A PROJECT REPORT ON

**“Traffic Sign Recognition Using Deep Learning”**

**Submitted**

*In the partial fulfilment of the requirements for*

*The award of the degree of*

**BACHELOR OF TECHNOLOGY**

In

**COMPUTER SCIENCE and ENGINEERING**

Submitted By

B.VENKAT (171FA04007)

J HARI KRISHNA (171FA04023)

Under the esteemed guidance of

**Mr. Ongole Gandhi, Assistant Professor (M.Tech)**

**Dr. Dondeti Venkatesulu, Professor and HOD**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**VIGNAN’S FOUNDATION FOR SCIENCE TECHNOLOGY AND RESEARCH**

(**Accredited by NAAC “A” grade**) **Vadlamudi, Guntur.**

A PROJECT REPORT ON

**“Traffic Sign Recognition Using Deep Learning”**

**Submitted**

*In the partial fulfilment of the requirements for*

*The award of the degree of*

**BACHELOR OF TECHNOLOGY**

In

**COMPUTER SCIENCE and ENGINEERING**

Submitted By

B.VENKAT (171FA04007)

J HARI KRISHNA (171FA04023)

Under the esteemed guidance of

**Mr. Ongole Gandhi, Assistant Professor (M.Tech)**

**Dr. Dondeti Venkatesulu, Professor and HOD**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**VIGNAN’S FOUNDATION FOR SCIENCE TECHNOLOGY AND RESEARCH**

(**Accredited by NAAC “A” grade**) **Vadlamudi, Guntur.**

****

**CERTIFICATE**

This is to certify that the Project Report entitled **“Traffic Sign Recognition using Deep Learning”** that is being submitted by **B Venkat (171FA04007), J Hari Krishna (171FA04023)** in partial fulfilment for the award of B.Tech degree in Computer Science and Engineering to the Vignan’s Foundation for Science, Technology and Research, Deemed to be University, is a record of bonafide work carried out by him under my supervision.

**Mr. Ongole Gandhi External Examiner Dr. Dondeti Venkatesulu Assistant Professor(M.Tech) Professor, HOD**

# ACKNOWLEDGEMENT

We are very grateful to our beloved Chairman **Dr. Lavu. Rathaiah**, and Vice Chairman

**Mr. Lavu. Krishna Devarayalu,** for their love and care.

It is our pleasure to extend our sincere thanks to Vice-Chancellor **Dr. M.Y.S. Prasad** and Dean Engineering & Management, **Dr. M. Santhi Sree Rukmini,** for providing an opportunity to do my academics in VFSTR.

It is a great pleasure for me to express my sincere thanks to **Dr. Dondeti Venkatesulu HOD, CSE** of VFSTR**,** for providing required infrastructure to do my Project.

We extend our whole hearted gratitude to **Mr. Ongole Gandhi**, under whose valuable guidance that the project came out successfully after each stage.

We feel it our responsibility to thank **Mrs. D. Radha Rani** and **Mr. Vijay Babu,** for helping in all required aspects during the course.

We thank all our **Faculty** members and **Programmers** of Department of Computer Science and Engineering who provided technical support to us in our academics throughout course.

Finally, we wish to express thanks to our family members for the love and affection overseas and forbearance and cheerful depositions, which are vital for sustaining effort, required for completing this work.

With Sincere regards,

**B Venkat (171FA04007)**

**J Hari Krishna (171FA04023)**

Date:

**ABSTRACT**

In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc. are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly.

Automatic detection and recognition of traffic signs is very important and could potentially be used for driver assistance to reduce accidents and eventually in driverless automobiles. In this project, Deep Convolutional Neural Network (CNN) is used to develop an Autonomous Traffic Sign recognition system. The proposed system works in real time detecting and recognizing traffic sign images. The contribution of this project is also contains a dataset which used is the GTSRB (German traffic sign recognition benchmark). It contains a Train folder that has traffic sign images in 43 different classes, a Test folder that has over 12,000 images for testing purposes. A test.csv file that contains the path of the test images along with their respective classes.

The images were taken from different angles and including other parameters and conditions. A total of 50,000 images were collected to form the database which we named Traffic and Road Signs GTSRB (German traffic sign recognition benchmark). The CNN architecture was used with varying parameters in order to achieve the best recognition rates. Experimental results show that the proposed CNN architecture achieved an accuracy of 100%, thus higher than those achieved in similar previous studies**.**  This project provides the result that user will get a voice message if The vehicle pass through the traffic sign on the road . The user will follow the instruction as per the voice over .

**Keywords**: German traffic sign recognition benchmark; Convolutional Neural Networks (CNN); Traffic Sign Detection, Road Sign; Traffic Signs Recognition

**Table of Contents**

Table of Contents

[ACKNOWLEDGEMENT 4](#_Toc74170912)

ABSTRACT……………………………………………………………………………………………5

CHAPTER- 1…………………………………………………………………………………………..9

INTRODUCTION……………………………………………………………………………………..9

[CHAPTER- 2 14](#_Toc74170913)

[BACKGROUND STUDY 14](#_Toc74170914)

[2.1 Literature Review 14](#_Toc74170915)

[2.2 Existing Methodologies: 15](#_Toc74170917)

[2.2.1Traffic sign Detection using HSV colour space: 15](#_Toc74170918)

[CHAPTER -3 19](#_Toc74170935)

[PROPROSED WORK 19](#_Toc74170936)

[3.1 Experimental Data: 20](#_Toc74170940)

[3.2 Preprocessing: 21](#_Toc74170942)

[3.2.1 Training 22](#_Toc74170945)

[3.2.2 Data augmentation 22](#_Toc74170946)

[3.3 Machine Learning Models 22](#_Toc74170947)

[3.3.1 Logistic Regression 23](#_Toc74170948)

[3.3.2 K-Nearest Neighborhood Classifier 25](#_Toc74170949)

[3.3.3 Naïve Bayes Classifier 25](#_Toc74170950)

[3.3.4 Decision Tree 26](#_Toc74170951)

[3.3.5 Support Vector Machine 27](#_Toc74170952)

[3.4 Deep Learning Models 28](#_Toc74170953)

[3.4.1 Convolutional Neural Network 28](#_Toc74170954)

[3.4.2 Transfer Learning 32](#_Toc74170955)

3.4.3 Pre-trained Models……………………………………………………………….34

3.4.4 Deep Convolutional Neural Network Architecture…………………………………………………………35

[CHAPTER – 4 40](#_Toc74170956)

[EXPERIMENTAL RESULTS 40](#_Toc74170957)

[4.1 Results and Discussions: 40](#_Toc74170958)

[4.2 Dataset: 41](#_Toc74170962)

[4.3 Dropout: 43](#_Toc74170965)

[4.4 Histogram Equalization 45](#_Toc74170966)

[4.5 Data Augmentation 47](#_Toc74170967)

[4.5 Testing On New Images 49](#_Toc74170968)

[4.6 Visualizing Our Activation Maps 51](#_Toc74170969)

[Layer 1 51](#_Toc74170970)

[Layer2: 52](#_Toc74170971)

CHAPTER - 5………………………………………………………………………………………………………………………………..55

[5.1 Implementation 57](#_Toc74170972)

[5.1.1 Workflow 57](#_Toc74170973)

5.2 SAMPLE CODE……………………………………………………………………………………………………………………..57

5.2.1 traffic\_sign.py………………………………………………………………………………………………………………..57

5.2.2 gui.py…………………………………………………………………………………………………………………………….62

[Output 70](#_Toc74170974)

CHAPTER -6…………………………………………………………………………………………71

CONCLUSION……………………………………………………………………………………….71

REFERENCES……………………………………………………………………………………….72

**List of Figures and Tables**

Fig 1. 1 Traffic sign that represents speed limit 100 kmph………………………………….11

Fig 1.2 Traffic signs globally used…………………………………………………………...12

Fig 2.1 The HSV Colour Space………………………………………………………………16

Fig 2.2 converting the RGB image to HSV Image……………………………………………..16

Fig 2.3 colour space threshold segmentation step diagram……………………………….......…17

Table 1 HSV color space threshold segmentation ranges………………………………………..17

Fig 2.4 The threshold rough segmentation image……………………………………………18

Fig 3.1 Block diagram of supervised learning……………………………………………….19

Fig 3.2 Sample Images in GTSRB Dataset…………………………………………………..20

Fig 3.3 Sample images after preprocessing…………………………………………...21

Fig 3.4 Logistic Function graph……………………………………………………………...24

Fig 3.5 k-NN Algorithm……………………………………………………………………..25

Fig 3.6 Equation for Posterior Probability estimation………………………………………..26

Fig 3.7 Decision Tree Example………………………………………………………………26

Fig 3.8 Support Vector Machine……………………………………………………………..27

Fig 3.9 Pixel representation of Image………………………………………………………..28

Fig 3.10 Convolution Operation……………………………………………………………..29

Fig 3.11 ReLU Operation…………………………………………………………………….29

Fig 3.12 Feature Map Visualization after Convolution and ReLU Operations………………30

Fig 3.13 Pooling Operation…………………………………………………………………..30

Fig 3.14 Feature Map Visualization after Pooling Operation………………………………..30

Fig 3.15 Flattening…………………………………………………………………………...31

Fig 3.16 Architecture of Convolutional Neural Network……………………………………32

Fig 3.17 Architecture of a model based on CNN…………………………………………….33

Fig 3.18 Fine-tuning strategies……………………………………………………………….34

Fig 3.19 Size-Similarity matrix (L) & decision map for fine-tuning pre-trained models (R)...35

Table 2 Summary of the Initial Proposed CNN Network (Model 1)………………………...39

Table 3 Summary of the Improved CNN Network with less number of layers (Model 2)…….41

Fig 4.1 graph representation of Dataset in graph………………………………………………41

Fig 4.2 Normalized Images…………………………………………………………………..42

Fig 4.3 Models Performance on Grayscale Normalised Images, After

The Introduction Of Dropout………………………………………………………………...44

Fig 4.4 Models Performance On Grayscale Equalized Images, With Dropout……………...46

Fig 4.5 Sample of 10 images where our model got the predictions wrong……………….....47

Fig 4.6 Sample Of Augmented Images……………………………………………………...48

Fig 4.7 5 new traffic signs — color…………………………………………………………49

Fig 4.8 Grayscale images…………………………………………………………………….50

Fig 4.9 Convolutional Layer 1……………………………………………………………….51

Fig 4.10 Convolutional Layer 2……………………………………………………………...52

Fig 4.11 Convolutional Layer 3……………………………………………………………...53

Fig 4.12 Libraries used for Development…………………………………………………….56

Fig 5.1 Epochs for traning val accuracy………………………………………….......………63

Fig 5.2 Ephoch for training loss……………………………………………………………..63

Fig 5.3 Output 1……………………………………………………………………………...70

Fig 5.4 Output 2……………………………………………………………………………...71

Fig 5.5 Output 3……………………………………………………………………………...72

**List of Abbreviations**

AI : Artificial Intelligence

ANN : Artificial Neural Network

CNN (ConvNet) : Convolutional Neural Network

DL : Deep Learning

DNN : Deep Neural Network

DR : Diabetic Retinopathy

DT : Decision Tree

FE : Feature Engineering

HE : Hard Exudates

HOG : Histogram of Oriented Gradients

KNN : K – Nearest Neighbour hood

LBP : Local Binary Patterns

MA : Micro Aneurysms

ML : Machine Learning

MLP : Multi Layered Perceptron

NB : Naïve Bayes

ReLU : Rectified Linear Unit

**CHAPTER - 1**

**INTRODUCTION**

Traffic sign detection and recognition has gained importance with advances in image processing due to the benefits that such a system may provide. The recent developments and interest in self-driving cars has also increased the interest in this field. An automated traffic sign detection and recognition system will provide the ability for smart cars and smart driving. Even with a driver behind the wheel, the system may provide vital information to the driver reducing human errors that cause accidents. Certainly with such a system integrated into vehicles, it is expected that the number of car accidents will be reduced greatly saving human lives and the monetary value associated with car accidents. Automated systems will be able to control traffic on both open roads and intersections as well.



**Fig 1. 1 Traffic sign that represents speed limit 100 kmph.**

The motivation behind developing such a system is clear due to the benefits of such a system in saving lives and saving cost. Therefore, the objective of this work is to develop an automatic GTSRB (German traffic sign recognition benchmark) based on deep learning algorithm. The proposed system has the ability to recognize the signs within images captured by cameras and processed by a Deep CNN network. Most car accidents are caused by human error either by drivers not noticing a certain sign or with drivers driving against the direction set by a certain traffic sign (i.e. traffic sign setting speed at 100 KM and driver driving at a greater speed).. For this reason, this project carries the major objective of developing and improving the efficiency and robustness of the traffic sign detection system for Traffic Signs as well as handling associated issues, a recognition system should also classify traffic signs into different classes in real-time environment and avoid recognition errors.

Machine learning is divided into supervised learning, unsupervised learning, semi-supervised learning, and reinforced learning. In this project, the choice of deep learning for an unsupervised learning approach is done by design because even though basic traffic signs are limited yet combined with road signs, street name signs, etc. the dataset becomes larger with endless possibilities. The ultimate goal is to have a system fitted into cars and that can detect and recognize any traffic sign to assist the driver or assist by producing a voice message. With deep learning algorithms, unlabeled data can be used and the system can extract features automatically without human intervention.

**Types of traffic signs:**

There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to.



**Fig 1.2 Traffic signs globally used.**

Traffic signs are an integral part of our road infrastructure. They provide critical information, sometimes compelling recommendations, for road users, which in turn requires them to adjust their driving behavior to make sure they adhere with whatever road regulation currently enforced. Without such useful signs, we would most likely be faced with more accidents, as drivers would not be given critical feedback on how fast they could safely go, or informed about road works, sharp turn, or school crossings ahead. In our modern age, around 1.3M people die on roads each year. This number would be much higher without our road signs.Naturally, autonomous vehicles must also abide by road legislation and therefore recognise and understand traffic signs.

Traditionally, standard computer vision methods were employed to detect and classify traffic signs, but these required considerable and time-consuming manual work to handcraft important features in images. Instead, by applying deep learning to this problem

A deep neural network model that can classify traffic signs present in the image into different categories. With this model, we are able to read and understand traffic signs which are a very important task for all autonomous vehicles.

Every country has some standards set for the design of different traffic signs like U-turn, Left-turn, Right-turn, No-entry, etc. Traffic sign recognition is the process of automatically identifying which of the following class the sign belongs to. The earlier Computer Vision techniques required lots of hard work in data processing and it took a lot of time to manually extract the features of the image. Now, deep learning techniques have come to the rescue and today we will see how to build a traffic recognition system for autonomous vehicles.

Traffic sign classification is the process of automatically recognizing traffic signs along the road, including speed limit signs, yield signs, merge signs, etc. Being able to automatically recognize traffic signs enables us to build “smarter cars”.

Self-driving cars need traffic sign recognition in order to properly parse and understand the roadway. Similarly, “driver alert” systems inside cars need to understand the roadway around them to help aid and protect drivers.

# 

# CHAPTER- 2

# BACKGROUND STUDY

## Literature Review

# Nowadays, recognition and classification of traffic signs are very important, especially for unmanned automatic driving. Extensive research has been done in the area of recognition and classification of traffic and road signs. In [1], the authors proposed a Convolutional Neural Network and Support Vector Machines (CNN-SVM) method for traffic signs recognition and classification. The coloring used in this method is YCbCr color space which is input to the convolutional neural network to divide the color channels and extracting some special characteristics. SVM is then used for classification. Their proposed method achieved a 90.6% accuracy for traffic signs recognition and classification. In [2], the authors proposed a color based segmentation method with Histogram Oriented Gradients (HOG) for feature extraction followed by SVM for classification. The model used CIECAM97 for color appearance, this model was applied to a segment to extract color information. Another model used for shape features is FOSTS [3] which achieved a 95% accuracy. In [4], the authors proposed feature extraction through HOG and local binary pattern (LBP) which are then input into an Extreme Learning Machine Network for classification and recognition. In [5], the authors propose a traffic sign recognition system based on extreme learning machine (ELM). Their method consists of feature extraction through extraction of histogram of the oriented gradient variant (HOGv) features followed by a single classifier trained by ELM. In [6], the authors developed a new dataset consisting of 100,000 images and also proposed a traffic sign detection and classification method based on a robust end-to-end CNN. The method achieved 84% accuracy. In [7], the authors proposed a multi-scale deconvolution network (MDN) for localized traffic sign detection. The method achieved 95.1% accuracy. In [8], the authors presented a survey of available techniques for road traffic sign recognition systems using CNN. Their work presented the available proposed techniques in addition to the challenges faced by CNN methods in terms of time complexity and accuracy. They further proposed a method to overcome the challenges which utilized canny edge detection to highlight the edges of the traffic symbols which is then input into a CNN for classification. To enhance classification and recognition, the introduced fuzzy classification technique used in [9]. In [10], the authors proposed a traffic sign recognition and classification system based on scale-aware CNN. Their system consists of two CNN’s; one for region proposals of traffic signs and the other for classification of each region. In addition, fully convolutional network (FCM) is utilized to achieve scale invariant detection. The system achieved 94.88% precision accuracy. In [11], the authors proposed a knowledge-based recurrent attentive neural network (KBRANN) for small object detection. Their method achieved 81% accuracy. In [12], the authors proposed an efficient algorithm for traffic sign detection on low cost embedded systems. Their method consists of color thresholding, shape detection and sign validation. They utilized an efficient color thresholding technique based on the red-blue angle color transformation (RBAT) and the red color normalized. Ellipse fitting technique is also employed for detecting thecircular signs. HOG is employed for validation. Their method achieved 97% precession accuracy. In [13], the authors analyzed the spatial transformers and stochastic optimization methods for deep neural network for traffic sign recognition. They finalized this with a proposed system that achieved 99.71% accuracy

# Existing Methodologies:

# 2.2.1Traffic sign Detection using HSV colour space:

# Overview: Color is an important feature of traffic sign, and traffic sign can be quickly located by color segmentation. Compared with RGB color space and HSI color space, the HSV color space has a faster detection speed, less affected by illumination, and has a preferable segmentation advantage. Figure 1 shows the HSV color space converted from the RGB color space. It represents the points in the RGB color space by an inverted cone, where H is the hue, S is the saturation and V is the value.

# H indicates the color change of the image. The position of the spectral color is represented by the angle, and different color values correspond to different angles. Red, green and blue are 120◦ apart, that is, 0◦ , 120◦ and 240◦ , respectively. S denotes the proportion of the current color purity to the maximum purity with the maximum value of 1 and the minimum value of 0. V represents the brightness change of the image. The maximum value is 1 in white and the minimum value is 0 in black. In the HSV color space, given that V is a fixed value set and H and S are highly unrelated, the HSV color space has good illumination adaptability when the illumination conditions change, and its computational complexity is small, which are conducive to the color space threshold segmentation

# 

# Fig 2.1 The HSV Colour Space.

# The conversion of RGB to an HSV image is shown in fig 2.2

# 

# Fig 2.2 converting the RGB image to HSV Image.

# Colour space threshold segmentation is required after conversion to the HSV color space. Figure 2.3 shows the colour space threshold segmentation step diagram

# 

# Fig 2.3 color space threshold segmentation step diagram.

# Common traffic signs mainly include red, yellow and blue colors. In order to meet the target requirements of real-time color segmentation, it is necessary to determine the corresponding threshold range. Through multiple test experiments, the three-channel threshold segmentation ranges of three colors are obtained on the premise of ensuring good segmentation effects, as shown in the table 2.1.

# 

# Table 2.1 HSV color space threshold segmentation ranges.

# /In the process of threshold segmentation, the pixels within the set threshold range are set to white, otherwise they are set to black, and the image is completely binarized. Since the traffic sign in the original picture is red, the obtained threshold coarse segmentation image only displays red. Figure 2.4 presents the threshold rough segmentation image

# 

# Fig 2.4 The threshold rough segmentation image.

# CHAPTER -3

# PROPROSED WORK

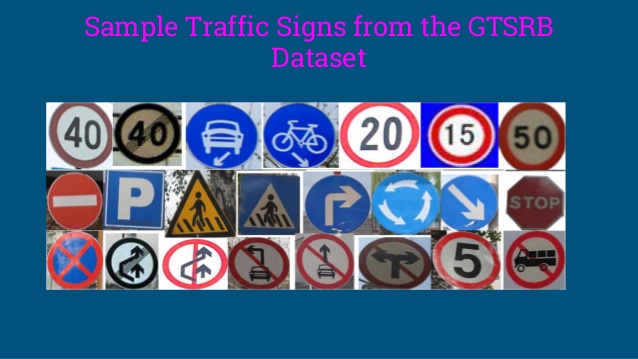
# The contribution of this project will be two folds; one is to develop a new database for Arabic Traffic and Road Signs and the other is develop and design a deep CNN architecture for Arabic Traffic sign recognition. Fig. 1 shows the high level view of the system. The collected data set is given as an input to the proposed CNN architecture for training, validation and testing. The detailed explanation of the CNN architecture is provided in the next section. Once the CNN is trained, it is ready to be used for classifying new images which were not part of the collected dataset. The GTSRB system depends on a group of standard Arabic Traffic signs. In recent years, number of authors done work in this field but to the author’s knowledge, this is the first time a complete database is developed for GermanTraffic. A Deep CNN architecture is also proposed for traffic sign recognition Generally, CNNs consist of multiple hidden layers between the input and output layers . The design of the proposed CNN is implemented using Python

# 

# Fig 3.1 Block diagram of supervised learning.

# 3.1 Experimental Data:

The dataset we have used for this project is the GTSRB (German traffic sign recognition benchmark). It contains a Train folder that has traffic sign images in 43 different classes, a Test folder that has over 12,000 images for testing purposes. A test.csv file that contains the path of the test images along with their respective classes.

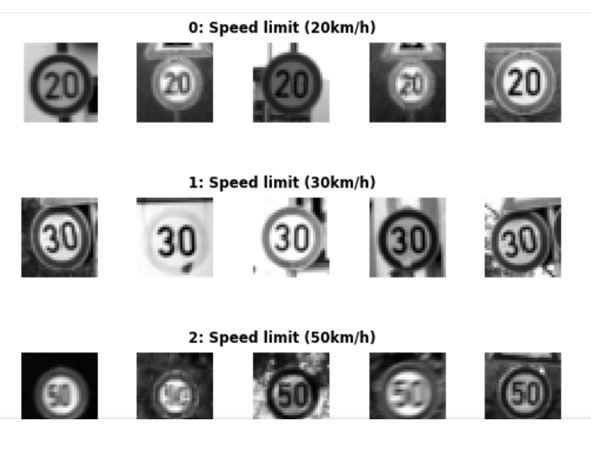


# Fig 3.2 Sample Images in GTSRB Dataset

# 3.2 Preprocessing:

# The images were RGB images with different dimensions which led to the need of pre-processing the images before inputting them to the CNN network. The images are transformed into Greyscale images and the dimension is also reduced to 30x30 pixels. Moreover, the total number of output classes is 24 classes each represents an Arabic traffic sign so all the images are carefully labeled and placed in their corresponding folders. The number of images per class differs from one class to another Fig. 3.3 shows a sample of each Traffic Sign after the preprocessing.

* Images are 30 (width) x 30 (height) x 3 (RGB color channels)
* Training set is composed of 34799 images
* Validation set is composed of 4410 images
* Test set is composed of 12630 images
* There are 43 classes (e.g. Speed Limit 20km/h, No entry, Bumpy road, etc.)



# Fig 3.3 Sample images after preprocessing.

### **Training**

Due to its complexity, Deep learning models require a large number of instances to avoid overfitting. However, for the majority of real-life problems, data is not abundant. In fact, few are the situations where there is an abundance of data, such as the ImageNet dataset. To overcome this issue, one could rely on two techniques: data augmentation and transfer learning. In this work, we made use of both techniques and we describe below

### **Data augmentation**

Data augmentation consists of increasing the training samples by transforming the images without losing semantic information. In this work, we applied three transformations to the training samples: rotation, horizontal flip, and scaling. Such transformations preserve the images and would not prevent a physician from interpreting the images.

## Machine Learning Models

**Machine learning** is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. **Machine learning** focuses on the development of computer programs that can access data and use it learn for themselves.

Machine Learning Models are broadly Classified into 3 categories. They are –

1. ***Supervised Learning:*** This algorithm consists of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.
2. ***Unsupervised Learning****:* In this algorithm, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: Apriori algorithm, K-means.
3. ***Reinforcement Learning:*** Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process

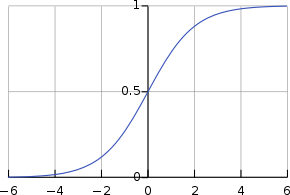
**List of Common Machine Learning Algorithms**

* 1. Logistic Regression
  2. kNN
  3. Naive Bayes
  4. Decision Tree
  5. Support Vector Machines
  6. XG Boost

### **3.3.1 Logistic Regression**

This is the most popular ML algorithm for binary classification of the data-points. With the help of [**logistic regression**](https://data-flair.training/blogs/logistic-regression-in-r/), we obtain a categorical classification that results in the output belonging to one of the two classes. For example, predicting whether the price of oil would increase or not based on several predictor variables is an example of logistic regression.

Logistic Regression has two components – **Hypothesis and Sigmoid Curve**. Based on this hypothesis, one can derive the resultant likelihood of the event. Data obtained from the hypothesis is then fit into the log function that forms the S-shaped curve called ‘sigmoid’. Through this log function, one can determine the category to which the output data belongs to. The sigmoid S-shaped curve is visualized as follows –



**Fig 3. 4 Logistic Function graph**

The above-generated graph is a result of this logistic equation –

1

(1 + 𝑒−𝑥)

In the above equation, e is the base of the natural log and the S-shaped curve that we obtain is between 0 and 1. We write the equation for logistic regression as follows

𝑒(𝑏0+𝑏1𝑥)

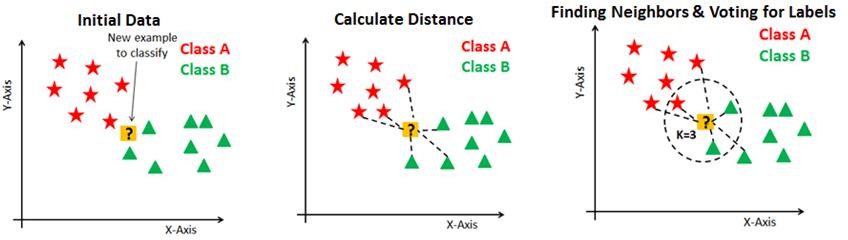
𝑦 =

(1 + 𝑒(𝑏0+𝑏1𝑥))

### **K-Nearest Neighborhood Classifier**

KNN is one of the many supervised machine learning algorithms that we use for data mining as well as machine learning. Based on the similar data, this classifier then learns the patterns present within. It is a non-parametric and a lazy learning algorithm. By non- parametric, we mean that the assumption for underlying data distribution does not hold valid. In lazy loading, there is no requirement for training data points for generating models.

The training data is utilized in testing phase causing the testing phase slower and costlier as compared with the training phase.

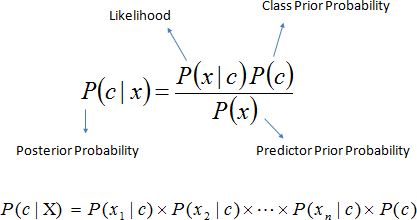


**Fig 3. 5 k-NN Algorithm.**

### **Naïve Bayes Classifier**

This is a classification technique based on *Bayes’ theorem* with an assumption of independence between predictors. In simple terms, a ***Naive Bayes classifier*** assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability.

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability **P(c|x)** from **P(c)**, **P(x)** and **P(x|c)**. Figure 3.8 represents Bayes Eaquation.



**Fig 3. 6 Equation for Posterior Probability estimation.**

### **Decision Tree**

***Decision Trees*** facilitate prediction as well as classification. Using the decision trees, one can make decisions with a given set of input. Let us understand decision trees with the following example –

Let us assume that you want to go to the market to purchase a shampoo. First, you will analyse if you really do require shampoo. If you run out of it, then you will have to buy it from the market. Furthermore, you will look outside and assess the weather. That is, if it is raining, then you will not go and if it is not, you will. We can visualize this scenario intuitively with the following visualization.



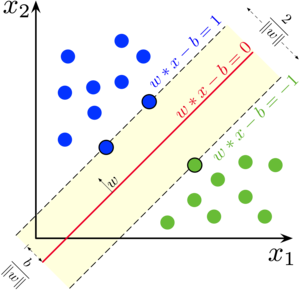
**Fig 3. 7 Decision Tree Example.**

With the same principle, we can construct a hierarchical tree to obtain our output through several decisions. There are two procedures towards building a decision tree – Induction and Pruning. In Induction, we build the decision tree and in pruning, we simplify the tree by removing several complexities.

### **Support Vector Machine**

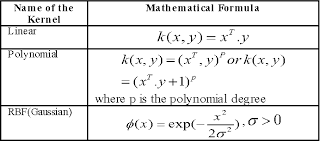
Support Vector Machines are a type of supervised machine learning algorithm that provides analysis of data for classification and regression analysis. While they can be used for regression, SVM is mostly used for classification. We carry out plotting in the n-dimensional space. Value of each feature is also the value of the specific coordinate. Then, we find the ideal hyperplane that differentiates between the two classes.

These support vectors are the coordinate representations of individual observation. It is a frontier method for segregating the two classes. The basic principle behind the working of Support vector machines is simple – Create a hyperplane that separates the dataset into classes.



**Fig 3. 8 Support Vector Machine.**

**Kernel Functions used in SVM:**



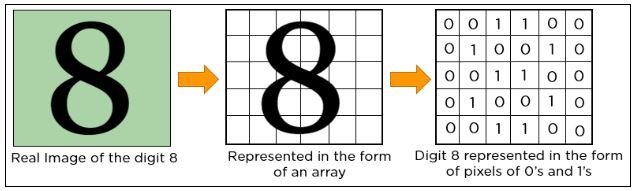
## Deep Learning Models

In this Section First, we discuss about Convolutional Neural Network which performs both Feature Extraction and Classification followed by Transfer Learning scheme and Some popular pre-training Models that we use for Feature extraction. Finally, we introduce a Deep Neural Network Architecture that we used for Diabetic Retinopathy Detection and Severity Prediction.

### **Convolutional Neural Network**

Yann LeCun, director of Facebook’s AI Research Group, is the pioneer of convolutional neural networks. He built the first convolutional neural network called LeNet in 1988. LeNet was used for character recognition tasks like reading zip codes and digits.

A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It’s also known as a ConvNet. A convolutional neural network is used to detect and classify objects in an image. In CNN, every image is represented in the form of an array of pixel values.

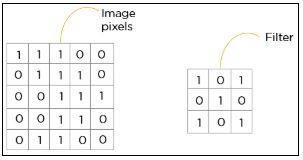


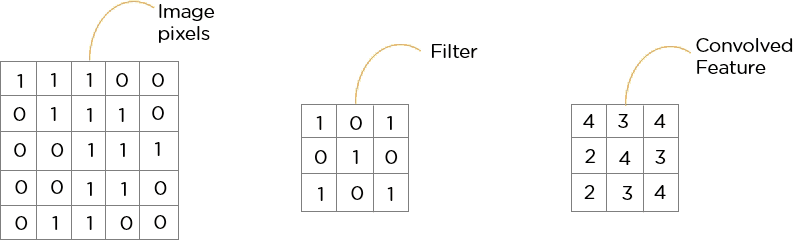
**Fig 3. 9 Pixel representation of Image.**

A convolution neural network has multiple hidden layers that help in extracting information from an image. The four important layers in CNN are:

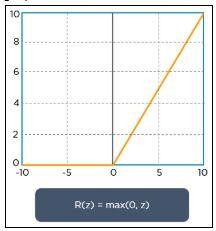
1. Convolution layer
2. ReLU layer
3. Pooling layer
4. Fully connected layer

***Convolution Layer:*** This is the first step in the process of extracting valuable features from an image. A convolution layer has several filters that perform the convolution operation. Every image is considered as a matrix of pixel values. Consider the following 5x5 image whose pixel values are either 0 or 1. There’s also a filter matrix with a dimension of 3x3. Slide the filter matrix over the image and compute the dot product to get the convolved feature matrix.



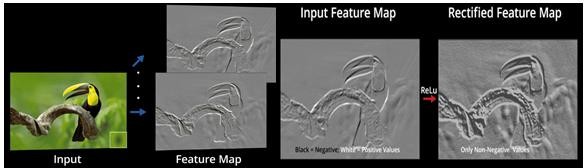


**Fig 3. 10 Convolution Operation.**

***ReLU Layer:*** ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer. ReLU performs an element- wise operation and sets all the negative pixels to 0. It introduces non-linearity to the network, and the generated output is a rectified feature map. Below is the graph of a ReLU function:

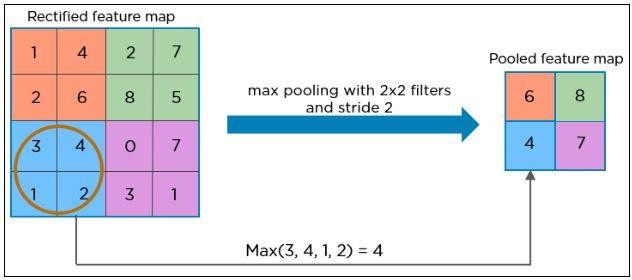
**Fig 3. 11 ReLU Operation**

The original image is scanned with multiple convolution and ReLU layers for locating the features.



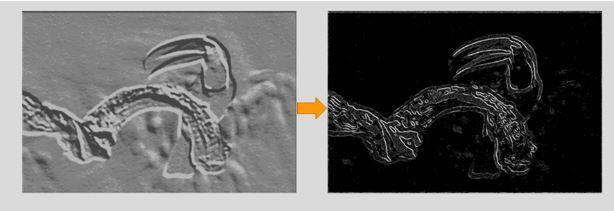
**Fig 3. 12 Feature Map Visualization after Convolution and ReLU Operations.**

***Pooling Layer:*** Pooling is a down-sampling operation that reduces the dimensionality of the feature map. The rectified feature map now goes through a pooling layer to generate a pooled feature map.



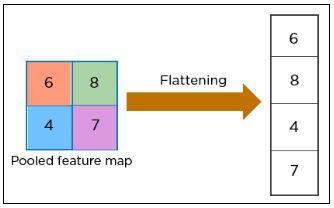
**Fig 3. 13 Pooling Operation.**

The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes, and beak.



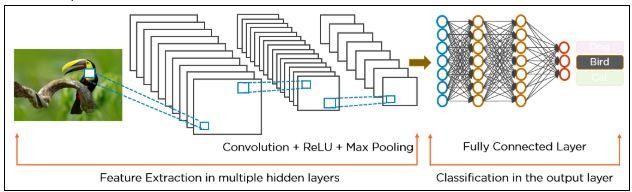
**Fig 3. 14 Feature Map Visualization after Pooling Operation.**

The next step in the process is called ***flattening***. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.



**Fig 3. 15 Flattening.**

The flattened matrix is fed as input to the ***fully connected layer*** to classify the image.



**Fig 3. 16 Architecture of Convolutional Neural Network.**

The downside of the Convolutional Neural Networks (CNN) is that they need enormous amounts of data for training, which is usually scarce for most of the real-time applications. This problem can be addressed by the introduction of Transfer Learning where the knowledge gained by a deep learning model can be transferred to other models without any training. This is achieved using pre-trained models.

### **Transfer Learning**

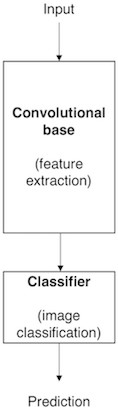
Transfer learning is a popular method in computer vision because it allows to **build accurate models in a timesaving way** (Rawat & Wang 2017) [25]. With transfer learning, instead of starting the learning process from scratch, you start from patterns thathave been learned when solving a different problem. This way you leverage previous learnings and avoid starting from scratch.

In computer vision, transfer learning is usually expressed using **pre-trained models**. A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem like the one that we want to solve. Accordingly, due to the computational cost of training such models, it is common practice to import and use models from published literature (e.g. VGG, Inception, ResNet). A comprehensive review of pre-trained models’ performance on computer vision problems using data from the ImageNet (Deng et al. 2009) [26] challenge is presented by Canziani et al. (2016). [27]

Several pre-trained models used in transfer learning are based on large **convolutional neural networks (CNN)** (Voulodimos et al. 2018) [28]. In general, CNN was shown to excel in a wide range of computer vision tasks (Bengio 2009) [29]. Its high performance and its easiness in training are two of the main factors driving the popularity of CNN over the last years. A typical CNN has two parts:

1. **Convolutional base**, which is composed by a stack of convolutional and pooling layers. The main goal of the convolutional base is to generate features from the image. For an intuitive explanation of convolutional and pooling layers.
2. **Classifier**, which is usually composed by fully connected layers. The main goal of the classifier is to classify the image based on the detected features. A fully connected layer is a layer whose neurons have full connections to all activation in the previous layer.

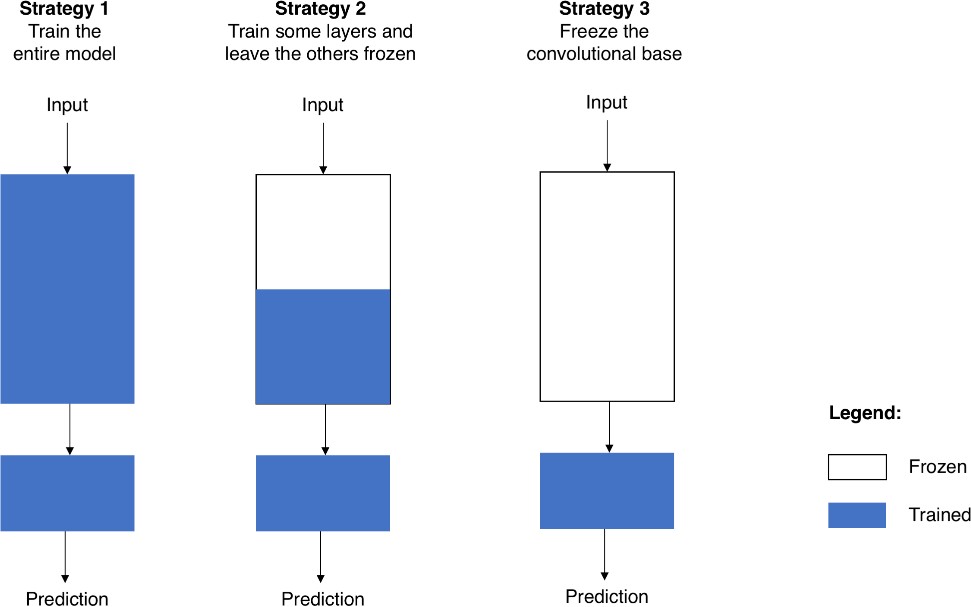
One important aspect of these deep learning models is that they can automatically learn **hierarchical feature representations**. This means that features computed by the first layer are general and can be reused in different problem domains, while features computed by the last layer are specific and depend on the chosen dataset and task. According to Yosinski et al. (2014) [30], ‘*if first-layer features are general and last-layer features are specific, then there must be a transition from general to specific somewhere in the network’*. As a result, the convolutional base of our CNN — especially its lower layers (those who are closer to the inputs) — refer to general features, whereas the classifier part, and some of the higher layers of the convolutional base, refer to specialized features.



**Fig 3. 17 Architecture of a model based on CNN.**

**Repurposing a pre-trained model:** When you’re repurposing a pre-trained model for your own needs, you start by removing the original classifier, then you add a new classifier that fits your purposes, and finally you must fine-tune your model according to one of three strategies**:**

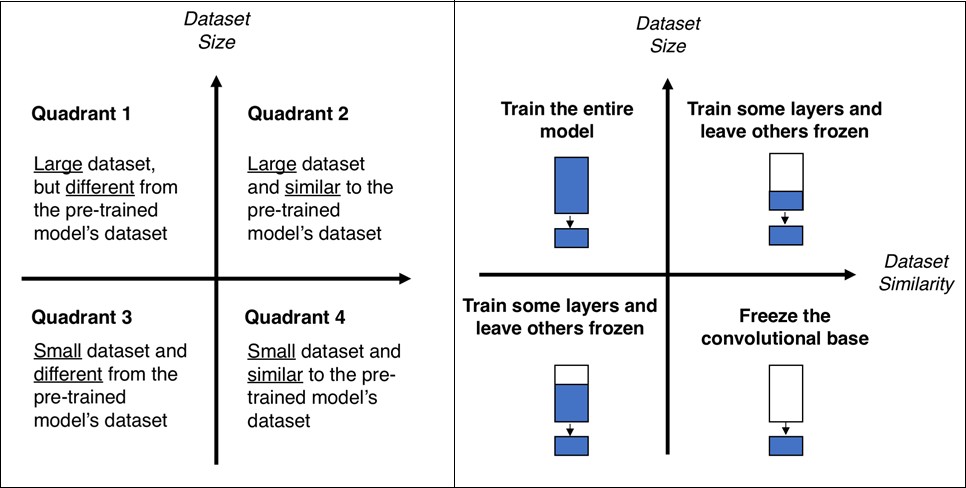
1. **Train the entire model.** In this case, you use the architecture of the pre-trained model and train it according to your dataset. You’re learning the model from scratch, so you’ll need a large dataset (and a lot of computational power).
2. **Train some layers and leave the others frozen.** As you remember, lower layers refer to general features (problem independent), while higher layers refer to specific features (problem dependent). Here, we play with that dichotomy by choosing how much we want to adjust the weights of the network (a frozen layer does not change during training).
3. **Freeze the convolutional base.** This case corresponds to an extreme situation of the train/freeze trade-off. The main idea is to keep the convolutional base in its original form and then use its outputs to feed the classifier. You’re using the pre- trained model as a fixed feature extraction mechanism, which can be useful if you’re short on computational power, your dataset is small, and/or pre-trained model solves a problem very similar to the one you want to solve.



**Fig 3. 18 Fine-tuning strategies.**

**Transfer Learning Process:** From a practical perspective, the entire transfer learning process can be summarised as follows:

1. **Select a pre-trained model**. From the wide range of pre-trained models that are available, you pick one that looks suitable for your problem. For example, if you’re using Keras, you immediately have access to a set of models, such as VGG (Simonyan & Zisserman 2014) [21], InceptionV3 (Szegedy et al. 2015) [31], and ResNet5 (He et al. 2015) [32].
2. **Classify your problem according to the Size-Similarity Matrix.** In Figure 3.21(L) you have ‘The Matrix’ that controls your choices. This matrix classifies your computer vision problem considering the size of your dataset and its similarity to the dataset in which your pre-trained model was trained. As a rule of thumb, consider that your dataset is small if it has less than 1000 images per class. Regarding dataset similarity, let common sense prevail. For example, if your task is to identify cats and dogs, ImageNet would be a similar dataset because it has images of cats and dogs. However, if your task is to identify cancer cells, ImageNet can’t be considered a similar dataset.
3. **Fine-tune your model.** Here you can use the Size-Similarity Matrix to guide your choice and then refer to the three options we mentioned before about repurposing a pre-trained model. Figure 3.21 (R) provides a visual summary of the text.



**Fig 3. 19 Size-Similarity matrix (L) & decision map for fine-tuning pre-trained models (R).**

**Classifiers on top of deep convolutional neural networks:** As mentioned before, models for image classification that result from a transfer learning approach based on pre-trained convolutional neural networks are usually composed of two parts**:**

* 1. **Convolutional base**, which performs feature extraction.
  2. **Classifier**, which classifies the input image based on the features extracted by the convolutional base.

Since in this section we focus on the classifier part, we must start by saying that different approaches can be followed to build the classifier. Some of the most popular are:

**Fully connected layers:** For image classification problems, the standard approach is to use a stack of fully connected layers followed by a softmax activated layer (Krizhevsky et al. 2012, Simonyan & Zisserman 2014, Zeiler & Fergus 2014) [33] [21] [34]. The softmax layer outputs the probability distribution over each possible class label and then we just need to classify the image according to the most probable class.

**Global average pooling:** A different approach, based on global average pooling, is proposed by Lin et al. (2013) [35]. In this approach, instead of adding fully connected layers on top of the convolutional base, we add a global average pooling layer and feed its output directly into the softmax activated layer. Lin et al. (2013) [35] provides a detailed discussion on the advantages and disadvantages of this approach.

**Linear support vector machines:** Linear support vector machines (SVM) is another possible approach. According to Tang (2013), we can improve classification accuracy by training a linear SVM classifier on the features extracted by the convolutional base.

**3.4.3 Pre-trained Models**

**VGG 16 [21] —** VGG16 is a pre-trained model trained on 14 million image dataset belonging to 1000 different classes in ILSVR (ImageNet) challenge. In this architecture, 2X2 filters with stride 1 are used for convolution operation and 2X2 filters with stride 2, same padding for max pooling operation across the network. At the end of architecture two fully connected dense layers of 4096 neurons are connected followed by softmax layer. We extracted features from two fully connected layers separately and used them for our experiments.

**Neural Architecture Search Network [23] —** Neural Architecture Search Network (NASNet) is a special kind of Deep Convolutional Neural Network which searches for a better architectural building block on small datasets like CIFAR10 and transfer it to larger datasets like ImageNet. It has a better regularization mechanism called ScheduledDropPath which significantly improves generalization. We collected the features from global average pooling layer and used them our experimental studies.

**Xception [22] —** Xception is a Deep ConvNet Architecture that supports Depthwise Separable Con- volution Operations and outperformed ResNet and Inception V3 in ILSVR challenge. We trained a DNN using output of global average pooling layer and it exceptionally performed well for Diabetic Retinopathy Identification task than any other models.

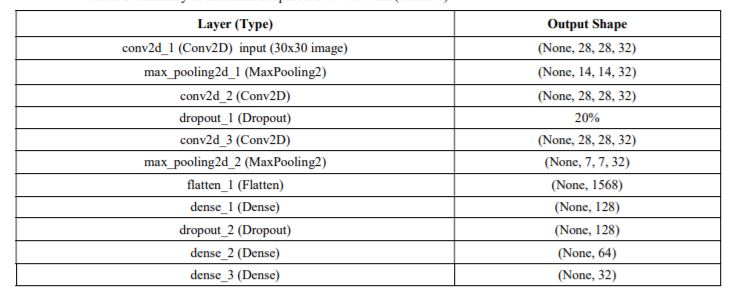
**Inception ResNet V2 [36] —** Inception ResNet V2 also referred as Inception V4, combine two different architectures called Inception V3 and ResNet152. It has both Inception and residual connections which boost the performance of model. In this also we obtained the features from global average pooling and trained proposed Deep Neural Network. It outperformed other models for the task of Diabetic Retinopathy Severity Prediction.

* + 1. **Deep Convolutional Neural Network Architecture:**

The proper choice of different design hyper-parameters, such as non-linearity and pooling variants, directly affects the performance of the network. Since there is no clear guidance on how to choose the CNN hyper-parameters, many researchers tend to use trial and error experimentation in order to discover good settings [15]. We analyzed previous research studies that have been carried in the area of deep learning generally in order to choose the hyper-parameters setting that is more likely suitable for this research. So by analyzing the work done in [16-18], the hyper-parameters setting in this work is as follows.

Pooling Layer: It helps in reducing the amount of computations needed in the training process by reducing the dimension of the image, and therefore the overfitting problem is reduced. The max-pooling technique was used since the convergence rate is faster as compared to other subsampling techniques, and thus has a better generalization performance. The maximum pixel value in a non-overlapping region, equals to the window of the pooling, is the output of this layer; this is beneficial in creating position invariance.

Non-Linearity: Since the relation between the images and their classes is not linear, introducing non-linearity in the CNN is needed. This is achieved by using non-linear activation so that the construction of the non-linear relation between the images and their classes is possible by the CNN. The Rectified Linear Unit (ReLU) is a widely used activation function in CNN. ReLU has the advantage that the CNN trains faster as compared to other functions. One of the ReLU variants was used which is the leaky ReLU since it overcomes the problem of dead neurons that is faced if the original ReLU is used. Basically, leaky ReLU does not output zero when the input values are less than zero, instead it outputs negative value. So after each convolution layer, and before the pooling layer, we added a leaky ReLU activation function. Dense Layer Activation Function and Loss Function: As mentioned above, we used the leaky ReLU activation function multiple times in the proposed network. However, the output of the fully connected layer, which is one dimensional vector, is passed to a softmax activation function to predict the label of that particular input. The input vector is transformed to a vector of the same size but the values range from zero to one only and the summation is always equals to 1. Softmax function outputs a probability distribution that is then converted to one-hot encoding vector. So the element that has the largest portion of probability distribution will have the value one in the one-hot vector and all other elements will have the value zero. To estimate the loss of softmax, the categorical cross-entropy function is used. Basically, categorical cross-entropy is used to measure the difference between the output of the softmax function and the one-hot encoding of the actual class. It is used as a part of gradient descent to evaluate CNN performance as an error measure between the true distribution and the predicted one. Softmax function and categorical cross-entropy are widely used in computer vision tasks when multiple classes are involved. After setting the needed hyper-parameters, the network was compiled using the Adam optimizer. The need of using optimization algorithms is the nature that CNN weights and biases are set automatically and they are of great importance in the field of machine learning. They work on optimizing a given function; whether maximizing or minimizing it with respect to its parameters. Since the loss function in CNN is differentiable with respect to its parameters, gradient descent based algorithms are often used; first order partial derivatives are also fast to compute. Adam is an optimization algorithm that is used to update network weights. It combines the advantages of other classical stochastic gradient descent which are Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). It is computationally efficient, requires less memory and has straightforward implementations. Given different parameters, this algorithm computes each parameter’s adaptive learning rate by estimating the gradient’s first and second moments [19]. Table 1 shows the summary of the parameters in each layer as well as the total parameters in the proposed network. This CNN network will be referred to as Model 1 in the results section.



**Table 2. Summary of the Initial Proposed CNN Network (Model 1)**

# CHAPTER – 4

# EXPERIMENTAL RESULTS

# 4.1 Results and Discussions:

# The improved optimized CNN network layers are shown in Table 3. As observed in the optimized CNN architecture, the number of layers have been reduced without compromising the accuracy of the network. This CNN will be referred to as Model 2. Table 3 shows the experimental results performed on the proposed CNN architecture. The experiments were performed on epochs size of 15. For each epoch, the batch sizes of 50,100, 200 and 400 are used. The table for the experiment shows four columns with Validation Accuracy (Val. Acc), Validation Loss (Val. Loss), Test Accuracy (Test Acc) and finally Test Loss (Test Loss). Different design parameters were changed and the effect of the changes was tested by evaluating accuracy. A total of 16 different experiments were performed on the CNN architecture in order to identify an optimized design. Note that all the preprocessing steps remain the same as described above. The proposed CNN models are shown in the Table 1 and Table 3. Experimental results show that the number of epochs and number of batch size directly affects the test accuracies achieved for the models. The number of epochs is directly proportional with test accuracy; increasing epoch size increases the test accuracy. While the opposite is true for batch size as it is observed that increasing batch size decreases test accuracy. Overall, the test accuracy obtained for the improved CNN network-model 2 is better than those achieved for the initial CNN-model 1. In fact, model 2 results for epoch 150 achieved 100% accuracy for all batch sizes. The result is better than those reported in recent literature, however, it is pointed out here that the CNN is applied for Arabic Traffic Sign dataset and no recent literature was found targeting Arabic Traffic sign detection and recognition. No relationship was found for validation accuracy for both models in relation to epoch size and batch size.

# 

# Table 3 Summary of the Improved CNN Network with less number of layers (Model 2)

# 4.2 Dataset:

# 

# Fig 4.1 graph representation of Dataset in graph

There are a number of challenges in the GTSRB dataset, **the first being that images are low resolution,** and worse, **have poor contrast** (as seen in **Figure 2** above). These images are pixelated, and in some cases, it’s extremely challenging, if not impossible, for the human eye and brain to recognize the sign.

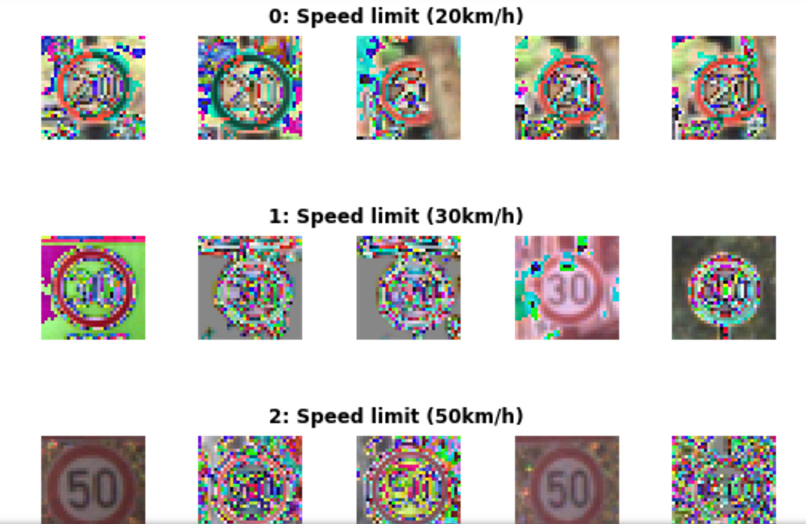
The second challenge with the dataset is **handling class skew:**

The top class *(Speed limit 50km/h)* has over 2,000 examples while the least represented class *(Speed limit 20km/h)* has under 200 examples — that’s an order of magnitude difference!

**In order to successfully train an accurate traffic sign classifier we’ll need to devise an experiment that can:**

* Preprocess our input images to improve contrast.
* Account for class label skew.

**4.2 Image Normalization:** We center the distribution of the image dataset by subtracting each image by the dataset mean and divide by its standard deviation. This helps our model treating images uniformly. The resulting images look as follows:



**Fig 4.2 Normalized images**

# 4.3 Dropout:

In order to improve the model reliability, we turned to dropout, which is a form of regularization where weights are kept with a probability p: the unkept weights are thus “dropped”. This prevents the model from overfitting. Dropout was introduced by Geoffrey Hinton, a pioneer in the deep learning space. His group’s [paper](http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf) on this topic is a must read to better understand the motivations behind the authors. There’s also a fascinating parallel with biology and evolution.  
In the project, the authors apply varying degrees of dropout, depending on the type of layer. I therefore decided to adopt a similar approach, defining two levels of dropout, one for convolutional layers, the other for fully connected layers:

P-conv: probability of keeping weight in convolutional layer  
p-fc: probability of keeping weight in fully connected layer

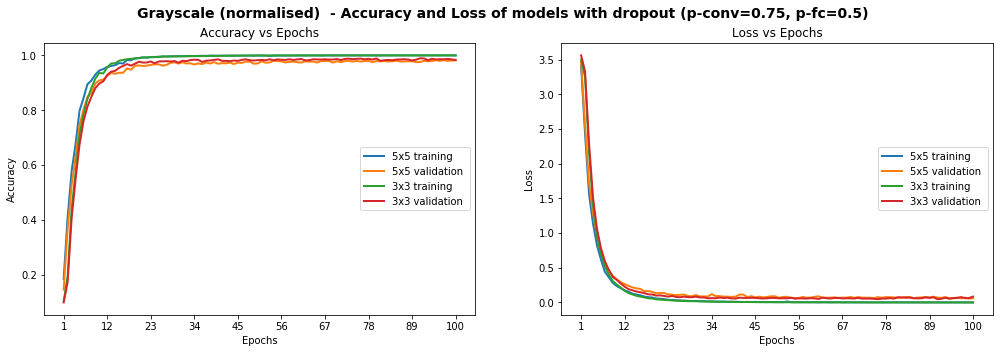
Moreover, the authors gradually adopted more aggressive (i.e. lower) values of dropout as they go deeper in the network. Therefore I also decided:

P-conv >= p-fc

That is, we will keep weights with a greater than or equal probability in the convolutional than fully connected layers. The way to reason about this is that we treat the network as a funnel and therefore want to gradually tighten it as we move deeper into the layers: we don’t want to discard too much information at the start as some of it would be extremely valuable. Besides, as we apply [MaxPooling](https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks) in the convolutional layers, we are already losing a bit of information.

We tried different parameters but ultimately settled on p-conv=0.75 and p-fc=0.5, which enabled us to achieve a test set accuracy of 97.55% on normalized grayscale images with the 3x3 model. Interestingly, we achieved over 98.3% accuracy on the validation set:

Training EdLeNet\_Norm\_Grayscale\_3x3\_Dropout\_0.50 [epochs=100, batch\_size=512]...  
  
[1] total=5.222s | train: time=3.139s, loss=3.4993, acc=0.1047 | val: time=2.083s, loss=3.5613, acc=0.1007  
[10] total=5.190s | train: time=3.122s, loss=0.2589, acc=0.9360 | val: time=2.067s, loss=0.3260, acc=0.8973  
...  
[90] total=5.193s | train: time=3.120s, loss=0.0006, acc=0.9999 | val: time=2.074s, loss=0.0747, acc=0.9841  
[100] total=5.191s | train: time=3.123s, loss=0.0004, acc=1.0000 | val: time=2.068s, loss=0.0849, acc=0.9832  
Model ./models/EdLeNet\_Norm\_Grayscale\_3x3\_Dropout\_0.50.chkpt saved  
[EdLeNet\_Norm\_Grayscale\_3x3\_Dropout\_0.50 - Test Set] time=0.686s, loss=0.1119, acc=0.9755



**Fig 4.3Models Performance on Grayscale Normalised Images, After The Introduction Of Dropout**

The graphs above show that the model is smooth, unlike some of the graphs higher up. We have already achieved the objective of scoring over 93% accuracy on the test set, but can we do better? Remember that some of the images were blurry and the distribution of images per class was very uneven. We explore below additional techniques we used to tackle each point.

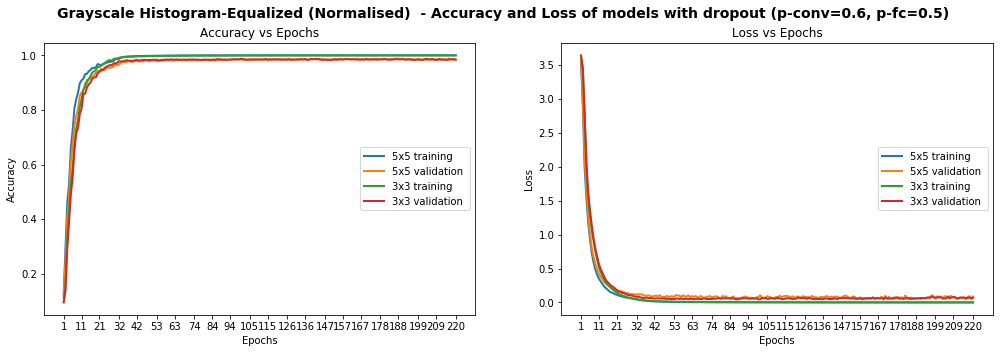
# 4.4 Histogram Equalization

Histogram Equalization is a computer vision technique used to increase the contrast in images. As some of our images suffer from low contrast (blurry, dark), we will improve visibility by applying OpenCV’s Contrast Limiting Adaptive Histogram Equalization (aka CLAHE) function.

We once again try various configurations, and find the best results, with **test accuracy of 97.75%**, on the 3x3 model using the following dropout values: p-conv=0.6, p-fc=0.5 .

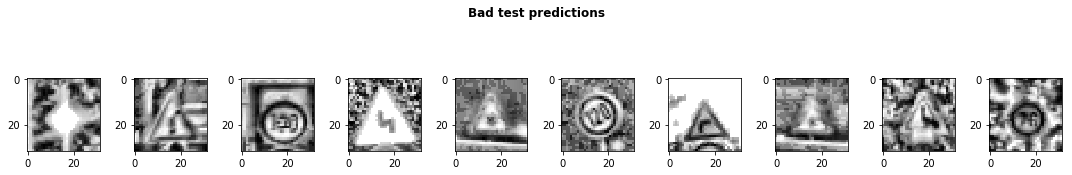
Training EdLeNet\_Grayscale\_CLAHE\_Norm\_Take-2\_3x3\_Dropout\_0.50 [epochs=500, batch\_size=512]...[1] total=5.194s | train: time=3.137s, loss=3.6254, acc=0.0662 | val: time=2.058s, loss=3.6405, acc=0.0655  
[10] total=5.155s | train: time=3.115s, loss=0.8645, acc=0.7121 | val: time=2.040s, loss=0.9159, acc=0.6819  
...  
[480] total=5.149s | train: time=3.106s, loss=0.0009, acc=0.9998 | val: time=2.042s, loss=0.0355, acc=0.9884  
[490] total=5.148s | train: time=3.106s, loss=0.0007, acc=0.9998 | val: time=2.042s, loss=0.0390, acc=0.9884  
[500] total=5.148s | train: time=3.104s, loss=0.0006, acc=0.9999 | val: time=2.044s, loss=0.0420, acc=0.9862  
Model ./models/EdLeNet\_Grayscale\_CLAHE\_Norm\_Take-2\_3x3\_Dropout\_0.50.chkpt saved  
[EdLeNet\_Grayscale\_CLAHE\_Norm\_Take-2\_3x3\_Dropout\_0.50 - Test Set] time=0.675s, loss=0.0890, acc=0.9775

We show below graphs of previous runs where we tested the 5x5 model as well, over 220 epochs. We can see a much smoother curve here, reinforcing our intuition that the model we have is more stable.



**Fig 4.4 Models Performance On Grayscale Equalized Images, With Dropout**

We identified 269 images that are model could not identify correctly. We display 10 of them below, chosen randomly, to conjecture why the model was wrong.



**Fig 4.5 Sample of 10 images where our model got the predictions wrong**

Some of the images are very blurry, despite our histogram equalization, while others seem distorted. We probably don’t have enough examples of such images in our test set for our model’s predictions to improve. Additionally, while 97.75% test accuracy is very good, we still one more ace up our sleeve: data augmentation.

# 4.5 Data Augmentation

We observed earlier that the data presented glaring imbalance across the 43 classes. Yet it does not seem to be a crippling problem as we are able to reach very high accuracy despite the class imbalance. We also noticed that some images in the test set are distorted. We are therefore going to use data augmentation techniques in an attempt to:

1. Extend dataset and provide additional pictures in different lighting settings and orientations
2. Improve model’s ability to become more generic
3. Improve test and validation accuracy, especially on distorted images

We use a nifty library called [imgaug](https://github.com/aleju/imgaug) to create our augmentations. We mainly apply affine transformations to augment the images. Our code looks as follows:

def augment\_imgs(imgs, p):  
 """  
 Performs a set of augmentations with a probability p  
 """  
 augs = iaa.SomeOf((2, 4),  
 [  
 iaa.Crop(px=(0, 4)), # crop images from each side by 0 to 4px (randomly chosen)  
 iaa.Affine(scale={"x": (0.8, 1.2), "y": (0.8, 1.2)}),  
 iaa.Affine(translate\_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)}),  
 iaa.Affine(rotate=(-45, 45)), # rotate by -45 to +45 degrees)  
 iaa.Affine(shear=(-10, 10)) # shear by -10 to +10 degrees  
 ])   
 seq = iaa.Sequential([iaa.Sometimes(p, augs)])  
   
 return seq.augment\_images(imgs)

While the class imbalance probably causes some bias in the model, we have decided not to address it at this stage as it would cause our dataset to swell significantly and lengthen our training time (we don’t have a lot of time to spend on training at this stage). Instead, we decided to augment each class by 10%. Our new dataset looks as 

**Fig 4.6Sample Of Augmented Images**

The distribution of images does not change significantly of course, but we do apply grayscale, histogram equalization and normalisation pre-processing steps to our images. We train for 2000 epochs with dropout (p-conv=0.6, p-fc=0.5) and achieve**97.86% accuracy on the test set:**

[EdLeNet] Building neural network [conv layers=3, conv filter size=3, conv start depth=32, fc layers=2]  
Training EdLeNet\_Augs\_Grayscale\_CLAHE\_Norm\_Take4\_Bis\_3x3\_Dropout\_0.50 [epochs=2000, batch\_size=512]...  
  
[1] total=5.824s | train: time=3.594s, loss=3.6283, acc=0.0797 | val: time=2.231s, loss=3.6463, acc=0.0687  
...  
[1970] total=5.627s | train: time=3.408s, loss=0.0525, acc=0.9870 | val: time=2.219s, loss=0.0315, acc=0.9914  
[1980] total=5.627s | train: time=3.409s, loss=0.0530, acc=0.9862 | val: time=2.218s, loss=0.0309, acc=0.9902  
[1990] total=5.628s | train: time=3.412s, loss=0.0521, acc=0.9869 | val: time=2.216s, loss=0.0302, acc=0.9900  
[2000] total=5.632s | train: time=3.415s, loss=0.0521, acc=0.9869 | val: time=2.217s, loss=0.0311, acc=0.9902  
Model ./models/EdLeNet\_Augs\_Grayscale\_CLAHE\_Norm\_Take4\_Bis\_3x3\_Dropout\_0.50.chkpt saved[EdLeNet\_Augs\_Grayscale\_CLAHE\_Norm\_Take4\_Bis\_3x3\_Dropout\_0.50 - Test Set] time=0.678s, loss=0.0842, acc=0.9786

**4.5 Testing On New Images**

We decided to test our model on new images as well, to make sure that it’s indeed generalised to more than the traffic signs in our original dataset. We therefore downloaded five new images and submitted them to our model for predictions.



**Fig 4.7 Download 5 new traffic signs — color**

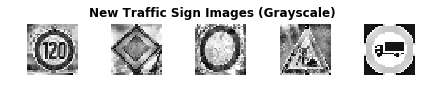
The ground truth for the images is as follows:

['Speed limit (120km/h)',  
 'Priority road',  
 'No vehicles',  
 'Road work',  
 'Vehicles over 3.5 metric tons prohibited']

The Images were chosen because of the following:

* They represent different traffic signs that we currently classify
* They vary in shape and color
* They are under different lighting conditions (the 4th one has sunlight reflection)
* They are under different orientations (the 3rd one is slanted)
* They have different background
* The last image is actually a design, not a real picture, and we wanted to test the model against it
* Some of them are in under-represented classes

The first step we took was to apply the same CLAHE to those new images, resulting in the following:



**Fig 4.8 Grayscale images**

We achieve perfect accuracy of 100% on the new images. On the original test set, we achieved 97.86% accuracy. We could explore blurring/distorting our new images or modifying contrast to see how the model handles those changes in the future.

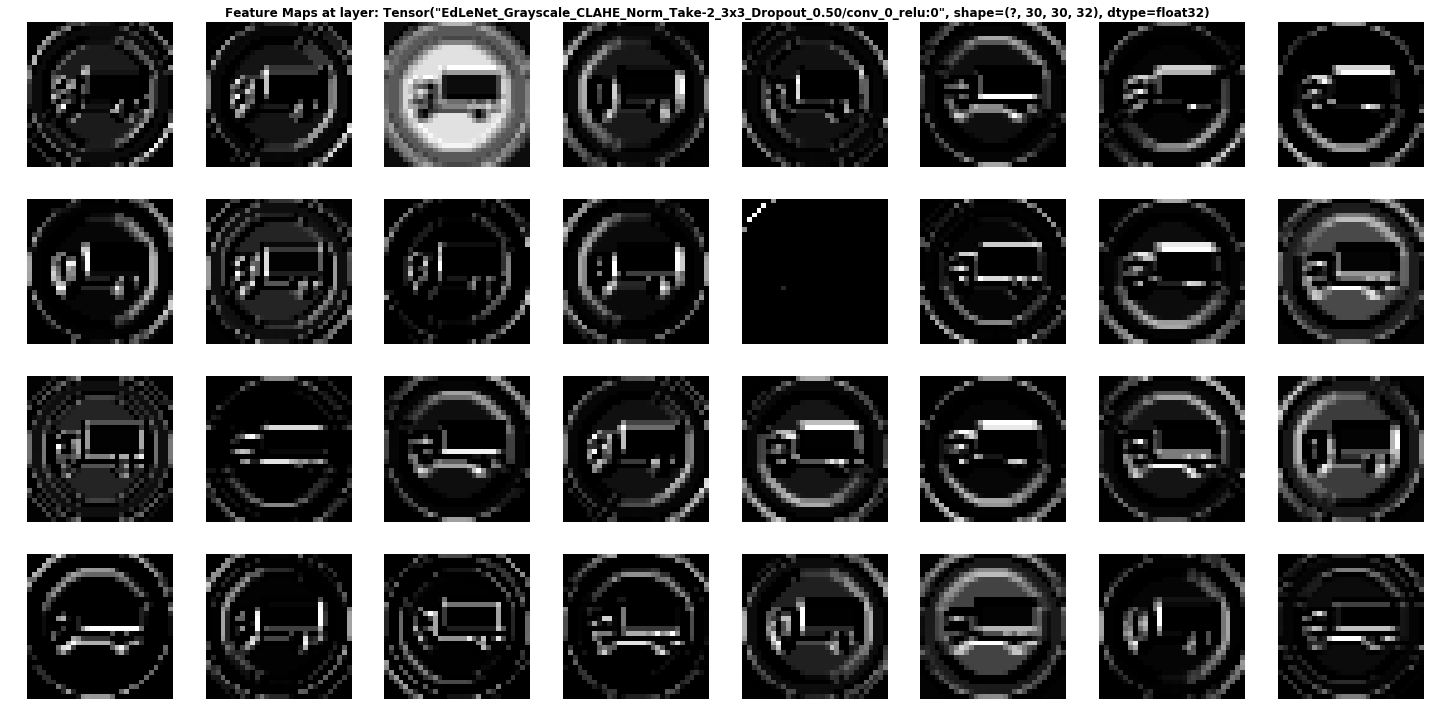
new\_img\_grayscale\_norm\_pred\_acc = np.sum(new\_img\_lbs == preds) / len(preds)  
print("[Grayscale Normalised] Predictional accuracy on new images: {0}%".format(new\_img\_grayscale\_norm\_pred\_acc \* 100))  
...  
[Grayscale Normalised] Predictional accuracy on new images: 100.0%

We also show the top 5 SoftMax probabilities computed for each image, with the green bar showing the ground truth. We can clearly see that our model is quite confident in its predictions. In the worst case (last image), the 2nd most likely prediction has a probability of around 0.1%. In fact our model struggles most on the last image, which I believe is actually a design and not even a real picture. Overall, we have developed a strong model!

# 4.6 Visualizing Our Activation Maps

We show below the results produced by each convolutional layer (before max pooling), resulting in 3 activation maps.

## ****Layer 1****



**Fig 4.9 Convolutional Layer 1**

We can see that the network is focusing a lot on the edges of the circle and somehow on the truck. The background is mostly ignored.

## Layer2:

Activation Map Of Second Convolutional Layer

It is rather hard to determine what the network is focusing on in layer 2, but it seems to “activate” around the edges of the circle and in the middle, where the truck appears.



**Fig 4.10 Convolution layer 2**

**Layer3:**

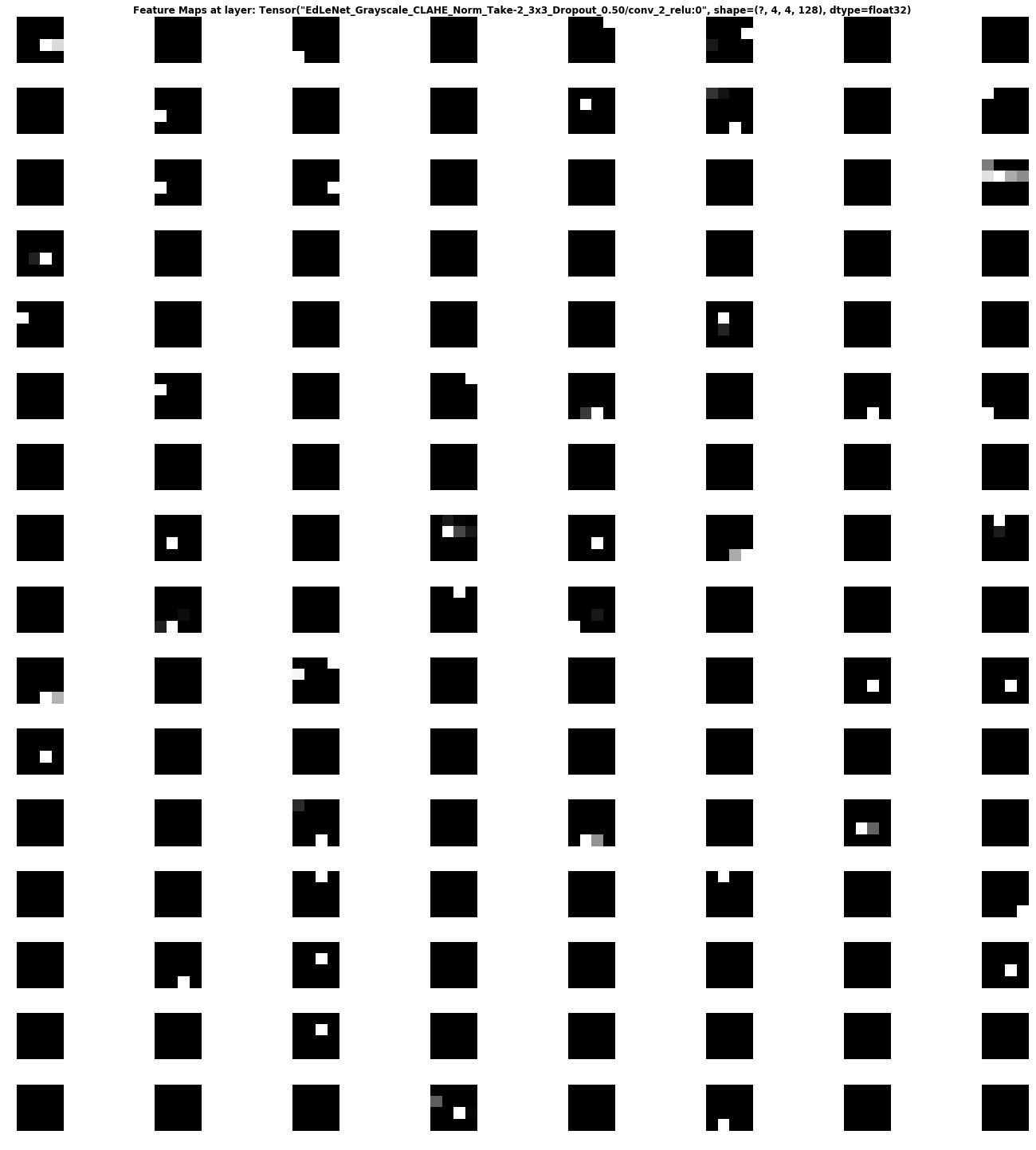
****

Fig 4.11 Convolutional Layer 3

This activation map is also hard to decipher… But it seems the network reacts to stimuli on the edges and in the middle once again.

**4.7.1 Hardware Requirements:**

We used a high-end GPU System with following configuration to conduct our Proposed experiment.

Workstation : Dell

Processor : Intel i5

RAM : 8 GB

Hard Disk Drive : 2 TB

**4.7.2 Software Requirements:**

Operating System : Windows 10 Pro

Programming Language : Python 3.6.6

Deep Learning Framework : TensorFlow & Keras 2.3

Supporting Libraries : Pandas, NumPy, matplotlib, sklearn,

Tkinter, gtts, os.

**Tensor Flow –** TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

**Keras –** Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Features of eras are:

* Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).Supports both convolutional networks and recurrent networks, as well as combinations of the two.
* Runs seamlessly on CPU and GPU.
* Keras is compatible with: **Python 2.7-3.6**.

**Pandas –** Pandas is the most popular python library that is used for data analysis. It provides highly optimized performance with back-end source code is purely written in ***C*** or ***Python***.

**NumPy –** NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation.

**Matplotlib –** Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

**Sklearn** – Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

**GTTS** – (Google Text-to-Speech), a **Python** library and CLI tool to interface with Google Translate's text-to-speech API. Writes spoken mp3 data to a file, a file-like object (bytestring) for further audio manipulation, or stdout . It features flexible pre-processing and tokenizing.

**Tkinter** - is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. **Tkinter** is included with standard Linux, Microsoft Windows and Mac OS X installs of Python. The name **Tkinter** comes from Tk interface. **Tkinter** was written by Fredrik Lundh.

**.**



**Fig 4.12 Libraries used for Development**

**CHAPTER -5**

**IMPLEMENTATION AND RESULTS**

In this Chapter, we give the details of the experimental studies we have performed to understand the efficiency of the proposed method. First, we introduce the dataset used for our studies followed by the performance of different Deep Learning models trained on hand crafted features. Later we present the results of proposed Deep Neural Network Architecture trained on features extracted from pre-trained ConvNet of color fundus images for detection and grading of Traffic Signs.

## 5.1 Implementation

## 5.1.1 Workflow

The process for training a neural network model can be broken down into four different phases.

* **Data Preparation**

First, you will need to collect your data and put it in a form the network can train on. This involves collecting images and labeling them. Even if you have downloaded a data set someone else has prepared, there is likely to be [preprocessing](https://en.wikipedia.org/wiki/Data_pre-processing) or preparation that you must do before you can use it for training. Data preparation is an art all on its own, involving dealing with things like missing values, corrupted data, data in the wrong format, incorrect labels, etc.

* **Creating the model**

Creating the neural network model involves making choices about various parameters and hyper parameters. You must make decisions about the number of layers to use in your model, what the input and output sizes of the layers will be, what kind of activation functions you will use, whether or not you will use dropout, etc.

* **Training the model**

After you have created your model, you simply create an instance of the model and fit it with your training data. The biggest consideration when training a model is the amount of time the model takes to train. You can specify the length of training for a network by specifying the number of epochs to train over. The longer you train a model, the greater its performance will improve, but too many training epochs and you risk overfitting.

* **Model Evaluation**

There are multiple steps to evaluating the model. The first step in evaluating the model is comparing the model's performance against a validation dataset, a data set that the model hasn't been trained on. You will compare the model's performance against this validation set and analyze its performance through different metrics.

There are various metrics for determining the performance of a neural network model, but the most common metric is "accuracy", the amount of correctly classified images divided by the total number of images in your data set.

After you have seen the accuracy of the model's performance on a validation dataset, you will typically go back and train the network again using slightly tweaked parameters, because it's unlikely you will be satisfied with your network's performance the first time you train. You will keep tweaking the parameters of your network, retraining it, and measuring its performance until you are satisfied with the network's accuracy. Finally, you will test the network's performance on a testing set. This testing set is another set of data your model has never seen before.

Therefore, the purpose of the testing set is to check for issues like overfitting and be more confident that your model is truly fit to perform in the real world.

**5.2 Sample Code:**

The code consist of two parts. The first part Consist of training code which trains the data and provide the accuracy. It gives the output in the form of Epochs. The second part is about GUI. It provides an interface to upload the image from the trained dataset and should classify it. There is a classify button in it if we click on classify we get a voice about what is the trffic sign and it will display on the screen.

**5.2.1 traffic\_sign.py:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

import tensorflow as tf

from PIL import Image

import os

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

data = []

labels = []

classes = 43

cur\_path = os.getcwd()

#Retrieving the images and their labels

for i in range(classes):

path = os.path.join(cur\_path,'train',str(i))

images = os.listdir(path)

for a in images:

try:

image = Image.open(path + '\\'+ a)

image = image.resize((30,30))

image = np.array(image)

#sim = Image.fromarray(image)

data.append(image)

labels.append(i)

except:

print("Error loading image")

#Converting lists into numpy arrays

data = np.array(data)

labels = np.array(labels)

print(data.shape, labels.shape)

#Splitting training and testing dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, random\_state=42)

print(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

#Converting the labels into one hot encoding

y\_train =to\_categorical(y\_train, 43)

y\_test =to\_categorical(y\_test, 43)

#Building the model

model = Sequential()

model.add(Conv2D(filters=32, kernel\_size=(5,5), activation='relu', input\_shape=X\_train.shape[1:]))

model.add(Conv2D(filters=32, kernel\_size=(5,5), activation='relu'))

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Dropout(rate=0.25))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'))

model.add(Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'))

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Dropout(rate=0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(rate=0.5))

model.add(Dense(43, activation='softmax'))

#Compilation of the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 15

history = model.fit(X\_train, y\_train, batch\_size=32, epochs=epochs, validation\_data=(X\_test, y\_test))

model.save("traffic\_classifier.h5")

#plotting graphs for accuracy

plt.figure(0)

plt.plot(history.history['accuracy'], label='training accuracy')

plt.plot(history.history['val\_accuracy'], label='val accuracy')

plt.title('Accuracy')

plt.xlabel('epochs')

plt.ylabel('accuracy')

plt.legend()

plt.show()

plt.figure(1)

plt.plot(history.history['loss'], label='training loss')

plt.plot(history.history['val\_loss'], label='val loss')

plt.title('Loss')

plt.xlabel('epochs')

plt.ylabel('loss')

plt.legend()

plt.show()

#testing accuracy on test dataset

from sklearn.metrics import accuracy\_score

y\_test = pd.read\_csv('Test.csv')

labels = y\_test["ClassId"].values

imgs = y\_test["Path"].values

data=[]

for img in imgs:

image = Image.open(img)

image = image.resize((30,30))

data.append(np.array(image))

X\_test=np.array(data)

pred = model.predict\_classes(X\_test)

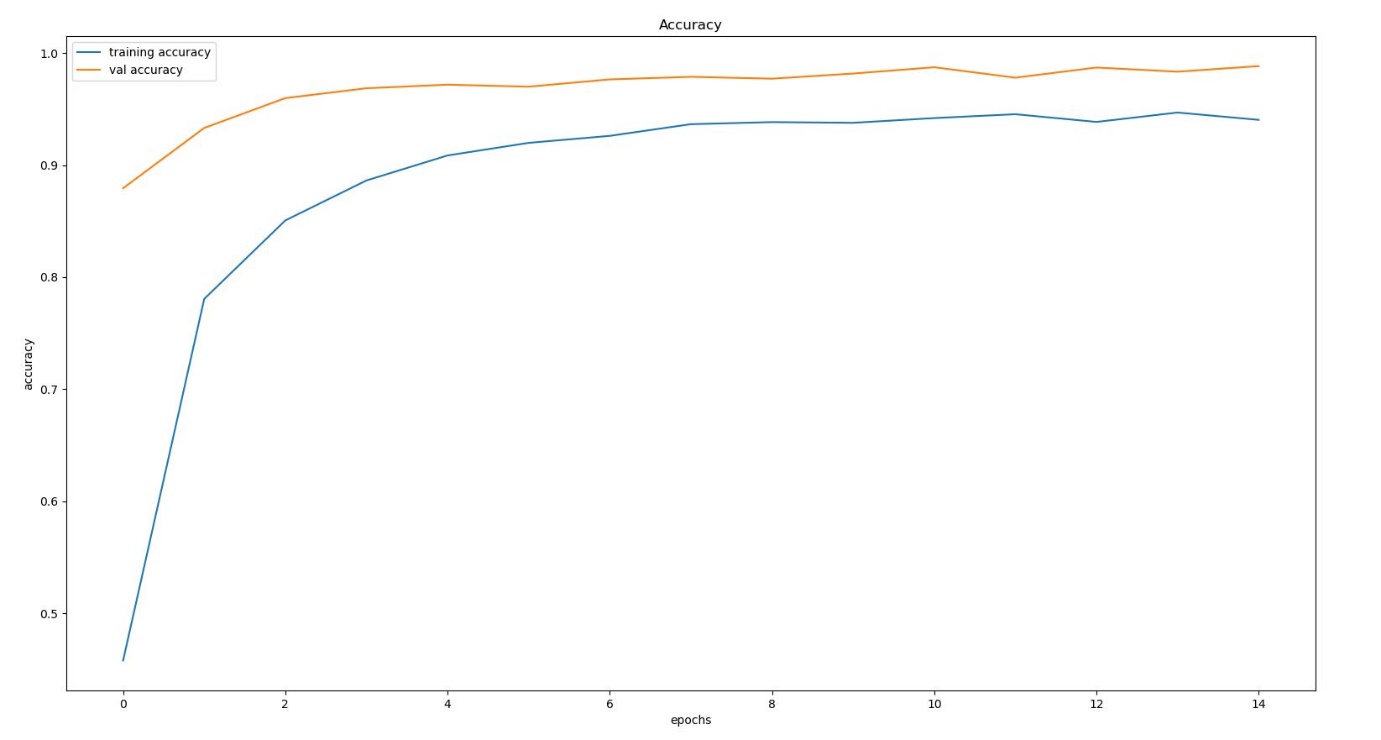
#Accuracy with the test data

from sklearn.metrics import accuracy\_score

print(accuracy\_score(labels, pred))

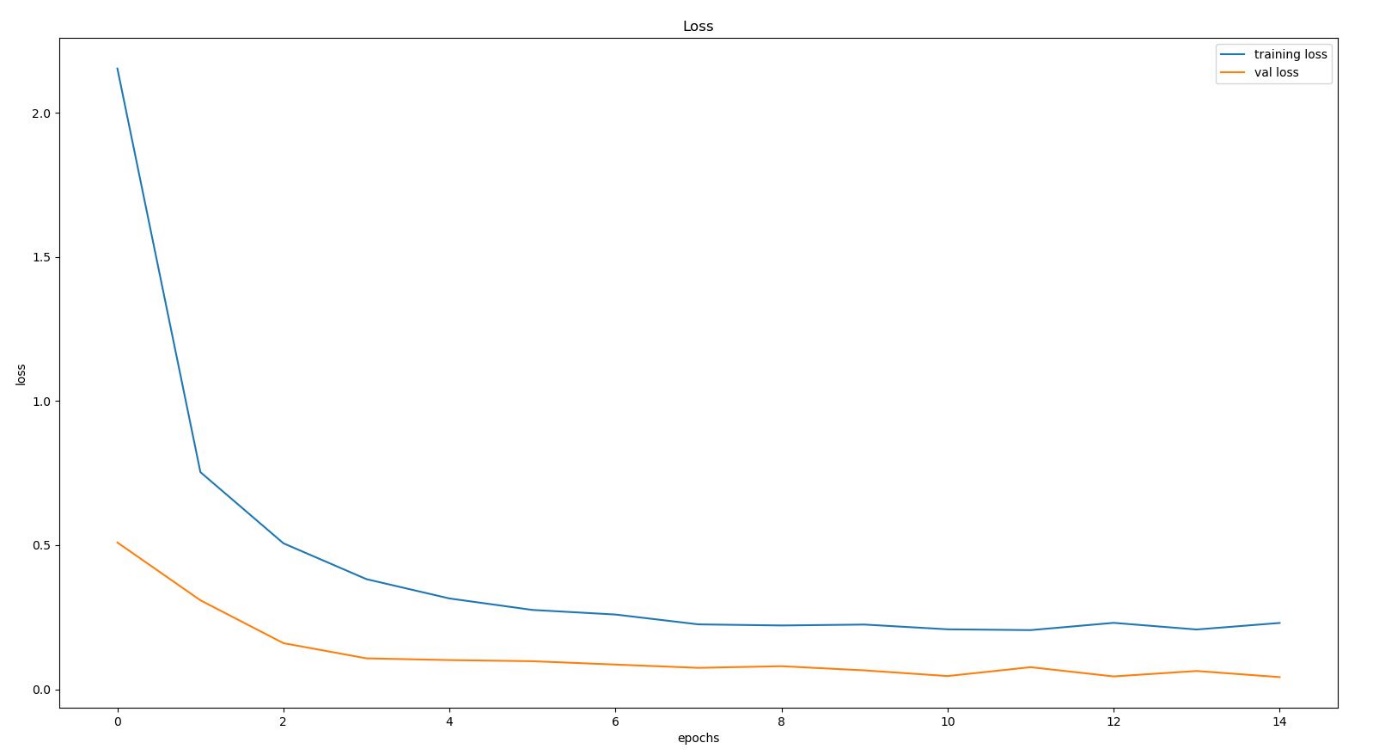
**Output:**

**1.**

****

**Fig 5.1 Epochs for traning val accuracy**

**2.**

****

**Fig 5.2 Ephoch for training loss**

**5.2.2 gui.py:**

Now let’s take a step ahead and build a nice graphical user interface for our deep learning model. A graphical user interface will save a lot of time in testing and seeing the results of our model prediction. The Tkinter is an inbuilt library of python to make a graphical user interface.

From the interface of the GUI application, we will ask the user for an image and extract the file path of the image. Then we use the trained model that will take the image data as input and provide us the class our image belongs to. We will then use the [**dictionary**](https://data-flair.training/blogs/python-dictionary/) to see the name of the class. Create a new python file, you can name it as traffic\_gui.py. Here’s the source code of our graphical user interface so you can run the file from the terminal using the “python gui.py” command.

import tkinter as tk

from tkinter import filedialog

from tkinter import \*

from PIL import ImageTk, Image

import tensorflow as tf

import numpy

#load the trained model to classify sign

from tensorflow.keras.models import load\_model

from gtts import gTTS

import os

#from playsound import playsound

model = load\_model('my\_model.h5')

#dictionary to label all traffic signs class.

classes = { 1:'Speed limit (20km/h)',

2:'Speed limit (30km/h)',

3:'Speed limit (50km/h)',

4:'Speed limit (60km/h)',

5:'Speed limit (70km/h)',

6:'Speed limit (80km/h)',

7:'End of speed limit (80km/h)',

8:'Speed limit (100km/h)',

9:'Speed limit (120km/h)',

10:'No passing',

11:'No passing vehicle over 3.5 tons',

12:'Right-of-way at intersection',

13:'Priority road',

14:'Yield',

15:'Stop',

16:'No vehicles',

17:'Vehicles > 3.5 tons prohibited',

18:'No entry',

19:'General caution',

20:'Dangerous curve left',

21:'Dangerous curve right',

22:'Double curve',

23:'Bumpy road',

24:'Slippery road',

25:'Road narrows on the right',

26:'Road work',

27:'Traffic signals',

28:'Pedestrians',

29:'Children crossing',

30:'Bicycles crossing',

31:'Beware of ice/snow',

32:'Wild animals crossing',

33:'End speed + passing limits',

34:'Turn right ahead',

35:'Turn left ahead',

36:'Ahead only',

37:'Go straight or right',

38:'Go straight or left',

39:'Keep right',

40:'Keep left',

41:'Roundabout mandatory',

42:'End of no passing',

43:'End no passing vehicle > 3.5 tons' }

#initialise GUI

top=tk.Tk()

top.geometry('800x600')

top.title('Traffic sign classification')

top.configure(background='#CDCDCD')

label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))

sign\_image = Label(top)

def classify(file\_path):

image = Image.open(file\_path)

image = image.resize((30,30))

image = numpy.expand\_dims(image, axis=0)

image = numpy.array(image)

print(image.shape)

pred = model.predict\_classes([image])[0]

global sign

sign= classes[pred+1]

label.configure(foreground='#011638', text=sign)

print(sign)

text\_val = sign

language = 'en'

obj = gTTS(text=text\_val, lang=language, slow=False)

obj.save("exam.mp3")

#playsound("exam.mp3")

os.system("exam.mp3")

def show\_classify\_button(file\_path):

classify\_b=Button(top,text="Classify Image",command=lambda: classify(file\_path),padx=10,pady=5)

classify\_b.configure(background='#364156', foreground='white',font=('arial',10,'bold'))

classify\_b.place(relx=0.79,rely=0.46)

def upload\_image():

try:

file\_path=filedialog.askopenfilename()

uploaded=Image.open(file\_path)

uploaded.thumbnail(((top.winfo\_width()/2.25),(top.winfo\_height()/2.25)))

im=ImageTk.PhotoImage(uploaded)

sign\_image.configure(image=im)

sign\_image.image=im

label.configure(text='')

show\_classify\_button(file\_path)

except:

pass

upload=Button(top,text="Upload an image",command=upload\_image,padx=10,pady=5)

upload.configure(background='#364156', foreground='white',font=('arial',10,'bold'))

upload.pack(side=BOTTOM,pady=50)

sign\_image.pack(side=BOTTOM,expand=True)

label.pack(side=BOTTOM,expand=True)

heading = Label(top, text="Know Your Traffic Sign",pady=20, font=('arial',20,'bold'))

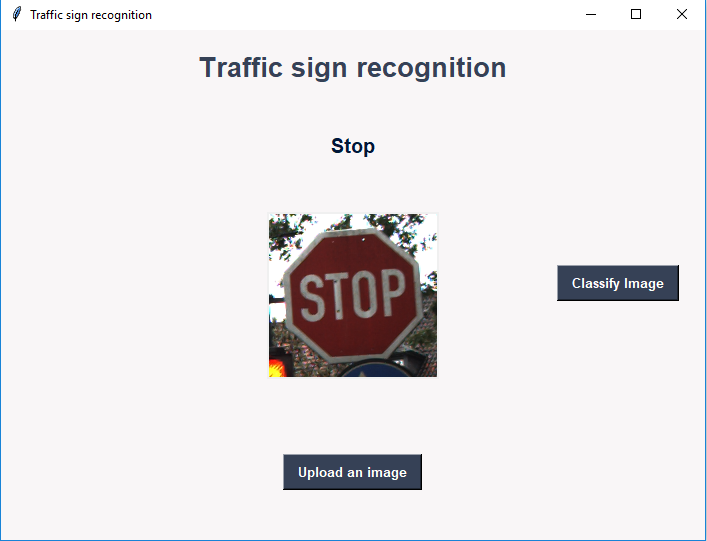
heading.configure(background='#CDCDCD',foreground='#364156')

heading.pack()

top.mainloop()

# Output

## 1.

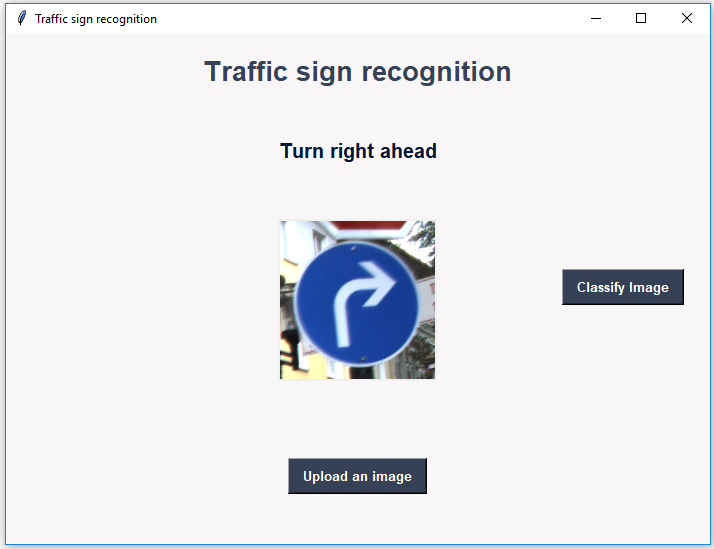


**Fig 5.3 Output 1**

## 2.



**Fig 5.4 Output 2**



**Fig 5.5 Output 3**

**Chapter – 6**

**Conclusion**

German traffic sign recognition benchmark (GTSRB) recognition system was designed using Convolutional Neural Networks (CNN). The training and testing dataset consists of more than 2,728 sign samples collected for 24 different traffic **s**igns. The dataset went through a preprocessing stage before inputting it to the network. It got partitioned into training, testing and validating datasets. The initial design was based on similar previous work which was used as a base for the subsequent improved design. The final Deep CNN architecture proposed in this work consists of two convolutional layers, two maxpooling layers one dropout layer and 3 dense layers. 100% accuracy was obtained for epoch 150 for all batch sizes. We demonstrated the usefulness of the designed CNN architecture by implementing a practical system that can take real time Traffic Sign via camera attached to vehicle, classifies them to the corresponding sign, and then give voice notification to driver or take automatic decision for autonomous cars. Future work will include increasing the size of the dataset and publishing it so that it can be used by other researchers for benchmarking purposed. Work will also continue on developing more robust and computationally low cost recognition systems.

**References:-**

* [*Bishop, C. M.*](https://en.wikipedia.org/wiki/Christopher_M._Bishop) (2006), Pattern Recognition and Machine Learning, Springer, [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*978-0-387-31073-2*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-387-31073-2)
* [*Friedman, Jerome H.*](https://en.wikipedia.org/wiki/Jerome_H._Friedman) (1998). "Data Mining and Statistics: What's the connection?". Computing Science and Statistics. **29** (1): 3–9.
* Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*978-0-07-042807-2*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-07-042807-2).
* [*Jeff Leek*](https://en.wikipedia.org/wiki/Jeffrey_T._Leek) (12 December 2013). [*"The key word in "Data Science" is not Data, it is Science"*](http://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/). Simply Statistics.
* Donoho, David (18 September 2015). [*"50 years of Data Science"*](http://courses.csail.mit.edu/18.337/2015/docs/50YearsDataScience.pdf)
* Data science and its application. Escoufier, Yves., Hayashi, Chikio (1918.)., Fichet, Bernard. Tokyo: Academic Press/Harcourt Brace. 1995
* Deep learning for computer vision(pyimagesearch.com)
* Datascience with python(dataflair.com)
* [Contrast Limiting Adaptive Histogram Equalization](http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html) (aka CLAHE) function. And lane line detection by towardsdatascience.com
* Medium.com articles on datacience and visualization
* Tensorflow.org
* Keras.io
* Anaconda.com
* Opencv.org