MAJOR PROJECT REPORT

ON

Traffic Sign Recognition for Advanced Driver Assistance System (ADAS) using Deep Learning techniques

BY

SHARIKA ANJUM MONDAL	**A91005118016**
PRITHIVI GUHA	**A91005118013**
KOUSTAV PAL	**A91005118015**

Guided by:

Prof. Kalyan Chatterjee Sir

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY, KOLKATA

MAY, 2022

DECLARATION

I , **Sharika Anjum Mondal** , student of B.Tech (ECE) hereby declare that the **project** title "**Traffic Sign Recognition for Advanced Driver Assistance System (ADAS) Deep Learning techniques"** which is submitted by me to the **department** of **Electronics and communication , Amity School of Engineering and Technology , Kolkata** , in fulfilment of requirement for the award of the degree of **Bachelor of Technology** has not been previously formed the basis for the award of any degree, diploma or any other similar title recognition.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the report other than the brief excerpts requiring only proper acknowledgement in scholarly writing and all such use is acknowledged.

Signature:

Place: Kolkata

Date: 27th May, 2022

CERTIFICATE

On the basis of report submitted by **Sharika Anjum Mondal**, student of **B.Tech (ECE)** , I hereby certify that the report "**Traffic Sign Recognition for Advanced Driver Assistance System (ADAS) Deep Learning techniques"** which is submitted to department of **Electronics and Communication, Amity School of Engineering and Communication, Kolkata** in fulfilment of requirement for the award of the **degree** of **Bachelor of Technology (2018-2022)** is an original contribution with existing knowledge and faithful record of work carried out by him/her under my guidance and supervision.

To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Kolkata

Date: 27th May, 2022

-------------------------------	------------------------------	------------------------------
Prof. Kalyan Chatterjee	Prof. Sayanti Banerjee	Dr. Birajashis Pattnaik
(Project Guide, ECE, ASETK)	(H.O.D, ECE, ASETK)	(H.O.I, ASETK)

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and our organisation. We would like to extend our sincere thanks to all of them.

We are highly indebted to all the faculties of the Department of Electronics & Communication Engineering of Amity School of Engineering and Technology, Kolkata for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express my special gratitude and thanks to our HOD - Mrs. Sayanti Banerjee ma’am for giving us such attention and time.

We would also like to express our gratitude towards Mr. Kalyan Chatterjee sir and Mr. Subhasish Roy sir of our department for their kind cooperation and encouragement which helped us in completion of this project.

Our thanks and appreciation also go to our batch mates in developing the project for their thorough support.

ABSTRACT

The traffic sign recognition system (TSRS) is an important component of an intelligent transportation system (ITS). Being able to interpret traffic signs properly and efficiently can increase driving safety. This project proposes a traffic sign identification approach based on deep learning, which primarily targets at the detection and classification of traffic signs while being trained on a traffic sign benchmark dataset. A traffic sign recognition and identification approach based on image processing is proposed, which is integrated with a convolutional neural network (CNN) to classify traffic signs. TensorFlow is used to implement CNN. We have 99.4% accuracy in identifying.

Keywords - Traffic Sign Recognition System (TSRS), Detection & Classification, Image Processing, CNN, TensorFlow

INDEX

1. Introduction	7
2. Literature Review	7 - 8
3. Our Propositions	8 - 14
3.1. Data Collection	8
3.2. Pre-processing	9 - 10
3.2.1. Image Grayscaling	9
3.2.2. Image Equalising	9
3.2.3. Normalise values of the Image	9
3.2.4. Addition of depth to the Image	10
3.2.5. Image Augmentation	10
3.3. Data Visualization	10
3.4. Model Building	10 - 14
3.4.1. Convolution layers	11 - 12
3.4.2. Pooling Operation	12
3.4.3. Drop-Out layer	13
3.4.4. Flattening Operation	13
3.4.5. Fully Connected Dense layer	13 - 14
3.4.6. Model Compilation	14
3.5. Model Training – Testing (Blind testing)	14
4. Model Evaluation	15 - 16
4.1.1. Loss Curve	15
4.1.2. Accuracy Curve	15 - 16
5. Results	16
6. Limitations	17 - 18
7. Conclusion	18
8. References	18 - 20
9. Appendix [CODE]	21 - 33

1. Introduction

Autonomous car or Self Driving car is a driverless ground vehicle which has a capability of sensing its external and internal environment and runs on its own, with little or no human input. A car, be it manual or autonomous, when it runs on road, has to follow traffic rules to maintain road safety. The traffic sign recognition (TSR) system plays an important role in this situation which makes the autonomous car learn about various traffic rules. It involves a front facing camera with a wide field of view that can scan the entire road for traffic signs. The Convolutional Neural Network (CNN) model is used to train the TSR to recognise various traffic signs in this project. The German Traffic Sign Recognition Dataset (GTSRB) is an image classification dataset which is used to train TSR in this project. Many big companies like Tesla, Uber, Mercedes etc and other researchers are still working on autonomous driving systems for achieving more accuracy. The market size is still growing. The global market for self-driving cars is projected to grow from 20.3 million in 2021 to about 63 million in 2030. In this paper, we look at how to create an accurate and real-time TSR model using deep learning.

2. Literature Review

TSR has always been a popular study topic in recent years. TSR is researched to recognise traffic sign region and non-traffic sign area in complex scene of photos, TSR is to extract the specific features represented by traffic sign patterns [20]. Existing TSR approaches are divided into two categories: those based on classical methods and those based on deep learning methods.
TSR approaches based on colour and shape of a given image's major phases are to extract the visual information contained in the candidate region, collect and segment the traffic signs in the image, and accurately label the signs using pattern classification [21]. TSR, on the other hand, requires colour and shape information, which is used to increase recognition accuracy.
The challenges of traffic sign lighting changes or colour fading, as well as traffic sign distortion and occlusion, remain unsolved [14]. Conventional machine learning approaches often choose certain visual cues and utilise them to identify traffic sign classes. Specific aspects include Haar-like characteristics, HOG characteristics, SIFT characteristics, and so on [3].
Traditional TSR approaches are based on template matching, which requires extracting and using invariant and comparable visual elements of traffic signals before running matching algorithms for pattern classification. Because of the changes in traffic signs, describing the visual aspects properly is a difficult challenge for these approaches' feature representation [17, 24].
As classifiers, neural networks, Bayesian classifiers, random forests, and Support Vector Machines (SVM) are used. However, because the effectiveness of traditional machine learning algorithms is dependent on the features supplied, they are prone to omitting important characteristics. Furthermore, appropriate feature description information is necessary for different classifiers. As a result, standard machine learning approaches have limits, and their real-time performance is not comparable.
Deep learning is a technique that uses a multilayer neural network to automatically extract and learn the properties of visual objects, which has applications in image processing [29]. CNN

models are among the most widely used deep learning algorithms for TSR. TSR algorithms are based on region proposals, sometimes known as two-stage detection algorithms; the main principle is selective search [10], and its advantages include excellent detection and positioning performance at the cost of a large number of calculations and high-speed computing hardware.
R-CNN, Fast R-CNN, and Faster R-CNN are all included in the CNN models. Faster R-CNN combines bounding box regression with object classification, employing end-to-end techniques to detect visual objects, which not only improves the accuracy of object detection but also increases the speed of object identification. Road signs are often identified from the driver's perspective; however, in this work, we examine the signs from the perspective of satellite photos. In [24], guided image filtering was used to eliminate visual artefacts such as hazy and haze from the input picture. For model training, the processed picture is input into the suggested networks.

3. Our Propositions

The task is divided into three primary sections:
- Pre-processing
- Model building and training
- Detection and classification of traffic signs

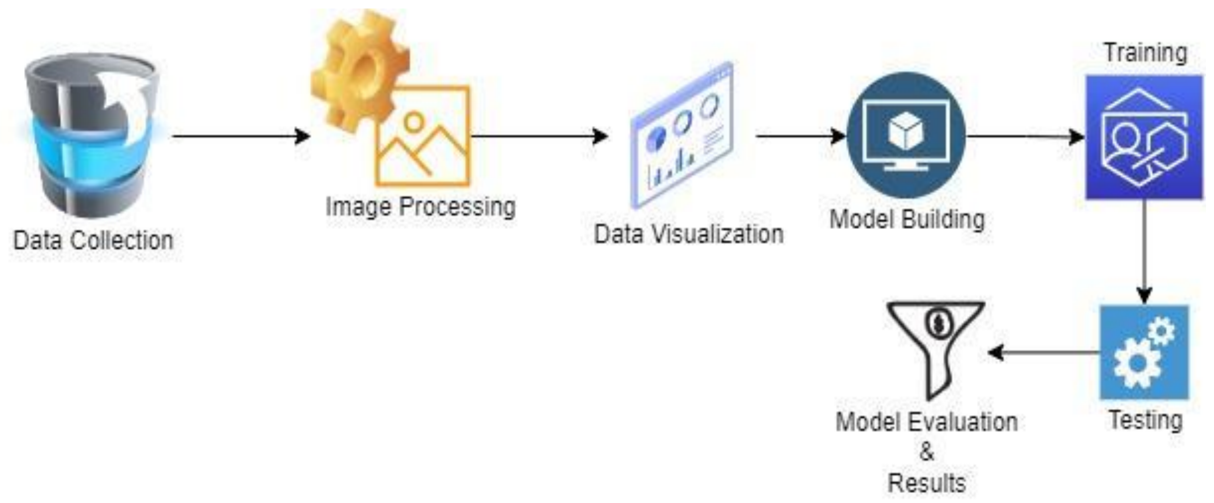


Fig 1: Summary of the workflow

3.1 Data Collection

The data was taken from a benchmark dataset in traffic sign recognition - GTSRB, which is around 300MB in size. It has over 50,000 photos of various traffic signs and is divided into 43 distinct classes.
This database consists primarily of traffic incidents caught by cameras. The photographs in this collection are divided into several categories, including sign type, sign condition (such as obstructed, damaged, or faded), weather circumstances, light geometry, and other characteristics. It varies greatly; some courses have a large number of photos, while others have only a handful. The dataset already has a predefined train folder that contains photos inside each class and a predefined test folder that was used for blind testing the model.

3.2 Preprocessing

Once the data has been collected and displayed, the pictures must be preprocessed to improve the model's performance.The pre-processing was done in stages. They are as follows:

- Image Grayscale
- Image Equalising
- Normalise values of the Image
- Addition of depth of the Image
- Image Augmentation

3.2.1 Image Grayscale

A grayscale (or grey level) picture is one in which the only colours are shades of grey. This stage is conducted for extracting descriptors rather than immediately operating on colour photos to simply simplify the algorithm complexity and lower the computing cost.

3.2.2 Image Equalising

Histogram Equalisation is an image processing method that uses a histogram to alter the contrast of a picture. It spreads out the most common pixel intensity values or expands out the image's intensity range to improve contrast. This procedure was carried out primarily to standardise the lighting in the system.

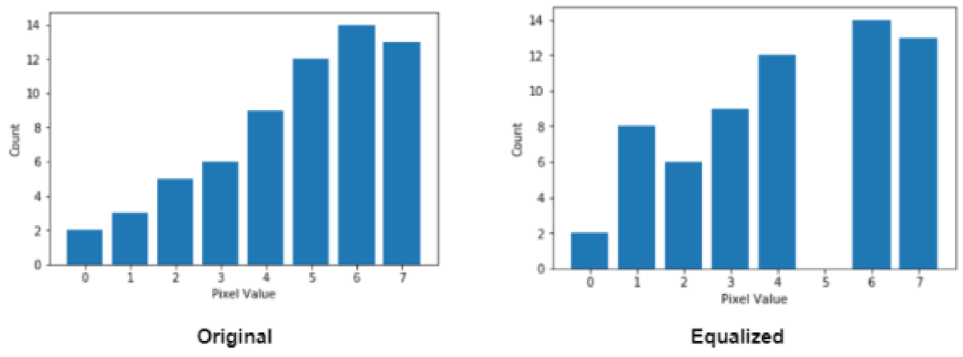


Fig 2: Image Equalisation technique

3.2.3 Normalise values of the Image

Data normalisation is a critical process that ensures each pixel has a consistent data distribution. This speeds up convergence while training the network. The main goal of normalisation is to make computation more efficient by lowering pixel values to 0 to 1 instead of 0 to 255.

3.2.4 Addition of depth of the Image

Because the picture pixels are already normalised to 0 and 1, a bit depth of 1 is maintained. The more colours a picture can store, the greater its bit depth. The most basic picture, a 1 bit image, can only display two colours: black and white. This is due to the fact that the 1 bit can only hold one of two values: 0 (black) or 1(white).

3.2.5 Image Augmentation

To avoid the expensive cost of gathering thousands of training photos, image augmentation was created to synthesise training data from an existing dataset. Image Augmentation is the technique of modifying pictures already in a training dataset to generate several changed variants of the same image. This not only gives us additional photos to train on, but it also exposes our classifier to a larger range of lighting and colouring scenarios, making our classifier more resilient. The following enhanced parameters are being considered:

- Width and height shift range of 10%
- A zoom in and zoom out range of 20%
- A shear range (the angle of the slant in degrees) of 10% is used.
- The picture is rotated randomly by 10^{O}.

3.3 Data Visualisation

The model has many learnable parameters, and analysing them will assist in determining how successfully the model has been trained and to what extent the model will function and provide the best outcomes.These metrics can provide insight into Neural Network training. Visualising the output of the hidden layer also helps a lot.

There are 43 separate folders in the dataset. The folders are numbered from 0 to 42, and each number corresponds to a different class of image. For each class, there are thousands of images. The graph depicts how many images are there for each class, and we may deduce which class of image will perform better than the others based on this graph.

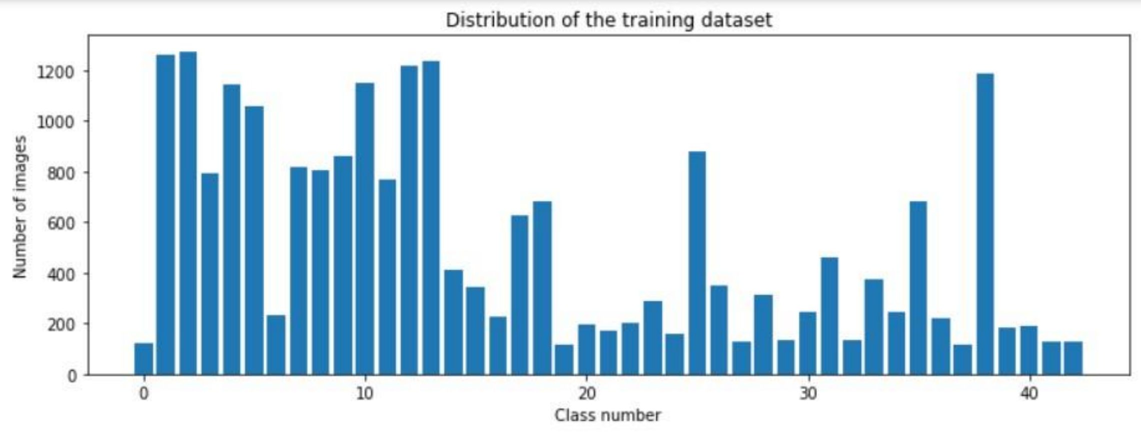


Fig 3: Data Visualisation of the no. of Images Vs. Class Number

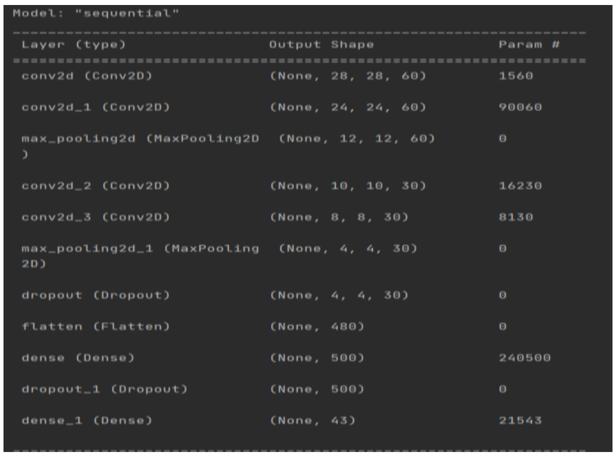
3.4 Model Building

Deep learning is a key subfield of machine learning due to its great performance across several domains. Convolutional Neural Network (CNN) is a strong image processing deep learning type that is frequently used in computer vision. It includes image and video recognition, as well as a recommender system and natural language processing (NLP).

CNN employs a multilayer system that includes an input layer, an output layer, and a hidden layer that includes several convolutional layers, pooling layers, and fully linked layers.First and foremost, a sequential class is launched since there are numerous layers to develop CNN, all of which must be in sequence.

The model's architecture remains as follows:

- 2 Conv2D layer (filter=60, kernel_size=(5,5), input_shape=(32,32,1),activation=”relu”) [Adding more convolution layers is equal to less features but can cause accuracy to increase.]
- MaxPooling2D layer (pool_size=(2,2))
- 2 Conv2D layer (filter=60, kernel_size=(5,5), input_shape=(32,32,1), activation=”relu”)
- MaxPooling2D layer (pool_size=(2,2)
- Dropout layer(rate = 0.5)
- Flattening
- Dense layer(43 nodes, activation='relu')
- Dropout layer(rate = 0.5)
- Dense layer(43 nodes, activation=” softmax”)



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 60)        1560

 conv2d_1 (Conv2D)           (None, 24, 24, 60)        90060

 max_pooling2d (MaxPooling2D  (None, 12, 12, 60)       0
 )

 conv2d_2 (Conv2D)           (None, 10, 10, 30)        16230

 conv2d_3 (Conv2D)           (None, 8, 8, 30)          8130

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 30)         0
 2D)

 dropout (Dropout)           (None, 4, 4, 30)          0

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 500)               240500

 dropout_1 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 43)                21543

_________________________________________________________________
```

Fig 4: Model Summary

Dense Layer is also referred to as a fully connected dense layer (or FC layer). Following the completion of the architecture, the model is delivered for compilation.

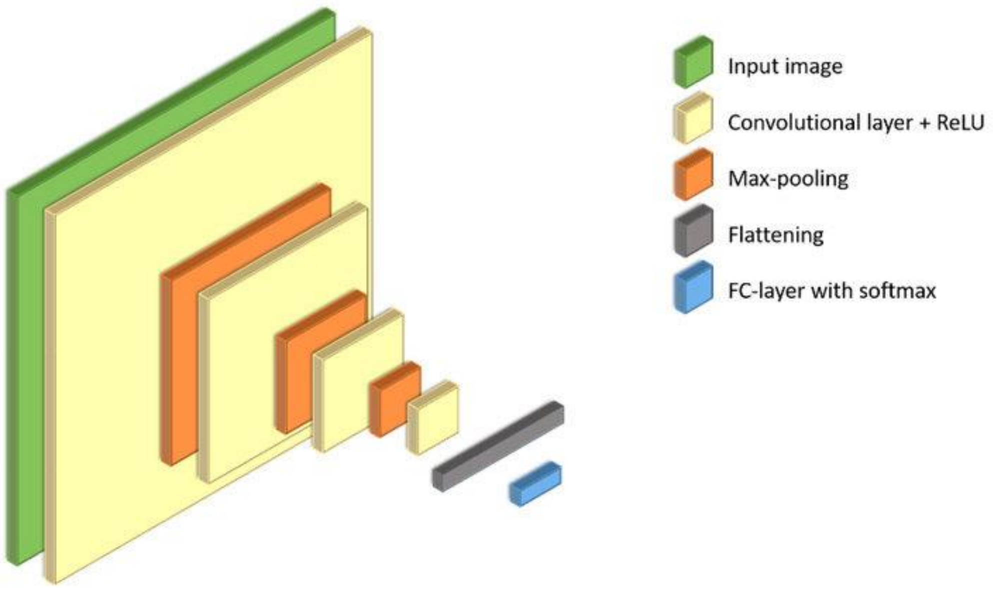


Fig 5: Model Architecture

3.4.1 Convolution Layers

To begin a sequential class since there are several levels to create a CNN, all of which must be done sequentially. First, two convolution layers with four parameters are added.

- **Filters** - Convolution's primary goal is to locate features in an image using a feature detector. Then place them in a feature map, which maintains individual picture characteristics.The feature detector, also known as a filter, is likewise randomly started, and after many iterations, the filter matrix value that is optimum for separating pictures is chosen. We're utilising 60 features in this case.
- **Kernel size** - Kernel size is the size of the filter matrix. We're utilising a 5*5 filter size here.
- **Shape input** - This option specifies the picture size: 32*32*1. Because the photos aren't in RGB format, the image's third dimension is 1.
- **Activation function, ReLu -** Because pictures are non-linear, the ReLu activation function is employed after the convolutional procedure to achieve non-linearity. ReLu is an abbreviation for Rectified linear activation function. If the input is positive, the relu function will output it directly; otherwise, it will output zero.

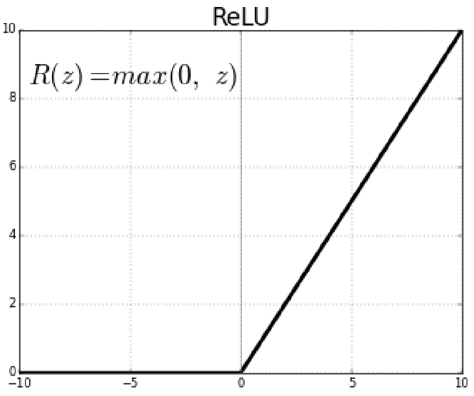


Fig 6: ReLu activation function

3.4.2 Pooling Operation

After CNN has been set up, the pooling procedure must be started. Pooling is a downsampling procedure on a picture. The pooling layer is used to minimise the feature maps' size. As a result, the Pooling layer minimises the number of parameters to learn as well as the amount of processing in the neural network.
Instead of precisely positioned features generated by the convolution layer, future actions are done using summarised features obtained by the pooling layer. As a result, the model becomes more resistant to fluctuations in the orientation of the feature in the picture.
Pooling may be classified into three types:

- Max Pooling
- Pooling on an Average
- Global Collecting

Max Pooling has been used in this case with a pool size of 2*2.

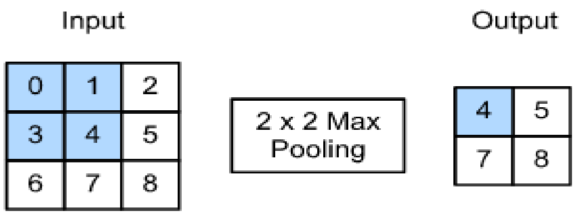


Fig 7: Max Pooling explained in brief

3.4.3 DropOut Layer

Dropout is applied to the input.

The Dropout layer, which helps prevent overfitting, randomly sets input units to 0 at a frequency of rate of 0.5 at each step throughout the training period. Inputs that are not set to 0 are scaled up by 1/(1 - rate) such that the sum of all inputs remains constant.

3.4.4 Flattening Operation

The flattening procedure converts the dataset into a 1-D array for input into the fully connected layer, which is the following layer.

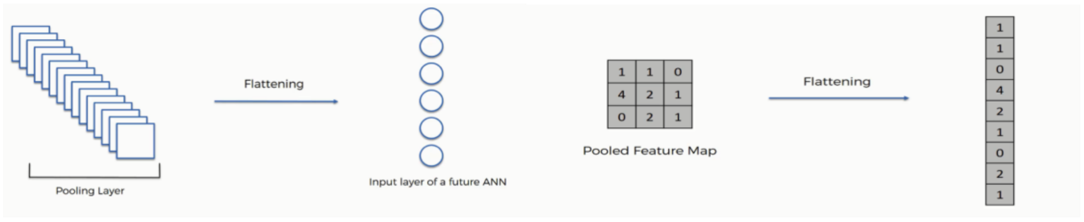


Fig 8: Flattening Operation explained

3.4.5 Fully Connected Dense layers

The flattening operation's output serves as input for the neural network using ReLu as the activation function. The artificial neural network's goal is to make the convolutional neural network more advanced and capable of identifying pictures. The output layer is constructed using the softMax activation function.

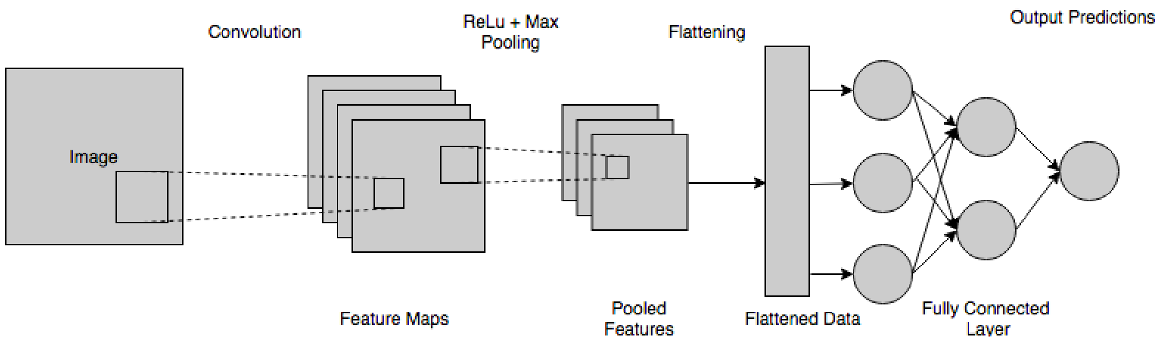


Fig 9: Dense Layer explained

- **No. of nodes in the Fully connected layer**
  This is the no. of nodes present in each layer, which is 43 in this case as there are 43 different classes in the dataset.
- **Activation function, SoftMax**
  It is employed as the neural network's final activation function to convert the neural network's output to a probability distribution over predicting classes. Softmax's output is

in the form of probability for each conceivable result for predicting class. The probability sum for all feasible prediction classes should be one. The equation for softmax is:

$$\sigma(\hat{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where,

σ = Softmax

$\hat{z}$ = Input Vector

e^{z_i} = Standard exponential function for input vector

K = No. of classes in the multi-class classifier

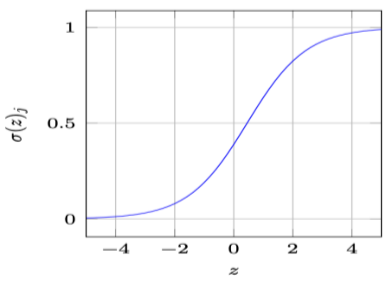


Fig 10: Softmax explained

e^{z_j} = Standard exponential function for output vector

3.4.6 Model compilation

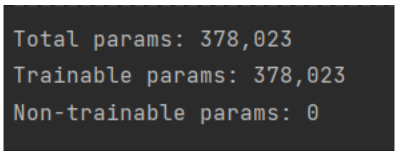
During compilation, three arguments were taken into account.

- **Loss function -** In this classification task, the categorical_crossentropy loss function is applied. This loss is an excellent indicator of how distinct two discrete probability distributions are from one another.
- **Optimizer -** The Adam Optimizer is used to alter the weights and learning rate of neural networks. By minimising the function, optimizers are employed to address optimization issues.
- **Metrics -** Accuracy is used to assess the performance of the Convolutional neural network method.

3.5 Model Training and Model Testing (Blind Testing) with real-time query using enabled web camera

The CNN model is trained on the training dataset using 30 iterations (epochs), with each iteration having 2000 steps. The model.fit() method is used to train our model, which works well following the successful construction of model architecture. We achieved 99.4 percent accuracy on training sets with 50 batch sizes and achieved stability after 30 epochs with 2000 steps in each epoch.
The value of trainable parameters is adjusted/modified during training according to their gradient. Non-trainable parameters are those whose value is not optimised as a function of their gradient during training. The sum of Trainable and Non-trainable params is Total params.

For blind testing, real-time inputs from our system's embedded web camera were captured using OpenCV and supplied into the system. The model performed admirably even with unknown and noisy inputs.



```
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
```

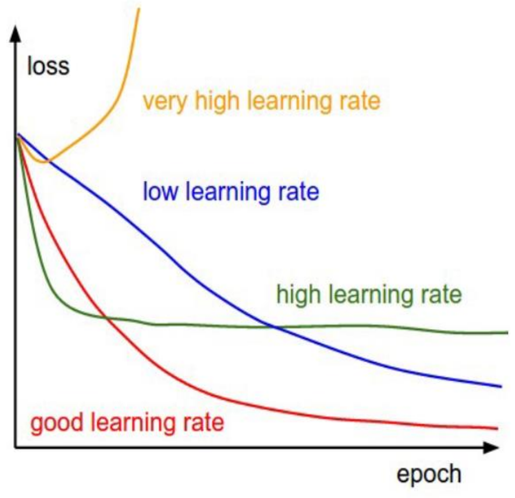
Fig 11: Trainable parameters

4. Model Evaluation

Different evaluation measures are utilised to understand the model's performance as well as its strengths and shortcomings, which is an important aspect of model development. It is critical to analyse the model's efficacy early on because it aids with model monitoring.

4.1 Loss Curve

The loss curve depicts the training process and the neural network's learning direction. After each epoch, the graph is plotted. The loss function is calculated across all data items during an epoch and is guaranteed to deliver the quantitative loss measure at that epoch. The learning rate can be estimated by comparing the graph to a reference graph.



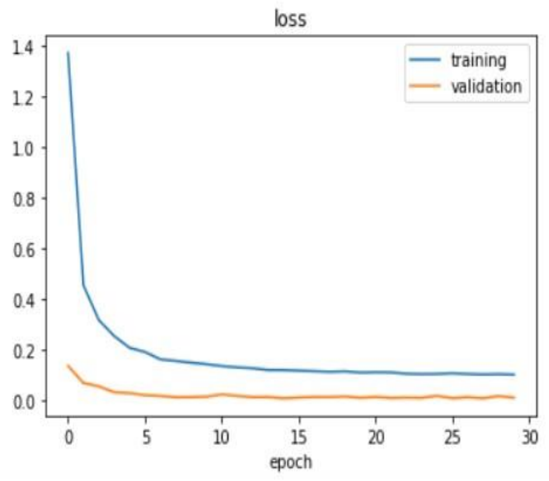
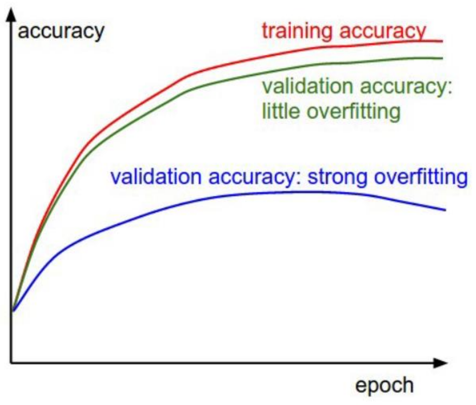


Fig 12 : Reference Graph for Learning Rate   Fig 13: Graph obtained from model

The image speaks for itself. It may be estimated that the model has a satisfactory learning rate by comparing the two graphs.

4.2 Accuracy Curve

The accuracy curve is one of the most significant graphs to understand the Neural Network's progress. The curve that includes both training and validation accuracy is more relevant. By comparing the graph to a reference graph, the advancement in the model's accuracy can be evaluated.



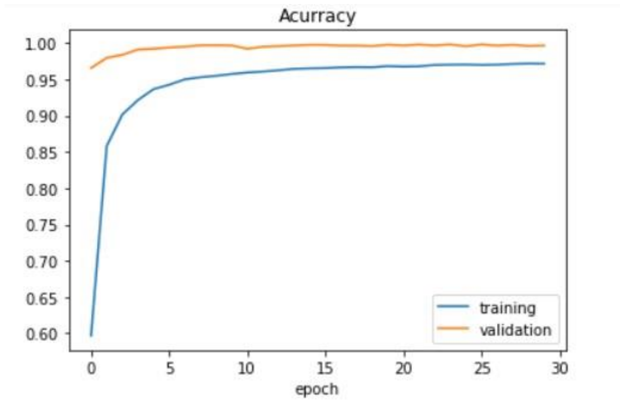
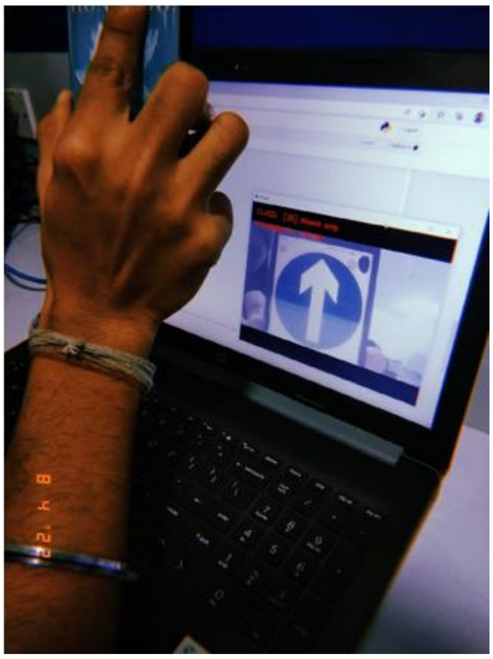


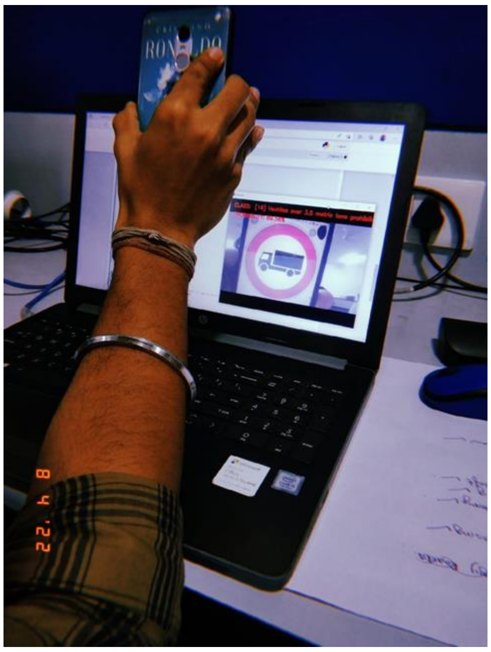
Fig 14: Reference Graph for Accuracy Curve   Fig 15: Graph obtained from model

Overfitting is seen by the gap between training and validation accuracy. The higher the overfitting, the larger the gap. The model's graph shows a relatively small gap, indicating that it has not been overfitted. The model is providing an accuracy of 99.42% which is pretty good.

5. Results

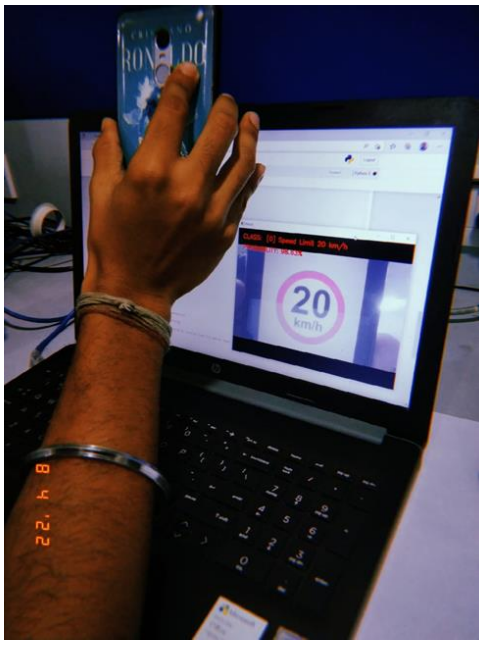


Detected class – Ahead only
Accuracy achieved – 100%



Detected class – Vehicles over 3.5 metric tons prohibited

Accuracy achieved – 89.56%



Detected class – Speed limit 20km/h

Accuracy achieved– 98.63%

6. Limitations

Despite its high level of accuracy, the model has several limits. They are also:

- The model fails to capture pose, view, orientation of the images because of the intrinsic inability of max pooling layer.

- Image processing should be done properly to detect and focus the sign boards properly and to read the images which are hazy.

7. Conclusion

This project explores the construction of a classification algorithm for the traffic sign recognition problem. When paired with image processing steps, the proposed technique for traffic sign classification produces extraordinarily good results: 99.4 percent of properly classified photographs. The suggested categorization technique is implemented using the TensorFlow framework. The developed method was tested on a device powered by an Intel Core i3 7th Gen Processor. In addition, an audio-enabled system warns drivers and passengers of upcoming traffic signs. In the future, we plan to teach the CNN to consider more traffic sign classes as well as potential severe weather conditions.

8. References

[1] Bangquan X, Xiong WX (2019)Real-time embedded traffic sign recognition using an efficient convolutional neural network. IEEE Access 7:53330–53346

[2] Chen Q, Huang N, Zhou J, Tan Z (2018) An SSD algorithm based on vehicle counting method. Chinese Control Conference

[3] Ellahyani A, Ansari M, Lahmeyer, Trémeau A (2018) Traffic sign recognition method for intelligent vehicles. J Opt Soc Am 35(11):1907–1914. https://doi.org/10.1364/JOSAA.35.001907

[4] Garg P, Chowdhury DR,More VN (2019) Traffic sign recognition and classification using YOLOv2, Faster R-CNN and SSD. International Conference on Computing, Communication and Networking Technologies

[5] Hao G, Yingkun Y, Yi Q (2019) General target detection method based on improved SSD. IEEE Joint International Information Technology and Artificial Intelligence Conference

[6] He Z, Nan F, Li X, Lee SJ, Yang Y (2020) Traffic sign recognition by combining global and local features based on semi-supervised classification. IET Intel Transport Syst 14(5):323–330

[7] Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507

[8] Hu GX, Hu BL, Yang Z, Huang L, Li P (2021) Pavement crack detection method based on deep learning models. Wirel Commun Mob Comput 2021. https://doi.org/10.1155/2021/5573590

[9] Huo A, Zhang W, Li Y (2020) Traffic sign recognition based on improved SSD model. International Conference on Computer Network, Electronic and Automation

[10] Jin Y, Fu Y, Wang W, Guo J, Ren C, Xiang X (2020)Multi-feature fusion and enhancement single shot detector for traffic sign recognition. IEEE Access 8:38931–38940

[11] Kuznetsova A, Maleva T, Soloviev V (2020) Detecting apples in orchards using YOLOv3 and YOLOv5 in general and close-up images. International Symposium on Neural Networks

[12] Li S, Gu X, Xu X, Xu D, Zhang T, Liu Z, Dong Q (2021) Detection of concealed cracks from ground penetrating radar images based on deep learning algorithms. Constr Build Mater 273:121949

[13] Lian J, Yin Y, Li L, Wang Z, Zhou Y (2021) Small object detection in traffic scenes based on attention feature fusion. Sensors 21(9):3031

[14] Lim K, Hong Y, Choi Y, Byun H (2017)Real-time traffic sign recognition based on a general purpose GPU and deep-learning. PLoS One 12(3):e0173317

[15] Liu X, YanW (2021)Traffic-light sign recognition using capsule network. Multimed Tools Appl 80:15161–15171

[16] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu C-Y, Berg AC (2016) SSD: Single shot multibox detector. European Conference on Computer Vision

[17] Qin Z, Yan W (2021)Traffic-sign recognition using deep learning. International Symposium on Geometry and Vision (ISGV). Springer, Berlin, pp 13-25

[18] Redmon J, Divvala S, Girshick R, Farhadi A (2016) You Only Look Once: Unified, real-time object detection. IEEE Conference on Computer Vision and Pattern Recognition

[19] Shi X, Hu J, Lei X, Xu S (2021) Detection of flying birds in airport monitoring based on improved YOLOv5. International Conference on Intelligent Computing and Signal Processing

[20] Sun W, Hongji D, Nie S, He X (2019) Traffic sign recognition method integrating multilayer features and kernel extreme learning machine classifier. Comput Mater Continua 60(1):147–161

[21] Wang C (2018) : Research and application of traffic sign detection and recognition based on deep learning. International Conference on Robots &; Intelligent System

[22] Wu Y, Qin X, Pan Y, Yuan C (2018) Convolution neural network based transfer learning for classification of flowers. IEEE International Conference on Signal and Image Processing

[23] Xiaoping Z, Jiahui J, Li W, Zhonghe H, Shida L (2021) People's fast moving detection method in buses based on YOLOv5. Int J Sens Sensor Netw 9(1):30

[24] Xing J, Yan W (2021) Traffic sign recognition using guided image filtering. International Symposium on Geometry and Vision (ISGV), Springer, Berlin, pp 85-99

[25] Xu S, Niu D, Tao B, Li G (2018) Convolutional neural network based traffic sign recognition system. In International Conference on Systems and Informatics (ICSAI), pp 957-961

[26] Yan B, Fan P, Lei X, Liu Z, Yang F (2021) A real-time apple target detection method for picking robots based on improved YOLOv5. Remote Sensing 13(9):1619

[27] Yao Y, Yang Y, Su X, Zhao Y, Feng A, Huang Y, Pu H (2019) Optimization of the bounding box regression process of SSD model. International Conference on Computer Engineering, Information Science & Application Technology

[28] Yu G, Fan H, Zhou H, Wu T, Zhu H (2020) Vehicle target detection method based on improved SSD model. J Artif Intell 2(3):125

[29] Zhang J, Hui L, Lu J, Zhu Y (2018)Attention-based neural network for traffic sign detection. International Conference on Pattern Recognition

APPENDIX

- **Main file**

```
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import cv2
from sklearn.model_selection import train_test_split
import pickle
import os
import pandas as pd
import random
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

################# Parameters #####################

path = "myData"  # folder with all the class folders
labelFile = 'labels.csv'  # file with all names of classes
batch_size_val = 50  # how many to process together
steps_per_epoch_val = 2000
epochs_val = 20
imageDimesions = (32, 32, 3)
testRatio = 0.2  # if 1000 images split will 200 for testing
validationRatio = 0.2  # if 1000 images 20% of remaining 800 will be 160 for validation
###################################################
```

```
################################# Importing of the Images
count = 0
images = []
classNo = []
myList = os.listdir(path)
print("Total Classes Detected:", len(myList))
noOfClasses = len(myList)
print("Importing Classes.....")
for x in range(0, len(myList)):
  myPicList = os.listdir(path + "/" + str(count))
  for y in myPicList:
      curImg = cv2.imread(path + "/" + str(count) + "/" + y)
      images.append(curImg)
      classNo.append(count)
  print(count, end=" ")
  count += 1
print(" ")
images = np.array(images)
classNo = np.array(classNo)

################################# Split Data
X_train, X_test, y_train, y_test = train_test_split(images, classNo, test_size=testRatio)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train,
test_size=validationRatio)

# X_train = ARRAY OF IMAGES TO TRAIN
# y_train = CORRESPONDING CLASS ID

################################# TO CHECK IF NUMBER OF IMAGES MATCHES TO
NUMBER OF LABELS FOR EACH DATA SET
```

```
print("Data Shapes")

print("Train", end="");

print(X_train.shape, y_train.shape)

print("Validation", end="");

print(X_validation.shape, y_validation.shape)

print("Test", end="");

print(X_test.shape, y_test.shape)

assert (X_train.shape[0] == y_train.shape[

   0]), "The number of images in not equal to the number of lables in training set"

assert (X_validation.shape[0] == y_validation.shape[

   0]), "The number of images in not equal to the number of lables in validation set"

assert (X_test.shape[0] == y_test.shape[0]), "The number of images in not equal to the number of
lables in test set"

assert (X_train.shape[1:] == (imageDimesions)), " The dimesions of the Training images are
wrong "

assert (X_validation.shape[1:] == (imageDimesions)), " The dimesionas of the Validation images
are wrong "

assert (X_test.shape[1:] == (imageDimesions)), " The dimesionas of the Test images are wrong"


################################ READ CSV FILE

data = pd.read_csv(labelFile)

print("data shape ", data.shape, type(data))


################################ DISPLAY SOME SAMPLES IMAGES  OF ALL THE
CLASSES

num_of_samples = []

cols = 5

num_classes = noOfClasses

fig, axs = plt.subplots(nrows=num_classes, ncols=cols, figsize=(5, 300))

fig.tight_layout()

for i in range(cols):

   for j, row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, len(x_selected) - 1), :, :],
cmap=plt.get_cmap("gray"))
        axs[j][i].axis("off")
        if i == 2:
            axs[j][i].set_title(str(j) + "-" + row["Name"])
            num_of_samples.append(len(x_selected))

################################ DISPLAY A BAR CHART SHOWING NO OF
SAMPLES FOR EACH CATEGORY
print(num_of_samples)
plt.figure(figsize=(12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Distribution of the training dataset")
plt.xlabel("Class number")
plt.ylabel("Number of images")
plt.show()


################################ PREPROCESSING THE IMAGES

def grayscale(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img



def equalize(img):
    img = cv2.equalizeHist(img)
    return img
```

```
def preprocessing(img):
   img = grayscale(img)  # CONVERT TO GRAYSCALE
   img = equalize(img)  # STANDARDIZE THE LIGHTING IN AN IMAGE
   img = img / 255  # TO NORMALIZE VALUES BETWEEN 0 AND 1 INSTEAD OF 0 TO
255
   return img



X_train = np.array(list(map(preprocessing, X_train)))  # TO IRETATE AND PREPROCESS
ALL IMAGES
X_validation = np.array(list(map(preprocessing, X_validation)))
X_test = np.array(list(map(preprocessing, X_test)))
cv2.imshow("GrayScale Images",
           X_train[random.randint(0, len(X_train) - 1)])  # TO CHECK IF THE TRAINING IS
DONE PROPERLY


############################### ADD A DEPTH OF 1
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_validation = X_validation.reshape(X_validation.shape[0], X_validation.shape[1],
X_validation.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)


############################### AUGMENTATAION OF IMAGES: TO MAKEIT
MORE GENERIC
dataGen = ImageDataGenerator(width_shift_range=0.1,
                             # 0.1 = 10%     IF MORE THAN 1 E.G 10 THEN IT REFFERS TO NO. OF
PIXELS EG 10 PIXELS
                             height_shift_range=0.1,
                             zoom_range=0.2,  # 0.2 MEANS CAN GO FROM 0.8 TO 1.2
                             shear_range=0.1,  # MAGNITUDE OF SHEAR ANGLE
                             rotation_range=10)  # DEGREES
dataGen.fit(X_train)
```

```
batches = dataGen.flow(X_train, y_train,
                       batch_size=20)  # REQUESTING DATA GENRATOR TO GENERATE
IMAGES  BATCH SIZE = NO. OF IMAGES CREAED EACH TIME ITS CALLED
X_batch, y_batch = next(batches)

# TO SHOW AGMENTED IMAGE SAMPLES
fig, axs = plt.subplots(1, 15, figsize=(20, 5))
fig.tight_layout()

for i in range(15):
    axs[i].imshow(X_batch[i].reshape(imageDimesions[0], imageDimesions[1]))
    axs[i].axis('off')
plt.show()

y_train = to_categorical(y_train, noOfClasses)
y_validation = to_categorical(y_validation, noOfClasses)
y_test = to_categorical(y_test, noOfClasses)


############################### CONVOLUTION NEURAL NETWORK MODEL
def myModel():
    no_Of_Filters = 60
    size_of_Filter = (5, 5)  # THIS IS THE KERNEL THAT MOVE AROUND THE IMAGE TO
GET THE FEATURES.
    # THIS WOULD REMOVE 2 PIXELS FROM EACH BORDER WHEN USING 32 32
IMAGE
    size_of_Filter2 = (3, 3)
    size_of_pool = (2, 2)  # SCALE DOWN ALL FEATURE MAP TO GERNALIZE MORE, TO
REDUCE OVERFITTING
    no_Of_Nodes = 500  # NO. OF NODES IN HIDDEN LAYERS
    model = Sequential()
```

```
   model.add((Conv2D(no_Of_Filters, size_of_Filter, input_shape=(imageDimesions[0],
imageDimesions[1], 1),
                      activation='relu')))  # ADDING MORE CONVOLUTION LAYERS = LESS
FEATURES BUT CAN CAUSE ACCURACY TO INCREASE
   model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='relu')))
   model.add(MaxPooling2D(pool_size=size_of_pool))  # DOES NOT EFFECT THE
DEPTH/NO OF FILTERS

   model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
   model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
   model.add(MaxPooling2D(pool_size=size_of_pool))
   model.add(Dropout(0.5))

   model.add(Flatten())
   model.add(Dense(no_Of_Nodes, activation='relu'))
   model.add(Dropout(0.5))  # INPUTS NODES TO DROP WITH EACH UPDATE 1 ALL 0
NONE
   model.add(Dense(noOfClasses, activation='softmax'))  # OUTPUT LAYER
   # COMPILE MODEL
   model.compile(Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
   return model


############################### TRAIN
model = myModel()
print(model.summary())
history = model.fit(dataGen.flow(X_train, y_train, batch_size=batch_size_val),
                    steps_per_epoch=len(X_train)//batch_size_val, epochs=epochs_val,
                    validation_data=(X_validation, y_validation), shuffle=1)

############################### PLOT
plt.figure(1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('loss')
plt.xlabel('epoch')
plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'])
plt.title('Acurracy')
plt.xlabel('epoch')
plt.show()
score = model.evaluate(X_test, y_test, verbose=0)
print('Test Score:', score[0])
print('Test Accuracy:', score[1])

# STORE THE MODEL AS A PICKLE OBJECT
pickle_out= open("model_trained.p","wb")  # wb = WRITE BYTE
pickle.dump(model,pickle_out)
pickle_out.close()
cv2.waitKey(0)
```

- **Test file**

```
import numpy as np
import cv2
import pickle

#############################################

frameWidth = 640  # CAMERA RESOLUTION
```

```
frameHeight = 480
brightness = 180
threshold = 0.75  # PROBABLITY THRESHOLD
font = cv2.FONT_HERSHEY_SIMPLEX
##############################################

# SETUP THE VIDEO CAMERA
cap = cv2.VideoCapture(0)
cap.set(3, frameWidth)
cap.set(4, frameHeight)
cap.set(10, brightness)
# IMPORT THE TRANNIED MODEL
pickle_in = open("model_trained.p", "rb")  ## rb = READ BYTE
model = pickle.load(pickle_in)



def grayscale(img):
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  return img

def equalize(img):
  img = cv2.equalizeHist(img)
  return img

def preprocessing(img):
  img = grayscale(img)
  img = equalize(img)
  img = img / 255
  return img
```

```
def getCalssName(classNo):
   if classNo == 0:
       return 'Speed Limit 20 km/h'
   elif classNo == 1:
       return 'Speed Limit 30 km/h'
   elif classNo == 2:
       return 'Speed Limit 50 km/h'
   elif classNo == 3:
       return 'Speed Limit 60 km/h'
   elif classNo == 4:
       return 'Speed Limit 70 km/h'
   elif classNo == 5:
       return 'Speed Limit 80 km/h'
   elif classNo == 6:
       return 'End of Speed Limit 80 km/h'
   elif classNo == 7:
       return 'Speed Limit 100 km/h'
   elif classNo == 8:
       return 'Speed Limit 120 km/h'
   elif classNo == 9:
       return 'No passing'
   elif classNo == 10:
       return 'No passing for vechiles over 3.5 metric tons'
   elif classNo == 11:
       return 'Right-of-way at the next intersection'
   elif classNo == 12:
       return 'Priority road'
   elif classNo == 13:
       return 'Yield'
   elif classNo == 14:
       return 'Stop'
   elif classNo == 15:
       return 'No vehicles'
   elif classNo == 16:
       return 'Vehicles over 3.5 metric tons prohibited'
   elif classNo == 17:
       return 'No entry'
   elif classNo == 18:
       return 'General caution'
   elif classNo == 19:
       return 'Dangerous curve to the left'
   elif classNo == 20:
       return 'Dangerous curve to the right'
   elif classNo == 21:
       return 'Double curve'
   elif classNo == 22:
       return 'Bumpy road'
   elif classNo == 23:
       return 'Slippery road'
   elif classNo == 24:
       return 'Road narrows on the right'
   elif classNo == 25:
       return 'Road work'
   elif classNo == 26:
       return 'Traffic signals'
   elif classNo == 27:
       return 'Pedestrians'
   elif classNo == 28:
       return 'Children crossing'
   elif classNo == 29:
       return 'Bicycles crossing'
   elif classNo == 30:
       return 'Beware of ice/snow'
   elif classNo == 31:
       return 'Wild animals crossing'
   elif classNo == 32:
       return 'End of all speed and passing limits'
   elif classNo == 33:
       return 'Turn right ahead'
   elif classNo == 34:
       return 'Turn left ahead'
   elif classNo == 35:
       return 'Ahead only'
   elif classNo == 36:
       return 'Go straight or right'
   elif classNo == 37:
       return 'Go straight or left'
   elif classNo == 38:
       return 'Keep right'
   elif classNo == 39:
       return 'Keep left'
   elif classNo == 40:
       return 'Roundabout mandatory'
   elif classNo == 41:
       return 'End of no passing'
   elif classNo == 42:
       return 'End of no passing by vehicles over 3.5 metric tons'


while True:
   # READ IMAGE
   success, imgOrignal = cap.read()

   # PROCESS IMAGE
   img = np.asarray(imgOrignal)
   img = cv2.resize(img, (320, 320))
   img = preprocessing(img)
   cv2.imshow("Processed Image", img)
   img = img.reshape(1, 32, 32, 1)
   cv2.putText(imgOrignal, "CLASS: ", (20, 35), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
   cv2.putText(imgOrignal, "Accuracy: ", (20, 75), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)

   # PREDICT IMAGE
   predictions = model.predict(img)
   classIndex = model.predict_classes(img)
   probabilityValue = np.amax(predictions)
   if probabilityValue > threshold:

       print(getCalssName(classIndex))
       cv2.putText(imgOrignal, str(classIndex) + " " + str(getCalssName(classIndex)), (120, 35),
font, 0.75, (0, 0, 255), 2,
            cv2.LINE_AA)
       cv2.putText(imgOrignal, str(round(probabilityValue * 100, 2)) + "%", (180, 75), font, 0.75,
(0, 0, 255), 2, cv2.LINE_AA)
       cv2.imshow("Result", imgOrignal)

   if cv2.waitKey(1) and 0xFF == ord('q'):
       break
```