MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

SUMY STATE UNIVERSITY

FACULTY OF ELECTRONICS AND INFORMATION TECHNOLOGY

COMPUTER SCIENCE DEPARTMENT

"Approved for defense."
Acting Head of the Department

_____ Igor SHELEHOV
(signature)

GRADUATION THESIS

for obtaining the educational degree of Bachelor

in the specialty 122 - Computer Science,
educational-professional program "Informatics"
on the topic: " Optimizing Neural Networks For Parameter Efficiency"
by the student of group IN-05AH, UDE VICTOR SOMUADINA.

The Bachelor Graduation Thesis contains the results of original research. The use of ideas, results, and texts of other authors is properly referenced to the respective sources.

_____ **UDE VICTOR SOMUADINA**
(signature)

Supervisor
Doctor of science, professor **KOLESNIKOV VALERII** _____
(signature)

Sumy – 2024

SUMY STATE UNIVERSITY
FACULTY OF ELECTRONICS AND INFORMATION TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

«Approved»
Acting Head of the Department
_____ Igor SHELEHOV
(signature)

TASK FOR THE GRADUATION THESIS

to obtain the educational degree of Bachelor
in the specialty 122 - Computer Science,
educational-professional program "Informatics"
by the student of group IN-05AH, UDE VICTOR SOMUADINA.

1. Topic of the Bachelor Graduation Thesis: " Optimizing Neural Networks For Parameter Efficiency" approved by the order of SumDU on _____
2. The deadline for the submission of the Bachelor Graduation Thesis_____.
3. Input data for the qualification work
4. Table of Contents for the Explanatory Memorandum (List of questions to be addressed) *1) Introduction. 2) Literary Review. 3) Optimization Algorithm 4) Proposed Methodology.*
5. List of graphic materials (with specific mention of mandatory drawings)
6. Project consultants (with the corresponding sections of the project they are associated with).

Section	Consultant	Signature, date	
		The assignment has been issued	The assignment has been accepted

7. Date of assignment issuance «____» _____ 20 ___

The assignment has been accepted for execution _____ (signature)

Supervisor _____ (signature)

Calendar Plan

№	Titles of the stages of the Bachelor Graduation Thesis	Deadline	Note
1	*Introduction*		
2	*Literary Review*		
3	*Optimization Algorithm*		
4	*Proposed Methodology*		
5	*Bachelor Graduation Thesis Formatting*		

Student _____ (signature)

Supervisor _____ (signature)

ABSTRACT

Note: 33 pages, 6 figures, 2 tables, 23 references, 1 app.

Justification of the relevance of the work's theme - Modern neural networks are highly over-parameterized. And what does it mean by this? Simply put, It means that a model has more parameters that seemingly exceed the dataset for training. Some Pruning techniques have the capability to remove a significant fraction of network parameters with little loss in accuracy. Recent methods based on dynamic reallocation of non-zero parameters have emerged allowing direct training of sparse networks such as Stochastic gradient descent (SGD), AdaDelta, AdaMax and Adabelief. We will be using models that require shorter training times and lower hardware requirements and have removed the hyperparameters to observe that the optimiser works with various models

Object of study: Process of Optimizing Neural Networks For Parameter Efficiency

Objective of study: The aim of this research is to provide an in-depth understanding of the modern neural networks, its role in various organizations, and its impact on operational efficiency information systems.

Research methods: Comparative analysis, experimental research, literature reviews, deductive analysis.

Results: An algorithm for optimizing the correspondence of the neural network model to the training data was proposed. The performance criteria of the neural network in the training and validation set are defined. The algorithm is implemented in software, which provides a better understanding of the step-by-step process of learning the model and implementing the optimization algorithm.

NEURAL NETWORK, TRAINING SET, VALIDATION SET, VALIDATION LOSS, TRAINING LOSS

CONTENTS

INTRODUCTION

Justification of the relevance of the work's theme - Modern neural networks are highly over-parameterized. And what does it mean by this? Simply put, It means that a model has more parameters that seemingly exceed the dataset for training. Some Pruning techniques have the capability to remove a significant fraction of network parameters with little loss in accuracy. Recent methods based on dynamic reallocation of non-zero parameters have emerged allowing direct training of sparse networks such as Stochastic gradient descent (SGD), AdaDelta, AdaMax and Adabelief. We will be using models that require shorter training times and lower hardware requirements and have removed the hyperparameters to observe that the optimiser works with various models

Object of study: Process of Optimizing Neural Networks For Parameter Efficiency

Subject of study. The main subject of this study is methodological aspects of neural network optimization.

Novelty. An algorithm for optimizing the correspondence of the neural network model to the training data was proposed. The performance criteria of the neural network in the training and validation set are defined. The algorithm is implemented in software, which provides a better understanding of the step-by-step process of learning the model and implementing the optimization algorithm.

Structure. This work consists of an introduction, a literary review, an optimization algorithm, a proposed methodology, conclusions, a list of used sources, and appendices.

.

1 LITERARY REVIEW

Neural Network can be referred to as a computational model which is inspired by the structure and functioning of the human brain. The model is designed to process information by learning from examples and recognising patterns in data, particularly in complex and high-dimensional datasets, just in a similar pattern to the human brain. Recent findings suggest that backpropagation is related to the synaptic updating method of the human brain bringing in the possibility for insight into understanding more of the exact learning mechanism in the cortex. However, it remains far from fully understanding the human brain's learning mechanics. However, these networks are widely used in different fields such as natural language processing, computer vision and speech recognition.

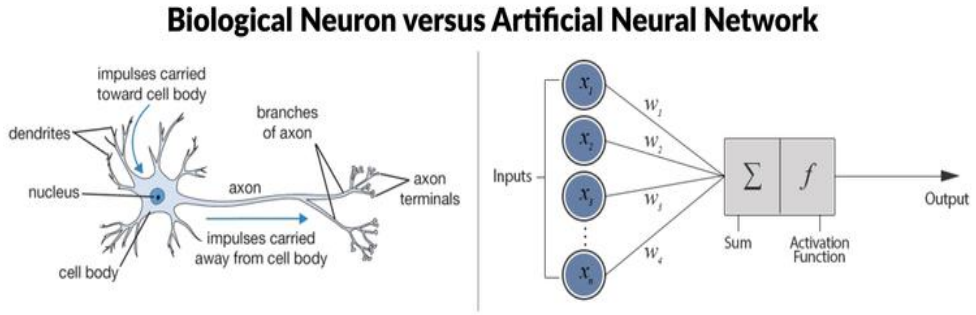


Figure 1.1 – Biological Neuron and Artificial Neuron

There are different types of neural networks which are popularly used today. Such as;

Multilayered Perceptrons(MLPs), Recurrent Neural Networks(RNN) and Convolutional Neural Networks (CNN).

1.1 Multilayered Perceptrons (MLPs)

MLPs go like the name states, a neural network with multiple and fully connected layers its layers are deeply connected and usually consist of the Input layer, the Hidden layer and the Output layer. The input layer consists of neurons that receive the initial

input data and is determined by the dimensionality of the input of data whereas the hidden layer is made up of layers of neurons which are in between the input and output layer. Each neuron in the hidden layer receives input from the neurons in the previous layer(the input layer or possibly another hidden layer) and produces output that is passed to the next layer and the output layer consists of the neurons that produce the final output for the neural network and the output number depends solely on the nature of the tasks. MLPs use a Backpropagation algorithm for training the model which increases the output by reducing the error in the predicted output and actual output.

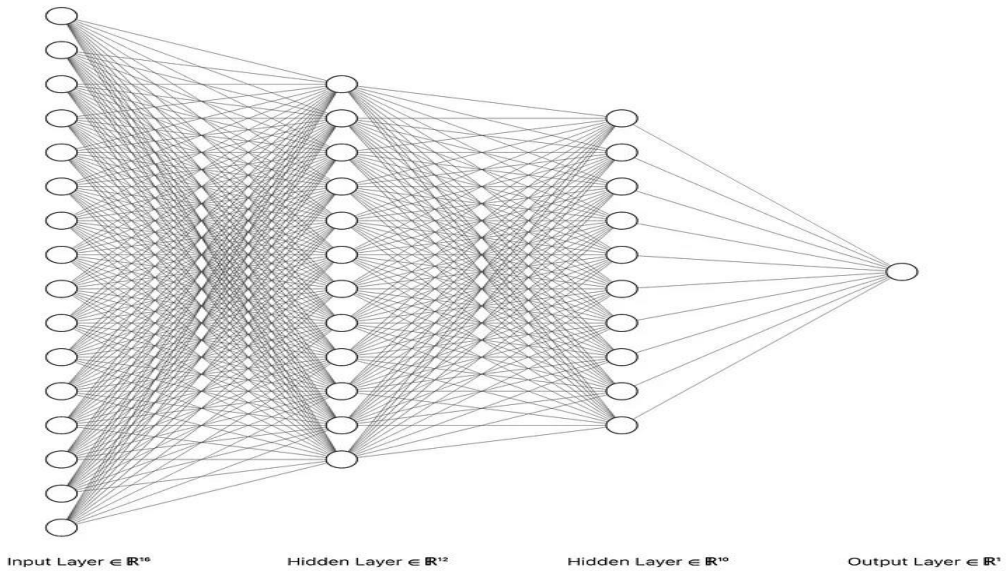


Figure 1.2 – Multilayered Perceptron

1.2 Recurrent Neural Networks

Recurrent Neural Networks are the neural network that allows previous outputs to be used as inputs and handle sequential data while having hidden states, While traditional deep neural networks assume the inputs and the outputs are independent of each other, unlike the MLPs which process data in a single pass. They are distinguished by their memory as they take information from prior inputs to influence the current input and output. RNNs share parameters across each layer of the network and have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network. It also uses the BackPropagation through time (BPTT)

algorithm to determine the gradients, which is slightly different from traditional backpropagation as it is specific to sequence data. They are usually used in Natural language processing.

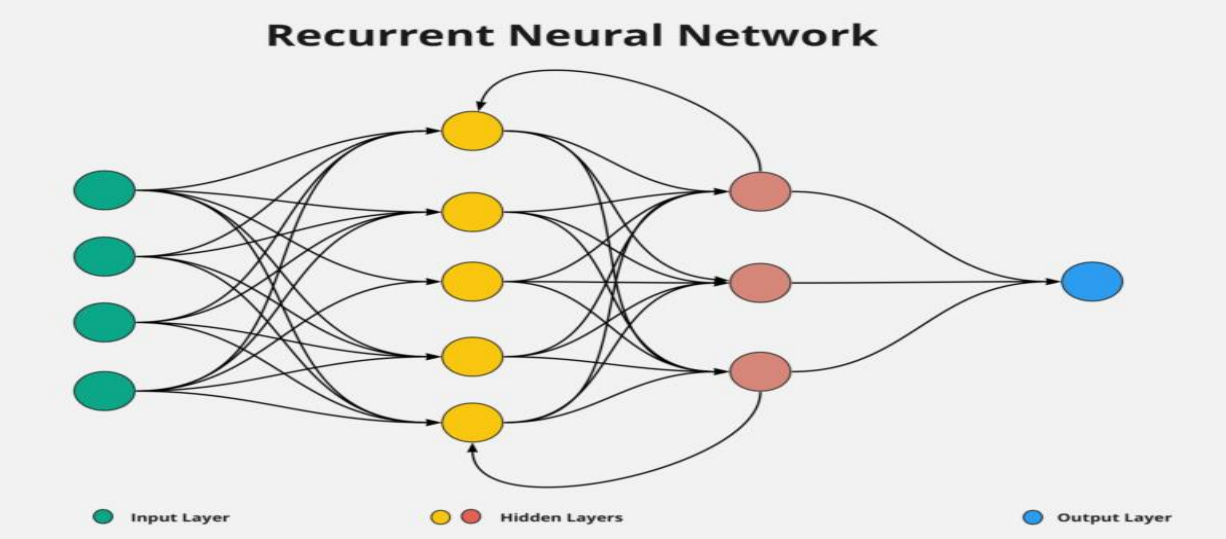


Figure 1.3 – Recurrent Neural Network

1.3 Convolutional Neural Networks

(CNNs) are specialized types of neural networks that are designed mainly for object recognition, Convolutional neural networks are distinguished from other neural networks by their superior performance including in image classification and detection. It is used widely for vision tasks to extract features from visual data. Convolutional neural networks are distinguished into convolutional layer, pooling layer and fully-connected layer. The Convolutional layer is the core of the CNNs where the majority of computation occurs and it requires a few components which are the input data, filter and a feature map. Another convolution layer can follow the initial convolution layer. when this happens the structure of the CNN can become hierarchical as the later layers. Ultimately the convolutional layer converts the image into numerical values. The pooling layer is known as downsampling and reducing the number of parameters in the input. Fully connected layers are fully connected to each node in the output layer which connects directly to a node in the previous layer.

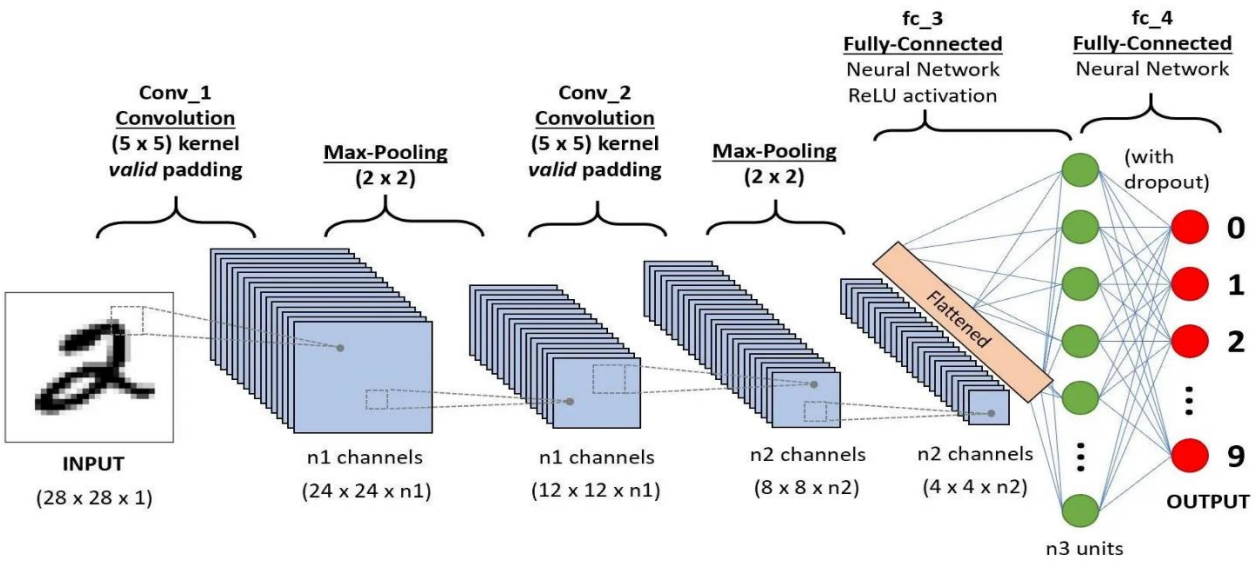


Figure 1.4 – A-CNN-sequence-to-classify-handwritten-digits

We will be talking about the convolution neural network in this context which was invented by researchers to improve image recognition technology. The model is trained to recognize and familiarize itself with patterns in visual objects and images to identify the shapes of certain objects or objects in an image. When training a convolutional model in the neural network a dataset of the image is required, once it is trained it can identify different objects and specific or identify the objects in images.

In Neural networks, they are dense and Sparse neural networks. A dense neural network is a type of neural network that uses multiple players of neurons to process data. Each neuron is given a layer which is deeply connected to the previous layer of neurons.

The difference between a sparse and dense neural network is based on the number of connections which each neuron has in the network. Sparse network, every neuron in the network is connected to a few neurons unlike in a dense network, in which each neuron is deeply connected to many other neurons around it which means the sparse has fewer connections than the dense network proving there is a going to be less performance and efficiency in carrying out a task especially ones required to be processed in a large amount of data. That's the main advantage of using a dense neural network because it can process more complex tasks due to its deep connections with each and every neuron.

In a neural network, the major components are the input layer, hidden layers, and the output layers. The process starts with the inputs which are received through the input layer, they are then processed and sent through the hidden layers and finally to the output layers. The hidden layers are important because they store information in the neurons until a sufficient amount of information is stored to be processed by the output layer which gives the trainer the prediction.

The main goal and objective of any deep neural network algorithm is to predict outputs. closer to the actual outputs. Optimization algorithms serve an important role in the propagation of the neural network while training. It speeds up the training process and finds the optimal parameters for a neural network. It helps to reduce the cost function, which is mostly based on the prediction error. When the Optimization algorithm executes, it solves every possibility iteratively until it reaches its peak satisfactory, which helps the neural network during the training process.

In training a neural network, one of the optimizing algorithms in the neural network used to minimize some function by iteratively moving in the direction of the steepest descent Is Gradient decent. It is one of the most basic and popular Optimization algorithms.

Table 1.1 – Optimization Algorithms

Optimization Algorithms	**Pros**	**Cons**	**References**
Adaptive Gradient	Works perfectly on data with sparse features	Genealiz and converges to sharp minima	2011
AdaMax	Capable of adjusting the learning rate based on data characteristics	Likely to overfit very fast	2015
Nadam	Uses decaying step size and hyperparameters to improve the performance	Generalized worse, converges to sharp minima	2016
EAdam	Smaller Stepsize	Computationally heavy	2020
Adaptative Delta	Keeping the learning rate optimally high	Accumulation of the squared gradients	2012
RMSProp	Works well on data with sparse features	Generalies worse, Converges to minima	2012

Continue of tabl.1.1

Optimization Algorithms	Pros	Cons	References
EVGO	Requires fewer parameters for tuning	Generalises worse and converges to sharp minima	2020
Adaptative Momentum	Converges faster	Generalises poorly	2015
LAMB	Fewer Parameters for tuning, faster computation	Generalises worse, converges to sharp minima	2020

There are kinds of gradient Optimization algorithms such as Vanilla gradient descent, Stochastic Gradient Descent (SGD) and Mini-batch SGD. According to research. The list of different types of Optimization algorithms in chronological order with the advantages and disadvantages depending on the data set and the machine learning models.

In recent years Deep neural Networks have succeeded in a wide range of application domains ranging from computer vision to machine translations to automatic speech recognition which stems from their ability to learn complex transformations by data example while achieving superior generalisation performance. Though they generalise well, deep neural networks learn more efficiently when they are highly over-parametrized (Zhang et al, 2016)More evidence proves the need for overparameterization to the geometry of the high-dimension lost landscapes (Dauphin et al., 2014; Choromanska et al., 2014; Goodfellow et al., 2014; Im et al., 2016; Wu et al., 2017; Liao & Poggio, 2017; Cooper, 2018; Novak et al., 2018).

Multiple techniques can compress large amounts of trained models, like weight precision reduction and pruning. These methods are highly effective in reducing the size of the network parameter with little degradation in accuracy they usually operate on pre-trained models or require the full overparameterized model to be stored and updated during a certain stage in training.

Deep neural networks are standard tools for solving computer vision problems. Deep neural networks are mostly used for image classification, and natural language

processing and rarely for audio recognition. Computer vision and image recognition have revolutionized and propelled advancement in fields such as healthcare, agriculture, and banking. There are open-source datasets for image classification such as CIFAR-100, ImageNet and MNIST online.

Deep learning is important in the development of artificial intelligence because it helps machines learn in a similar way that humans do. It also allows machines to learn from their experiences so that machines can automatically adapt and learn new pieces of information on their own without much human intervention. In a deep neural network, there are various Optimisers, most of which use stochastic descent algorithms for better accuracy.

1.4 Task definition

The aim of this research is to provide an in-depth understanding of the modern neural networks, its role in various organizations, and its impact on operational efficiency information systems. Here are some tasks in this process:

1. Create training set and validation set.
2. Annalise optimization algorithms of neural network's parameters
3. Propose an algorithm for optimizing the correspondence of the neural network model to the training data.
4. Define the performance criteria of the neural network in the training and validation set.
5. Implement the algorithm in software

2 OPTIMIZATION ALGORITHMS

Optimization in neural networks plays a very important role in the training of the training process. It is very crucial for the efficient performance of the neural network model and to give optimum performance in solving a task. As aforementioned one of the most common methods for Optimization methods is gradient descent. It is always repeatedly adjusting the network parameter to improve performance.

In training a neural network, it is required to set the parameters in a way that there are underlying models that help the model perform and achieve peak performance with the tasks.

While training models loss functions are required to be defined to be able to evaluate the network performance, A loss function is referred to as the error function which is very important in model training that quantifies the difference between the predicted outputs of a model and the actual targeted output. During training, a learning algorithm such as a backpropagation algorithm uses the gradient descent of the loss function to adjust these parameters and minimize loss, therefore improving the model's performance on the dataset. Recent studies have proposed many Optimization algorithms and each has its advantages and disadvantages. We will briefly talk about the current Optimisers used in Optimization.

2.1 Stochastic Gradient-Descent

In training neural network models different methods are applied to complete various tasks. The most common approach used in training models is gradient descent. The primary goal of gradient descent is to identify model parameters that provide the maximum accuracy on the training and test datasets. SGD has been successfully applied to large-scale and sparse machine-learning problems often encountered, by repeating this process over and over again, the network will learn the weights needed to produce the optimal output for a given output.

SGD has been around for a while and has been used commonly in Optimization and has several advantages such as; It is very easily implemented in the training of a model and it produces high-quality results, second it is very easy to implement while

using large datasets because it is efficient in computation, third In traditional gradient descent algorithms, SGD is less likely to get stuck because it updates the parameters using a few das point at a data making it more likely to find the global minimum. One of the main disadvantages of SGD is that it requires a lot large number of iterations to learn the optimal set of parameters, the performance of this method can degrade when presented with large complex datasets. There is a higher computation burden on the Optimization algorithm when training models when we don't have more data points. There are some situations where SGD can be slow in situations where the gradient is small, this is because of the update rule in the algorithms which only depends on the gradients at each iteration.

The most commonly used algorithm in the SGD

$$\theta = \theta - \eta \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}$$

Where;

θ(Theta): is the parameter we are trying to optimize

η(Eta): Is the learning rate. It determines the size of the jump

∇θJ(θ;x(i);y(i)): This is the gradient of a loss function calculated with the parameter θ

∇: tells the direction of the steep ascent of the loss function

J(θ;x(i);y(i)): Represents the cost of the model prediction where x(i) is the input data and y(i) is the corresponding target value.

This formula performs a single update for the SGD parameter using the information from a training example. The formula stated that we update the parameter vector by subtracting the scale version of the gradient in the negative direction.

And the most referred-to SGD algorithm ;

```
      for epoch in range(np_epochs):
          np. random.shuffle(data)  # Shuffle data for each epoch

          for example in data:
            params_grad = evaluate_gradient(loss_function, example, params)
            params = {key: value - learning_rate * params_grad[key] for key, value in
params.items()}
```

2.2 AdaGrad

AdaGrad is short for adaptative gradient, it signifies that the learning rates are adjusted or adapted over time based on the previous gradients.

The learning rate is adapted component-wise to the parameters by incorporating knowledge of past observations. It performs larger updates(High learning rates) for those parameters that are related to infrequent features and smaller updates (Low learning rates)for frequent ones. It performs smaller updates as a result, it is well-suited when dealing with sparse data.

AdaGrad deals with the aforementioned problems by independently adaption the learning rate for each weight component. If the gradient corresponding to a certain weight vector component is large, then the respective learning rate will be small. During weight update, instead of using a normal learning rate, AdaGrad scales it by dividing learning rate by the square root of the accumulated gradients. One of the benefits of using Adagrad is that the learning rates change with each training parameter and it does not need manual tuning for learning rate. As for the disadvantages; the learning rate is always decreasing which results in slow training and it is computationally expensive to calculate

The AdaGrad Formulation

$$\omega_t = \omega_{t-1} - \eta \frac{\partial L}{\partial_{\omega(t-1)}}$$

Where:

W(t): Value of w at current iteration,

W(t-1): Value of the value at the previous iteration

η(Eta): This is the learning rate, A small positive value that controls the magnitude of the update

∂L/∂_(ω(t-1)): This represents the partial derivative of a loss function with respect to the previous time step.

AdaGrad Algorithm:

```
        # Update parameters with AdaGrad
        for i, (param, grad) in enumerate(zip(params, gradients)):
          historical_grad_sum[i] += grad**2
                  adjusted_learning_rate     =     learning_rate     /     (eps     +
np.sqrt(historical_grad_sum[i]))
          param -= adjusted_learning_rate * grad

        return params
```

2.3 Adaptative Momentum

Adaptative momentum also known as ADAM is an algorithm developed by DR.Hinton at the University of Toronto. It was developed initially to be used for training deep neural network models in machine translations and speech recognition applications. The main components are Minibatch gradient descent and the learning rate schedules. Minibatch gradient descents is a method of calculating the learning individual rate of a model rather than the entire batch. This allows the algorithm to adapt quickly during training.

Recently, ADAM has been introduced as an implementation of the gradient-free stochastic optimization method by Kingma and BA. In this algorithm, both the step size and learning rate are scholastically controlled by the temperature parameter. So it can be considered the RMS-prop with momentum it combines learning rates with momentum.

ADAM Formulation

$$\Delta\omega_t = -\eta \frac{x_t}{\sqrt{y_t} + \epsilon} * gt$$

Where

gt is the gradient at time t

xt is the exponent average of the gradient with w

η is the learning rate

ADAM Algorithm

```
#Implementing Adam
def adam(params, grads, lr, b1=0.9, b2=0.999, eps=1e-8):
  m, v = [0] * len(params), [0] * len(params)
  for t, p, g in zip(count(), params, grads):
    m = b1 * m + (1 - b1) * g
    v = b2 * v + (1 - b2) * g * g
    m_hat = m / (1 - b1**t)
    v_hat = v / (1 - b2**t)
    p -= lr * m_hat / (np.sqrt(v_hat) + eps)
  return params

from itertools import count
```

2.4 RMSprop

RMSprop stands for Root Mean Square Propagation is a technique for reducing the noise in the neural network by smoothing out the errors as they are propagated through the network, it is an algorithm designed to address some of the issues encountered with the stochastic gradient descent. Some researchers believe that adding a single layer of neurons to a deep neural network can reduce the network accuracy by up to 10%. However, recent research shows that using RMSprop can help reduce this effect and improve the accuracy of deep neural networks.

Two key techniques to decrease error in deep learning models are to propagate error throughout the network and smooth out values before transferring them to the next layer through weight decay, and to regularise network bias using a loss function that penalises high values of the network weight. Deep network training can become laborious and complex as a result of this procedure, which is frequently carried out by adding a little learning rate to weight updates at each layer. RMSprop, on the other hand, is a more effective and efficient method since it doesn't require a regulariser and is simpler and easier to apply in deep learning models.

RMSProp was proposed by Geoffrey Hinton, it was used as an extension of gradient descent and AdaGrad uses the decaying average of partial gradients in the adaption of the steps for each parameter.

The benefits of using RMSprop are that it has a higher convergence by adaption the learning rate, It can converge to the optimal solution faster than SGD especially when dealing with noisy gradients. Although it has these benefits there are some limitations of it such as hyperparameter tuning which causes the decay rate and initial learning rate that need to be tuned for specific tasks.

$$V_{dw} = \beta V_{dw} + (1 - \beta) dw^2$$
$$W = W - \alpha \frac{dw}{\sqrt{Vdw}}$$

Where:

V_dw: The exponentially decaying moving average of the squared gradient parameter w

β: This is used as to control the decay rate of the moving average. The higher **β** places more weight on past squared gradients and the lower **β** on more recent updates.

dw: This is the current gradient of the loss function with respect to parameter w.

(1-**β**): This term scales the contribution of the current squared gradient to the moving average.

RMSProp Algorithm

```
      # RMSProp algorithm
      def RMSProp(objective_function, derivative_function, values_range, n_iterations,
step_size, Beta):
          # list of all solution points
          all_solutions_list = list()
          # initial point generation within the range
           current_solution_point = values_range[:, 0] + rand(len(values_range)) *
(values_range[:, 1] - values_range[:, 0])
          # squared gradients average
          squared_gradient_avg = [0.0 for _ in range(values_range.shape[0])]
```

3 PROPOSED METHODOLOGY

The primary purpose of this work is to develop an optimization technique to improve the accuracy of neural networks. In addition, the optimization techniques reduce the time and the complexity of the neural network. In this section, we will be talking about briefly the proposed methodology.

Training of neural networks is usually cumbersome and takes a lot of time sometimes it takes or even weeks. Because of this, there is a need for improved and efficient training speed to carry out such applications especially when considering the parallelisations of CNN.

Why CNN?, We moved to CNN because of the weight sharing in CNN while performing convolution operations on input raw images. Which tremendously cut down parameters in the whole network making network computation less incentive. The other thing is dimensionally reduction because of the introduction of poling layers in the network The proposed optimization method is a modified version of Adam optimiser in which we remove the additional hyperparameter.

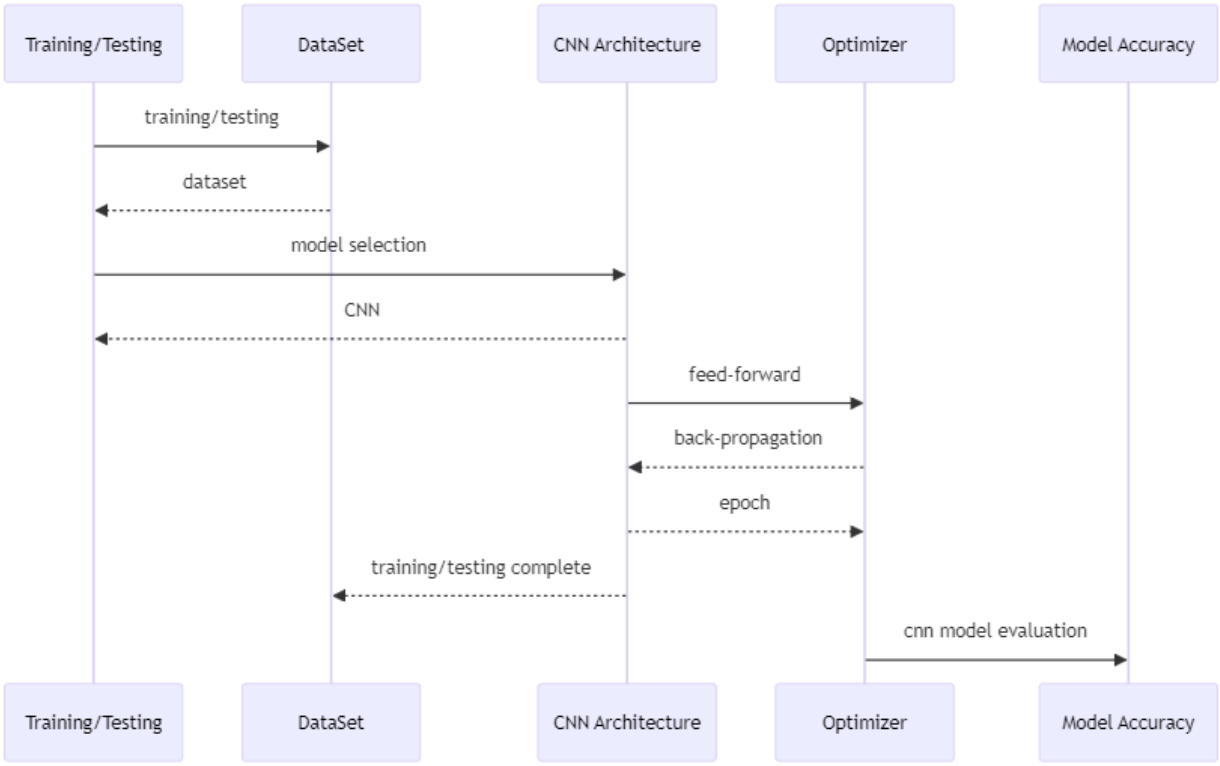


Figure 3.1 – Proposed sequence diagram for proposed Optimization technique.

3.1 Pseudo code for Adam optimiser method

Algorithm: Adaptive Moment Estimation (Adam)

Require: α: Stepsize

Require: β1, β2 ∈ [0, 1): Exponential decay rates for the moment estimates

Require: f(θ): Stochastic objective function with parameters θ

Require: θ0: Initial parameter vector

1. m0 ← 0 (Initialize 1st moment vector)
2. v0 ← 0 (Initialize 2nd moment vector)
3. t ← 0 (Initialize timestep)

   while θt not converged do

   1.t ← t + 1

1. gt ← ∇θft(θt−1) (Get gradients w.r.t. stochastic objective at timestep t)
2. mt ← β1 · mt−1 + (1 − β1) · gt (Update biased first moment estimate)
3. vt ← β2 · vt−1 + (1 − β2) · g²t (Update biased second raw moment estimate)
4. $m\hat{}tm\hat{}t$ ← mt / (1 − β1^t) (Compute bias-corrected first-moment estimate)
5. $v\hat{}tv\hat{}t$ ← vt / (1 − β2^t) (Compute bias-corrected second raw moment estimate)
6. θt ← θt−1 − α · $m\hat{}tm\hat{}t$ / ($\sqrt{v\hat{}tv\hat{}t}$ + ε) (Update parameters)

   end while

   return θt (Resulting parameters)

3.2 Experimental Environment And Results

This gives more insight into the environment we conducted and the implementation of the realization techniques during the training of the model.

We will be working on detecting handwritten digits using the CNN model that is provided using MNIST(Modified National Institute of Standards and Technology Database) data to make a simple model made up of handwritten digits between 0 and 9 [1,2]. Where each image has a pixel of 28 x 28, the dataset is split into two portions for training and testing sets. The training set contains 60,00 images of handwritten digits used to train models and the Testing set contains 10,00 images of handwritten digits to train

models on unseen data that is being to have a very low run time with the Adam algorithm. We are using the MNIST data set as it is more simple and easy to use.

Table 3.1 – Experimental Environment

Component	Description
System Model	Intel core i5 vPro 8th Gen
Operating System	Windows 11
GPU	Intel ® UHD Graphic 620
RAM	16GB
ROM	500 GB SSD
IDE	Visual Studio Code
Environment	Jupyter
Programming Language	Python 3.8
Libraries	Numpy, Matplotlib, Sklearn, Plotly, TensorFlow, VisualKeras

In the training of the model in the Jupyter environment, we applied the Adam Optimiser algorithm in Cell [13] and [21] for more efficiency in the training of the model.

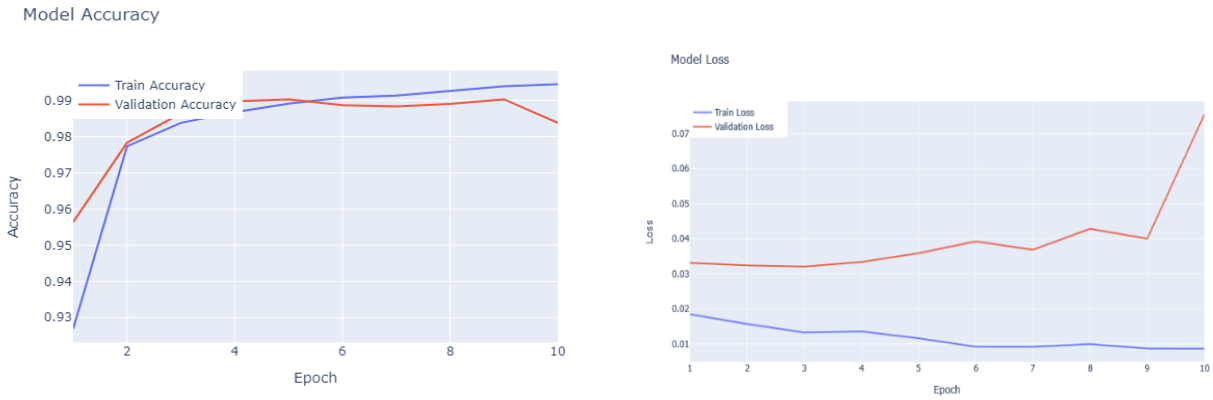


Figure 3.2 – Model Accuracy

We made sure to apply it also in the extra layer of the model which we added for a more accurate result. With Tensorflow the Adam Optimization algorithm is already included in the library which makes it easier to implement especially for beginners. A lower learning rate for the convolution layers is often used in practice when applying. We show the effectiveness of Adam in deep CNNs During the training process, the proposed

achieved an accuracy of 98.98% whereas during the testing process, it achieved an accuracy of 97.95% Our CNN architecture has two alternating stages that are followed by a full connected layer of 1000 rectified linear hidden units (ReLU’s).

CONCLUSION

In the course of the experiment. Training loss is used to measure how a neural network model fits the training data. The error of the model is assessed with the training set. It is evaluated by taking the sum of errors in the training sets.

Validation loss is a metric used to measure the performance of the neural network on the validation set.

If the validation loss is greater than the training loss, it shows that the model is not fitted, which means the model is unable to train and give accurate data. The validation loss is greater than the training loss and changes frequently, it is known as overfitting. The program below gives more insight into the step-by-step process for training the model and the implementation of the optimization algorithm.

REFERENCES

1. Wang, S.Y.; Wang, O.; Zhang, R.; Owens, A.; Efros, A.A. CNN-generated images are surprisingly easy to spot... for now. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 8695–8704.
2. Sun, R.Y. Optimization for deep learning: An overview. J. Oper. Res. Soc. China 2020, 8, 249–294. [CrossRef]
3. Weinan, E.; Ma, C.; Wu, L. A comparative analysis of optimization and generalization properties of two-layer neural network and random feature models under gradient descent dynamics. Sci. China Math 2019.
4. Duchi, J.; Hazan, E.; Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 2011, 12, 2121–2159.
5. Zeiler, M.D. Adadelta: An adaptive learning rate method. arXiv 2012, arXiv:1212.5701.
6. Hinton, G. Neural networks for machine learning. Coursera video lectures. 2012. Available online: https://archive.org/details/academictorrents_743c16a18756557a67478a7570baf24a59f9cda6 (accessed on 1 December 2022).
7. Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G. On the importance of initialization and momentum in deep learning. In Proceedings of the International Conference on Machine Learning, PMLR, Atlanta, GA, USA, 17–19 June 2013; pp. 1139–1147.
8. Zhang, Z. Improved Adam optimizer for deep neural networks. In Proceedings of the 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 4–6 June 2018; IEEE: New York, NY, USA, 2018; pp. 1–2.
9. Dozat, T. Incorporating Nesterov momentum into Adam. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.

10. Wang, Y.; Zhou, P.; Zhong, W. An optimization strategy based on the hybrid algorithm of Adam and SGD. In Proceedings of the MATEC Web of Conferences, 2018; EDP Sciences: Paris, France, 2018; Volume 232, p. 03007.
11. Newton, D.; Yousefi, F.; Pasupathy, R. Stochastic gradient descent: Recent trends. In Recent Advances in Optimization and Modeling of Contemporary Problems; INFORMS: Catonsville, MD, USA, 2018; pp. 193–220.
12. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. arXiv 2014, arXiv:1412.6980.
13. Liu, Z.; Xu, X.; Fang, D.; Gan, D. Optimizing CNN Using Adaptive Moment Estimation for Image Recognition. In Proceedings of the 2023 IEEE International Conference on Control, Electronics and Computer Technology (ICCECT), Jilin, China, 2023; pp. 454-463. doi: 10.1109/ICCECT57938.2023.10140526.
14. Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
15. DataCamp. Available online: https://www.datacamp.com/ (accessed on 1 May 2024).
16. IBM. Neural Networks. Available online: https://www.ibm.com/topics/neural-networks (accessed on 1 May 2024).
17. GeeksforGeeks. ML | Stochastic Gradient Descent (SGD). Available online: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ (accessed on 1 May 2024).
18. Databricks. Adagrad. Available online: https://www.databricks.com/glossary/adagrad (accessed on 1 May 2024).
19. DeepAI. Machine Learning. Available online: https://deepai.org/machine-learning (accessed on 1 May 2024).
20. Scaler. RMSprop. Available online: https://www.scaler.com/topics/deep-learning/rmsprop/ (accessed on 1 May 2024).
21. Zhang, L.C. Tutorial on Gradient Descent. Available online: https://www.cs.toronto.edu/~lczhang/321/tut/tut06.html (accessed on 1 May 2024).

22. Mirzaei, A. Hand-written digit detection CNN. Available online: https://www.kaggle.com/code/amirhosseinmirzaie/hand-written-digit-detection-cnn-tensorflow/notebook (accessed on 1 May 2024).
23. ChatGPT, OpenAI's language model. 2024.

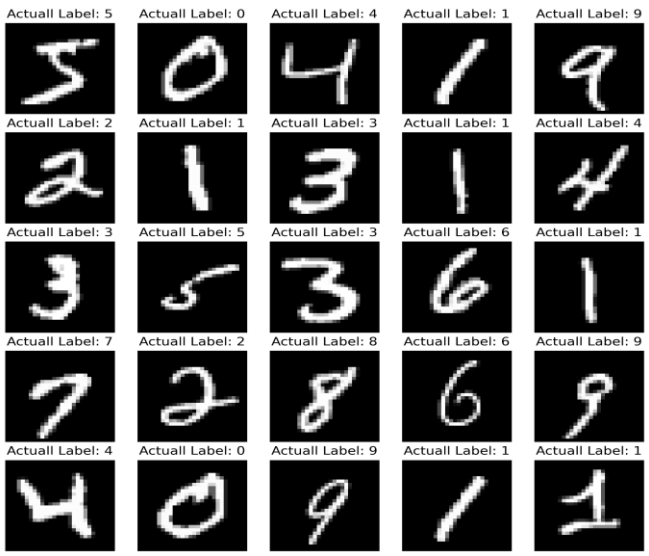
APPENDIX

Importing all necessary Library for our Model

In [6]:
```
import numpy as np # Numerical data and algebra
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns # Also for statistical data visualisation and works with matpotlib
import plotly.graph_objects as go
import tensorflow as tf # Tensorflow offers the impementation of the Adam optimzing algorithm
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt # For creating visual plots to represent data
import random
import visualkeras # For visualising the tensorflow(Keras) architecture of the neura network
```

In [7]:
```
# Importing the mnist Dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

In [8]:
```
# Data and Labels
plt.figure(figsize=(10,10), dpi=270)
for i in range (25):
    plt.subplot(5,5,i+1)
    plt.imshow(train_images[i].reshape(28,28), cmap='gray')
    plt.title(f"Actuall Label: {train_labels[i]}")
    plt.axis('off')

plt.show()
```

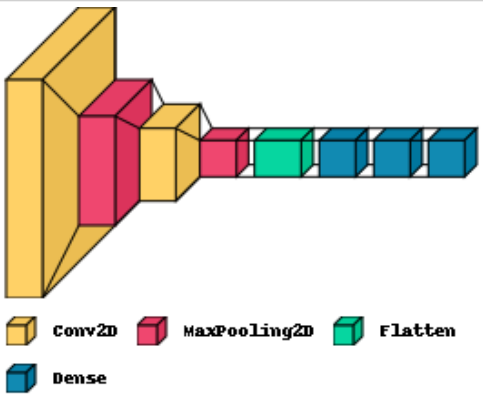


In [9]:
```
# Defining neccesarry functions needed for the model evaluation

def plot_training_history_with_plotly(history):
    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=list(range(1, len(history.history['accuracy']) + 1)),
        y=history.history['accuracy'],
        mode='lines',
        name='Train Accuracy'
    ))

    fig.add_trace(go.Scatter(
        x=list(range(1, len(history.history['val_accuracy']) + 1)),
        y=history.history['val_accuracy'],
        mode='lines',
        name='Validation Accuracy'
    ))

    fig.update_layout(
        title='Model Accuracy',
        xaxis_title='Epoch',
        yaxis_title='Accuracy',
        legend=dict(x=0, y=1),
    )

    fig.show()

    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=list(range(1, len(history.history['loss']) + 1)),
        y=history.history['loss'],
        mode='lines',
        name='Train Loss'
    ))

    fig.add_trace(go.Scatter(
        x=list(range(1, len(history.history['val_loss']) + 1)),
        y=history.history['val_loss'],
        mode='lines',
        name='Validation Loss'
    ))

    fig.update_layout(
        title='Model Loss',
        xaxis_title='Epoch',
        yaxis_title='Loss',
        legend=dict(x=0, y=1),
    )

    fig.show()
```

In [10]:
```
# Split into traingin data and testing data
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

In [11]:
```
# Modelling
lanet5_model = models.Sequential([
    layers.Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(16, (5, 5), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(120, activation='relu'),
    layers.Dense(84, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

In [12]:
```
visualkeras.layered_view(lanet5_model, scale_xy=5, legend=True)
```



```
In [13]: lanet5_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
         lanet5_model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 24, 24, 6)	156
max_pooling2d_10 (MaxPooling2D)	(None, 12, 12, 6)	0
conv2d_14 (Conv2D)	(None, 8, 8, 16)	2416
max_pooling2d_11 (MaxPooling2D)	(None, 4, 4, 16)	0
flatten_5 (Flatten)	(None, 256)	0
dense_12 (Dense)	(None, 120)	30840
dense_13 (Dense)	(None, 84)	10164
dense_14 (Dense)	(None, 10)	850

Total params: 44,426
Trainable params: 44,426
Non-trainable params: 0

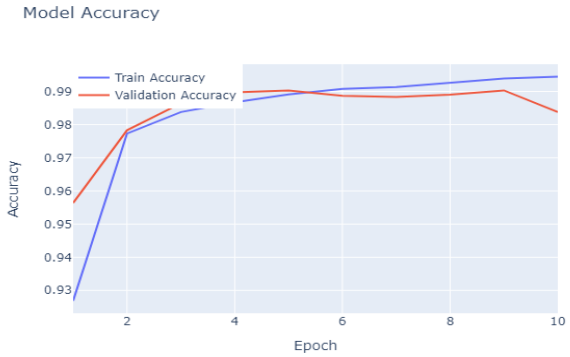
```
In [15]: lanet5_history = lanet5_model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_data=
                                           (test_images, test_labels))
```

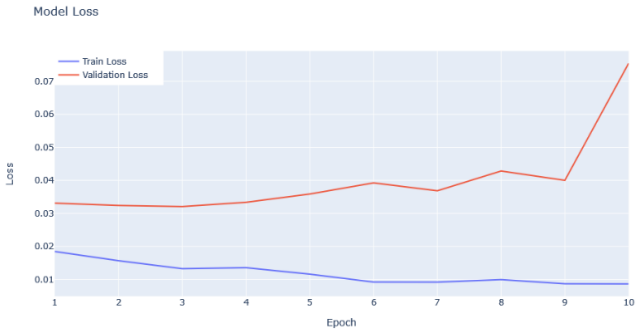
```
Epoch 1/10
938/938 [==============================] - 24s 23ms/step - loss: 0.2441 - accuracy: 0.9269 - val_loss: 0.1336 - val_accuracy: 0.9564
Epoch 2/10
938/938 [==============================] - 21s 22ms/step - loss: 0.0740 - accuracy: 0.9773 - val_loss: 0.0667 - val_accuracy: 0.9783
Epoch 3/10
938/938 [==============================] - 23s 24ms/step - loss: 0.0539 - accuracy: 0.9838 - val_loss: 0.0415 - val_accuracy: 0.9864
Epoch 4/10
938/938 [==============================] - 22s 24ms/step - loss: 0.0430 - accuracy: 0.9867 - val_loss: 0.0335 - val_accuracy: 0.9897
Epoch 5/10
938/938 [==============================] - 20s 22ms/step - loss: 0.0344 - accuracy: 0.9891 - val_loss: 0.0334 - val_accuracy: 0.9903
Epoch 6/10
938/938 [==============================] - 26s 28ms/step - loss: 0.0290 - accuracy: 0.9907 - val_loss: 0.0387 - val_accuracy: 0.9887
Epoch 7/10
938/938 [==============================] - 23s 24ms/step - loss: 0.0261 - accuracy: 0.9913 - val_loss: 0.0353 - val_accuracy: 0.9883
Epoch 8/10
938/938 [==============================] - 21s 23ms/step - loss: 0.0226 - accuracy: 0.9926 - val_loss: 0.0343 - val_accuracy: 0.9890
Epoch 9/10
938/938 [==============================] - 21s 22ms/step - loss: 0.0186 - accuracy: 0.9939 - val_loss: 0.0337 - val_accuracy: 0.9903
Epoch 10/10
938/938 [==============================] - 25s 27ms/step - loss: 0.0166 - accuracy: 0.9944 - val_loss: 0.0584 - val_accuracy: 0.9838
```

```
In [16]: # Test loss
         test_loss, test_acc = lanet5_model.evaluate(test_images, test_labels)
         print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0584 - accuracy: 0.9838
Test accuracy: 0.9837999939918518
```

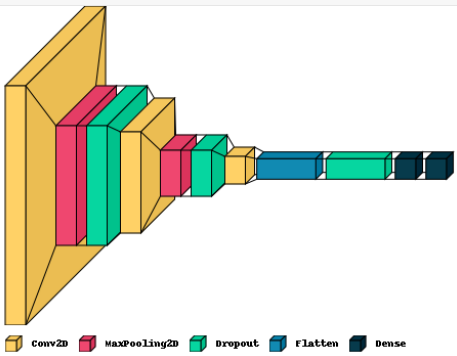
```
In [17]: plot_training_history_with_plotly(lanet5_history)
```





In [19]:
```
# Adding a second layer
model_2 = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
layers.MaxPooling2D((2, 2)),
layers.Dropout(0.25),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Dropout(0.25),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.Flatten(),
layers.Dropout(0.5),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')
])
```

In [20]:
```
visualkeras.layered_view(model_2, scale_xy=9, legend=True)
```



In [21]:
```
# Optimizing with the Adam optimizer provided by the tensorflow library
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model_2.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_12 (MaxPooling2D)	(None, 13, 13, 32)	0
dropout_7 (Dropout)	(None, 13, 13, 32)	0
conv2d_16 (Conv2D)	(None, 11, 11, 64)	18496

max_pooling2d_13 (MaxPoolin g2D) (None, 5, 5, 64) 0

dropout_8 (Dropout) (None, 5, 5, 64) 0

conv2d_17 (Conv2D) (None, 3, 3, 64) 36928

flatten_6 (Flatten) (None, 576) 0

dropout_9 (Dropout) (None, 576) 0

dense_15 (Dense) (None, 64) 36928

dense_16 (Dense) (None, 10) 650

===
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

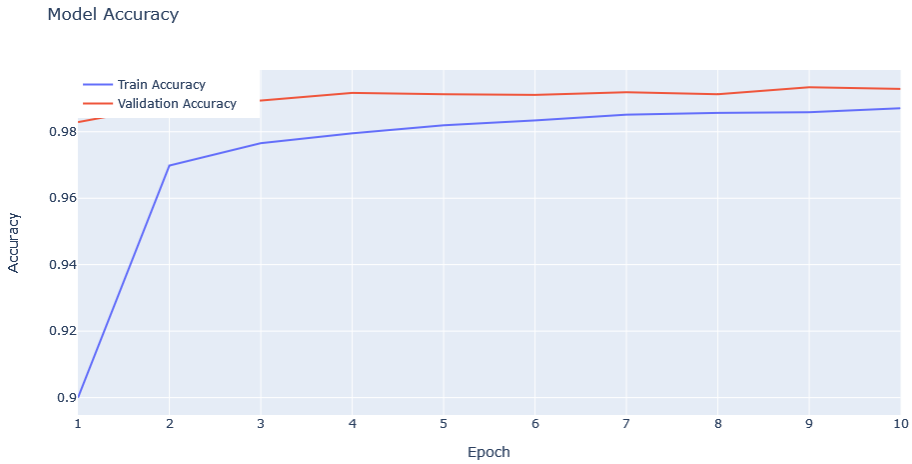
```
In [22]: model_2_history = model_2.fit(train_images, train_labels, epochs=10, batch_size=64, validation_data=(test_images, test
```

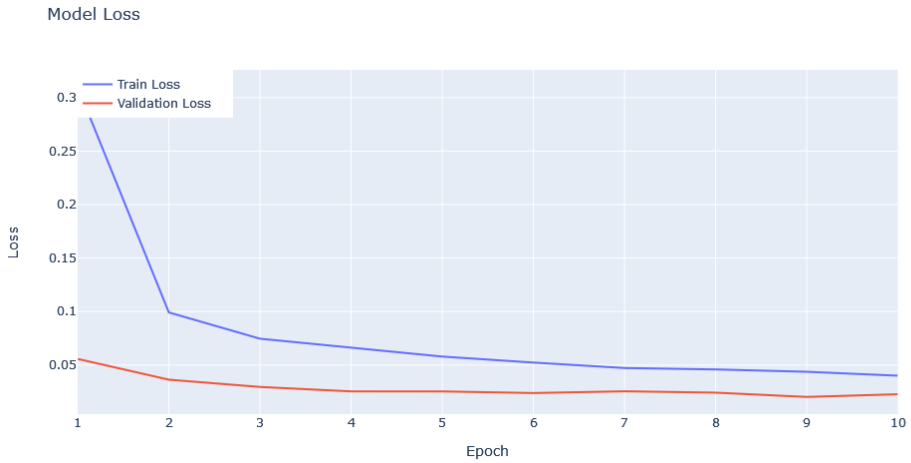
```
Epoch 1/10
938/938 [==============================] - 49s 48ms/step - loss: 0.3098 - accuracy: 0.9000 - val_loss: 0.0558 - val_accuracy: 0.9829
Epoch 2/10
938/938 [==============================] - 46s 49ms/step - loss: 0.0993 - accuracy: 0.9698 - val_loss: 0.0364 - val_accuracy: 0.9880
Epoch 3/10
938/938 [==============================] - 47s 50ms/step - loss: 0.0747 - accuracy: 0.9766 - val_loss: 0.0296 - val_accuracy: 0.9894
Epoch 4/10
938/938 [==============================] - 46s 49ms/step - loss: 0.0662 - accuracy: 0.9795 - val_loss: 0.0255 - val_accuracy: 0.9917
Epoch 5/10
938/938 [==============================] - 46s 49ms/step - loss: 0.0580 - accuracy: 0.9819 - val_loss: 0.0254 - val_accuracy: 0.9913
Epoch 6/10
938/938 [==============================] - 50s 53ms/step - loss: 0.0524 - accuracy: 0.9834 - val_loss: 0.0239 - val_accuracy: 0.9911
Epoch 7/10
938/938 [==============================] - 46s 49ms/step - loss: 0.0473 - accuracy: 0.9851 - val_loss: 0.0256 - val_accuracy: 0.9919
Epoch 8/10
938/938 [==============================] - 45s 48ms/step - loss: 0.0460 - accuracy: 0.9857 - val_loss: 0.0243 - val_accuracy: 0.9913
Epoch 9/10
938/938 [==============================] - 48s 51ms/step - loss: 0.0438 - accuracy: 0.9859 - val_loss: 0.0203 - val_accuracy: 0.9934
Epoch 10/10
938/938 [==============================] - 46s 49ms/step - loss: 0.0401 - accuracy: 0.9871 - val_loss: 0.0228 - val_accuracy: 0.9929
```

```
In [23]: test_loss, test_acc = model_2.evaluate(test_images, test_labels)
         print(f'Test accuracy: {test_acc}')
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.0228 - accuracy: 0.9929
Test accuracy: 0.992900013923645
```

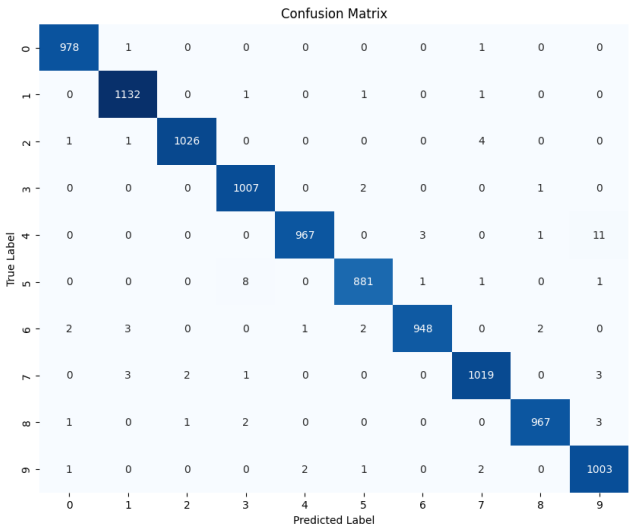
```
In [24]: plot_training_history_with_plotly(model_2_history)
```






```
In [21]: predictions = model_2.predict(test_images)

313/313 [==============================] - 2s 7ms/step
```

```
In [22]: true_labels = np.argmax(test_labels, axis=1)
predicted_labels = np.argmax(predictions, axis=1)

conf_matrix = confusion_matrix(true_labels, predicted_labels)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.arange(10), yticklabels=np.arange(10))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

print('\nClassification Report:\n', classification_report(true_labels, predicted_labels))
```



Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	980
1	0.99	1.00	1.00	1135
2	1.00	0.99	1.00	1032
3	0.99	1.00	0.99	1010
4	1.00	0.98	0.99	982
5	0.99	0.99	0.99	892
6	1.00	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	1.00	0.99	0.99	974
9	0.98	0.99	0.99	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

In [27]:
```
plt.figure(figsize=(10, 10))
plt.subplots_adjust(hspace=0.5)

for i in range(25):
    random_index = random.randint(0, test_images.shape[0] - 1)

    # Display the test image
    plt.subplot(5, 5, i + 1)
    plt.imshow(test_images[random_index].reshape(28, 28), cmap='gray')
    plt.title(f"Actual: {np.argmax(test_labels[random_index])}\nPredicted: {predictions[random_index].argmax()}")
    plt.axis('off')

plt.show()
```