DURBAN UNIVERSITY OF TECHNOLOGY

Learning Rate Optimisation of an Image Processing Deep Convolutional Neural Network

By

Sibusiso Blessing Buthelezi

Student Number: 21856164

A dissertation submitted in fulfilment of
the requirements for the degree of
Master of Engineering

Department of Electronics and Computer Engineering
Faculty of Engineering and the Built Environment

Supervisor: Dr S. Reddy

Co-Supervisor: Prof B. Twala

2020

DECLARATION

This dissertation is the student's own work. every cited work or text has been properly referenced. It has not been partially or fully submitted at any other University.

This research was duly supervised by Dr S. Reddy and Prof B. Twala at the Durban University of Technology.

Submitted by:

…… …… 22/10/2020..

Sibusiso Blessing Buthelezi Date

Student Number: 21856164

Approved for Final Submission by:

…… …… 03/11/2021

Supervisor: Dr S. Reddy Date

…… …03/11/2021 03/11/2021

Co-Supervisor: Prof B. Twala Date

ACKNOWLEDGEMENT

I would like to thank my supervisor. Dr S. Reddy. for his remarkable guidance. wisdom. and contribution to the thesis. I would also like to expand my gratitude to my co-supervisor Prof Twala for his input and guidance in this research. My deepest appreciation to Mrs Chetty for her assistance through the course of this research. Lastly, I am grateful to those who contributed directly and indirectly to the completion of this thesis.

ABSTRACT

The major contribution of this dissertation is the proposal of the use of mathematical models to identify an optimal learning rate for an image processing deep convolutional neural network (DCNN). This model is derived from a nonlinear regression relationship between the learning rate and the accuracy of a test DCNN model. This relationship is meant to (A) resolve the problem of arbitrarily selecting the initial learning rate (B) reduce computational resource requirement and (C) reduce training instabilities. An algorithm is developed to analyse an inputted DCNN model and subsequently render output parameters that may be used to aid in the selection of an OLR.

The benefit of an OLR includes improved training stability and reduced computational resources. The results rendered by the OLR algorithm proposes that an optimal learning rate improves model performance; this is described by the test model average accuracy of 91%. Furthermore, a model validation graph is also extrapolated. which will illustrate the mathematical model accuracy and the region of interest (ROI). The ROI defines the region in the learning rate spectrum with a positive effect on model performance.

DECLARATION OF RESEARCH WORK PUBLICATION

The work on optimising the learning rate for an image processing deep convolutional neural network has been published in partnership with the Durban University of Technology.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Term	Definition
η	Learning Rate
$\nabla C(w)$	Model Cost Function
w_i	Model Weights Parameter
η_{min}	Minimum Learning Rate
η_{max}	Maximum Learning Rate
η_0	Initial Learning Rate
$\sigma(x)$	Activation Function
f	Probability of 0.5
τ	Search Threshold
$m_i^{'}$	First Momentum
$v_i^{'}$	Second Momentum
$\epsilon^{'}$	Exponential Decay Rate
ABP	Batch ABP Algorithm
ALRAO	All Learning Rates At Once
ALRC	Adaptive Learning Rate Clipping
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
COVID 19	Corona Virus Disease of 2019
CPU	Central Processing Unit
DCNN	Deep Convolutional Neural Network
DNN	Deep Neural Network
GPU	Graphic Processing Unit
LOG-BP	Learning Rate Optimizing Genetic Back-Propagation
LR	Learning Rate
OLR	Optimal Learning Rate
RAM	Random Access Memory
ReLU	Rectified Linear Function
RGB	Red Green Blue
ROI	Region of Interest
SGD	Stochastic Gradient descent
SSD	Solid State Drive
STC	Search Then Converged

1 CHAPTER 1: INTRODUCTION

A deep convolutional neural network (DCNN) [1, 2] is a subset of deep learning that provides systems with the ability to learn graphical data and improve on the experience [3]. DCNN models are designed to mimic how the human visual cortex processes visual information. The learning process is accomplished by propagating data through the model several times until the model is optimised to classify data of similar characteristics [4]. The speed at which the model learns is determined by the learning rate hyperparameter. The learning rate is a critical training parameter that is used to finetune the performance of the model. Various techniques may be implemented to identify an ideal learning rate for training a model. These techniques usually incorporate the concept of a dynamic learning rate that changes throughout training [5]. Two of the most common techniques include learning rate scheduling [6] and adaptive learning rate [7]. The former computes the learning rate as a monotonic function that decreases after a given set of training data. and the latter computes the learning rate as a function of the model cost function concerning the model parameters [8]. Although these techniques have been proven to render acceptable results. they suffer from several problems when implemented.

A scheduled learning rate is a function of a monotonic function, meaning the learning rate will continuously decrease throughout training [9]. The problem with this technique is the open-loop nature of the learning rate, whereby the learning rate changes irrespective of the impact on the model undergoing training. The second problem is the selection of the initial learning rate to train the model before the monotonic function executes. Adaptive learning rate complements a scheduled learning rate by introducing a closed-loop into the adaptation. In an adaptive learning rate algorithm, the learning rate is a function of the model status while the training is in progress [10]. The learning rate is updated based on the status of the model in contrast to a scheduled learning rate.

An adaptive learning rate is plagued with several problems that may reduce the practicality of a DCNN model. An adaptive learning rate consists of several cost function gradients that must be stored in system memory to update all model parameters. The learning rate is also different for all model parameters and subsequently, more computational units are required to cope with the immense amount of computations. An adaptive learning rate also suffers from an arbitrarily selected initial learning rate to initialise the training session. The proposed technique implements a static constant learning rate [11] for all model parameters; the learning rate is optimised as an average of all model parameters. An optimal learning rate is meant to resolve most of the problems constrained to the other optimisation techniques.

In industry, there is now a direct correlation between machine learning and profit. An inefficient system may result in unplanned losses that were not initially estimated. Training a system model with a randomly selected learning rate may require system hardware upgrades such as high-end graphical processing units (GPU). multi-core central processing units (CPU), more random access memory (RAM), more cooling systems. and increased server space. These problems may be resolved by implementing an optimal learning rate that requires less computation to achieve the project objective.

In the commercial sector, more end-user applications implement machine learning systems such as voice assistance, facial recognition, data inscription, and graphic processing social media. Most of these applications run on smartphones with limited system memory, low powered processing units and limited battery life. Training machine learning applications to meet an end-users daily routine requires highly efficient resource allocation. Using an inefficient learning rate may result in a smartphone overheating, running low on memory and conflicting with the user experience. An optimal learning rate will ensure minimum resource usage over a wide range of applications. This has the potential to render high model accuracy at low resource requirements.

Learning rate optimisation may also be implemented in the educational sector to illustrate how system performance may be improved by manipulating the learning rate hyper-parameter. Before engaging in advanced learning rate optimisation techniques, an individual may initially be introduced to a constant optimal learning rate. This learning rate may provide the basis for training systems that implement machine learning models.

A DCNN model comprises two interconnected layers working in conjunction to classify the input data. The first layer is referred to as the convolutional layer [12, 13], this layer facilitates the extraction of essential features from the input data, these features may include edges, texture, and gradient. The second layer is referred to as a fully connected deep neural network [12, 14, 15] and is responsible for defining the input data as a distribution of output probabilities. This is illustrated in Figure 1.1 where the kernel matrix represents the convolutional layer and the web of interconnected neurons, indicated by circles, represents the fully connected layer. Features are extracted in the convolution layers and are subsequently classified in the fully connected layer.

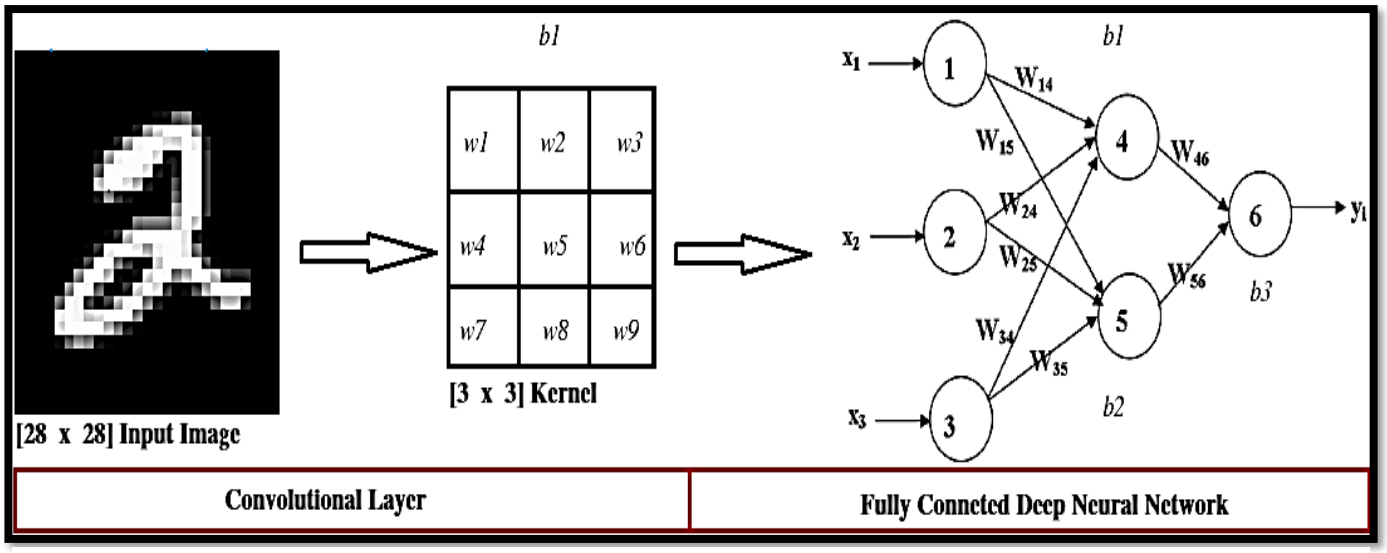


FIGURE 1.1: DEEP CONVOLUTIONAL NEURAL NETWORK (DCNN) ARCHITECTURE [16]

The two model layers consist of parameters that are optimised to improve the model performance, these parameters are referred to as weight w_i and bias b_i [2-4]. Initially, These parameters are assigned values randomly, however, as the model undergoes training these parameters are optimised to better classify the training data. The model parameters are trained by a stochastic gradient descent (SGD) algorithm [17-19] that implements the model negative gradient [20] and the learning rate hyperparameter [11, 21, 22]. The SGD algorithm is given by Equation 1.1.

$$w_{i+1} = w_i - \eta \nabla C(w_i). \qquad (1.1)$$

Where w_{i+1} denote the weight value for the next iteration, w_i denote the current weight value, η is the learning rate and $-\nabla C(w_i)$ is the negative cost function gradient [20] concerning the model parameters w_i and b_i. The SGD algorithm is built on top of the backpropagation algorithm [15, 23]. which is responsible for computing the negative gradient $-\nabla C(w_i)$ of the cost function concerning the model parameters. During training, data is propagated forward through the model and the model performance is measured as a cost function $C(w_i)$. The backpropagation algorithm computes the gradient that will minimise the magnitude of the cost function for all model parameters $-\nabla C(w_i)$. The magnitude of the gradient is usually too small to contribute any significant optimisation to the weight and bias constants, as such the gradient is amplified by the learning rate hyperparameter. The learning rate is meant to aid the optimisation of the model parameters by manipulating the size of the negative gradient $-\eta \nabla C(w_i)$. Figure 1.2 illustrates the manner at which the model is optimised; initially the model cost function is very large owing to the randomly initialised model parameters. During training,

the position of the model in the cost function plane is gradually reduced until the model position is at the absolute minimum.

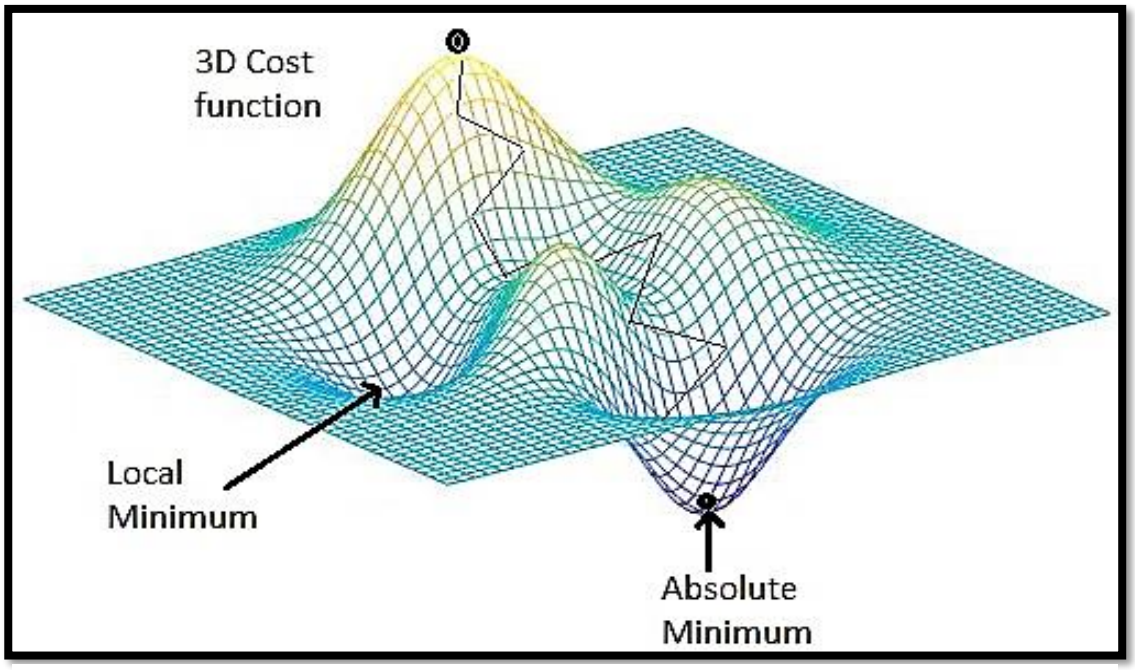


FIGURE 1.2: MODEL COST FUNCTION [24]

The learning rate determines the speed at which the model will learn for a given single iteration of training data. Model training is meant to optimise the model parameters, consequently, the learning rate that aids the optimisation process should also be optimised to improve the training session. An inefficient learning rate has the potential to degrade the performance of the model, whilst an optimal learning rate has the benefit of improving the model performance and provides training stability. The goal is to establish an optimal learning rate to exploit the benefit of improved model accuracy and training stability.

1.1 RESEARCH HYPOTHESIS

The proposed research seeks to answer the question "What is an optimal learning rate for an image processing Deep Convolutional Neural Network (DCNN)?" This arises as a result of the overhead computational resource requirements associated with other learning rate optimisation algorithms that arbitrarily select the learning rate. The learning rate is a critical hyperparameter and an optimal learning rate for an image processing DCNN should be selected systematically based on mathematical evidence.

1.2 RESEARCH OBJECTIVES

1. Develop an algorithm to analyse a DCNN model within the predefined range of learning rates bounded between $[\eta_{min} = 0.001;\ \eta_{max} = 0.1]$.
2. Generate an optimal learning rate for an image processing DCNN.
3. Formulate a mathematical model, which describes the relationship between the model accuracy and learning rate.
4. Plot a model validation graph that illustrates the model performance over the predefined range of learning rates $[\eta_{min} = 0.001;\ \eta_{max} = 0.1]$.

1.3 METHODOLOGY OVERVIEW

The research objectives are realised through quantitative experimentation using an image processing DCNN as the base model. The experiments comprise the learning rate as an independent variable and the model accuracy as a dependent variable. The objective of the experiments is to observe the relationship between the learning rate and model accuracy. The base model will accept images of handwritten digits; these are separated into two categories viz, (i) training and (ii) test or validation data. The base model will learn by processing the training data and the model accuracy will be evaluated using the testing data. The research has been divided into three stages; this includes developing an algorithm, performing learning rate experiments. and system modelling. This is illustrated in Figure 1.3 below.

FIGURE 1.3: PROJECT OVERVIEW STRUCTURE

1.3.1 STAGE 1 (DEVELOPING AN ALGORITHM)

The first stage of the process will facilitate the development of the algorithm that will analyse the given input model. The algorithm will accept a DCNN model as an input argument and perform preliminary checks to evaluate the structure and the properties of the input model. If the input model meets the acceptance criteria the model is propagated to the next stage of the process and if the model fails the acceptance criteria the algorithm provides feedback and terminates the session.

1.3.2 STAGE 2 (LEARNING RATE EXPERIMENT)

The second stage of the process evaluates the input model from the preceding stage with the learning rate range of $[\eta_{min} = 0.001;\ \eta_{max} = 0.1]$. The input model is initially trained with the learning rate $\eta_{min} = 0.001$ and once training is completed the learning rate is recorded along with the respective model accuracy. The learning rate is then incremented by a factor of 0.0001 before the next iteration. The above process iterates several instances until the escape criterion is reached, the algorithm terminates the session once the learning rate has reached the uppermost limit of $\eta_{min} = 0.1$, the algorithm terminates the session and initialises the last stage of the process.

1.3.3 STAGE 3 (SYSTEM MODELLING)

The final stage of the process accepts the training data recorded from the preceding stage as an input argument to compile the output parameters. The algorithm compiles three output parameters, which include an array of the optimal learning rate, a mathematical model, and a visualisation graph. The optimal learning rate array will be populated with learning rate values resulting from the highest model accuracy. The mathematical model will describe the relationship between the learning rate and model accuracy. The visualisation graph will illustrate how best the mathematical model merges with the data obtained in stage 2 of this process. The algorithm will terminate all sessions once all output parameters have complied.

1.4 THESIS STRUCTURE

Chapter 2: Literature Review discusses previously proposed models and how these studies relate to the chosen research. This section is meant to provide the context for the proposed research. The literature review will be categorised into three subcategories and these include the constant learning rate, scheduled learning rate. and adaptive learning rate.

Chapter 3: Deep Convolutional Neural Network dissects the structure of a DCNN model and evaluates the basic building blocks of the model. The structure illustrates how data propagates forward through the model to perform classification tasks. A DCNN structure provides a comprehensive idea of the mechanics that allows the model to imitate human-like vision. The structure will also demonstrate the various parameters that are affected by the learning rate.

Chapter 4: Network Training (Back Propagation) extends the previous chapter and discusses the optimisation of the model parameters. This section focuses primarily on the SGD algorithm and how

backpropagation is implemented to update all model parameters. This chapter will also mention the effect of the learning rate on the algorithm and the model parameters.

Chapter 5: Learning Rate Optimisation Experiment conducts an experiment on the base model to establish an optimal learning rate. The model is trained over a range of learning rate values bounded by $[\eta_{min} = 0.001; \eta_{max} = 0.1]$. Subsequently, the results of the experiment are captured and used to derive a mathematical model that may aid the selection of an optimal learning rate. This section also discusses the methodology described by the OLR algorithm as well as the test model.

Chapter 6: Discussion objectively analyses the results obtained in the previous chapter and discusses the methodology, defines the results concerning the field of machine learning, and compares the proposed method with the existing body of knowledge. This section also discusses the limitations of the algorithm used to generate the output parameters.

Chapter 7: Conclusion summarises the study along with possible improvements on the existing results as well as the limitations of the research. This section provides recommendations for future research on the learning rate optimisation knowledge area.

2 CHAPTER 2: LITERATURE REVIEW

2.1 INTRODUCTION

This chapter discusses algorithms that have been proposed to optimise the learning rate of deep learning models [25, 26]. A deep learning model comprises several interconnected parameters referred to as weight and bias [27, 28]. During the training session, these parameters are optimised to increase model performance. The algorithm responsible for optimising model parameters is referred to as stochastic gradient descent (SGD) [18], which implements the negative cost function gradient [20] along with the learning rate hyperparameter [6, 29] to optimise model parameters. The learning rate plays an important role in optimising model parameters; as a result, an optimal learning rate has the potential to improve model performance.

In areas of machine learning [27], there have been several proposed algorithms that manipulate the learning rate to improve both model performance and the training session. Learning rate optimisation may be categorised into three areas; these include constant [30], scheduled [17], and adaptive learning rate [7]. These categories also define algorithms used in rendering optimal learning rates. A constant learning rate remains constant for all model parameters whilst training is in progress. A scheduled learning rate changes monotonically after a predefined duration or epoch. An adaptive learning rate changes based on model dynamics such as the gradient and momentum.

2.2 CONSTANT LEARNING RATE

2.2.1 OPTIMIZING NEURAL-NETWORK LEARNING RATE USING A GENETIC ALGORITHM

According to Kanada [31], the learning rate for any generic machine learning model may be optimised by combining a backpropagation algorithm [4] with a genetic algorithm. Here an optimal learning rate is meant to resolve the problem of bad learning rate scheduling [10] and controlling the locality phenomenon [32]. This phenomenon occurs when a model is unable to escape the local minimum of the cost function. Kanada [31] proposes a solution to the problem by using an algorithm referred to as learning rate optimizing genetic back-propagation (LOG-BP) that searches for an optimal learning rate. The algorithm returns an optimal learning rate as output and accepts a model as an input. The algorithm creates several threads of the same input model and trains these threads using different learning rates [33]. Once training has been terminated all model threads capture the model performance to evaluate the highest performing model. Model threads resulting in poor model performance are terminated whilst model threads resulting in high model performance are duplicated and assigned a new learning rate defined by a probability distribution of 0.5 of the last learning rate. Equation 2.1 below defines the next iteration learning rate.

$$\eta' = f\eta_0 \ (f \ is \ probability \ 0.5) \qquad (2.1)$$

The probability distribution equation above computes a new learning rate after a single training epoch, where η' refers to the new learning rate, f is the probability distribution and η_0 is the initial learning rate. The process is iterated several times until an optimal learning rate is obtained. The resulting learning rate is a constant value that allows the model to avoid locality and to improve the learning rate schedule. The results of the study are demonstrated in Figure 2.1 below.

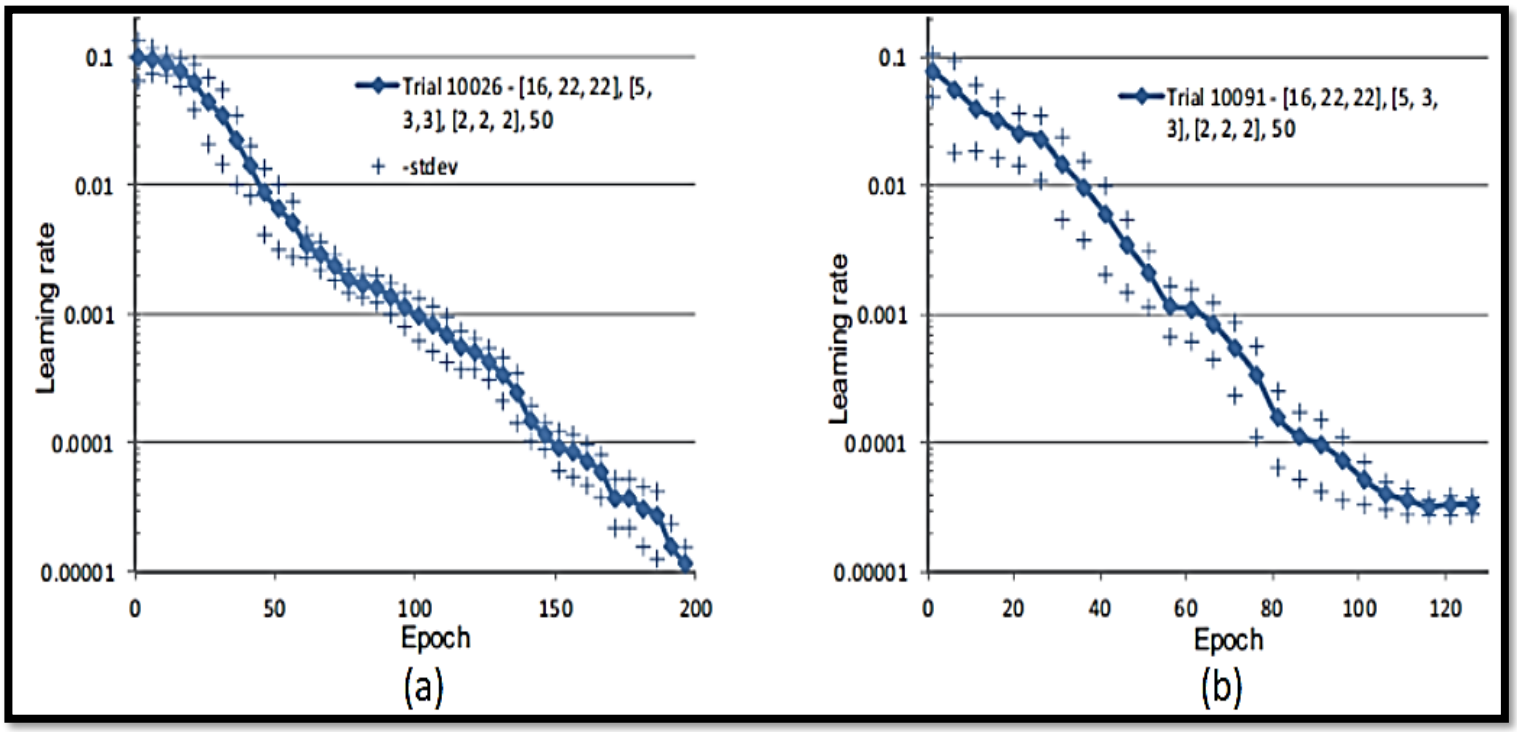


FIGURE 2.1: (A) LEARNING WITH 12 THREADS AND (B) LEARNING RATE WITH 24 THREADS

In Figure 2.1(a), the algorithm starts with an initial learning rate of 0.1 and 12 model threads. The algorithm trains the 12 model threads and terminates all model threads that perform poorly whilst duplicating the remaining threads. After 200 epochs the algorithm terminates and the optimal learning rate is observed to be 0.00001. Figure 2.1(b) represents the same algorithm consisting of 24 model threads instead of 12 threads. The results demonstrate that more threads result in faster learning rate searching. The 24 threads terminate after 120 epochs whilst 12 threads terminate after 200 epochs. More threads improve the speed of the algorithm.

Although the LOG-BP algorithm renders acceptable results, some drawbacks that may not be practical in some scenarios. The high number of threads created may result in high memory usage and a high

number of computational cores. The algorithm is said to require a high-end graphics processing unit (GPU) [34] with at least 12 gigabytes of fast speed GPU memory. The high specification requirement may not be feasible for small applications, as such a different algorithm may be used to obtain the same results but at lower resource requirements. The proposed OLR algorithm may provide a solution to the high computational resources required for generating an optimal learning rate. The OLR algorithm is expected to generate an optimal learning rate whilst utilising low computational as well as low memory resources.

2.2.2 ALL LEARNING RATES AT ONCE

Leonard et al, [35] developed an algorithm, referred to as *all learning rates at once* (ALRAO). to improve deep learning model performance. The ALRAO algorithm is meant to resolve the problem of new models that have not been trained and analysed. The performance of the algorithm is measured against an existing algorithm referred to as stochastic gradient descent (SGD)[18]. The ALRAO algorithm attempts to improve model performance by assigning all model parameters different learning rates within a predefined range of $[\eta_{min}; \eta_{max}]$. Prior to training the model, the ALRAO algorithm creates a vector of sampled log-uniformly within the range of $[\eta_{min} = 10^{-5}; \eta_{max} = 10]$.

During the training session. all model parameters are randomly assigned values contained within the created vector. The ALRAO algorithm was evaluated against the SGD algorithm. Whilst the ALRAO algorithm assigns all model parameters with different random learning rates, the SGD algorithm trains all model parameters with the same constant learning rate, which is optimised to reduce the model cost function. In the experiment, the SGD algorithm is trained with discrete predefined optimal learning rate within the range of $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1},\ 1,\ 10]$. The results of the ALRAO algorithm are tabulated and compared with the results obtained by the SGD algorithm. Table 2.1 below shows the results captured from both experiments.

TABLE 2.1: ALRAO COMPARED WITH THE SGD ALGORITHM

Model	SGD with optimal LR			ALRAO	
	LR	Loss	Accuracy %	Loss	Accuracy %
MobileNet	0.1	0.3	90.2	0.42	88.1
GoogLeNet	0.01	0.45	89.6	0.47	88.9

Table 2.1 shows that the SGD algorithm outperforms the ALRAO algorithm by a narrow margin. The experiment is considered a success owing to the random nature of the ALRAO algorithm learning rate

in contrast to the optimised SGD learning rate. The idea of training model parameters with different random learning rates results in noisy model performance.

The problem of arbitrarily selecting a learning rate may be resolved by initially defining the model performance against the learning rate. The relationship between the two parameters will allow for the selection of a range that will maximise the effectiveness of the learning rate on model parameters. The range of the learning rate is constrained between two extreme limits $[\eta_{min} = 10^{-5}; \eta_{max} = 10]$. The wide range results in model saturation towards the extreme limits.

The proposed OLR algorithm will provide a suitable range of optimal learning rates referred to as the region of interest. This range will result in learning rate values that cause more positive effects on model performance. The ALRAO algorithm may outperform the SGD algorithm by incorporating the region of interest as an integral part of the ALRAO algorithm. The resulting model performance may improve, and the ALRAO algorithm will be based on mathematical evidence. in contrast to intuition.

2.3 SCHEDULE LEARNING RATE

2.3.1 A NOVEL LEARNING RATE SCHEDULE IN OPTIMISATION FOR NEURAL NETWORKS

The schedule learning rate (SLR) [36] method involves the process of scheduling the decline of the learning rate as a function of time to improve the training session. SLRs usually adopt a monotonic function to compute the learning rate as a function of time or epoch [8]. According to Park et al. [37], the learning rate of an artificial neural network (ANN) model may be improved by scheduling the learning rate to change as the function of the cost function [20] for a given iteration. The next iteration learning rate is calculated using Equation 2.2 below. where η_{i+1} refers to the new learning rate, η_0 is the initial learning rate, and $C(w_n)$ is the cost function with respect to the weight in the n^{th} layer.

$$\eta_{i+1} = \eta_0 C(w_n) \tag{2.2}$$

The idea of a learning rate as a function of the cost function is meant to resolve problems such as the model locality [32]. The locality phenomenon occurs when the model is stuck at a local minimum instead of the absolute minimum of the cost function. When the learning rate is driven by the cost function. the model will continue to learn until the cost function is zero, defining the absolute minimum. The analogy of this setup may be understood as a closed-loop system [38], where the output is sent back to the input to improve the next cycle of the system.

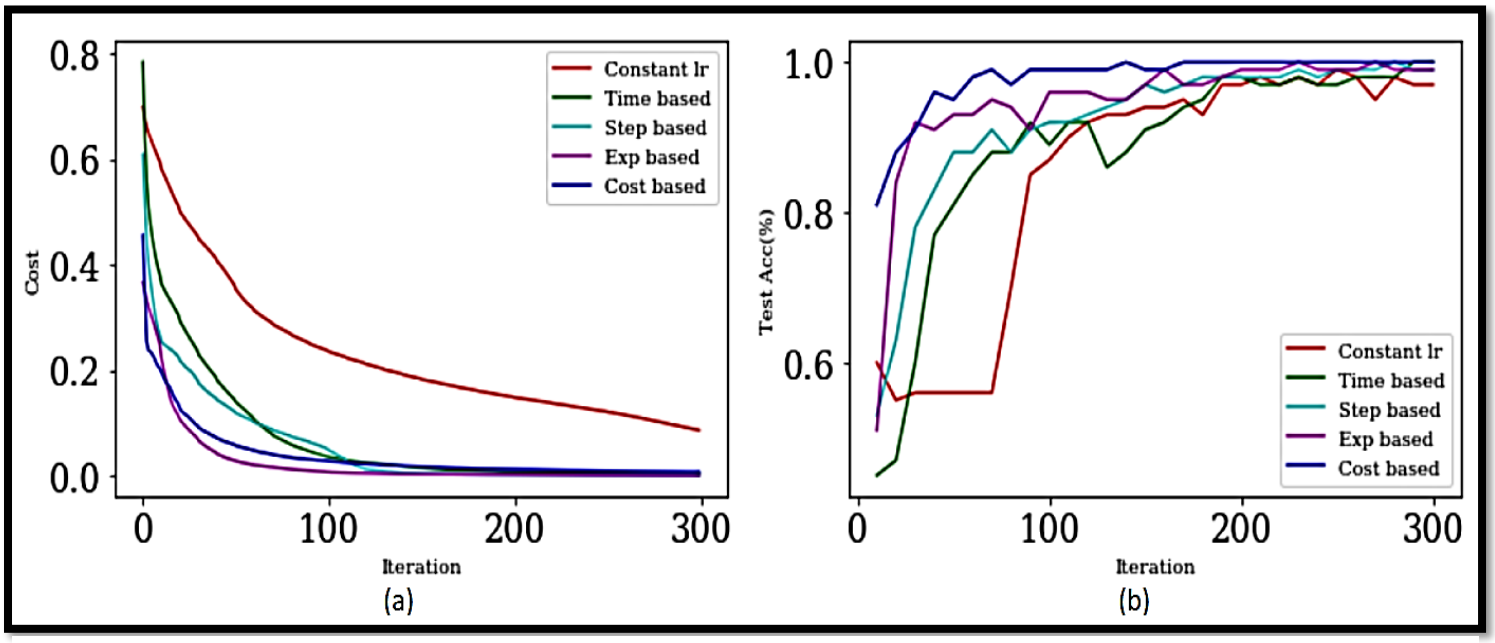


FIGURE 2.2: (A) COST FOR 5 ALGORITHMS (B) ACCURACY FOR THE 5 ALGORITHMS

The cost function based learning rate proposed by Park et al. [37] is compared with other learning rate optimisation techniques such as constant, schedule, and adaptive learning rates. The results of the experiments are illustrated in Figure 2.2 above.

In Figure 2.2(a) the cost function based learning rate for a binary classification task exceeds all tested algorithms except the exponential decay algorithm, Figure 2.2(b) illustrates the accuracy of the five algorithms. The cost function based learning rate algorithm appears to perform exponentially well during the early iterations while other algorithms seem to exhibit more noise in the validation plot. As the training session nears the final iterations, the other algorithms seem to recover and the overall performance for all algorithms seems to be bounded within a distribution of less than 10%.

The cost function based learning rate improves the model performance as evidenced in the above results, however, the equation that defines the cost function based learning rate is defined by an initial learning rate η_0, which is selected based on intuition. There does not appear to be any mathematical evidence for selecting the initial learning rate. The cost function based learning rate may be further improved by selecting an optimised η_0. An optimised η_0 will result in a faster cost function convergence, while obtaining early improved model validation. The proposed optimal learning rate algorithm may provide an optimal η_0 learning rate to further improve the model performance.

2.3.2 SEARCH-THEN-CONVERGE

Christian and Darken [6] propose a learning rate scheduling algorithm, referred to as *search then converged* (STC). that gradually changes the learning rate after every training iteration. The STC algorithm forces the learning rate to decrease gradually during the early stages of training and once a predefined threshold is reached the learning rate starts decreasing more rapidly. The concept behind the STC algorithm attempts to resolve the problem of locality [32] by manipulating the learning rate.

Initially, the model is trained using a high learning rate in hopes that it will eventually skip any local minimum points in the cost function; this is referred to as the *search* in the context of the STC algorithm. Once the *search* is completed the model converges to the absolute minimum of the cost function. The STC algorithm is defined by Equation 2.3 below.

$$\eta(t) = \frac{\eta_0}{(1+\frac{t}{\tau})}. \qquad (2.3)$$

Where $\eta(t)$ refers to the learning rate for the next iteration, η_0 is the initial learning rate constant, t is the number of iteration and τ is the search threshold. For values of $t < \tau$ the denominator is very small in magnitude, meaning the learning rate $\eta(t)$ will remain very large. Once $t > \tau$ the denominator will start contributing a large value to $\eta(t)$; this will cause the learning rate to be reduced more rapidly.

The result obtained in the study illustrates that the STC algorithm can yield results that exceed the constant learning rate [30] and the average schedule learning rate [39] models. Figure 2.3 illustrates the performance between the constant. average schedule and the STC algorithm learning rates.

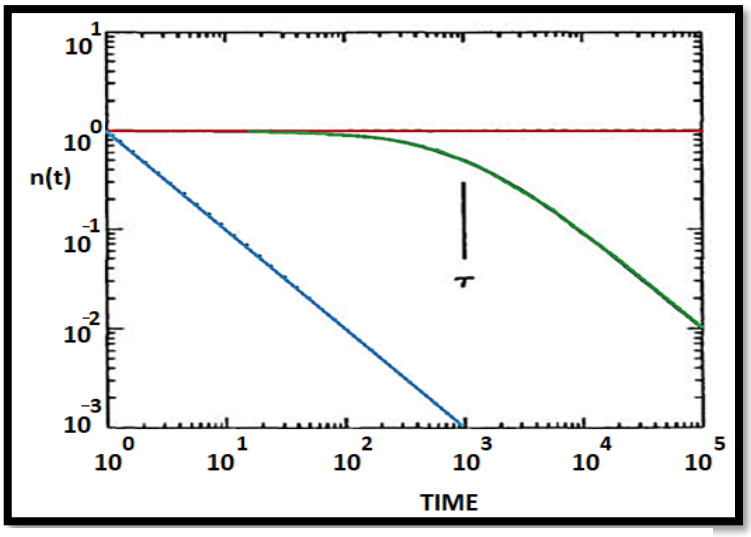


FIGURE 2.3: LEARNING RATE WITH RESPECT TO TIME

The plot illustrates how the learning rate changes with time for the three algorithms; red denotes the constant learning rate, green the STC algorithm learning rate. and blue denotes the average scheduling learning rate. Whilst the result illustrates a smooth converging learning rate for the STC algorithm there is still a problem with the initial learning rate η_0, which is selected arbitrarily. A large initial learning rate may result in a long training session and training instabilities. By initially measuring the performance of the model and understanding the dynamics of the model; an initial optimal learning rate may be chosen to improve the performance of the STC algorithm.

2.4 ADAPTIVE LEARNING RATE

2.4.1 ADAPTIVE LEARNING RATE CLIPPING STABILISES LEARNING

The adaptive learning rate (ALR) [40] method involves a dynamic learning rate that changes based on the gradient and other tunable parameters of the model. Jeffery and Richard [5] propose an adaptive learning rate that is used to minimise the loss spike in the backpropagation gradient [15] that is a consequence of a bad batch in training data. These spikes are said to be caused by high learning rates, small training batches and high order cost functions. While the cost function and the batch size may be changed with ease, the learning rate tends to be difficult to optimise. The study proposes an adaptive learning rate algorithm that implements an adaptive learning rate to minimise the loss spikes in the model; this is referred to as adaptive learning rate clipping (ALRC). The algorithm clips the unwanted spikes by comparing the backpropagation gradient with a predefined criterion. The ALRC algorithm adapts the learning rate to influence the loss spikes propagated back through the model. The learning rate is constantly optimised to suppress the loss spikes as a result of bad training batches.

The ALRC algorithm uses the mathematical formula described by Equation 2.4 below to adaptively change the learning rate.

$$\eta_{th} = \eta \frac{m_i^{'}}{\sqrt{v_i^{'}+\epsilon^{'}}} . \qquad (2.4)$$

where η_{th} refers to the next iteration learning rate for a specific parameter, η is the current learning rate, m_i' is the first momentum, v_i' is the second momentum and ϵ' is the exponential decay rate, Figure 2.4 provides comparisons between models with and without an adaptive learning rate.

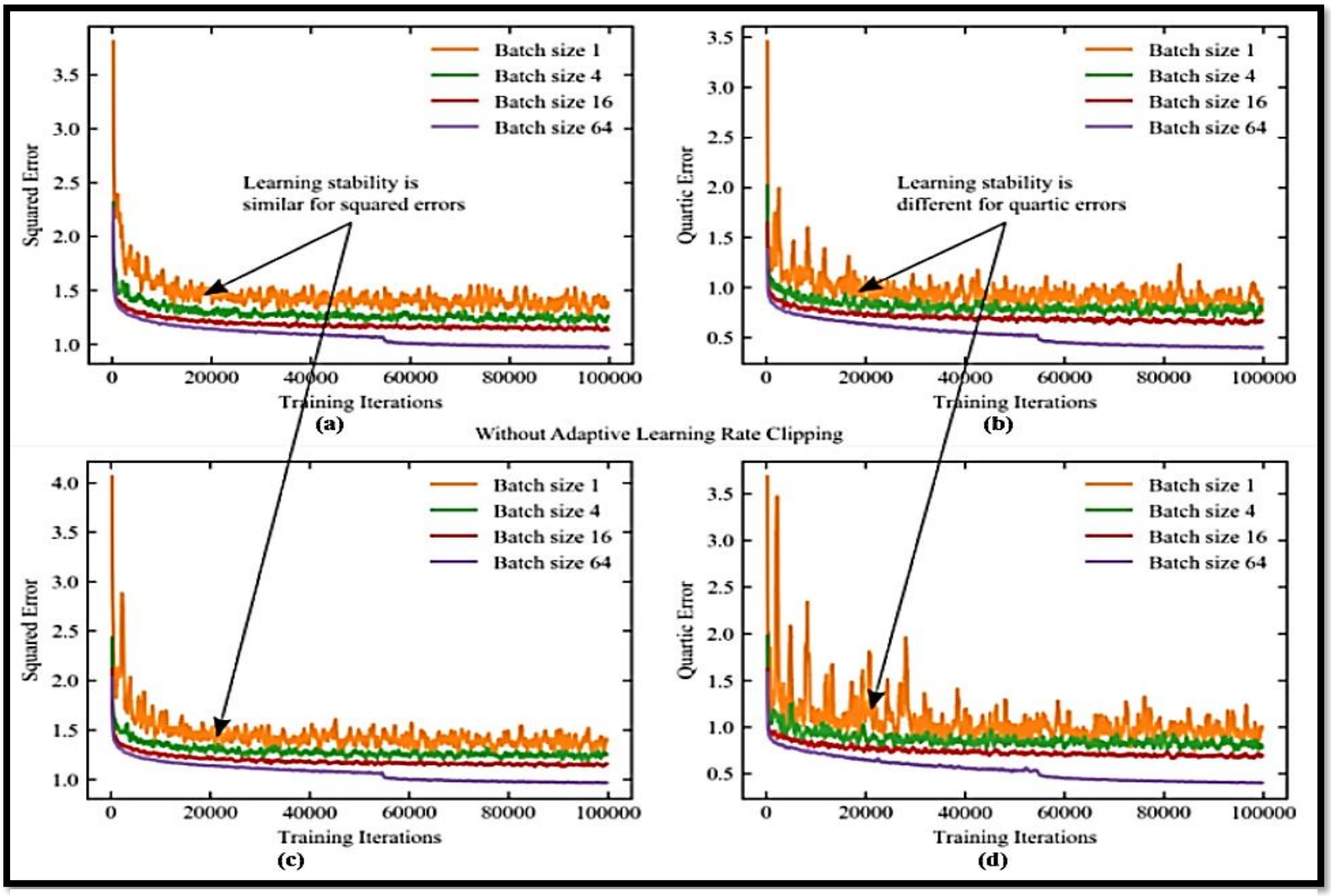


FIGURE 2.4: ADAPTIVE LEARNING RATE VERSUS NORMAL LEARNING RATE

Figures 2.4(a) and (b) show an adaptive learning rate that consists of minimum spikes; this is in contrast to Figures 2.4(c) and (d) that show the effect without ALRC. The large loss spikes in the cost function without ALR is a result of large errors that the algorithm attempts to correct. These loss spikes

are generated when a bad batch of training data is fed into a model. Although an ALR can suppress loss spikes, the result comes at the cost of over computation; this is owing to the need to update the learning rate.

Even though the cost function without an adaptive learning rate consists of large loss spikes, the level of minimising the cost function is fairly close to an adaptive learning model. For sensitive models, the ALRC algorithm may be used, but for flexible models, a constant optimal learning rate is more appropriate. An optimal constant learning rate proposed may be able to match the performance of an adaptive learning rate without the overhead computation and memory requirements.

2.4.2 Learning Rate Adaptation In Stochastic Gradient Descent

Plagianakos et al. [41] calculate an optimal learning rate such that the problem of the locality phenomenon, as well as convergence speed, are resolved. The proposed algorithm uses the cost function gradient to adapt the learning rate for a given model. The algorithm uses three gradients to update the learning rate; these include the current gradient $\nabla E_i(w_i)$, the gradient from the previous iteration $\Delta E_{i-1}(w_{i-1})$ and the gradient from two iterations back $\Delta E_{i-2}(w_{i-2})$. The learning rate is updated subsequent to a single iteration using Equation 2.5 below.

$$\eta^{i+1} = \eta^i + \gamma_1 \left(\nabla E_{p-1}\left(w^{i-1}\right).\nabla E_i\left(w^i\right)\right) + \gamma_2(\nabla E_{p-2}(w^{i-2}).\nabla E_{p-1}\left(w^{i-1}\right) \qquad (2.5)$$

The adaptive learning rate (ALR). denoted by η^{i+1}, is a function of the current learning rate η^i. the current gradient $\nabla E_i\left(w^i\right)$, the previous gradient $\nabla E_{p-1}\left(w^{i-1}\right)$ and the second previous gradient $\nabla E_{p-2}(w^{i-2})$. An ALR depends on the magnitude of the cost function gradient with respect to the model parameters. As the gradient increases so too does the learning rate; this, in turn, increases the cost function convergence. As the cost function gradient decreases, the learning rate is also reduced, which results in the cost function approaching the absolute minimum. Table 2.2 shows the results of the algorithms compared with other optimisation algorithms for an exclusive or binary classification.

TABLE 2.2: ALGORITHM TRAINING ACCURACY

Algorithm	Min	Mean	Max	Succ.
Batch BP	176	1693.9	3840	17%
Batch ABP	144	1430.4	3708	49%
On–line BP	72	724.2	2972	43%
$ALAP_1$	56	736.1	3900	38%
$ALAP_2$	40	816.9	3960	37%
$ALAP_3$	52	1000.5	3636	43%
Algorithm–1	44	680.2	3388	48%

Algorithm-1 performs fairly well with an accuracy of 48% while lagging behind the Batch ABP algorithm with an accuracy of 49%. The results demonstrate high performance for Algorithm-1, using the gradient to update the learning rate. Algorithm-1 achieves high performance due to the stored gradients from different iterations. The integration of several gradients provides stability whilst sacrificing computational resources. The algorithm has to store, in memory, the gradients from two iterations back, the previous iteration and the current iteration. For large models the memory overhead may not be desirable; this may be resolved by limiting the number of iteration gradients to one instead of three. Another alternative is to utilise an optimal constant learning rate to overcome the overhead computation and memory usage as proposed in the current research.

3 CHAPTER 3: DEEP CONVOLUTIONAL NEURAL NETWORK

3.1 INTRODUCTION

This chapter examines the internal structure of a deep convolutional neural network (DCNN) model and discusses the interaction between the fundamental building blocks of the model. A DCNN is defined as a class of deep neural network models that may be, for example, designed to process image classification tasks [12]. The structure of a DCNN consists of two interconnected layers, the convolutional layer [13] and a fully connected deep neural network layer [42]. The convolutional layer is responsible for extracting essential features from the input image, whilst a fully connected deep neural network layer generates predictions based on the extracted features from the preceding convolutional layer.

The logical architecture of a DCNN resembles a closed-loop system consisting of a forward propagation and backpropagation path [38]. Forward propagation refers to data propagating from the input to the output of the model whilst backpropagation is the opposite. When data propagates forward, it excites the model parameters and the excitation is measured against a predefined output setpoint where an error function is generated. The error function is propagated back from the output to the input of the model whilst performing corrective measures to optimise the performance of the model. Figure 3.1 illustrates a DCNN model that processes images of handwritten digits. The input image is propagated through the convolutional layer to generate the feature maps. These features are flattened into a vector and subsequently; fed into the fully connected layer to be classified as a probability distribution of the output labels.

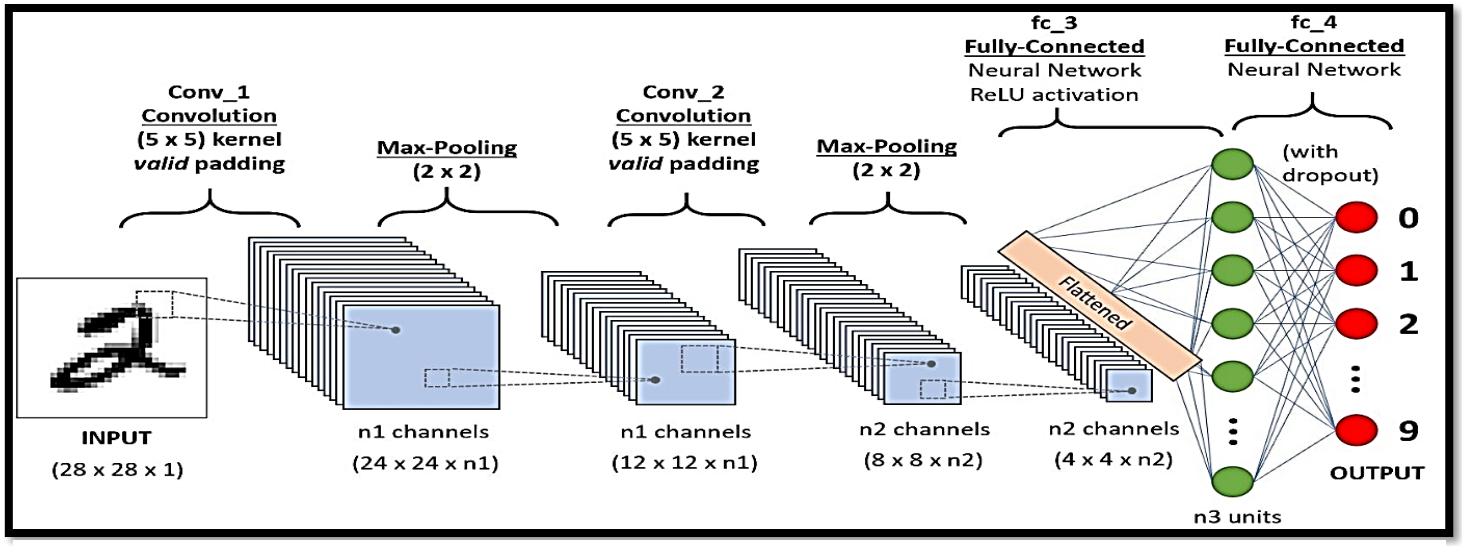


FIGURE 3.1: DCNN NETWORK MODEL [43]

3.2 CONVOLUTIONAL LAYER

In mathematics, convolution refers to an integral that expresses the area of overlap of the function *g(t)* as it is shifted over by function *f(t),* the operation is denoted as $(f * g)(t)$ [44]. In a convolutional neural network (CNN), convolution is an operation that masks the desired data from an input image using a kernel matrix [1]. A kernel is a matrix that learns to extract essential features from an input image. A kernel matrix convolves the input image and the resulting matrix is referred to as a feature map. A convolutional layer may comprise several kernels grouped to extract high-level features; multiple kernels are referred to as filters [45-47]. Figure 3.2 illustrates the convolution operation. whereby the kernel matrix convolves the input image to extract visual features.

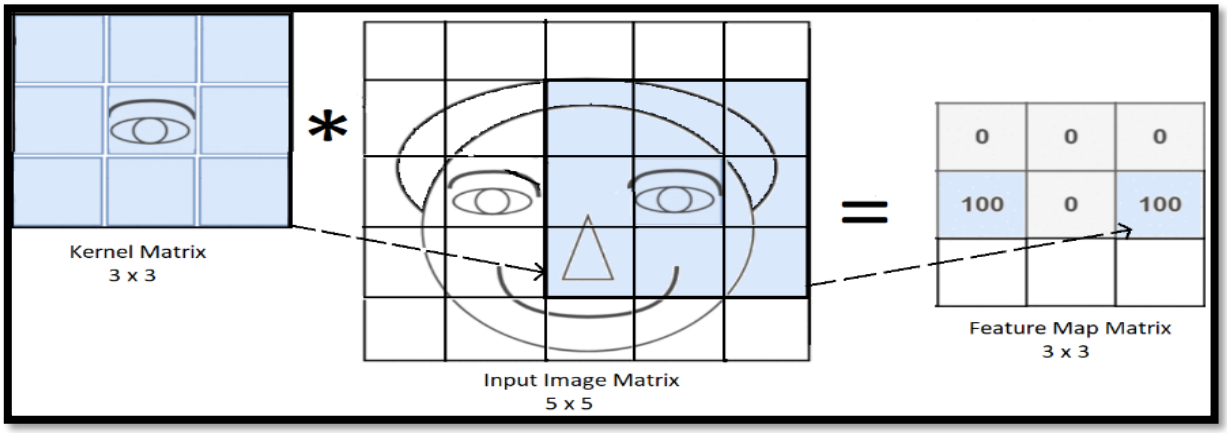


FIGURE 3.2: CONVOLUTION OPERATION [2]

The feature map matrix elements are calculated as a dot product of the input image matrix under the kernel matrix, receptive field. A receptive field defines the area of the kernel matrix suspended above the input image as the kernel convolves the image [48]. The convolution operation is mathematically expressed by Equation 3.1; where the indexes of rows and columns of the result matrix are denoted with m and n, respectively. The equation defines the dynamics of the kernel above the input image, the kernel is shifted from the far left of the input to the far right of the input at unit interval steps. The kernel is then reset to the far left, a unit row down of the input image and shifted from left to right once again. This operation is iterated several times until the entire image is convolved [49].

$$G[m.n] = (f * h)[m.n] = \sum_j \sum_k h[j.k] f[m - j.n - k] \qquad (3.1)$$

A single layer of the network may consist of several filters extracting different features, these features may include edge detection, blur, and texture. The features detected by the kernel are learned in the course of the training session. The features are propagated forward and combined to generate more

unique features. This operation is iterated over several convolutional layers to acquire in-depth features that enhance the unique properties of the input image data; this is illustrated in Figure 3.3 below.

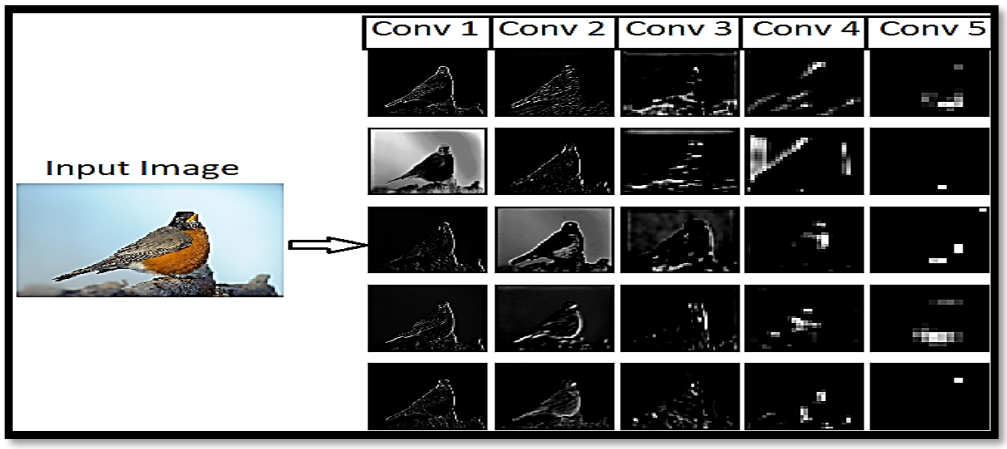


FIGURE 3.3: DEEP LAYER FEATURES MAPS [50]

In Figure 11 Conv 1, Conv 2, Conv 3, Conv 4, and Conv 5 refer to the extracted (i) foreground focus, (ii) gradient, (ii) horizontal edges, (iv) vertical edges and (v) texture, respectively. The kernel matrices adapt over the course of training to detect these features. The complexity of the features detected depends on the number of convolution layers. Deep convolution layers detect features that are more specific to the input image; in this case, these may include the beak of the bird as well as the texture of the feathers.

The convolution operation possesses several tunable parameters, referred to as hyperparameters, which may be adjusted to improve the performance of the convolution operation [51]. Two of these parameters include matrix padding and propagation stride; these are discussed below.

3.2.1 PADDING HYPERPARAMETER

Matrix padding is a hyperparameter that defines the dimensions of the feature map resulting from the convolution operation [52]. Figure 3.4 illustrates how the feature map matrix is reduced after the convolution operation. The reduced feature map leads to data loss in subsequent layers. This problem is resolved by padding the borders of the image with zeros before the convolutional layer. This allows the feature map to retain the same dimensions as the input image. This is illustrated in Figure 12 below.

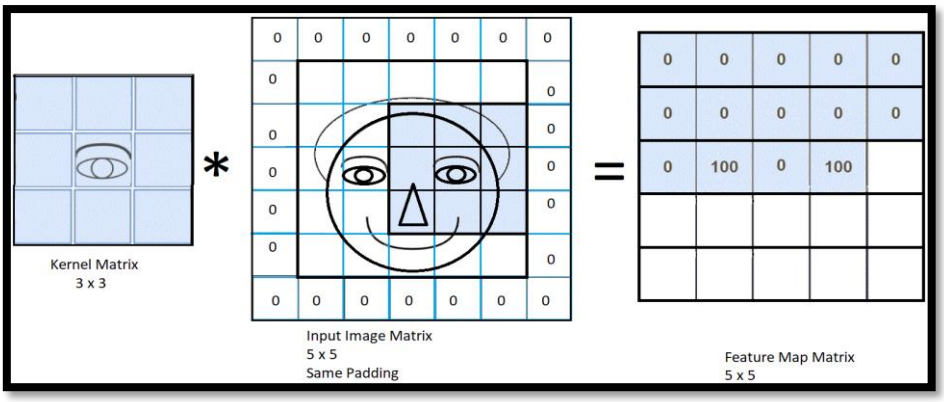


FIGURE 3.4: SAME PADDING

The operation that borders the input image with zeros is referred to as the *same padding* configuration [53], whilst the operation that retains the reduced image is referred to as *valid padding*. Valid padding may be used to reduce the dimensions of the input image, which consequently reduces the number of parameters propagated forward. The padding width is expressed by Equation 3.2.

$$p = \frac{f-1}{2}. \tag{3.2}$$

where p denotes the padding width and f represents the filter dimension. The equation has a precondition that enforces an odd filter dimension of (3×3, 5×5 and 7×7).

3.2.2 STRIDE HYPERPARAMETER

The stride is a hyperparameter that defines the step size of the convolving kernel [3], a unity stride enforces the convolving kernel to shift across the input image at a unit step. A stride value of $n > 1$ has a similar effect on the feature map as the valid padding operation; where the kernel omits much of the essential image data. A higher *nth* stride may be better suited for large images with similar localised pixel values. A lower *nth* stride captures more detailed feature map data as the stride can convolve a much greater image area. The dimension of a feature map matrix is calculated using Equation 3.3. as a function of the padding width and the size of the stride.

$$N_{out} = \left[\frac{N_{in}+2p-f}{s} + 1\right]. \tag{3.3}$$

where the dimension of the output matrix is denoted as N_{out}, N_{in} is the input matrix, p is the padding width, f is the dimension of the kernel and s is the stride size.

For each convolutional layer, there is an activation function that is responsible for optimising the model parameters. An activation function is a distinctive factor between a linear regression model [54] and a DCNN model. Linear regression approximates linear functions whilst a DCNN can approximate complex nonlinear functions.

3.3 ACTIVATION FUNCTION

Activation functions are an essential component of a DCNN model; these functions provide DCNN models with the ability to approximate complex real-world data. An activation function is a mathematical function that introduces non-linearity to the otherwise linear DCNN model [55]. An activation function serves a dual purpose, which is defined by the data propagation vector through the model. As data propagates forward an activation function determines the active state of the inputted data. In backpropagation, an activation function is used to optimise the model parameters by introducing a gradient [23], this is discussed in Chapter 4. Activation functions consisting of a dynamic gradient are more favourable as they offer a wide range of optimisation.

In image processing, the rectified linear unit (ReLU) is the most favourable and widely used activation function [56]. A DCNN model implements the ReLU function in all alternating layers excluding the last layer of the model. Figure 3.5 illustrates the ReLU function.

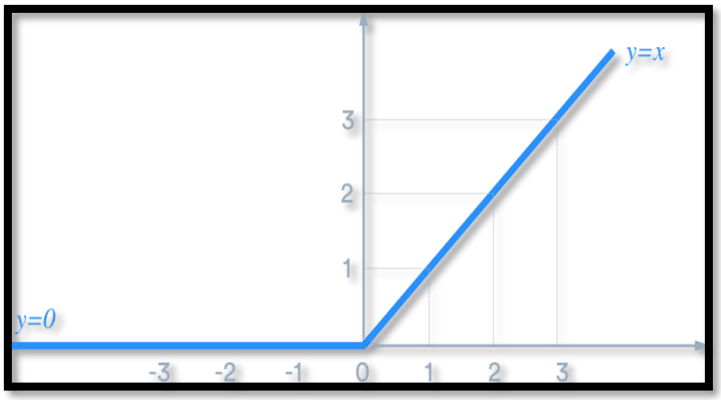


FIGURE 3.5: RELU ACTIVATION FUNCTION [57]

The ReLU function is zero for $x < 0$, $\max(0, x)$ for $x \geq 0$ and a gradient of 0 and 1, respectively; this is expressed in Equation 3.4.

$$\sigma(x) = \begin{cases} 0 \, for \, x < 0 \\ \max(0.x) \, for \, x \geq 0 \end{cases} \tag{3.4}$$

The derivative of the ReLU function is 0 for *x* is less than 0 and 1 for *x* greater than or equal to 0. The gradient of the function is applicable when propagating back through the model; this gradient is expressed in Equation 3.5.

$$\sigma'(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases} \tag{3.5}$$

The last layer of the model implements the Softmax function [58] as the final element that classifies the input data; the Softmax function outputs a vector that expresses the input data as a probability distribution of potential outcomes [58]. The sum of these probabilities is always a unit. The properties of the Softmax function are expressed in Equation 3.6.

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{3.6}$$

where $\sigma(x_i)$ denotes a vector of possible probabilities for the output labels, e^{x_i} is an exponential output of a distinct neuron in the output layer and $\sum_{j=1}^{n} e^{x_j}$ represents the sum of the exponential output for all output neurons. The derivative of the Softmax function is expressed by Equation 3.7; this gradient will also be applicable in the optimisation phase of the model.

$$\sigma'(x_i) = \sigma(x_i)\left(1 - \sigma(x_j)\right), \tag{3.7}$$

where $\sigma'(x_i)$ denotes the change in the activation function with respect to the neuron output x_i and $\sigma(x_i)$ refers to the original output of the softmax function at the neuron output x_i.

The feature map matrix resulting from the convolution operation is propagated forward through the activation function; the elements of the feature map matrix are mapped to the activation function. The ReLU function permits all values of $x \geq 0$ to progress while suppressing values of $x < 0$ as zero. The output of the activation function is a feature map matrix mapped to a nonlinear function. The feature map matrix is propagated forward and subsequently, down-sampled by the pooling layer to reduce the number of parameters.

3.4 POOLING LAYER

The pooling layer performs a down-sampling operation that reduces the dimension of the feature map after it has been mapped to the activation function. The pooling operation reduces the number of

parameters that a model has to learn whilst maintaining translation-invariant geometry [28]. The operation determines the weights permitted to the output from the pooling receptive field. There are two variants of the pooling operation and these include the max-pooling and the average pooling. Figure 3.6 illustrates the max-pooling operation.

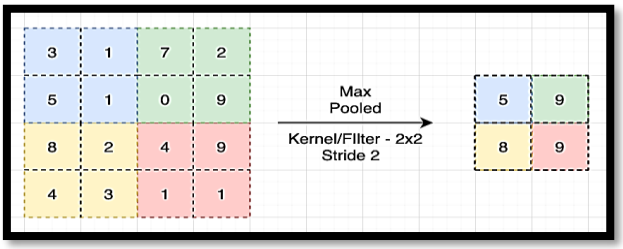


FIGURE 3.6: MAX POOLING OPERATION [59]

Max pooling selects the highest value under the pooling receptive field, whilst average pooling performs an average for the values in the pooling receptive field. The pooling receptive field convolves the feature map matrix and the resulting feature map is a down-sampled version of the input feature map. This is the last layer of the convolution layer.

In summary, the convolutional layers utilise filters to extract features from the input image. These features are propagated forward through an activation function to introduce non-linearity. Lastly, the feature map resulting from the activation function is downsampled by the pooling operation to reduce the number of parameters. The output of the convolutional layer is determined by the number of filters; in complex data classification, several filters are applied and consequently, several feature maps are generated as an output of the convolutional layer. The convolutional layer outputs a two-dimensional matrix, which is not suitable as an input to a fully connected layer. The two-dimensional matrix is flattened into a one-dimensional vector and subsequently fed into the fully connected layer; this is discussed in the following section.

3.5 FULLY CONNECTED DEEP NEURAL NETWORK

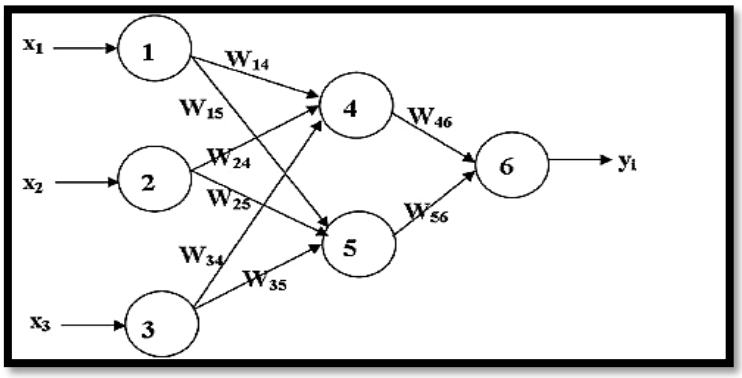


FIGURE 3.7: DNN ARCHITECTURE [60]

A deep neural network (DNN) is a multilayer connection of neurons. these neurons are interconnected to form a complex network that resembles the human brain architecture [42]; this is illustrated in Figure 3.7. A DNN comprises an input layer that accepts a vector as an input, a hidden layer that consists of layers of interconnected neurons and an output layer consisting of the various predictions. A DNN is classified as either supervised or unsupervised, depending on the feedback architecture. A supervised model consists of data labels in the output layer; these labels inform the model about the accuracy of the prediction. An unsupervised network classifies data as groups and clusters. Figure 3.8 below is used as a frame of reference as data propagates forward through the network.

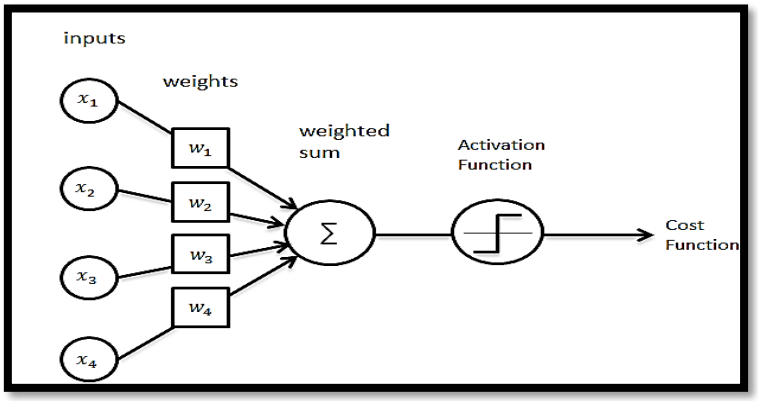


FIGURE 3.8: PERCEPTRON MODEL[61]

3.5.1 MODEL NOTATION

Table 3.1 shows all the symbols used to calculate the transformations as training data propagate forward through the model.

TABLE 3.1: DNN EQUATION NOTATION

SYMBOLS	DESCRIPTION
L	Total number of layers
L	The selected layer
J	Selected node
Y	The desired output
W	Weight
Z	The output of the neuron
σ	Activation function
A	Activation function output
B	Bias term

The input layer x_j is made up of nodes that accept the input vector, this is the only function of the input node. The weights w_i represent the parameters learned during the training session. The weights of the network are randomly initialised when the network is created; as the network is trained these weight values change to improve the model performance. The output of a neuron is a weighted sum Z of the weight from the preceding input node; added with a bias term b. The bias ensures a non-zero weight sum as the neuron output. The neuron weighted sum is expressed by Equation 3.8.

$$Z = \sum_{j=0}^{N-1}(w_i \times x_i) + b. \qquad (3.8)$$

The output of the neuron Z is an input to the activation function; the activation function, which is discussed in section 3.3 introduces non-linearity to the model. The ReLU function σ^l accepts the output of the neuron Z and generates an activation function output a with non-linear properties; this is expressed by Equation 3.9 below.

$$a = \sigma^l(Z) = \sigma^l\left(\sum_{j=0}^{N-1}(w_i \times x_i) + b\right) = Max\left(0.\sum_{j=0}^{N-1}(w_i \times x_i) + b\right). \qquad (3.9)$$

The process above is iterated for all the neurons in the network except the last neurons, the last layer of the model implements the Softmax function in contrast to other layers. The Softmax function computes a probability distribution for all possible output predictions. The probability distribution sum is constrained to a value of one. In a supervised model architecture, the prediction is compared with an output label to evaluate the accuracy of the model; this is expressed by Equation 3.10.

$$C_0 = \sum_{j=0}^{N-1}(a - y)^2. \qquad (3.10)$$

where the square difference between the label y and the output prediction a is referred to as the cost function. which defines the performance of the model [62]. The objective of a cost function is to measure the performance of the model with each iteration. The optimisation of the model is achieved by adjusting the weights and biases in a manner that minimises the cost function.

3.6 SUMMARY

The fundamental concept for extracting features from input data has proven to be an efficient model for computer vision. A DCNN model learns to extract features through an iterative process of training and validation. The kernel matrix dynamically evolves as the model trains to resemble all possible states of the training data. This provides the ability to mask features from data never exposed to the model. A combination of filters allows the model to learn advanced features and discard unwanted background information. The extracted features are propagated to a fully connected neural network. where the features are classified as a distribution of the output probabilities.

The convolutional layer and the fully connected deep neural network consists of learnable parameters. These parameters learn the data set as the model trains, where a convolutional layer learns by adjusting the kernel matrix elements or weight, whilst a fully connected deep neural network adjusts the weights connecting the neurons. The model learns as the generated error is propagated back through the model whilst adjusting all the weight and biases. This technique is discussed in Chapter 4 as backpropagation.

4 CHAPTER 4: MODEL OPTIMISATION (BACKPROPAGATION)

4.1 INTRODUCTION

This chapter evaluates the effect of an optimisation algorithm on a Deep Convolutional Neural Network (DCNN) model. An optimisation algorithm is a mathematical function that measures the performance of the model and computes a corrective action aimed at optimizing the model [14]. Gradient descent is a class of optimisation algorithms that implement backpropagation to optimise the model. Backpropagation is a technique that computes the gradient of the model cost function relative to the model parameters (weights and biases) [4]. The calculated gradient is used to adjust the model parameters in the direction that minimises the cost function.

4.2 GRADIENT DESCENT

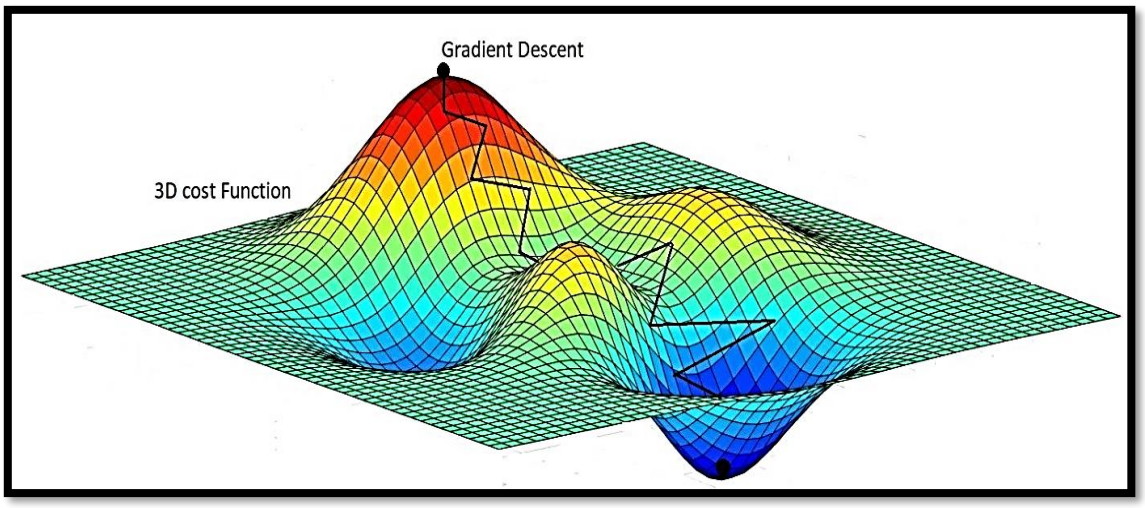


FIGURE 4.1: GRADIENT DESCENT [63]

The cost function is a multidimensional function that expresses the performance of the model relative to the training data [62]. Gradient descent attempts to descend on the slope of the cost function to reach the lowest point possible; this is illustrated in Figure 4.1. The lowest point of the cost function defines the highest model accuracy for the given training data. The class of gradient descent consists of batch and stochastic gradient descent (SGD) [64]. The distinction between these algorithms is the amount of training data required to update the model parameters. Batch gradient descent updates the model parameters after the entire training data [15], whilst SGD updates the model parameters for every instance of training data.

The SGD algorithm is one of the most commonly used optimisation algorithms to train DCNN models. During the training session, the SGD algorithm uses the backpropagation algorithm to compute the partial derivative of the cost function relative to the model parameters. The gradient of a function defines the steepest ascent whilst the negative gradient defines the steepest descent.

$$W = w - \eta \nabla C(w). \qquad (4.1)$$

Equation 4.1 expresses how the SGD algorithm optimises the weight W by computing the gradient of the cost function $\nabla C(w)$ with respect to the weight w. The gradient is then multiplied by the learning rate η and subtracted from the weight w, the current position that must be adjusted to optimise the model. The process is iterated for the entire training data until an accepted model accuracy is obtained. The gradient $\nabla C(w)$ is managed by the backpropagation algorithm and the learning rate is a hyperparameter that must be configured before training the model.

4.3 BACKPROPAGATION

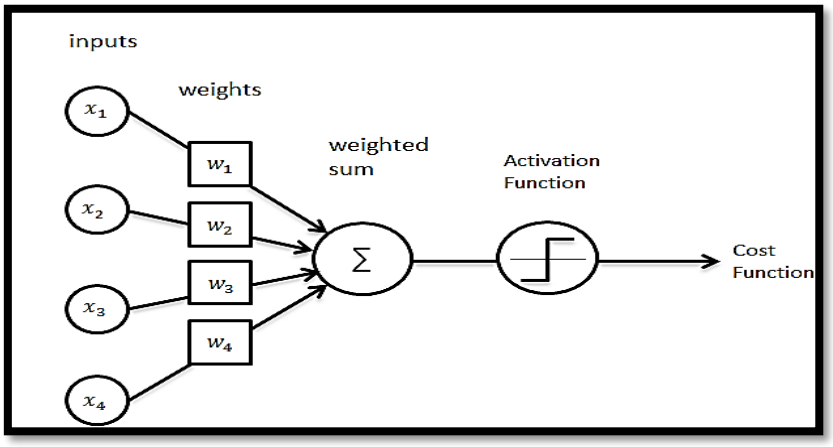


FIGURE 4.2: PERCEPTRON MODEL [61]

The backpropagation algorithm computes the gradient of the cost function with respect to all the model parameters. The objective is to alter the model parameters to an optimal point that consequently causes the cost function to descend to an absolute minimum. To achieve the objective, the model is excited with an instance of training data and the model performance is measured as a cost function. The cost function C_0 is the square difference between the model prediction a^l and the actual output label y, this is given by Equation 4.2. The cost function is then propagated back through the model to optimise the model parameters in a manner that improves the model performance for the next iteration.

$$C_0 = \left(a^l - y\right)^2. \tag{4.2}$$

To observe the backpropagation technique the cost function C_0 is propagated back through the model to optimise the weight (w_1).

Figure 4.3 illustrates how the input x_1 is propagated forward to excite the model and generate the cost function C_0.

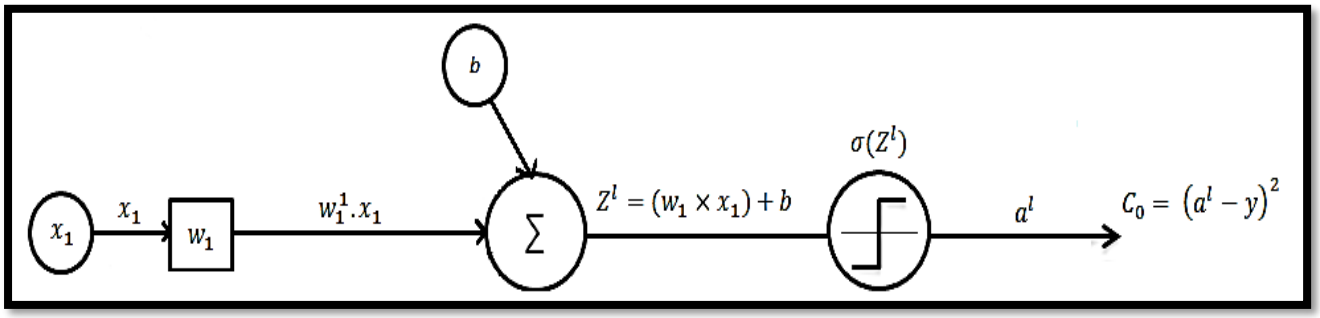


FIGURE 4.3: SINGLE NEURON MODEL

The initial step is to compute the partial derivative of the cost function C_0 with respect to the model prediction a^l; this is expressed by Equation 4.3 and illustrated in Figure 4.3.

$$\frac{\partial C_0}{\partial a^l} = 2(a^l - y) \tag{4.3}$$

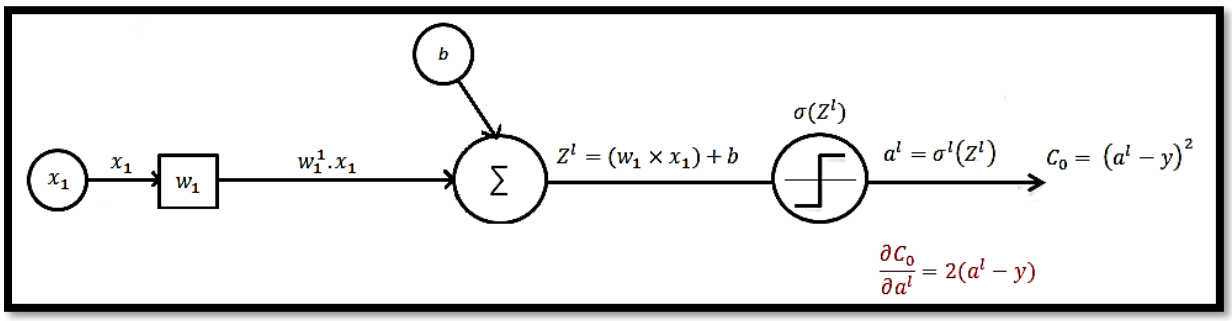


FIGURE 4.4: PARTIAL DERIVATIVE OF C_0 WITH RESPECT TO a^l

The second step is to compute the partial derivative of the model prediction a^l expressed by Equation 4.4, with respect to the weighted sum Z^l expressed by Equation 4.5 and illustrated in Figure 4.5.

$$a^l = \sigma^l(Z^l) \tag{4.4}$$

$$\frac{\partial a^l}{\partial Z^l} = \sigma(Z^l)(1 - \sigma(Z^l)) \tag{4.5}$$

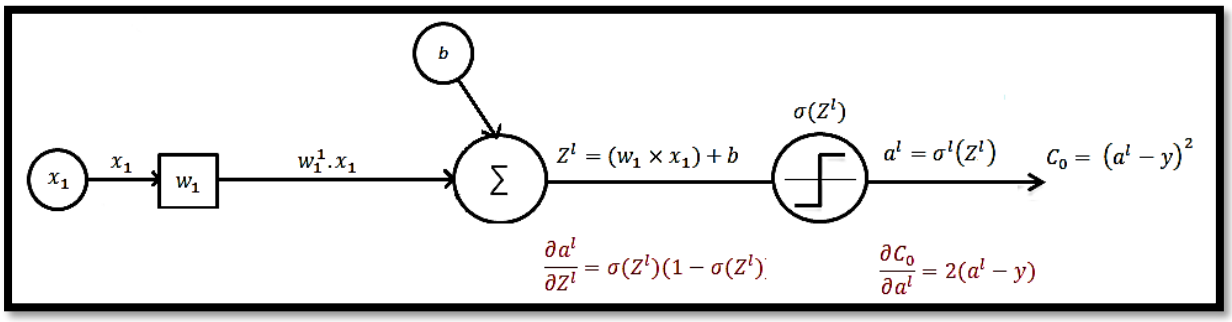


FIGURE 4.5: PARTIAL DERIVATIVE OF a^l WITH RESPECT TO Z^l

The last step is to compute the partial derivative of the weighted sum Z^l expressed by Equation 4.6, with respect to the weight w_1; this is expressed by Equation 4.7 and illustrated in Figure 4.6.

$$Z^l = (w_1 \times x_1) + b \tag{4.6}$$

$$\frac{\partial Z^l}{\partial w_1} = x_1 \tag{4.7}$$

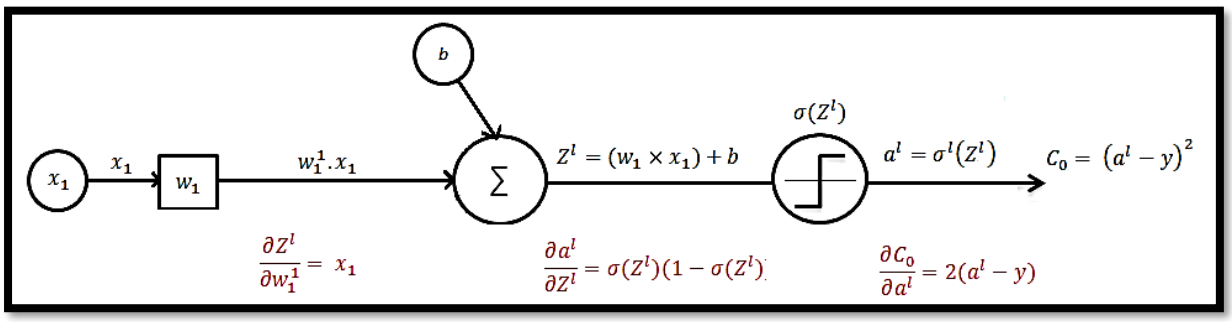


FIGURE 4.6: PARTIAL DERIVATIVE OF Z^l WITH RESPECT TO w_1

The derivative of the cost function C_0 with respect to the weight w_1 is defined by Equation 4.8 below. The equation expresses the relationship between the cost function C_0 and the weight w_1; this is shown in Figure 4.7. To calculate the partial derivative for other parameters only the indexes are altered to refer to the parameters to be optimised.

$$\frac{\partial C_0}{\partial w_1} = \frac{\partial Z^l}{w_1} \times \frac{\partial a^l}{\partial Z^l} \times \frac{\partial C_0}{\partial a^l} \tag{4.8}$$

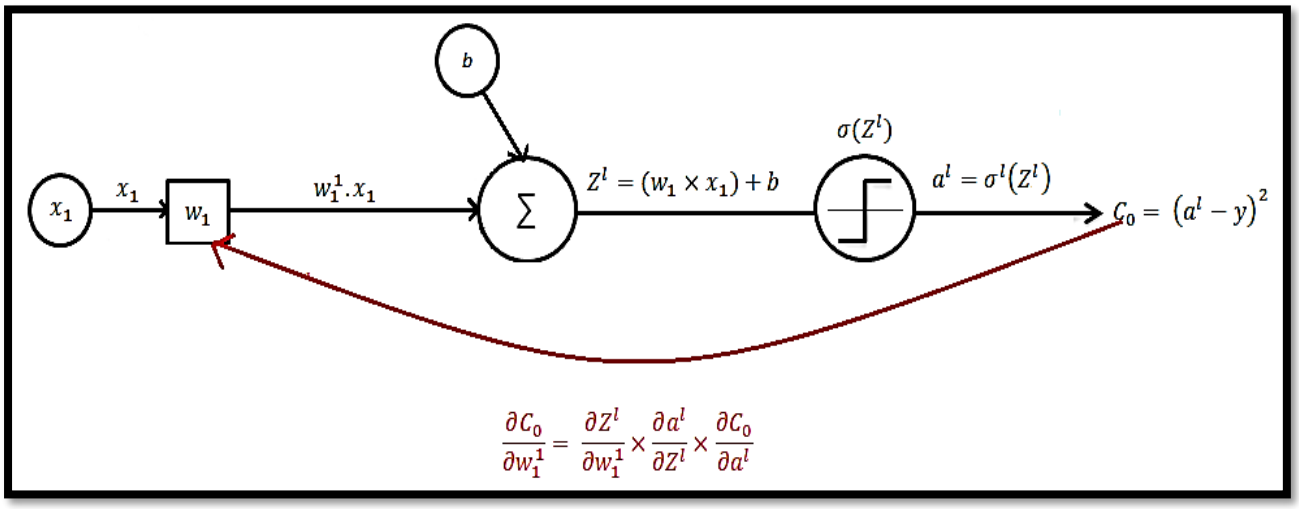


FIGURE 4.7: PARTIAL DERIVATIVE C_0 WITH RESPECT TO w_1

The partial derivative $\frac{\partial C_0}{\partial w_1}$ defines how a change in the weight w_1 causes a change in the cost function. The weight w_1 is adjusted in a manner that will cause the cost function to converge to the absolute minimum. The SGD algorithm makes use of the partial derivative to converge the cost function, this is achieved by introducing the negative sign in the gradient. Equation 4.9 is used to compute the optimal position of the weight w_1. The capital letter W_1 refers to the new weight value, small letter w_1 refers to the current weight value, $\nabla C(w_1)$ refers to the gradient $\frac{\partial C_0}{\partial w_1^1}$ and η is the learning rate.

$$W_1 = w_1 - \eta \frac{\partial C_0}{\partial w_1} \tag{4.9}$$

The optimisation algorithm extends to the convolutional layer to optimise the filters to improve feature masking.

4.4 CONVOLUTIONAL LAYER BACKPROPAGATION

Backpropagation begins in the fully connected layer and extends to the convolutional layer to optimise the kernel matrix responsible for extracting features from the input data. The kernel matrix consists of weights that must be optimised to filter the desired features. The cost function from the fully connected layer is propagated through the pooling layer, activation function and to the kernel matrix [49]. When the kernel is fully optimised it can mask all possible features from the training data. Figure

4.8 below illustrates the process of propagating the cost function from the input of the fully connected layer to the kernel matrix.

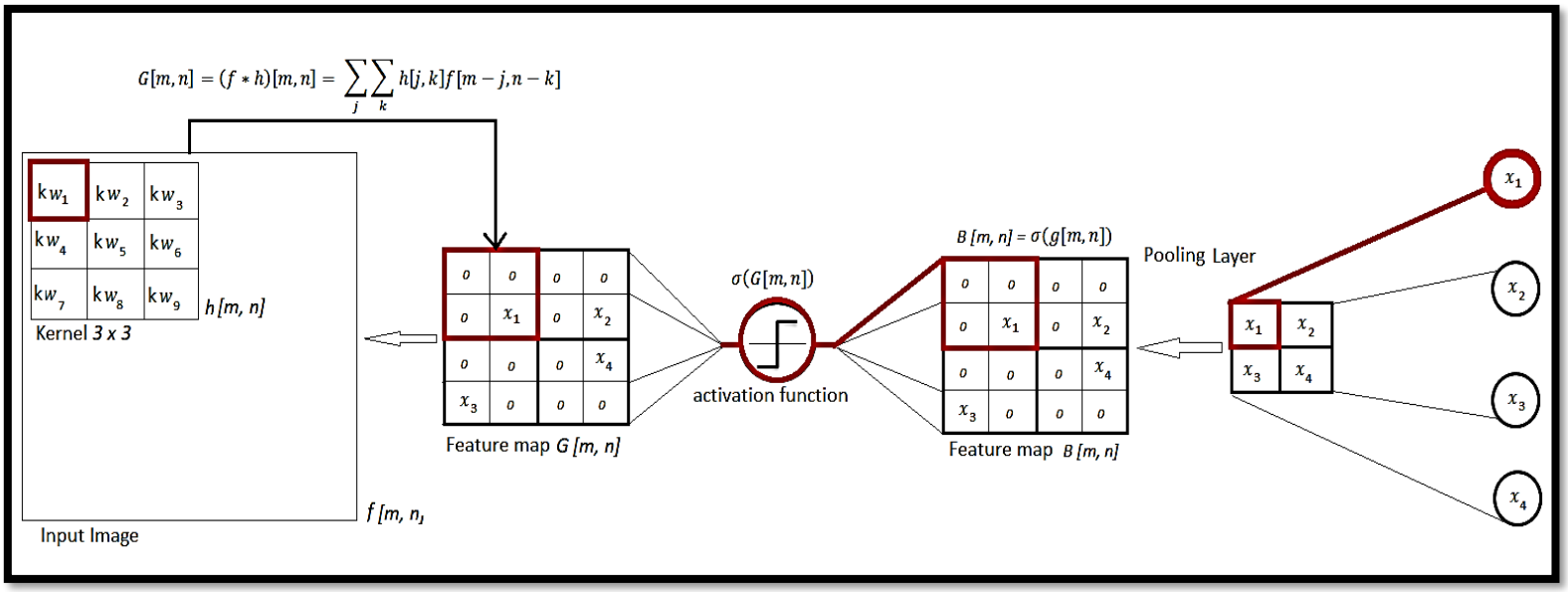


FIGURE 4.8: CONVOLUTIONAL LAYER BACKPROPAGATION

In backpropagation, the pooling layer upscales the resulting pooling matrix to recreate the feature map matrix $b[m,n]$. Subsequent to recreating the feature map $b[m,n]$, the matrix elements are populated with pooling layer elements. The non-zero elements of the feature map $b[m,n]$ correspond to the location of the dominant element from the forward propagation process. In summary, the pooling layer selects the highest value within the pooling receptive field [28].

After the feature map $b[m,n]$ there is an activation function with nonlinear properties; here a partial derivative is calculated to propagate through the activation function. The partial derivative is calculated as a function of the weight w_1 from the fully connected layer with respect to the feature map $g[m,n]$ resulting from the convolution operation. This is expressed by Equation 4.10 and illustrated in Figure 4.9.

$$\frac{\partial w_1}{\partial g[m.n]} = \sigma'(g[m.n]) \qquad (4.10)$$

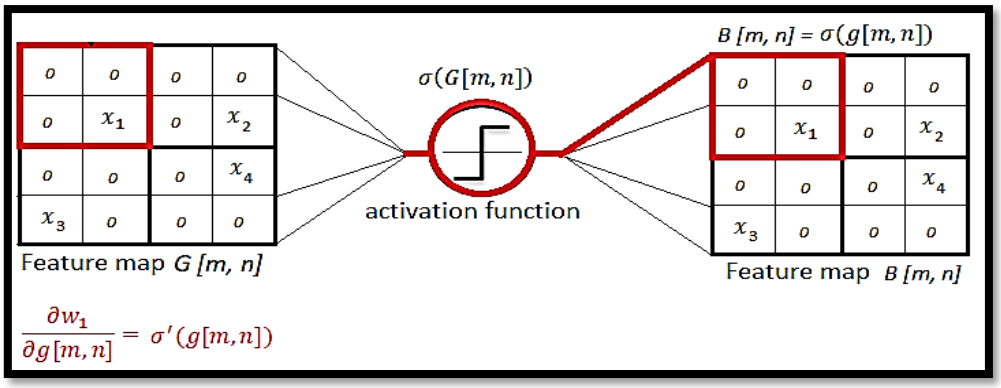


FIGURE 4.9: PARTIAL DERIVATIVE FOR $\frac{\partial w_1}{\partial g[m.n]}$

The feature map $g[m,n]$ is a function of the convolution operation between the input image $f[m,n]$ and the kernel matrix $h[m,n]$. The next step is to compute the partial derivative of the feature map $g[m,n]$ with respect to the weight of the kernel matrix $h[m,n]$. In the course of forward propagation, the convolution operation is governed by Equation 4.11 below.

$$G[m.n] = (f * h)[m.n] = \sum_j \sum_k h[j.k] f[m - j.n - k]. \qquad (4.11)$$

Where the partial derivative of the feature map $g[m,n]$ with respect to the kernel matrix $h[m,n]$ results in the input image at the location $f[m-j, n-k]$. This is expressed by Equation 4.12 and illustrated in Figure 4.10.

$$\frac{\partial g[m.n]}{\partial h[j.k]} = f[m - j.n - k] \qquad (4.12)$$

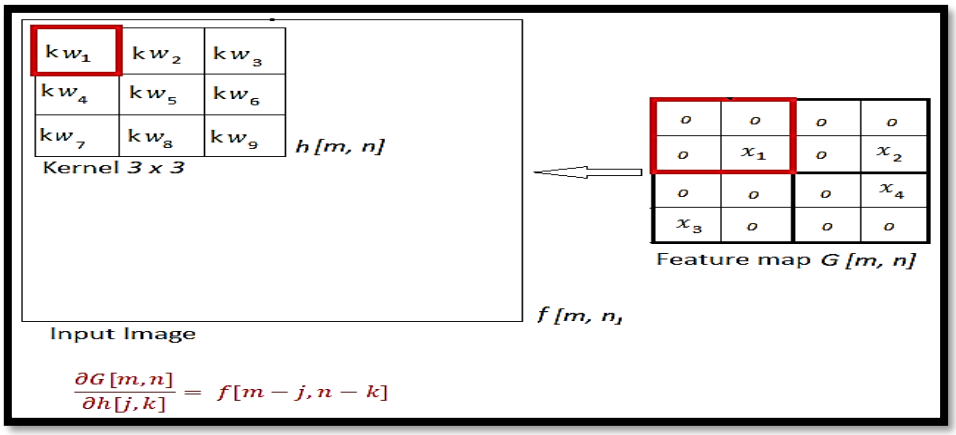


FIGURE 4.10: PARTIAL DERIVATIVE OF $\frac{\partial g[m.n]}{\partial h[j.k]}$

The partial derivative of $\frac{\partial g[m.n]}{\partial h[j.k]}$ allows the kernel matrix to resemble the input image; consequently, when the kernel matrix convolves the input image, essential features are masked more effectively. Equation 4.13 defines the partial derivative from the cost function to the kernel parameters, the other parameters are also calculated using the same equation with different indexes. In the course of training, the partial derivative expressed by Equation 4.13 is calculated for all the training instances until the model parameters render acceptable results.

$$\frac{\partial C_0}{\partial w_1^1} = \frac{\partial g[m.n]}{\partial k w_1^1} \times \frac{\partial w_1^1}{\partial g[m.n]} \times \frac{\partial Z^l}{\partial w_1^1} \times \frac{\partial a^l}{\partial Z^l} \times \frac{\partial C_0}{\partial a^l} \tag{4.13}$$

The SGD algorithm takes advantage of the backpropagation technique to compute the gradient; the resulting gradient is multiplied with the learning rate and an inverse sign is introduced to change the direction from ascending to descending. The product of the gradient and learning rate is subtracted from the current weight position kw_1 to produce the new weight KW_1; this is expressed in Equation 4.14. The weight position KW_1 has a higher potential to produce an acceptable prediction as opposed to the previous weight position kw_1.

$$KW_1 = kw_1 - \alpha \frac{\partial C_0}{\partial k w_1} \tag{4.14}$$

Since the SGD algorithm adjusts the model parameter for every instance, the descending accuracy is stochastic in nature. This has the advantage of a fast convergence for a small amount of training data whilst preserving the training resources such as memory and processing power. The computation of the gradient is a single step in the direction to optimise the model, the next step is to configure an optimal learning rate that will further improve the training session.

4.5 LEARNING RATE

The learning rate is a hyperparameter that determines the learning step size of the model [65]. Altering the learning rate affects the action required to optimise the model, When fine-tuning a DCNN model the learning rate is configured to aid the model descent more rapidly whilst avoiding overshooting the desired cost function minimum.

$$W = w - \boldsymbol{\eta} \nabla C(w) \tag{4.15}$$

Equation 4.15 expresses the relationship between the learning rate and the control action geared at minimizing the cost function.

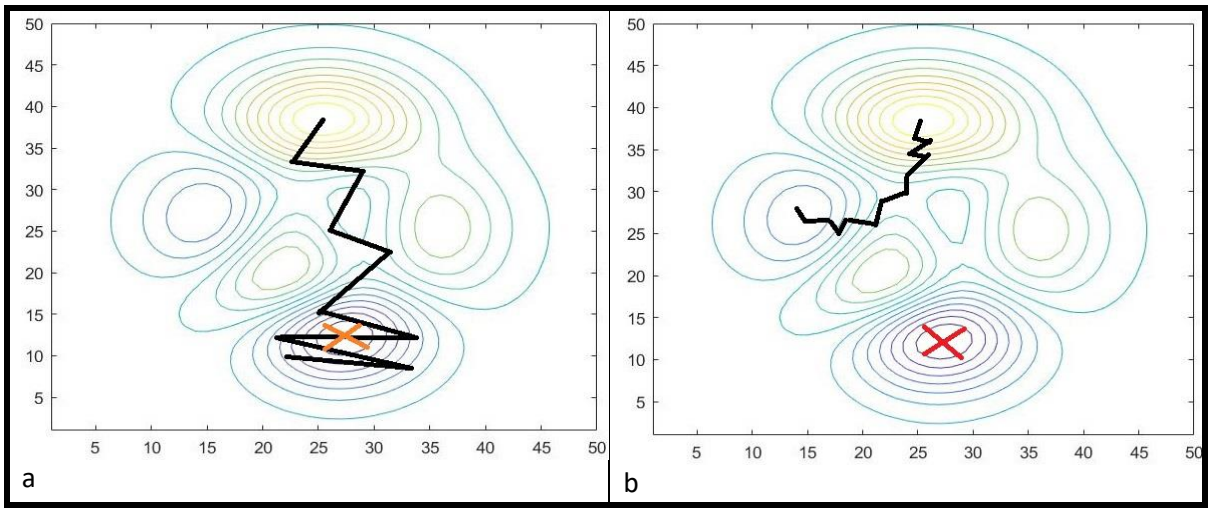


FIGURE 4.11: (A) CONTOUR FOR A HIGH LEARNING RATE AND (B) CONTOUR FOR A LOW LEARNING RATE

A high learning rate η results in the model oscillating around the lowest point of the cost function; this may result in an unstable training session. A low learning rate η causes the model to take an infinitely long time to reach the lowest point of the cost function. A low learning rate is also responsible for settling the model on a local minimum as opposed to the lowest point. The effect of a high and a low learning rate is illustrated in Figure 4.11(a) and (b), respectively.

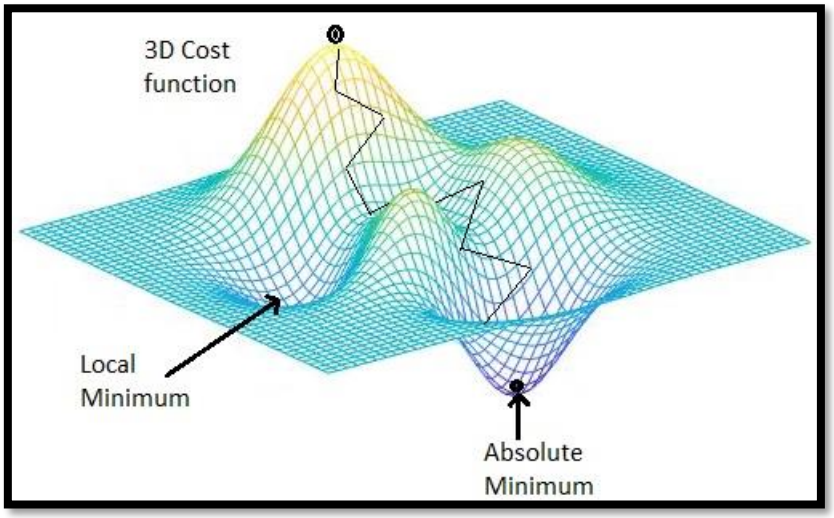


FIGURE 4.12: COST FUNCTION

The main objective is to establish an optimal learning rate that converges the cost function rapidly whilst settling on the lowest point of the cost function; this is illustrated in Figure 4.12. The learning

may be configured to settle within the range of [0.1 – 0.001]. depending on the complexity of the model. Chapter 5 provides a more in-depth discussion on the approach to computing the optimal learning rate.

4.6 SUMMARY

When a DCNN model is initially created all the parameters are initialised by a randomizing function, this results in a random prediction. The optimisation algorithm is introduced to change the parameters to an optimal value that may render accurate predictions. The optimisation algorithm measures the model performance and defines a cost function to govern the model performance. The cost function is a multidimensional function of all the model parameters.

The optimisation algorithm uses backpropagation to establish the relationship between the cost function and the model parameters. Backpropagation provides a function that describes the change of the cost function with respect to the model parameters. The optimisation algorithm implements the negative gradient to the model parameters to alter the current state in a manner that minimises the cost function. The process is iterated several times until the model accuracy meets the acceptance criteria.

5 CHAPTER 5: LEARNING RATE OPTIMISATION EXPERIMENT

5.1 INTRODUCTION

This chapter discusses the development of the algorithm used to model the relationship between the learning rate and model performance. The learning rate is a tunable hyperparameter that manipulates the step size taken to optimise the model parameters [22, 41]. When training a DCNN model the learning rate is initially arbitrarily selected and is gradually adjusted to obtain the desired model performance. An arbitrarily selected learning rate poorly affects model performance as well as training stability. A large learning rate causes an unstable training session [66], whilst a low learning rate causes the training session to take an infinitely long time to reach the absolute cost function minimal [67]. Consequently, the objective is to develop a mathematical model that describes the relationship between the learning rate and model performance. This model is expected to provide an optimal learning rate that has the potential to improve model performance whilst maintaining training stability. Figure 5.1 below illustrates the structure of the optimal learning rate (OLR) algorithm; this algorithm is developed to achieve the project objectives.

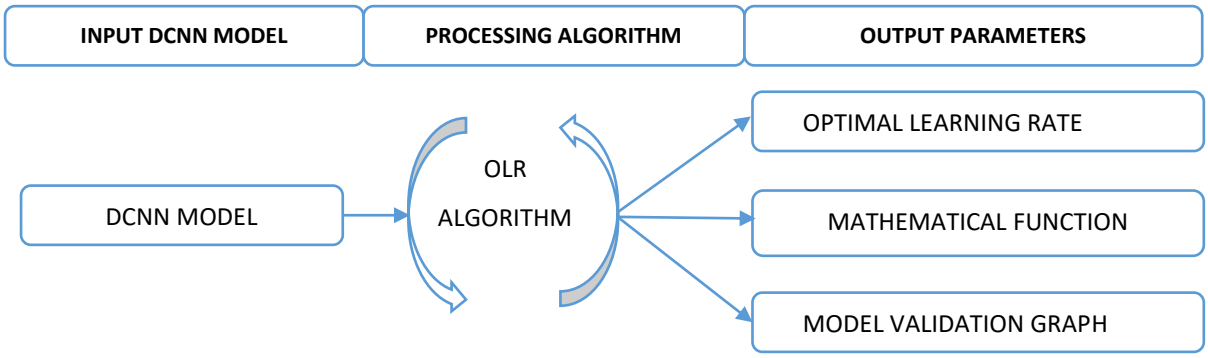


FIGURE 5.1: OLR ALGORITHM ARCHITECTURE

The OLR algorithm uses the prediction from the DCNN model as an input argument, subsequently, the output parameters are generated. These parameters include the optimal learning rate, a mathematical model and a model validation graph. The OLR algorithm uses the input model as a frame of reference, to evaluate the authenticity of the output parameters. The optimal learning rate is evaluated by training the model and archiving the highest model accuracy. This accuracy is expected to match the accuracy obtained when the same learning rate is evaluated on the mathematical function. The same accuracy is also expected to reflect on the model validation graph. The optimal learning rate is expected to yield the same results when evaluated on the test model, mathematical function and the model validation graph.

5.2 EXPERIMENT SETUP

The experiment aimed at identifying an OLR is governed by the OLR algorithm; this algorithm defines the initial conditions required to render the desired output parameters. A test model is accepted as an input into the OLR algorithm; this model serves as a test subject. The experiment describes the learning rate as an independent variable while model accuracy is treated as a dependent variable. The learning rate is manipulated to observe the model performance; subsequently, the model accuracy is recorded in parallel with the respective learning rate. This process is iterated several instances until the predefined range of learning rates has been evaluated. The experiment setup is clearly discussed as a flow diagram for the OLR algorithm. The test DCNN model that will be used in the experiment is discussed below.

5.3 TEST DCNN MODEL

In the proposed model, the optimal learning rate is exploited for an image classification model [2, 3]; in this case, handwritten digits ranging from 0 to 9. The model consists of three convolutional layers and a fully connected deep neural network layer. The convolutional layers extract features and reduce the number of parameters in the input image [26]. Subsequently, the fully connected deep neural network layer flattens the features into a vector, which is classified as a probability distribution of the output digits ranging from 0 to 9[58]. This is illustrated in Figure 5.2 as well as Annexure A.

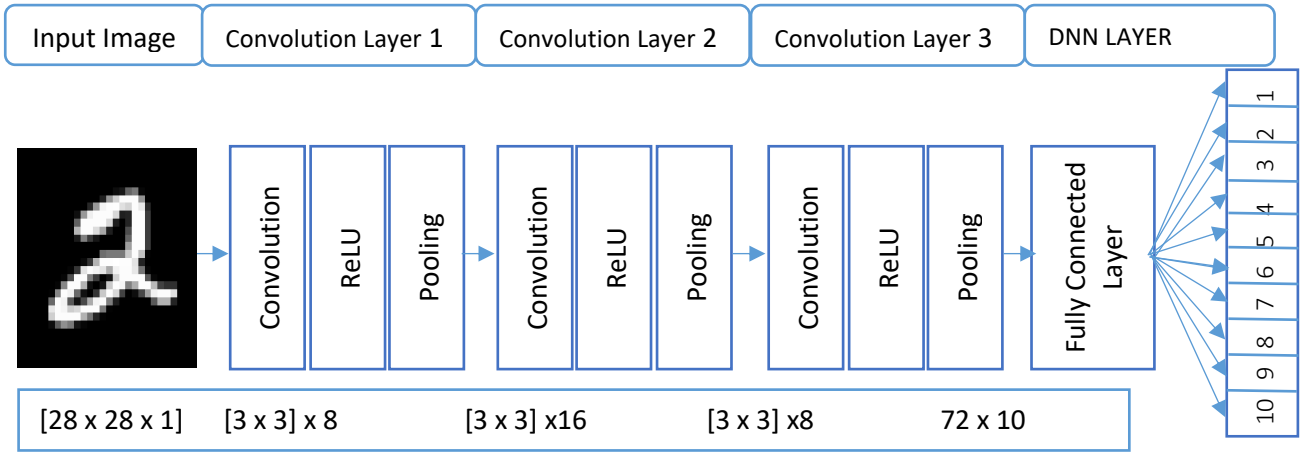


FIGURE 5.2: DCNN MODEL STRUCTURE

Convolution Layer 1 accepts a $[28 \times 28 \times 1]$ input matrix of a handwritten digit; the input image is then convolved by an $8 \times [3 \times 3]$ kernel matrices, which generates an $8 \times [28 \times 28]$ feature map. as illustrated in Figure 5.3. The convolutional operation enhances various features of the input image; these features are necessary to enhance the uniqueness of the input image and allow for high prediction accuracy.

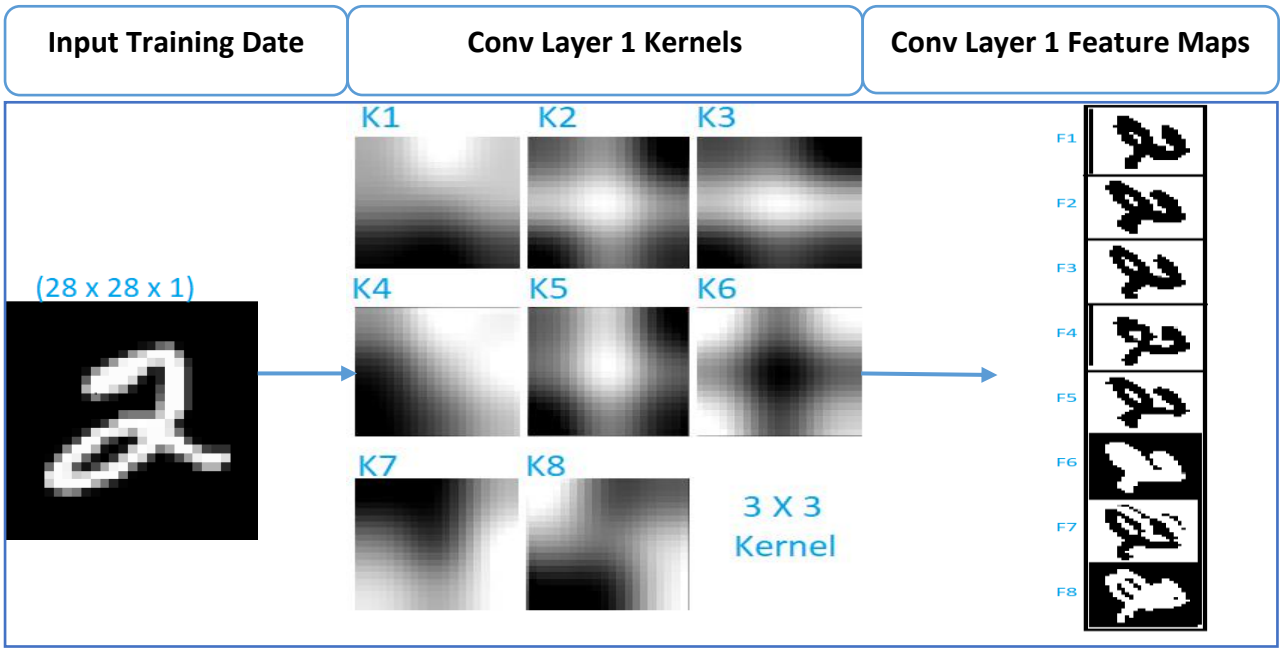


FIGURE 5.3: CONVOLUTIONAL LAYER 2 WITH (8 KERNELS)

Convolution Layer 2 initially combines Convolution Layer 1 feature to form a single feature map matrix; this new feature map matrix is subsequently propagated through a $16\ [3 \times 3]$ kernel matrix producing 16 new feature maps with enhanced features. The feature maps obtained from Convolution Layer 2 become more unclear to the human eye as a result of advanced feature detection. This is illustrated in Figure 5.4.

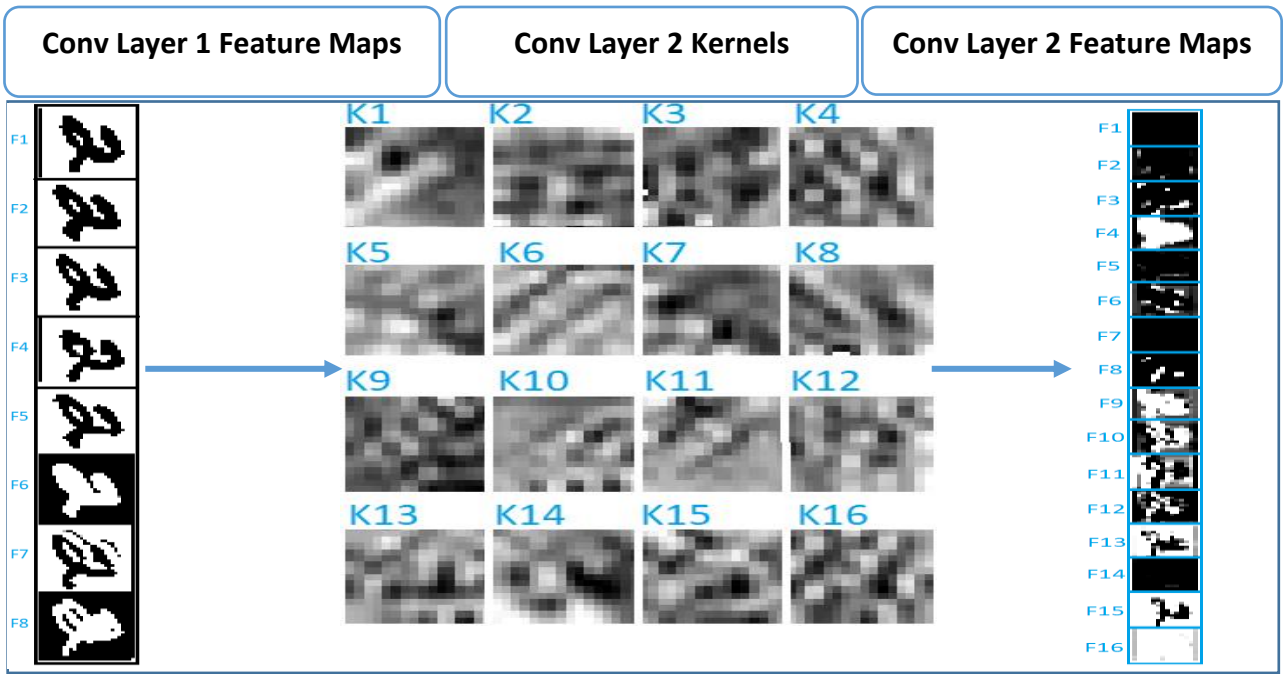


FIGURE 5.4: CONVOLUTIONAL LAYER 2 WITH (16 KERNELS)

Similarly Convolution Layer 3 initially combines Convolution Layer 2 feature to form a single feature map matrix; this new feature map matrix is subsequently propagated through an $8\ [3 \times 3]$ kernel matrices producing 8 new feature maps with more enhanced features; this is illustrated in Figure 5.5.

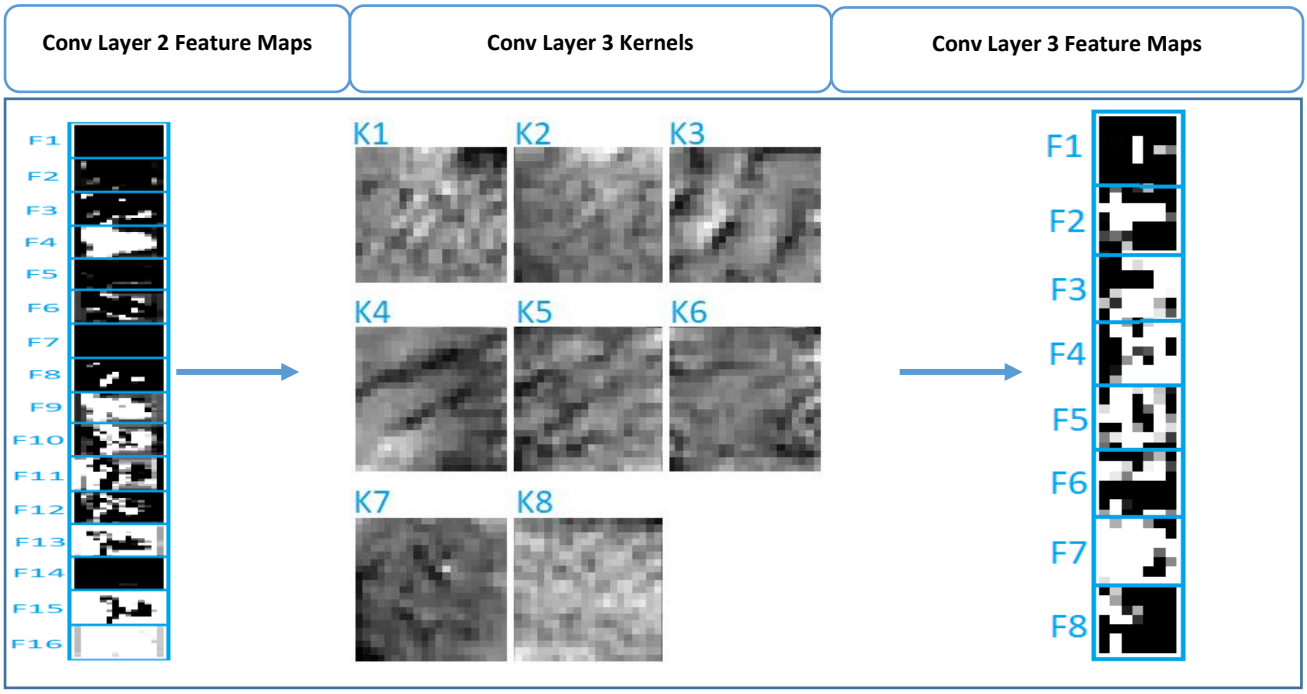


FIGURE 5.5: CONVOLUTIONAL LAYER 3 WITH (8 KERNELS)

The fully connected layer in Figure 5.6 initially takes the 8 feature maps from Convolutional Layer 3 and flattens them into a vector of values ranging from 0 to 255. Subsequently, the vector is propagated through the layer by performing linear transformations until the output is predicted. The output prediction is a vector of ten digits ranging from 0 to 9. The input data is classified as a probability distribution of possible outcomes as described below by the Softmax function.

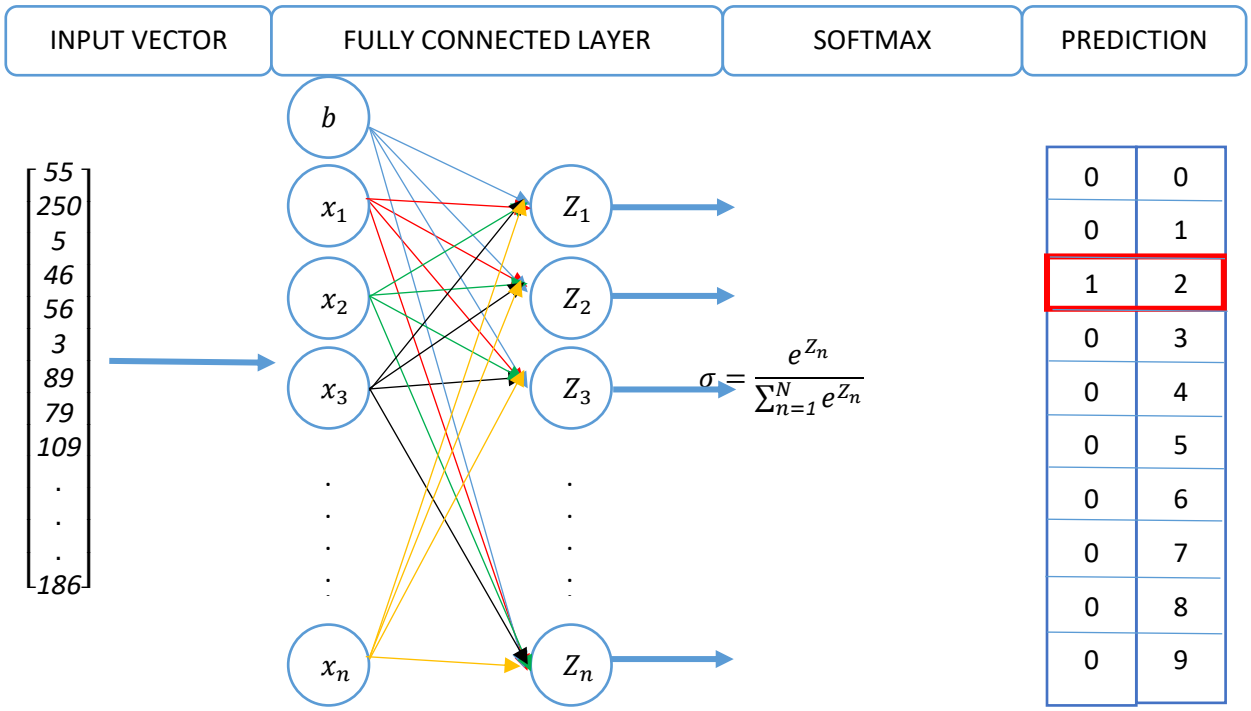


FIGURE 5.6: FULLY CONNECTED NEURAL NETWORK

The final prediction is determined using the Softmax function. which generates a distribution of probabilities that sum up to a value of one. The Softmax function normalises the logits vector by constraining all values to a range between zero and one. The logits vector refers to the last layer of the fully connected layer denoted by Z_n in Figure 5.6. The output of the Softmax function is a vector that expresses the probability distribution for the potential output labels, The number of probabilities is determined by the number of output predictions.

The probability distribution vector is calculated as an exponential function of the output prediction Z_n for the individual logits divided by the sum of all exponential predictions $\sum_{n=1}^{N} e^{Z_n}$. The prediction with the highest output value results in the highest probability value as compared to the other predictions. This prediction describes the output label most likely to march the input data. The next section evaluates the steps implemented by the OLR algorithm to analyse the given data and render the desired output parameters.

5.4 METHODOLOGY (OLR ALGORITHM)

The OLR algorithm consists of six steps as illustrated by the flow diagram in Figure 5.7.

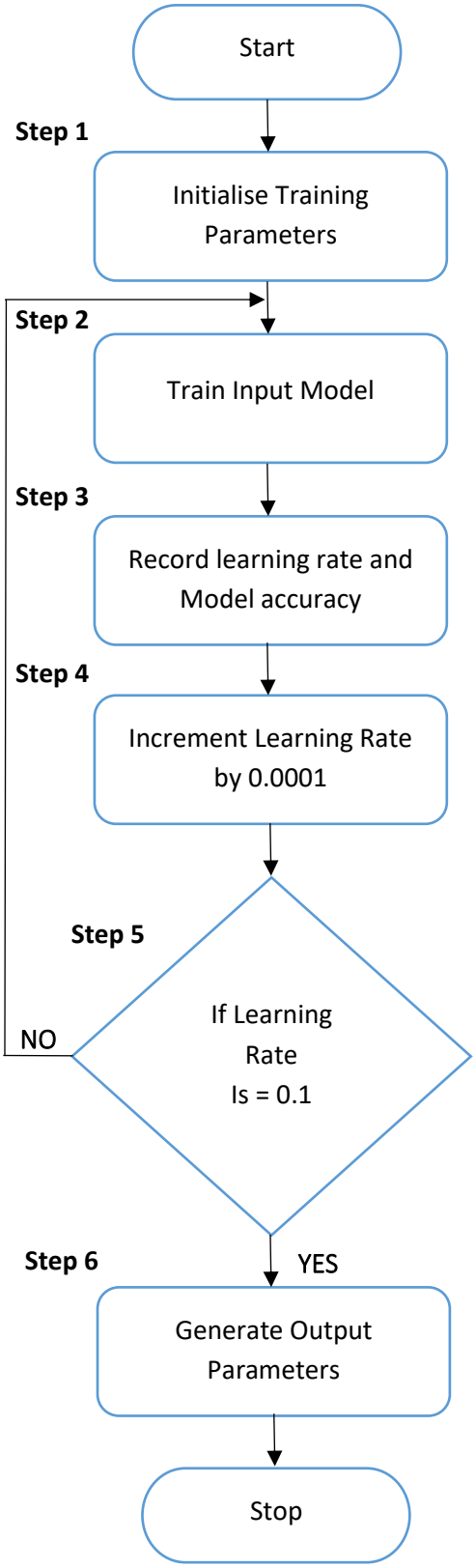


FIGURE 5.7: OLR ALGORITHM FLOW DIAGRAM

5.4.1 STEP 1: INITIALISATION

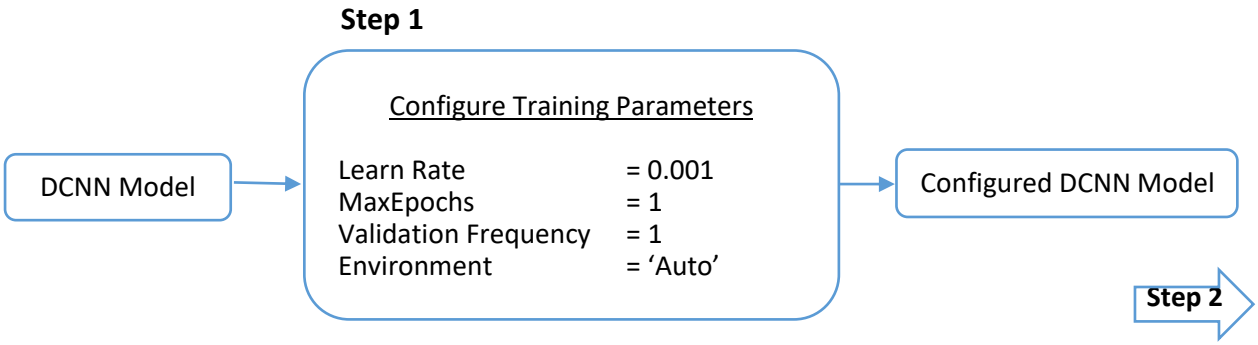


FIGURE 5.8: STEP 1 CONFIGURATION BLOCK

This step consists of a single input, a process block, and a single output; this is shown in Figure 5.8. The input is a DCNN model consisting of convolution layers and a fully connected deep neural network layer. The process block assigns the training parameters to the input model. These parameters determine the nature as well as the duration of the training session. The values assigned to these parameters are meant to fulfil the research objectives as well as the minimum computational resource requirements. These values are discussed below in parallel with the training parameters.

- **Learning rate**: is a hyperparameter that determines the speed at which the model parameters are optimised. The learning rate is initially configured as $\eta_{min} = 0.001$ and is then gradually incremented until the final value is reached; where $\eta_{max} = 0.1$. This range offers an active region of learning rate values, which results in a model accuracy greater than zero. Learning rate values beyond this range saturate the model performance to a value of zero; therefore all values beyond this range are excluded. The initial value of $\eta_{min} = 0.001$ is chosen as a small value that results in a recognisable model accuracy above zero. This value is also not susceptible to the locality phenomenon. The upper learning rate limit of $\eta_{max} = 0.1$ is determined as an average range whereby the model accuracy is saturated and is also susceptible to oscillations around the cost function absolute minimum.

- **MaxEpochs**: is a training parameter that determines the maximum number of full cycles at which the model is exposed to the training data. In the case of the OLR algorithm, the number of epochs is configured to be 1; this value allows the model to learn the input data over a single cycle of training. This cycle provides an accurate measure of the model performance against the learning rate. In critical deep learning systems, the number of epochs may be increased to measure the model behaviour over a number of training cycles. More epochs provide more

details about the model behaviour. A single epoch is a minimum value that may be used to provide any significant measure of the model performance.

- **Validation Frequency**: is a training parameter that determines the frequency at which the model performance is evaluated. In the case of the OLR algorithm, the model performance is measured after a single training example; this increases the resolution at which the model performance is measured. A much larger value may omit some of the data points when measuring the performance of the model. This has the potential of reducing the accuracy of measuring the model performance.

- **Environment**: is a training parameter that determines the training environment. This environment may be configured as one of three states; these include *Auto, CPU,* and *GPU*. When *Auto* is selected the deep learning library selects the available processing environment. The *GPU* options may only be selected if the underlying hardware comprises an NVidia GPU, which is supported by the deep learning library. The *CPU* option is auto-selected if the underlying hardware lacks an NVidia GPU. The OLR algorithm uses the *Auto* option as a default configuration, this allows the library to choose a suitable computational environment.

The output of this step is a configured DCNN model consisting of all the necessary configurations to meet the OLR algorithm training criteria. The configured DCNN model is subsequently propagated to Step 2 of the OLR algorithm.

5.4.2 STEP 2: MODEL TRAINING

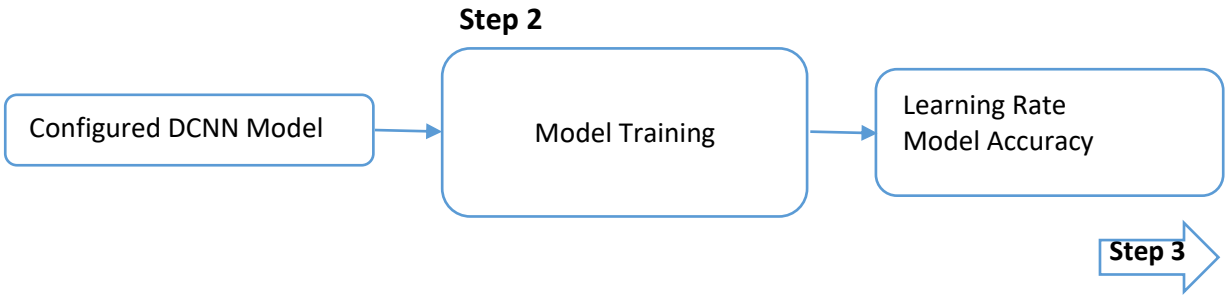


FIGURE 5.9: STEP 2 PROCESS BLOCK

This step consists of a single input parameter propagated from Step 1, a process block. and two output parameters; this is shown in Figure 5.9 as well as Annexure B. The input is a configured DCNN model with the training configurations required to obtain the desired output parameters. The configured

DCNN model is trained in the process block using 900 images of handwritten digits. The training session executes over a single cycle of training data; this is determined by the number of epochs configured in Step 1. The number of training examples depends on the availability of training data. Given an abundant source of training data, this number may be increased to improve the diversity of the model. The output of the process block includes the learning rate used to train the model as well as the resulting model accuracy. The two output parameters are propagated to Step 3 discussed below.

5.4.3 STEP 3: DATA LOGGING

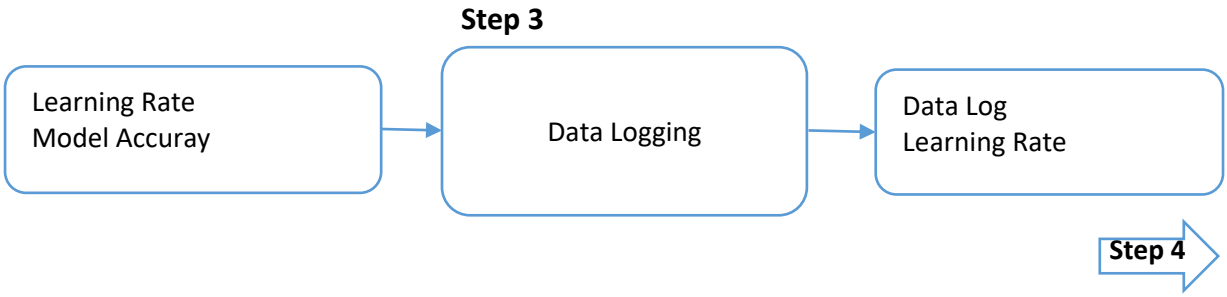


FIGURE 5.10: STEP 3 PROCESS BLOCK

This step consists of two input parameters propagated from Step 2, a process block, and two output parameters; this is shown in Figure 5.10. The input block comprises the learning rate as well as the model accuracy, which is a consequence of the training session in Step 2. These parameters are logged into a matrix variable; whereby the former and latter are populated in the first and second column respectively; this is shown in Annexure C. Owing to the range of learning rates $[\eta_{min} = 0.001, \eta_{max} = 0.1]$ as well as the incremental factor; the length of the matrix variable will consist of 990 rows and 2 columns. The output of the process block includes a data log variable saved in the system memory as well as the input learning rate. The learning rate is subsequently propagated to Step 4 for further analysis.

5.4.4 STEP 4: LEARNING RATE CRITERION

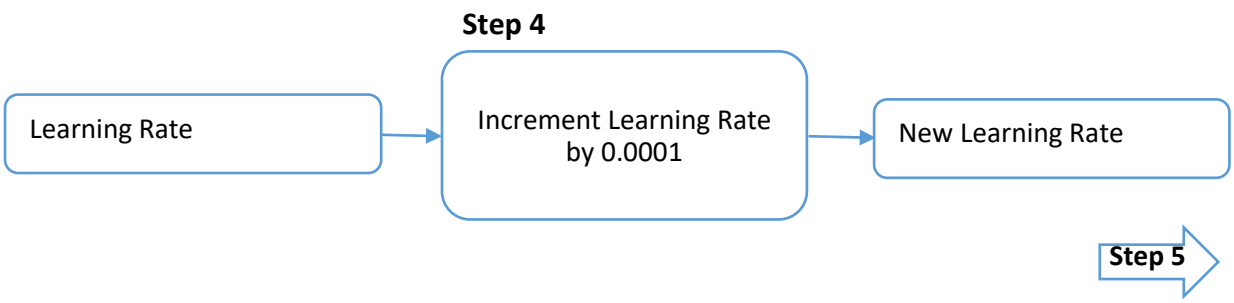


FIGURE 5.11: STEP 4 PROCESS BLOCK

This step consists of a single input parameter propagated from Step 3, a process block, and a single output parameter; this is shown in Figure 5.11. The input learning rate is incremented by a factor of 0.0001 to generate a new learning rate for the next iteration of training. This factor was chosen as a small value that causes a significant change in model accuracy between consecutive training iterations. The lower limit of this value is determined by the saturation in the model performance between two consecutive training iterations. A smaller incremental factor has the potential to produce a more accurate mathematical model as compared to a large incremental factor. Depending on the desired accuracy as well as the availability of computational resources the incremental factor may be manipulated to an even smaller value; given the lower limit condition is satisfied. The new learning rate resulting from the incremental operation is propagated through to Step 5.

5.4.5 STEP 5: ALGORITHM DECISION

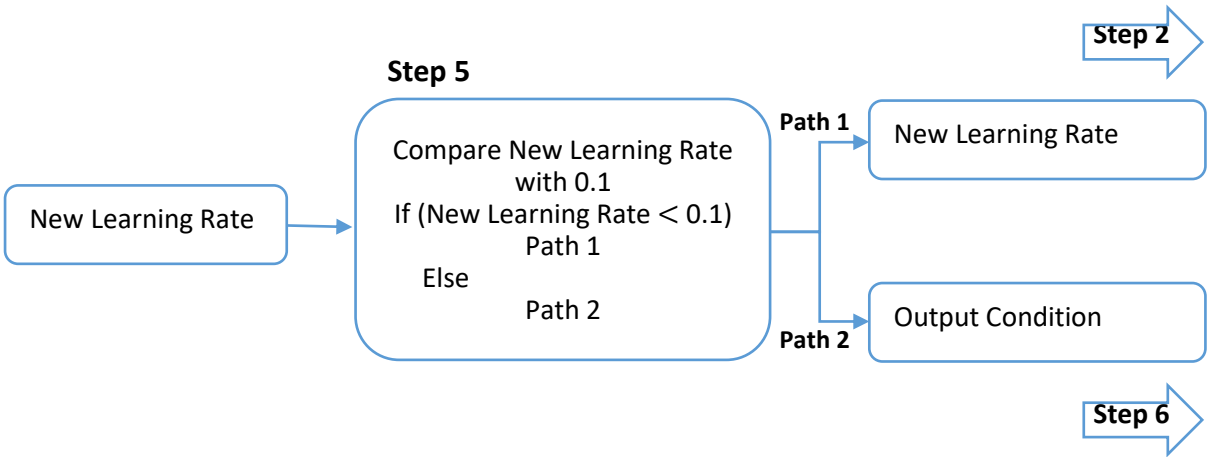


FIGURE 5.12: STEP 5 PROCESS BLOCK

This step consists of a single input parameter propagated from Step 4, a process block, and two dependent output paths; this is shown in Figure 5.12. The input represents the new learning rate calculated in Step 4 as an increment of the original learning rate. The new learning rate is used to select the logic path taken by the algorithm to satisfy the research objective. When the new learning rate is below the upper learning rate limit of $\eta_{max} = 0.1$, then path 1 is selected and the algorithm returns to Step 2 with the old learning rate reconfigured as the new learning rate. When the new learning rate is equal to the upper learning rate limit of $\eta_{max} = 0.1$ then path 2 is selected; subsequently, the condition to generate the final output parameters is triggered, resulting in the propagation to Step 6.

5.4.6 STEP 6: OUTPUT PARAMETER GENERATION

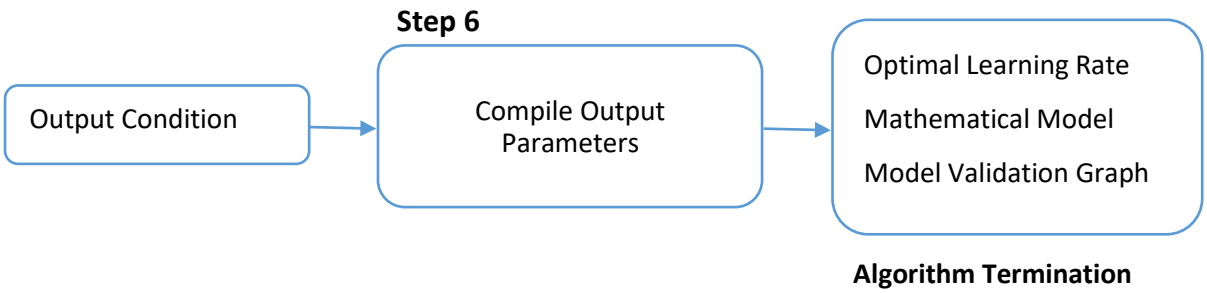


FIGURE 5.13: STEP 6 COMPILING OUTPUT PARAMETERS

The last step of the algorithm comprises a single input parameter, a process block, and three final output parameters; this is shown in Figure 5.13. The input represents the output condition generated in Step 5; this condition informs the algorithm to initiate the development of the final output parameters. The process of developing the final output parameters is discussed in the subsequent section. These parameters include an array of optimal learning rates; a mathematical model as well as a model validation graph. The array of optimal learning rates comprises learning rate values that result in the highest model performance. The mathematical model describes the relationship between the learning rate and model performance. The model validation graph illustrates the same relationship being modelled by the mathematical function. The model validation graph also shows the RIO region; this region describes a range of learning rates that result in a model accuracy greater than 90%; this region provides a much wider range of learning rates, which positively affect the performance of the model.

5.5 COMPILING EXPERIMENT RESULTS

5.5.1 MATHEMATICAL FUNCTION

The mathematical function that describes the relationship between the learning rate and the model accuracy is derived using the least square technique as a 6th order polynomial [68]. The least-square technique defines a line that best fits the input data, which in this case was obtained in Step 3 of the OLR algorithm. The least-squares technique aims to minimise the variance between the values estimated by the polynomial and the expected values from the collected data [69].

The order of the polynomial is determined by how the data is scattered across a two-dimensional plane. When the data points are scattered in the form of a curve more variables are required to approximate the dimensions of the curvature; this is the opposite for data points scattered in a straight

line. Equation 5.1 expresses the nature of the polynomial used to approximate the data captured in Step 3 of the OLR algorithm.

$$f(\eta) = a_6\eta^6 + a_5\eta^5 + a_4\eta^4 + a_3\eta^3 + a_2\eta^2 + a_1\eta + \epsilon \qquad (5.1)$$

A 6th order polynomial has a sufficient number of variables to approximate a complex curve to a high degree of accuracy. Once the order of the polynomial has been established the next step is to compute the coefficients of the polynomial $f(\eta)$. These coefficients represent a system of linear equations defined by the Vandermonde matrix [70]. which is expressed in Equation 5.2.

INPUT MATRIX | VARIABLES | OUTPUT MATRIX

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{N} x_i^k & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \\ \vdots \\ \sum_{i=1}^{N} x_i^k y_i \end{bmatrix}$$

$$N = 990$$
$$k \in R\,[1-6]$$
$$(5.2)$$

The system of linear equations defined by the Vandermonde matrix may be solved using Crammer's rule [71] to obtain the coefficients $[a_0\ to\ a_k]$. The Crammers rule technique resolves the system of linear equations by computing a series of matrix determinants [72], which are subsequently used to compute the desired polynomial coefficients. The first step is to calculate the determinant of the input matrix; the resulting determinant will be denoted as M; this is expressed by Equation 5.3.

INPUT MATRIX

$$M = \begin{bmatrix} N & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{N} x_i^k & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \quad \begin{matrix} N = 990 \\ k \in R\,[1-6] \end{matrix} \qquad (5.3)$$

The next step is to compute the determinant of the input matrix as the output matrix vector is substituted as a vector in the input matrix. The output matrix y is shifted from left to right whilst replacing the existing column and computing the determinant until the last column of the input matrix. This is expressed by Equation 5.4; the output vector y is denoted by the red column.

$$M_i = \begin{bmatrix} \sum_{i=1}^{N} y_i & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i y_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \sum_{i=1}^{N} x_i^k y_i & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \quad \begin{matrix} N = 990 \\ k \in R\ [1-6] \end{matrix} \qquad (5.4)$$

The resulting determinants are denoted as $[M_1, M_2, M_3, M_4, M_5, M_6]$, where M_1 *to* M_6 represents the determinants of the input matrix as the output vector is substituted from columns 1 to 6. These determinants are used to compute the coefficients of the variables a_1 *to* a_6 respectively. The individual coefficients are calculated as the determinant $M_i = [M_1, M_2, M_3, M_4, M_5, M_6]$ is divided by the determinant M; this is expressed in Equation 5.5.

$$a_k = \frac{det(M_i)}{det(M)} \quad where \quad M_i = [M_1, M_2, M_3, M_4, M_5, M_6] \qquad (5.5)$$

Equation 5.6 represents the resulting polynomial that approximates the data collected in Step 3 of the OLR algorithm. As stated earlier the polynomial is a 6th order function with the ability to capture the curvature of the collected data.

$$f(\eta) = (-3.13612 \times 10^7)\eta^6 + (1.0717 \times 10^7)\eta^5 - (1.4265 \times 10^6)\eta^4 + (9.2993 \times 10^4)\eta^3 - (3.1267 \times 10^3)\eta^2 + 51.7175\eta + 0.6287$$

$$valid\ for\ 0.001 \leq \eta \geq 0.1 \qquad (5.6)$$

The accuracy of the function is validated by plotting the function on the same axis as the data collected in Step 3 of the OLR algorithm. The plot in Figure 5.14 illustrates the accuracy of the mathematical model as the learning changes from 0.001 to 0.1 with an incremental factor of 0.0001.

The red line in Figure 5.14 represents the mathematical model while the blue cycles represent the model accuracy for each of the learning rates in the defined range. The function $f(\eta)$ is observed to accurately merge with the collected data; however, there are some minor deviations around the extreme limits of the selected range of learning rates. This is owing to the instability problem resulting

from the large learning rate values. The mathematical model demonstrates that the input model performance may be approximated to a high degree of accuracy.

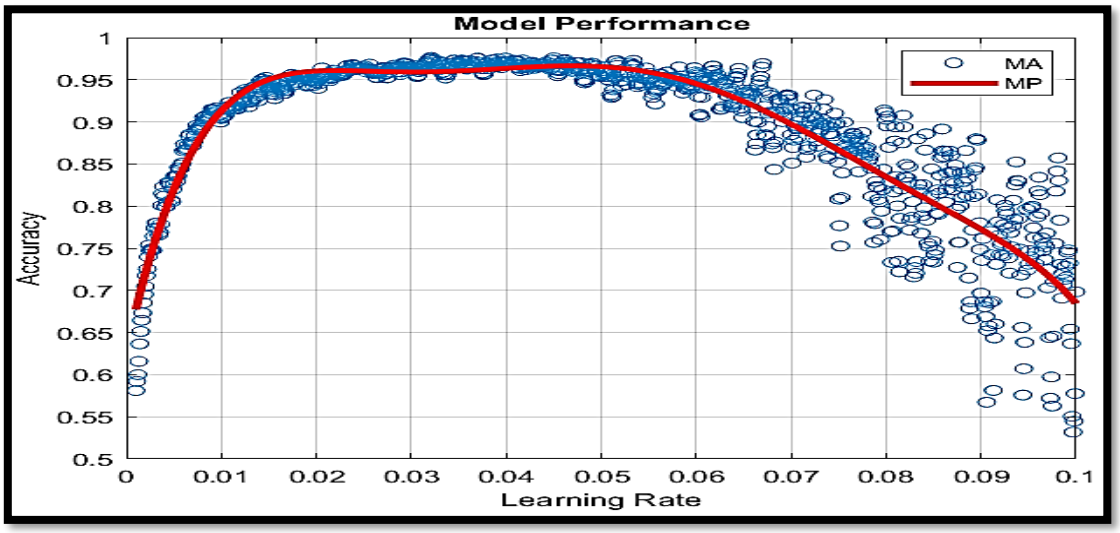


FIGURE 5.14: MODEL PERFORMANCE

5.5.2 FORMULATING THE OPTIMAL LEARNING RATE ARRAY

The optimal learning rate is derived by taking the derivative of the polynomial $f(\eta)$ in Equation 5.6 and solving for the roots of the derivative $f'(\eta)$ as expressed in Equation 5.7.

$$f'(\eta) = (-1.8817 \times 10^{8})\eta^{5} + (5.3582 \times 10^{7})\eta^{4} - (5.7058 \times 10^{6})\eta^{3} + (2.7898 \times 10^{5})\eta^{2} - (6.2533 \times 10^{3})\eta + 51.7175$$

$$valid\ for\ 0.001 \leq \eta \geq 0.1 \qquad (5.7)$$

The root of Equation 5.7, defines the point where the function $f(\eta)$ has the highest model accuracy for a given learning rate; these roots represent the optimal learning rates, which result in the highest model performance. When the function $f'(\eta)$ is plotted against the learning rate range of $[\eta_{min} = 0.001, \eta_{max} = 0.1]$; the roots of the function are observed as $x_n\ where\ n\ \epsilon\ R\ [1 - \infty]$ in Figure 5.15.

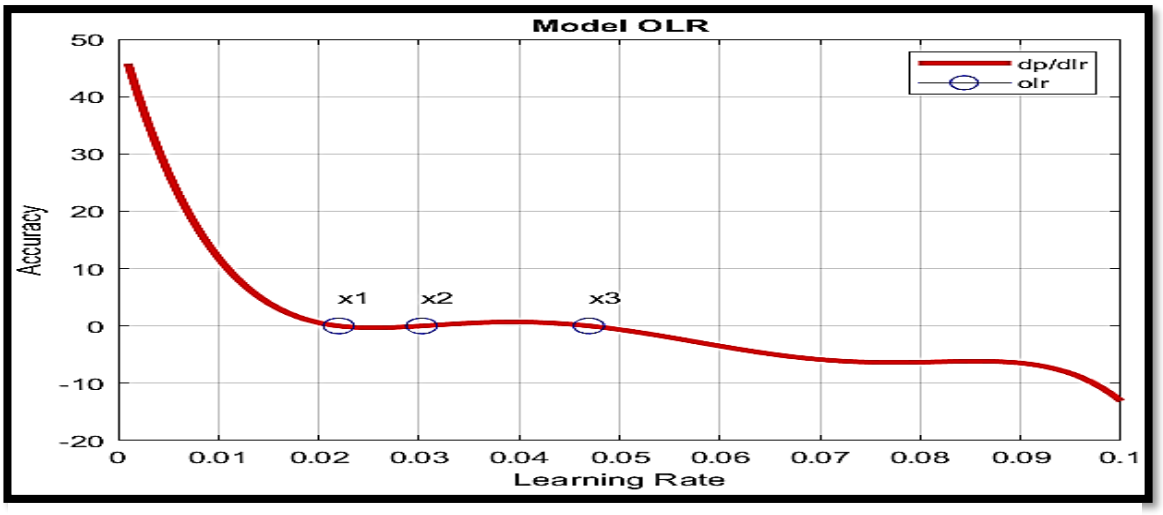


FIGURE 5.15: MODEL OPTIMAL LEARNING RATE

The graph in Figure 5.15 illustrates the change in the model performance in response to the change in the learning rate. The root of the function $f'(\eta)$ defines the highest point in the function $f(\eta)$. As observed the root of $f'(\eta)$ lies where the change in accuracy is zero; this describes the highest peak of the function $f(\eta)$. The negative portion of the function $f'(\eta)$ illustrates the negative gradient resulting from the graph illustrated in Figure 5.14. The array of optimal learning rate is compiled based on the highest peaks of the function $f(\eta)$; these peaks are denoted as the root of the function $f'(\eta)$. Depending on the nature of the input model as well as the response to the learning rate; the number of optimal learning rates may vary considerably.

5.6 EXPERIMENTAL RESULTS

The output of the OLR algorithm includes (A) a mathematical model, (B) an array of optimal learning rates and (C) a model validation graph.

5.6.1 OPTIMAL LEARNING RATE

The optimal learning rate is calculated by taking the 1st order derivative of the function $f(\eta)$, denoted by $f'(\eta)$ and subsequently computing the roots by solving for $f'(\eta) = 0$. The roots of $f'(\eta)$ define the highest points of $f(\eta)$ or, in other words, the optimal learning rate array; for Equation 5.7 the following results were obtained:

$$\eta = [0.0220.\, 0.0303.\, 0.0469].$$

This array of learning rates results in the highest model accuracy. The base model responds more positively towards these values as compared to the other values in the defined range of learning rates.

The accuracy of the learning rates in the array may be validated by observing the region encompassing the array values. The optimal learning rates are said to be accurate when encompassed by the region of interest discussed in the subsequent section.

5.6.2 Model Approximation Function

The model accuracy function $f(\eta)$ defines the relationship between model accuracy and the learning rate. The function allows for the model accuracy to be defined over a wide range of the learning rate values $[\eta_{min} = 0.001, \eta_{max} = 0.1]$.

$$f(\eta) = (-3.13612 \times 10^{7})\eta^{6} + (1.0717 \times 10^{7})\eta^{5} - (1.4265 \times 10^{6})\eta^{4} + (9.2993 \times 10^{4})\eta^{3} - (3.1267 \times 10^{3})\eta^{2} + 51.7175\eta + 0.6287$$

$$valid\ for\ 0.001 \leq \eta \geq 0.1 \quad (5.8)$$

5.6.3 Model Validation Graph

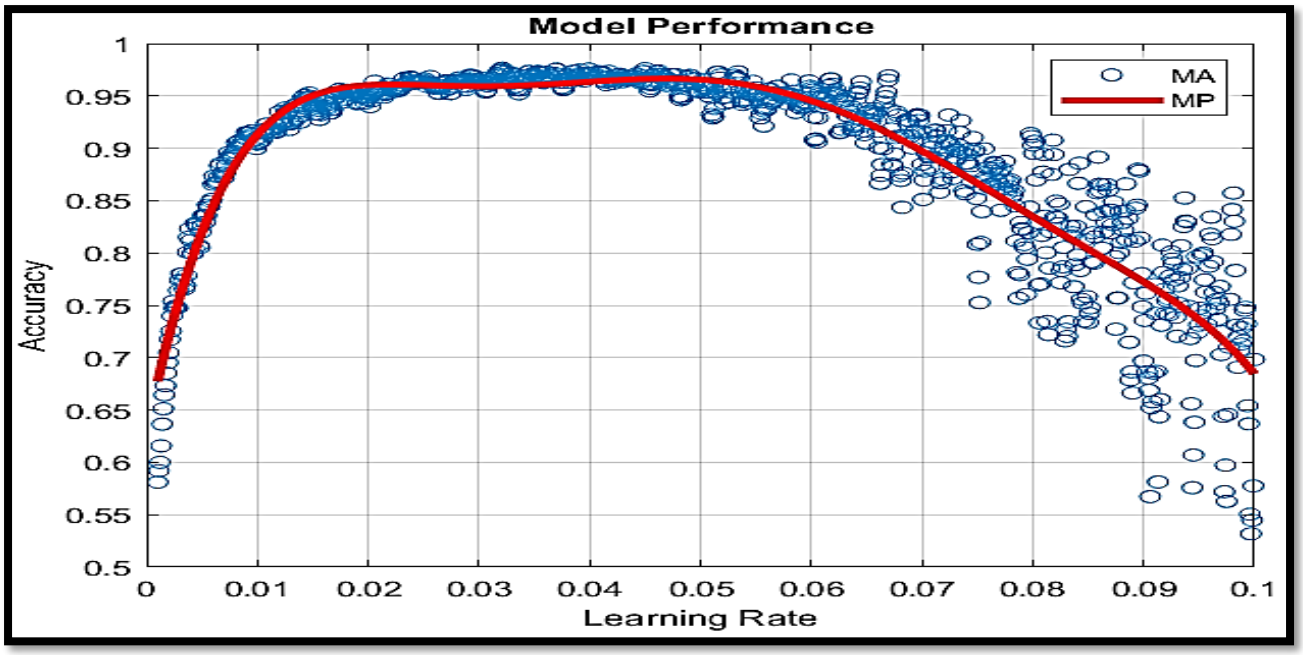


Figure 5.16: Model Performance

The graph illustrated in Figure 5.16 demonstrates how accurate the model polynomial $f(\eta)$ is able to predict the dynamics of the input model.

5.7 SUMMARY

The OLR algorithm attempts to define the properties of the input model in a manner that provides essential data for training the model. The output parameters generated by the algorithm encompass the optimal learning rate that defines the highest model accuracy, the mathematical approximation functions that define the relationship between the model learning rate and the model accuracy and lastly graphical evidence illustrating the approximation function merging with the data. All three output parameters may be used to select the most optimal learning rate that will generate a highly efficient training session whilst rendering a high performing model.

The implementation of these output parameters has eliminated the arbitrary selection of the learning rate, which results in longer training time and unstable oscillations around the absolute minimum. The OLR algorithm may be implemented on various models to evaluate the learning with respect to other model parameters, such as the number of filters, number of layers and the magnitude of the stride. The results obtained in the output of the OLR algorithm will be analysed in great depth in the subsequent section. The subsequent section looks to explain the limitations, improvements and alternatives to the results obtained and how these results compare with the existing body of knowledge.

6 CHAPTER 6: DISCUSSION

6.1 INTRODUCTION

This research proposes a model for determining the optimal learning rate [21] for a deep convolutional neural network (DCNN)[49]. This is meant to resolve several problems that exist when arbitrarily selecting the learning rate hyperparameter [35]; these include the potential to increase training time and reduce model accuracy, which may result in model training instability. The challenges are overcome through the development of an algorithm that searches for an optimal learning rate over a given range of learning rate values $[\eta_{min}; \eta_{max}]$.

The benefits of an optimal learning rate are high model accuracy, reduced training time and an efficient training session. The OLR algorithm attempts to eliminate the need for arbitrarily selecting the learning rate as a critical hyperparameter. The OLR algorithm provides a mathematical framework for selecting an optimal learning rate. The resulting mathematical evidence allows for flexible model tuning within a predefined range of learning rate values $[\eta_{min}; \eta_{max}]$. This chapter will provide an in-depth analysis of the methodology, results obtained and how the obtained results compare to the existing body of knowledge.

6.2 METHODOLOGY ANALYSIS

The challenges stated above are resolved through the development of an OLR algorithm, which analyses the input model to generate a mathematical model. The algorithm accepts a deep convolutional neural network (DCNN) model as an input and generates parameters that describe the model both mathematically and graphically. The algorithm initially trains the input model with a learning rate of $\eta_{min} = 0.001$, the initial learning rate is then incremented by a factor of 0.0001 after a single training epoch. The algorithm records the learning rate in parallel with the respective model accuracy, once the learning rate reaches the upper limit of $\eta_{max} = 0.1$ the algorithm terminates the training thread and begins the process of compiling the output parameters. The algorithm makes use of non-linear regression to establish a relationship between the learning rate and model accuracy. The regression model is subsequently used to render the output parameters and these include an array of optimal learning rates, a mathematical approximating function and a model validation graph. The output parameters are validated by selecting a random learning rate within the range of $[\eta_{min}; \eta_{max}]$ to train the model. The result obtained should be the same for both the mathematical function and illustrated in the model validation graph.

6.3 OBSERVATION AND ANALYSIS

6.3.1 GRAPHICAL DATA

The algorithm records the learning rate along with the respective model accuracy, Figure 6.1 illustrates the model accuracy with reference to the learning rate range from $[\eta_{min}; \eta_{max}]$. Initially, the model accuracy is very low for learning rates between the ranges of [0.001; 0.01]. As observed in the plot below, the highest model accuracy recorded within the range of [0.001; 0.01] is 93% at the learning rate of 0.009.

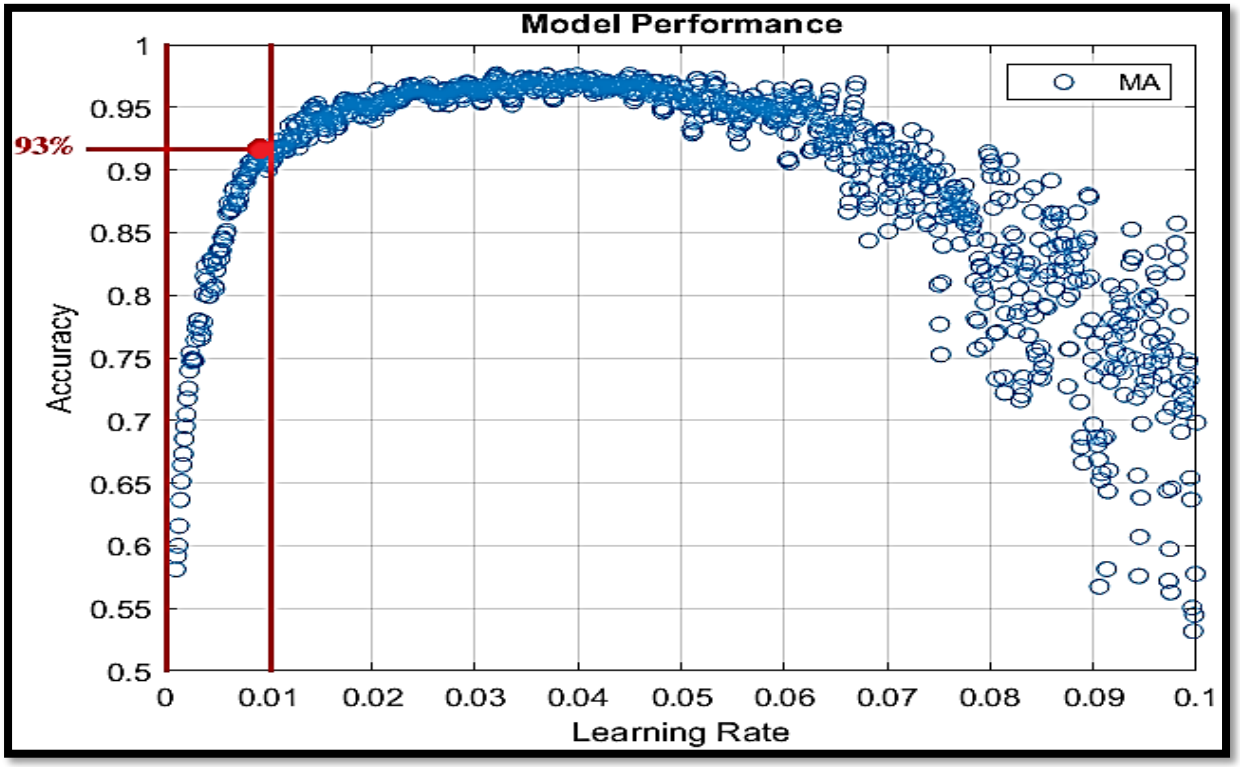


FIGURE 6.1: MODEL PERFORMANCE IN THE RANGE OF [0.001; 0.01]

The low model accuracy within this range is a consequence of the model locality phenomenon [32]. The locality phenomenon occurs when the model is stuck in a local minimum of the cost function, instead of the absolute minimum. The learning rate is too small to contribute any significant update to the weight and biases of the model. When the learning rate reaches a local minimum the model assumes it has reached the absolute minimum as the change contributed by the low learning rate is too small. Figure 6.2 illustrates the locality phenomenon in a two-dimensional plane cost function.

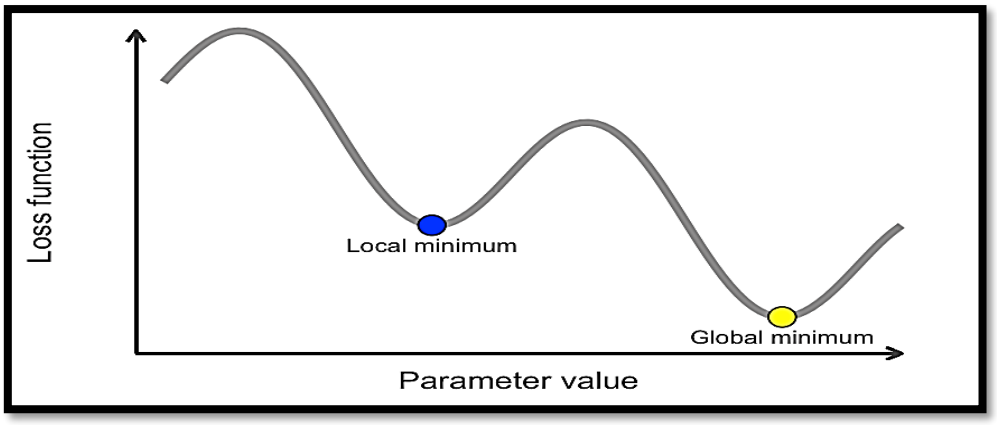


FIGURE 6.2: LOCALITY PHENOMENON [73]

As the learning rate is incremented to the range between [0.1; 0.6], the model accuracy is at its highest peaks. In the context of this research, the range between [0.1; 0.6] is referred to as the ROI. This region consists of several model peaks, these peaks are resulting from the desired optimal learning rate. The peaks in Figure 6.3 below define the learning rate that causes the cost function to converge faster thus rendering a higher model accuracy. The various peaks in the ROI may be visualised below constrained between the two red lines.

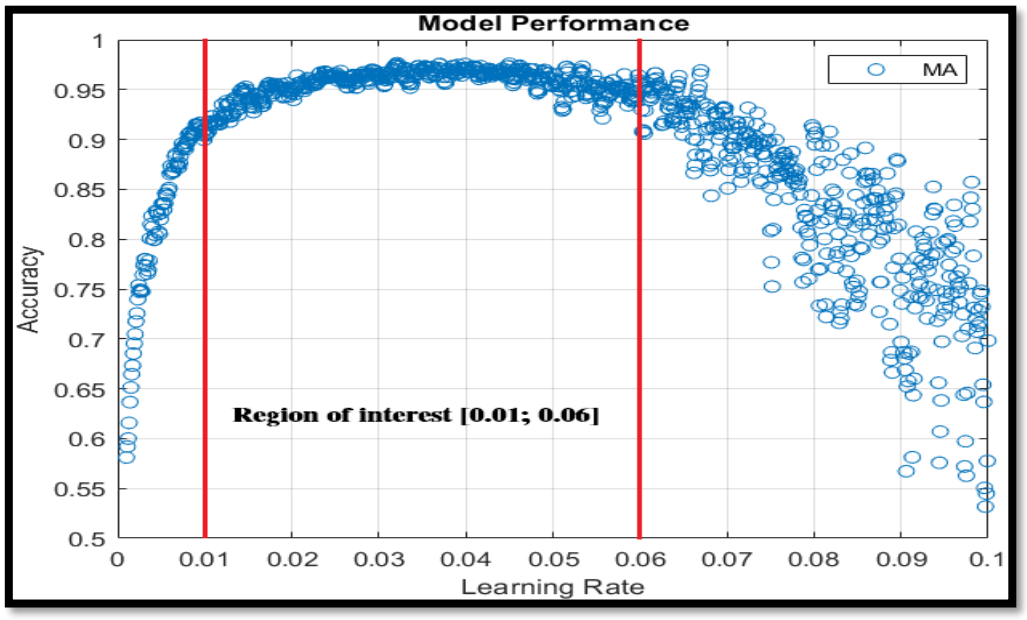


FIGURE 6.3: REGION OF INTEREST WITHIN THE RANGE OF [0.01; 0.06]

The peaks within the ROI results in high model accuracy above 90%. The learning rate within the ROI provides the foundation for selecting an optimal learning rate for the given input model. Once the learning rate is incremented past the ROI region and to the range of [0.06; 0.01], the model accuracy

begins to decline due to the large learning rates. Figure 6.4 below illustrates the decline region as $\eta \geq 0.06$.

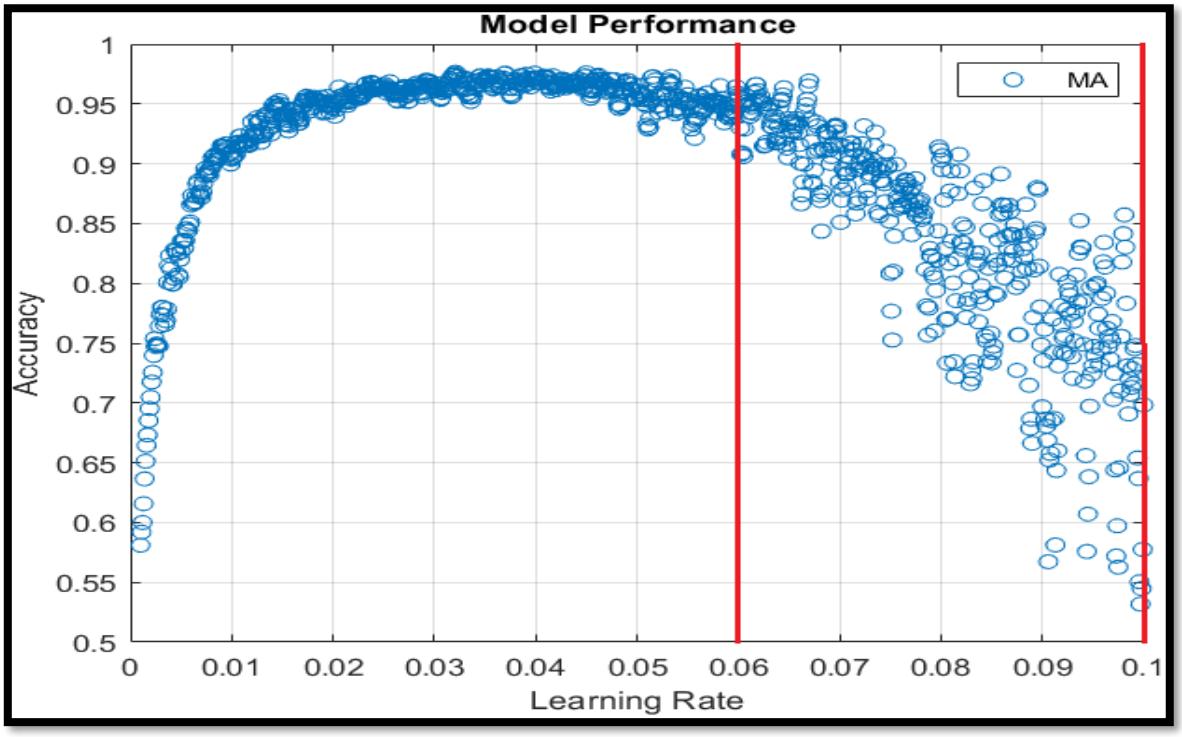


FIGURE 6.4: MODEL PERFORMANCE DECLINE REGION

A large learning rate results in oscillations around the model cost function absolute minimum. When the model cost function is approaching the absolute minimum, the learning rate is contributing significantly to the gradient descent [18]; this is described by Equation 6.1 below.

$$w_{i+1} = w_i + \eta \nabla C(w_i) \tag{6.1}$$

A large learning rate results in an overshoot forcing the gradient descent algorithm to change direction and overshoot in the opposite direction until the training session becomes unstable. The training instability may be visualised using Figure 6.5 below. The model is oscillating around the absolute minimum point as denoted by the red cross.

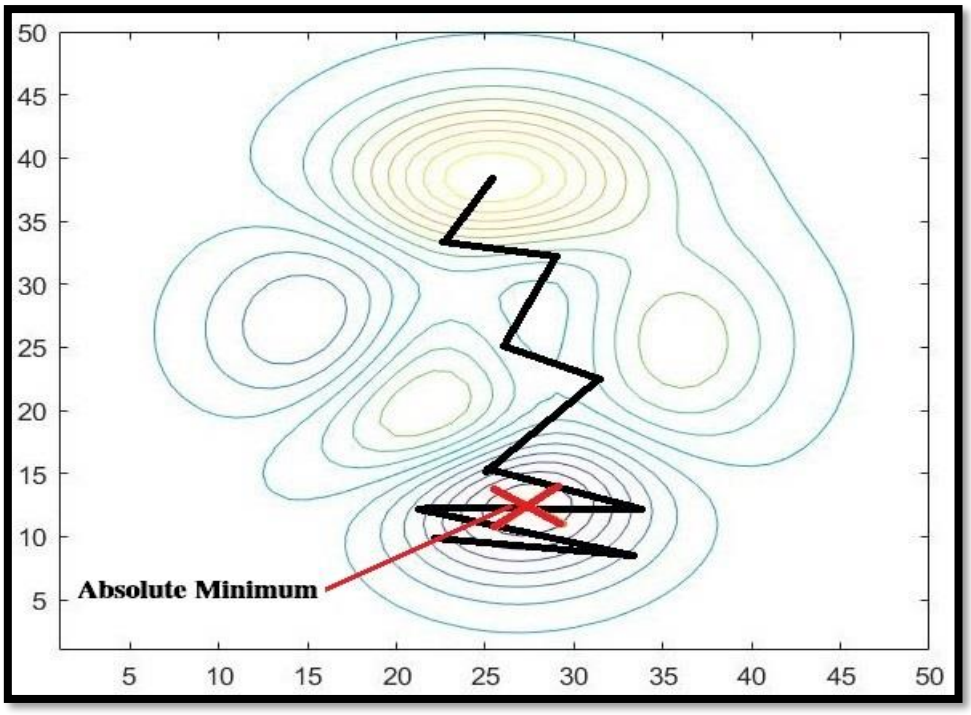


FIGURE 6.5: MODEL TRAINING INSTABILITY [74]

The above instability problem may be resolved by various techniques that reduce the learning rate during the training session. The most common technique includes the use of an adaptive learning rate as well as scheduling the learning rate as a monotonic function [8].

6.3.2 MATHEMATICAL DATA

While the graphs above Figure 6.1 provides clear visualisation for the model performance. Equation 6.2 describes the mathematical notation. which is a 6th order polynomial that allows for the red line in the graph Figure 6.6 to best fit the plotted model accuracy against the learning rate.

$$f(x) = (-3.13612 \times 10^{7})x^{6} + (1.0717 \times 10^{7})x^{5} - (1.4265 \times 10^{6})x^{4} + (9.2993 \times 10^{4})x^{3} - (3.1267 \times 10^{3})x^{2} + 51.7175x + 0.6287$$

$$valid\ for \mid 0.001 < x > 0.1 \qquad (6.2)$$

Similar to the graphical data the equation is valid within the defined range of [0.001; 0.1], the range of the equation defines all the points tested on the input model.

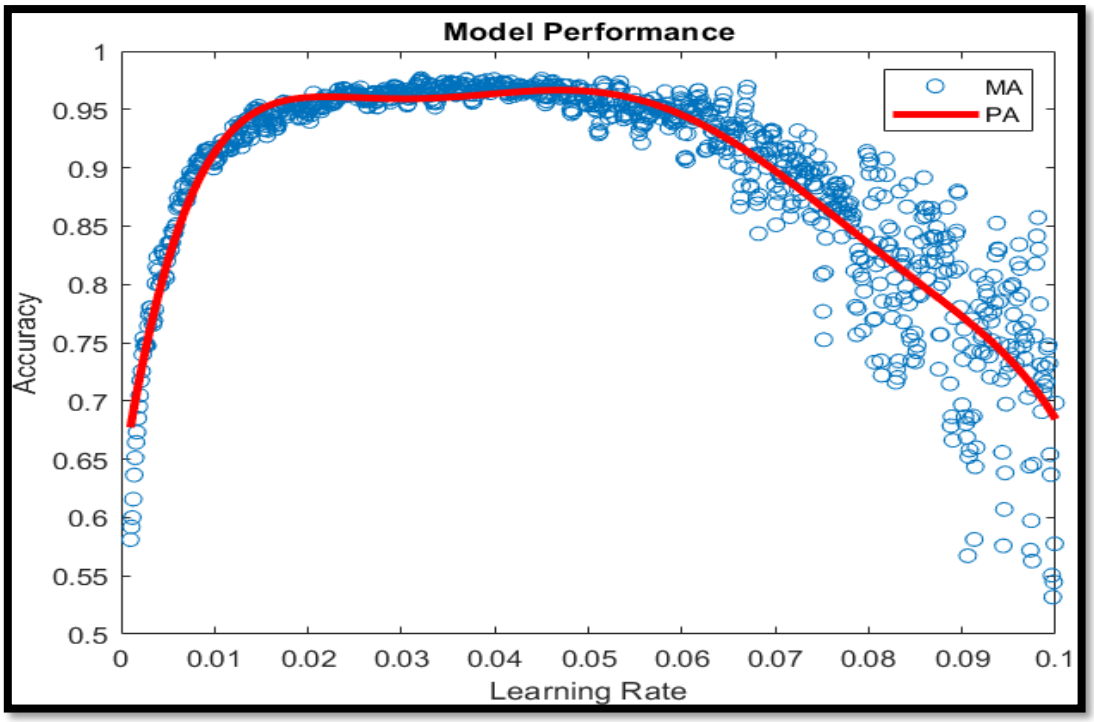


FIGURE 6.6: BEST FIT LINE DEFINED BY $f(x)$

Equation 6.2, which defines the relationship between the learning rate and the model accuracy, is also used to calculate the array of optimal learning rates. The optimal learning rate is calculated by taking the 1st order derivative for the function $f(x)$. The resulting function $f'(x)$ is then evaluated as $f'(x) = 0$ to compute the root of $f'(x)$, which defines the highest points of the original function $f(x)$.

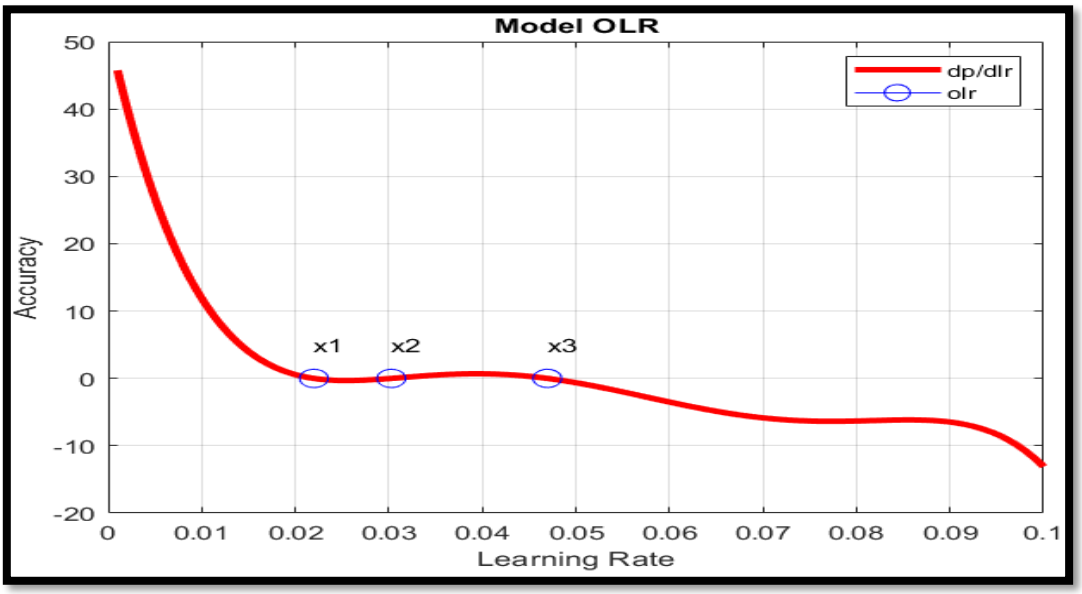


FIGURE 6.7: OPTIMAL LEARNING RATE AS A FUNCTION OF $f'(x)$

The number of roots is dependent on the behaviour of the model with respect to the learning rate. Different input models may generate a different number of roots defining the optimal learning rate; consequently. the size of the array is not fixed.

6.4 COMPARATIVE RESEARCH RESULTS

The literature review discussed several knowledge areas that attempt to define various algorithms and techniques used to obtain an optimal learning rate. This section will evaluate how the proposed techniques compare with the existing body of knowledge. The main objective of the proposed research is to define an optimal learning rate that may improve the model performance of a DCNN. This is accomplished through the use of a non-linear regression model; this approach provides a robust optimal learning rate, which is derived from the performance of the test model. This is compared with the methods that have been previously implemented.

6.4.1 OPTIMIZING NEURAL-NETWORK LEARNING RATE USING A GENETIC ALGORITHM

Kanada [31] developed an algorithm to optimise the learning rate and control the locality problem[32]. The algorithm creates several threads of the same model to train and measure the resulting accuracy [75]. The threads with the lowest model accuracy are terminated and the threads with the highest model accuracy are duplicated and trained again with a learning rate probability distribution of 0.5. This process is iterated several times until the algorithm reaches the highest accuracy possible. Whilst the algorithm manages to find an optimal learning rate, it comes at the cost of computational resources and overhead memory usage.

The creation of several threads requires a high amount of memory, which may not be available in the case of duplicating the research. In comparison, the results obtained by the OLR algorithm provides a comprehensive relationship between the learning rate and the model accuracy whilst utilising low memory and computational units. The research by Kanada [31] is useful in a scenario where high-end computer resources are available. The optimal learning rate generated by the OLR algorithm may also resolve the problem of arbitrarily selecting the initial learning rate.

6.4.2 ALL LEARNING RATES AT ONCE

Leonard et al. [35] proposed an algorithm that randomly initialises the learning rate for all model parameters. All model parameters learn at a different rate with an arbitrarily selected learning rate; the results obtained are described to be comparable with the stochastic gradient descent (SGD) algorithm [18]. The randomly selected learning rate results in unpredictable model performance,

which may either be desired or risky. The algorithm also consumes a large amount of memory as all model parameters have to be trained with a random, unique learning rate.

Selecting the learning rate randomly may be improved by integrating the OLR algorithm to provide both mathematical and graphical evidence of the learning rate to model accuracy. The graphical results may be used to select a range that may result in high model accuracy instead of intuition. The region of interest may be used to randomly initialise the model learning rate. Implementing the region of interest improves the probability of high model accuracy in contrast to absolutely randomising the learning rate.

6.4.3 A Novel Learning Rate Schedule In Optimisation For Neural Networks

Park et al. [37] implemented a learning rate scheduling technique where the learning rate is not based on the number of epochs but is a function of the model cost function. The learning rate is scheduled to change based on the current performance of the model measured as a cost function. This technique provides feedback to drive the learning rate schedule. Whilst feedback learning rate scheduling provides excellent results, it is also susceptible to instability problems. When the model starts training the cost function is large consequently the learning is proportionally large. As a result of the large learning rate, the model tends to be unstable as attempts are made to recover.

The initial learning rate as a function of the cost function may prove to be ineffective in a highly stochastic model. An alternative algorithm may be used to evaluate the behaviour of the model before conducting the actual model training. The OLR algorithm provides results that illustrate the dynamics of the model for various learning rates. Model behaviour may be identified before training and an alternative solution may be implemented.

6.5 RESEARCH LIMITATIONS AND RECOMMENDATIONS

The optimal learning rate research is meant to provide a systematic understanding of the learning rate with respect to the model accuracy. The algorithm that computes the optimal learning rate for a given input model is constrained in different areas that might hinder versatility in other machine learning models. This section will look to examine the limitations of the algorithm, the optimal learning rate and the application.

The OLR algorithm responsible for generating the desired output parameters is limited with respect to the size of the input model. As the number of model layers increases, the algorithm struggles to

cope with the overhead computation. In a more complex model, the algorithm begins to suffer from the same problem it was desired to resolve. The algorithm may be used on a medium to a low model consisting of fewer parameters. In the event of a complex model, the system that is executing the OLR model may require hardware upgrades such as overclocking, a multi-core CPU and a high-end multi-core GPU [76].

The optimal learning rate is defined as the learning rate that causes the model cost function to converge faster in fewer training epochs. The nature of a static optimal learning rate is limited with respect to the application and the desired accuracy. In applications requiring very high accuracy above 99.5 %, the optimal learning rate may struggle to obtain results exceeding 99.5%. As stated above, the learning rate should not be so large as to render the model training session unstable; also the learning rate should not be too small to cause a long training duration. A learning rate within the limit of $[\eta_{min}; \eta_{max}]$ may render excellent results on small to medium applications.

Machine learning may be subdivided into several categories ranging from deep neural networks to reinforcement learning. The OLR algorithm may accommodate most models based on deep neural networks. In models implementing reinforcement learning [77], the OLR algorithm fails to manage the immense amount of computation involved. The OLR algorithm is also not suited for unsupervised learning models [78] that arrange data as clusters or groups. In an unsupervised model, the OLR algorithm may require additional alterations to manage the unsupervised nature of the model.

6.6 SUMMARY

The proposed research looks to define an optimal learning rate for a DCNN model, which is meant to resolve several problems resulting from intuitively selecting a learning rate. An OLR algorithm is developed with the intention to analyse and define the input model both mathematically and graphically. The output results generated by the OLR algorithm prove to be useful in selecting an optimal learning rate based on mathematical evidence. An optimal learning rate has the benefit of improving model performance, improving training time and providing training stability.

An optimal constant learning rate has proven to be an excellent choice in training medium to small deep learning models. As discussed above most algorithms tend to utilise a large amount of computational memory and rely heavily on multi-threading computation [33]. In small applications where the user is concerned with the application rather than the underlying concept, an optimal

learning rate is the desired choice for training any given model. Whilst the OLR algorithm is limited to medium and small applications, it has the potential to be scaled up to meet the desired requirement.

7 CHAPTER 7: CONCLUSION

The scope of this research was to establish an optimal learning rate for an image processing deep convolutional neural network (DCNN). The aim of the research outlined in Chapter 1 was fully achieved as evidenced by the results obtained in Chapter 5, an optimal learning rate was established to improve the model performance, training stability and reduce computational resources. The research aim was achieved as a set of objectives also outlined in Chapter 1, the satisfaction of these objectives are discussed below.

7.1 RESEARCH OBJECTIVES

The first objective was to develop an algorithm that analyses the model of interest within a predefined range $[\eta_{min} = 0.001;\ \eta_{max} = 0.1]$ of learning rates and compile the desired output parameters. An optimal learning rate (OLR) algorithm was developed to accept, analyse and compile output parameters as desired.

The second objective was to compile an optimal learning rate for an image processing deep convolutional neural network (DCNN). Depending on the size and complexity of the input model, the number of optimal learning rates may vary considerably. The optimal learning rates were compiled as the peaks of the function $f(\eta)$, denoting the highest model accuracy. The optimal learning rate value improves model accuracy, training stability, and reduces computational resources.

The third objective was to formulate a mathematical model that describes the relationship between the learning rate and model accuracy. The function $f(\eta)$ was shown to be capable of approximating the model accuracy within the predefined range of learning rates $[\eta_{min} = 0.001;\ \eta_{max} = 0.1]$. The mathematical function was validated as a line that best fits the training data. The mathematical function also satisfies the research hypothesis, which was to determine if the optimal learning could be approximated by a mathematical function.

The final research objective was to compile a model validation graph that could be used to complement the objectives above. A plot of the model performance is shown to merge with the mathematical function $f(\eta)$, to a high degree of accuracy. The graphical evidence also illustrates the region of interest, which results in learning rate values that impact the model performance more positively.

The research objectives were all fully achieved to satisfy the research aim and to verify the research hypothesis. Whilst all research objectives were satisfied there are some limitations to the research.

7.2 RESEARCH AND RESOURCE LIMITATIONS

As stated in Chapter 1 the research attempted to find an optimal learning rate whilst implementing as low computational resources as possible. The hardware system used for conducting the experiments consisted of a Core i5, with 4 hardware cores and 8 logic cores, system random access memory (RAM) of 8 GB, integrated intel graphics card and a 250 GB solid-state drive (SSD); the software platform used was MATLAB 2019b. MATLAB is optimised to run DCNN models on top of an NVidia GPU, however, the computer used in the experiments included an unsupported integrated Intel GPU. The impact of the experiments being conducted on the CPU instead of the GPU resulted in slow training sessions and longer experimentation times than initially estimated. The COVID pandemic of 2020 also limited access to equipment that could have aided in the progress of the research.

The research experiment was conducted using grayscale images of handwritten digits. The experiment was limited to grayscale data due to the extra computational resources required to process the RGB colour dimensions. Non-grayscale images resulted in exponentially long training time in contrast to grayscale images and required more filters in the convolutional layer to capture extra details in the image.

As stated above the experiment was executed on the CPU instead of the GPU, this resulted in tasks being executed in series as compared to the parallel computing offered by the GPU. The over computation executed in the CPU resulted in the CPU experiencing overheating problems. In an attempt to dissipate the heat, the built-in fan will run at full speed for the entire duration of the experiment. This resulted in a system warning of overheating and in some cases, the system will shut down to allow the CPU to cool down before resuming operations.

7.3 RESEARCH RECOMMENDATIONS

In light of the research results outlined in Chapter 6, several changes may be implemented to improve the various sectors impacted. An optimal learning rate may improve both the industrial and academic sectors. Below are recommendations made to the individual sectors impacted by the research result and how these results may bring about change to the existing systems.

7.3.1 INDUSTRIAL SECTOR

The research has demonstrated that the relationship between the learning rate and model accuracy may be modelled to a high degree of accuracy. In industries implementing machine learning to improve plant efficiency, it is recommended to integrate the mathematical function derived in Chapters 5 and 6 to offer more flexible system tuning. The mathematical function is capable of predicting the model behaviour over a wide range of learning rate values.

The region of interest proves that the learning rate is more effective between the range $[\eta_{min} = 0.01;\ \eta_{max} = 0.06]$. The learning rate within this region results in high model performance as compared to learning rate values residing outside of the region of interest, It is recommended to use any learning rate value within the range of the region of interest to train a model as compared to complete intuition. A learning rate in the region of interest has a high probability of producing high model accuracy.

7.3.2 ACADEMIC SECTOR

As more academic literature introduces machine learning as an integral part of the syllabus, it is essential to also introduce system optimisation to improve system efficiency. The mathematical and graphical data outlined in Chapter 5 provide a clear concept of the relationship between the learning rate and model accuracy. Instead of describing the relationship as a mathematical function, it may be visualised to provide a clear understanding of the concept. The OLR algorithm used to compile a model optimal learning rate may also form the foundation for understating which learning rate is better for training the model, this eliminates the idea of intuitively selecting a learning rate value.

7.4 FURTHER WORK

7.4.1 LEARNING RATE WITH MOMENTUM

The learning rate optimisation scheme implemented in this research is based on the SGD optimisation algorithm, which relies on the learning rate and the negative cost function gradient. An alternative variant of the SGD algorithm introduces a momentum hyperparameter to increase the model descent speed. As with the learning rate, the momentum hyperparameter is intuitively chosen to train a model. Further work might exploit the optimisation of the momentum hyperparameter to better impact the model performance. This may require defining a multi-dimensional function that defines the model dynamics for both the learning rate and the momentum hyperparameter.

7.4.2 MONOTONIC WITH FEEDBACK

In Chapter 2 it was identified that a learning rate may be scheduled as a monotonic function to gradually decrease the learning rate over the course of training. It was also mentioned that a scheduled learning rate resembles an open-loop system. Further work may seek to provide a closed-loop scheduled learning rate with an optimal momentum hyperparameter. This will improve the model training and provide training stability.

7.4.3 ADAPTIVE LEARNING RATE

An adaptive learning rate as outlined in Chapter 2, dynamically changes the learning rate as the function of the model gradient concerning the model parameter. Further work may seek to minimise the amount of memory consumed by an adaptive learning rate by reducing the number of gradients used to derive the learning rate update.

8 BIBLIOGRAPHY

[1] S. Allawi. T. Abed Mohammed. and S. Alzawi. "Understanding of a Convolutional Neural Network." 2017. DOI: 10.1109/ICEngTechnol.2017.8308186.

[2] T. Ganegedara. "Intuitive Guide to Convolution Neural Networks." https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-convolution-neural-networks-e3f054dd5daa (accessed 02/05/2020. 2020).

[3] T. Christopher. "An introduction to Convolutional Neural Networks." https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7 (accessed 30/04/2020. 2020).

[4] M. Buscema. "Back Propagation Neural Networks." *Substance use & misuse.* vol. 33. pp. 233-70. 1998. DOI: 10.3109/10826089809115863.

[5] M. E. Jeffrey and B. Richard. "Adaptive Learning Rate Clipping Stabilises Learning." *arXiv:1906.09060v1.* vol. 1. 21 Jun 2019. [Online]. Available: https://www.researchgate.net/publication/333971781_Adaptive_Learning_Rate_Clipping_Stabilises_Learning.

[6] D. Christian and M. John. "Note on Learning Rate Schedules for Stochastic Optimisation." [Online]. Available: https://pdfs.semanticscholar.org/713f/55820406c9540428ae5ec2a0428010d6800c.pdf.

[7] U. Google Inc. and U. New York University. "ADADELTA: AN ADAPTIVE LEARNING RATE METHOD." *arXiv:1212.5701v1.* 22 Dec 2012 2012. [Online]. Available: https://arxiv.org/pdf/1212.5701.pdf.

[8] C.-P. Chen and F. Qi. "Completely monotonic function associated with the Gamma functions and proof of Wallis' inequality." *Tamkang Journal of Mathematics.* vol. 36. pp. 303-. 2005. DOI: 10.5556/j.tkjm.36.2005.101.

[9] C. Wei-Sheng. Z. Yong. J. Yu-Chin. and L. Chih-Jen. "A Learning-rate Schedule for Stochastic Gradient Methods to Matrix Factorisation." *Department of Computer Science National Taiwan University. Taipei. Taiwan*. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/papers/libmf/mf_adaptive_pakdd.pdf.

[10] Y. Rahul and S. Snehanshu. "A novel adaptive learning rate scheduler for deep neural networks." *DOI: 10.13140/RG.2.2.28333.95201.* February 2019. [Online]. Available: https://www.researchgate.net/publication/331208233_A_novel_adaptive_learning_rate_scheduler_for_deep_neural_networks.

[11] A. authors. "LEARNING WITH RANDOM LEARNING RATES." 2019. [Online]. Available: https://openreview.net/pdf?id=S1fcnoR9K7.

[12] K. O'Shea and R. Nash. "An Introduction to Convolutional Neural Networks." *ArXiv e-prints.* 2015.

[13] S. Sakib. Ahmed. A. Jawad. J. Kabir. and H. Ahmed. "An Overview of Convolutional Neural Network: Its Architecture and Applications." 2018. DOI: 10.20944/preprints201811.0546.v1.

[14] G. Nowakowski. Y. Dorogyy. and O. Doroga-Ivaniuk. "Neural network structure optimisation algorithm." *Journal of Automation. Mobile Robotics and Intelligent Systems.* vol. 12. pp. 5-13. 2018. DOI: 10.14313/JAMRIS_1-2018/1.

[15] X. Zhou. "Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation." *Journal of Physics: Conference Series.* vol. 1004. p. 012028. 2018. DOI: 10.1088/1742-6596/1004/1/012028.

[16] S. Sumit. "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way." https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed 02/05/2020. 2020).

[17] A. authors. "RETHINKING LEARNING RATE SCHEDULES FOR STOCHASTIC OPTIMISATION." *ICLR.* 2019. [Online]. Available: https://openreview.net/pdf?id=HJePy3RcF7.

[18] H. Bo-Yang. L. Wei. and W. I-Chen. "Stochastic Gradient Descent with Hyperbolic-Tangent Decay on Classification." *2.* 12 Nov 2018. [Online]. Available: https://arxiv.org/pdf/1806.01593.pdf.

[19] D. Christian and E. M. John. "Note on Learning Rate Schedules for Stochastic Optimisation." pp. 832--838. 1991. [Online]. Available: http://papers.nips.cc/paper/400-note-on-learning-rate-schedules-for-stochastic-optimisation.pdf.

[20] A. Abumallouh. Z. Qawaqneh. and B. Barkana. "A New Cost Function Combining Deep Neural Networks (DNNs) and l2.1-Norm with Extraction of Robust Facial and Superpixels Features in Age Estimation." *Applied Sciences.* vol. 8. p. 1943. 2018. DOI: 10.3390/app8101943.

[21] M. Arlin. "Learning rate and learning rate variance under mastery learning conditions." Ph D. University of Chicago. 1973.

[22] J. Brownlee. "Understand the Impact of Learning Rate on Neural Network Performance." 2019. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[23] S. Sathyanarayana. "A Gentle Introduction to Backpropagation." *Numeric Insight. Inc Whitepaper.* 2014.

[24] A. G. Amin. "Optimisation in Deep Learning." https://medium.com/ai%C2%B3-theory-practice-business/optimisation-in-deep-learning-5a5d263172e (accessed 19 July 2020. 2020).

[25] J. Schmidhuber. "Deep Learning in Neural Networks: An Overview." *Neural Networks.* vol. 61. 2014. DOI: 10.1016/j.neunet.2014.09.003.

[26] I. Namatēvs. "Deep Convolutional Neural Networks: Structure. Feature Extraction and Training." vol. 20. pp. 40-47. 2017. DOI: 10.1515/items-2017-0007.

[27] B. Benowa. Y. Zhan. B. Ghansah. D. wornyo. and F. Banaseka. "A Review of Deep Machine Learning." *International Journal of Engineering Research in Africa.* vol. 24. pp. 124-136. 2016. DOI: 10.4028/www.scientific.net/JERA.24.124.

[28] V. Chandrasekhar *et al.*. "Compression of Deep Neural Networks for Image Instance Retrieval." 2017.

[29] J. Brownlee. "Understand the Impact of Learning Rate on Neural Network Performance." January 25. 2019. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[30] L. Blier. P. Wolinski. and Y. Ollivier. "Learning with Random Learning Rates." pp. 449-464. 2020. DOI: 10.1007/978-3-030-46147-8_27.

[31] Y. Kanada. "Optimizing Neural-network Learning Rate by Using a Genetic Algorithm with Per-epoch Mutations." 2016. DOI: 10.1109/IJCNN.2016.7727372.

[32] N. Ay. "On the Locality of the Natural Gradient for Deep Learning." 2020.

[33] F. Trahay. E. Brunet. and A. Denis. "An analysis of the impact of multi-threading on communication performance." pp. 1 - 7. 2009. DOI: 10.1109/IPDPS.2009.5160893.

[34] "List of Nvidia graphics processing units." https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units (accessed.

[35] B. Leonard. W. Pierre. and O. Yann. "Learning with Random Learning Rates." *arXiv:1810.01322.* vol. 3. 29 Jan 2019. [Online]. Available: https://ecmlpkdd2019.org/downloads/paper/805.pdf.

[36] R. Yedida and S. Saha. "A novel adaptive learning rate scheduler for deep neural networks." 2019. DOI: 10.13140/RG.2.2.28333.95201.

[37] J. Park. D. Yi. and S. Ji. "A Novel Learning Rate Schedule in Optimisation for Neural Networks and Its Convergence." *Symmetry.* vol. 12. p. 660. 2020. DOI: 10.3390/sym12040660.

[38] S. Preuße. H.-C. Lapp. and H.-M. Hanisch. "Closed-loop System Modeling. Validation. and Verification." 2012.

[39] L. Yaeger. R. Lyon. and B. Webb. "Effective Training of a Neural Network Character Classifier for Word Recognition." 1996.

[40] M. Moreira and E. Fiesl. "Neural Networks with Adaptive Learning Rate and Momentum Terms." October 1995. [Online]. Available: https://publications.idiap.ch/downloads/reports/1995/95-04.pdf.

[41] V. P. Plagianakos. G. D. Magoulas. and M. N. Vrahatis. "LEARNING RATE ADAPTATION IN STOCHASTIC GRADIENT DESCENT." [Online]. Available: http://www.dcs.bbk.ac.uk/~gmagoulas/c-7.pdf.

[42] N. Hordri. S. Yuhaniz. and S. M. Shamsuddin. "Deep Learning and Its Applications: A Review." 2016.

[43] S. Sumit. "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way." https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed 19 July 2020. 2020).

[44] E. W. Weisstein. "Convolution." https://mathworld.wolfram.com/Convolution.html (accessed 30/04/2020. 2020).

[45] S. Hasan. S. Boussakta. and A. Yakovlev. "FPGA-Based Architecture for Nine Parallel 2-D MRI Filtering Algorithms." *American Journal of Engineering and Applied Sciences.* vol. VOL. 4.. pp. PP. 566-575.. 2012.

[46] R. Kumar. A. Banerjee. B. Vemuri. and H. Pfister. "Trainable Convolution Filters and Their Application to Face Recognition." *IEEE transactions on pattern analysis and machine intelligence.* vol. 34. 2011. DOI: 10.1109/TPAMI.2011.225.

[47] A. Aguilar-González. M. Arias-Estrada. M. Perez-Patricio. and J. camas anzueto. "An FPGA 2D-convolution unit based on the CAPH language." *Journal of Real-Time Image Processing.* 2015. doi: 10.1007/s11554-015-0535-1.

[48] K. Koutini. H. Eghbal-zadeh. M. Dorfer. and G. Widmer. "The Receptive Field as a Regulariser in Deep Convolutional Neural Networks for Acoustic Scene Classification." pp. 1-5. 2019. DOI: 10.23919/EUSIPCO.2019.8902732.

[49] S. Hu. "Convolution-Neural-Network (CNN) based Deep-Learning in Handwriting Recognition." 2017.

[50] J. Brownlee. "How to Visualise Filters and Feature Maps in Convolutional Neural Networks." https://machinelearningmastery.com/how-to-visualise-filters-and-feature-maps-in-convolutional-neural-networks/ (accessed 30/05/2020. 2020).

[51] N. Mboga. C. Persello. J. R. Bergado. and A. Stein. "Detection of Informal Settlements from VHR Images Using Convolutional Neural Networks." *Remote Sensing.* vol. 9. p. 1106. 2017. DOI: 10.3390/rs9111106.

[52] M. Dwarampudi and N. Reddy. "Effects of padding on LSTMs and CNNs." 2019.

[53] Z. Wei. H. Hu. H.-w. Zhou. and A. Lau. "Characterizing Rock Facies Using Machine Learning Algorithm Based on a Convolutional Neural Network and Data Padding Strategy." *Pure and Applied Geophysics.* pp. 1-13. 2019. DOI: 10.1007/s00024-019-02152-0.

[54] P. Samuels and M. Gilchrist. "Simple Linear Regression." 2014.

[55] N. e. al. "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning." *arXiv:1811.03378.* vol. 1. 2018. [Online]. Available: https://arxiv.org/pdf/1811.03378.pdf.

[56] A. F. Agarap. "Deep Learning using Rectified Linear Units (ReLU)." 2018.

[57] S. Sivarajkumar. "ReLU — Most popular Activation Function for Deep Neural Networks." https://medium.com/@sonish.sivarajkumar/relu-most-popular-activation-function-for-deep-neural-networks-10160af37dda (accessed 25/08/2020. 2020).

[58] B. Gao and L. Pavel. "On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning." 2017.

[59] R. Kartikeya. "Pooling Layer — Short and Simple." https://medium.com/ai-in-plain-english/pooling-layer-beginner-to-intermediate-fa0dbdce80eb (accessed 25/08/2020. 2020).

[60] S. Parth. "Challenges in Deep Learning." https://hackernoon.com/challenges-in-deep-learning-57bbf6e73bb (accessed 29/04/2020. 2020).

[61] A. A. Schreef. "The Perceptron." http://ataspinar.com/2016/12/22/the-perceptron/ (accessed 25/082020. 2020).
[62] T. Klamt and S. Behnke. "Towards Learning Abstract Representations for Locomotion Planning in High-dimensional State Spaces." 2019. DOI: 10.1109/ICRA.2019.8794144.
[63] Matlab. "Mesh." https://www.mathworks.com/help/matlab/ref/mesh.html (accessed 25/08/2020. 2020).
[64] R. Kaleem. S. Pai. and K. Pingali. "Stochastic Gradient Descent on GPUs." *ACM International Conference Proceeding Series.* vol. 2015. pp. 81-89. 2015. DOI: 10.1145/2716282.2716289.
[65] Y. Ming. C.-T. Lin. S. Bartlett. and W.-W. Zhang. "Quantum topology identification with deep neural networks and quantum walks." *npj Computational Materials.* vol. 5. pp. 1-7. 2019. DOI: 10.1038/s41524-019-0224-x.
[66] D. Wilson and T. Martinez. "The need for small learning rates on large problems." vol. 1. pp. 115 - 119 vol.1. 2001. DOI: 10.1109/IJCNN.2001.939002.
[67] S. Zhou and K. K. Lai. "An Improved EMD Online Learning-Based Model for Gold Market Forecasting." *Smart Innovation. Systems and Technologies.* vol. 10. pp. 75-84. 2011. DOI: 10.1007/978-3-642-22194-1_8.
[68] M. Ismail. "-least-square method." 2018.
[69] Neutrium. "FITTING OF A POLYNOMIAL USING LEAST SQUARES METHOD." https://neutrium.net/mathematics/least-squares-fitting-of-a-polynomial/ (accessed 25/05/2020. 2020).
[70] M. Yaici and K. Hariche. "A particular block Vandermonde matrix." *ITM Web of Conferences.* vol. 24. p. 01008. 2019. DOI: 10.1051/itmconf/20192401008.
[71] I. Kyrchei. "Cramer's rule for some quaternion matrix equations." 2010.
[72] I. Kyrchei. "Cramer's rule for quaternionic systems of linear equations." *Journal of Mathematical Sciences.* vol. 155. pp. 839-858. 2008. DOI: 10.1007/s10958-008-9245-6.
[73] "Chapter 13 Deep Learning." https://bradleyboehmke.github.io/HOML/gbm.html (accessed 05/07/2020. 2020).
[74] S. Buthelezi. "Optimal learning rate Images." The Durban University of Technology. 2020.
[75] S. Salehian. J. Liu. and Y. Yan. "Comparison of Threading Programming Models." pp. 766-774. 2017. DOI: 10.1109/IPDPSW.2017.141.
[76] A. Tourani. "CUDA Tutorial - How to Start with CUDA?." 2018. DOI: 10.13140/RG.2.2.22460.49289.
[77] V. Heidrich-Meisner. M. Lauer. C. Igel. and M. Riedmiller. "Reinforcement learning in a Nutshell." pp. 277-288. 2007.
[78] S. Becker. "Unsupervised Learning Procedures for Neural Networks." *Int. J. Neural Syst..* vol. 2. pp. 17-33. 1991. DOI: 10.1142/S0129065791000030.

ANNEXURE A

```
function olr(layer)

    model_data = cnn(layer);

    load(model_data);
    rdp = [];
    p=polyfit(data_log1,data_log3,6);
    df = polyder(p)
    y = polyval(p,data_log1);
    y2 = polyval(df,data_log1);
    rp = roots(p);
    rd = roots(df);
    rd = real(rd);

    tiledlayout(2,2);

    ax1 = nexttile;
    plot(ax1,data_log1,data_log5,'o');
    hold on;
    plot(ax1,data_log1,y,'LineWidth',3,'Color','r');
    hold off;
    grid on;

    rdr = 0;
    title(ax1,'Model Performance')
    xlabel(ax1,'Learning Rate')
    ylabel(ax1,'Accuracy')
    legend(ax1,'MA','MP')

    ax2 = nexttile;
    plot(ax2,data_log1,y2,'LineWidth',3,'Color','r');
    str = ["x1" "x2" "x3" "x4" "x5" "x6" "x7" "x8" "x9" "x10"];
    reset = 1;

    for i=1:1:length(rd)
        if(polyval(df,rd(i))>=-1)&& ((polyval(df,rd(i)))<=1)
            rdp(reset) = rd(i);
            reset = reset+1;
    end
    end

    rdp = sort(rdp);

    for i=1:1:length(rdp)
          hold on
          plot(ax2,rdp(i), 0,'Marker', 'o', 'MarkerSize', 10, 'Color', 'b');
          text(rdp(i),5,str(i))
    end

    model_data2 = cnn2(rdp(1));
    load(model_data2);
```

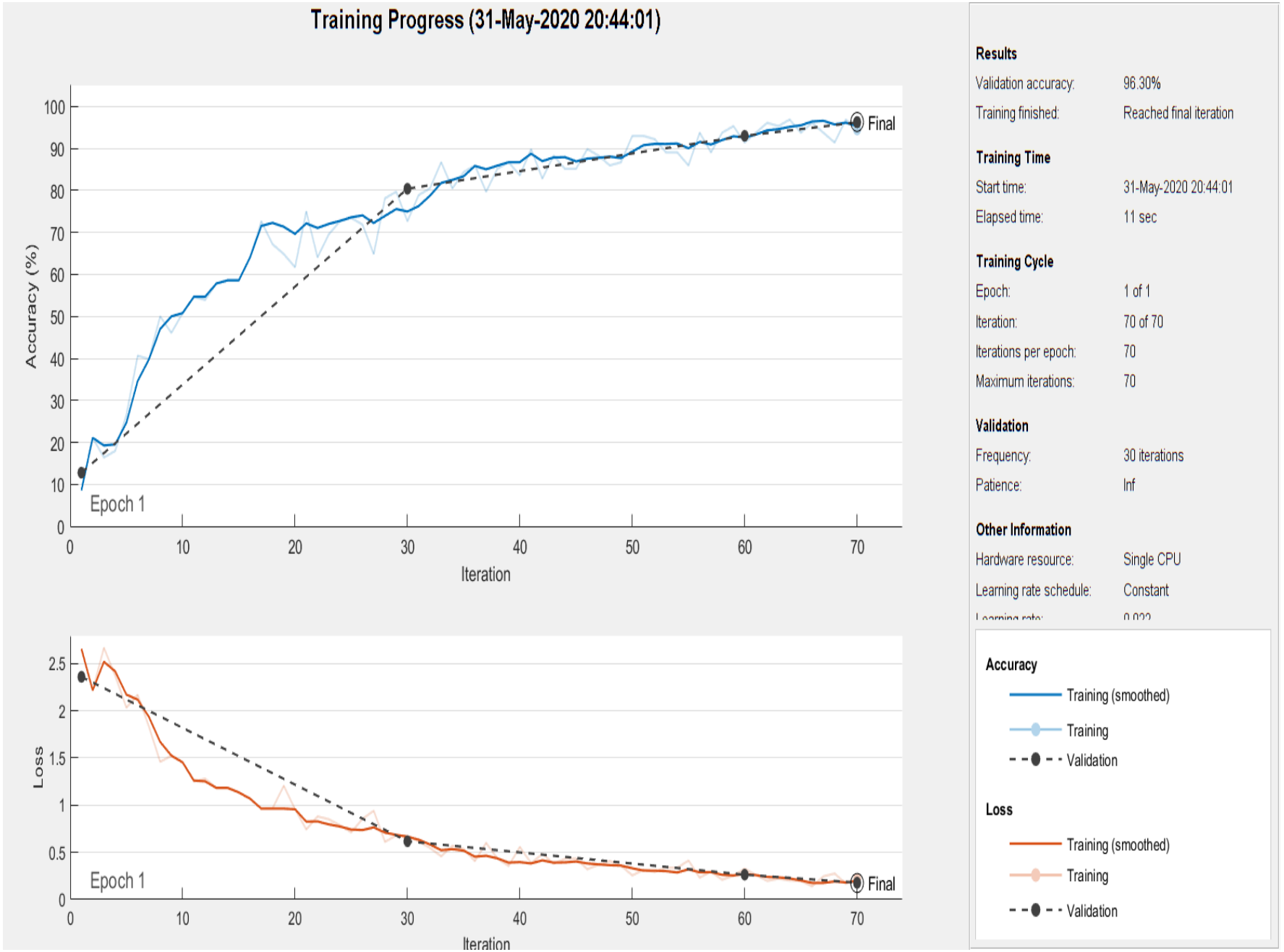
```
title(ax2,'Model Performance')
    xlabel(ax2,'Learning Rate')
    ylabel(ax2,'Accuracy')
    legend(ax2,'dp/dlr','olr')
    hold on
    grid on

    ax3 = nexttile([1 2]);
    plot(ax3,iteration, trainingAccuracy, 'Color', 'b');
    title(ax3,'Model Performance')
    xlabel(ax3,'Iteration')
    ylabel(ax3,'Training Accuracy')

text(iteration(280),trainingAccuracy(280),append('\bullet',num2str(trainingAc
curacy(280))));

end
```

```
function model_data_olr = cnn(layers)

data_log1 = [];
data_log2 = [];
data_log3 = [];
data_log4 = [];
data_log5 = [];
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');


numTrainFiles = 900;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');

layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];

log = 0;
for n = 0.001:0.0001:0.1
    log =log+1;

    plot(data_log1,data_log2,'b--o');

options = trainingOptions('sgdm', ...
    'InitialLearnRate',n, ...
    'MaxEpochs',1, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(imdsTrain,layers,options);

YPred = classify(net,imdsValidation);
```

```
YValidation = imdsValidation.Labels;
accuracy = sum(YPred == YValidation)/numel(YValidation)
data_log1(log,1) = n;
if log ==1
    data_log2(log,1) = accuracy;
    data_log3(log,1) = accuracy;
    data_log4(log,1) = accuracy;
    data_log5(log,1) = accuracy;
    close all force;
elseif log ==2
    data_log2(log,1) = accuracy;
    data_log3(log,1) = (accuracy + data_log2((log-1),1))/2;
    data_log4(log,1) = (accuracy + data_log2((log-1),1))/2;
    data_log5(log,1) = (accuracy + data_log2((log-1),1))/2;
    close all force;
elseif log ==3
    data_log2(log,1) = accuracy;
    data_log3(log,1) = (accuracy + data_log2((log-1),1))/2;
    data_log4(log,1) = (accuracy + data_log2((log-1),1) + data_log2((log-
2),1))/3;
    data_log5(log,1) = (accuracy + data_log2((log-1),1) + data_log2((log-
2),1))/3;
    close all force;
elseif log >=4
    data_log2(log,1) = accuracy;
    data_log3(log,1) = (accuracy + data_log2((log-1),1))/2;
    data_log4(log,1) = (accuracy + data_log2((log-1),1) + data_log2((log-
2),1))/3;
    data_log5(log,1) = (accuracy + data_log2((log-1),1) + data_log2((log-
2),1) + data_log2((log-3),1))/4;
    close all force;
end
end
save ('deep_neural_network_class_hand_cnn01.mat');
model_data_olr = 'deep_neural_network_class_hand_cnn01.mat';
end
```

```
function accuracy = cnn2(olr)

digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');


numTrainFiles = 900;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');

layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm', ...
    'InitialLearnRate',olr, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',1, ...
    'Verbose',false, ...
    'Plots','training-progress', ...
    'ExecutionEnvironment','aut', ...
    'OutputFcn',@(info)stopIfAccuracyNotImproving(info,3));

net = trainNetwork(imdsTrain,layers,options);

YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation)
end
```

ANNEXURE B


Training Progress (31-May-2020 20:44:01)
Accuracy (%)
Iteration
Epoch 1
Final
Loss
Iteration
Epoch 1
Final
Results
Validation accuracy: 96.30%
Training finished: Reached final iteration
Training Time
Start time: 31-May-2020 20:44:01
Elapsed time: 11 sec
Training Cycle
Epoch: 1 of 1
Iteration: 70 of 70
Iterations per epoch: 70
Maximum iterations: 70
Validation
Frequency: 30 iterations
Patience: Inf
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation

ANNEXURE C

Model Performance									
LR	**MA**	**LR**	**MA**	**LR**	**MA**	**LR**	**MA**	**LR**	**MA**
0.001	58.1	0.021	95.25	0.041	97.275	0.061	94.925	0.081	89.4
0.0011	59.2	0.0211	95.775	0.0411	97.275	0.0611	94.3	0.0811	83.525
0.0012	60	0.0212	95.575	0.0412	96.775	0.0612	93.975	0.0812	80.025
0.0013	61.575	0.0213	95.225	0.0413	96.45	0.0613	94.725	0.0813	73.45
0.0014	63.65	0.0214	94.875	0.0414	96.325	0.0614	94.225	0.0814	72.2
0.0015	65.125	0.0215	95.125	0.0415	96.075	0.0615	94.975	0.0815	78.55
0.0016	66.45	0.0216	95.225	0.0416	96.75	0.0616	95.975	0.0816	82.025
0.0017	67.325	0.0217	95.625	0.0417	97.35	0.0617	96.625	0.0817	87.5
0.0018	68.525	0.0218	96.15	0.0418	96.825	0.0618	96.625	0.0818	90.775
0.0019	69.525	0.0219	95.925	0.0419	96.575	0.0619	96.125	0.0819	89.4
0.002	70.475	0.022	96.125	0.042	96.425	0.062	95.375	0.082	83.45
0.0021	71.75	0.0221	95.6	0.0421	96.2	0.0621	95.25	0.0821	84.95
0.0022	72.55	0.0222	95.775	0.0422	96.525	0.0622	95.625	0.0822	83.375
0.0023	73.975	0.0223	95.8	0.0423	96.525	0.0623	95.2	0.0823	82.8
0.0024	75.375	0.0224	95.425	0.0424	96.775	0.0624	95	0.0824	84.675
0.0025	74.725	0.0225	95.8	0.0425	96.95	0.0625	91.475	0.0825	77.15
0.0026	74.925	0.0226	95.6	0.0426	97.05	0.0626	91.525	0.0826	78.325
0.0027	74.875	0.0227	95.7	0.0427	97.075	0.0627	91.525	0.0827	78.675
0.0028	74.75	0.0228	95.5	0.0428	96.75	0.0628	91.8	0.0828	80.525
0.0029	76.425	0.0229	95.7	0.0429	96.6	0.0629	94.7	0.0829	71.6
0.003	77.4	0.023	95.75	0.043	96.35	0.063	94.475	0.083	72.7
0.0031	78	0.0231	96.025	0.0431	96.625	0.0631	94.95	0.0831	72.025
0.0032	77.975	0.0232	96.625	0.0432	96.5	0.0632	93.15	0.0832	73.425
0.0033	77.325	0.0233	96.7	0.0433	96.65	0.0633	93.6	0.0833	87.975
0.0034	76.55	0.0234	96.25	0.0434	96.85	0.0634	93.025	0.0834	82.55
0.0035	76.875	0.0235	96.175	0.0435	96.825	0.0635	93.475	0.0835	81.625
0.0036	77.85	0.0236	96.1	0.0436	96.875	0.0636	94	0.0836	81.925
0.0037	80.075	0.0237	96.425	0.0437	96.55	0.0637	91.725	0.0837	76.775
0.0038	81.5	0.0238	96.8	0.0438	96.65	0.0638	92.075	0.0838	78.875
0.0039	82.325	0.0239	96.725	0.0439	96.875	0.0639	91.775	0.0839	82.55

0.004	81.2	0.024	96.7	0.044	96.55	0.064	93.15	0.084	79.8
0.0041	79.95	0.0241	96.425	0.0441	96.85	0.0641	94.75	0.0841	86.65
0.0042	79.925	0.0242	96.175	0.0442	97.125	0.0642	95.25	0.0842	88.575
0.0043	80.425	0.0243	96.325	0.0443	96.975	0.0643	95.375	0.0843	78.325
0.0044	82.85	0.0244	96.425	0.0444	96.725	0.0644	95.75	0.0844	75.575
0.0045	82.725	0.0245	96.225	0.0445	96.7	0.0645	96.35	0.0845	75.2
0.0046	82.475	0.0246	96.3	0.0446	96.225	0.0646	92.125	0.0846	73.475
0.0047	80.725	0.0247	96.25	0.0447	96.35	0.0647	91.675	0.0847	81.275
0.0048	80.525	0.0248	96.35	0.0448	97.325	0.0648	91.3	0.0848	84.425
0.0049	82	0.0249	96.5	0.0449	97.35	0.0649	90.5	0.0849	75.825
0.005	82.625	0.025	96.65	0.045	97.6	0.065	93.775	0.085	73.35
0.0051	83.575	0.0251	96.625	0.0451	96.925	0.0651	93.4	0.0851	74.225
0.0052	83.5	0.0252	96.2	0.0452	96.175	0.0652	93.55	0.0852	74.75
0.0053	82.95	0.0253	95.9	0.0453	96.1	0.0653	92.775	0.0853	79.15
0.0054	83.6	0.0254	96.25	0.0454	95.475	0.0654	92.625	0.0854	81.925
0.0055	84.575	0.0255	96.4	0.0455	95.725	0.0655	93.375	0.0855	79.025
0.0056	84.35	0.0256	96.85	0.0456	96.15	0.0656	93.125	0.0856	81.55
0.0057	84.4	0.0257	96.2	0.0457	96.05	0.0657	93.775	0.0857	83.65
0.0058	84.925	0.0258	95.975	0.0458	96.425	0.0658	93.3	0.0858	85.75
0.0059	85.125	0.0259	95.825	0.0459	97.075	0.0659	92.575	0.0859	89.15
0.006	86.55	0.026	95.725	0.046	96.825	0.066	91.6	0.086	86.575
0.0061	87.35	0.0261	95.875	0.0461	97.3	0.0661	90.05	0.0861	85.875
0.0062	86.7	0.0262	95.8	0.0462	97.4	0.0662	86.65	0.0862	86.5
0.0063	86.675	0.0263	95.3	0.0463	96.95	0.0663	87.4	0.0863	80.45
0.0064	86.825	0.0264	95.225	0.0464	97.075	0.0664	88.45	0.0864	82
0.0065	87.3	0.0265	95.825	0.0465	96.625	0.0665	89.95	0.0865	83.95
0.0066	88.3	0.0266	95.825	0.0466	96.825	0.0666	94.4	0.0866	82.95
0.0067	88.55	0.0267	95.975	0.0467	96.575	0.0667	94.925	0.0867	86.525
0.0068	87.625	0.0268	96.225	0.0468	96.85	0.0668	95.375	0.0868	86.025
0.0069	87.375	0.0269	96.4	0.0469	96.375	0.0669	96.5	0.0869	84.1
0.007	87.175	0.027	96.35	0.047	95.975	0.067	96.925	0.087	81.925
0.0071	87.65	0.0271	96.475	0.0471	96.125	0.0671	93.7	0.0871	84.075
0.0072	88.35	0.0272	95.725	0.0472	96.275	0.0672	91.85	0.0872	82.7

0.0073	88.975	0.0273	95.65	0.0473	97.05	0.0673	88.475	0.0873	84.875
0.0074	89.6	0.0274	95.2	0.0474	96.525	0.0674	88.275	0.0874	79.65
0.0075	89.075	0.0275	95.25	0.0475	97	0.0675	89.45	0.0875	72.725
0.0076	89.35	0.0276	96.15	0.0476	96.475	0.0676	91	0.0876	75.7
0.0077	89.2	0.0277	95.925	0.0477	96.125	0.0677	93.475	0.0877	75.7
0.0078	89.025	0.0278	96.425	0.0478	97.025	0.0678	93.15	0.0878	80.025
0.0079	89.65	0.0279	96.5	0.0479	96.25	0.0679	91.175	0.0879	83.95
0.008	89.65	0.028	96.425	0.048	96.3	0.068	87.375	0.088	81
0.0081	90.5	0.0281	96.525	0.0481	94.975	0.0681	86.925	0.0881	82.9
0.0082	90.025	0.0282	96.725	0.0482	94.625	0.0682	84.35	0.0882	82.2
0.0083	90.6	0.0283	96.35	0.0483	94.925	0.0683	87.625	0.0883	84.025
0.0084	90.75	0.0284	96.3	0.0484	94.775	0.0684	91.25	0.0884	86.6
0.0085	90.225	0.0285	96.4	0.0485	96.1	0.0685	91.125	0.0885	81.25
0.0086	90.55	0.0286	96	0.0486	96.075	0.0686	92.725	0.0886	83.3
0.0087	90.4	0.0287	95.9	0.0487	96.1	0.0687	92.675	0.0887	71.475
0.0088	90.7	0.0288	95.625	0.0488	96.25	0.0688	92.225	0.0888	67.875
0.0089	90.925	0.0289	95.9	0.0489	96.8	0.0689	92.575	0.0889	68.65
0.009	91.525	0.029	96.55	0.049	96.975	0.069	91.175	0.089	66.625
0.0091	91.425	0.0291	96.575	0.0491	96.95	0.0691	90.025	0.0891	77.15
0.0092	91.475	0.0292	97	0.0492	96.65	0.0692	90.6	0.0892	81.125
0.0093	91.45	0.0293	96.925	0.0493	96.25	0.0693	91.5	0.0893	84.275
0.0094	91.15	0.0294	96.525	0.0494	96.15	0.0694	91.675	0.0894	84.575
0.0095	91.675	0.0295	96.925	0.0495	96.225	0.0695	93.075	0.0895	88
0.0096	91.375	0.0296	96.775	0.0496	96.625	0.0696	93.225	0.0896	87.825
0.0097	90.75	0.0297	96.425	0.0497	95.475	0.0697	92.65	0.0897	81.425
0.0098	90.875	0.0298	96.575	0.0498	95.375	0.0698	93.15	0.0898	78.025
0.0099	89.975	0.0299	96.375	0.0499	95.475	0.0699	91.6	0.0899	74.85
0.01	90.375	0.03	96.075	0.05	95.35	0.07	88.475	0.09	69.675
0.0101	90.475	0.0301	96.25	0.0501	96.075	0.0701	85.1	0.0901	73.55
0.0102	90.6	0.0302	96.4	0.0502	96.55	0.0702	86.95	0.0902	76.15
0.0103	91.65	0.0303	96.675	0.0503	95.775	0.0703	87.825	0.0903	68.65
0.0104	90.75	0.0304	96.475	0.0504	95.375	0.0704	89.55	0.0904	68.05
0.0105	91.45	0.0305	96.25	0.0505	95.825	0.0705	91.45	0.0905	66.875

0.0106	91.8	0.0306	95.375	0.0506	95.675	0.0706	89.15	0.0906	56.725
0.0107	91.425	0.0307	95.5	0.0507	95.025	0.0707	88.175	0.0907	65.225
0.0108	92.325	0.0308	95.95	0.0508	95.325	0.0708	89.15	0.0908	65.775
0.0109	92.325	0.0309	96.225	0.0509	94.85	0.0709	90.1	0.0909	68.525
0.011	91.675	0.031	96.325	0.051	93.225	0.071	90.525	0.091	77
0.0111	91.7	0.0311	96.075	0.0511	92.9	0.0711	91.7	0.0911	74.225
0.0112	91.525	0.0312	96.125	0.0512	93.05	0.0712	91.75	0.0912	68.7
0.0113	91.9	0.0313	96.175	0.0513	93.8	0.0713	91.2	0.0913	58.125
0.0114	91.875	0.0314	96.65	0.0514	95.325	0.0714	91.1	0.0914	64.35
0.0115	91.825	0.0315	96.925	0.0515	97.25	0.0715	89.5	0.0915	66
0.0116	91.625	0.0316	97.15	0.0516	97.275	0.0716	85.8	0.0916	73.1
0.0117	91.525	0.0317	97.05	0.0517	96.7	0.0717	86.7	0.0917	80.75
0.0118	92.1	0.0318	97.35	0.0518	97	0.0718	87.075	0.0918	78.15
0.0119	92.65	0.0319	97.475	0.0519	95.6	0.0719	89.55	0.0919	76.525
0.012	93.5	0.032	97.4	0.052	94.8	0.072	90.1	0.092	74.225
0.0121	92.3	0.0321	97.65	0.0521	94.875	0.0721	90.575	0.0921	75.425
0.0122	92.525	0.0322	97.5	0.0522	94.275	0.0722	91.675	0.0922	75.65
0.0123	92.45	0.0323	97.3	0.0523	95.725	0.0723	90.7	0.0923	74.1
0.0124	92.225	0.0324	97	0.0524	95.75	0.0724	93.175	0.0924	77.8
0.0125	93.85	0.0325	96.375	0.0525	95.925	0.0725	89.7	0.0925	80.1
0.0126	93.45	0.0326	96.225	0.0526	96.15	0.0726	90.325	0.0926	79.425
0.0127	93.425	0.0327	96.425	0.0527	94.425	0.0727	87.75	0.0927	79
0.0128	93.675	0.0328	96.825	0.0528	95.8	0.0728	87.125	0.0928	77.425
0.0129	91.8	0.0329	97.05	0.0529	95.75	0.0729	89.825	0.0929	73.85
0.013	92.325	0.033	97.175	0.053	95.3	0.073	88.875	0.093	72.05
0.0131	92.625	0.0331	96.2	0.0531	96.975	0.0731	89.1	0.0931	76.85
0.0132	92.45	0.0332	95.825	0.0532	96.65	0.0732	89.825	0.0932	75.175
0.0133	94.05	0.0333	95.9	0.0533	96.8	0.0733	90.3	0.0933	77.65
0.0134	94.15	0.0334	95.575	0.0534	97.3	0.0734	91.525	0.0934	80.7
0.0135	93.95	0.0335	95.675	0.0535	96.7	0.0735	92.65	0.0935	78.425
0.0136	93.35	0.0336	95.2	0.0536	95.45	0.0736	89.65	0.0936	82.525
0.0137	92.95	0.0337	95.55	0.0537	94.475	0.0737	89.525	0.0937	85.25
0.0138	93.125	0.0338	95.625	0.0538	94.175	0.0738	85.925	0.0938	83.05

0.0139	92.5	0.0339	96.325	0.0539	94.5	0.0739	85.65	0.0939	82.975
0.014	93.85	0.034	96.925	0.054	94.65	0.074	88.6	0.094	78.525
0.0141	94.85	0.0341	96.4	0.0541	95.625	0.0741	89.375	0.0941	74.9
0.0142	94.625	0.0342	96.55	0.0542	95.2	0.0742	88.575	0.0942	71.825
0.0143	95.075	0.0343	96.675	0.0543	94.35	0.0743	89.925	0.0943	65.6
0.0144	94.775	0.0344	96.9	0.0544	95.425	0.0744	88.325	0.0944	57.575
0.0145	94.5	0.0345	97.35	0.0545	95.45	0.0745	86.825	0.0945	60.7
0.0146	94.825	0.0346	97.325	0.0546	95.05	0.0746	91.025	0.0946	63.825
0.0147	95.275	0.0347	97.075	0.0547	96.4	0.0747	89.675	0.0947	69.725
0.0148	94.975	0.0348	96.85	0.0548	95.4	0.0748	89.85	0.0948	79.65
0.0149	94.6	0.0349	96.575	0.0549	95.075	0.0749	85.2	0.0949	72.425
0.015	94.175	0.035	96.925	0.055	95.775	0.075	80.825	0.095	73.1
0.0151	93.3	0.0351	97.025	0.0551	95.05	0.0751	77.675	0.0951	74.725
0.0152	94.1	0.0352	97.275	0.0552	95.3	0.0752	75.25	0.0952	73.975
0.0153	93.925	0.0353	97.35	0.0553	94.6	0.0753	81	0.0953	79.825
0.0154	93.275	0.0354	97.25	0.0554	92.9	0.0754	83.95	0.0954	80
0.0155	93.625	0.0355	97.2	0.0555	93.325	0.0755	86.175	0.0955	77.425
0.0156	92.75	0.0356	96.575	0.0556	93.85	0.0756	89.95	0.0956	76.25
0.0157	93.2	0.0357	96.35	0.0557	92.125	0.0757	87.2	0.0957	73.2
0.0158	94	0.0358	96.25	0.0558	93.55	0.0758	86.95	0.0958	74.525
0.0159	94.575	0.0359	96.525	0.0559	93.775	0.0759	88.15	0.0959	78.725
0.016	94.475	0.036	96.8	0.056	93.725	0.076	86.325	0.096	81.8
0.0161	94.05	0.0361	97.25	0.0561	95.625	0.0761	86.55	0.0961	83.425
0.0162	93.875	0.0362	97.275	0.0562	95.7	0.0762	86.5	0.0962	81.35
0.0163	93.775	0.0363	96.975	0.0563	94.775	0.0763	87.8	0.0963	79.15
0.0164	94.2	0.0364	97	0.0564	94.5	0.0764	88.05	0.0964	76.225
0.0165	93.375	0.0365	96.4	0.0565	94.325	0.0765	88.675	0.0965	74.675
0.0166	93.7	0.0366	96.475	0.0566	94.5	0.0766	88.55	0.0966	75.55
0.0167	93.375	0.0367	96.775	0.0567	95.125	0.0767	88.275	0.0967	73.675
0.0168	93.7	0.0368	97.175	0.0568	95.325	0.0768	86.85	0.0968	75.175
0.0169	94.875	0.0369	97.525	0.0569	95.925	0.0769	87.6	0.0969	76.8
0.017	94.775	0.037	97.575	0.057	95.925	0.077	86.6	0.097	70.3
0.0171	95.225	0.0371	97.075	0.0571	95.225	0.0771	84.075	0.0971	72.475

0.0172	95.35	0.0372	96.85	0.0572	95.125	0.0772	88.025	0.0972	64.4
0.0173	95.575	0.0373	96.825	0.0573	94.9	0.0773	86.15	0.0973	57.2
0.0174	95.65	0.0374	96.3	0.0574	95.25	0.0774	86.95	0.0974	59.725
0.0175	95.4	0.0375	95.8	0.0575	95.3	0.0775	88.75	0.0975	56.275
0.0176	95.25	0.0376	95.6	0.0576	95.325	0.0776	85.675	0.0976	64.575
0.0177	94.4	0.0377	95.675	0.0577	95.2	0.0777	86.1	0.0977	71
0.0178	94.7	0.0378	96.275	0.0578	94.25	0.0778	86.825	0.0978	75.575
0.0179	94.975	0.0379	96.875	0.0579	94.8	0.0779	86.35	0.0979	81.775
0.018	94.675	0.038	97.05	0.058	95.1	0.078	86.975	0.098	84.15
0.0181	95.55	0.0381	96.725	0.0581	95.2	0.0781	85.35	0.0981	85.725
0.0182	95.525	0.0382	96.325	0.0582	96.525	0.0782	85.575	0.0982	83.025
0.0183	95.35	0.0383	96.7	0.0583	95.45	0.0783	85.975	0.0983	78.325
0.0184	95.125	0.0384	96.725	0.0584	95.125	0.0784	84.5	0.0984	73.1
0.0185	95.3	0.0385	97.35	0.0585	94.7	0.0785	81.15	0.0985	69.075
0.0186	95.3	0.0386	97.35	0.0586	93.425	0.0786	78.125	0.0986	72.025
0.0187	95.275	0.0387	96.95	0.0587	94.575	0.0787	75.675	0.0987	70.725
0.0188	95.55	0.0388	96.65	0.0588	93.975	0.0788	77.925	0.0988	71.3
0.0189	95.2	0.0389	96.525	0.0589	94.175	0.0789	82.925	0.0989	72.85
0.019	94.25	0.039	96.95	0.059	94.175	0.079	82.325	0.099	71.6
0.0191	94.35	0.0391	97.475	0.0591	93.8	0.0791	82.225	0.0991	74.525
0.0192	94.85	0.0392	97.175	0.0592	94.4	0.0792	80.8	0.0992	74.8
0.0193	94.95	0.0393	97.05	0.0593	94.025	0.0793	80.5	0.0993	73.225
0.0194	95.625	0.0394	96.725	0.0594	93.925	0.0794	76.025	0.0994	65.4
0.0195	95.2	0.0395	96.425	0.0595	94.25	0.0795	79.4	0.0995	63.675
0.0196	94.825	0.0396	97	0.0596	94.325	0.0796	81.95	0.0996	55.05
0.0197	94.325	0.0397	97.1	0.0597	95.175	0.0797	84.375	0.0997	53.175
0.0198	94.675	0.0398	97.15	0.0598	96.425	0.0798	91.4	0.0998	54.475
0.0199	94.875	0.0399	97.575	0.0599	95.925	0.0799	91.125	0.0999	57.75
0.02	94.65	0.04	97.2	0.06	95.725	0.08	90.3	0.0991	74.525
0.0201	95.15	0.0401	97.1	0.0601	95.55	0.0801	89.5	0.0992	74.8
0.0202	93.975	0.0402	97.075	0.0602	92.975	0.0802	90.675	0.0993	73.225
0.0203	94.525	0.0403	96.8	0.0603	90.8	0.0803	86.975	0.0994	65.4
0.0204	95.075	0.0404	96.8	0.0604	90.9	0.0804	82.9	0.0995	63.675

0.0205	95.25	0.0405	96.75	0.0605	90.6	0.0805	76.95	0.0996	55.05
0.0206	96.4	0.0406	97	0.0606	92.9	0.0806	73.325	0.0997	53.175
0.0207	95.65	0.0407	97	0.0607	95.375	0.0807	77.05	0.0998	54.475
0.0208	95.425	0.0408	97.325	0.0608	95.2	0.0808	81.675	0.0999	57.75
0.0209	95.45	0.0409	97.175	0.0609	94.925	0.0809	87.725	0.1	69.825

TABLE 8.1 MODEL PERFORMANCE DATA SET