Design of a Neural Network Architecture for Traffic Light Detection in Autonomous Vehicles

A Thesis presented to the

Department of

Computer Science and Engineering

African University of Science and Technology

Abuja, Nigeria.

In partial fulfillment of the Requirements for the Degree of

Masters of Science in Computer science

By

Akubo Raphael Ede

June 2019.

African University of Science and Technology [AUST]

Knowledge is Freedom

APPROVAL BY

Supervisor

Surname: BEN ABDALLAH

First name: ABDERAZEK

Signature

The Head of Department

Surname: DAVID

First name: Amos

Signature:

CERTIFICATION

This is to certify that the thesis titled “Design of a Neural Network Architecture for Traffic light detection in Autonomous Vehicles” submitted to the Department of Computer Science and Engineering, African University of science and technology Abuja, Nigeria, for the award of Master’s Degree, is a record of original research carried out by Akubo Raphael Ede in the department of Computer Science and Engineering.

SIGNATURE PAGE

DESIGN OF A NEURAL NETWORK ARCHITECTURE FOR TRAFFIC LIGHT DETECTION IN AUTONOMOUS VEHICLES

By

Akubo Raphael Ede

A THESIS APPROVED BY THE DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RECOMMENDED:

Supervisor, Prof Ben Abdallah

Head of Department, Prof David Amos

APPROVED BY:

Chief Academic Officer

Date

ABSTRACT

Over the ongoing years, innovatively propelled nations have kept on joining the quest for growing completely self-sufficient driven vehicles. This Autonomous vehicles intend to address issues of driver profitability and effectiveness. Dependable traffic light discovery is a vital segment for self-sufficient driving. Recognizing the traffic lights amidst everything is a standout amongst the most significant errand. The focus of this research is to develop and find the optimal parameters for an efficient Neural Network Architecture to aid a hardware engineer to implement on a hardware for the autonomous vehicle. This is done by designing an Artificial Neural Network (ANN) that would be capable of detecting and correctly classifying any traffic light within the city of Abuja, Nigeria. This study first attempts to develop a reliable traffic sign detector by constructing MLP, training using BP, and tuning various Convolutional Neural Networks (CNN). Images for training are obtained from Abuja city metropolis

Keywords: Backpropagation (BP), neural networks, CNN, MLP, ANN

DEDICATION

This research work is dedicated to the only one and Amazing God who saw me through this phase of my life without so much stress.

ACKNOWLEDGMENTS

This thesis would not have been possible without the invaluable guidance and vision of Professor Ben Abdallah. The guidance and mentorship he offered were second to none.

I would like to appreciate the support of my family and friends. Mr. and Mrs. Akubo, but for you, I may not be at AUST, writing this thesis. Thank you for always being there for me while growing up, and for teaching me. The end is now here. And equally so to my siblings; Simon, Mercy, Ladi, Esther, Faith, for their continued support. Demi, for being my biggest cheerleader. To Yerima, who have assisted in so many appreciable ways.

TABLE OF CONTENTS

Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ReLU	Rectified Linear Unit
ANN	Artificial Neural Network
MLP	Multi-Layer Perceptron
Pred.	Prediction
CNN	Convolutional Neural Networks (CNN)
ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
CV	Computer Vision
DNN	Deep Neural Network

CHAPTER ONE

INTRODUCTION

1.1 Research Background

Computer vision takes root in signal processing; wherein the effect of a system on a signal is studied, and frameworks for exploring this effect are examined. The black box is an excellent illustration of the basic premise of signal processing. In general, a signal is input into the black box, propagates through the unknown system therein, and another signal is output. The usual question we tend to ask is, “what is in the black box?” Typically, we may look at the input and output in terms of specific characteristics called metrics to infer some quality of the process undergone inside the black box. A research engineer may even explore different metrics that seem intuitive based on other information about the process and describe the unknown system in a novel way. The issue with the discovery attitude is that no measurement or set of measurements can show with sureness the substance of the black box in light of the fact that numerous interesting capacities exist with a similar arrangement. While overseeing multifaceted nature, utilizing presumptions and approximations may do the trick for some different fields of building. The information in Machine vision is excessively entangled and voluminous for this methodology.

Conventional traffic light detection methods often suffer from false positives in an urban environment because of complex backgrounds. To overcome such limitation, Deep Neural Network is emphasized, which is fast, but weak to false positives (Lee & Park, 2017). To realize autonomous vehicles, image recognition with high accuracy and high speed is necessary for the vehicle environment. ANNs are recently used in many machine

learning applications, from speech recognition and natural language processing to computer vision, and image recognition.

Conventional traffic light serves as the input to the black box, which by design, using the neural network will produce the desired output.

1.2 Artificial Neural Networks

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it, thus producing the desired output, which is very useful for decision making in autonomous vehicle systems. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. These systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules (Marcel van Gerven *et al*., 2017).

1.3 Neural Network Architectures

There are many neural network architectures; their different layers of neurons regularly organize them. These layers comprise of input, hidden, and output layers. Two metrics are frequently used to measure the neural network size, the number of neurons and the number of parameters. It is essential to mention that the network size plays an integral part in designing a neural network.

1.4 Problem Statement

Efficiency, accuracy, and quick decision making is a common challenge in the field of Autonomous Vehicles. Rapid processing of data helps in speeding up any operation to be performed on such data. Many frameworks for traffic light detection using machine learning have been proposed in the past but were rather time-in-efficient in either the reduction of the dimension or in terms of the efficiency of the machine learning algorithms used. The need for better approaches that can improve the computational problems associated with image processing or classification is in high demand and cannot be overemphasized.

This thesis aims to explain the design of a Neural Network architecture that will be used for image classification and in this case, a traffic light detection in autonomous vehicles. It describes the theory behind the neural network and Autonomous Vehicles, and traffic light dataset as its only input that can be designed to test and evaluate the algorithm's capabilities. The thesis will show that the Artificial Neural Network can, with an image resolution of 64 × 64 and a training set with 55 images, make decisions with a 0.54 confidence level.

1.5 Research Aim and Objectives:

Reliable traffic light detection is a crucial component for autonomous driving. One of the main tasks that such a vehicle must perform well is the task of following the rules of the road. Identifying the traffic lights amid everything is one of the most critical tasks.

The main objectives of the research are:

i. Design a neural network for image classification.

ii. Develop and find the optimal parameters for an efficient Neural Network Architecture to aid a hardware engineer to implement on hardware for the autonomous vehicle to improve driver productivity, enhance transportation efficiency, and increase safety.

1.6 Limitation of the Study

This research study is limited to the design of neural networks and classification of traffic light images for autonomous vehicles only. The study highlights the significance of neural networks in autonomous vehicle decision making.

Chapter Two

Literature Review

2.1 Artificial Neural Network

An ANN is a basic model of an animal brain. In a brain, a neuron is a cell that processes chemical or electrical signals. The neuron is connected to other neurons and creates a network, which is a human brain contains tens of billions of connected cells (Bruce & Otter, 2016)

The neuron in a brain has input routes called dendrites, a cell body and output routs called axons. When the electrochemical signal is transmitted through the dendrites, the neuron is activated. The cell body then determines the weight of the signal and if a threshold is passed, the neuron “fires” through the axon (Bell, 2014)

Artificial neural networks are motivated by Biological neural frameworks. Signal transmission in biological neurons through synapses is an unpredictable chemical process in which explicit transmitter substances are discharged from the sending side of the neural connection. The impact is to raise or lower the electrical potential inside the body of the getting cell. On the off chance that this potential achieves an edge, the neuron fires. It is this characteristic of the biological neurons that the artificial neuron model proposed by McCulloch Pitts endeavors to duplicate. The following neuron model shown in figure (2.1) is widely used in artificial neural networks with some variations.

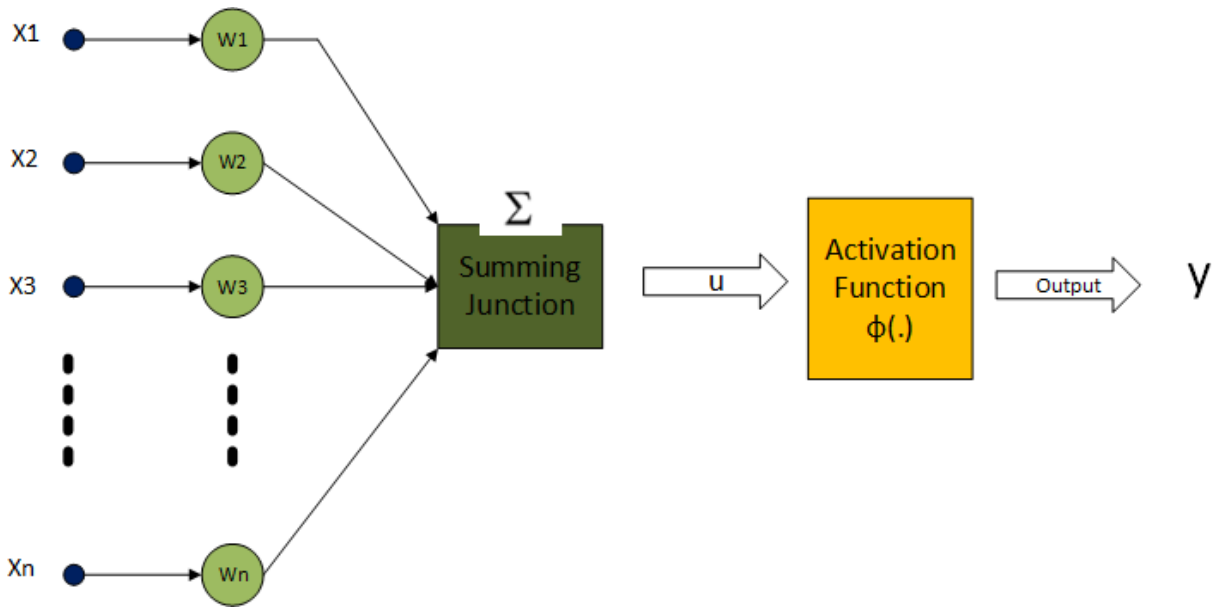

Figure 2. 1*: The Artificial Neural Network Architecture*

The network consists of multiple layers of feature-detecting neurons. Where each layer has many neurons that respond to different combinations of input signal from the previous layers. As shown in Figure (2.2), the layers are developed with the goal that the primary layer distinguishes a lot of crude patterns in the information, the second layer recognizes pattern of patterns, and the third layer identifies patterns of those patterns, etc.

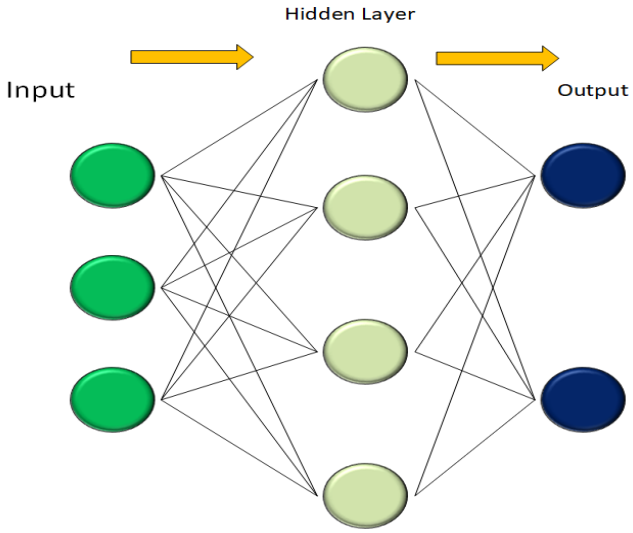

Figure 2. 2*: Artificial Neural Network*

2.1.1 Perceptron

In an ANN, the neurons are named perceptron and the axon and dendrites are called links. Looking at the figure below, the link from perceptron j to perceptron i transmits the output of j to i. This output has the notation a_j. The link also has a corresponding numeric weight, $W_{j,\,i,}$ which determines the strength and sign of the connection. (Bruce & Otter, 2016). The perceptron function is two-folded. Firstly, it computes the weighted sums of its inputs:

$$Z_i = \sum_{j=0}^{n} W_{j,i} a_j$$

Secondly, it applies an activation function, g, to compute the output from the perceptron (Russel and Norvig, 1995). The output is a numeric value in the range from zero to one.

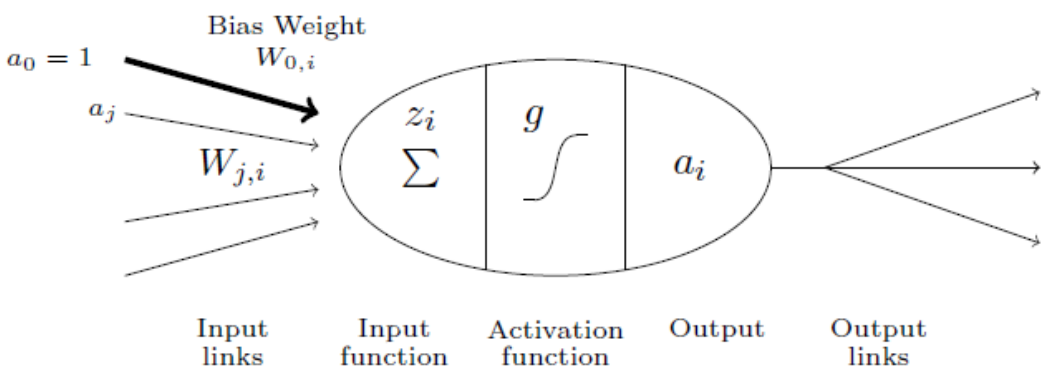

Figure 2. 3: the Perceptron

2.1.2 The Neuron Model (Single-Input Neuron)

A single-input neuron is shown in Figure 2.4. The scalar input ‘’*p”* is multiplied by the scalar weight ‘’*w”* to form ‘’*wp”*, one of the terms that are sent to the summer. The other

input, ‘’1’’, is multiplied by ‘’a’’ bias ‘’*b”* and then passed to the summer. The summer output ‘’*n”*, often referred to as the net input, goes into an activation function, which produces the scalar neuron output *a*.

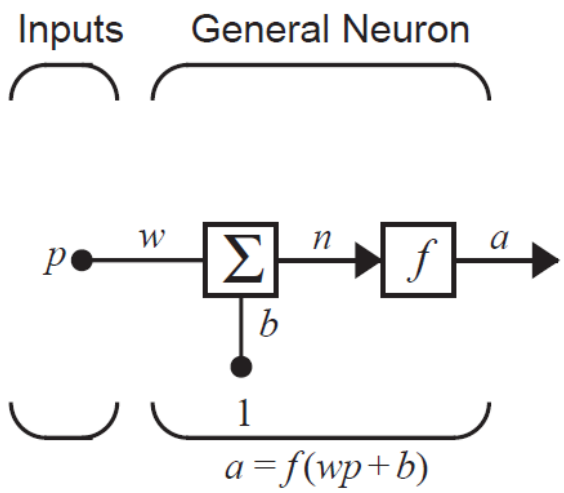

Figure 2. 4: Single input Neuron

The neuron output is calculated as

$$a = f(wp + b).$$

If, for instance, w = 2, p = 4 and b = -1.5, then

$$A = f(2(4) – 1.5) = f(6.5)$$

The actual output depends on the particular transfer function that is chosen.

We will discuss transfer functions in the next section.

The bias is much like weight, except that it has a constant input of 1. W and b are both adjustable scalar parameters of the neuron. However, if you do not want to have a bias in a particular neuron, it can be omitted

2.1.3 Activation function

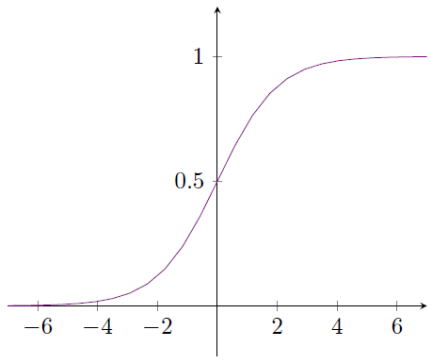

Figure 2. 5: The sigmoid function 1/1 + E $^{(-x)}$

All together for the ANN to be effective, the actuation function is critical. It is designed so that when the perceptron is fed with the correct input the perceptron becomes active, i.e. the function computes to a value near +1. Vice versa, the perceptron should become inactive, i.e. the function computes to a value near 0, when it is fed with the incorrect input (Awad and Khanna, 2015).

It is the most fundamental pieces of a neuron. The nonlinearity of the activation function makes it possible to approximate any function. It squashes numbers to the range [0, 1].

2.1.4 Cost function

As the prominent saying goes, "we gain from our slip-ups". Which is valid, in actuality, as an awful choice frequently is trailed by an undesirable occasion. Whenever an individual is looked with a comparable decision, they are more averse to settle on that equivalent wrong choice once more.

This is also true for ANNs, where the decision is a prediction which is associated with a cost. This cost is calculated with a cost function based on the error of the prediction. There

are a few cost functions utilized for machine learning, yet for this project, the logistic regression cost function (equation below) is used. It is based on the maximum likely-hood estimation in probability theory (Raschka, 2015).

$$J(W) = -1/M\left(\sum_{m=1}^{M} y_m \log g(x_m) + (1 - y_m) \log(1 - g(x_m))\right)$$

Since the desired outputs from the training examples y_m {0, 1}, one of the two terms is always zeroed out. This can then be seen as two cost functions corresponding to the possible values of y_m, where each assigns a very high cost to a prediction opposite to what the target output is. This cost function was, and is often, chosen for ANNs because of its feature of being a convex function, making it easier to minimize than a non-convex cost function.

2.1.5 Forward propagation

The manner in which the ANN makes a prediction is by taking the given information parameters and enhancing or debilitating the initiation of neurons in the system in different ways, yielding some output parameters that are probabilities for each yield case. The weights of the connections decide how much each perceptron's activation is to be changed before being fed to the following layer of the ANN. This process is called forward propagation (Russel and Norvig, 1995).

The activation for a neuron of the first layer is simply the input parameter for that neuron.

2.2.6 Backward propagation

To limit the prediction cost, we have to investigate the factors contributing to calculating the cost. There are three: input signal, the weights, and the activation function. Changing the input to get the right output is impossible since the objective here is to get the system to utilize a genuine image to give a satisfactory answer. The activation function is out of our control when the learning is in progress, hence it is the weights that need to be changed to minimize the cost (Bruce & Otter, 2016)

To change the weights of the system in a manner that limits the prediction cost, we need to grasp how the weights influence the output. The BP ANNs represents a kind of ANN, whose learning's algorithm is based on the Deepest-Descent technique. If provided with an appropriate number of Hidden units, they will also be able to minimize the error of nonlinear functions of high complexity (Buscema, 1998).

2.2 Generalization and overfitting

In machine learning, there is a common problem which is over-fitting of the algorithm. The reason for utilizing an ANN is, to generalize from the training examples to all possible inputs, rather than coordinating the training inputs superbly and be unfit to make great expectations on new data sources. Imagine a 5th order polynomial matching 4 data points perfectly. The impact of this in an ANN is that it figures out how to predict training image set effectively and correctly, in the long run learning the possible noise of the inputs themselves. To handle this issue, the dataset is part into three subsets. One for training, one for validation and one for testing. While training the network on the training set, the

cost of the errors on the validation set are plotted for each iteration. This will produce a curve of a decreasing cost for the first iterations until the ANN has been over-fitted, where the cost on the validation set will start to increase again. With this data, one can set the optimal number of epochs (iterations) for the learning algorithm for it to generalize satisfyingly. The test set is then used to calculate how many percent of the examples in the test set the ANN predicts correctly (Bruce & Otter, 2016).

2.3 Autonomous Vehicles

An Autonomous Vehicle (AV) is an innovation that plans to increase independent vehicle driving altogether or partially for self-sufficient and safety purposes. The technology itself consists of four underlying technologies: environment perception and modeling, localization and map building, path planning and decision-making, and motion control (Cheng, 2011). An AV uses machine vision, such as 3D cameras, and other sensors (such as Laser-Imaging Detection and Ranging – LIDAR -and GPS). Input signals from the machine vision and sensor are integrated with stored data by artificial-intelligence software to decide how the vehicle should operate based on traffic rules (Manyika et al., 2013).

2.4 Artificial Neural Network in Autonomous Vehicles

In most AVs, traffic recognition and vehicle driving are two separate modules. The early AV systems, however, used road images as input for driving commands. One example is the Autonomous Land Vehicle in a Neural Net (ALVINN). This is the Carnegie Mellon University's AV that uses a single hidden layer back-propagation network, with the input consisting of a 30×32 two-dimensional video-frame (Cheng, 2011).

Chapter Three

Research Methodology

3.0 Introduction

This chapter aims to explain in details the research methods and the methodology implemented for this project. This chapter will explain the choice of the research approach, the research design, as well as the advantages and disadvantages of the research tools chosen. This will be followed by a discussion on their ability to produce valid results, meeting the aims and objectives set by this dissertation. The chapter tells more about the data analysis methods which have been used. It concludes with a brief discussion on the ethical considerations and limitations posed by the research methodology, as well as problems encountered during the research.

3.1 Data Creation and Development

3.1.1 Convolution Neural Network (CNN) Training and Testing Data

Convolution Neural Networks require a large amount of training and validation images to be tuned to a best local maximum in terms of measured performance against the validation set and test set. The actual global maximum will most likely never be found and is computationally impractical to attempt to make sure it is discovered every time. It is often also not in our interest to find the global maximum, as such a solution is likely to fall victim to being an overfit model that performs very well on the validation set, and perhaps on your test set, but on a broader class of images, it cannot perform as well (Kornhauser, 2016).

The crucial idea to take away is that ignoring computational limitation, more information (more images) is in every case better, assuming those images are well curated to

accurately and proportionally resemble the diverse set of scenarios the detector is expected to perform under. For instance, in the event that one is distinguishing traffic lights, and every one of the pictures are of a red light 1 meter before the vehicle, where the traffic light takes up an enormous bit of the picture and is brilliantly lit on a radiant day, and being seen from a precisely opposite point to the face of the light, at that point preparing an exact finder to identify such a sign is an a lot simpler assignment. It is a simpler errand, yet then your indicator is likewise just getting down to business well under that precise, simple situation. In the event that the red light is seen at an edge, on the off chance that it is in part impeded, on the off chance that it is under the shadow of a tree, in the event that it is 10 meters away, and so on, any of those practical situations in reality or blend of them will battle to be distinguished by the identifier. What's more, such a poor detector can't be utilized by itself as a centerpiece in autonomous vehicle expected to protect riders. Training a CNN involves creating a network architecture by tuning hyperparameters (number of layers, types of layers, order of layers, dimensions of kernels, number of kernels, pooling sizes, pooling padding size, dimensions of input and output to a layer, et cetera) and then letting the optimizer run across thousands of images in the training set, thousands of times, to estimate optimal values for the thousands of parameters in the layers via backpropagation and stochastic gradient descent. The measure of parameters being assessed is colossal, and the result is in large part dependent on a well-curated choice of images for the training set. It is additionally critical to take note of that "thousands" can without much of a stretch be supplanted by "several thousands," "many thousands," or "millions" and onwards. Everything relies upon numerous factors, such as the difficulty of the detection problem, the measure of

preparing information accessible, the simplicity of development of new information, and the computational "capability" (Hardware) accessible to the researcher.

3.2 Abuja Traffic Light Data set

Traffic Light images were gotten from the streets of Abuja metropolis. The Abuja Traffic light Data set, is a dataset containing three traffic light colors, with light sizes ranging from 6x6 to 64x64 pixels. Images were collected using several different cameras and vary in size from 640x480 to 1024x522 pixels. Each annotation of a sign includes many useful facts such as light type, light position and size, and a few more statistics about what camera took the image. Some examples of such Images:

Figure 3. 1: Image taken at Abuja, Nigeria

3.3 Training and Test Set Construction

It is essential to recollect these sets of images need to precisely represent the environment in which the self-governing vehicle will be utilized. The objective is to have the system classify these pictures with high precision and speed. The dimension of the images is actually not much of a problem since based on other research papers, this is well inside the scope of measurements of preparing images used to develop similar CNN's. The main essentialness of bringing this up is that the design of the CNN needs to match the dimensions of the image it is being utilized on. So it should be trained and tested on real-world images..

Looking through my Abuja Dataset, some images largely fall into one of two groups: 704 x 480 or 640 x 480. Without distorting the image to stretching, those two images can have their dimensions scaled down to 64 x 64. It is much preferred, however, due to the vast majority of traffic light falling into the first category, there was no choice but to go with using the 704 x 480 category. Due to the non-availability of time, images having a higher cover of traffic light were selected to allow for quick processing and result delivery.

Python scripts were developed and implemented via Spyder software, to automatically go through the 55 images, and sort them into two groups. I am grouping them into training and test sets.

DATA SET	**NO OF IMAGES**
Training Set	55
Test Set	14

Table 3. 1: Training, Validation, Testing Set Counts

In the diagram below, the Imageset was classified into;

- **Training Set**: Collection of input-output patterns that are used to train the network.
- **Test Set**: Collection of input-output patterns that are used to assess network performance.

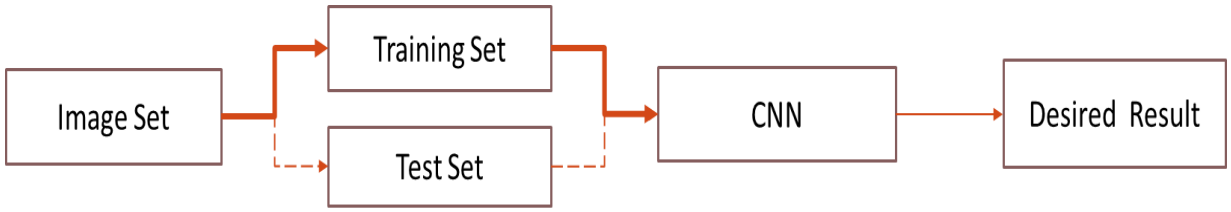

Figure 3. 2: Block Diagram of the Image set

3.3 CNN Introduction

The reason why I used CNN is not trivial. Many alternatives can be explored, such as spiking neural networks, deep neural networks, recurrent neural networks, and etcetera. However, Convolution Neural Networks happens to be a common choice for computer vision challenges such as image recognition and pattern recognition. CNNs are extensively employed in pattern and image-recognition challenges as they have a number of advantages compared to other techniques.

Convolutional Neural Networks are a type of artificial neural network. Artificial neural networks (ANNs) are models that are biologically inspired variants of real neural networks within a biological brain. Essentially, an ANN is a network of neurons where each neuron is only connected to certain other neurons, and all neurons connections are assigned weights based on how paramount those connections are. It is all these weights that are

being optimized/learned via training the CNN on the training set. A neural network is a system of interconnected artificial “neurons” that exchange messages between each other (Kornhauser, 2016).

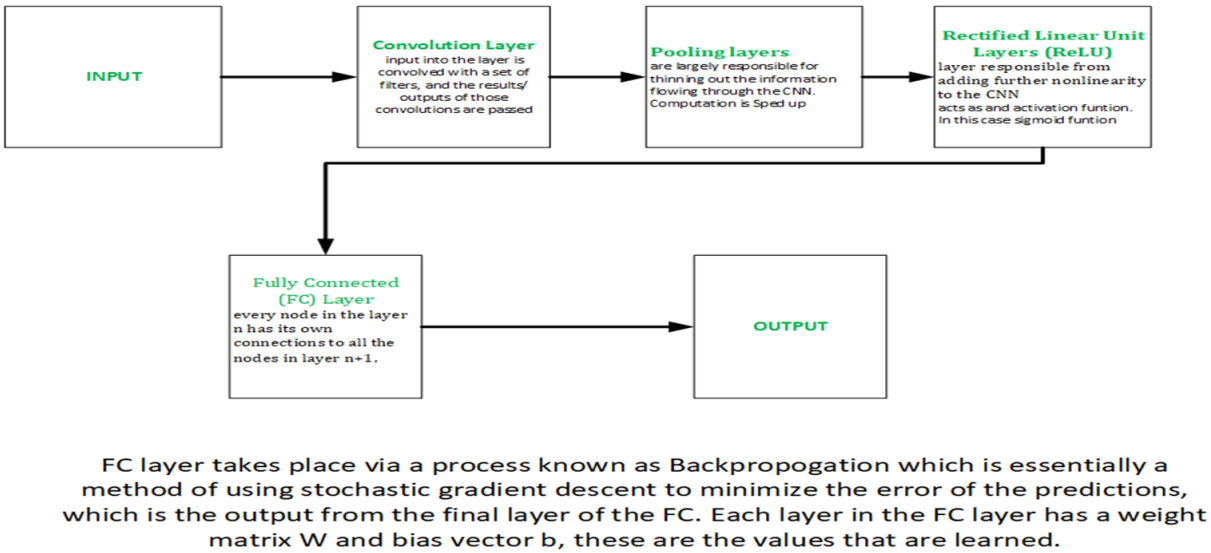

Figure 3. 3: Showing the sequence of steps employed by a typical CNN

The connections have numeric weights that are tuned during the training process so that a properly trained network will respond correctly when presented with an image or pattern to recognize. The network consists of multiple layers of feature-detecting “neurons.” Each layer has many neurons that respond to different combinations of inputs from the previous layers. As shown in Figure 1, the layers are built up so that the first layer detects a set of primitive patterns in the input, the second layer detects patterns of patterns, the third layer detects patterns of those patterns, and so on (Hijazi et al., 2010).

The composition of a CNN will be broken further. A Convolution Neural Network is an ANN composed of several types of layers. These layers include:

- Convolution Layer

- Pooling Layer
- Rectified Linear Unit Layer
- Fully Connected layer

3.3.1 Convolution Layer

The convolution layer serves to extract different features of the input. The first convolution layer extracts low-level features like edges, lines, and corners. Higher-level layers extract higher-level features. Figure 6 illustrates the process. The input is of size N x N x D and is convolved with H kernels, each of size k x k x D separately. Convolution of input with one kernel produces one output feature, and with H kernels independently produces H features. Starting from the top-left corner of the input, each kernel is moved from left to right, one element at a time. Once the top-right corner is reached, the kernel is moved one element in a downward direction, and again the kernel is moved from left to right, one element at a time. This process is repeated until the kernel reaches the bottom-right corner. Convolutional Layers (CONV). The CONV layer is where CNN gets its name from, as stated earlier, it is essentially a process where the input into the layer is “convolved” with a set of filters, and the outputs of those convolutions are passed as input into the next layer. A filter can be thought of as a sliding window with weights (a matrix of numbers) that is sliding across the entire image (a bigger matrix of numbers), and at each position it takes a dot product with the portion of the image it covers, which will be part of the output (Kornhauser, 2016). The weights on these filters are what is being learned during training. The filter object in a CONV layer is four dimensional: (1) number of filters, (2) number of channels (number of input matrices), (3) filter height, and (4) filter width.

Usually, it is very hard for the human eye to get an idea of what these filters are doing and especially if the filters do not come from the first convolutional layer.

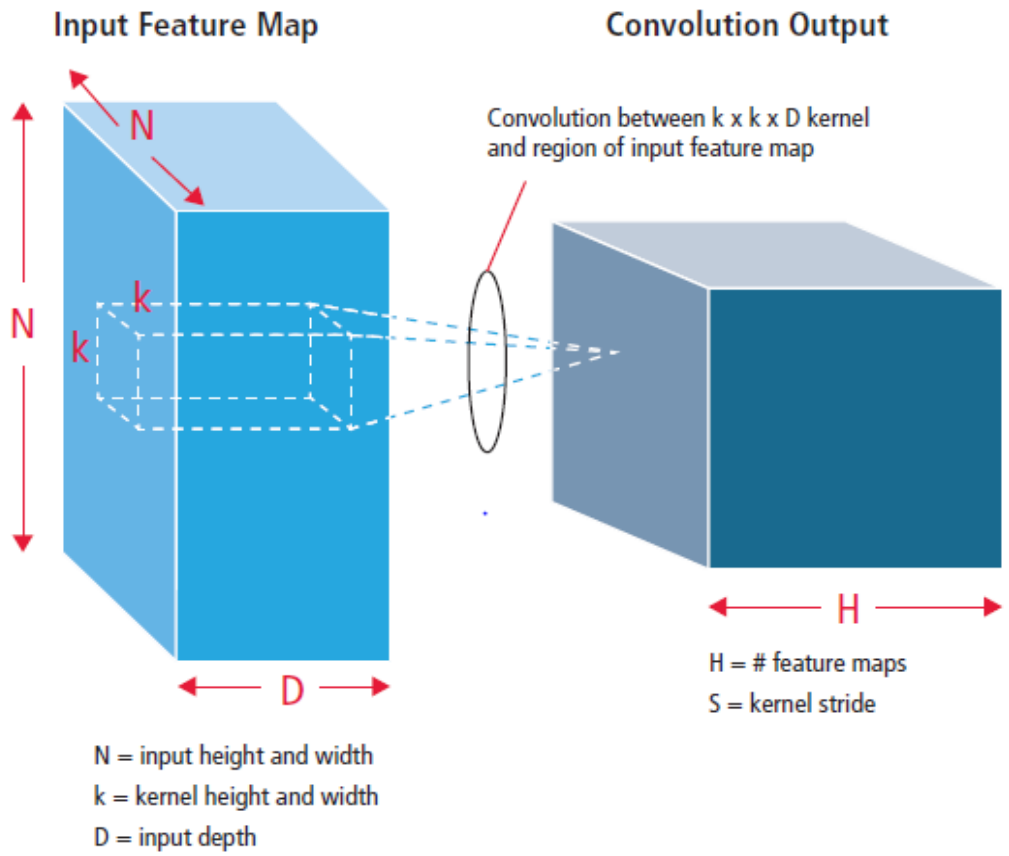

Figure 3. 4: Pictorial representation of a convolution process (Hijazi et al., 2010)

3.3.2 Pooling Layer

The POOL layers are largely responsible for thinning out the information (originally the image when it first enters) flowing through the CNN, which is good, as it can speed up computation and remove unnecessary details or noise. However, if too much pooling is used, it can thin out the data too quickly and impair object detection, thus finding the right balance is important. The intuition behind the idea of pooling is that after a filter has

identified a feature (and thus having a high dot product value), the exact location of the feature is not important, but rather what is important is only its relative location to other features. Thus one can divide the current input matrix into 2x2 or 3x3 tiles, and take the maximum, average, or L2-norm of those values, and have that one value replace those 4 or 9 values in the output (I choose to use max pooling as that has been shown to work best in practice). For my CNNs they all use max pooling and the pool size is 2x2. See below illustration made to show the max pooling occurring in this CNN (Hijazi et al., 2010).

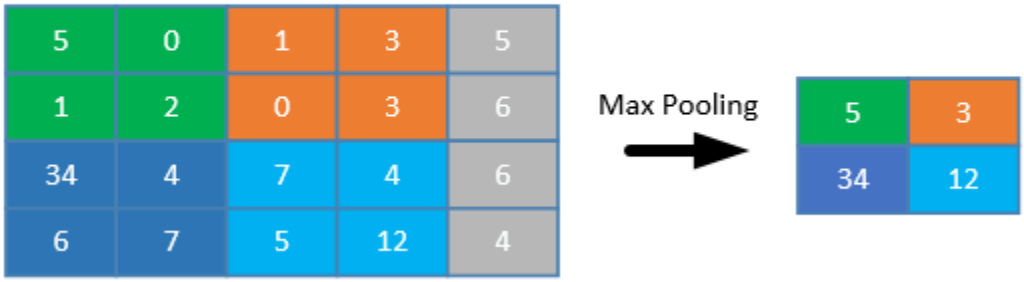

Figure 3. 5: Example of Max-Pooling technique

3.3.3 Rectified Linear Unit Layer

Rectified Linear Unit Layers (ReLU). The ReLU layer is the layer responsible for adding further nonlinearity to the CNN, it is supposed to act as an activation function to allow CNN to train faster. There is a debate about which function to use in this layer. However, there are three major functions in use:

1. ReLU Activation function
2. Sigmoid Activation Function
3. Tanh Activation Function

The main reason why we use the sigmoid function is that it exists between(0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since the probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. However, ReLU is the most used activation function in the world right now. Since it is used in almost all the convolutional neural networks or deep learning.

As such, I used Both the ReLU and Sigmoid AF to carry out my research with the result discussed in chapter 4 of this thesis.

3.3.4 Fully Connected Layer (FC)

Fully Connected (FC) Layer. This is typically the last layer in the CNN after some combination of any amount of layers before, where their types correspond to the previous three layers mentioned. The FC layer is unique in that it is the only section of the CNN where every node in the layer n has its own connections to all the nodes in layer n+1, so long as layers n and n+ 1 are part of the FC section. As one can imagine, so many weights make this computationally expensive, thus having combinations of CONV, POOL and ReLU layers before this step, and thus significantly reducing the size of the input into the FC layer is very important. If the FC layer has Hidden Layers (HLs) within it, which it usually does, then it can also be considered a Multilayer Perceptron (MLP). Learning in the MLP/FC layer takes place via a process known as Backpropagation which is essentially a method of using stochastic gradient descent to minimize the error of the predictions, which is the output from the final layer of the FC. Each layer in the FC layer has a weight matrix W and bias vector b; these are the values that are learned. The last layer of the CNN in this research is logistic regression (LR) layer; this is because it is a common technique shown to improve performance. The hidden layer (HL) essentially

transforms the input into a linearly separable space, and then the Logistic Regression (LR) layer classifies (makes predictions from) the output from HL.

3.4 Back Propagation

The learning algorithm's goal is to minimize the error between the expected (or teacher) value and the actual output value that was determined in the Forward Computation while the weight and bias is updated. The following steps are performed:

- Randomly choose the initial weights
- In case the error is too large and for each training pattern, apply the inputs to the network
- Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer
- Calculate the error at the outputs
- Use the output error to compute error signals for pre-output layers
- Use the error signals to compute weight adjustments
- Apply the weight adjustments
- Evaluate performance using the test set.

(Murakami, Okuyama, & Abdallah, 2018)

The figure below shows the block diagram of the learning process

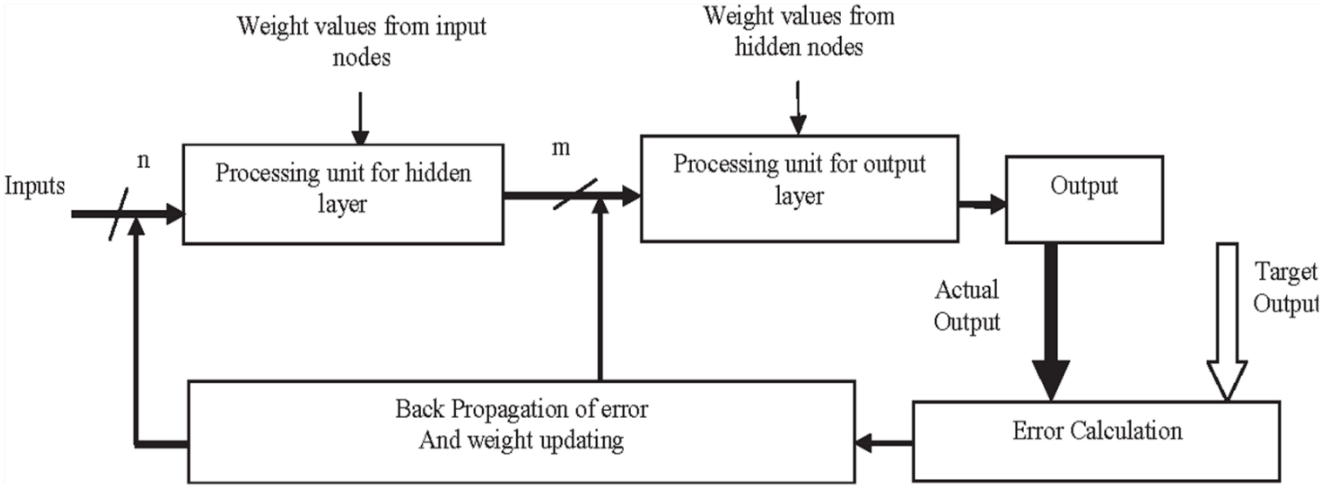

Figure 3. 6: Showing the steps necessary throughout the learning process

The final construction for the CNN in this study is as follows:

INPUT IMAGE >>> [[CONV -> POOL -> RELU]]*3 >>> HL>>> RELU -> LR -> (**OUTPUT**)

From the above, the "*3" notation means the 3 layers in the brackets are repeated 3 times. So basically the CNN can be thought of as having 12 layers in between input and output. Various architectures were attempted and this one seemed to work best in my simulation and instinctively appeared to be legitimate, to help for visualization. Below is an illustration of a typical CNN architecture:

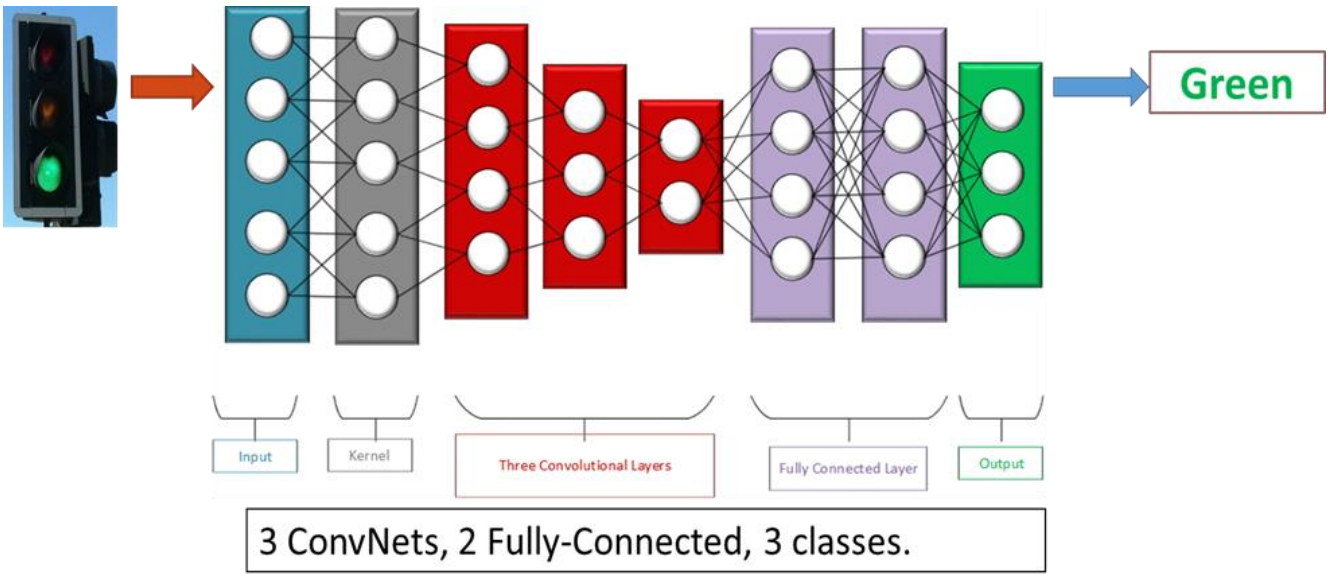

Figure 3. 7: Architecture of the proposed CNN

3.5 Proposed System Architecture

The figure below shows the proposed system architecture of the neural network for its implementation in an autonomous vehicle. The project consists of two phases, that is, the software implementation and the hardware implementation. Due to the limited period, the software implementation was carried out with the hardware phase left for future work. The hardware consists of an FPGA which is used for parallelism and speed of computation as the autonomous vehicle would require fast processing to make decisions, to avoid events of failure.

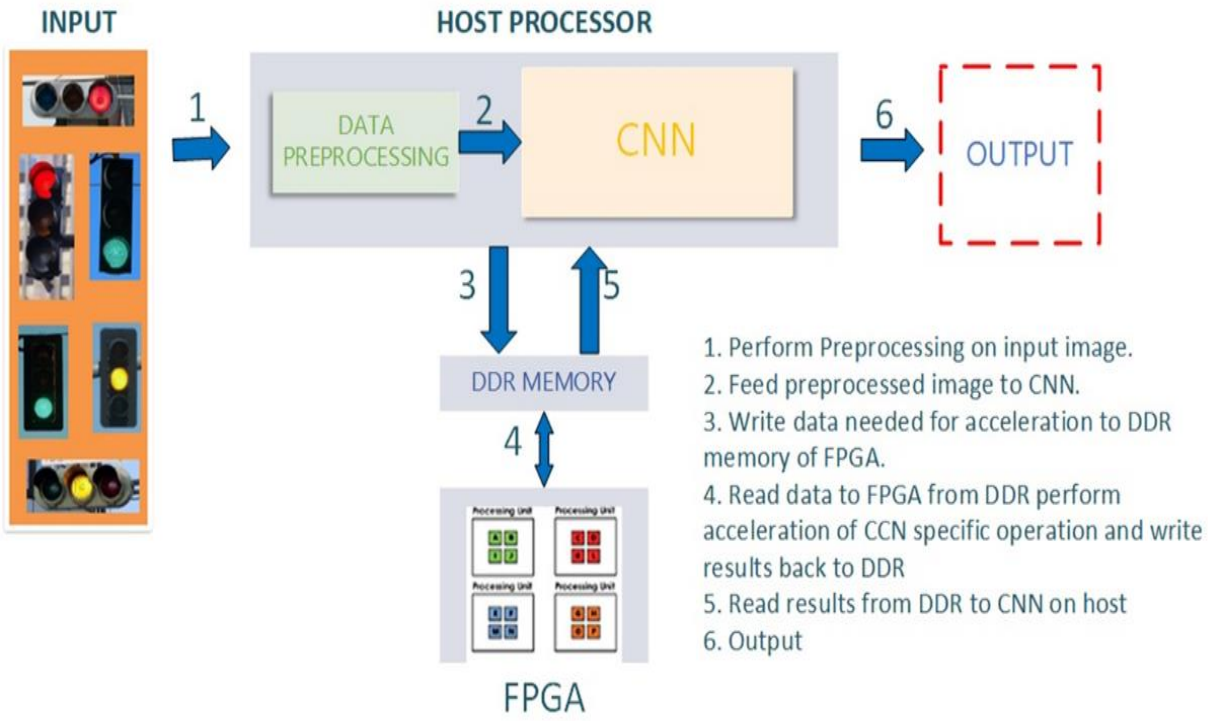

Figure 3. 8: Architecture of the proposed CNN

3.5 CNN Construction and Implementation

Choosing how to architect a CNN was difficult in enormous part in light of the fact that at the earliest reference point, one battles with how to settle on choices naturally. This is on the grounds that a novice with CNNs has zero or almost no instinct about such theoretical ideas. For instance, if the CNN isn't performing well, what hyper-parameters should be changed?

This design was implemented on a:

- **Hardware:** 2.40 GHz Intel Core-i5 4770 and 8 GB of RAM
- **Software:** Anaconda/Spyder, Python 3.7.6 Version

There are so many hyperparameters that were changed, such as the epoch, the number of filters, and the number of layers, amongst others. For this intuition, I had to spend a lot of time on training neural networks, making such changes, and observing the effect. The study mainly relied on the Deep Learning Tutorial on Datacamp. I finally settled on using the ReLU activation function for the three convolution layer, ten epochs, and two fully connected layers to achieve a result which is detailed in the next chapter.

Chapter Four

Evaluation and Result

4.1 Evaluation

4.1.1 Execution time evaluation with RELU AF

The table below shows that the execution time taken was 10s to train & 14s to predict, amounting to 24s in total.

No of Epochs	ET for training (seconds)	ET for prediction (Seconds)	Total time
10	10	14	24

Table 4. 1: Execution time evaluation with ReLU AF

4.1.2 Execution time evaluation with Sigmoid AF

The table below shows that the execution time taken was 11s to train & 14s to predict, amounting to 25s in total.

No of Epochs	ET for training (seconds)	ET for prediction (Seconds)	Total time
10	11	14	25

Table 4. 2: Execution time evaluation with Sigmoid AF

4.1.3 Accuracy evaluation with RELU AF

As can be seen from the figure below, the 3 layers produced a correct detection rate of 57.14%, while there was accuracy of 35% and 26.41% for 2 layers and 4 layers respectively.

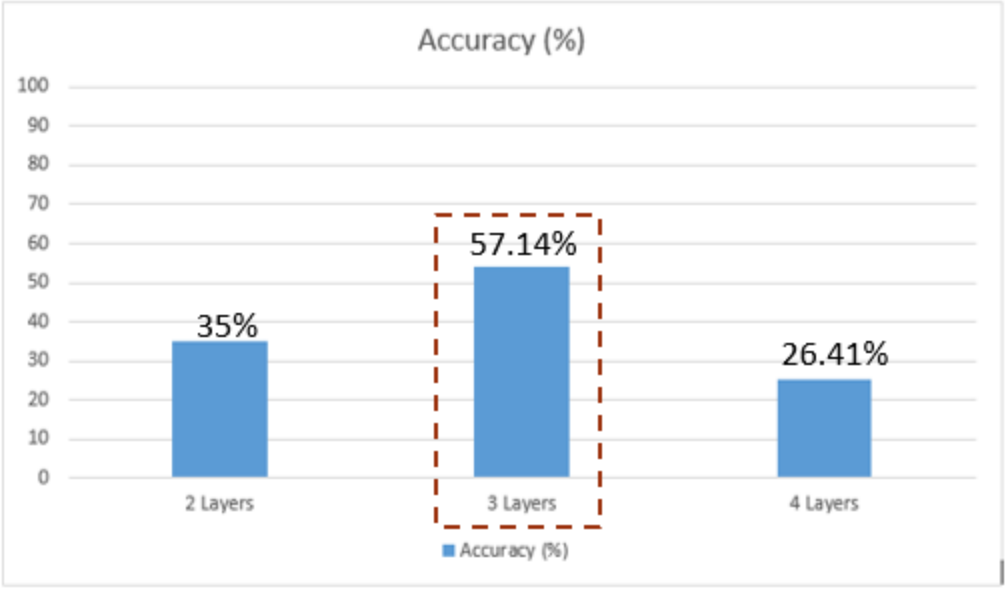

Figure 4. 1: Accuracy graph for different Hidden layers using ReLU AF

4.1.4 Accuracy evaluation with Sigmoid AF

As can be seen from the chart below, the 3 layers produced a correct detection rate of 35.71%, while there was an accuracy of 29% and 31% for 2 layers and 4 layers respectively.

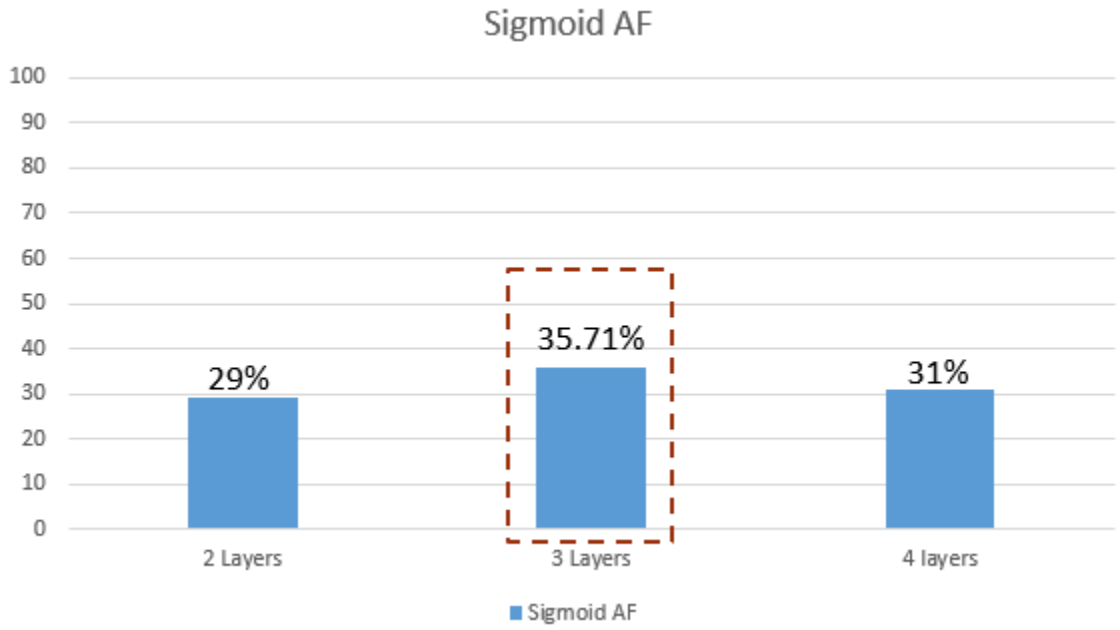

Figure 4. 2: Accuracy graph for different Hidden layers using Sigmoid AF

4.2 EVALUATION SUMMARY/ RESULT

The summary of the final result is displayed in the figure below;

- Accuracy Difference of 21.43
- Speed difference of 1s
- RELU AF provided the optimum performance

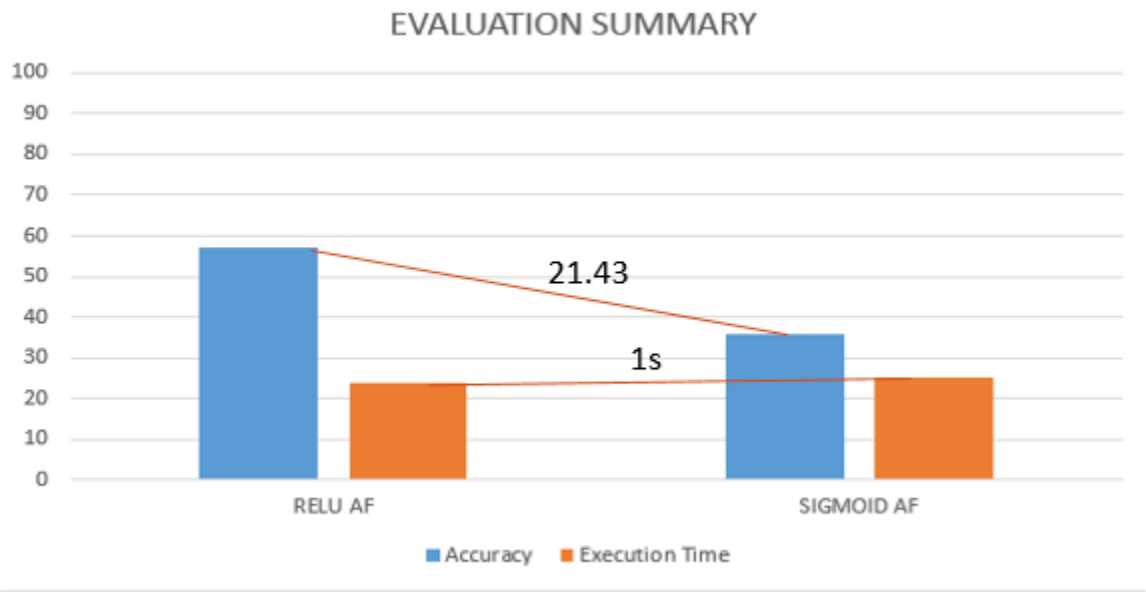

Figure 4. 3: Summary that determines the optimum parameters

Chapter Five

Conclusion

This thesis examined the design of a neural network architecture for autonomous vehicles, which was implemented using python programming language and keras library. With 55 training images and 14 test images, the neural network implementation produces a correct detection rate of 57.14%, which is due to the small number of images. Also, its speed of computation was faster using ReLU activation function. However, the speed and accuracy were affected by trying different parameters, which resulted in different outputs. To summarize:

- **Problem** - Runtime and accuracy in autonomous vehicle classification poses a big drawback.
- **Merit** - Actualization and hope it brings to autonomous vehicle safety on our streets.
- **Research Goal** - Design a neural network architecture for traffic light detection in autonomous vehicles.
- **Achievements** - Ability to use a real life data set (images from Abuja metropolis) to achieve a 57% correct detection rate.

Future Research Work – The future work will hopefully focus on the use of more images, implementation on hardware, and further evaluation of the CNN model for Traffic-Light Recognition in Autonomous Vehicles.

Final Thoughts

Working on designing a neural network architecture for traffic light detection in autonomous vehicles was a great pleasure. Based on all the work that has been done so

far, training and testing on CNN seem to be a very tractable and effective method. There are so many new techniques one can try with relative ease. This will hopefully be implemented in my future works! Nigeria, being a developing nation with limited research on autonomous vehicle technology, is required to explore this emerging technology further to ensure a high level of safety and trust. We can make a difference.

REFERENCE

Autonomous Intelligent Ve- hicles. (n.d.).

Bell, J. (n.d.). *Jason Bell-Machine Learning_ Hands-On for Developers and Technical Professionals-Wiley (2014)*.

Bruce, W., & Otter, E. V. O. N. (2016). *Artificial Neural Network Autonomous Vehicle Artificial Neural Network controlled vehicle*.

Civín, L. (1998). *Back propagation neural metworks user manual. 33*(2), 1–28.

Hijazi, B. S., Kumar, R., Rowen, C., & Group, I. P. (2010). *What is a CNN?* 5–16. https://doi.org/10.1142/9789812798589_0002

Kornhauser, A. A. L. (2016). *CONVOLUTIONAL NEURAL NETWORKS APPLIED TO TRAFFIC SIGN DETECTION IN GRAND THEFT AUTO V*. (June).

Lee, G. G., & Park, B. K. (2017). Traffic light recognition using deep neural networks. *2017 IEEE International Conference on Consumer Electronics, ICCE 2017*, 277–278. https://doi.org/10.1109/ICCE.2017.7889317

Manyika, J., Chui, M., Bughin, J., Dobbs, R., Bisson, P., & Marrs, A. (2013). Disruptive technologies: Advances that will transform life, business, and the global economy. *McKinsey Global Institute*, (May), 1–164. Retrieved from http://www.mckinsey.com/insights/business_technology/disruptive_technologies%5Cnhttp://www.chrysalixevc.com/pdfs/mckinsey_may2013.pdf

Murakami, Y., Okuyama, Y., & Abdallah, A. Ben. (2018). *SRAM Based Neural Network System for Traffic-Light Recognition in Autonomous Vehicles Paper Contribution System Architecture Preliminary Evaluation Conclusion and Future work*. 1–27.

Murakami, R., Okuyama, Y., & Abdallah, A. Ben. (2018). *”Animal Recognition and*

Identification with Deep Convolutional Neural Networks for Farm Monitoring", Information Processing Society Tohoku Branch Conference, Koriyama, Japan.

The H. Vu, Okuyama, Y., & Abdallah, A. Ben. (2019), *''Analytical performance assessment and high-throughput low-latency spike routing algorithm for spiking neural network systems,'' The Journal of Supercomputing. DOI: https://doi.org/10.1007/s11227-019-02792-y*

The H. Vu, Abdallah, A. Ben. (2019), *"A Low-latency K-means based Multicast Routing Algorithm and Architecture for Three Dimensional Spiking Neuromorphic Chips", IEEE International Conference on Big Data and Smart Computing (BigComp 2019), Kyoto, Japan, Feb 28 - Mar 2, 2019 [best paper award]*

The H. Vu, Murakami, R., Okuyama, Y., & Abdallah, A. Ben. (2018). *"Efficient Optimization and Hardware Acceleration of CNNs towards the Design of a Scalable Neuro-inspired Architecture in Hardware", IEEE International Conference on Big Data and Smart Computing (BigComp 2018), Shanghai, China, January 15-18, 2018.*

APPENDIX

CODE

#Image Preprocessing and CNN Training

```
import numpy
import matplotlib.pyplot as plt
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
import load_data
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from tensorflow.python.keras.callbacks import TensorBoard
import time
from tensorflow import keras
import tensorflow as tf

K.set_image_dim_ordering('tf')
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

def pre_process(X):

    # normalize inputs from 0-255 to 0.0-1.0
    X=X.astype('float32')
    X = X / 255.0
    return X

def one_hot_encode(y):

    # one hot encode outputs
    y = np_utils.to_categorical(y)
    num_classes = y.shape[1]
    return y,num_classes

def define_model(num_classes,epochs):
    # Create the model
```

```
    model = Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), padding='same',
activation='relu', kernel_constraint=maxnorm(3)))
    model.add(Conv2D(32, (3, 3), activation='sigmoid'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='sigmoid', padding='same',
kernel_constraint=maxnorm(3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='sigmoid', kernel_constraint=maxnorm(3)))
    model.add(Dense(128, activation='sigmoid'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    lrate = 0.01
    decay = lrate/epochs
    sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    print(model.summary())
    return model


# load data
X,y=load_data.load_datasets()

# pre process
X=pre_process(X)

#one hot encode
y,num_classes=one_hot_encode(y)


#split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=7)

epochs = 10
#define model
model=define_model(num_classes,epochs)


# Fit the model
history=model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs,
batch_size=32)
```

```
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

# serialize model to JSONx
model_json = model.to_json()
with open("model_face.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_face.h5")
print("Saved model to disk")
```

#CNN TESTING

```
import numpy as np
import os
from scipy import  misc
from keras.models import model_from_json
import pickle

# Loading int2word dict
classifier_f = open("int_to_word_out.pickle", "rb")
int_to_word_out = pickle.load(classifier_f)
classifier_f.close()
```

```
def load_model():
    # load json and create model
    json_file = open('model_face.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights("model_face.h5")
    #print("Loaded model from disk")
    return loaded_model

def pre_process(image):
    image = image.astype('float32')
    image = image / 255.0
    return image

def load_image():
    index=0
    for index in range( len(os.listdir("images")) ):
        img=os.listdir("images")[index]
        image=np.array(misc.imread("images/"+img))
        image = misc.imresize(image, (64, 64))
        image=np.array([image])
        image=pre_process(image)
        model=load_model()
        prediction=model.predict(image)
        print(index+1,":",int_to_word_out[np.argmax(prediction)],"=",np.max(prediction) )

    return image

load_image()#run it
'''image=load_image()
model=load_model()
prediction=model.predict(image)


#print the predictions
print(int_to_word_out[np.argmax(prediction)],np.max(prediction) )
'''
```