A Project Report

on

TRAFFIC SIGNS RECOGNITION

Submitted in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

by

J. Nicole (16WH1A1226)

K. Lakshmi Prasanna (16WH1A1234)

T. Soumya (16WH1A1258)

Under the esteemed guidance of

Ms. D. Kavya Shruthi
Assistant Professor

Department of Information Technology

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC '**A**' Grade, Approved by AICTE, and
Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

April 2020

DECLARATION

We hereby declare that the work presented in this project entitled "TRAFFIC SIGNS RECOGNITION" submitted towards completion of the major project in IV year of B.Tech IT at "BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN", Hyderabad is an authentic record of our original work carried out under the esteem guidance of Ms. D. Kavya Shruthi, Assistant Professor, IT department.

J. Nicole
(16WH1A1226)

K. Lakshmi Prasanna
(16WH1A1234)

T. Soumya
(16WH1A1258)

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Information Technology

Certificate

This is to certify that the Project report on "Traffic Signs Recognition" is a bonafide work carried out by J. Nicole (16WH1A1226), K. Lakshmi Prasanna (16WH1A1234), T. Soumya (16WH1A1258), in the partial fulfillment for the award of B.Tech degree in Information Technology, BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN, Bachupally, Hyderabad, affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Guide
Ms. D. Kavya Shruthi
Assistant Professor
Department of IT

Head of the Department
Dr. Aruna Rao S L
Professor and HOD
Department of IT

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal**, BVRIT Hyderabad, for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr**. **Aruna Rao S L, Head**, Department of Information Technology, BVRIT Hyderabad for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, Ms. D. Kavya Shruthi, Assistant Professor, Department of IT, BVRIT Hyderabad, for her constant guidance, encouragement and moral support throughout the project.

Finally, We would also like to thank our Project coordinator, all the faculty and staff of IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

J. Nicole
(16WH1A1226)
K. Lakshmi Prasanna
(16WH1A1234)
T. Soumya
(16WH1A1258)

ABSTARCT

Traffic signs recognition (TSR) is one of the most important background research topics for enabling autonomous vehicle driving systems. Autonomous driving systems require special handling of input data. A deep learning based road traffic signs recognition method is developed which is very promising in the development of Advanced Driver Assistance Systems (ADAS) and autonomous vehicles. The system architecture is designed to extract main features from images of traffic signs to classify them under different categories. The presented method uses Convention Layer Network to extract a deep representation of traffic signs to perform the recognition. CNN have a high recognition rate, thus making it desirable to use for implementing various computer vision tasks. TensorFlow is used for the implementation of the CNN. We have achieved good results for traffic signs on the Traffic Signs data set.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

Term/Abbreviation	Definition
TSR	Traffic Signs Recognition
ADAS	Advanced Driver Assistance Systems
CNN	Convolutional Neural Network
GPS	Global Positioning System
GUI	Graphical User Interface
PIL	Python Imaging Library
SVM	Support Vector Machine
IDE	Integrated Development Environment
RGB	Red Green Blue

TABLE OF CONTENTS

1. INTRODUCTION

Traffic-signs identification and classification is an intriguing part in computer-vision and it is particularly important with regards to self-governing vehicle innovation. Traffic signs recognition is an innovation by which a vehicle can perceive to recognize the traffic sign put on the street for example, "speed limit" or "turn ahead". Traffic signs can be analyzed utilizing front oriented cameras in numerous modern vehicles and trucks. They insist the driver by providing commands, admonitions and some of the times by taking control of the vehicle itself.

1.1 OBJECTIVE

Traffic Signs Recognition (TSR) is to build 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. The properties of traffic signs and their implications for image processing for the recognition task can be understood. There is a recognition, that is invariant to in-plane transformations such as translation, rotation, and scaling based on shape measures. It classifies the results based on the image recognition.

1.2 PROBLEM IN EXISTING SYSTEM

The systems can take advantages of Global Positioning System (GPS). It could be always flawless if an updated traffic sign location database would be available. But few cars have GPS installed and traffic sign localization databases are not available for download. Tam T. Le has proposes the Support Vector Machine (SVM) method to retrieve candidate region of traffic sign in image processing, which concerns blocks of pixels; therefore, the information of neighbor pixels can help to handle the diversification of both training and testing data. It means that SVMs can return a recall rate better than that of single-pixel-based algorithms. In this approach, instead of deciding whether a pixel has an interested color or not, it chooses blocks of interested color through the results of SVMs.

1.3 SOLUTION

The solution for the Existing System is, by using feature extraction of a pixel block will help to reduce complexity of the calculation in SVMs because the dimensions of the input vectors, support vectors, and the hyper-plane are only equal to two times of the number of pixels in each block. Convention Layer Network is used to extract a deep representation of traffic signs to perform the recognition. It is constituted of a Convolutional Neural Network (CNN) modified by connecting the output of all convolutional layers helps to extract features.

Figure 1: Traffic Signs Recognition System

1.4 FEATURES

Traffic signs have been designed to be principally distinguishable from the natural and/or man-made backgrounds. They are characterized by many features make them recognizable with respect to the environment. They are designed in fixed 2-D shapes like triangles, circles, octagons, or rectangles. The colours of the signs are chosen to be far away from the environment,

which make them easily recognizable by the drivers. The information on the sign has one colour and the rest of the sign has another colour. The tint of the paint that covers the sign should correspond to a specific wavelength in the visible spectrum. The signs are located in well-defined locations with respect to the road, so that the driver can, more or less, expect the location of these signs. They may contain a pictogram, a string of characters or both. The road signs are characterized by using fixed text fonts, and character heights. They can appear in different conditions, including partly occulted, distorted, damaged and clustered in a group of more than one sign.

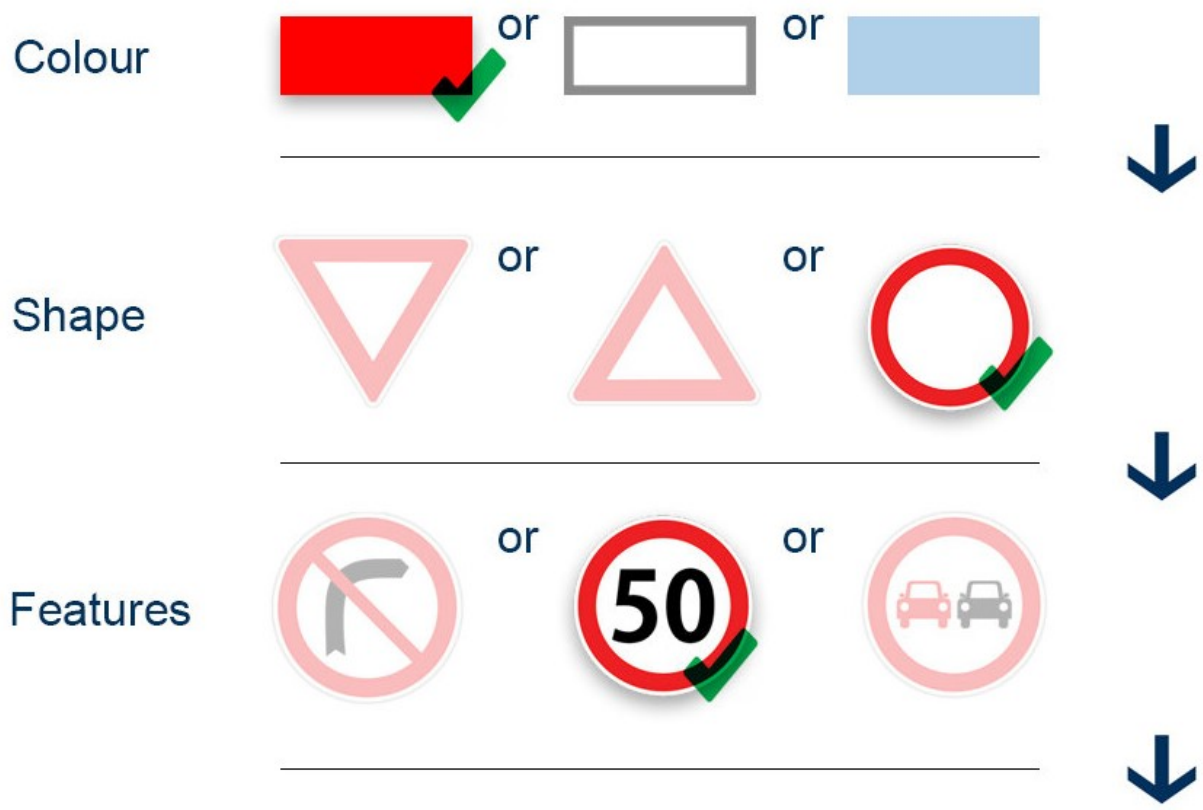

Figure 2: Traffic Signs Recognition Graphics

2. LITERATURE SURVEY

Different traffic sign recognition methods have been proposed in the recent past. They usually consist of two sequential processes, namely a detection stage that identifies a Region of Interest, and a recognition stage that identifies the exact type of sign or rejects the identified Region of Interest. There are four stages to detect traffic signals. Those are Image procurement (remove deburring), Color segmentation (to detect color), Blob detection (region & shape) and Classification using multiple neural network (high accuracy).

2.1. DETECTION METHODS

In this, we review the techniques that have been used for the detection of traffic signs. As mentioned above, the main task in detection is creating candidate image regions which are probably traffic signs. Therefore, we need to search for ROIs from image sequences and preprocess them for the sign recognition stage. Several methods can be used for extracting RoIs, based on main characteristics of traffic signs such as shape and color. In general, we can divide the detection methods into three main groups:

• Detection based on color information

• Detection based on shape information

• Detection based on hybrid methods

In the following Sections, we discuss the common approaches used in each of the three groups.

2.1.1 Detection Based on Color Information

One of the prevalent approaches for sign detection is using color information. By using thresholding or advanced segmentation methods, finding the areas of an image which contains the color of interest is possible. However, the main drawback of this method is its high sensitivity to variable illumination conditions. Depending on the time of the day and weather conditions, the colors may be inconsistent. Different types of color spaces are used. The most widely used ones are RGB, HSI/HSV, YUV, YCbCr, CIELab, and CIECAM97.

RGB Color Space

RGB is the most frequently used color space for most applications in image processing and computer vision. The intensity values of RGB components are between 0 and 255. Since there are three color channels, a total of 2563 = 16, 777, 216 colors can be displayed.

Authors in [7] proposed a new method for detecting white signs by using RGB color space. They used chromatic and achromatic filters in order to help them in detecting white signs. These are computed as:

$$f(R,G,B) = (|R - G| + |G - B| + |B - R|) / 3D$$

where R, G, and B represent the brightness of the red, green, and blue channels. D is the degree of extraction of an achromatic color, and is determined experimentally. If the value of f(R, G, B) is less than 1, then it is said to represent achromatic colors, and conversely if it is greater than 1 [7].

HSV Color Space

The color space that has been used for road sign segmentation is known as HSV stands for Hue, Saturation, and Value. They are the three components of this color model, defined as follows:

• **Hue:** Hue is an angle between 0 and 360 degrees and it represents the color.

• **Saturation:** This value represents the range of gray in color space. It varies from 0 to 1. 0 means the color is gray and 1 means the color is a primary color. For example, the saturation value of white is 0.

• **Value:** This value which is also called lighting varies from 0 to 1. It indicates how dark or how bright a color is. 0 means completely dark and 1 means completely bright.

Hasan Fleyeh proposed a different solution for color detection and segmentation of road signs based on fuzzy sets. Paclik et al used this color space due to its similarity to human perception of

colors also applied the HSV color space. They converted the RGB color space to the HSV color space following this equation: V = max(R, G, B).

2.1.2 Detection Based on Shape Information

Detection based on shape information has important characteristic of a traffic sign is its shape (circular, triangular, octagonal, and rectangular). While color-based detectors are popular, there are many other approaches based on the shapes of traffic signs. The main drawback of color-based detectors is their sensitivity to weather conditions and variations in luminance. In contrast to color-based detectors, shape-based detectors do not suffer from weather conditions and variations in luminance. Therefore, many researchers prefer to use shape-based techniques for the detection of traffic signs. Several approaches for shape-based traffic sign detection have been proposed in the current literature. We proceed to review the most common ones.

The Hough Transform

The Hough Transform (HT) can be used for the detection of lines, circles, rectangles or other curves. This method was first introduced in 1962 and its first application was finding lines in image sequences. The main advantage of HT is its immunity to noise, scaling, and rotation. Authors in [8] used the Hough Transform for circumference in order to detect circular signs, while for triangular-signs detection they used the Hough transform for straight lines. Loy and Barnes [9] proposed a method for traffic sign detection based on fast radial symmetry transform, and the general technique is almost identical to the Hough transform. They first create a gradient magnitude image and then threshold the output image in order to discard points with low magnitudes.

Edge Detection

Edge detection is one of the basic techniques of image processing for finding the boundaries of different objects in images. The main application of edge detection in areas such as image processing and computer vision is segmentation when images can be divided into areas corresponding to various objects. The Canny edge detector was used by M. A. Garcia - Garrido, M. A. Sotelo, because this method preserves contours, which is necessary for shaped based

traffic sign detectors. Aoyagi and Asakura proposed a system that used Gaussian and Laplacian filter. Corner-point detection is yet another technique for finding shapes of interest. Paulo and Correia approached traffic sign detection based on the Harris corner detector. They identified the triangular and square shapes by finding the corners of each ROI, using the Harris corner detection.

Neural Networks

The approach for shape detection is the use of trained Neural Networks (NN). Zhu et al. used a neural network for detecting triangular signs. [10] is another example of using NN for traffic sign shape detection.

Template Matching

The common method in image processing and pattern recognition is template matching. Template Matching is a high-level machine vision technique that allows identifying the parts of an image (or multiple images) that match a given image pattern.

Gavrila proposed a shaped-based system, based on distance transforms and template matching. The first stage of this technique is finding the edges in the original images. In the second stage, a distance transform image is created. Matching a template with the edge image is also possible, but the advantage of template matching with DT image is that the resulting similarity measure is much smoother.

Gradient features

In recent years, Histogram of Oriented Gradients (HoG) features were used in many contributions for traffic sign feature extraction. The HoG detector was first introduced for the task of pedestrian detection. This method begins with dividing the imagery into a set of blocks. The HoG is then computed for each block. HoG has different parameters that can influence the accuracy of the detection stage. HoG features possess several advantages in comparison to other shaped-base methods, including high accuracy, scale invariance, local contrast normalization, and coarse spatial sampling.

2.1.3 Detection Based on Hybrid Methods

Both color-based and shape-based methods have some advantages and disadvantages. Thus, a combination of the two reviewed methods is also prevalent among researchers. Many sign detection systems include a color segmentation stage followed by some kind of shape extraction stage. The work of Fang et al. includes using the hue as a color feature and an edge detector method for shape feature extraction are other examples of integrating color-based methods with the ones based on shape analysis. Mathias et al. et al. used Integral Channel Features (ICF) for sign detection which was first established by Dollar et al. for the task of pedestrian detection. Integral channel features include a combination of different orientation channels, color space channels, and gradient magnitude channels. Sekanina and Torresen proposed an algorithm for detection of Norwegian speed limit signs. They used the RGB color space for the color segmentation stage and template matching for locating speed limit signs. The converted versions of the RGB color space and a Laplacian of Gaussian (LoG) edge detector have been used for detection of triangular traffic signs.

2.2 RECOGNITION METHODS

The next stage in TSDR systems is sign recognition, which ascertains whether the detected candidate is an actual traffic sign or not. Different ways exist for recognizing the detected sign candidates. We introduce the most widely used methodologies in this Section. We have reviewed Neural Networks for the task of sign detection. This method is also prevailing for the task of traffic sign recognition. Six Neural Networks have been trained with the back propagation method for six different classes of road signs. Their proposed algorithm was tested on 200 different traffic signs. Authors in [11] also selected six categories of road signs for recognition. These are stop, yield way, no left turn, no right turn, speed limit 60, and speed limit 90 signs. In order to recognize the exact type of signs, they used a series of one to one architectural Multi-Layer Perception (MLP) Neural Networks. They took advantage of Resilient Back Propagation (RP) and Scaled Conjugate Gradient (SCG) algorithms for training their neural networks. The average accuracy of system using RP classifiers is 91%. And the system using SCG classifiers also resulted in a 91% recognition rate.

The other multi-layer network was used in to classify the extracted candidate road signs. They performed the classification based on a developed and tested feed-forward MLP neural network classifier. They also used The Conjugate Gradient Descent optimization algorithm in order to achieve better results. They obtained an average classification rate of 91%. They implemented two different back propagation Neural Networks for recognition of circular and triangular road signs. One of the networks identifies the triangular signs while the other one identifies the circular ones. The authors considered the speed-limit and end-of-speed-limit; stop, forbidden overtaking, and end-of-forbidden-overtaking signs for the circular ones, while for triangular signs, the yield way sign, and dangerous curves signs were selected. Finally, they reported a 98.5% recognition rate for speed limit signs and a 97.2% recognition rate for warning signs. Aoyagi and Asakura also proposed a traffic sign recognition module using Neural Net14 works. They classified their recognition category into three classes: the speed sign, other traffic sign, and not a traffic sign. They used 324 input layers units, 15 hidden layer units, and 3 output layer units. Back propagation is also used in their learning process.

Method for traffic sign classification is using the Joint Transform Correlation (JTC). JTC is one of the main techniques in pattern recognition. In a JTC, the unknown input scene and the known reference image are displayed side-by-side in the input plane known as the input joint image which is Fourier Transformed (FT). Then, these FT patterns are joined with each other in order to build a pattern called the Joint Power Spectrum (JPS). Based on JPS values between a test image and a template one, computing a correlation is possible. If two correlation peaks are detected, a match is found. This normalized cross-correlation made the system invariant to lighting conditions. They reached the identification rate of 46%. Perez and Javidi presented a non-linear correlator that performs many correlations between an input scene and different reference targets. According to them, non-linear filters provide invariance to distortions of the target, noise robustness, and rejection of background noise.

2.3 INFORMATION GATHERING

Many algorithms and methodologies have been proposed for road traffic sign detection. Tam T. Le has proposes the Support Vector Machine (SVM) method to retrieve candidate region of traffic sign in real-time image processing in which it utilizes a block of pixels as an input vector.

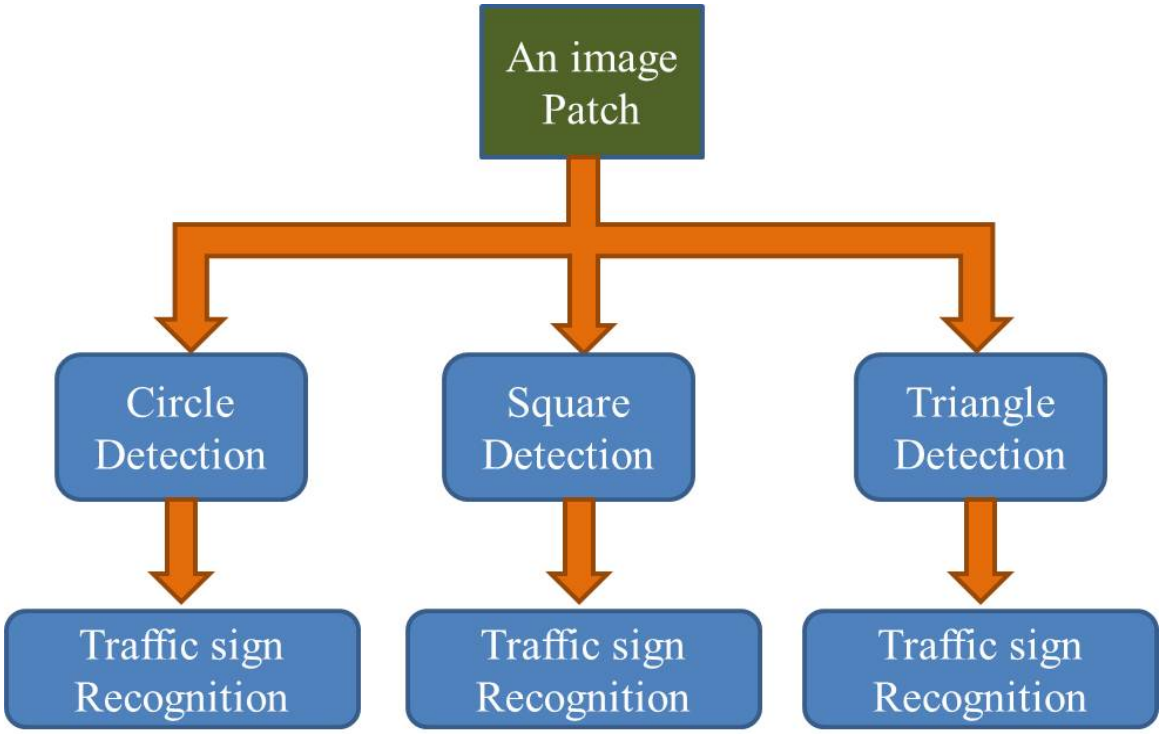

Figure 3: Basic approach for Traffic Signs Recognition

Another method has also been proposed by Hassan Shojania based on thresholding, convolution masks and geometric constraint method but in this they didn't implemented the pictographic recognition stage. Auranuch Lorsakul proposes the system with Neural Network technique with canny edge detection method; in this test sign images including distortion images are provided into the program in order to identify the network generalization. Because the algorithm attempts to detect the circle or ellipse in the test images, this requires name processing time in the much complex background images that have high numbers of the potential area candidates.

3. REQUIREMENT SPECIFICATION

3.1 SOFTWARE REQUIREMENT

To implement this project we will be using Keras which is a popular deep learning framework for **Python** and some additional library scikit-learn, numpy, PIL, pandas, tkinter, and **Spyder**.
Spyder is an open-source cross-platform Integrated Development Environment (IDE) for scientific programming in the Python language. Spyder integrates with a number of prominent packages in the scientific python stack. Spyder uses Qt for its GUI, and is designed to use either of the PyQt or PySide Python bindings. QtPy, a thin abstraction layer developed by the Spyder project and later adopted by multiple other packages, provides the flexibility to use either backend.
For this project, we are using the public dataset available at Kaggle: Traffic Signs Dataset.

Libraries

- **Keras**

  Keras is an open-source neural-network library written in Python. It allows for easy and fast prototyping. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It runs seamlessly on Central Processing Unit and Graphics Processing Unit. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

- **Keras.models**

  There are two main types of models available in Keras: the Sequential model and the Model class used with the functional Application programming interface. These models have a number of methods and attributes in common:

- model.layers() is a flattened list of the layers comprising the
- model.summary() prints a summary representation of your model.

- **Keras.optimizers**

  An optimizer is one of the two arguments required for compiling a Keras model. You can either instantiate an optimizer before passing it to model.compile() or you can call it by its name. In the latter case, the default parameters for the optimizer will be used.

- **TensorFlow**

  TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Its flexible architecture allows for the easy deployment of computation across a variety of platforms and from desktops to clusters of servers to mobile and edge devices.

- **NumPy**

  NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- **Pandas**

  Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

- **Matplotlib**

  Matplotlib is a plotting library for the Python programming language and its numerical mathematics is extension NumPy. It provides an object-oriented API for embedding plots

into applications using general-purpose Graphical User Interface (GUI) toolkits.

- **Python Imaging Library (PIL)**

  It is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

3.2 HARDWARE REQUIREMENT

Operating System - Windows 8 or higher version is required.
Hard Drive - Minimum 32 GB, Recommended 64 GB or more.
Processor Needed - Minimum core i5 CPU and 2.50 GHz processor is needed.
Physical Memory - Minimum 8GB RAM is necessary.

4. DESIGN OF THE SYSTEM

Using a fully connected neural network to make an image classification requires a large number of layers and neurons in the network, which increases the number of parameters leading the network to over-fitting (memorizing the training data only). The input image may also lose its pixels correlation properties since all neurons (carrying pixels values) are connected to each other.

Convolutional neural networks have emerged to solve these problems through their kernel filters to extract main features of the input image and then inject them into a fully connected network to define the class.

The chosen architecture in our application is convolutional neural network (Fig. 4). It contains layers of convolution and simplification functions made by 5x5 kernel filters, Batch Normalization and a max pooling filter of 2x2 to reduce at last the input image of 32x32 into 16 maps of 5x5. The feature images carry most important features to define a specified traffic signs class by processing them into layers of fully connected network.

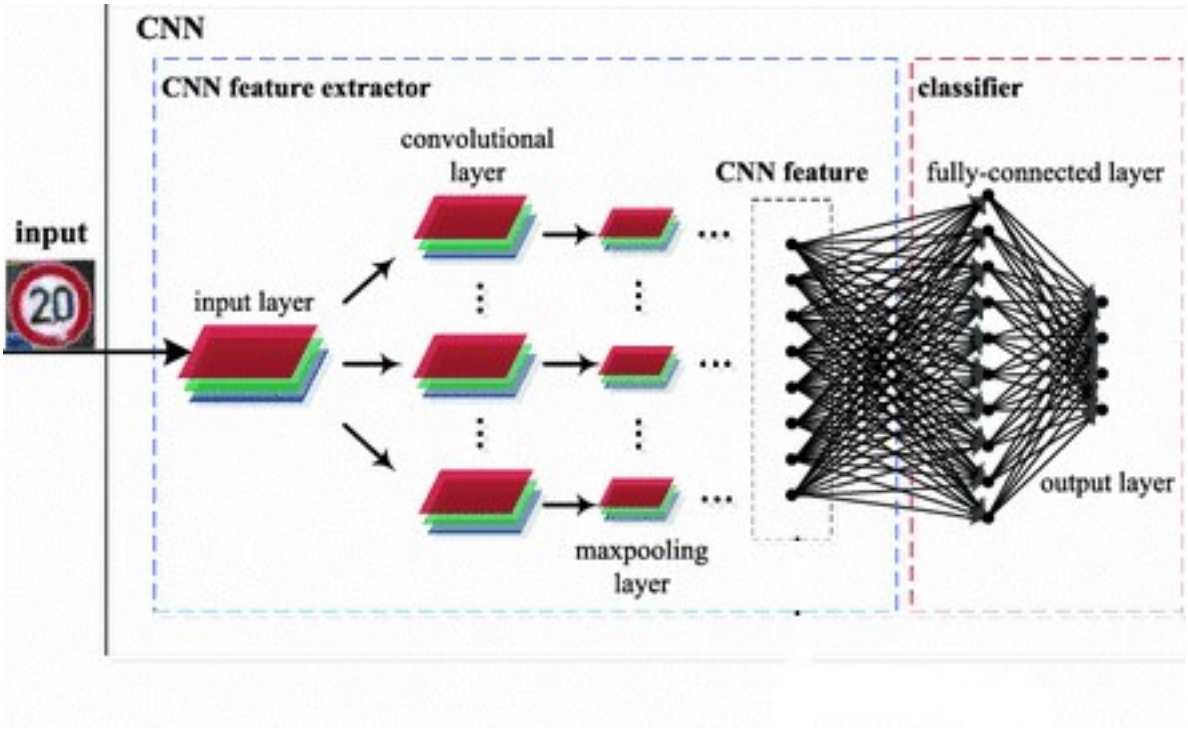

Figure 4: Design for Traffic Signs Recognition

Convolution: The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input data using a convolution filter to produce a feature map.

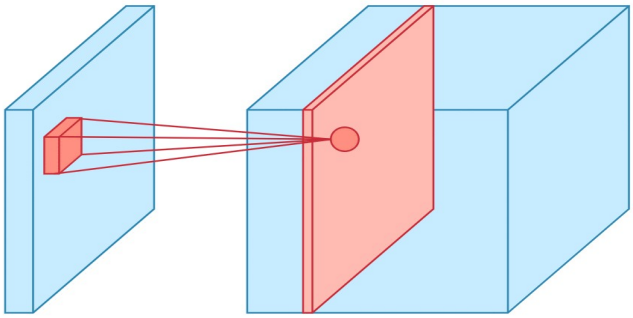

Figure 5: Convolution filter over the input

Max Pooling: After a convolution operation we usually perform pooling to reduce the dimensionality. The most common type of pooling is max pooling which just takes the max value in the pooling window.

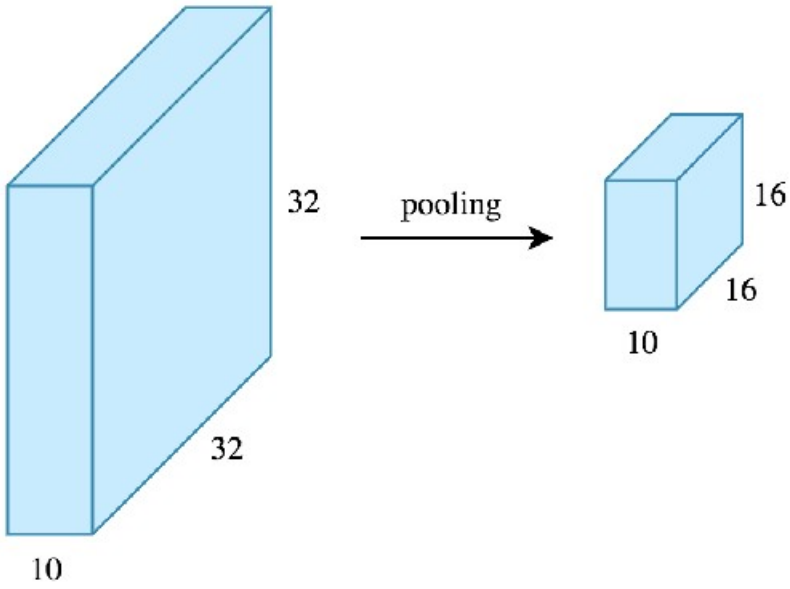

Figure 6: Max Pooling

Fully Connected: After the convolution + pooling layers we add a couple of fully connected layers to wrap up the CNN architecture. The output of both convolution and pooling layers are 3D

volumes, but a fully connected layer expects a 1D vector of numbers. So we flatten the output of the final pooling layer to a vector and that becomes the input to the fully connected layer. Flattening is simply arranging the 3D volume of numbers into a 1D vector.

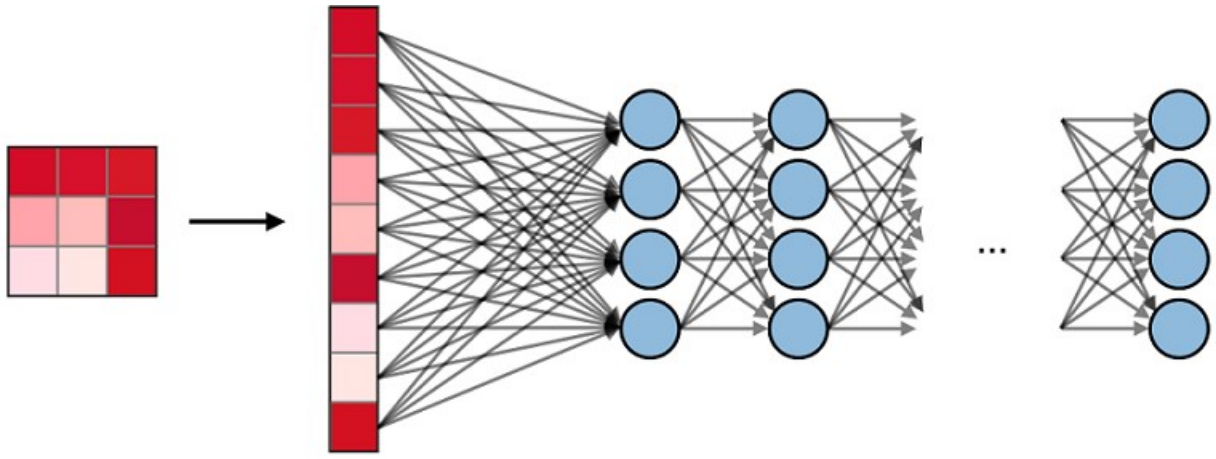

Figure 7: Fully Connected

Dropout: The most popular regularization technique is dropout. It is used to prevent overfitting.

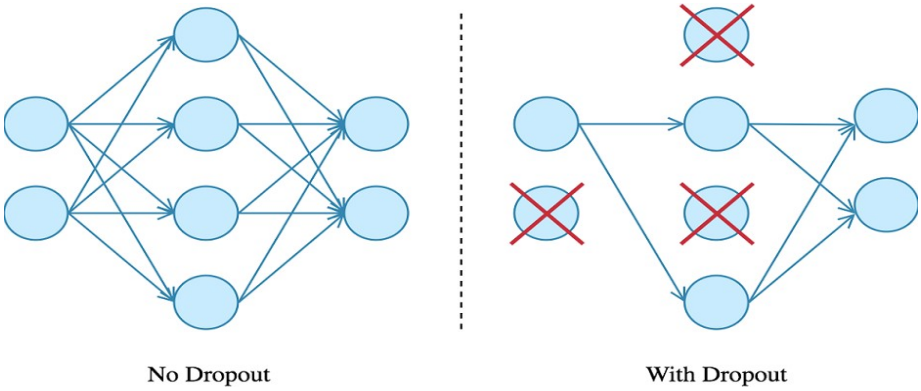

Figure 8: Dropout Visualization

Softmax: The softmax step can be seen as a generalized logistic function that takes as input a vector of scores and outputs a vector of output probability through a softmax function at the end of the architecture.

Dense Layer: Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.

Output = activation (dot (input, kernel) + bias)

ReLU (Rectified Linear Unit) Activation Function: ReLU stands for rectified linear unit, and is a type of activation function. Mathematically, it is defined as y = max (0, x).

Batch Normalization: Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks. We can use higher learning rates because batch normalization makes sure that there's no activation that's gone really high or really low. And by that, images that previously couldn't get to train, it will start to train. It reduces overfitting because it has a slight regularization effects. To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

4.1 DATASET

A rich dataset is needed in object recognition based on neural network in order to train the system and evaluate its results. For the purpose of traffic signs classification, we used the German Traffic Sign Benchmark (GTSB) which contains 43 classes divided into 3 categories as represented in table 4.1.

The "German Traffic Sign Recognition Benchmark" is a multi-category classification. Automatic recognition of traffic signs is required in advanced driver assistance systems and constitutes a challenging real world computer vision and pattern recognition problem. A comprehensive, lifelike dataset of more than 50,000 traffic sign images has been collected. It reflects the strong variations in visual appearance of signs due to distance, illumination, weather conditions, partial occlusions, and rotations. The images are complemented by several precomputed feature sets to allow for applying machine learning algorithms without background

knowledge in image processing. The dataset comprises 43 classes with unbalanced class frequencies.

Table 4.1: The Dataset Distribution

Category	Task	Number of images	Shape
Training data	Used to train the network	34799	4 dimensions tensor to determine the image index in the dataset, the pixel's row-column and the information it carries (Red Green Blue value)
Validation data	Allows to supervise the network performances while training it	4410	
Testing data	Used to evaluate the final network	12630	

- Sample images of dataset with 43 different classes

Figure 9: Images from Dataset

5. MODULES

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.

5.1 EXTRACTION

The Extraction module consists of 2 phases: one is giving traffic signs image as input and Detection of traffic signs.

Traffic Signs Detection:

The purpose of traffic sign detection is to find the locations and sizes of traffic signs in natural scene images. The well-defined colors and shapes are two main cues for traffic sign detection. Thus, we can divide the detection methods into two categories: color-based and shape-based. Color-based methods are usually fast and invariant to translation, rotation and scaling. As color can be easily affected by the lighting condition, the main difficulty of color-based methods is how to be invariant to different lighting conditions. These methods tend to follow a common scheme: the image is transformed into a color space and then thresholding in RGB (Red Green Blue) space.

Techniques using shapes could be a good alternative when colors are missing or when it is hard to detect colors. Shape-based techniques should be able to avoid difficulties related to invoking colors for sign detection and robust to handle in-plane transformations such as translation, scaling and rotation. Much effort has been exerted to develop these techniques and the results are very promising. It is proved by many research groups that it is enough to use shapes of road signs to detect them. One of the points supporting the use of shape information for road signs recognition is the lack to standard colours among the countries. In situations in which it is difficult to extract colour information such as twilight time and night time, shape detection will be a good alternative. Using shapes to detect road and traffic signs has certain properties and it increases the robust in detection.

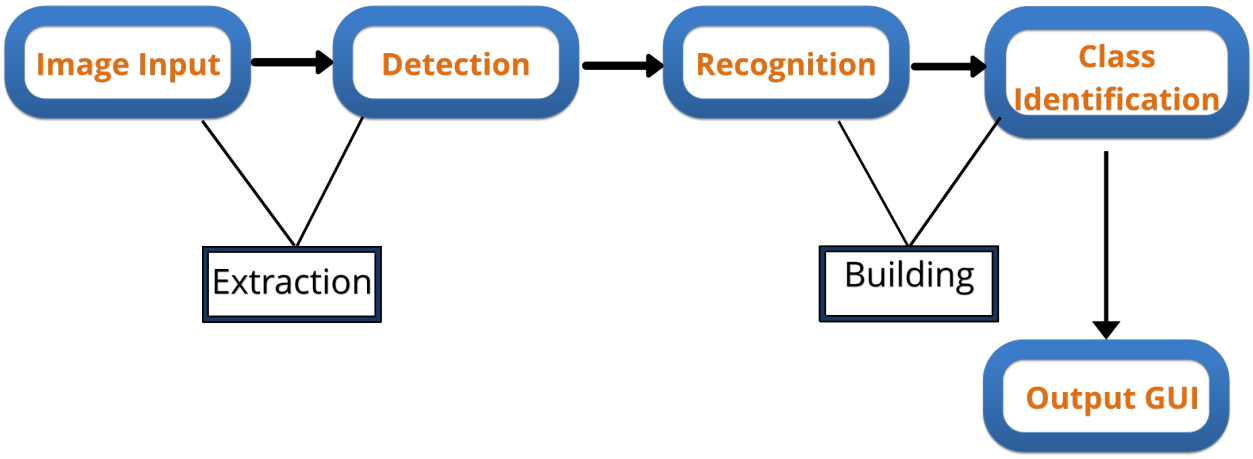

Figure 10: Pipeline for Traffic Signs Recognition

5.2 BUILDING

The Building module consists of 2 phases: one is Traffic Signs Recognition and Traffic Signs Class Identification.

Traffic Signs Recognition and Class Identification:

Generally, word recognition is used to point out that a sign is identified while word classification is invoked to indicate that the sign is assigned in a certain category based on certain features. The output of the detection stage is a list of candidate objects that could be probable road signs. This list is forwarded to the recognizer for further evaluation, and then to the classifier to decide whether the objects in the list are road signs, and in this case the classifier responds with a sign code.

Firstly, it should be robust to the geometrical status of sign, such as the vertical or horizontal orientation, the size, and the position of the sign in the image. Secondly, it should be robust to noise. Thirdly, the recognition should be carried out quickly if it is designed for real time applications. Furthermore, the classifier must be able to learn a large number of classes and as much a priori knowledge about road signs should be employed into the classifier design, as possible.

Neural networks are a suitable alternative for recognition and class Identification of road signs. There are two distinct advantages of using neural networks. First, the input image does not have

to be transformed into another representation space. Second, the result of the class identification depends only on the correlation between the network weights and the network itself if the network topology is assumed to be chosen from the beginning.

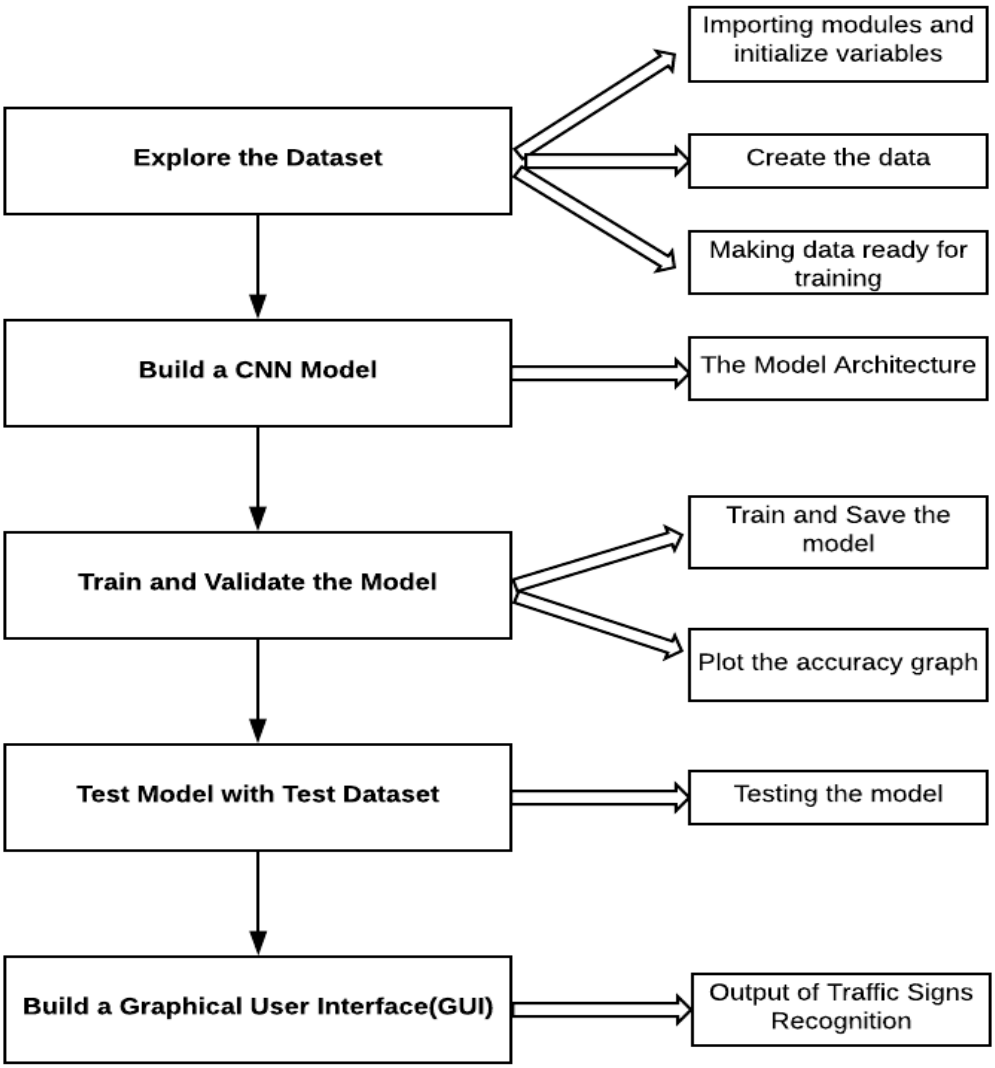

Figure 11: Steps for Traffic Signs Recognition

Building a Graphical User Interface

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.

The loaded trained model 'trafficsignrecognition.h5' using Keras. And then we build the GUI for uploading the image and a button is used to classify which calls the classify() function. The classify() function is converting the image into the dimension of shape (1, 30, 30, 3). This is - because to predict the traffic sign we have to provide the same dimension we have used when building the model. Then we predict the class, the model.predict_classes(image) returns us a number between (0-42) which represents the class it belongs to.

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. We will then use the dictionary to see the name of the class.

Figure 12: Graphical User Interface

6. IMPLEMENTATION

6.1 METHODOLOGY

Street security and traffic management is a very accusatory current issue and a theme dive for research by experts over the globe. There can be various incitements prompting these disasters like poor street upkeep, neglectful driving, mental condition of driver, casual attitude of pedestrians. Another major reason prompting to this may be the poor law implementation and improvised upkeep of street traffic signs. Blocked or then again decayed signs may delude the driver. One of the approaches involves four stages to detect traffic signals. Those are Image procurement (remove deburring), Colour segmentation (to detect colour), Blob detection (region & shape) and Classification using multiple neural network (high accuracy).

One more methodology focuses on the recognition and classification of traffic signs dependent on the investigations on traffic signs abroad and home which likewise incorporate the present state, mechanical issues and advancement inclination. Joined with the noteworthy highlights of the shape and inward structure of the traffic signs, three new components put together algorithms with respect to street signs ID have been actualized, including image pre-treatment, include feature extraction and classifier. The feature extraction work of traffic signs is led with the algorithms on Convolutional Neural Networks. Convolutional Neural Networks has two exclusive points of interest of extracting feature. One is known as local perception vision. It is commonly viewed that individual's vision of the outside world is from local to global.

With respect to an image, the factual properties of one section are equivalent to others. It implies that we can apply the attributes we learned in one section to the others. So for every position in an image, we can include the same learning qualities. From that point onward, we can include different convolutional kernels, adapting arrows are classified: straight, turn-left, turn-right, straight or turn-left. Based on some guidelines, we pick 70% images as the training samples and the remaining 30% images as the testing samples. The samples are partitioned into more types of feature. Different images produced by different convolutional kernels can be viewed as the different channels of an image.

Import the necessary modules and initialize variables

First import the os module, numpy, pandas, matplotlib, pil, keras and we will explain the role of each function when we use them. We also created some global variables data, labels which is an empty list in which we will store the data and labels. Explore the train folder in which you will find that there are 43 different classes. The cur directory variable will hold the absolute path of the project file.

Create the data

An image is made up of pixels and each pixel has 3 values to specify its colour i.e. RGB. In order for machines to understand the image, we have to convert the image into numbers. For this purpose, we use the PIL library that can perform many image manipulation tasks. If you have observed clearly then you will see that the images are of different width and heights. So we also have to resize all the images to a fixed size like 30x30. Traverse through all the classes, open the image using pil and also resize the image to 30x30 dimensions. Then we will append the data and label in the data and label list respectively. After the finish, the shape of our data and label as (39209, 30, 30, 3) and (39209,)

Making data ready for training

While training a model, it is important to provide random inputs of different classes to the model so that the model can generalize better. That is why we are going to use the sklearn train_test_split( ) function that will randomly split the data into training and validation set. After splitting the data the shape of training and testing sets are-

Shape of x_train: (31367, 30, 30, 3) and y_train: (31367,)
Shape of x_test: (7842, 30, 30, 3) and y_test: (7842,)

Data augmentation

A big limitation of deep neural networks is that they may have millions of parameters, tuning which requires a vast data set. This however is not always possible. In such cases, data

augmentation helps us generate additional training examples. We will generate additional data samples by applying affine transformation to the image. Affine transformations refer to transformations that do not alter the parallelism of lines, i.e. can be represented as a linear operation on the matrix. We will specifically use rotation, shearing and translation to simulate the effect of viewing the sign from different angles and different distances.

Model Architecture

The first module in the model is comprised of 3x3 filters. These filters have the effect of changing colour maps. In most applications, changing colour map can result in significant improvements in performance. However, it is not clear what the best colour map is for different applications, therefore using 3x3 filters results in a case were the network itself ends up choosing the best colour map.

The next 3 modules are composed of 32, 64 and 128 (respectively) 3X3 filters followed by maxpooling and dropouts. The output from each of the convolution module is fed into a feedforward layer. Rationale being that the fully connected layer has access to outputs from low level and higher level filters and has the ability to choose the features that works the best. The feedforward layers are composed of hidden layers with 1024 neurons. Additional dropout layers are applied after the fully connected layers.

The idea of using drop outs heavily is to avoid overfitting and force the network to learn multiple models for the same data. The last softmax layer is used to compute the log-loss of model prediction.

Training and saving the model

The model is defined and the data is ready. To start the training of our model we use the model.fit() function which takes the training set, validation set, batch size and no of epochs. After training the model we save the model.

Plot the accuracy graph

With the help of matplotlib functions, we will plot the graph of training and validation accuracy.

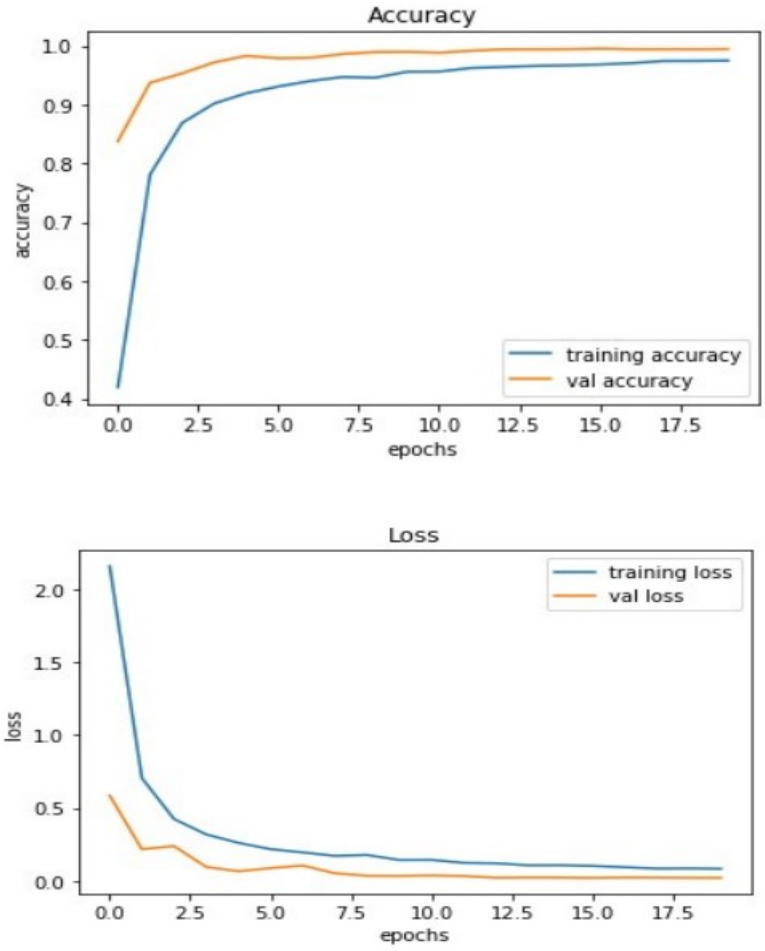

Figure 13: Graph plotting

Testing the model

To test our model we have a test folder that contains around 12,000 images. The test.csv file contains the path of the image along with the label of the class. Pandas is a great library to extract path and label from the CSV file and then with the help of sklearn accuracy_score() function, we can compare the real values with the predicted values of our model.

Source Code

Explore the dataset

#Importing all the necessary modules and initialize variables

Import the os module, numpy, pandas, matplotlib, pil, cv2 and we will explain the role of each function when we use them. We also created some global variables data, labels which is an empty list in which we will store the data and labels.

```
import os

import cv2

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from PIL import Image

import tensorflow as tf

from sklearn.model_selection import train_test_split

data=[]

labels=[]

height = 30

width = 30

channels = 3

num_classes = 43
```

```
n_inputs = height * width*channels
```

 # The dataset has folders from 0–42 i.e. 43 classes.

The dataset consists of different classes like stop, yield, speed limit, intersection and many more.

```
for i in range(num_classes) :
    path = "C:\\Users\\lenovo\\Downloads\\TrafficSignsRecognition\\Train\\{0}/".format(i)
    print(path)
    Class=os.listdir(path)
```

#iterating on all the images of the index folder.

Traverse through all the classes, open the image using Pillow and also resize the image to 30x30 dimensions. Then append the data and label in the data and label list respectively.

```
for a in Class: try
            image=cv2.imread(path+a)
            image_from_array = Image.fromarray(image, 'RGB')
            size_image = image_from_array.resize((height, width))
            data.append(np.array(size_image))
            labels.append(i)
        except AttributeError:
            print(" ")
x_train = np.array(data)
```

```
x_train = x_train/255.0
```

 # Applying Transformations.

Transformations are applied using image data generator with rotation, shift and shear range.

```
from keras.preprocessing.image import ImageDataGenerator

from keras.utils import to_categorical

y_train = np.array(labels)

y_train = to_categorical(y_train, num_classes)

X_train,X_valid,Y_train,Y_valid=train_test_split(x_train,y_train,test_size = 0.3,random_state=0)

datagen = ImageDataGenerator(featurewise_center=False, featurewise_std_normalization=False,

    width_shift_range=0.1, height_shift_range=0.1, zoom_range=0.2, shear_range=0.1,

            rotation_range=10.) datagen.fit(X_train)
```

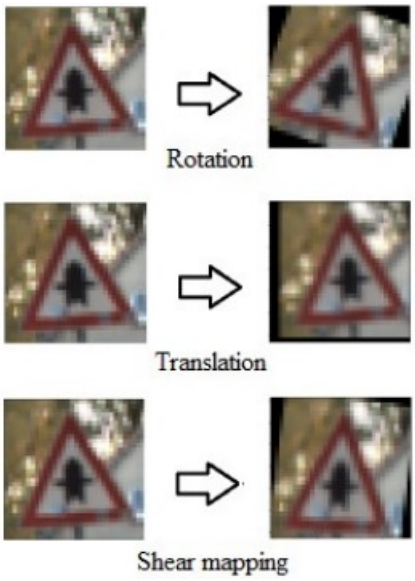

Figure 14: Geometric transformation for data augmentation

#one hot encoding the labels

The list of labels ranges from 0 to 42 that represent each category but the neural network needs a different format that is one hot encoding. One hot encoding is a vector representation where all elements of the vector are 0 except one, which has 1 value.

```
y_train=np.array(labels)

y_train = to_categorical(y_train, num_classes)

X_train,X_valid,Y_train,Y_valid= rain_test_split(x_train,y_train,test_size = 0.3,random_state=0)

print("Train :", X_train.shape)

print("Valid :", X_valid.shape)
```

#Showing images of trained data

The images in the train data set are shown by using show() method.

```
def show_images(images, labels, amount):

    for i in range(amount):

        index = int(random.random() * len(images))

        plt.axis('off')

        plt.imshow(images[index])

        plt.show()

        print("Size of this image is " + str(images[index].shape))

        print("Class of the image is " + str(labels[index]))
```

```
print("Train images")

show_images(X_train, Y_train, 3)
```

Figure 15: Images of trained data

The Model Architecture

#Build a CNN Model

A Convolutional Neural Network (CNN) is made up of convolutional and pooling layers. At each layer, the features from the image are extracted that helps in classifying the image.

We have also used the dropout layer which is used to handle the overfitting of the model. The dropout layer drops some of the neurons while training but not when we are predicting. We compile the model with categorical_crossentropy because our dataset has multi classes to be classified.

```
import keras

from keras.utils import to_categorical

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten
```

```
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

def cnn_model():

    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=X_train.shape[1:]))

    model.add(BatchNormalization())

     model.add(Dropout(0.5))

    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))

    model.add(BatchNormalization())

    model.add(Dropout(0.5))

    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

    model.add(BatchNormalization())

    model.add(Dropout(0.5))

    model.add(Conv2D(128, kernel_size=(5, 5), activation='relu'))

    model.add(BatchNormalization())

    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.5))

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

```
    return model

model = cnn_model()

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
```

Train and Validate the model

#Train and save the model

The model is defined and the data is ready. To start the training of our model we use the model.fit() function which takes the training set, validation set, batch size and no of epochs.

After training the model for 20 epochs we will save the model in a traffic_recognition.h5 file.

```
epochs = 20

history = model.fit(X_train, Y_train, validation_data=(X_valid, Y_valid), batch_size=32,

            epochs = epochs,verbose=1)

model.save('Traffic_Recognition.h5')
```

#Plot the accuracy graph

With the help of matplotlib functions, we will plot the graph of training and validation accuracy.

```
plt.figure(0)

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

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

plt.title('Accuracy')
```

```
plt.xlabel('epochs')

plt.ylabel('accuracy')

plt.legend()

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()
```

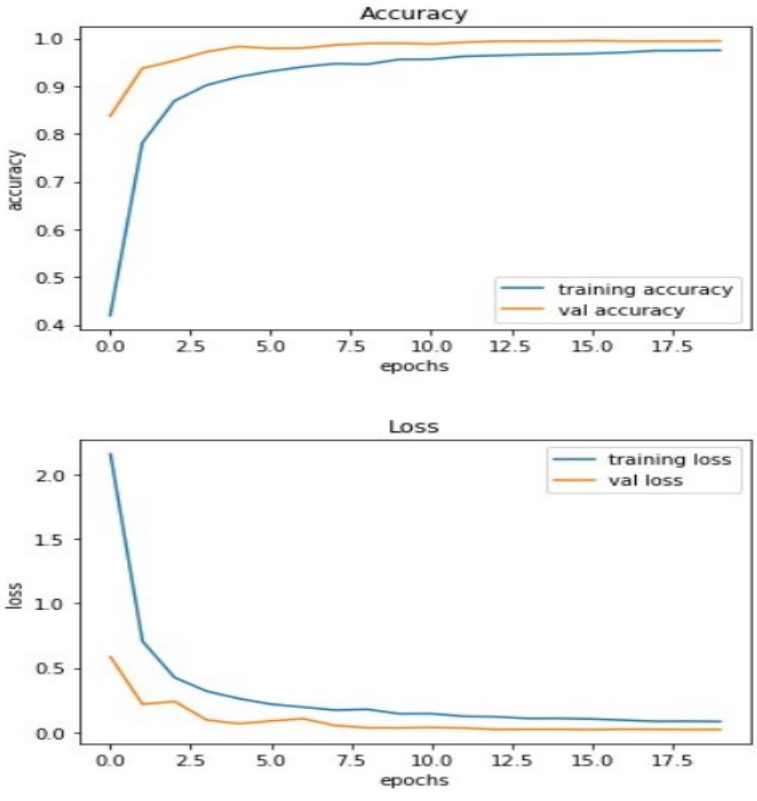

Figure 16: Accuracy graph plotting

Test model with Test Dataset

#Testing the model

To test our model we have a test folder that contains around 12,000 images. The test.csv file contains the path of the image along with the label of the class.

```
y_test=pd.read_csv("C:\\Users\\lenovo\\Downloads\\TrafficSignsRecognition\\Test.csv")

labels=y_test['Path'].values

y_test=y_test['ClassId'].values data=[]

for f in labels:

image=cv2.imread("C:\\Users\\lenovo\\Downloads\\TrafficSignsRecognition\\Test/"+f.replace('Test/', ''))

image_from_array = Image.fromarray(image, 'RGB')

size_image = image_from_array.resize((height, width)) data.append(np.array(size_image))

X_test=np.array(data) X_test = X_test.astype('float32')/255

pred = model.predict_classes(X_test)
```

#Accuracy with test data set

Pandas is a great library to extract path and label from the CSV file and then with the help of sklearn accuracy_score() function, we can compare the real values with the predicted values of our model.

```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, pred)
```

Building a GUI

#Importing necessary modules

The Tkinter is an inbuilt library of python to make a graphical user interface.

```
import tkinter as tk

from tkinter import filedialog

from tkinter import *

from PIL import ImageTk, Image

import numpy from keras.models import load_model
```

#dictionary to label all traffic signs class

This dictionary consists of 43 different classes of traffic signs.

```
model=load_model('C:\\Users\\lenovo\\Downloads\\TrafficSignsRecognition\\Traffic_Recognitionh5')

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 veh over 3.5 tons',

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

13:'Priority road',

14:'Yield',

15:'Stop',

16:'No vehicles',

17:'Veh > 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 veh > 3.5 tons' }
```

```
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)
  pred = model.predict_classes([image])[0]
  sign = classes[pred+1]
  print(sign)
  result.configure(text=sign)
def show_classify_btn(file_path):
classify_b=Button(top,text="ClassifyImage",command=lambda:classify(file_path),padx=10,pady=5)
classify_b.configure(bg='#364156', fg='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
    result.configure(text='')
    show_classify_btn(file_path)
except: pass
if __name__=="__main__":
```

 # Initialize GUI

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.

```
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign recognition')
top.configure(bg='#f9f6f7')
heading = Label(top, text="Traffic sign recognition",pady=20, font=('arial',20,'bold'))
heading.configure(background='#f9f6f7',fg='#364156')
heading.pack()
result=Label(top, font=('arial',15,'bold'))
result.configure(fg='#011638',bg='#f9f6f7')
```

```
sign_image = Label(top)

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

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

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

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

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

top.mainloop()
```

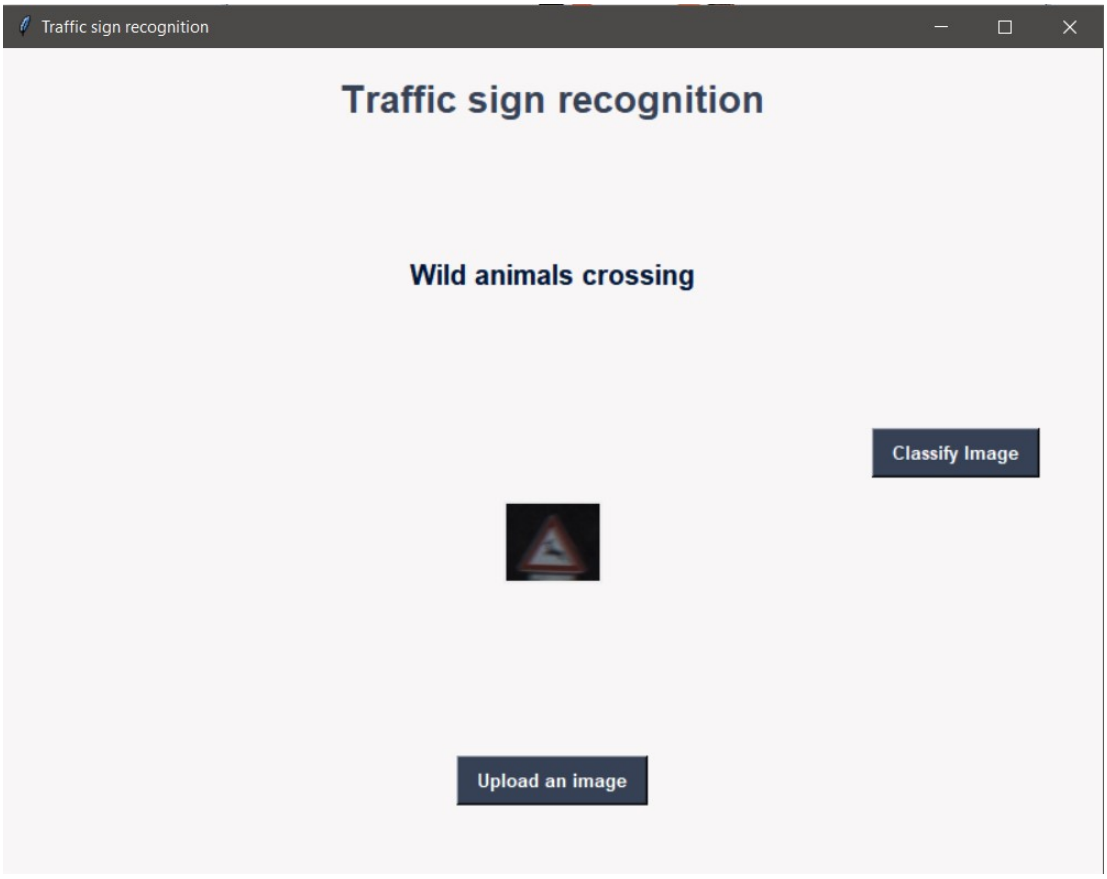

Figure 17: Traffic Signs Recognition Output

7. TESTING

Outputs for Traffic Signs Recognition

Here the user has to upload an image by clicking on upload an image button. Then the user has to click the classify image button, that will take the image data as input and provide us the class our image belongs to.

1. Below, is the output image of traffic sign representing 50km/h speed limit. This image is taken from Traffic Sign data set, which is stored in testing data and trained using training data set.

Figure 18: TSR output for Speed Limit

2. Below, is the output image of traffic sign representing Yield. This image is taken from Traffic Sign data set, which is stored in testing data and trained using training data set.

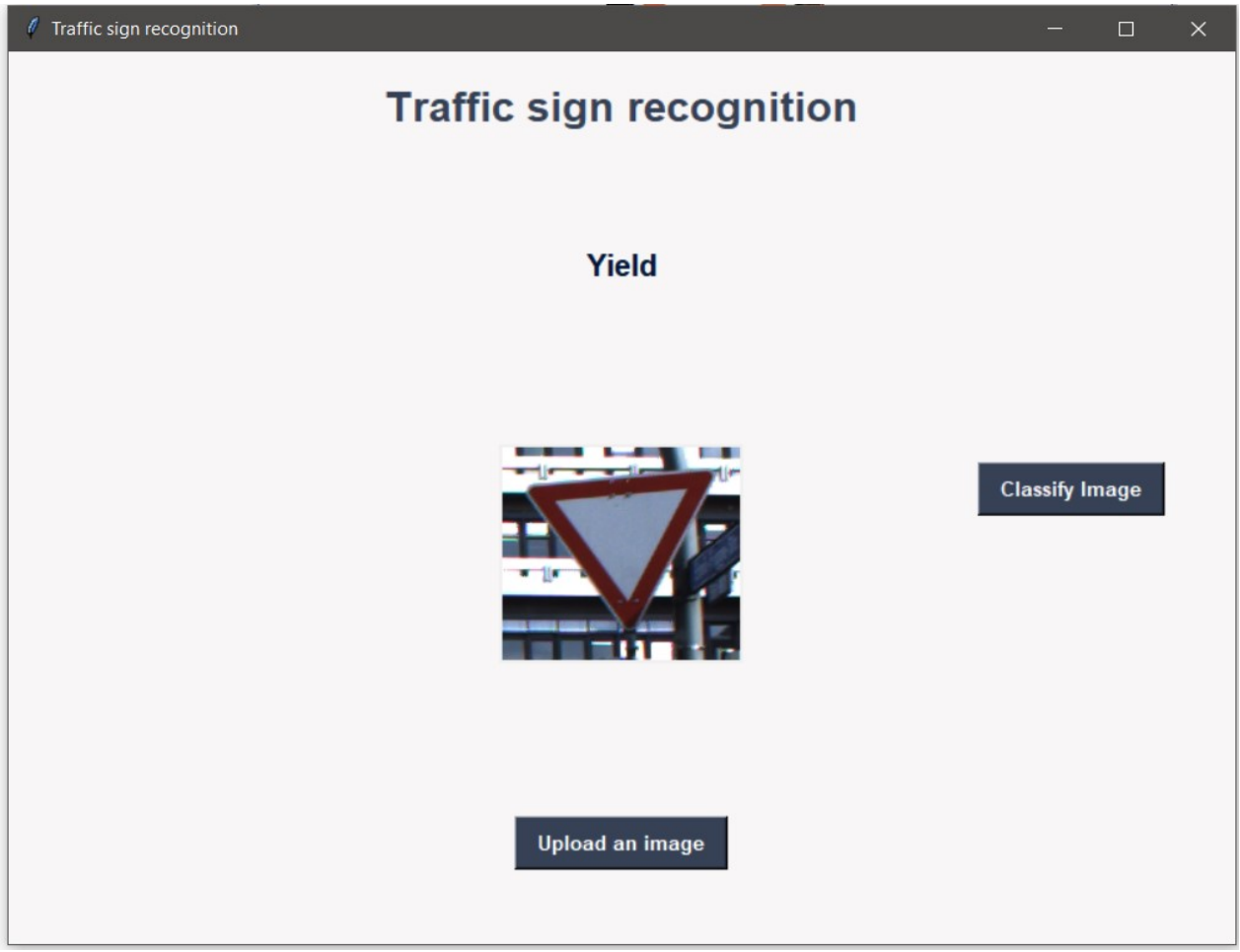

Figure 19: TSR output for Yield

3. Below, is the output image of traffic sign representing Road work. This image is taken from Traffic Sign data set, which is stored in testing data and trained using training data set.

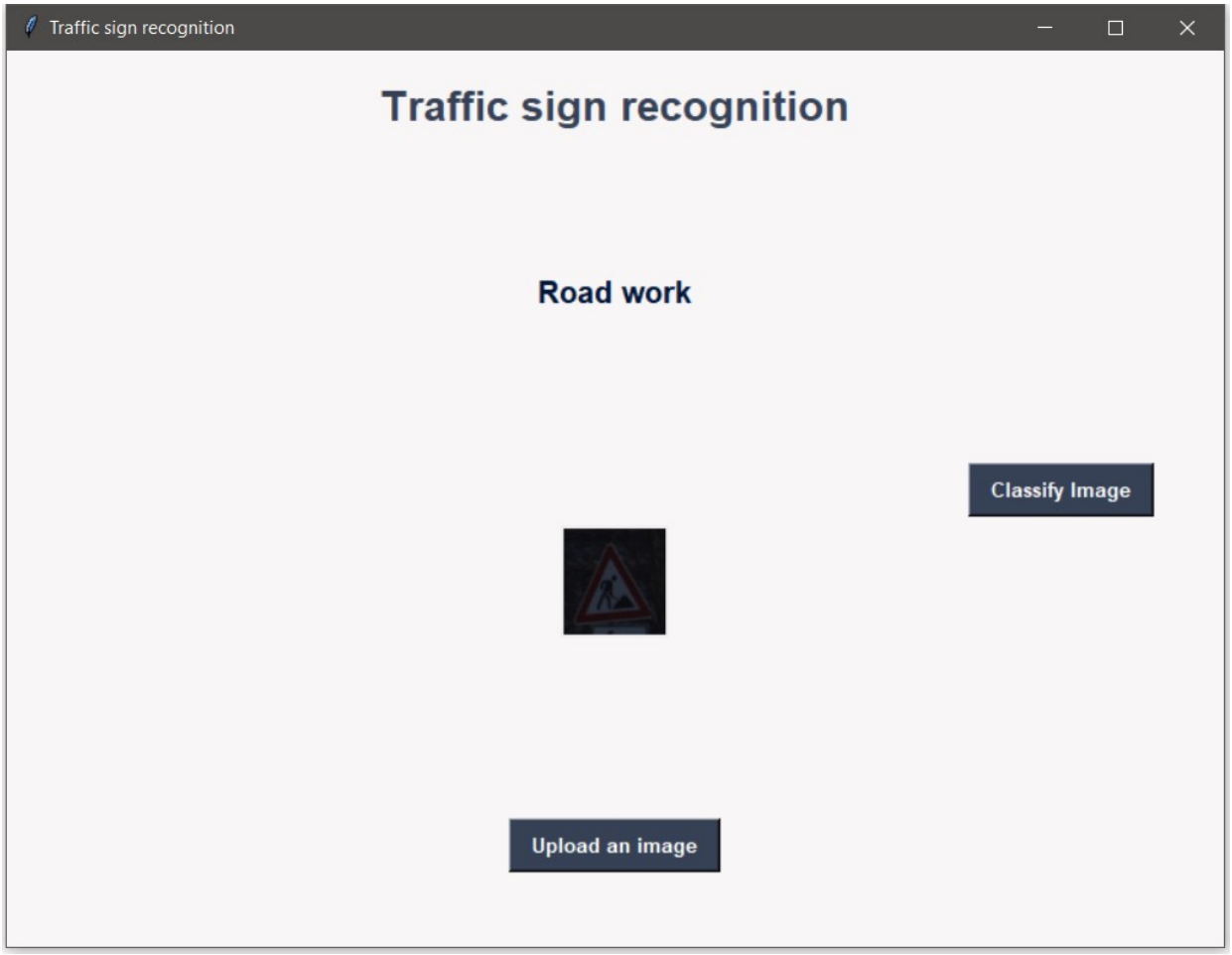

Figure 20: TSR output for Road Work

4. Below, is the output image of traffic sign representing No entry. This image is taken from Traffic Sign data set, which is stored in testing data and trained using training data set.

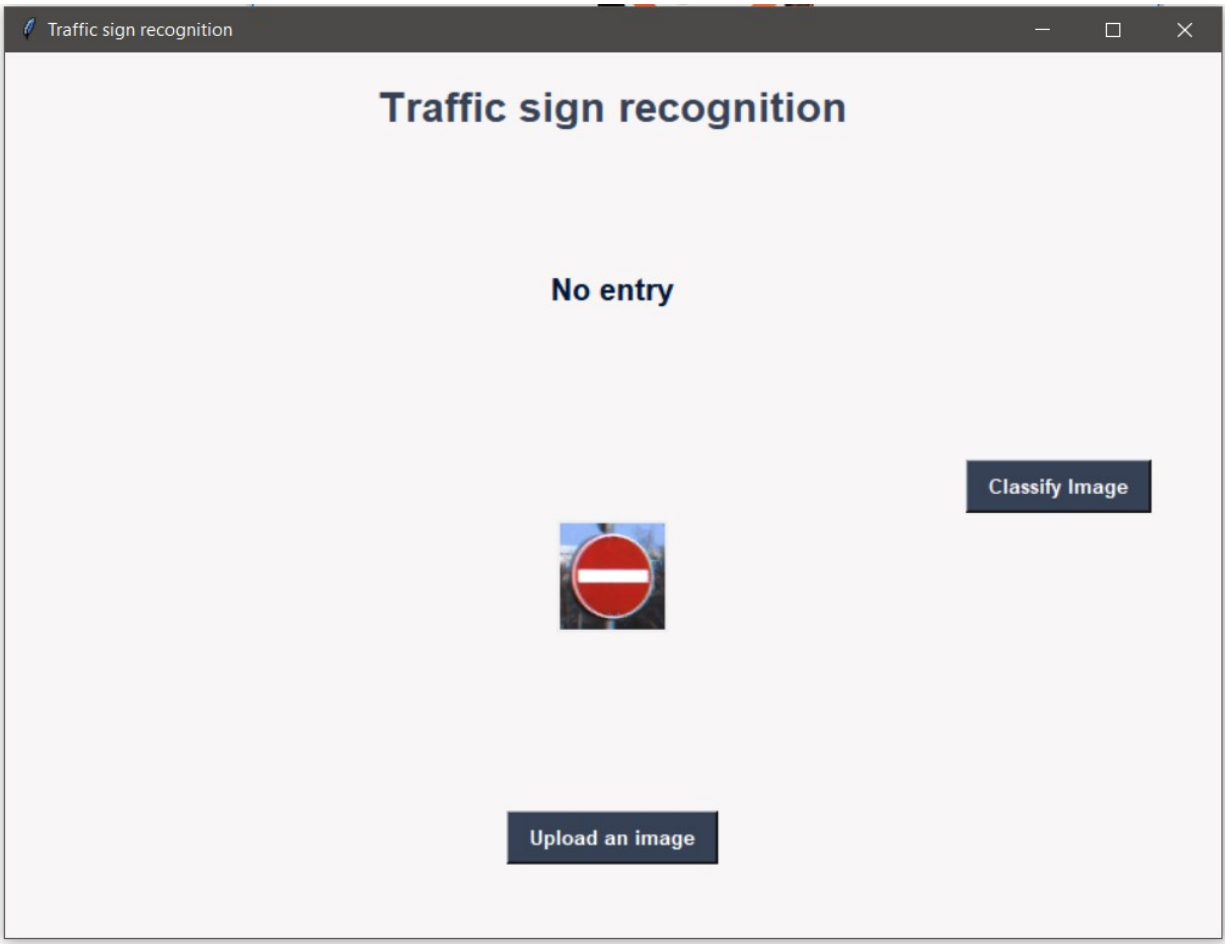

Figure 21: TSR output for No Entry

8. CONCLUSION AND FUTURE SCOPE

8.1 CONCLUSION

This system provides approach for detecting road traffic signs with quality output. It includes efficient feature extraction methods which results in appropriate outcomes. The CNN model is the finest techniques of Deep Learning which ensures accuracy in the achieved output. The Adam method incorporates all the aspects of CNN. The model we have used gives the test accuracy of 98%. This algorithm has a best speculation, and it can be trusted that it is used to identify more conventional traffic signs. Correct traffic sign detection is essential for accurate classification because if particular sign is not detected properly it cannot produce proper result.

8.2 FUTURE SCOPE

In future, we can increase the number of classes and we can include other sign containing different colours and shapes into account. A text to speech module can also be added to this application. In the current application, the driver would have to read the text printed on the classified sign, but with the help of a voice module, more comfort is guaranteed. The overall performance could also be improved and customized with the help of more data sets and from different countries.

9. REFERENCES

[1] Traffic signs recognition with deep learning - IEEE Conference Publication - https://ieeexplore.ieee.org/document/8652024

[2] https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/data - GTSRB - German Traffic Sign Recognition Benchmark | Kaggle

[3] https://www.irjet.net/archives/V6/i5/IRJET-V6I5440.pdf

[4] An Automatic Traffic Sign Detection and Recognition System Based on Color Segmentation, Shape Matching, and SVM

[5] Paper_93-Traffic_Sign_Detection_and_Recognition.pdf

[6] https://www.diva-portal.org/smash/get/diva2:523372/FULLTEXT01.pdf

[7] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. Lopez-Ferreras, "Road-sign detection and recognition based on support vector ma- ´ chines," IEEE transactions on intelligent transportation systems, vol. 8, no. 2, pp. 264–278, 2007.

[8] M. A. Garcia-Garrido, M. A. Sotelo, and E. Martin-Gorostiza, "Fast traffic sign detection and recognition under changing lighting conditions," in IEEE Intelligent Transportation Systems Conference, pp. 811–816, 2006.

[9] G. Loy and N. Barnes, "Fast shape-based road sign detection for a driver assistance system," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 1, pp. 70–75, 2004.

[10] C.-Y. Fang, S.-W. Chen, and C.-S. Fuh, "Road-sign detection and tracking," IEEE transactions on vehicular technology, vol. 52, no. 5, pp. 1329–1341, 2003.

[11] Y.-Y. Nguwi and A. Z. Kouzani, "Automatic road sign recognition using neural networks," in The IEEE International Joint Conference on Neural Network Proceedings, pp. 3955– 3962, 2006.