Capstone Project Report

Autonomous Vehicle Control Through Traffic Signs

BE Third Year- EE

Group No. 1

Submitted by:
Akanksha Singh – 101904104
Ayush Vashisth – 101904020
Gursimar Singh Nagpal – 101904114
Vinay Kumar Jain – 101904017
Yashvardhan Panchauri – 101904095

Supervisor(s):
Dr. Sanjay K. Jain, (Professor)
Dr. Vishal Srivastav, (Assistant Professor)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Department Of Electrical & Instrumentation Engineering

Thapar Institute of Engineering and Technology, Patiala

Feb-Dec 2022

1.1. INTRODUCTION

Autonomous vehicles are the vehicles of the future. And this fact is not hidden that, many developed countries have started replacing human-driven vehicles with autonomous vehicles on roads. Features like self-driving nature, environmentally friendly, automatic obstacle detection, less fuel consumption, time-efficient, robustness; etc are making them a hot topic of discussion at present time. But autonomous vehicles are still a distant dream for India. The Indian Motor Vehicles Act, 1988 and the rules, that regulate the operation of vehicles here, do not currently allow fully automated systems. This is because these vehicles will find most Indian roads too chaotic to function. This chaos can be due to the poor road condition, accumulation of vehicles in a single place beyond its capacity, or maybe due to some environmental cause. Negligence of drivers toward traffic signs, traffic signals and not to mention traffic rules sometimes causes critical accidents and therefore must be given serious attention. Moreover, in current traffic management systems, there is a high probability that the driver may miss some of the traffic signs on the road because of overcrowding due to neighboring vehicles. And not to miss, with the continuous growth of vehicle numbers in urban agglomerations around the world, this problem is only expected to grow worse. One of the ways to limit this problem is by using Traffic Sign Detection And Recognition (TSDR) System. A TSDR system is developed and implemented as a part of the Intelligent Transportation System (ITS), which aims to minimize the number of accidents due to drivers' negligence and wrong decisions. This system detects and recognizes traffic signs from and within images captures by cameras or imaging sensors displaying to the user what traffic rules are applicable at that stretch of road. Even though the technology warns the user about the detected traffic sign, there are chances that the user might violate them, and violating the traffic sign rules might lead to accidents where the life of the people would be at stake. Hence, to avoid encountering such conditions we would like to implement a – " TSDR system" on IOT-based hardware which will culminate as a proto-model of the real-life autonomous vehicle system that can support drivers and increase driving safety**.**

1.2. LITERATURE SURVEY

The first research on traffic sign recognition was back in 1987; Akatsuka and Imai tried to make a very basic traffic sign recognition system. A system capable of by itself recognizing other traffic signs and used as an assistance for drivers, telling them about the presence of some specific restriction or danger in speeding or construction work. It can be used to provide automatic recognition for specific traffic signs. Generally, the procedure of a traffic sign recognition system is divided into two parts, detection and, classification. A. Detection The aim of using traffic sign detection is to locate the regions of interest (ROI) in which a traffic sign is likely to be found. Cropping the excess space is what's required to get the main sign. This technique is callas ROI. ROI locates the traffic sign in based on its shape dimensions etc. The traffic signs are cropped which is useful. The background image is removed as it is not an area of interest. By this, we assume, that a large part of the image area can be ignored as not required. Traffic signs are designed with set colors and shape making them easier to differentiate and recognize. B. Classification In this part the sign is classified based on the type shape color and the information the sign is giving. Histogram of oriented gradients (HOG) Features: HOG decomposes an image into small squared cells, computes a histogram of oriented gradients in each cell, normalizes the result using a block-wise pattern, and return a descriptor for each cell. Many different techniques have been applied to detect traffic signs. Most of these techniques are based on using HOG and SIFT features.

Many research works were presented in the literature. Ohgushi etal. introduced a traffic signs classifier based on color information and Bags of Features (BoF) as a features extractor and a support vector machine (SVM) as a classifier. The proposed mothed struggle in recognizing the traffic signs in real conditions especially when the sign is intensively illuminated or partially occluded. Some research investigated the detection of the traffic sign without performing the classification process. Wu et al. proposed a method to detect only round traffic signs on the Chinese roads. On the other side, researchers focus on detecting and recognizing the traffic sign. The proposed method only detects round signs and cannot detect other signs' shapes. A three steps method to detect and recognize traffic signs was proposed by Wali et al. The first step was data preprocessing. The second was detecting the existence of the sign and the third was classifying it. For the detection process, they apply the color segmentation with shape matching and for the classification process, they use SVM as a classifier. The proposed method achieves 95.71% of accuracy. Lai et al introduced a traffic signs recognition method using a smartphone. They used color detection to perform color space segmentation and shape recognition methods using template matching by calculating the similarity. Also, an optical character recognition.

(OCR) was implemented inside the shape border to decide on the signing class. The proposed method was very limited to red traffic signs only. Gecer et al. propose to use color-blob-based COSFIRE to recognize traffic signs. The proposed method was based on a Combination of Shifted Filter Responses with computing the response of different filters in different regions in each channel of the color space (ie. RGB). A system based on a BoW descriptor enhanced using spatial histogram was used by Shamset al. to improve the classification process based on an SVM classifier.

1.3. NEED ANALYSIS

The major idea of the system is to increase automation which would result in the reduction of human error and an increase in reliability, efficiency, and speed. In current traffic management systems, there is a high probability that the driver may miss some of the traffic signs on the road because of overcrowding due to neighboring vehicles. With the continuous growth of vehicle numbers in urban agglomerations around the world, this problem is only expected to grow worse. A visual-based traffic sign recognition system can be implemented in the automobile with an aim of detecting and recognizing all emerging traffic signs. The same would be displayed to the driver with alarm-triggering features if the driver refuses to follow the traffic signs.

1.4. PROBLEM FORMULATION

Considering the object recognition and interpretation abilities of humans, it is a hard task to try to develop a computer-based system that should be able to support people in everyday life. There are a lot of conditions that are changing continuously such as luminance and visibility, which are handled by the human recognition system with ease but present serious problems for computer-based recognition. Looking at the problem of road and traffic sign recognition shows that the goal is well defined and it seems to be a simple problem. Road signs are located in standard positions and they have standard shapes, and standard colors and their pictograms are known. To see the problem in its full scale, however, a number of parameters that affect the performance of the detection system need to be studied carefully. Road sign images are acquired using a digital camera for the purpose of the current analysis. However, still, images captured from a moving camera may suffer from motion blur. Moreover, these images can contain road signs which are partially or totally occluded by other objects such as vehicles or pedestrians. Other problems, such as the presence of objects similar to road signs, such as buildings or billboards, can affect the system and make sign detection difficult. The system should be able to deal with traffic and road signs in a wide range of weather and illumination variant environments such as different seasons and different weather conditions e.g. sunny, foggy, rainy, and snowy conditions. Different potential difficulties are depicted in one section of this chapter. Using the system in different countries can make the problem even worse. Different countries use different colors and different pictograms. The system should also be adaptive, which means it should allow continuous learning otherwise the training should be repeated for every country. To deal with all these constraints, road sign recognition should be provided with a large number of sign examples to allow the system to respond correctly when a traffic sign is encountered.

1.4.1. What is Traffic Sign Recognition?

Traffic Sign Recognition is a field that is concerned with the detection and recognition of road and traffic signs in traffic scenes acquired by a camera. It is a technique that uses computer vision and artificial intelligence to extract the road signs from outdoor images taken in uncontrolled lighting conditions where these signs may be occluded by other objects and may suffer from different problems such as color fading, disorientation, and variations in shape and size.

1.4.2. Potential Difficulties

In addition to the complex environment of the roads and the scenes around them, signs can be found in different conditions such as aged, damaged, disoriented, etc hence the detection and recognition of these signs may face one or more of the following difficulties:

1. The color of the sign fades with time as a result of long exposure to sunlight, and the reaction of the paint with the air, Figure 1.4.2.1
2. Visibility is affected by the weather conditions such as fog, rain, clouds, and snow, as shown in Figure 1.4.2.2
3. Visibility can be affected by local light variations such as the direction of the light, the strength of the light depending on the time of the day and the season, and the shadows generated by other objects.
4. Colour information is very sensitive to the variations of light conditions such as shadows, clouds, and the sun. It can be affected by illuminant color (daylight), illumination geometry, and viewing geometry.
5. The presence of obstacles in the scene, such as trees, buildings, vehicles, and pedestrians even signs which occlude in other signs as given in Figure 1.4.2.3

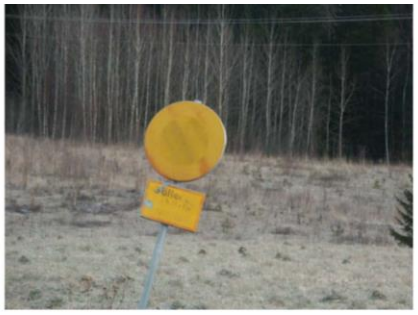


Fig. 1.4.2.1 Faded Sign

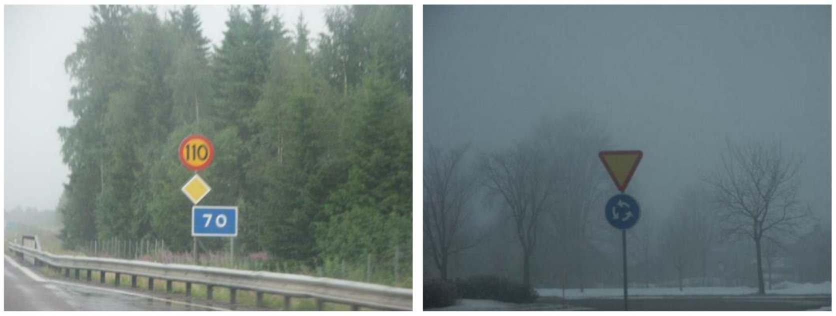


Fig. 1.4.2.2 Bad Weather Condition (Rain & Snow)

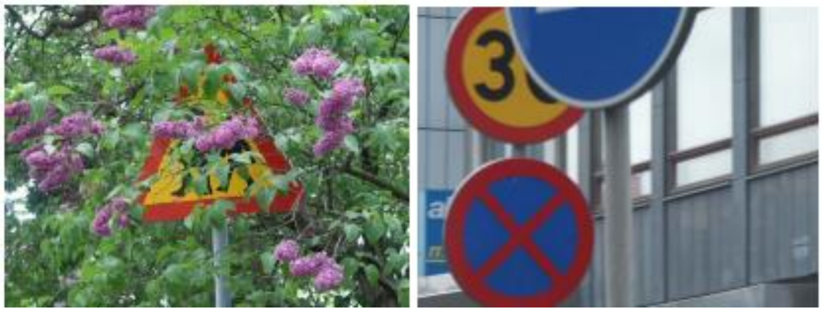


Fig. 1.4.2.3 The presence of obstacles in the scene

1.5. OBJECTIVES

All through this project, we aim to make a system that can be used universally for traffic sign detection and recognition, which in turn would help us learn more about deep learning and IoT.

The two phases of the objectives are:

1. To develop an automated model for the detection of Traffic Signs on the roadsides by using concepts like Deep Learning, Learning, Data Analysis & Image Processing
2. To implement a prototype of the Traffic Sign Detection & Recognition (TSDR) System with violation control on a miniature vehicle by using concepts of Communication and IoT.

1.6. METHODOLOGY

The framework we proposed is categorized into 2 components Hardware Part and the Software Part. The build of the system is majorly concentrated on a cost-effective, out-of-the-box solution using a mini embedded computer Raspberry Pi. To provide fast processed results deep learning techniques have to be used with the help of the TensorFlow and Keras platforms. The major work in the proposed prototype is the detection of traffic signs using Convolutional NeuralNetwork like speed limits 40 and 60, No left, No right, and No U-turn. If the traffic signs have been detected by the CNNmethod, then the detected signs are encrypted into specific binary codes and they are logged in IoT cloud data. A NodeMCU continuously receives the logged data and sends it to the Arduino UNO which resultantly controls the motors according to the control signal that has been sent to IC L293D. The entire block diagram for the proposed prototype is given in Figure 1.6.1

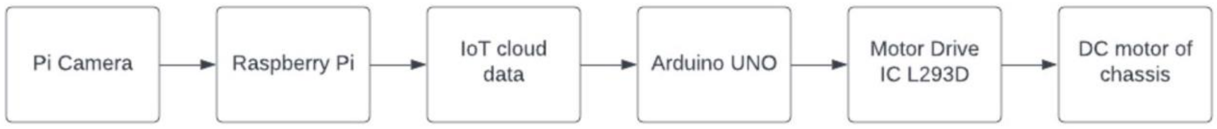


Fig. 1.6.1 Block diagram of Proposed Prototype

SOFTWARE IMPLEMENTATION:

We will build a customized deep neural network model that will be trained on the traffic signs dataset. The general workflow is explained with the help of a flowchart as shown in Figure 1.6.2. Later, during the implementation stage, the model will be used to detect real-time traffic signals.

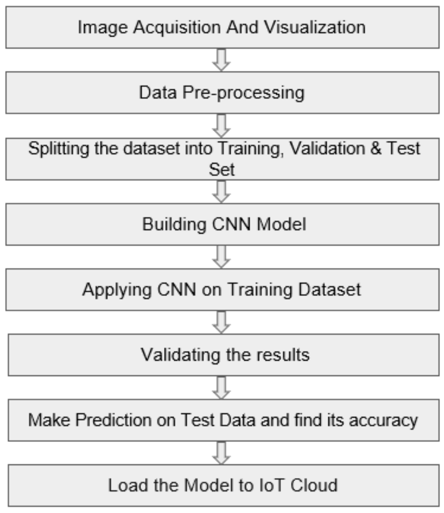


Image Acquisition And Visualization
Data Pre-processing
Splitting the dataset into Training, Validation & Test Set
Building CNN Model
Applying CNN on Training Dataset
Validating the results
Make Prediction on Test Data and find its accuracy
Load the Model to IoT Cloud

Fig. 1.6.2 Block diagram of Software Implementation

HARDWARE IMPLEMENTATION:

We will be using a pi camera connected to the Camera Serial Interface (CSI) connector of the Raspberry pi to capture the images. The images are captured in a continuous manner and every frame is processed by the already trained model to detect the traffic sign if present. And based on the result the message will be sent to the RC car. Here Arduino coding is been used for basic locomotion of the RC car, violation control of the car if so traffic sign is detected, and as well as for the NodeMCU to retrieve data in continuous from the IoT data log. A Traffic Sign Detection and Recognition (TSDR) system works in two stages. Detection- In this stage, the presence or absence of a traffic sign is detected. Recognition- In this stage, the detected traffic sign is recognized and classified. A prototype of the proposed system for the detection of Indian traffic signs is shown in Figure 1.6.3. The proposed scheme can be broadly divided into 5 stages:

1. Video and frame capturing
2. Preprocessing
3. Traffic sign detection
4. Character/icon extraction
5. Recognition

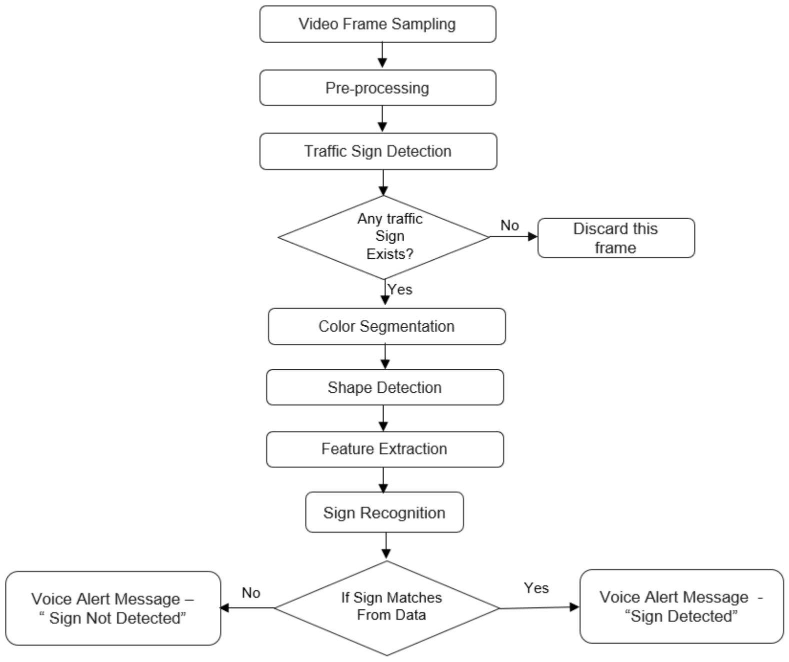


Fig. 1.6.3 Block diagram of Traffic Sign Detection

1.7. DELIVERABLES

1. The proposed proto-model will be able to detect traffic signs and will use the Voice Alert Message to notify the driver regarding the type of sign.
2. Furthermore, the proto-model will be tested keeping the following conditions into consideration while capturing images from Video frame sampling in case the traffic sign is detected. The conditions are perspective distortion, lighting changes, partial occlusions, and shadows.
3. In addition, it has to provide information about the presence of possible problems: Lack of visibility, Bad conditions. And Bad placement.

1.8. NOVELTY

The various components of methodology and deliverables, mentioned above are implemented and researched separately in several research papers and projects. Through this project, we want to incorporate all the separately researched attributes into a single system and present it in the form of IoT based Hardware proto-model.

1.9. CONCLUSION

These proto-model benefits are generally focused on driver convenience. But it has many other real-time applications. For example, it prevents accidents and traffic jams by controlling the speed of the car according to the speed limits, and by detecting traffic signs. Furthermore, it effectively controls traffic signals in a dynamically changing environment, and it’s barely affected by the traffic problem complexity.

2. WORK PROGRESS

2.1. SOFTWARE IMPLEMENTATION

We will build a customized deep neural network model that will be trained on the traffic signs dataset. The general workflow is explained with the help of a flowchart as shown in Figure.2.1.1. Later, during the implementation stage, the model will be used to detect real-time traffic signals.

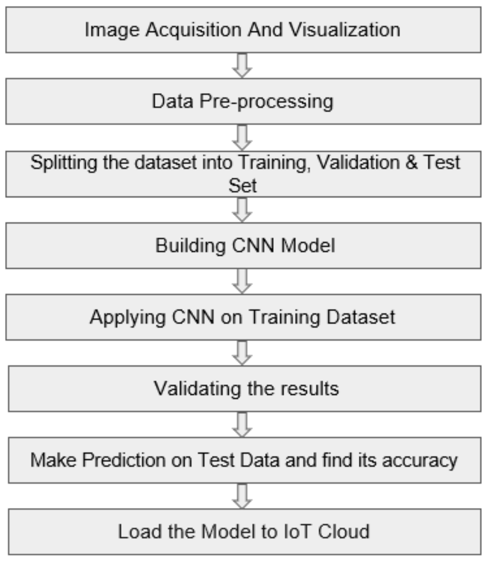


Image Acquisition And Visualization
Data Pre-processing
Splitting the dataset into Training, Validation & Test Set
Building CNN Model
Applying CNN on Training Dataset
Validating the results
Make Prediction on Test Data and find its accuracy
Load the Model to IoT Cloud

Fig. 2.1.1 Block diagram of Software Implementation

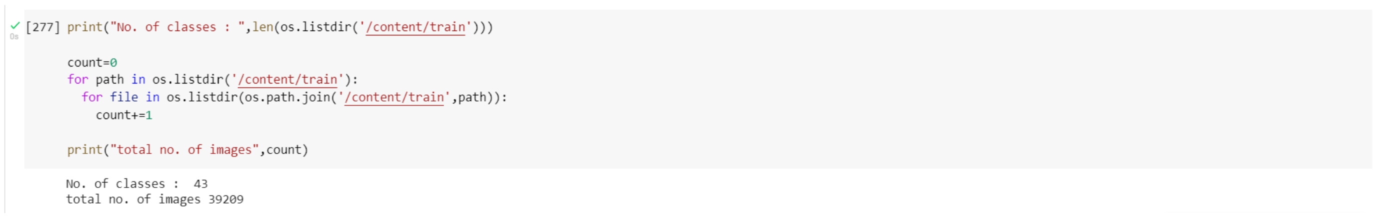
Step 1: IMAGE ACQUISITION AND VISUALIZATION

Dataset 1: GTSRB - German Traffic Sign Recognition Benchmark

The dataset is taken from the Kaggle repository.

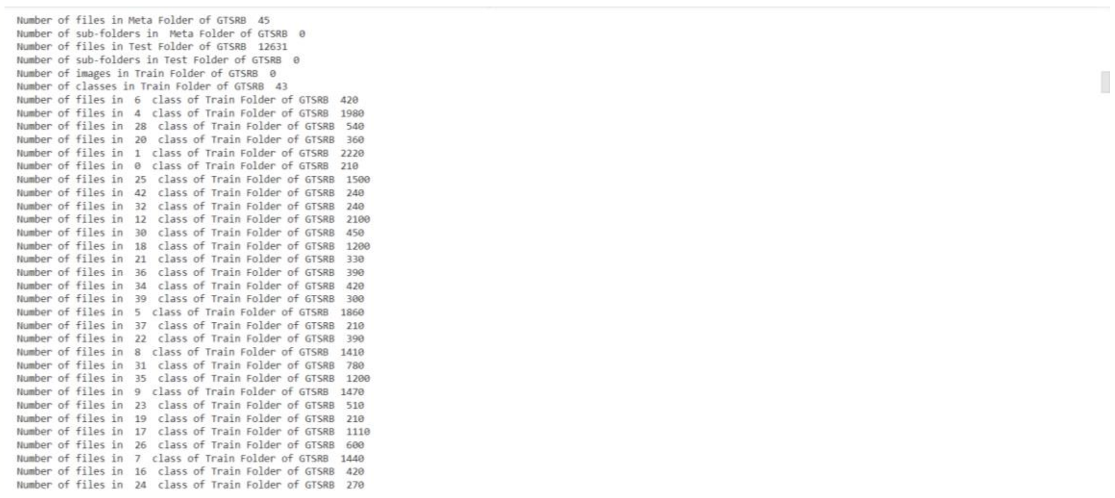
Link: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

Here is the dataset for classifying 43 different classes (Figure 2.1.4) of traffic signs with total images of 39209. In addition, there are additional 12631 images in the Test folder. The ***meta*** folder should have 43 different images (ranging from 0 to 42). The detail of each folder is mentioned in Figure.2.1.3 and the types of traffic signs are shown through a bar graph in Figure 2.1.4



```
[277] print("No. of classes : ",len(os.listdir('/content/train')))

      count=0
      for path in os.listdir('/content/train'):
        for file in os.listdir(os.path.join('/content/train',path)):
          count+=1

      print("total no. of images",count)

      No. of classes :  43
      total no. of images 39209
```

Fig. 2.1.2 Output of Number of classes and images in GTSRB Dataset



```
Number of files in Meta Folder of GTSRB  45
Number of sub-folders in  Meta Folder of GTSRB  0
Number of files in Test Folder of GTSRB  12631
Number of sub-folders in Test Folder of GTSRB  0
Number of images in Train Folder of GTSRB  0
Number of classes in Train Folder of GTSRB  43
Number of files in  6  class of Train Folder of GTSRB  420
Number of files in  4  class of Train Folder of GTSRB  1980
Number of files in  28  class of Train Folder of GTSRB  540
Number of files in  20  class of Train Folder of GTSRB  360
Number of files in  1  class of Train Folder of GTSRB  2220
Number of files in  0  class of Train Folder of GTSRB  210
Number of files in  25  class of Train Folder of GTSRB  1500
Number of files in  42  class of Train Folder of GTSRB  240
Number of files in  32  class of Train Folder of GTSRB  240
Number of files in  12  class of Train Folder of GTSRB  2100
Number of files in  30  class of Train Folder of GTSRB  450
Number of files in  18  class of Train Folder of GTSRB  1200
Number of files in  21  class of Train Folder of GTSRB  330
Number of files in  36  class of Train Folder of GTSRB  390
Number of files in  34  class of Train Folder of GTSRB  420
Number of files in  39  class of Train Folder of GTSRB  300
Number of files in  5  class of Train Folder of GTSRB  1860
Number of files in  37  class of Train Folder of GTSRB  210
Number of files in  22  class of Train Folder of GTSRB  390
Number of files in  8  class of Train Folder of GTSRB  1410
Number of files in  31  class of Train Folder of GTSRB  780
Number of files in  35  class of Train Folder of GTSRB  1200
Number of files in  9  class of Train Folder of GTSRB  1470
Number of files in  23  class of Train Folder of GTSRB  510
Number of files in  19  class of Train Folder of GTSRB  210
Number of files in  17  class of Train Folder of GTSRB  1110
Number of files in  26  class of Train Folder of GTSRB  600
Number of files in  7  class of Train Folder of GTSRB  1440
Number of files in  16  class of Train Folder of GTSRB  420
Number of files in  24  class of Train Folder of GTSRB  270
```

Fig 2.1.3 Number of images in different classes of GTSRB dataset

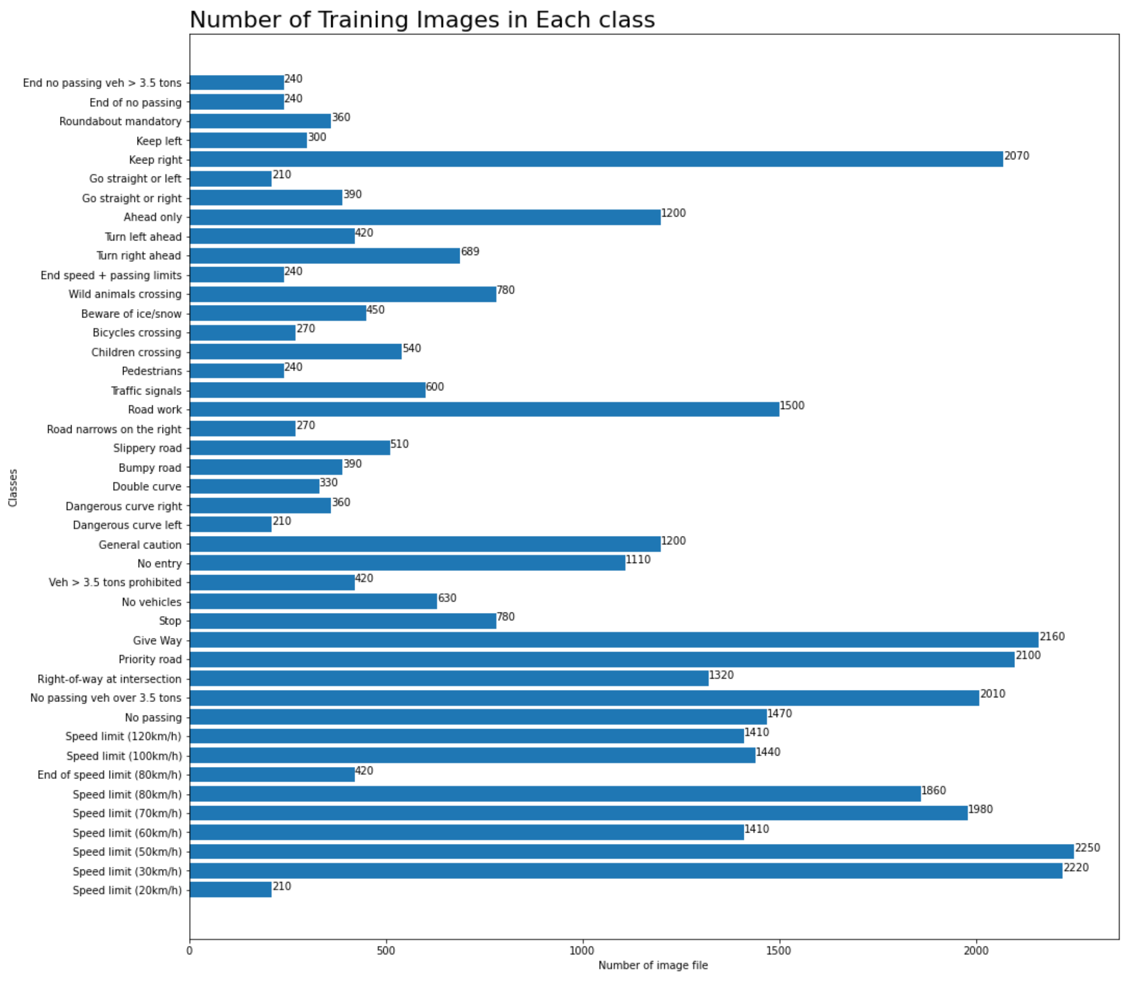


Fig. 2.1.4 Bar Graph showing the number of image files in each class of Traffic signs of GTSRB dataset

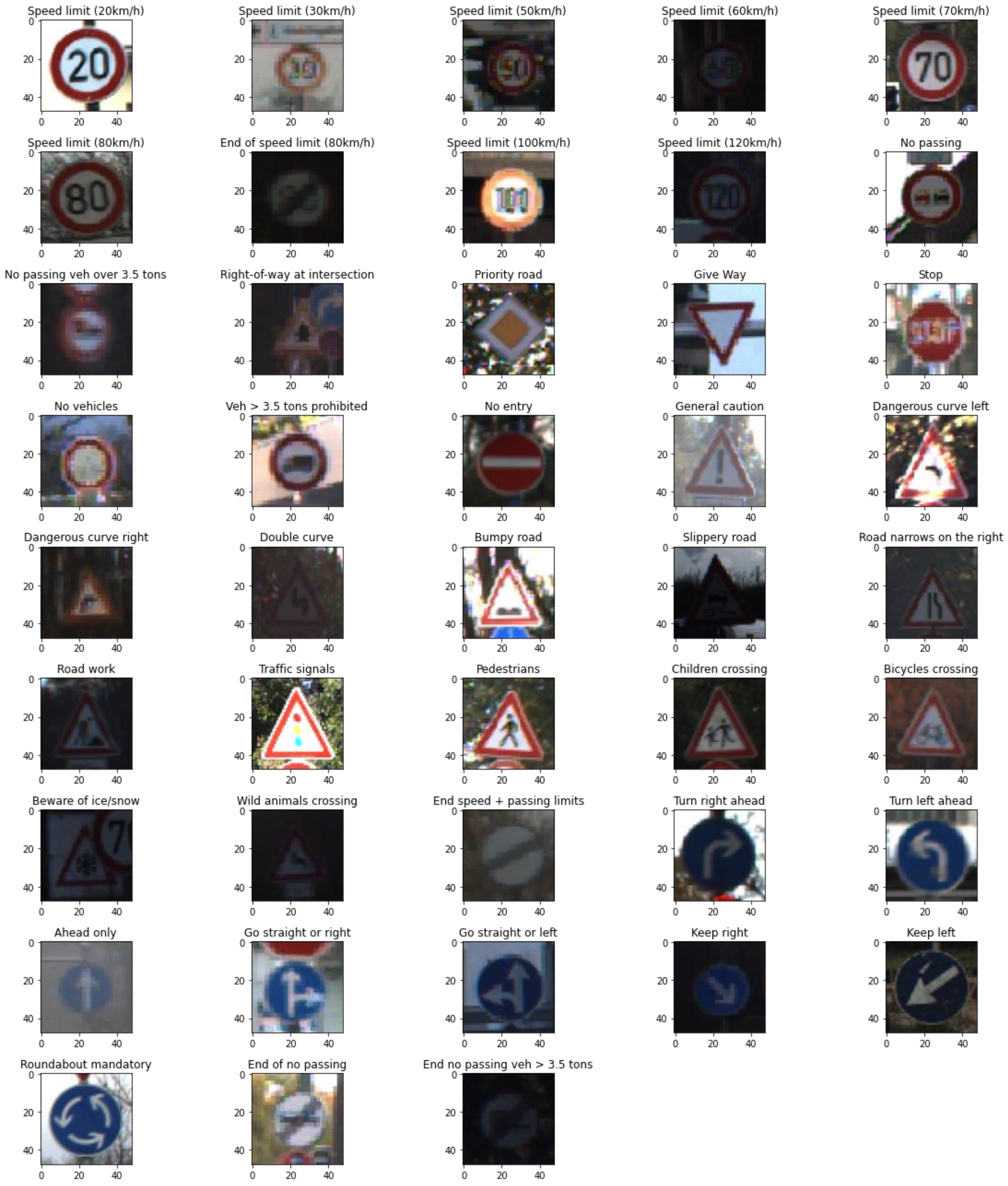


Fig. 2.1.5 Different classes in GTSRB dataset

<u>DATASET 2</u><u>: Traffic Sign Dataset – Classification</u>

The dataset is taken from the Kaggle repository.

Link: https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification

Here is the dataset for classifying 57 different classes (Figure 2.1.7) of traffic signs with total images of 4168. In addition, there are additional 1994 images for testing.

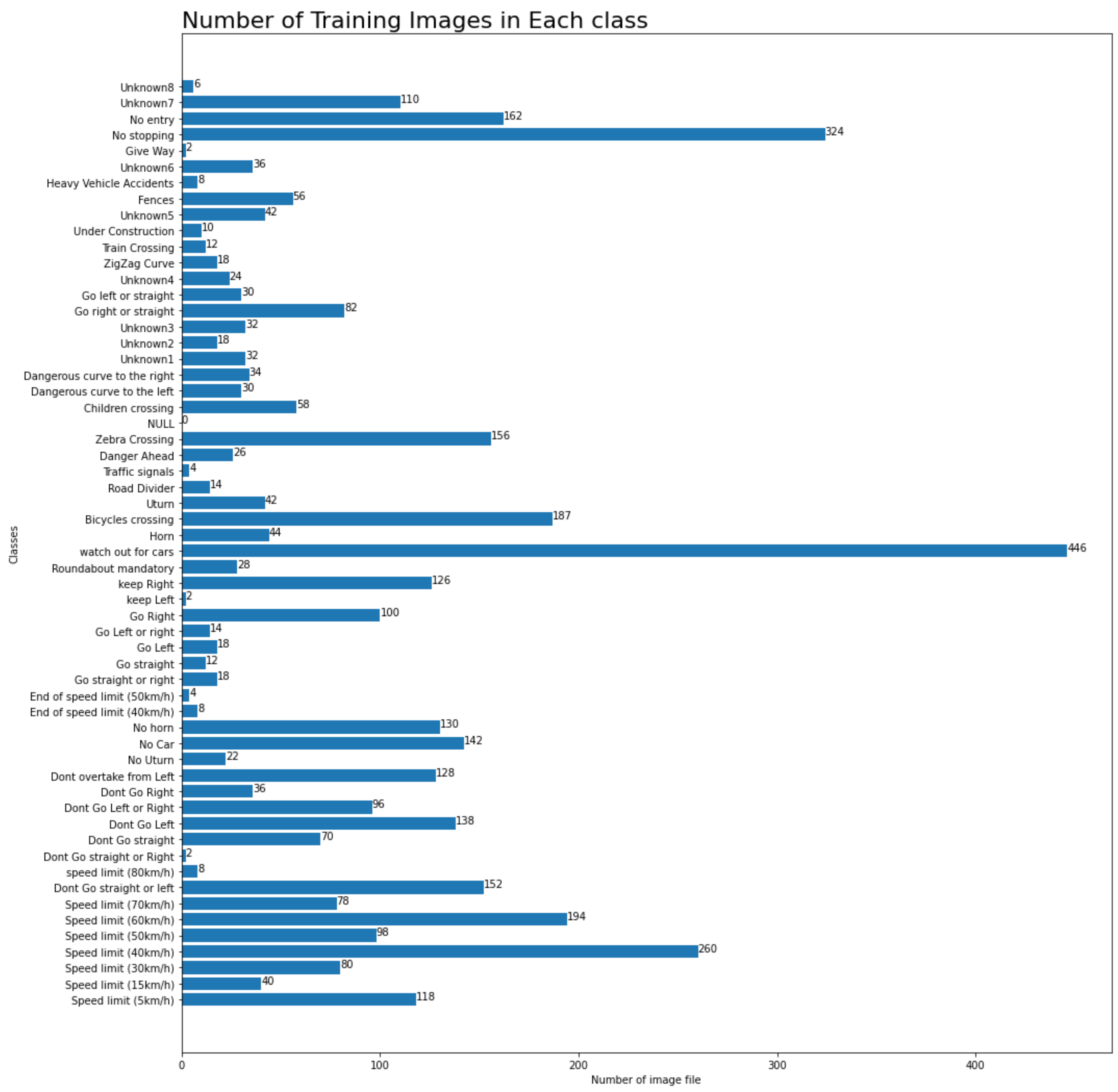


Fig. 2.1.6 Bar Graph showing the number of image files in each class of Traffic signs of Traffic Sign Dataset

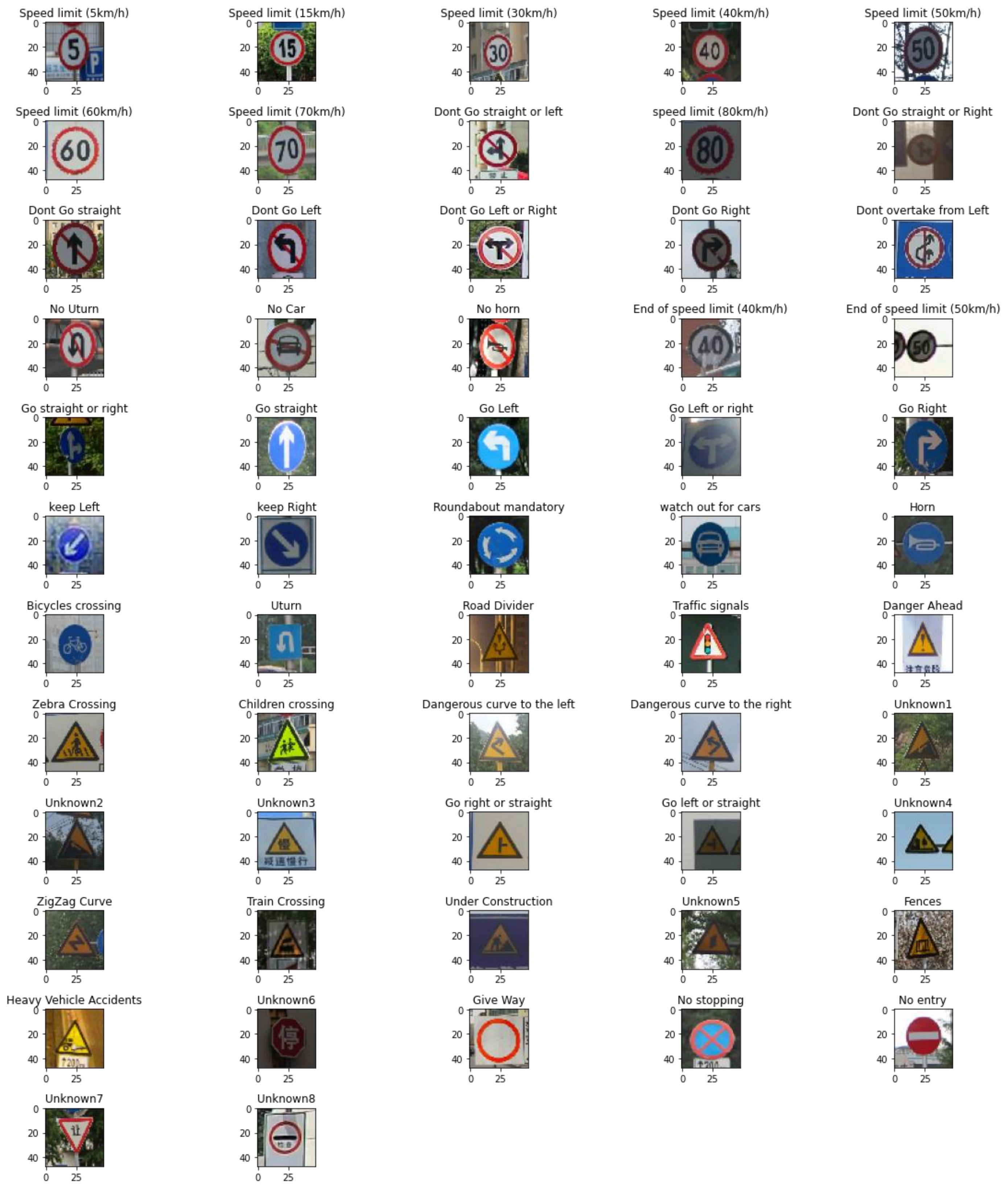


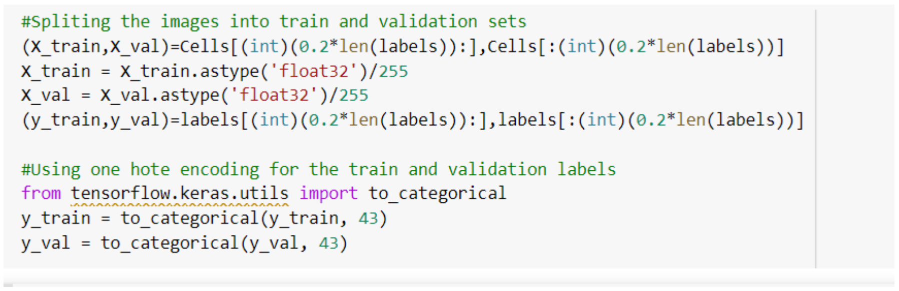
Fig. 2.1.7 Different classes in Traffic Sign Dataset

Step 2: DATA PREPROCESSING

Upon some initial analysis, it was noticed that there was a high-class imbalance in the dataset. Figure 2.1.4 and Figure 2.1.6 show the data count per class. If the model is trained on this dataset, it poses a problem as this makes the model biased towards the class with higher data frequency. To resolve this, data augmentation was performed. Images from low-frequency classes were picked up and random rotation and brightness variation was performed. Eventually, data was normalized for all classes. Then, standardization was performed on the data to normalize its mean and make it unit variance.

Step 3: Splitting the data into Training, Validation, and Test set

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 sets in the ratio of 80:20. The list of labels ranges from 0 to 42 that represents each category but the neural network needs a different format which is one hot encoding. One hot encoding is a vector representation where all elements of the vector are 0 except one, which has a 1 value. Code for the given in Figure 2.1.8. Here, we don't need to split the data for the test set an additional folder containing random test images is already present.



```
#Spliting the images into train and validation sets
(X_train,X_val)=Cells[(int)(0.2*len(labels)):],Cells[:(int)(0.2*len(labels))]
X_train = X_train.astype('float32')/255
X_val = X_val.astype('float32')/255
(y_train,y_val)=labels[(int)(0.2*len(labels)):],labels[:(int)(0.2*len(labels))]

#Using one hote encoding for the train and validation labels
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train, 43)
y_val = to_categorical(y_val, 43)
```

Fig. 2.1.8 Code for Splitting the dataset into Training and Validation Set

Step 4: Build CNN Model

In the field of deep learning, a CNN model represents a class of deep neural networks. CNN's have multilayer perceptron, which is fully connected. The meaning of full connection in multilayer perceptron is that each neural in one layer is connected to all neurons of the next layer. This architecture can effectively prevent data from overfitting. Their specific applications have image and video recognition, classification, medical image analysis, natural language processing, etc. A convolutional neural network is composed of an input and an output layer as well as 10 as several hidden layers (Xu, Ren, Liu, & Jia, 2014). The hidden layers also consist of a series of convolutional layers that are based on multiplication and other dot product. ReLU layer is regarded as the activation function and is generally followed by other convolutions like pooling layers, fully connected layers, and normalization layers. They refer to as hidden layers as their inputs and outputs are hidden by the activation function and final convolution. The architecture of a typical CNN is introduced as follows: Convolutional Layer, Pooling Layer, ReLU Layer, Fully Connected Layer.

Pooling Layer: The pooling layer is responsible for reducing the spatial size of the convolved feature. There are 2 dominant types of pooling, including max pooling and average pooling. Max Pooling returns the maximum value from the portion of the image covered by the kernel. The average pooling returns the average across all the values from the portion of the images covered by the Kernel. Figure 2.1.9 shows an example of how pooling is done. In practice, the max-pooling is performed better than average pooling. Max pooling was used to discard the noisy activations and denoising while reducing dimensionalities. In terms of average pooling, it just simply performs the dimensionality reduction to suppress noises. Hence, we can say that max pooling is a better option compared to average pooling.

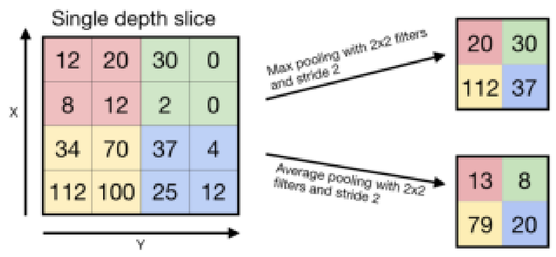

Single depth slice
x
y
12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
Max pooling with 2x2 filters and stride 2
20 30
112 37
Average pooling with 2x2 filters and stride 2
13 8
79 20

Fig. 2.1.9 Max and average pooling with a filter of size 2*2 and stride 2

<u>ReLU Layer</u>: Rectified linear unit utilized the non-saturating activation function *f(x)* = max(0,x) to remove negative values from an activation map and replace them with number 0. This operation enhances the nonlinearity of the decision function and the whole network without influencing the reception domain of the CONV layer. There are also other options available for increasing the nonlinear properties, such as the saturating hyperbolic tangent *f(x)* = tanh(x), *f(x)* = |tanh(x)| and the sigmoid function $\sigma(x)$ = (1 + e^ −x) ^−1. ReLU has gain more popularity compared to other functions due to the faster speed of training the neural network . Graph of ReLU is shown in Figure 2.1.10

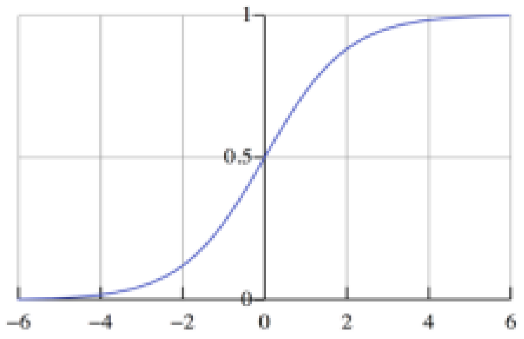


Fig. 2.1.10 The sigmoid function

<u>Fully Connected Layer:</u> In a fully connected layer, the neurons connect to all activations as seen in the regular neural networks as shown in Figure 2.1.11. Generally, inserting a fully connected layer is a cheap way to capture the nonlinear combination of high-dimensional featural.

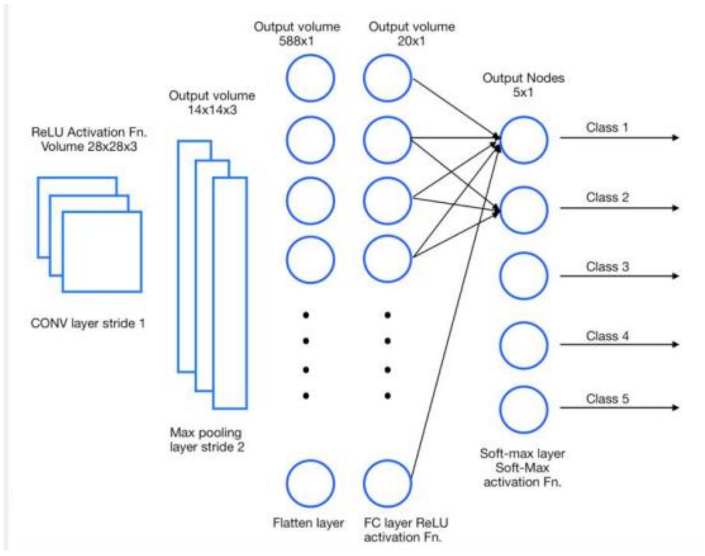


Fig. 2.1.11 Fully Connected Layer (FC layer)

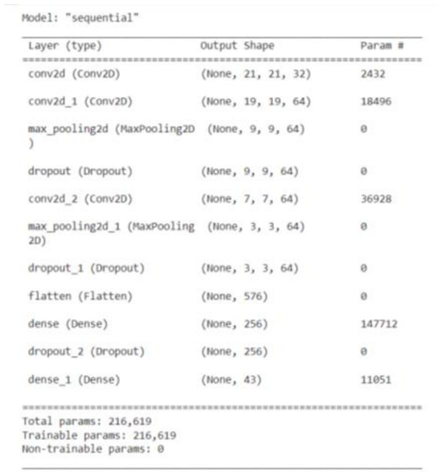
Loss Layer: The loss layer is normally the last layer in a neural network, which specifies the penalization process of training to the deviation between true labels and predicted results. Different tasks should apply with different loss functions. For instance, the Softmax loss is to predict a single class between dominant and certain low-level features in images. Sigmoid cross-entropy is to predict independent values of probability in [0,1].

Typical Convolutional Neural Networks: AlexNet, VGGNet, ResNet. The typical CNN architecture is composed of blocks of convolutional layers and pooling 14 layers followed by a fully connected layer and SoftMax layer at the end.

Object Detection Models: Faster R-CNN, You Only Look Once (YOLO)

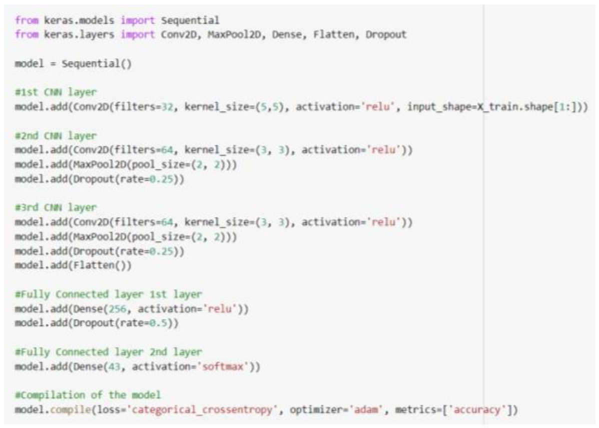
PROPOSED MODEL ARCHITECTURE

The implemented CNN model has two stacks made up of two and one Convolutional Layer respectively followed by two fully convolutional layers. We have used Max Pooling after every stack to discard the noisy activations and denoising while reducing dimensionalities. Also, the Dropout layer was applied after every stack to avoid overfitting. Figure 2.1.12 and Figure 2.1.13 shows the Summary & Code of the CNN model from Google Colab, where we have executed our model. In the following Figure 2.1.14 flowchart of the developed architecture is shown.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 21, 21, 32)        2432

 conv2d_1 (Conv2D)           (None, 19, 19, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 9, 9, 64)         0
 )

 dropout (Dropout)           (None, 9, 9, 64)          0

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_1 (MaxPooling  (None, 3, 3, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 3, 3, 64)          0

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 256)               147712

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 216,619
Trainable params: 216,619
Non-trainable params: 0
_________________________________________________________________
```

Fig. 2.1.12 Summary of CNN Model



```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

model = Sequential()

#1st CNN layer
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))

#2nd CNN layer
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

#3rd CNN layer
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())

#Fully Connected layer 1st layer
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))

#Fully Connected layer 2nd layer
model.add(Dense(43, activation='softmax'))

#Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fig. 2.1.13 Code For CNN Model

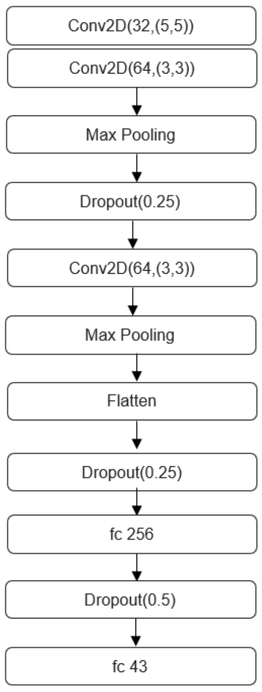
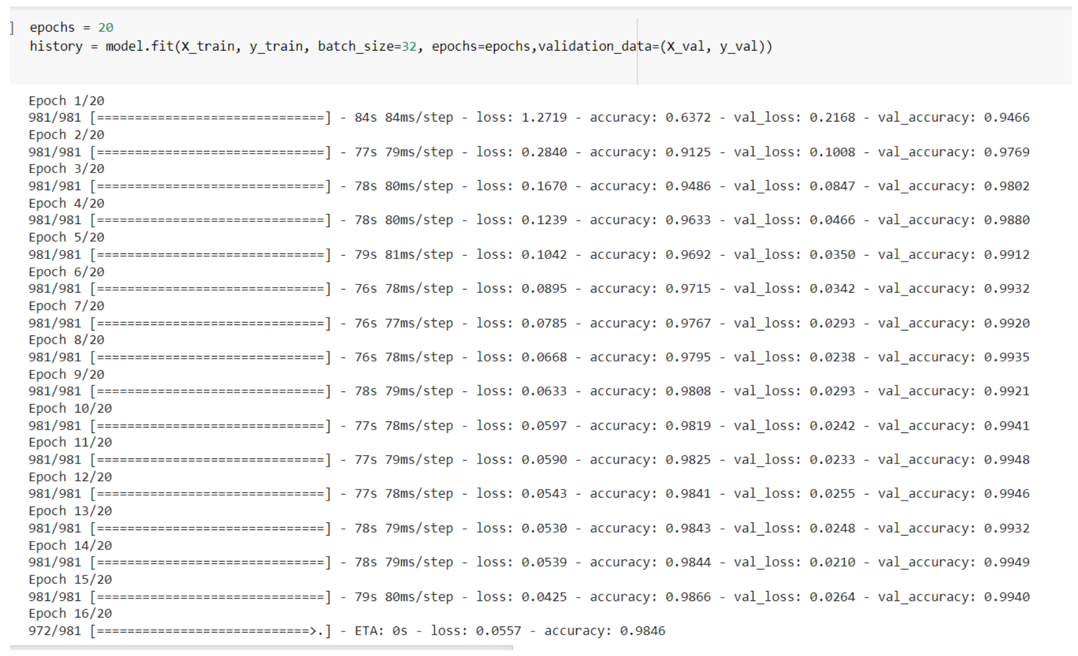


Fig 2.1.14 Block Diagram of proposed CNN Architecture (Fully connected layer is abbreviated as fc)

The first two convolutional layers have 32 and 64 convolution filters with kernel sizes = 5 and 3 respectively. Grid Search over different kernel dimensions : 3, 5, 7, 9,11, 19, and 31 for the convolutional layer was performed. The highest accuracy was achieved with kernel sizes 5 and 3 respectively. The subsequent one and last convolution layers have 64 convolution filters with kernel size 3. This is followed by Max pooling to gather the essence of all received feature maps and lastly flatten. Its output is given to a fully connected hidden layer consisting of 256 neurons. The final output layer has 43 neurons with softmax as an activation function to calculate the probability corresponding to 43 classes. Most of this architecture was designed using Random Search for various parameters.

Step 5: Apply CNN on Training Dataset

To train our model, we will use the model.fit() method that works well after the successful building of model architecture. All of architectures were trained using an adam optimizer at a learning rate = 0.001 (default learning rate) with batch size = 32 and epochs = 20. However, till epoch 15 we received 98.66 accuracy so we terminated the model. As this is a classification problem, the categorical cross-entropy loss function was used. All of the experiments were performed in python Google Colab using the Keras framework. All the detail about the code and its execution is given in Figure 2.1.15



```
epochs = 20
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,validation_data=(X_val, y_val))
```

```
Epoch 1/20
981/981 [==============================] - 84s 84ms/step - loss: 1.2719 - accuracy: 0.6372 - val_loss: 0.2168 - val_accuracy: 0.9466
Epoch 2/20
981/981 [==============================] - 77s 79ms/step - loss: 0.2840 - accuracy: 0.9125 - val_loss: 0.1008 - val_accuracy: 0.9769
Epoch 3/20
981/981 [==============================] - 78s 80ms/step - loss: 0.1670 - accuracy: 0.9486 - val_loss: 0.0847 - val_accuracy: 0.9802
Epoch 4/20
981/981 [==============================] - 78s 80ms/step - loss: 0.1239 - accuracy: 0.9633 - val_loss: 0.0466 - val_accuracy: 0.9880
Epoch 5/20
981/981 [==============================] - 79s 81ms/step - loss: 0.1042 - accuracy: 0.9692 - val_loss: 0.0350 - val_accuracy: 0.9912
Epoch 6/20
981/981 [==============================] - 76s 78ms/step - loss: 0.0895 - accuracy: 0.9715 - val_loss: 0.0342 - val_accuracy: 0.9932
Epoch 7/20
981/981 [==============================] - 76s 77ms/step - loss: 0.0785 - accuracy: 0.9767 - val_loss: 0.0293 - val_accuracy: 0.9920
Epoch 8/20
981/981 [==============================] - 76s 78ms/step - loss: 0.0668 - accuracy: 0.9795 - val_loss: 0.0238 - val_accuracy: 0.9935
Epoch 9/20
981/981 [==============================] - 78s 79ms/step - loss: 0.0633 - accuracy: 0.9808 - val_loss: 0.0293 - val_accuracy: 0.9921
Epoch 10/20
981/981 [==============================] - 77s 78ms/step - loss: 0.0597 - accuracy: 0.9819 - val_loss: 0.0242 - val_accuracy: 0.9941
Epoch 11/20
981/981 [==============================] - 77s 79ms/step - loss: 0.0590 - accuracy: 0.9825 - val_loss: 0.0233 - val_accuracy: 0.9948
Epoch 12/20
981/981 [==============================] - 77s 78ms/step - loss: 0.0543 - accuracy: 0.9841 - val_loss: 0.0255 - val_accuracy: 0.9946
Epoch 13/20
981/981 [==============================] - 78s 79ms/step - loss: 0.0530 - accuracy: 0.9843 - val_loss: 0.0248 - val_accuracy: 0.9932
Epoch 14/20
981/981 [==============================] - 78s 79ms/step - loss: 0.0539 - accuracy: 0.9844 - val_loss: 0.0210 - val_accuracy: 0.9949
Epoch 15/20
981/981 [==============================] - 79s 80ms/step - loss: 0.0425 - accuracy: 0.9866 - val_loss: 0.0264 - val_accuracy: 0.9940
Epoch 16/20
972/981 [============================>.] - ETA: 0s - loss: 0.0557 - accuracy: 0.9846
```

Fig. 2.1.15 Training the CNN model in the Train dataset

Step 6: Validating the result

With the help of matplotlib functions, we will plot the graph of training and validation accuracy as shown in Figure 2.1.16

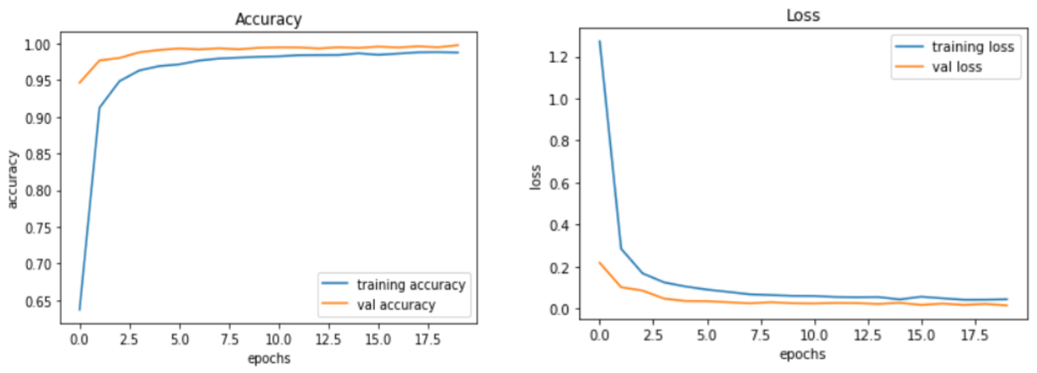
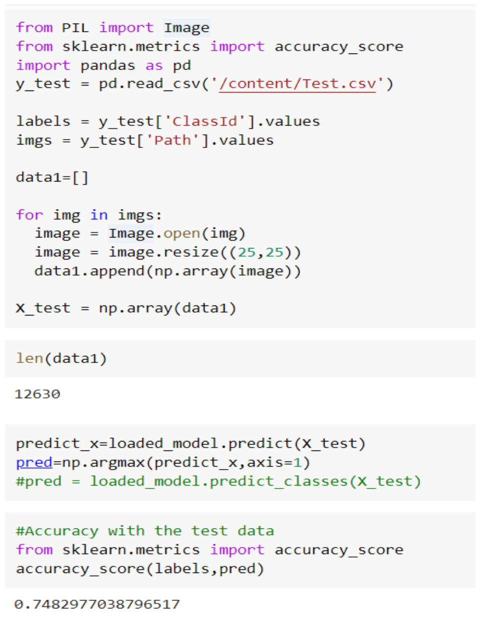


Fig. 2.1.16 Validation Loss and Accuracy Curve

Step 6: Make Prediction on Test Data and find its Accuracy

A folder named “test” is available in our dataset. Inside that, we got the main working comma-separated file called “test.csv”. It comprises two things, the image paths, and their respective class labels. We can use the pandas’ python library to extract the image path with corresponding labels. Total 12630 test images are there for training, Next, we need to resize our images to 25×25 pixels to predict the model and create a numpy array filled with image data. To understand how the model predicts the actual labels, we need to import accuracy_score from the sklearn. Metrics as shown in Fig. 2.1.17



```
from PIL import Image
from sklearn.metrics import accuracy_score
import pandas as pd
y_test = pd.read_csv('/content/Test.csv')

labels = y_test['ClassId'].values
imgs = y_test['Path'].values

data1=[]

for img in imgs:
  image = Image.open(img)
  image = image.resize((25,25))
  data1.append(np.array(image))

X_test = np.array(data1)
```

```
len(data1)
```

```
12630
```

```
predict_x=loaded_model.predict(X_test)
pred=np.argmax(predict_x,axis=1)
#pred = loaded_model.predict_classes(X_test)
```

```
#Accuracy with the test data
from sklearn.metrics import accuracy_score
accuracy_score(labels,pred)
```

```
0.7482977038796517
```

Fig. 2.1.17 Accuracy of Test Data and Code

FULL SOURCE CODE

https://colab.research.google.com/drive/1GYb3clx-7EfV-1QuSXA_w_ABUgPVlYn7?usp=sharing

CONCLUSION

It can be observed from Fig that the accuracy we received is around 74%. But if we compare it with the training accuracy, it happens to be 98.66%. From this, we can conclude that on the training dataset the CNN model was overfitted due to which on the test dataset we got 74% accuracy. We need to further perform data augmentation techniques and make changes in CNN network parameters (learning rate, epoch, kernel size; etc) and layers to get better accuracy on test data.

3. DIFFICULTIES WE CROSSED WHILE GOING THROUGH THIS PROJECT

While studying dataset2, we found that the graph was not coming correct. After analyzing it thoroughly we found that class 18 and class 19 were the same as classes 3 and 4. But after observing it visually, it was concluded that the name of the sign was written wrong. Similarly, class 30 and class 36 were the same. So we merged the 2 classes into 1.

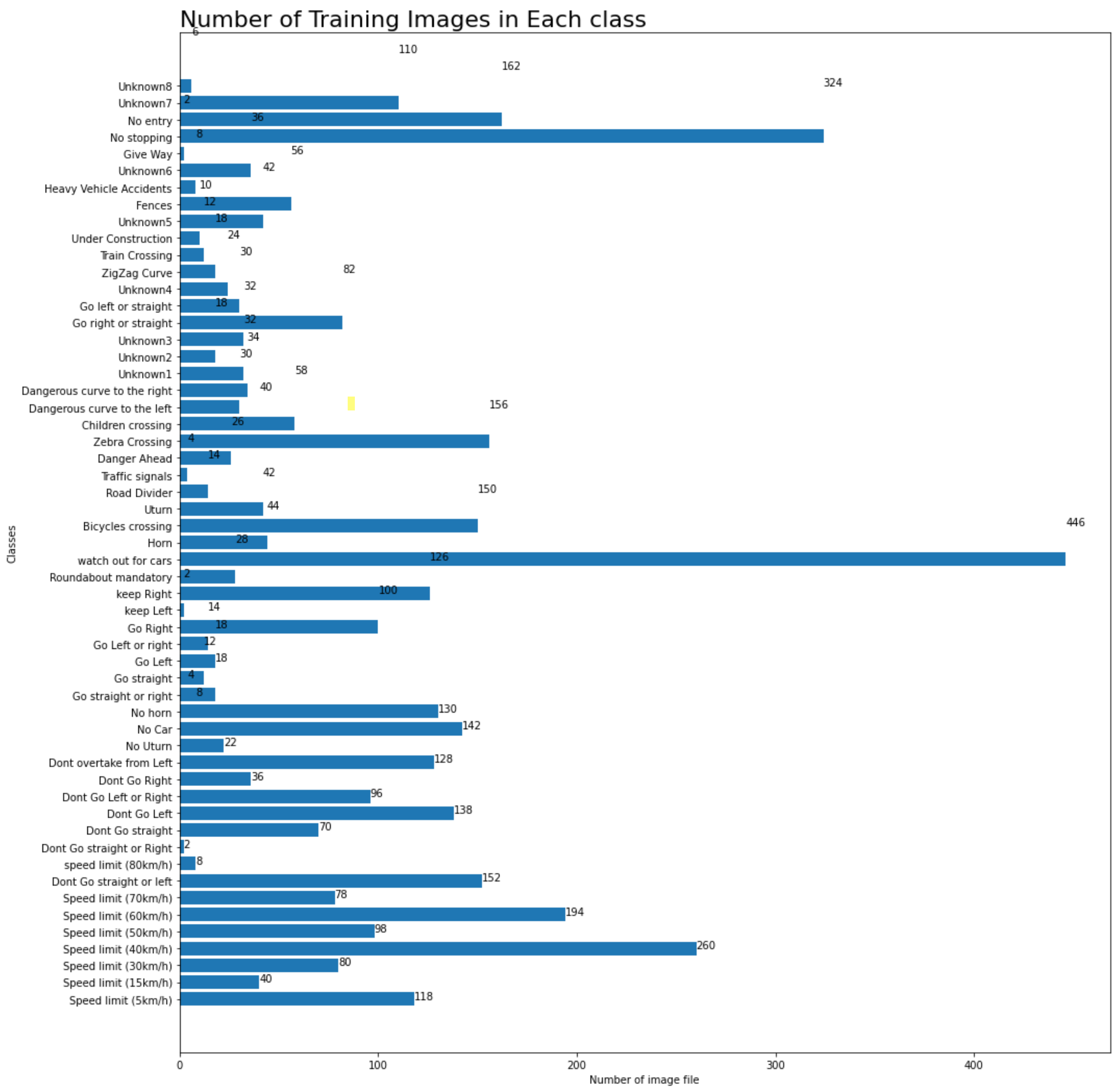
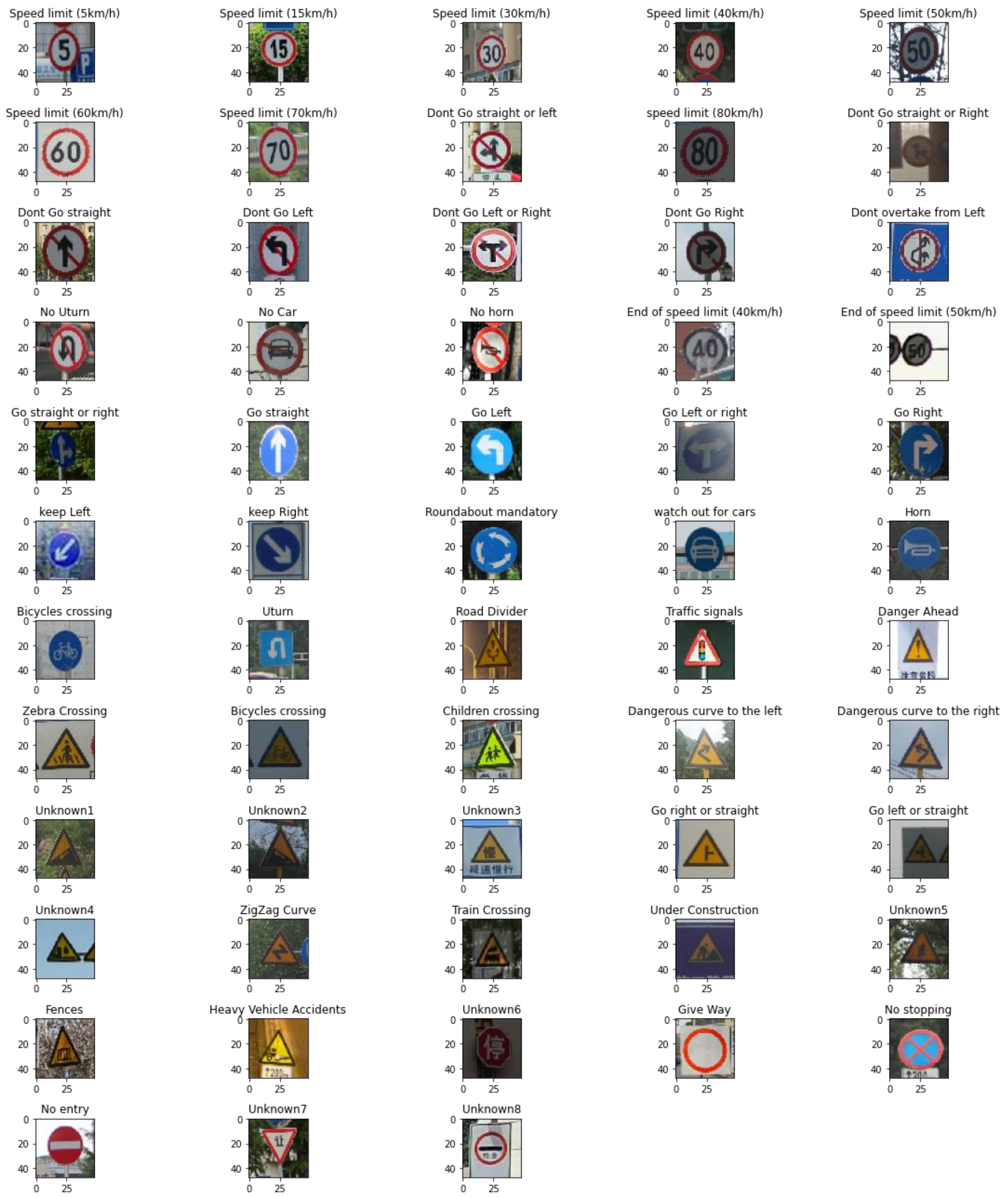


Fig 3.1. In this bar graph, you can observe that the bars and the values are not fully aligned.



• **Fig 3.2.** In the figure, there are 2 times bicycle crossings.

REFERENCE

Dr. D. Y. Patil., „A Road Sign Detection and the Recognition for Driver Assistance Systems‟ International Conference on Energy Systems and Applications (ICESA 2015).

P. Shopa, Mrs. N. Sumitha, Dr. P.S.K Patra. (2014), „Traffic Sign Detection and Recognition Using OpenCV‟, International Conference on ICICES2014 - S.A.Engineering College, Chennai, Tamil Nadu, India.

H. Akatsuka and S. Imai, “Road signposts recognition system," Training Journal, pp. 12-11, 2013

Y. Xie, L. F Liu, C. H. Li, and Y. Y. Qu. “Unifying visual saliency with HOG feature learning for traffic sign detection.” In IEEE Intelligent Vehicles Symposium, , 2009, pp. 24-29