pipeline design produced by the SPHH and identify whether there are additional areas that the SPHH can optimize. For example if there are any hyper-parameters that are being kept static those hyper-parameters should be added to the list of design decisions that the SPHH selects values for.

(d) Add, remove or alter the low level perturbative heuristics that the SPHH is using.
(e) Alter the hyper-parameters of the SPHH itself, for example the number of iterations, the way in which the design is initialized, the move acceptance criteria, the heuristic selection technique, the design representation.

6. Test the revised SPHH using both text and image processing datasets, if the SPHH again does not design a deep neural network pipeline that produces results that are on par with manually design deep neural network pipelines, revise the SPHH again until the aforementioned condition is met or until no further performance improvements occur. If the SPHH again does not design a deep neural network pipeline that produces results that are on par with manually design deep neural network pipelines, the reasons for this will be identified and reported.

Note that the development and implementation of the SPHH is discussed in detail in Chapter 8.

6.3 Problem domains and datasets

This research makes use of two problem domains, namely text processing and image processing. The design of the deep neural network pipeline is automated for both these problem domains. These two problem domains and the datasets they are represented by are discussed in the sections below.

6.3.1 Text processing

The text processing problem domain consists of datasets that can be divided into two subdomains: sentiment analysis and spam detection.

The sentiment analysis datasets that are used in this study are as follows:

1. ACL IMDB movie reviews dataset [102]: This dataset is made up of movie reviews made on the IMDB website. Importantly, only highly polarized reviews - reviews with less than or equal to 4 stars or reviews with more than or equal to 7 stars - are included. This dataset comprises 50000 instances, where 25000 have positive sentiment and 25000 have negative sentiment.
2. Amazon product reviews dataset [104]: This dataset is made up of user reviews on the Amazon website. The following categories from the "unprocessed.tar" folder were selected: apparel, automotive, baby, beauty, books, cameraphoto, cellphonesservice, computervideogames, dvd, electronics, gourmetfood, grocery, healthpersonalcare, jewelrywatches, kitchenhousewares, magazines, music, musicalinstruments, officeproducts, outdoorliving, software, sportsoutdoors. toolshardware, toysgames and video. This resulted in a total of 38548 instances, of those instances 21972 had positive sentiment and 16576 had negative sentiment.

3. Coronavirus Tweets dataset [55]: This dataset is made up of tweets gathered from Twitter from March 2020 onwards and contains tweets relating to COVID-19. The full dataset is continually growing as the COVID-19 pandemic is an ongoing event, for this reason only a subset of the full dataset was used, namely the dataset file named "coronavirustweets01.csv". This contains a total of 831328 instances where the sentiment score ranges between -1.0 to 1.0.
4. Sentiment 140 dataset [107]: This dataset is made up of tweets gathered from Twitter where the emoticons within the tweets decided the sentiment of the tweet. This dataset contains a total of 1.6 million instances where 800000 of those instances had a postive sentiment and 800000 of those instances had a negative sentiment.
5. Yelp reviews dataset [109]: This dataset is made up of user reviews for businesses as recorded on the website Yelp. A total of 342858 instances were used from the Yelp academic reviews dataset, where the stars attribute given for each instance was used to determine sentiment. The stars attribute is an integer within the range 1 to 5, inclusive.

The spam detection datasets used in this study are as follows:

1. Enron email dataset [111]: This dataset contains a subset of the total Enron dataset, which is made up of leaked emails from employees that worked at Enron in 2002. This dataset specifically focuses on six users: farmer-d, kaminski-v, kitchen-l, williams-w3, beck-s, and lokay-m. The total number of instances in this dataset is 33716, where 17171 of these emails are spam and 16545 of these emails are ham.
2. SMS Spam dataset [113]: This dataset consists of a collection of short messaging system (SMS) messages. This dataset has a total of 5575 instances where 4850 of those instances are ham and 725 of those instances are spam.
3. Spam Assassin dataset [115]: This dataset was created from a selection of email messages made available publically for the express purpose of testing spam filtering systems for email clients. This dataset has a total of 6047 instances, where 1897 of those instances are spam and 4150 of those are ham.
4. YouTube comments spam dataset [116]: This dataset contains comments from five YouTube videos, the total number of instances in this dataset comes to 1956. With 951 instances being ham and 1005 instances being spam.

6.3.2 Image processing

The image processing problem domain consists of datasets that can be divided into two subdomains: image classification and image segmentation.

The image classification datasets used in this study are as follows:.

1. Oral lesion dataset [145]: This dataset consists of 352 color photos of patient mouths that show some kind of oral lesion, where 165 of the images

are classified as benign lesions and 187 of the images are classified as malignant lesions. The Oral Lesion Dataset also contains 2622 augmented images, where 1156 of the augmented images are classified as benign and 1115 of the augmented images are classified as malignant.
2. University of Pretoria (UP) oral lesion images dataset [146], this dataset consists of color photos taken of patient mouths that display both benign and malignant lesions. The photos were provided by the Periodontics and Oral Medicine department at the University of Pretoria. The dataset consists of 69 images where 60 of the images were classified as benign and 9 of the images were classified as malignant.

The image segmentation datasets used in this study are as follows:

1. Iowa State University (ISU) maize disease dataset - first version [143], this dataset was created from maize plants grown in a greenhouse at Iowa State University. The maize plants were inoculated by spraying the leaves with common rust urediniospores. At different stages of infection, maize leaves were removed and scanned on a flatbed scanner at 1200 DPI. This dataset consists of 1040 images in total
2. Iowa State University (ISU) maize disease dataset - Rp1D inoculation method, this dataset was created from maize plants grown in a greenhouse at Iowa State University. The maize plants were inoculated by means of the Rp1D method. At different stages of infection, maize leaves were removed and scanned on a flatbed scanner at 1200 DPI. This dataset consists of 167 images in total
3. Iowa State University (ISU) maize disease dataset - Tilt inoculation method, this dataset was created from maize plants grown in a greenhouse at Iowa State University. The maize plants were inoculated by means of the Tilt method. At different stages of infection, maize leaves were removed and scanned on a flatbed scanner at 1200 DPI. This dataset consists of 1114 images in total

6.4 Experiments

Three experiments are conducted as part of this research, these three experiments are each discussed in turn below.

6.4.1 Experiment 1: Automated design

The first experiment runs the SPHH on each of the text processing and image processing datasets specified in Section 6.4. Thirty independent runs of the SPHH are performed on each dataset and the best and average result is reported at the end of the 30 independent runs. For each of the runs the deep neural network pipeline that produces the best accuracy is recorded and compared to results of manually designed pipelines from the literature. This experiment is

done towards fulfilling Objective 1 of this study, which is to determine whether automated design of a deep neural network pipeline is effective for both the text processing and image processing domains.

6.4.2 Experiment 2: Transfer learning

Experiment 2 is done in fulfillment of Objective 2 of this study, which is to investigate the effects of transfer learning on the automated design of a deep neural network pipeline. For this study the pipeline design is the knowledge that is transferred in the transfer learning process.

A deep neural network pipeline is designed by the SPHH for each of the datasets included in this study and each best performing deep neural network pipeline design will be a transfer learning source. When the SPHH is used without transfer learning (as in Experiment 1) the deep neural network pipeline design is initialized by randomly choosing the techniques per stage and their associated hyper-parameter values. When the SPHH has transfer learning applied, the deep neural network pipeline design is initialized by setting the techniques chosen per stage and all their associated hyper-parameter values to that of a previously designed deep neural network pipeline (the transfer learning source). The SPHH continues to optimize and tune the various components of the pipeline design for the target dataset after transfer learning is applied, with the hypothesis being that the search process is now starting in a more ideal location and therefore ought to produce better or equally good results.

In order for transfer learning to be used, the source domain and the target domain must be the same. In other words, a text processing deep neural network pipeline design must be used as a transfer learning source when the SPHH is designing a deep neural network pipeline for a target text processing dataset. Conversely, only an image processing deep neural network pipeline design can be used as a transfer learning source when the SPHH is designing a deep neural network pipeline for a target image processing dataset. The different transfer learning source and target configurations that are tested in this experiment are indicated with an 'x' in Table 6.1 below.

			Transfer learning source								
			Text processing domain								
			ACL IMDB movie reviews	Amazon product reviews	Coronavirus	Sentiment 140	Yelp	Enron	SMS Spam	Spam assassin	YouTube comments spam
Transfer learning target	Text processing domain	ACL IMDB movie reviews	x	x	x	x	x				
		Amazon product reviews	x	x	x	x	x				
		Coronavirus	x	x	x	x	x				
		Sentiment 140	x	x	x	x	x				
		Yelp	x	x	x	x	x				
		Enron						x	x	x	x
		SMS Spam						x	x	x	x
		Spam assassin						x	x	x	x
		YouTube comments spam						x	x	x	x

			Transfer learning source				
			Image processing domain				
			UP oral lesion images	Karnataka oral lesion	ISU V1	ISU Rp1d	ISU Tilt
Transfer learning target	Image processing domain	UP oral lesion images	x	x			
		Karnataka oral lesion	x	x			
		ISU V1			x	x	x
		ISU Rp1d			x	x	x
		ISU Tilt			x	x	x

Table 6.1. The transfer learning configurations for both the text processing and image processing domains that will be run as part of the test for transfer learning efficacy in Experiment 2 and reusability in Experiment 3.

6.4.3 Experiment 3: Reusability

The goal of Experiment 3 is to determine whether the designs produced by the SPHH are disposable or reusable. The test for reusability is done by taking an existing deep neural network pipeline design that was designed for one dataset and applying it as-is (without any further optimization by the SPHH) to a different dataset. As for Experiment 2, in Experiment 3 the source domain and the target domain must be the same. In other words, a text processing deep neural network pipeline design is only tested for reusability on text processing datasets. Conversely, an image processing deep neural network pipeline design is only tested for reusability on image processing datasets. The different reusability configurations that are tested in this experiment are indicated with an 'x' in Table 6.1.

6.5 Performance measures

The performance of the SPHH is analysed using two performance measures in order to determine the efficacy of the proposed automated design system, these two performance measures are discussed in turn below.

6.5.1 Accuracy

The first performance measure is accuracy. When executing any deep neural network pipeline, the results produced consist of four numbers: The training accuracy, the training loss, the testing accuracy and the testing loss. The testing accuracy refers to the accuracy of the deep neural network when applied to unseen dataset instances and is what is used as the performance measure.

6.5.2 Design time

The second performance measure is design time, this refers to the time it takes to produce a deep neural network pipeline design. Note that this does not refer to the time it takes for that deep neural network pipeline design to execute. The time it takes the SPHH to execute is referred to as the automated design time and the time it takes a human to design a deep neural network pipeline by hand is referred to as the manual design time. In order to consider the SPHH an effective means of automating the design of the deep neural network pipeline, the automated design time must be less than the manual design time.

6.6 Statistical comparison

The significance of the results obtained are determined by performing a left one-tailed Mann-Whitney U test at a statistical significance of 0.05. Additional basic statistical measures such as the minimum value, maximum value, average, mean and standard deviation is also calculated and reported on.

In Experiment 1 the testing accuracy of manually derived deep neural network pipelines from the literature is compared to the testing accuracy of the deep neural network pipelines designed by the SPHH. For Experiment 1 the manual design time in minutes is compared to the automated design time in minutes.

In Experiment 2 the testing accuracy of pipeline designs produced by the SPHH without transfer learning is compared to the pipeline designs produced by the SPHH that is using transfer learning. For Experiment 2 the automated design time in minutes when using transfer learning is compared to the automated design time in minutes without using transfer learning.

In Experiment 3 the testing accuracy of pipeline designs when they are applied to the datasets for which they were designed is compared to the testing accuracy of pipeline designs when they are applied to the datasets for which they were not designed. The design time does not affect the reusability of a design, therefore this performance measure is not considered for Experiment 3.

6.7 Technical specifications

The selection perturbative hyper-heuristic (SPHH) is what performs the automated design of the deep neural network pipeline. The SPHH itself is developed using Python 3.9 but incorporates the well known industry standard libraries Keras and Tensorflow for the neural network aspects of this research. The NLTK library is used as it compiles many frequently used text processing algorithms and the Pillow library is used for image processing utilities. The Ray multiprocessing library is used to allow for distributed computation. The SPHH is discussed in more detail in Chapter 7.

All experiments are performed by means of resources offered by the CHPC (Centre for High Performance Computing). A cluster of 24 CPUs and a GPU was used for each independent run. The GPU is a Nvidia V100 and the CPUs are Intel 5th generation. Experimentation was set up such that the neural network training would run on a GPU and the remainder of computation on the CPU.

6.8 Summary

This chapter presented the research methodology. The proof-by-demonstration research methodology that this study employs was discussed at length in Section 6.2. Section 6.3 listed the image processing and text processing datasets this study makes use of along with the characteristics of the datasets. Section 6.4 listed the three experiments that this study conducts and linked them to the research objectives. Section 6.5 presents the performance measures that this study uses and Section 6.6 discusses the way in which statistical comparison is

done. Section 6.7 presents the technical specifications for the high performance computing architectures used during this study.

CHAPTER 7

Single point hyper-heuristic approach

7.1 Introduction

This chapter describes the single point selection-perturbative hyper-heuristic (SPHH) that automates the design of the deep neural network pipeline by first creating a random design string (DS) and then optimizing that DS. The DS specifies what techniques the deep neural network pipeline itself (abbreviated as DNNP) will comprise of. The DS and technique that can be selected to create the DNNP are discussed in Section 7.2. The overall SPHH algorithm as well as the different processes that the algorithm consists of is discussed in Section 7.3. This chapter concludes with a summary in Section 7.4.

7.2 The design string (DS)

The design string (DS) specifies the design of the deep neural network pipeline, indicating both the techniques that will be used at each stage as well as the hyper-parameter values for the selected techniques. The structure of the design string is discussed in Section 7.2.1 and the different techniques that are available to be selected from are discussed in Section 7.2.2.

7.2.1 The structure of the DS

The design string (DS) differs slightly depending on whether the design is being created for text processing or image processing. The DS is composed of n components, with each component representing a design decision in the DNNP. In the case of image processing the length of the DS is thirteen, representing thirteen design decisions and in the case of text processing it is sixteen. Figure 7.1 describes the design decision that each component of the DS represents for an image processing deep neural network pipeline.

Position in chromosome	Gene	Value definition		Stage
1	Augmentation stage technique	[1..8]		Augmentation
2	Mean normalization	use/exclude	position (one of [1..5])	Preprocessing
3	Gaussian blur	use/exclude	position (one of [1..5])	
4	Move from RGB to HSV	use/exclude	position (one of [1..5])	
5	Segmentation using k-means	use/exclude	position (one of [1..5])	
6	Contrast enhancement	use/exclude	position (one of [1..5])	
7	Deep neural network technique	Segmentation - [1..3] Classification - [1..18]		Processing

8	Deep neural network technique specific parameters	{
		parameter_1: value,
		parameter_2: value,
		...
		}		

9	Deep neural network technique optimizer	[1..8]		
10	Deep neural network technique loss function	[1..11]		
11	Deep neural network technique activation function	[1..9]		
12	Deep neural network technique kernel initializer	[1..8]		
13	Deep neural network technique bias initializer	[1..8]		

Fig. 7.1. Definition of the string representing the different design decisions for an image processing pipeline

The first column represents the position of the gene in the chromosome overall. Gene 1 represents the augmentation stage and can have an integer value of between 1 and 8 (inclusive), the augmentation technique that each value represents is discussed in Section 7.2.2.1. Gene 2, 3, 4, 5 and 6 each represent an image preprocessing technique that can either be used in the preprocessing stage or excluded from the preprocessing stage. Additionally each preprocessing technique must be assigned a position if it is used, where this position will determine the order in which the preprocessing techniques get applied. If the preprocessing technique is excluded, then the position is set to be -1. The preprocessing techniques are discussed further in Section 7.2.2.2. Gene 7 represents the neural network architecture and gene 8 represents the architecture specific hyper-parameter values and are discussed in Section 7.2.2.4 and Section 7.2.2.5 respectively. Gene 9, 10, 11, 12 and 13 represent values for general deep neural network architecture hyper-parameters and are further discussed in Section 7.2.2.5.

Figure 7.2 describes the design decision that each component of the DS represents for a text processing deep neural network pipeline. Gene 1, 2, 3, 4, 5, 6, 7 and 8 each represent a text preprocessing technique that can either be used in the preprocessing stage or excluded from the preprocessing stage. Additionally each preprocessing technique that is used must be assigned a position, where this position will determine the order in which the preprocessing techniques get applied. If the preprocessing technique is excluded, then the position is set to be -1. The preprocessing techniques are discussed further in Section 7.2.2.2. Gene

9 represents the feature engineering technique that is used and can have an integer value between 1 and 6 (inclusive), the feature engineering technique that each value represents is discussed in Section 7.2.2.1. Gene 10 represents hyper-parameters that are specific to the chosen feature engineering technique. Gene 11 represents the neural network architecture chosen, the available architectures are discussed in Section 7.2.2.4. Gene 12, 13, 14, 15 and 16 represent values for general deep neural network architecture hyper-parameters and are further discussed in Section 7.2.2.5.

Position in chromosome	Gene	Value definition		Stage
1	Spelling correction	use/exclude	position (one of [1..8])	Preprocessing
2	Lemmatization	use/exclude	position (one of [1..8])	
3	Stemming	use/exclude	position (one of [1..8])	
4	Remove stop words	use/exclude	position (one of [1..8])	
5	Remove hashtags	use/exclude	position (one of [1..8])	
6	Removal of URLs and URI-type strings	use/exclude	position (one of [1..8])	
7	Removal of punctuation	use/exclude	position (one of [1..8])	
8	Conversion of all text to lowercase	use/exclude	position (one of [1..8])	
9	Feature engineering technique	[1..6]		Feature engineering

10	Feature engineering technique specific parameters	{
		parameter_1: value,
		parameter_2: value,
		...
		}		

11	Deep neural network technique	[1..4]		Classification
12	Deep neural network technique optimizer	[1..8]		
13	Deep neural network technique loss function	[1..11]		
14	Deep neural network technique activation function	[1..9]		
15	Deep neural network technique kernel initializer	[1..8]		
16	Deep neural network technique bias initializer	[1..8]		

Fig. 7.2. Definition of the string representing the different design decisions for a text processing pipeline

7.2.2 Deep neural network pipeline (DNNP) techniques

The SPPH optimizes the DS, the DS specifies techniques and hyper-parameter values for the different stages of the DNNP. This section lists the techniques that are available to be selected at the different stages for both the text processing and image processing domain.

7.2.2.1 Data augmentation stage The data augmentation stage is specific to image processing and is not needed for text processing. Research has shown that data augmentation techniques can result in better performance of subsequent classification [184] and segmentation [185] deep neural network techniques. Techniques that can be used for data augmentation are listed below (these techniques are discussed in detail in Chapter 4 Section 4.2):

1. Vertical flipping of an image [120]
2. Horizontal flipping of an image [120]
3. Rotation of an image [120]
4. Cropping an image to be smaller than the original [120]
5. Changing the position of the centre point of the image while preserving image size i.e. translation [120]
6. Changing the color space of an image [120]
7. Adding noise to an image [121]
8. Sharpening the image [121]

7.2.2.2 Preprocessing stage For the text processing domain the techniques that can be used at the preprocessing stage are listed below (these techniques are discussed in detail in Chapter 3 Section 3.2):

1. Apply spelling correction to the text
2. Lemmatization
3. Stemming
4. Removal of stop words
5. Conversion of hashtags into normal words
6. Removal of URLs and URI-type strings
7. Removal of punctuation
8. Conversion of all text to lowercase

For the image processing domain the techniques that can be used at the preprocessing stage are listed below (these techniques are discussed in detail in Chapter 4 Section 4.3):

1. Mean normalization [122]
2. Gaussian blur [123]
3. Moving an image from RGB to HSV color space [123]
4. Segmenting an image using K-means clustering [123]
5. Enhance contrast [124]

7.2.2.3 Feature engineering stage The feature engineering stage is specific to text processing and is not needed for image processing. Techniques that can be used for feature engineering are listed below (these techniques are discussed in detail in Chapter 3 Section 3.3):

1. One hot encoding
2. Term frequency - inverse document frequency (TF-IDF) [82]
3. Word2Vec (CBOW model) [80]
4. Word2Vec (skipgram model) [80]
5. GloVe [83]
6. FastText [81]

7.2.2.4 Neural network architecture selection For both text and image processing domains a neural network architecture must be selected and hyper-parameter tuned, this is the final stage of the pipeline. The neural network architectures that can be selected differ between the text processing and image processing domains, additionally different neural network architectures are used for the subdomains of image processing namely, image segmentation and image classification.

Neural networks that can be selected for text processing are listed below (these neural network architectures and the reason for their inclusion are discussed in detail in Chapter 3 Section 3.4):

1. Convolutional neural network (CNN)
2. Deep neural network (DNN)
3. Recurrent neural network (RNN)
4. Long short-term memory (LSTM)

Neural networks that can be selected for image processing, specifically image segmentation are listed below (these neural network architectures and the reason for their inclusion are discussed in detail in Chapter 4 Section 4.4):

1. Poly YOLO [141]
2. U-Net [139]
3. Mask R-CNN [140]

Neural networks that can be selected for image processing, specifically image classification are listed below (these neural network architectures and the reason for their inclusion are discussed in detail in Chapter 4 Section 4.4):

1. Xception [132]
2. VGG16 and VGG19 [125]
3. ResNet50, ResNet101 and ResNet152 [126]
4. ResNet50V2, ResNet101V2 and ResNet152V2 [127]
5. InceptionV3 [130]
6. InceptionResNetV2 [131]
7. MobileNet [133]
8. MobileNetV2 [186]
9. DenseNet121, DenseNet169 and DenseNet201 [128]

7.2.2.5 Hyper-parameter tuning For both image processing and text processing each of the deep neural network architectures have the following hyper-parameters that will also be tuned as part of the automated design process:

1. Optimizer algorithm, which can take one of the following values: Stochastic gradient descent, RMSProp, Adam, Adadelta, Adagrad, Adamax, NAdam and Ftrl.
2. Loss function, which can take one of the following values: Binary cross entropy, Poisson, Mean squared error, Mean absolute error, Mean absolute percentage error, Mean squared logarithmic error, Cosine similarity, Squared hinge, Categorical hinge and hinge.
3. Activation function, which can take one of the following values: ReLU, Sigmoid, Softmax, Softplus, Tanh, SeLU, eLU, Exponential and softsign.
4. Kernel initializer, which can take one of the following values: Random normal, Random uniform, Truncated normal, Zeros, Ones, Glorot normal, Glorot uniform and constant
5. Bias initializer, which can take the same values as specified for the above kernel initializer.

The deep neural network architectures that are listed for image segmentation have additional speciality hyper-parameters (such as the non-maximal suppression threshold for the Mask R-CNN) that are also tuned. The feature engineering stage also contains certain techniques such as FastText that have hyper-parameters that are tuned by the SPPH.

7.3 Single point hyper-heuristic algorithm

The pseudo-code for the SPHH is provided in Algorithm 1 below and describes how the SPHH will execute, the abbreviations used in Algorithm 1 are as follows:

1. DS - design string
2. DNNP - deep neural network pipeline
3. LLPH - low level perturbative heuristic

As described in Algorithm 1, first a DS is created randomly and greedy initialization is applied to that DS. The greedy initialization algorithm is discussed in detail in Section 7.3.1. After greedy initialization completes the SPHH will execute for 50 iterations. The reason why the SPHH executes for 50 iterations is because experimentation and parameter-tuning of the SPHH using different datasets shows that for all datasets the SPHH converges within 50 iterations and no further exploration occurs beyond 50 iterations.

At each iteration of the SPHH a low level perturbative heuristic (LLPH) is selected. The different LLPHs that can be applied to the DS are presented in Section 7.3.2 and the selection method for the LLPH is detailed in Section 7.3.4. Once a LLPH is selected, it is applied to the DS and a DNNP is created from

Algorithm 1 Single point selection-perturbative hyper-heuristic

```
create random DS
apply greedy initialization
iteration_limit ← 50
current_iteration ← 0

while current_iteration ≤ iteration_limit do

    select LLPH
    apply LLPH to DS
    create DNNP from DS
    evaluate DNNP
    update rank for LLPH

    if move is accepted then
        keep changes made to DS by LLPH
    else
        revert changes made to DS by LLPH
    end if

    current_iteration + 1

end while

return DS for the best performing DNNP as solution
```

that DS. The DNNP is then evaluated. The way in which the DNNP is constructed from the DS and evaluated is described in Section 7.3.3. The rank for the LLPH is updated using the performance of the DNNP and a move acceptance component (described in Section 7.3.4) determines whether the DS will stay as-is or revert the changes made by the LLPH.

Throughout the 50 iterations the SPHH keeps track of the best performing DNNP and the DS it was created from and that DS will be returned as the solution at the end of 50 iterations. The hyper-parameter values that were used for the SPHH are described in Section 7.3.5

7.3.1 Greedy initialization

To initialize the search, the SPHH will first randomly create a DS and then use greedy initialization. Greedy initialization applies each LLPH in the heuristic space to the DS one at a time and the resulting DNNP is evaluated. The LLPH that results in the best DNNP will have its move accepted. The reason for using greedy initialization is because a baseline performance can be established for each of the LLPHs, additionally the DS will have already been improved slightly at the start of algorithm execution.

Algorithm 2 Greedy initialization for the single point selection-perturbative hyper-heuristic.

```
create random DS
best_performing_DS ← null

for LLPH ∈ the heuristic space do
    select LLPH
    apply LLPH to DS
    create DNNP from DS
    evaluate DNNP
    update rank for LLPH

    if best_performing_DS is null then
        best_performing_DS ← DS
    end if
    if DS has better performance than best_performing_DS then
        best_performing_DS ← DS
    end if

end for

return best_performing_DS
```

Note that in the case of transfer learning being used, the DS will not be created randomly as the DS is set to be the DS from a previous execution of the SPHH.

7.3.2 Low level perturbative heuristics (LLPH)

The SPHH selects a low-level perturbative heuristic (LLPH) to apply to the DS in an attempt to improve the DS and subsequent performance of the DNNP. This section describes the LLPHs used in this study. The LLPHs are grouped into three categories, domain-independent, image processing and text processing.

The domain-independent LLPHs include:

1. Add new preprocessing technique: This heuristic will add a new preprocessing technique in a random position to the preprocessing stage.
2. Remove preprocessing technique: This heuristic will remove a randomly selected preprocessing technique from the preprocessing stage.
3. Select deep neural network architecture: This heuristics will select a different deep neural network architecture for the processing stage. The pool of available deep neural network architectures is determined by the problem domain.
4. Select optimizer: This heuristic will select a new optimizer for the deep neural network architecture being used.
5. Select loss function: This heuristic will select a new loss function for the deep neural network architecture being used.
6. Select activation function: This heuristic will select a new activation function for the deep neural network architecture being used.
7. Select kernel initializer: This heuristic will select a new function to initialize the kernel for a deep neural network architecture with.
8. Select bias initializer: This heuristic will select a new function to initialize the bias for a deep neural network architecture with.

The LLPHs specific to image processing are:

1. Select augmentation stage technique: This heuristic will select a different augmentation technique to replace the current augmentation technique.
2. Select new augmentation stage parameter value: Where applicable, this heuristic will select a new value for any parameters that are unique to the augmentation technique being used.
3. Select new CNN parameter value: Certain image segmentation deep neural network architectures have parameters that are unique to them, but still need to be tuned, for example the non-maximal suppression threshold parameter seen in the Mask R-CNN neural network architecture. If applicable, this low-level perturbative heuristic will be included.

The LLPHs specific to text processing are:

1. Select feature engineering stage technique: This heuristic will select a different feature engineering technique to replace the current feature engineering technique.
2. Select new feature engineering parameter value: Where applicable, this heuristic will select a new value for any parameters that are unique to the feature engineering technique being used.

7.3.3 Evaluation of the deep neural network pipeline (DNNP)

The DNNP that is constructed from the DS will be executed in its entirety on the dataset for which the SPHH is creating a design. The DNNP will use 10 fold cross-validation, making sure to keep the folds consistent between iterations. The testing accuracy that is returned by the DNNP will determine the fitness of the DS from which the DNNP was constructed.

7.3.4 Heuristic selection and move acceptance

The hyper-heuristic consists of two components, the heuristic selector and move acceptor. This section describes the choice function used for heuristic selection. The choice function is used to rank each perturbative heuristic using the following formula [7]:

$$f(hi) = \alpha f1(hi) + \beta f2(hi) + \delta f3(hi)$$

The low-level perturbative heuristic is represented as hi. $f1$, represents hi's performance over the previous n times that it was applied. $In(hi)$ is the change in the deep neural network pipeline's testing accuracy from the last invocation. $Tn(hi)$ is the time difference in seconds between the last application of hi.

$$f1(hi) = \sum_n \alpha^{n-1} \frac{In(hi)}{Tn(hi)}$$

The second constituent part of f is $f2$ which represents a comparison between hi and all other low level perturbative heuristics in the heuristic space hj, in the instances where hi and hj were applied in succession. $In(hj, hi)$ is now the difference in the deep neural network pipeline's testing accuracy from one successive application of hj and hi to the next. $Tn(hj, hi)$ is the time difference in seconds between the last successive invocation of hj and hi.

$$f2(hj, hi) = \sum_n \beta^{n-1} \frac{In(hj,hi)}{Tn(hj,hi)}$$

The last part $f3$ is the time in CPU seconds since hi was last applied.

$$f3(hi) = \delta(hi)$$

The equation f has three extra parameters α, β and δ. Both α and β are used to increase or decrease the influence that hi's recent performance has. The δ

parameter is used to increase diversity.

The low-level perturbative heuristic with the highest rank is selected to be applied to the DS. After the DNNP created from the perturbed DS has been evaluated, this perturbation/move can be either accepted or not. For this research, Adapted Iteration Limited Threshold Accepting (AILTA) [187] is used to determine whether to accept the perturbation or not.

AILTA works by accepting moves that result in the design performing either equally or better than it was prior to the move. AILTA will also accept moves that result in design performance degrading when certain conditions are met. AILTA has two hyper-parameters: an iteration limit and a threshold. Moves that degrade design performance will be accepted if the current iteration is greater than the iteration limit and if the performance of the solution is less than the threshold of the best performance obtained thus far.

The iteration limit and threshold values are then adapted as the search progresses to allow for more exploration of the heuristic space early on and exploitation of good design towards the end of the SPHH's execution.

7.3.5 SPHH Parameter Values

The six parameters in Table 7.1 were all tuned in tandem by first doing some by-hand empirical experimentation to get to a good starting point, then further tuned by means of a basic grid search. The iteration count value specifically was validated afterwards to ensure that no further solution improvement will happen beyond 50 iterations.

Parameter	Value
Number of iterations	50
AILTA iteration limit	10
AILTA threshold	0.15
Choice function - alpha value	0.5
Choice function - beta value	0.5
Choice function - delta value	0.25

Table 7.1. The SPHH parameters along with their values are listed in the table above.

7.4 Summary

This chapter describes in detail the various components of the SPHH that will be used to automate the design of the deep neural network pipeline. The design string representation is given in Section 7.2.1. The deep neural network pipeline itself and the techniques that can be selected at each stage is described in Section 7.2.2. The greedy initialization method that is used is discussed in more detail in Section 7.3.1 and low level perturbative heuristics are listed in Section 7.3.2. The fitness evaluation for the design string is discussed in Section 7.3.3 and the selection method and move acceptance criteria is detailed in Section 7.3.4.

CHAPTER 8

Results and discussion

8.1 Introduction

This chapter presents the results from the three experiments described in Chapter 6 Section 6.4. Each experiment has the results for the text processing and image processing datasets presented and discussed. Experiment 1 is presented in Section 8.2, here the designs from the SPHH are compared to the results of manually derived pipelines from the literature. Experiment 2 is presented in Section 8.3 and the results of using transfer learning with the SPHH are compared to the results of not using transfer learning with the SPHH. Lastly Experiment 3 is presented in Section 8.4 where the results from reusing pipeline designs are compared to the results of designing a deep neural network pipeline from scratch for each dataset. Finally Section 8.5 summarizes the chapter.

8.2 Experiment 1 - Automated design of the deep neural network pipeline

8.2.1 Text processing

Table 8.1 (sentiment analysis datasets) and Table 8.2 (spam detection datasets) shows the performance of the SPHH automated design as compared to the results from the deep neural network pipelines from the literature (see Chapter 3 for more details). The SPHH automated design is compared to the manually designed pipelines in terms of two performance metrics: testing accuracy (indicated in the table as "Acc.") and application time (indicated in the table as "Time." and given in minutes). The application time refers to the time it takes in minutes for a deep neural network pipeline to run to completion, in other words, for all stages to complete. Note that where the application time was not available, this was indicated with "n/a".

From Table 8.1 and Table 8.2 we can see that the SPHH exceeds the performance of the techniques from the literature survey for the Amazon dataset, Coronavirus dataset, Enron dataset and YouTube comments spam dataset. For the remainder of the datasets the SPHH falls just a few percentage points (in some cases less than an entire percentage) short of the performance from the techniques in the literature.

	ACL IMDB movie reviews		Amazon product reviews		Coronavirus		Sentiment 140		Yelp	
	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time
SPHH	94.57	162.32	87.92	93.04	92.06	64.42	91.36	209.23	87.92	237.4
[188]	95.68	900	-	-	-	-	-	-	-	-
[101]	97.4	n/a	-	-	-	-	-	-	-	-
[189]	90.3	n/a	82.0	n/a	-	-	-	-	-	-
[190]	-	-	86.54	n/a	-	-	-	-	-	-
[103]	-	-	87.86	10	-	-	-	-	-	-
[191]	-	-	-	-	81.4	n/a	-	-	-	-
[93]	-	-	-	-	90.67	395.27	-	-	-	-
[69]	-	-	-	-	-	-	84.41	1440	-	-
[192]	-	-	-	-	-	-	84.9	n/a	-	-
[106]	-	-	-	-	-	-	92.0	n/a	-	-
[64]	94.0	n/a	-	-	-	-	-	-	89.0	n/a
[68]	-	-	-	-	-	-	-	-	88.62	n/a

Table 8.1. Testing accuracy and application time of the SPHH designed deep neural network pipelines compared with other deep neural network pipeline techniques from the literature for the sentiment analysis datasets.

	Enron		SMS Spam		Spam assassin		YouTube comments spam	
	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time
SPHH	98.5	94.04	92.96	47.43	91.77	194.42	98.2	43.09
[110]	97.47	170.75	98.1	225.15	-	-	-	-
[94]	-	-	95.33	n/a	-	-	-	-
[112]	-	-	98.11	n/a	-	-	-	-
[71]	95.9	n/a	-	-	95.9	n/a	-	-
[114]	-	-	-	-	98.67	n/a	-	-
[66]	-	-	96.4	n/a	-	-	94.7	n/a
[193]	-	-	-	-	-	-	96.61	n/a

Table 8.2. Testing accuracy and application time of the SPHH designed deep neural network pipelines compared with other deep neural network pipeline techniques from the literature for the spam detection datasets.

When considering the difference between the application time for the pipeline designs produced by the SPHH and the manually derived designs in the literature, results show that the SPHH designs pipelines with application times that are either equal to or lower than the manually derived pipelines from the literature.

The results prove that the automation of the deep neural network pipeline for text processing is indeed effective, confirming the first objective of this research. For some datasets the SPHH designs pipelines that achieve better performance than the manually designed pipelines while for other datasets the SPHH designs pipelines that only match the performance of the manually designed pipelines, the SPHH is seen as effective in both cases as the aim of this study is to create an automated design technique that can replace the manual design process. Table 8.3 breaks down the best performing pipeline designs produced by the SPHH for a more holistic view of the experimentation. The most popular techniques per stage are listed. Table 8.3 illustrates clearly that different datasets require different techniques per stage. This table also shows that the SPHH avoids techniques that are simpler and from the literature survey are shown to be used less often. For example one-hot encoding does not appear in the table as a popular feature engineering technique, which mirrors the choice human data scientists might make when manually designing a deep neural network pipeline, since one-hot encoding is quite a basic technique. In other words, the SPHH is in fact performing informed design and not just simply moving around the search space randomly.

Table 8.4 compares the time that it takes the SPHH to design a deep neural network pipeline to the time that it takes to manually design a deep neural network pipeline for all the text processing datasets. The time it takes the SPHH to design a deep neural network pipeline is the automated design time. The manual design times in Table 8.4 are estimates that are based on the author's experience of working with these datasets. When manually designing a deep neural network pipeline, multiple different pipeline configurations are trialled by hand. The runtime of a deep neural network pipeline depends on the size of the dataset which is why certain datasets in Table 8.4 have longer manual design times (such as Spam assassin) and other datasets such as SMS Spam have shorter manual design times. Although the manual design times in Table 8.4 are estimates, they sufficiently illustrate that the automated design time is well within an acceptable time frame for all the datasets listed.

Dataset	Best performing design		
	Preprocessing stage	Feature engineering stage	Classification stage
ACL IMDB movie reviews	Stemming, Lemmatization	Word2Vec CBOW	RNN
Amazon product reviews	Remove punctuation, Remove URI like objects, Lemmatization, Stemming, Convert hashtags, Convert text to lowercase, Spelling correction, Remove stop words	GloVe	LSTM
Coronavirus	Convert hashtags, Lemmatization, Stemming, Spelling correction	FastText	LSTM
Sentiment 140	Convert hashtags, Spelling correction, Remove punctuation, Remove URI-like objects, Stemming, Convert text to lowercase, Remove stop words, Lemmatization	Word2Vec CBOW	DNN
Yelp	Spelling correction, Lemmatization, Stemming	Word2Vec CBOW	LSTM
Enron	Stemming	Word2Vec CBOW	LSTM
SMS Spam	Lemmatization, Remove stop words, Convert hashtags, Spelling correction, Stemming	Word2Vec CBOW	LSTM
Spam assassin	Convert hashtags, Remove URI-like objects, Remove stop words, Spelling correction	FastText	LSTM
YouTube comments spam	Stemming, Remove stop words, Spelling correction, Convert hashtags, Lemmatization	GloVe	DNN

Table 8.3. This table illustrates the best performing designs produced by the SPHH for the text processing domain.

Dataset	Manual design time	SPHH design time
ACL IMDB movie reviews	72	18.93
Amazon product reviews	72	17.33
Coronavirus	72	12.42
Sentiment-140	96	23.70
Yelp	96	28.94
Enron	72	18.99
SMS Spam	48	11.09
Spam assassin	96	27.45
YouTube comments spam	48	8.23

Table 8.4. This table compares the time that it takes (in hours) to manually design a deep neural network pipeline to the time it takes the SPHH to design a deep neural network pipeline (automated design time) for each of the text processing datasets.

8.2.2 Image processing

Table 8.5 shows the performance of the SPHH automated design as compared to the manually designed deep neural network pipelines from the literature (see Chapter 4 for more details). For the ISU Rp1d and ISU Tilt datasets there are no published results available, making this research the current state of the art. The other datasets in the image processing domain each only have a single deep neural network pipeline described in the literature, making it their state-of-the-art (SotA) technique. The SotA technique refers to the technique that provides the best performance for that dataset currently.

Subdomain	Dataset	SotA		SPHH	
		Acc.	Time.	Acc.	Time
Classification	Karnataka oral lesion dataset	92.43 [144]	606.25	98.77	628.32
	UP oral lesion images dataset	82.13 [144]	n/a	92.01	162.57
Segmentation	ISU V1	68.02 [143]	6845	72.45	711.98
	ISU Rp1d	-	-	69.94	733.32
	ISU Tilt	-	-	70.56	841.24

Table 8.5. Testing accuracy and application time of the SPHH designed deep neural network pipelines compared with other deep neural network pipeline techniques from the literature for the image processing datasets.

From Table 8.5 we can see that the SPHH exceeds the performance achieved by previous work for all datasets listed. The results confirm that the automation of the deep neural network pipeline for image processing is effective. When considering the difference between the application time for the pipeline designs produced by the SPHH and the manually derived designs in the literature, we can see that in the instances where application times are available, the SPHH designs pipelines with application times that are either equal to or lower than the manually derived pipelines from the literature. Considering that the SPHH has an augmentation stage, which results in the total number of dataset instances increasing, this is an especially encouraging result.

Table 8.6 breaks down the best performing pipeline designs produced by the SPHH for a more holistic view of the experimentation. The most popular techniques per stage are listed. Table 8.6 shows that for the two image classification datasets, different techniques are clearly required per stage. For image segmentation we can see that similar design choices are being made across all three datasets, this is in fact expected behaviour, since the three datasets have the same provenance and are visually similar. The SPHH making similar design choices for the three datasets again proves that the design process is informed.

Subdomain	Dataset	Best performing design		
		Augmentation stage	Preprocessing stage	Classification stage
Classification	Oral lesion dataset	Crop	Contrast enhancement	XCeption
	UP oral lesion images dataset	Scale	Mean normalization	Resnet-50
Segmentation	ISU V1	Flip	RGB to HSV	U-net
	ISU Rp1d	Translate	RGB to HSV	U-net
	ISU Tilt	Flip	Mean normalization Gaussian blur	U-net

Table 8.6. Best performing designs produced by the SPHH for the image processing datasets.

Table 8.7 compares the time that it takes the SPHH to design a deep neural network pipeline to the time that it takes to manually design a deep neural network pipeline for all the image processing datasets. Table 8.7 illustrates that the automated design time improves on the manual design time across all the image processing datasets.

Dataset	Manual design time	SPHH design time
Karnataka oral lesion dataset	120	24.5
UP oral lesion images dataset	120	15.83
ISU V1	336	29.92
ISU Rp1d	336	29.33
ISU Tilt	336	30.02

Table 8.7. This table compares the time that it takes (in hours) to manually design a deep neural network pipeline to the time it takes the SPHH to design a deep neural network pipeline (automated design time) for each of the image processing datasets.

8.3 Experiment 2 - Transfer learning for automated design

8.3.1 Text processing

Table 8.8 (sentiment analysis) and Table 8.9 (spam detection) shows the performance of the SPHH when using the best performing design from different datasets as the initial design, ergo, when using transfer learning. Transfer learning was incredibly effective on the whole, however the choice of source for transfer learning does seem to have an effect on the efficacy of the transfer learning. In certain instances such as when using the Yelp dataset as the source of design for the Coronavirus dataset, the use of transfer learning resulted in poorer performance. However for other datasets, such as when using the SMS Spam dataset

as source and the Enron dataset as target, transfer learning resulted in better performance than when no transfer learning was used.

The different transfer learning configurations are checked for statistical significance using a left-tailed Mann-Whitney U test at 0.05 significance. The Mann-Whitney U test compares the results of running the SPHH on the target dataset with transfer learning and without transfer learning. Table 8.8 (sentiment analysis) and Table 8.9 lists the p-values for this comparison where a p-value larger than 0.05 indicates that the SPHH's performance when not using transfer learning is assumed to be better than or equal to the SPHH's performance when using transfer learning. A p-value smaller than 0.05 indicates that the SPHH's performance when using transfer learning is assumed to be better than the SPHH's performance when not using transfer learning.

Table 8.10 shows the average iteration at which the SPHH converges on a design. From the results we can see that the use of transfer learning in certain instances does not affect the average iteration at which the SPHH converges (such as when using the Amazon product reviews dataset as source and Sentiment 140 dataset as target) and in other instances transfer learning dramatically decreases the average iteration at which the SPHH converges (such as when using the Coronavirus dataset as source and Amazon product reviews dataset as target). When the use of transfer learning decreases the search effort (in this case time) required to arrive at a solution, we consider that application of transfer learning to be successful.

When comparing the performance of the SPHH with transfer learning to the performance of the SPHH without transfer learning, the results are statistically significant for the ACL IMDB, Sentiment 140, Yelp, SMS Spam and Spam Assassin datasets. The other datasets do not show a statistically significant difference between the results of the SPHH when using transfer learning and not using transfer learning.

The second objective of this research inquired into whether transfer learning would be effective when doing automated design of the deep neural network pipeline. The results in Table 8.8, Table 8.9 and Table 8.10 show that transfer learning improves performance in certain configurations, but it cannot be used thoughtlessly. The target and source must be carefully considered as there is a correlation between the source and target datasets and performance.

Transfer learning target dataset	Metrics	Transfer learning design source				
		ACL IMDB movie reviews	Amazon product reviews	Coronavirus	Sentiment 140	Yelp
ACL IMDB movie	Acc.	96.42	92.75	96.72	96.96	87.13
	p-value	1.823e-8	1	4.63e-9	0.001751	1
Amazon product	Acc.	85.49	84.59	88.80	88.24	88.07
	p-value	1	0.9998	0.3922	0.5383	0.7587
Coronavirus	Acc.	89.82	90.85	90.50	92.58	88.97
	p-value	1	1	1	0.5088	1
Sentiment 140	Acc.	87.55	92.28	87.78	90.00	92.02
	p-value	1	3.521e-7	0.9995	0.9782	0.00125
Yelp	Acc.	88.57	88.48	83.42	86.13	87.15
	p-value	0.001078	0.4094	1	1	1

Table 8.8. Results of running the SPHH with transfer learning for the sentiment analysis text processing datasets.

Transfer learning target dataset	Metrics	Transfer learning design source			
		Enron	SMS Spam	Spam assassin	Youtube comments spam
Enron	Acc.	95.78	97.32	93.79	97.67
	p-value	1	0.9996	1	0.9992
SMS Spam	Acc.	91.19	94.07	95.35	91.28
	p-value	1	0.000004757	0.00004073	1
Spam assassin	Acc.	91.58	93.03	93.04	90.37
	p-value	0.9999	0.0007984	0.00007923	1
YouTube comments spam	Acc.	98.11	96.15	96.48	96.18
	p-value	0.1855	4.4723	1	0.9993

Table 8.9. Results of running the SPHH with transfer learning for the spam detection text processing datasets.

Transfer learning target dataset	No transfer learning	Transfer learning design source								
		ACL IMDB movie reviews	Amazon product reviews	Coronavirus	Sentiment 140	Yelp	Enron	SMS Spam	Spam assassin	Youtube comments spam
ACL IMDB movie reviews	40	26	27	27	37	32				
Amazon product reviews	40	28	26	26	27	29				
Coronavirus	39	27	28	26	32	30				
Sentiment 140	44	40	44	43	29	43				
Yelp	43	42	40	42	41	40				
Enron	40						30	31	30	30
SMS Spam	39						29	28	28	27
Spam assassin	42						39	40	40	36
YouTube comments spam	40						26	26	27	29

Table 8.10. This table displays the average iteration at which the search process converged for the different transfer learning configurations as well as when using the SPHH without any transfer learning.

8.3.2 Image processing

Table 8.11 and Table 8.13 shows the performance of the SPHH when using the best performing design from different image processing datasets as the initial design, ergo when using transfer learning. The results from Table 8.11 and Table 8.13 indicate that the efficacy of transfer learning when used in the image processing domain is also dependent on the source and target being used.

The different transfer learning configurations are checked for statistical significance using a left-tailed Mann-Whitney U test at 0.05 significance. The Mann-Whitney U test compares the results of running the SPHH on the target dataset with transfer learning and without transfer learning. Table 8.11 (sentiment analysis) and Table 8.13 lists the p-values for this comparison where a p-value larger than 0.05 indicates that the SPHH's performance when not using transfer learning is assumed to be better than or equal to the SPHH's performance when using transfer learning. A p-value smaller than 0.05 indicates that the SPHH's performance when using transfer learning is assumed to be better than the SPHH's performance when not using transfer learning.

When comparing the performance of the SPHH with transfer learning to the performance of the SPHH without transfer learning, the results are statistically significant for the UP oral images, ISU Rp1d and ISU Tilt datasets. The other datasets do not show a statistically significant difference between the results of the SPHH when using transfer learning and not using transfer learning.

Table 8.12 and Table 8.14 shows the average iteration at which the SPHH converges on a design. From the results we can see that the use of transfer learning in certain instances does not affect the average iteration at which the SPHH converges (such as when using the ISU Rp1d dataset as source and ISU V1 dataset as target) and in other instances transfer learning dramatically decreases the average iteration at which the SPHH converges (such as when using the Karnataka oral lesion dataset as source and the Karnataka oral lesion dataset as target).

Transfer learning target dataset	Metrics	Transfer learning design source	
		UP oral lesion images dataset	Karnataka oral lesion dataset
UP oral lesion images dataset	Acc.	92.20	91.02
	p-value	0.03622	0.9999
Karnataka oral lesion dataset	Acc.	97.00	98.99
	p-value	1	0.8942

Table 8.11. Results of running the SPHH with transfer learning for the image classification datasets.

Transfer learning target dataset	No transfer learning	Transfer learning design source	
		UP oral lesion images dataset	Karnataka oral lesion dataset
UP oral lesion images dataset	41	39	38
Karnataka oral lesion dataset	42	41	32

Table 8.12. This table displays the average iteration at which the search process converged for the different transfer learning configurations as well as when using the SPHH without any transfer learning.

Transfer learning target dataset	Metrics	Transfer learning design source		
		ISU V1	ISU Rp1d	ISU Tilt
ISU V1	Acc.	73.02	72.09	70.62
	p-value	0.674	0.9999	1
ISU Rp1d	Acc.	70.41	70.62	69.53
	p-value	0.07472	0.008477	0.9915
ISU Tilt	Acc.	71.37	70.99	71.67
	p-value	7.322e-11	0.00002633	0.00008407

Table 8.13. Results of running the SPHH with transfer learning for the image segmentation datasets.

Transfer learning target dataset	No transfer learning	Transfer learning design source		
		ISU V1	ISU Rp1d	ISU Tilt
ISU V1	40	29	40	30
ISU Rp1d	40	40	36	41
ISU Tilt	41	34	39	33

Table 8.14. This table displays the average iteration at which the search process converged for the different transfer learning configurations as well as when using the SPHH without any transfer learning.

8.4 Experiment 3 - Reusability of the deep neural network pipeline designs

8.4.1 Text processing

Table 8.15 shows the results of using the best performing design from different datasets as-is for a given dataset, in other words, reusing the design. In order to consider a design reusable, it must provide results comparable to a design that is explicitly created for a given dataset. Table 8.15 and Table 8.16 reveals interesting results in that in certain instances such as when the Coronavirus dataset is the target and the ACL IMDB movie reviews dataset is the source, the design does appear to be reusable. Another instance of a design being reusable is seen when the Enron dataset is the target and the YouTube comments spam dataset is the source. However for Table 8.15 and Table Table 8.16 it is clear that there are plenty of instances of design reuse producing exceedingly poor results, meaning that the design is not reusable. Overall a significant amount of the designs produced by the SPHH do not appear to be reusable for the text processing domain.

Transfer learning target dataset	Transfer learning design source				
	ACL IMDB movie reviews	Amazon product reviews	Coronavirus	Sentiment 140	Yelp
ACL IMDB movie reviews	-	80.10	62.16	80.32	51.78
Amazon product reviews	78.04	-	53.31	54.68	53.60
Coronavirus	92.50	91.12	-	93.99	75.09
Sentiment 140	57.66	53.27	46.35	-	54.39
Yelp	83.09	84.76	66.42	63.83	-

Table 8.15. Results for reusing pipeline designs created for one sentiment analysis text processing dataset on another.

Transfer learning target dataset	Transfer learning design source			
	Enron	SMS Spam	Spam assassin	Youtube comments spam
Enron	-	87.02	97.80	95.70
SMS Spam	89.09	-	88.11	88.10
Spam assassin	86.72	86.20	-	85.12
YouTube comments spam	72.14	61.88	59.91	-

Table 8.16. Results for reusing pipeline designs created for one spam detection text processing dataset on another.

8.4.2 Image processing

Table 8.17 and Table 8.18 show the results of using the best performing design from different image processing datasets as-is for a given dataset, in other words, reusing the design. With the exception of the ISU Tilt design being used for the ISU V1 dataset, for both the image classification and image segmentation the designs do not appear to be reusable as the performance is not comparable to the performance results from a design that was explicitly created for a given dataset.

Transfer learning target dataset	Transfer learning design source	
	UP oral lesion images dataset	Karnataka oral lesion dataset
UP oral lesion images dataset	-	89.43
Karnataka oral lesion dataset	93.09	-

Table 8.17. Results for reusing pipeline designs created for one image classification dataset on another.

Transfer learning target dataset	Transfer learning design source		
	ISU V1	ISU Rp1d	ISU Tilt
ISU V1	-	71.57	72.80
ISU Rp1d	67.32	-	67.99
ISU Tilt	68.94	69.43	-

Table 8.18. Results for reusing pipeline designs created for one image segmentation dataset on another.

8.5 Summary

This chapter presented the results of the experimentation that was performed using the SPHH defined in Chapter 7. Three experiments were performed, the first experiment investigates whether automated design of the deep neural network is effective and the results are presented in Section 8.2. The use of automated design to create a deep neural network pipeline is found to be effective. The second experiment investigates the use of transfer learning for automated design of the deep neural network pipeline and the results of experimentation are presented in Section 8.3. According to the results discussed in Section 8.3 transfer learning can be effective when used for the automated design of the deep neural network pipeline, however the source and target domains used must be chosen correctly. Future work can automate the process of selecting the source and target domains or perform an analysis to find a correlation between characteristics of the

domains and performance. The third experiment sought to determine whether the deep neural network pipelines produced by the SPHH are reusable, Section 8.4 presents these results and concludes that the designs are not reusable.

CHAPTER 9

Conclusions and future work

9.1 Introduction

This thesis analyses the process of deep neural network pipeline construction and design, this process is automated by means of a selection perturbative hyper-heuristic (SPHH). The SPHH is used to perform various experiments to investigate two research objectives. This chapter presents the conclusions that can be drawn per experiment and addresses the research objectives presented in Chapter 1. This chapter also makes recommendations for future work that will extend the work that has been put forth by this thesis.

9.2 Conclusions

9.2.1 Experiment 1 - Objective 1

The first objective of this research is to automate the design of the deep neural network pipeline using a selection perturbative hyper-heuristic for both text processing and image processing. This research has shown that automated design of a deep neural network pipeline using a selection perturbative hyper-heuristic is effective for the image processing and text processing domains. Automating the design of a deep neural network pipeline by means of a SPHH results in pipelines that are competitive with manually derived pipeline designs published in the literature for some of the datasets presented in both the text processing and image processing domains.

9.2.2 Experiment 2 - Objective 2

The second objective of this research is to investigate the effects of transfer learning on the automated design of a deep neural network pipeline. Transfer learning was found to produce results comparable or better than the results achieved when using the SPHH without transfer learning. Importantly however transfer learning is only effective when the correct target and source are chosen, for some target datasets negative transfer occurs when using certain deep neural network pipeline designs as the transfer learning source. Negative transfer occurs when the accuracy of the designed deep neural network pipeline decreases. Positive transfer occurs when the accuracy of the design deep neural network pipeline increases and/or the SPHH converges on a equally well performing design at an earlier iteration.

When comparing the performance of the SPHH with transfer learning to the performance of the SPHH without transfer learning, the results are statistically significant for the ACL IMDB, Sentiment 140, Yelp, SMS Spam and Spam Assassin datasets. The other text processing datasets do not show a statistically significant difference between the results of the SPHH when using transfer learning and not using transfer learning. A general pattern is observed where the SMS Spam and Spam assassin datasets can be used interchangeably as a transfer learning source and a transfer learning target for one another. Another pattern that emerges from the results is that ACL IMDB movie reviews is generally a good transfer learning target dataset. The Enron and YouTube comments datasets do not benefit from transfer learning and they also cannot be used as sources for transfer learning.

For the image processing datasets when comparing the performance of the SPHH with transfer learning to the performance of the SPHH without transfer learning, the results are statistically significant for the UP oral images, ISU Rp1d and ISU Tilt datasets. The other image processing datasets do not show a statistically significant difference between the results of the SPHH when using transfer learning and not using transfer learning. A general pattern is observed where ISU Rp1d and ISU V1 are generally good transfer learning sources. Interestingly ISU V1 is generally a poor transfer learning target dataset. The Karnataka oral lesion dataset is also generally a poor transfer learning target dataset.

9.2.3 Experiment 3 - Test for reusability

This work conducted experiments to determine whether the designs produced by the SPHH are reusable. The results show that reusing designs does not provide comparable performance to the results achieved when specifically creating a design for a dataset. In other words, the designs are not reusable for either the image processing or text processing domain and deep neural network pipelines must be designed for each dataset in order to achieve acceptable results.

9.3 Future work

Extension of the research presented in this thesis will involve the following:

- **Applying the automated design system to more domains.**
  This research made use of both the text processing and image processing domains, specifically sentiment analysis, spam detection, image segmentation and image classification. Future work will include the use of the SPHH for other domains like audio restoration and chemical engineering.

- **Analysing the SPHH to try and make designs more reusable.**
  This study found the designs produced by the SPHH to not be reusable,

work will be done to alter the SPHH to produce reusable deep neural network pipeline designs.

– **Automating the transfer learning process to ensure only positive transfer occurs.**
The transfer learning configurations (the source and target domain pairs) that this study used were derived manually; future work will automate this process to ensure that correct transfer learning configurations are chosen that will only result in positive transfer. In this study all aspects of the design were transferred, future work will investigate transferring only certain aspects of the design where the aspects to be transferred are chosen automatically by some optimization algorithm.

– **Use automated design to construct a pipeline for unsupervised deep learning techniques instead of deep neural networks.**
The state of the literature indicates a steady increase of interest in unsupervised deep learning techniques. The effect of transfer learning and examination of the reusability of unsupervised deep learning pipeline designs will also be conducted.

9.4 Summary

This chapter presented the conclusions that this work has come to based on the results presented in Chapter 8 with reference to the objectives set in Chapter 1. Recommendations for future work that emanated from the work done in this thesis are also presented.

Bibliography

[1] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. V. Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," 2018.

[2] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231216315533

[3] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[4] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[5] J. D. Romano, T. T. Le, W. Fu, and J. H. Moore, "Tpot-nn: Augmenting tree-based automated machine learning with neural network estimators," *Genetic Programming and Evolvable Machines*, vol. 22, no. 2, pp. 207–227, 2021.

[6] N. Pillay and R. Qu, *Automated Design of Machine Learning and Search Algorithms*. Springer, 2021.

[7] ——, *Hyper-Heuristics: Theory and Applications*. Springer, 2018.

[8] T. Nyathi and N. Pillay, "On the transfer learning of genetic programming classification algorithms," in *International Conference on the Theory and Practice of Natural Computing*. IEEE, 2021.

[9] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[11] A. Lydia and S. Francis, "Adagrad—an optimizer for stochastic gradient descent," *Int. J. Inf. Comput. Sci*, vol. 6, no. 5, 2019.

[12] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[13] E. Dogo, O. Afolabi, N. Nwulu, B. Twala, and C. Aigbavboa, "A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks," in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*. IEEE, 2018, pp. 92–99.

[14] M. Moreira and E. Fiesler, "Neural networks with adaptive learning rate and momentum terms," Idiap, Tech. Rep., 1995.

[15] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[16] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, 1969.

[17] M. Awad and R. Khanna, *Deep Neural Networks*. Berkeley, CA: Apress, 2015, pp. 127–147. [Online]. Available: https://doi.org/10.1007/978-1-4302-5990-9_7

[18] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[19] R. Collobert and S. Bengio, "Links between perceptrons, mlps and svms," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 23.

[20] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[21] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[22] G. Ososkov and P. Goncharov, "Shallow and deep learning for image classification," *Optical Memory and Neural Networks*, vol. 26, no. 4, pp. 221–248, 2017.

[23] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for natural language processing," *arXiv preprint arXiv:1606.01781*, vol. 2, p. 1, 2016.

[24] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning (adaptive computation and machine learning series)," 2016.

[25] C. Nebauer, "Evaluation of convolutional neural networks for visual recognition," *IEEE transactions on neural networks*, vol. 9, no. 4, pp. 685–696, 1998.

[26] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[27] M. Yaqub, J. Feng, M. S. Zia, K. Arshid, K. Jia, Z. U. Rehman, and A. Mehmood, "State-of-the-art cnn optimizer for brain tumor segmentation in magnetic resonance images," *Brain Sciences*, vol. 10, no. 7, p. 427, 2020.

[28] P. Verma, V. Tripathi, and B. Pant, "Comparison of different optimizers implemented on the deep learning architectures for covid-19 classification," *Materials Today: Proceedings*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785321013316

[29] T. Gaiceanu and O. Pastravanu, "On cnn applied to speech-to-text – comparative analysis of different gradient based optimizers," in *2021 IEEE*

15th International Symposium on Applied Computational Intelligence and Informatics (SACI), 2021, pp. 000 085–000 090.

[30] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[31] H.-J. Yoo, "Deep convolution neural networks in computer vision: a review," *IEIE Transactions on Smart Processing and Computing*, vol. 4, no. 1, pp. 35–43, 2015.

[32] A. Jacovi, O. S. Shalom, and Y. Goldberg, "Understanding convolutional neural networks for text classification," *arXiv preprint arXiv:1809.08037*, 2018.

[33] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[34] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," in *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2017, pp. 0588–0592.

[35] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[36] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1. IEEE, 1996, pp. 347–352.

[37] A. Alqushaibi, S. J. Abdulkadir, H. M. Rais, and Q. Al-Tashi, "A review of weight optimization techniques in recurrent neural networks," in *2020 International Conference on Computational Intelligence (ICCI)*, 2020, pp. 196–201.

[38] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[40] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[41] X. Wang, W. Jiang, and Z. Luo, "Combination of convolutional and recurrent neural network for sentiment analysis of short texts," in *Proceedings of COLING 2016, the 26th international conference on computational linguistics: Technical papers*, 2016, pp. 2428–2437.

[42] G. Liang, H. Hong, W. Xie, and L. Zheng, "Combining convolutional neural network with recursive neural network for blood cell image classification," *IEEE Access*, vol. 6, pp. 36 188–36 197, 2018.

[43] K. K. Chandriah and R. V. Naraganahalli, "Rnn/lstm with modified adam optimizer in deep learning approach for automobile spare parts demand forecasting," *Multimedia Tools and Applications*, pp. 1–15, 2021.

[44] N. Sakinah, M. Tahir, T. Badriyah, and I. Syarif, "Lstm with adam optimization-powered high accuracy preeclampsia classification," in *2019 International Electronics Symposium (IES)*. IEEE, 2019, pp. 314–319.

[45] B. Krause, L. Lu, I. Murray, and S. Renals, "On the efficiency of recurrent neural network optimization algorithms," in *NIPS Optimization for Machine Learning Workshop*, 2015.

[46] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.

[47] Y. D. Prabowo, H. L. H. S. Warnars, W. Budiharto, A. I. Kistijantoro, Y. Heryadi *et al.*, "Lstm and simple rnn comparison in the problem of sequence to sequence on conversation data using bahasa indonesia," in *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*. IEEE, 2018, pp. 51–56.

[48] D. Shen, *Text Categorization*. Boston, MA: Springer US, 2009, pp. 3041–3044. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_414

[49] D. Droba, "Methods used for measuring public opinion," *American Journal of Sociology*, vol. 37, no. 3, pp. 410–423, 1931.

[50] P. D. Turney and M. L. Littman, "Measuring praise and criticism: Inference of semantic orientation from association," *acm Transactions on Information Systems (tois)*, vol. 21, no. 4, pp. 315–346, 2003.

[51] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," *arXiv preprint cs/0205070*, 2002.

[52] M. V. Mäntylä, D. Graziotin, and M. Kuutila, "The evolution of sentiment analysis—a review of research topics, venues, and top cited papers," *Computer Science Review*, vol. 27, pp. 16–32, 2018.

[53] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N project report, Stanford*, vol. 1, no. 12, p. 2009, 2009.

[54] Y. Inc, "Yelp reviews dataset," https://www.yelp.com/academic dataset, 2014.

[55] R. Lamsal, "Design and analysis of a large-scale covid-19 tweets dataset," *Applied Intelligence*, pp. 1–15, 2020.

[56] E. Ferrara, "The history of digital spam," *Communications of the ACM*, vol. 62, no. 8, pp. 82–91, 2019.

[57] E. G. Dada, J. S. Bassi, H. Chiroma, A. O. Adetunmbi, O. E. Ajibuwa *et al.*, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, p. e01802, 2019.

[58] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes?" in *CEAS*, vol. 17. Mountain View, CA, 2006, pp. 28–69.

[59] S. A. Project, "Spam assassin public corpus," https://spamassassin.apache.org/publiccorpus/, 2015.

[60] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.

[61] H. Gong, Y. Li, S. Bhat, and P. Viswanath, "Context-sensitive malicious spelling error correction," in *The World Wide Web Conference*, 2019, pp. 2771–2777.

[62] A. P. Pimpalkar and R. J. R. Raj, "Influence of pre-processing strategies on the performance of ml classifiers exploiting tf-idf and bow features," *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 9, no. 2, pp. 49–68, 2020.

[63] A. C. M. V. Srinivas, C. Satyanarayana, C. Divakar, and K. P. Sirisha, "Sentiment analysis using neural network and lstm," in *IOP Conference Series: Materials Science and Engineering*, vol. 1074, no. 1. IOP Publishing, 2021, p. 012007.

[64] M. S. Başarslan and F. Kayaalp, "Sentiment analysis on social media reviews datasets with deep learning approach," *Sakarya University Journal of Computer and Information Sciences*, vol. 4, no. 1, pp. 35–49, 2021.

[65] A. Barushka and P. Hajek, "Spam filtering using integrated distribution-based balancing approach and regularized deep neural networks," *Applied Intelligence*, vol. 48, no. 10, pp. 3538–3556, 2018.

[66] G. Chetty, H. Bui, and M. White, "Deep learning based spam detection system," in *2019 International Conference on Machine Learning and Data Engineering (iCMLDE)*. IEEE, 2019, pp. 91–96.

[67] A. Laucuka *et al.*, "Communicative functions of hashtags," *Economics and culture*, vol. 15, no. 1, pp. 56–62, 2018.

[68] E. S. Alamoudi and N. S. Alghamdi, "Sentiment classification and aspect-based sentiment analysis on yelp reviews using deep learning and word embeddings," *Journal of Decision Systems*, pp. 1–23, 2021.

[69] D. Hankamer and D. Liedtka, "Twitter sentiment analysis with emojis."

[70] W. Etaiwi and G. Naymat, "The impact of applying different preprocessing steps on review spam detection," *Procedia computer science*, vol. 113, pp. 273–279, 2017.

[71] S. Srinivasan, V. Ravi, M. Alazab, S. Ketha, A.-Z. Ala'M, and S. K. Padannayil, "Spam emails detection based on distributed word embedding with deep learning," in *Machine Intelligence and Big Data Analytics for Cybersecurity Applications*. Springer, 2021, pp. 161–189.

[72] D. M. Christopher, R. Prabhakar, and S. Hinrich, "Introduction to information retrieval," pp. 32–31, 2008.

[73] A. K. Uysal and S. Gunal, "The impact of preprocessing on text classification," *Information processing & management*, vol. 50, no. 1, pp. 104–112, 2014.

[74] P. Cunningham, "Dimension reduction," in *Machine learning techniques for multimedia*. Springer, 2008, pp. 91–112.

[75] R. Ahuja, A. Chug, S. Kohli, S. Gupta, and P. Ahuja, "The impact of features extraction on the sentiment analysis," *Procedia Computer Science*, vol. 152, pp. 341–348, 2019.

[76] A. Madasu and S. Elango, "Efficient feature selection techniques for sentiment analysis," *Multimedia Tools and Applications*, vol. 79, no. 9, pp.

6313–6335, 2020.

[77] T.-I. F. Abidemi and T. N. Toyin, "Feature extraction for sms spam detection."

[78] M. A. Hassan and N. Mtetwa, "Feature extraction and classification of spam emails," in *2018 5th International Conference on Soft Computing & Machine Intelligence (ISCMI)*. IEEE, 2018, pp. 93–98.

[79] A. Bhoi and S. Joshi, "Various approaches to aspect-based sentiment analysis," *arXiv preprint arXiv:1805.01984*, 2018.

[80] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[81] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[82] C. Sammut and G. I. Webb, Eds., *TF–IDF*. Boston, MA: Springer US, 2010, pp. 986–987. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_832

[83] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[84] A. I. Kadhim, "Term weighting for feature extraction on twitter: A comparison between bm25 and tf-idf," in *2019 international conference on advanced science and engineering (ICOASE)*. IEEE, 2019, pp. 124–128.

[85] H. H. Do, P. Prasad, A. Maag, and A. Alsadoon, "Deep learning for aspect-based sentiment analysis: a comparative review," *Expert Systems with Applications*, vol. 118, pp. 272–299, 2019.

[86] N. C. Dang, M. N. Moreno-García, and F. De la Prieta, "Sentiment analysis based on deep learning: A comparative study," *Electronics*, vol. 9, no. 3,

p. 483, 2020.

[87] P. Hajek, A. Barushka, and M. Munk, "Fake consumer review detection using deep neural networks integrating word embeddings and emotion mining," *Neural Computing and Applications*, vol. 32, no. 23, pp. 17 259–17 274, 2020.

[88] A. Barushka and P. Hajek, "Review spam detection using word embeddings and deep neural networks," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 2019, pp. 340–350.

[89] M. Gupta and B. Gupta, "A comparative study of breast cancer diagnosis using supervised machine learning techniques," in *2018 second international conference on computing methodologies and communication (ICCMC)*. IEEE, 2018, pp. 997–1002.

[90] C. N. Kamath, S. S. Bukhari, and A. Dengel, "Comparative study between traditional machine learning and deep learning approaches for text classification," in *Proceedings of the ACM Symposium on Document Engineering 2018*, 2018, pp. 1–11.

[91] A. M. Ramadhani and H. S. Goo, "Twitter sentiment analysis using deep learning methods," in *2017 7th International annual engineering seminar (InAES)*. IEEE, 2017, pp. 1–4.

[92] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daumé III, "Deep unordered composition rivals syntactic methods for text classification," in *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, 2015, pp. 1681–1691.

[93] S. Das and A. K. Kolya, "Predicting the pandemic: sentiment evaluation and predictive analysis from large-scale tweets on covid-19 by deep convolutional neural network," *Evolutionary Intelligence*, pp. 1–22, 2021.

[94] P. K. Roy, J. P. Singh, and S. Banerjee, "Deep learning to filter sms spam," *Future Generation Computer Systems*, vol. 102, pp. 524–533, 2020.

[95] A. U. Rehman, A. K. Malik, B. Raza, and W. Ali, "A hybrid cnn-lstm model for improving accuracy of movie reviews sentiment analysis," *Multimedia Tools and Applications*, vol. 78, no. 18, pp. 26 597–26 613, 2019.

[96] R. Taheri and R. Javidan, "Spam filtering in sms using recurrent neural networks," in *2017 Artificial Intelligence and Signal Processing Conference (AISP)*. IEEE, 2017, pp. 331–336.

[97] L. Kurniasari and A. Setyanto, "Sentiment analysis using recurrent neural network," in *Journal of Physics: Conference Series*, vol. 1471, no. 1. IOP Publishing, 2020, p. 012018.

[98] A. Siji, N. Sheena, and N. Suresh Kumar, "A state of art: Machine learning approaches in email spam filters."

[99] A. Yadav and D. K. Vishwakarma, "Sentiment analysis using deep learning architectures: a review," *Artificial Intelligence Review*, vol. 53, no. 6, pp. 4335–4385, 2020.

[100] D. Tang, B. Qin, and T. Liu, "Deep learning for sentiment analysis: successful approaches and future challenges," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 6, pp. 292–303, 2015.

[101] S. Gogineni and A. Pimpalshende, "Predicting imdb movie rating using deep learning," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 2020, pp. 1139–1144.

[102] 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*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[103] W. Widayat, T. B. Adji, and Widyawan, "The effect of embedding dimension reduction on increasing lstm performance for sentiment analysis," in *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2018, pp. 287–292.

[104] J. Blitzer, M. Dredze, and F. Pereira, "Biographies, bollywood, boomboxes and blenders: Domain adaptation for sentiment classification," in *Proceedings of the 45th annual meeting of the association of computational linguistics*, 2007, pp. 440–447.

[105] S. Das and A. K. Kolya, "Predicting the pandemic: sentiment evaluation and predictive analysis from large-scale tweets on covid-19 by deep convolutional neural network," *Evolutionary Intelligence*, pp. 1–22, 2021.

[106] A. C. M. V. Srinivas, C. Satyanarayana, C. Divakar, and K. P. Sirisha, "Sentiment analysis using neural network and lstm," in *IOP Conference Series: Materials Science and Engineering*, vol. 1074, no. 1. IOP Publishing, 2021, p. 012007.

[107] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N project report, Stanford*, vol. 1, no. 12, p. 2009, 2009.

[108] E. S. Alamoudi and N. S. Alghamdi, "Sentiment classification and aspect-based sentiment analysis on yelp reviews using deep learning and word embeddings," *Journal of Decision Systems*, pp. 1–23, 2021.

[109] Y. Inc, "Yelp reviews dataset," https://www.yelp.com/academic dataset, 2014.

[110] A. Barushka and P. Hajek, "Spam filtering using integrated distribution-based balancing approach and regularized deep neural networks," *Applied Intelligence*, vol. 48, no. 10, pp. 3538–3556, 2018.

[111] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes?" in *CEAS*, vol. 17. Mountain View, CA,

2006, pp. 28–69.

[112] R. Taheri and R. Javidan, "Spam filtering in sms using recurrent neural networks," in *2017 Artificial Intelligence and Signal Processing Conference (AISP)*. IEEE, 2017, pp. 331–336.

[113] A. T. A. and J. M. G. Hidalgo., "Sms spam collection," http://www.dt.fee.unicamp.br/ tiago/smsspamcollection/., 2018.

[114] M. H. Lee *et al.*, "A study of efficiency information filtering system using one-hot long short-term memory," *International Journal of Advanced Culture Technology*, vol. 5, no. 1, pp. 83–89, 2017.

[115] S. A. Project, "Spam assassin public corpus," https://spamassassin.apache.org/publiccorpus/, 2015.

[116] T. C. Alberto, J. V. Lochter, and T. A. Almeida, "Tubespam: Comment spam filtering on youtube," in *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE, 2015, pp. 138–143.

[117] V. Nasir and F. Sassani, "A review on deep learning in machining and tool monitoring: methods, opportunities, and challenges," *The International Journal of Advanced Manufacturing Technology*, pp. 1–27, 2021.

[118] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 843–852.

[119] Á. Casado-García, C. Domínguez, M. García-Domínguez, J. Heras, A. Inés, E. Mata, and V. Pascual, "Clodsa: a tool for augmentation in classification, localization, detection, semantic segmentation and instance segmentation tasks," *BMC bioinformatics*, vol. 20, no. 1, pp. 1–14, 2019.

[120] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[121] C. Zhang, P. Zhou, C. Li, and L. Liu, "A convolutional neural network for leaves recognition using data augmentation," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE, 2015, pp. 2143–2150.

[122] K. K. Pal and K. S. Sudeep, "Preprocessing for image classification by convolutional neural networks," in *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, 2016, pp. 1778–1781.

[123] P. Sharma, P. Hans, and S. C. Gupta, "Classification of plant leaf diseases using machine learning and image preprocessing techniques," in *2020 10th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 2020, pp. 480–484.

[124] A. Rodríguez-Cristerna, C. P. Guerrero-Cedillo, G. A. Donati-Olvera, W. Gómez-Flores, and W. C. A. Pereira, "Study of the impact of image preprocessing approaches on the segmentation and classification of breast lesions on ultrasound," in *2017 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, 2017, pp. 1–4.

[125] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[126] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[127] ——, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[128] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference*

on computer vision and pattern recognition, 2017, pp. 4700–4708.

[129] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[130] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[131] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[132] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[133] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[134] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[135] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[136] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[137] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[138] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[139] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[140] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[141] P. Hurtik, V. Molek, J. Hula, M. Vajgl, P. Vlasanek, and T. Nejezchleba, "Poly-yolo: higher speed, more precise detection and instance segmentation for yolov3," *arXiv preprint arXiv:2005.13243*, 2020.

[142] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid *et al.*, "Fiji: an open-source platform for biological-image analysis," *Nature methods*, vol. 9, no. 7, pp. 676–682, 2012.

[143] M. Gerber and N. Pillay, "Quantifying common rust severity in maize using mask r-cnn and genetic algorithms for hyper-parameter tuning." in *The 2021 International Joint Conference on Neural Networks*, 2021.

[144] A. A and A. B, "A comparative study of supervised and unsupervised neural networks for oral lesion detection," in *The 2021 IEEE Symposium Series on Computational Intelligence*, 2021, p. Under review.

[145] H. S. Chandrashekar, A. Geetha Kiran, M. S, M. Dinesh, and B. Nanditha, "Oral images dataset,"

https://data.mendeley.com/datasets/mhjyrn35p4/2, 2021.

[146] R. Khamissa, "University of pretoria - oral images dataset," https://example.com, 2021.

[147] T. Wu, S. Wen, Y. Xiang, and W. Zhou, "Twitter spam detection: Survey of new approaches and comparative study," *Computers & Security*, vol. 76, pp. 265–284, 2018.

[148] R. Liu, Y. Shi, C. Ji, and M. Jia, "A survey of sentiment analysis based on transfer learning," *IEEE Access*, vol. 7, pp. 85 401–85 412, 2019.

[149] M. M. Mirończuk and J. Protasiewicz, "A recent overview of the state-of-the-art elements of text classification," *Expert Systems with Applications*, vol. 106, pp. 36–54, 2018.

[150] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[151] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[152] F. Sultana, A. Sufian, and P. Dutta, "Evolution of image segmentation using deep convolutional neural network: a survey," *Knowledge-Based Systems*, vol. 201, p. 106062, 2020.

[153] B. Pandey, D. K. Pandey, B. P. Mishra, and W. Rhmann, "A comprehensive survey of deep learning in the field of medical imaging and medical natural language processing: Challenges and research directions," *Journal of King Saud University-Computer and Information Sciences*, 2021.

[154] E.-G. Talbi, "Automated design of deep neural networks: A survey and unified taxonomy," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–37, 2021.

[155] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[156] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning*. Springer, Cham, 2019, pp. 3–33.

[157] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[158] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[159] T. T. H. Dinh, T. H. Chu, and Q. U. Nguyen, "Transfer learning in genetic programming," in *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2015, pp. 1145–1151.

[160] D. Whitley and J. P. Watson, "Complexity theory and the no free lunch theorem," *Search methodologies*, pp. 317–339, 2005.

[161] J. D. Romano, T. T. Le, W. Fu, and J. H. Moore, "Tpot-nn: Augmenting tree-based automated machine learning with neural network estimators," *Genetic Programming and Evolvable Machines*, vol. 22, no. 2, pp. 207–227, 2021.

[162] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231216315533

[163] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv preprint arXiv:2003.05689*, 2020.

[164] B. Wang, H. Xu, J. Zhang, C. Chen, X. Fang, Y. Xu, N. Kang, L. Hong, C. Jiang, X. Cai *et al.*, "Vega: towards an end-to-end configurable automl pipeline," *arXiv preprint arXiv:2011.01507*, 2020.

[165] N. Pillay and T. Nyathi, "Automated design of classification algorithms," in *Automated Design of Machine Learning and Search Algorithms*. Springer, 2021, pp. 171–184.

[166] Y.-W. Chen, Q. Song, and X. Hu, "Techniques for automated machine learning," *ACM SIGKDD Explorations Newsletter*, vol. 22, no. 2, pp. 35–50, 2021.

[167] N. Pillay, R. Qu, D. Srinivasan, B. Hammer, and K. Sorensen, "Automated design of machine learning and search algorithms [guest editorial]," *IEEE Computational intelligence magazine*, vol. 13, no. 2, pp. 16–17, 2018.

[168] H. Andersen, S. Stevenson, T. Ha, X. Gao, and B. Xue, "Evolving neural networks for text classification using genetic algorithm-based approaches," in *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2021, pp. 1241–1248.

[169] M. van Knippenberg, V. Menkovski, and S. Consoli, "Evolutionary construction of convolutional neural networks," in *International Conference on Machine Learning, Optimization, and Data Science*. Springer, 2018, pp. 293–304.

[170] B. Guo, J. Hu, W. Wu, Q. Peng, and F. Wu, "The tabu_genetic algorithm: a novel method for hyper-parameter optimization of learning algorithms," *Electronics*, vol. 8, no. 5, p. 579, 2019.

[171] L. K. Ramasamy, S. Kadry, and S. Lim, "Selection of optimal hyper-parameter values of support vector machine for sentiment analysis tasks using nature-inspired optimization methods," *Bulletin of Electrical Engineering and Informatics*, vol. 10, no. 1, pp. 290–298, 2021.

[172] P. M. Chanal, M. S. Kakkasageri, and S. K. S. Manvi, "Security and privacy in the internet of things: computational intelligent techniques-based approaches," in *Recent Trends in Computational Intelligence Enabled Research*. Elsevier, 2021, pp. 111–127.

[173] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[174] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[175] W. Zhang, L. Deng, L. Zhang, and D. Wu, "Overcoming negative transfer: A survey," *arXiv preprint arXiv:2009.00909*, 2020.

[176] L. Torrey and J. Shavlik, *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques*. IGI Global, 2010.

[177] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural automl," *arXiv preprint arXiv:1803.02780*, 2018.

[178] D. Yogatama and G. Mann, "Efficient transfer learning method for automatic hyperparameter tuning," in *Artificial intelligence and statistics*. PMLR, 2014, pp. 1077–1085.

[179] M. A. Ardeh, Y. Mei, and M. Zhang, "Transfer learning in genetic programming hyper-heuristic for solving uncertain capacitated arc routing problem," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 49–56.

[180] L. Muñoz, L. Trujillo, and S. Silva, "Transfer learning in constructive induction with genetic programming," *Genetic Programming and Evolvable Machines*, vol. 21, no. 4, pp. 529–569, 2020.

[181] T. Chugh, M. Singh, S. Nagpal, R. Singh, and M. Vatsa, "Transfer learning based evolutionary algorithm for composite face sketch recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 117–125.

[182] G.-S. Hao, G.-G. Wang, Z.-J. Zhang, and D.-X. Zou, "Optimisation of the high-order problems in evolutionary algorithms: an application of trans-

fer learning," *International Journal of Wireless and Mobile Computing*, vol. 14, no. 1, pp. 56–63, 2018.

[183] C. Johnson, "What is research in computing science," *Computer Science Dept., Glasgow University. Electronic resource: http://www. dcs. gla. ac. uk/johnson/teaching/research_skills/research. ht ml*, 2006.

[184] Z. Hussain, F. Gimenez, D. Yi, and D. Rubin, "Differential data augmentation techniques for medical imaging classification tasks," in *AMIA annual symposium proceedings*, vol. 2017. American Medical Informatics Association, 2017, p. 979.

[185] S. O'Gara and K. McGuinness, "Comparing data augmentation strategies for deep image classification," in *Irish Machine Vision and Image Processing Conference (IMVIP)*, 2019.

[186] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[187] M. Misir, K. Verbeeck, P. De Causmaecker, and G. V. Berghe, "Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[188] D. S. Sachan, M. Zaheer, and R. Salakhutdinov, "Revisiting lstm networks for semi-supervised text classification via mixed objective function," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, p. 6940–6948, Jul 2019. [Online]. Available: http://dx.doi.org/10.1609/aaai.v33i01.33016940

[189] L. Xiao, H. Zhang, W. Chen, Y. Wang, and Y. Jin, "Transformable convolutional neural network for text classification." in *IJCAI*, 2018, pp. 4496–4502.

[190] K.-P. Lai, W. Lam, and J. C. Ho, "Domain-aware recurrent neural network for cross-domain sentiment classification," in *Proceedings of the 3rd International Conference on Data Science and Information Technology*, 2020, pp. 181–185.

[191] K. Chakraborty, S. Bhatia, S. Bhattacharyya, J. Platos, R. Bag, and A. E. Hassanien, "Sentiment analysis of covid-19 tweets by deep learning classifiers—a study to show how popularity is affecting accuracy in social media," *Applied Soft Computing*, vol. 97, p. 106754, 2020.

[192] E. Hamdi, S. Rady, and M. Aref, "A deep learning architecture with word embeddings to classify sentiment in twitter," in *International Conference on Advanced Intelligent Systems and Informatics*. Springer, 2020, pp. 115–125.

[193] A. K. Uysal, "Feature selection for comment spam filtering on youtube," *Data Science and Applications*, vol. 1, no. 1, pp. 4–8, 2018.

Add appendix content here