Traffic Sign Classification

Project Report Submitted by

Ankith Bhandary
(4NM17CS020)

Dhruv Shetty
(4NM17CS057)

Kishan
(4NM17CS090)

Manukashyap U V
(4NM17CS101)

UNDER THE GUIDANCE OF

Mr. Ramesha Shettigar
Asst. Professor
Dept. of CSE, NMAMIT, NITTE

in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Engineering
(Computer Science & Engineering)

from

Visvesvaraya Technological University

Department of Computer Science and Engineering
NMAM Institute of Technology, Nitte – 574110
(An Autonomous Institution affiliated to VTU Belgaum)

July 2021

CERTIFICATE

Certified that the project work entitled ***Traffic Sign Classification*** *is a bonafide work carried out by Ankith Bhandary (4NM17CS020), Dhruv Shetty (4NM17CS057), Kishan (4NM17CS090), Manukashyap U V (4NM17CS101) in partial fulfilment for the award of Degree of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2020-2021. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project prescribed for the said Degree.*

Name & Signature of Guide **Signature of HOD** **Signature of the Principal**

External Viva

Name of the Examiners	**Signature with Date**
1.	
2.	

ACKNOWLEDGEMENT

The satisfaction that accompanies the completion of any task would be incomplete without the mention of all the people, without whom this endeavour would have been a difficult one to achieve. Their constant blessings, encouragement, guidance and suggestions have been a constant source of inspiration.

First and foremost, our gratitude to our project guide, **Mr. Ramesha Shettigar** for his constant guidance throughout the course of this project and for the valuable suggestions.

We also take this opportunity to express a deep sense of gratitude to the project coordinators for their valuable guidance and support.

We acknowledge the support and valuable inputs given by, **Dr. Jyothi Shetty,** the Head of the Department, Computer Science and Engineering, NMAMIT, Nitte.

Our sincere thanks to our beloved principal, **Dr. Niranjan N Chiplunkar** for permitting us to carry out this project at our college and providing us with all the needed facilities.

Finally, thanks to staff members of the Department of Computer Science and Engineering and our friends for their honest opinions and suggestions throughout the course of our project.

Ankith Bhandary (4NM17CS020)

Dhruv Shetty (4NM17CS057)

Kishan (4NM17CS090)

Manukashyap U V (4NM17CS101)

ABSTRACT

With the recent revelation that fossil fuels will dry up globally in the next 25 to 30 years, the automobile industry looked in various directions in search of a reliable alternative to help us ease into an age without petrol and diesel. The most popular of the options that came to be, is that of electric vehicles. Companies like Tesla burst onto the scene with their electric cars, set to help the customers rely on electricity rather than fossils to help them get to their destinations.

Another key idea that electric vehicles brought with their uprise was the need to automate the vehicles. Autopilot system, road trip planner and other ideas turned out to be crucial to elevate electric vehicles above their diesel and petrol counterparts. One of the components leading to the success of the autopilot system was the inclusion of traffic sign classification.

Even though traffic signs have been around for ages now, people tend to ignore most of them, of which the speed limit is the most prominent. If we can create a machine learning model that will detect and classify the traffic signs, the vehicle can take over, under the scenario where the driver ignores the signs. While autopilot isn't recommended as of now due to its many intricacies, traffic sign classification is a must, and has to be instilled to improve the driving experience.

With this project we aim to create a machine learning model that is trained over a benchmark set of traffic signs and be able to detect most of the traffic signs with most accuracy. We also aim to create a camera interface to this model that can recognise the traffic signs in front of it in close to real time.

TABLE OF CONTENTS

TRAFFIC SIGN CLASSIFICATION **1**

ACKNOWLEDGEMENT **3**

ABSTRACT **4**

TABLE OF CONTENTS **5**

TABLE OF FIGURES **6**

INTRODUCTION **7**

1.1 COMPUTER VISION 7
1.2 CONVOLUTION NEURAL NETWORK 7
1.3 COMPUTER VISION IN AUTOMOBILES 7
1.4 TRAFFIC SIGN CLASSIFICATION 8

LITERATURE SURVEY **9**

2.1 EXISTING SYSTEMS 9
2.2 PROPOSED SYSTEM 10

PROBLEM DEFINITION **11**

3.1 PROBLEM STATEMENT 11
3.2 PROBLEM DEFINITION 11

SYSTEM REQUIREMENTS SPECIFICATION **12**

4.1 HARDWARE REQUIREMENTS 12
4.2 SOFTWARE REQUIREMENTS 12

SYSTEM DESIGN **13**

5.1 CONVOLUTION OPERATION 14
5.2 MAX POOLING 14
5.3 MODEL SUMMARY 15

IMPLEMENTATION **17**

6.1 IMPORTING THE REQUIRED LIBRARIES 17
6.2 SETTING UP THE PARAMETERS 17
6.3 IMPORTING THE IMAGES 17
6.4 SPLITTING THE DATA 18
6.5 DATASET VALIDATION 18
6.6 READING THE LABEL FILE AND DISPLAYING SAMPLE IMAGES 18
6.7 DISPLAY INFORMATION ABOUT THE DATA SET 19
6.8 PRE-PROCESSING THE IMAGES 20
6.9 IMAGE AUGMENTATION 21
6.10 CREATING THE CONVOLUTION NEURAL NETWORK 21
6.11 TRAINING THE MODEL 22
6.12 PLOTTING THE LOSS AND VARIATION GRAPH 22
6.13 SAVING THE MODEL 23
6.14 CREATING THE GUI FOR THE PRACTICAL APPLICATION OF THE MODEL 23

RESULTS AND CONCLUSION **26**

7.1 RESULTS 26
7.1.1 Comparison with other systems *26*
CONCLUSION AND FUTURE WORKS 28
8.1 CONCLUSION 28
8.2 FUTURE WORK 28
8.2.1 Dataset *28*
8.2.2 Refinement *28*
8.2.3 Data Logging *29*
8.2.4 Other Implementation *29*
REFERENCES 30
9.1 PUBLISHED PAPERS: 30
9.2 BOOKS, VIDEOS AND STUDY MATERIALS: 30

TABLE OF FIGURES

Figure 1 Abstract View of the Model 13
Figure 2 Detailed View of the Model 13
Figure 3 Convolution Operation 14
Figure 4 Max pooling Operation 15
Figure 5 Welcome Screen 24
Figure 6 OpenCV Interface 24

CHAPTER - 1

INTRODUCTION

1.1 COMPUTER VISION

Definition: “Computer vision is an interdisciplinary scientific field that deals with how computers can gain a high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.”

Computer vision enables the computer to partially interact with the real world and gain valuable data over time that will enable next level of automation in the day-to-day tasks leading to a safer and comfortable world for humans while also decreasing the global energy footprint.

1.2 CONVOLUTION NEURAL NETWORK

Definition: “In deep learning, a convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics.”

Neural networks try to mimic the behaviour of the human brain to learn from real-world data and to solve real-world problems in the process. Convolution neural networks are one of the more efficient ways of using machine learning on images since they are capable of differentiating and identifying the different characters in each image, thereby accurately predicting the image class.

1.3 COMPUTER VISION IN AUTOMOBILES

Lately, the use of computer vision in the field has skyrocketed. Computer vision provides unprecedented security and comfort for the automobile and its occupants.

Some of these services are collision detention, traffic sign detection, driver awareness detection, side collision detection, traffic awareness etc. But these features require a huge amount of Research and Development (R&D) budget and also highly capable processing units to be installed on the vehicles. Thus, these features are only available in luxury vehicles whose cost would be the entire lifetime earnings of a common man a century ago.

1.4 TRAFFIC SIGN CLASSIFICATION

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

Traffic signs come in various shapes and colours. This helps strengthen the system by providing an extra layer of segmentation. Speed limit and stop signs are always in a circular sign board, while construction and road work ahead signs come inside a triangular sign board. This helps optimize the system to be designed. Another category is that of colour. All cautionary signs have been designed with bright red colour across the sign board. This helps the drivers notice the signboard even when the visibility is comparatively lower than that on a clear day. These two parameters have been documented during the early days of traffic sign designing and thus helps improve the system’s functioning.

CHAPTER - 2

LITERATURE SURVEY

Over the years with the innovation and advent of electric vehicles the concept of autonomous driver assistant systems have become entities of higher importance. Although most of the systems used by a few of the high profile vehicle manufactures are closed source and sophisticated, through paper publications and open-source licensed software we can learn about these systems. Even though the effort to autonomizing the transportation systems is going on for a long time, it is still in its infancy thus a lot of effort from throughout the world is required to move past the current limitations.

2.1 EXISTING SYSTEMS

In the case of traffic sign classification, there are numerous ways to solve the problem. Each method has its advantages and disadvantages.

In the 80s the infant systems mainly consisted of two-fold detection stages consisting of shape-based detection and content-based detection. Some papers use the contracting curve density algorithm to refine the data set and using a multi-layered perceptron for classification. The accuracy of these systems was low and they could not detect many traffic signs. [1]

Some systems use a high amount of image processing such as colour segmentation, blob detection, binarization and ostu thresholding etc. [5]. But these lack the speediness required for real-time detections.

Deep learning has been proven to work well in solving the problem of traffic sign classification. There have been papers of using R-CNN[2] or spatial transformation, GoogleNet.[4] They achieve high enough accuracy but sometimes tend to be complex and require high computing power to train and run the models.

2.2 PROPOSED SYSTEM

We propose to use convolution neural networks using TensorFlow with Keras interface to solve the problem of traffic sign classification.

Our data set consists of over 35000 images of traffic signs of different shapes and sizes in a different environment, timing and seasons. We import the images and remove any imperfect images that might affect the model. We equalise the dimensions and maintain a constant colour space among the images. Later the images will be standardised in terms of lighting by equalising their histograms.

The training data will be passed to our CNN model having multiple convolutions and max-pooling layers and it will be run over multiple epoch and the constraints are modified till we get consistent and satisfactory results.

We will produce a prototype interface to showcase the capabilities of the model using eel python library which enables us to harness the powers of web designing in python applications.

We aim to create a model with high accuracy and a prototype light enough to run on systems with low computing power.

CHAPTER - 3

PROBLEM DEFINITION

3.1 PROBLEM STATEMENT

Creation of an automatic, fast and light system that is capable of real-time detection, classification and interpretation of the traffic signs.

3.2 PROBLEM DEFINITION

Traffic sign classification is one of the most important uses of computer vision in automobiles. It has several aspects that include differentiating the road and surroundings, identifying the sign poles, closing in on the traffic sign and classifying it. We plan to tackle the issue of classifying the traffic signs once we have the sign in the view of the camera since it is the basic block of a traffic sign classification system.

Our goal is to create a machine learning model that is capable of classifying most of the traffic signals and also enable real-time interpretation of those signals. We aim to create an end product that can see the traffic signal through the camera and can interpret it onto the display while also being light on its resources.[8]

We plan to use OpenCV for the image input part of the system and the CNN model for the classification and interpretation of the traffic signs.[6]

CHAPTER - 4

SYSTEM REQUIREMENTS SPECIFICATION

4.1 HARDWARE REQUIREMENTS

- Processor: Intel i5 or equivalent processor.
- GPU: Tesla P100 16GB VRAM.
- RAM: 8GB or above.
- Storage: 5GB or above.

4.2 SOFTWARE REQUIREMENTS

- Language: Python 3
- Platform: Anaconda or Kaggle
- Environment: Conda
- Notebook: Jupyter
- Libraries:
    - Eel
    - Keras
    - TensorFlow
    - OpenCV
    - NumPy
    - Pandas
    - Sklearn

CHAPTER - 5

SYSTEM DESIGN

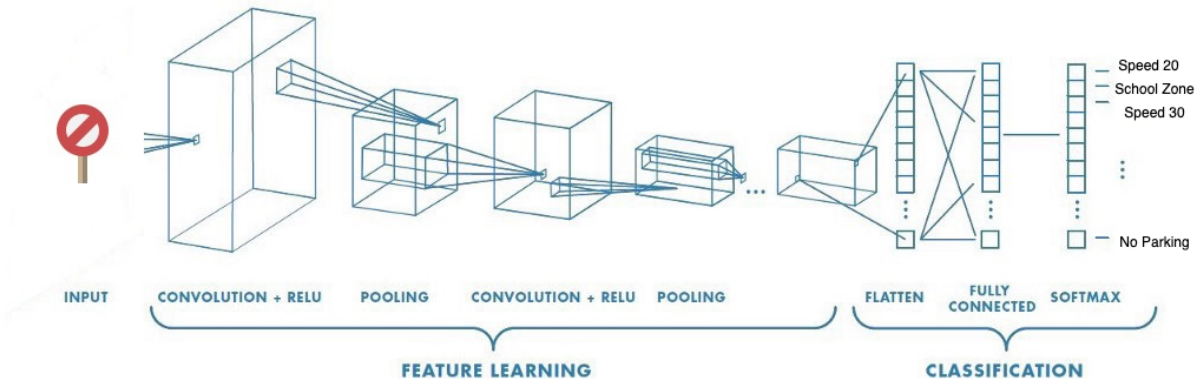


Figure 1 Abstract View of the Model

CNN is a machine learning algorithm that can take an input image, assign importance to various aspects or objects in the image, and be able to differentiate one from another.[7]

[6]CNN works by extracting the features from the images. Any CNN consists of the following:

1. The input layer which is a grayscale image.
2. The output layer is a binary or multi-class label.
3. Hidden layers consist of convolution layers, ReLU (Rectified Linear Unit) layers, the pooling layer, and a fully connected Neural Network.

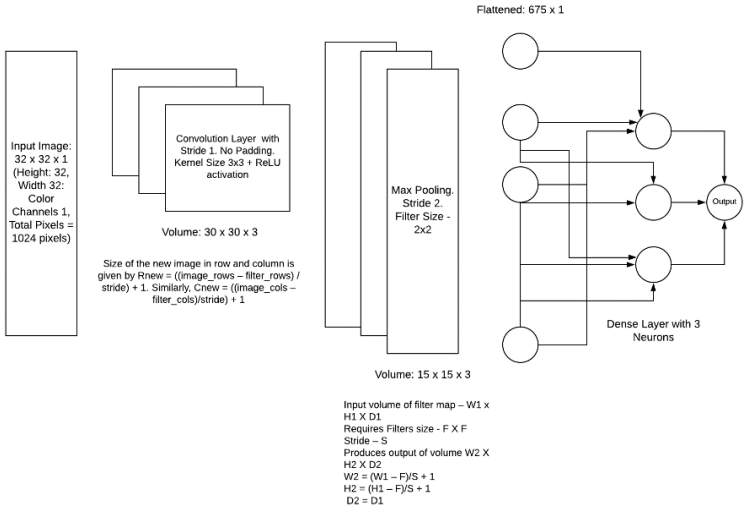


Figure 2 Detailed View of the Model

5.1 CONVOLUTION OPERATION

A few samples of convolution operation happening on a 4x4 image matrix with 2x2 kernel and with stride=1

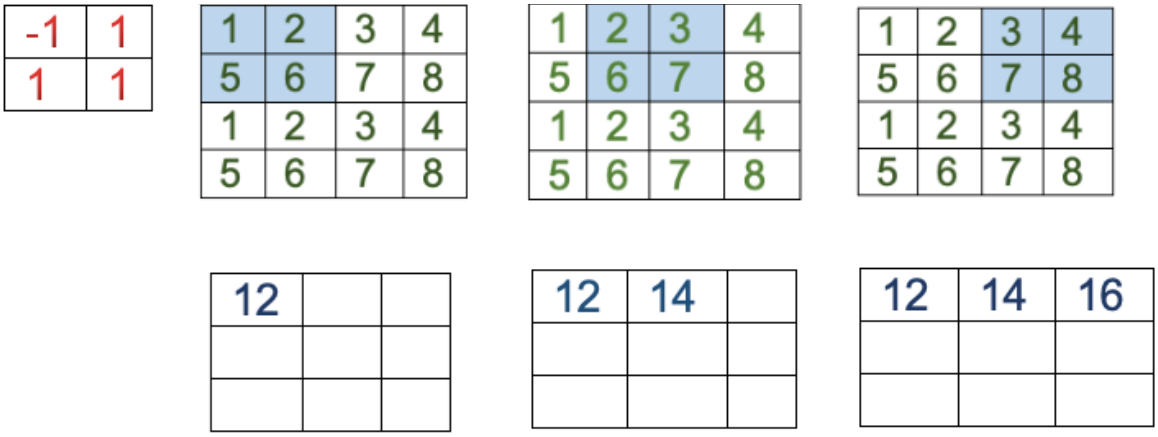


Figure 3 Convolution Operation

Here,

12=-1x1+1x2+1x5+1x6

14=-1x2+1x3+1x6+1x7

16=-1x3+1x4+1x7+1x8

When performing convolutions in the qth layer, one can align the filter at $L_{q+1} = L_q - F_q + 1$ positions along the height and $B_{q+1} = B_q - F_q + 1$ along the width of the image. This results in a total of $L_{q+1} \times B_{q+1}$ possible dot products, which defines the size of the next hidden layer. In the previous example, the values of L_2 and B_2 are therefore defined as follows:

$L_2 = 4 - 2 + 1 = 3$

$B_2 = 4 - 2 + 1 = 3$

5.2 MAX POOLING

A sample of max pooling operation happening on a 4x4 image matrix with 2x2 pooling with stride =2

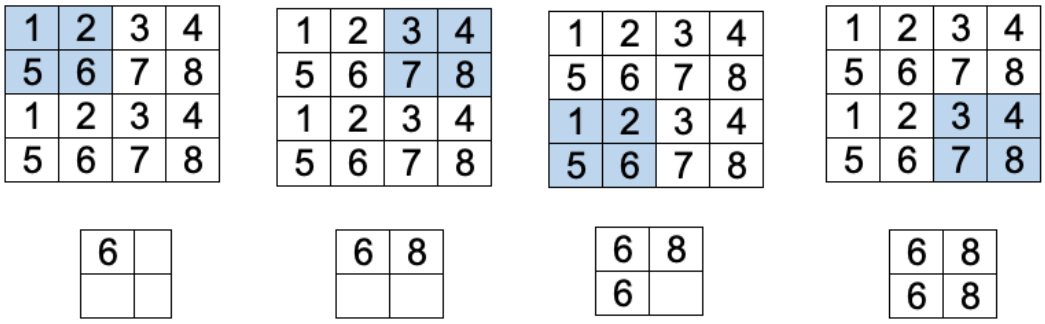


Figure 4 Max pooling Operation

For stride $S_q > 1$ is used in pooling. In such cases, the length of the new layer will be $(L_q - P_q)/S_q + 1$ and the breadth will be $(Bq - Pq)/Sq + 1$

$L_2 = (4 - 2) / 2 + 1 = 2$

$B_2 = (4 - 2) / 2 + 1 = 2$

5.3 MODEL SUMMARY

Conv2D Filters = 60, Size of filter = (5,5) input shape= 32x32x3, ReLU activation function. So $L_2 = (32-5+1) = 28$, $B_2 = (32-5+1) = 28$ and there are 60 filters Hence output shape is **(28,28,60)**
Conv2D_1 Filters = 60, Size of filter = (5,5) input shape = 28x28x60, relu activation function. So $L_3 = (28-5+1) = 24$, $B_3 = (28-5+1) = 24$ and there are 60 filters Hence output shape is **(24,24,60)**
MaxPooling2D Pool size = (2,2), Stride= 2 Input shape = 24x24x60 So $L_4 = (24-2)/2+1 = 12$, $B_4 = (24-2)/2+1 = 12$ Hence output shape is **(12,12,60)**
Conv2D_2 Filters = 30, Size of filter = (3,3) input shape = 12x12x60, relu activation function.

So L_5 = (12-3+1) = 10, B_5= (12-3+1) = 10 and there are 30 filters Hence output shape is **(10,10,30)**
Conv2D_3 Filters = 30, Size of filter = (3,3) input shape = 10x10x30, ReLU activation function. So L_6 = (10-3+1) = 8, B_6= (10-3+1) = 8 and there are 30 filters Hence output shape is **(8,8,30)**
MaxPooling2D Pooli size = (2,2), Stride= 2 Input shape = 8x8x30 So L_7 = (8-2)/2+1 = 4, B_7 = (8-2)/2+1 = 4 Hence output shape is **(4, 4, 30)**
Dropout Drop out applied = 0.5 Output shape = **(4,4,30)**
Flatten Here data is converted into 1-D array Output shape is 4x4x30 = (**480)**
Dense Number of nodes = 500 Output shape is **(500)**
Dropout_1 Drop out applied = 0.5 Output shape = **(500)**
Dense_1 Number of classes (nodes) = 43 Output shape = **(43)**

IMPLEMENTATION

6.1 IMPORTING THE REQUIRED LIBRARIES

```
import NumPy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from 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
from keras.preprocessing.image import ImageDataGenerator
```

6.2 SETTING UP THE PARAMETERS

Different parameters such as the path for the dataset, batch size, epochs, steps per epoch etc are set up.

```
data_path = "../input/tsc-data-set/myData/myData"  # folder with all the class folders
label_file = "../input/tsc-data-set/labels.csv"  # file with all names of classes
batch_size = 50  # how many to process together
steps_per_epoch = 400
epochs = 50
image dimensions = (32, 32, 3)
test_ratio = 0.2    # if 1000 images split will 200 for testing
validation_ratio = 0.2  # if 1000 images 20% of remaining 800 will be 160 for validation
```

6.3 IMPORTING THE IMAGES

The images are imported from the dataset and loaded into the kernel for further processing.

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

```
images = np.array(images)
classNo = np.array(classNo)

Output
Total Classes Detected: 43
Importing Classes.....
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42
```

6.4 SPLITTING THE DATA

The dataset is split into test, train and validation sets.

```
X_train, X_test, y_train, y_test = train_test_split(
    images, classNo, test_size=test_ratio)
X_train, X_validation, y_train, y_validation = train_test_split(
    X_train, y_train, test_size=validation_ratio)
```

6.5 DATASET VALIDATION

The dataset and the label files are compared to check if there are any mismatches in the dataset.

```
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:] == (image_dimensions)
       ), " The dimesions of the Training images are wrong "
assert(X_validation.shape[1:] == (image_dimensions)
       ), " The dimesionas of the Validation images are wrong "
assert(X_test.shape[1:] == (image_dimensions)
       ), " The dimesionas of the Test images are wrong"
Output

Data Shapes
Train(22271, 32, 32, 3) (22271,)
Validation(5568, 32, 32, 3) (5568,)
Test(6960, 32, 32, 3) (6960,)
```

6.6 READING THE LABEL FILE AND DISPLAYING SAMPLE IMAGES

Each label is read from the CSV file and corresponding images in the dataset is displayed for visual validation of the dataset.

```
# READ CSV FILE
data = pd.read_csv(label_file)
print("data shape ", data.shape, type(data))
# DISPLAY SOME SAMPLES IMAGES  OF ALL THE CLASSES
num_of_samples = []
cols = 5
num_classes = noOfClasses
```

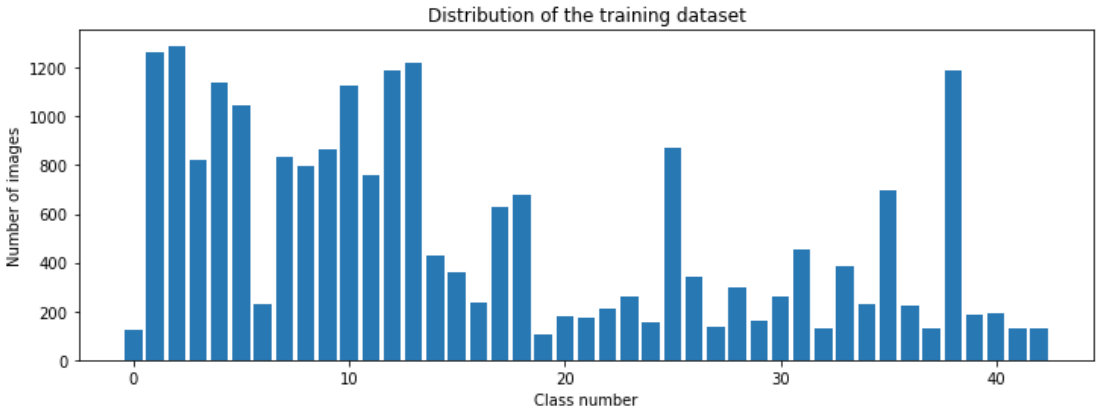
```
fig, axs = plt.subplots(nrows=num_classes, ncols=cols, figsize=(5, 100))
fig.tight_layout()
for i in range(cols):
    for j, row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(
            0, len(x_selected) - 1), :, :], cmap=plt.get_cmap("gray"))
        axs[j][i].axis("off")
        if i == 2:
            axs[j][i].set_title(str(j) + "-"+row["Name"])
            num_of_samples.append(len(x_selected))
```

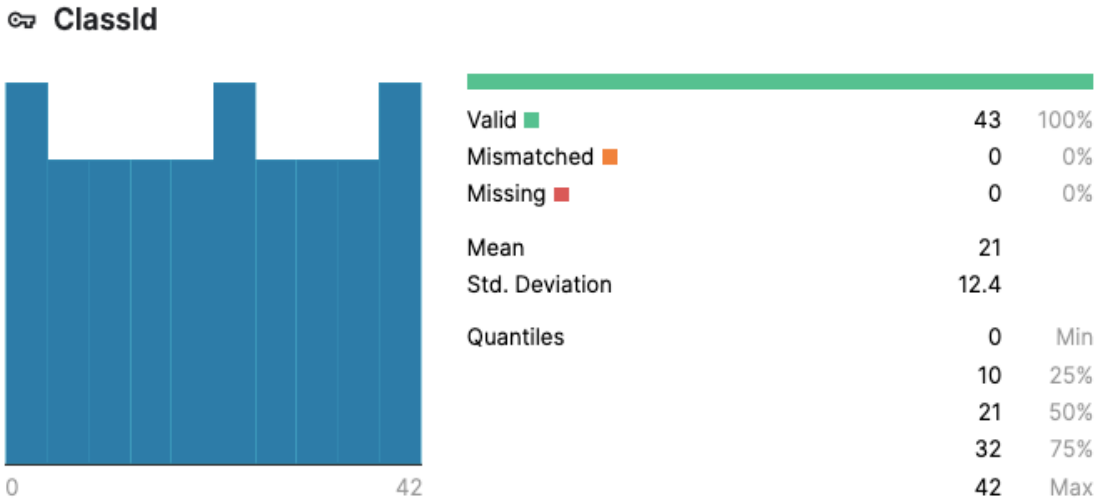
Output

6.7 DISPLAY INFORMATION ABOUT THE DATA SET

Some information about the distribution of images of different classes in the dataset is displayed.

```
# 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()
```





6.8 PRE-PROCESSING THE IMAGES

All of the images in the dataset is pre-processed to make all of them have the same dimensions and all of them are converted to grey scale images for easier processing.

```
# 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
# TO IRETATE AND PREPROCESS ALL IMAGES
X_train = np.array(list(map(preprocessing, X_train)))
```

```
X_validation = np.array(list(map(preprocessing, X_validation)))
X_test = np.array(list(map(preprocessing, X_test)))
# 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)
```

6.9 IMAGE AUGMENTATION

```
# 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)
# REQUESTING DATA GENRATOR TO GENERATE IMAGES  BATCH SIZE = NO. OF IMAGES CREAED EACH TIME
ITS CALLED
batches = dataGen.flow(X_train, y_train, batch_size=20)
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(image_dimensions[0], image_dimensions[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)
```

Output

6.10 CREATING THE CONVOLUTION NEURAL NETWORK

After experimenting with different model parameters the most ideal CNN model is setup.

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

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

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

6.11 TRAINING THE MODEL

```
# TRAIN
model = myModel()
print(model.summary())
history = model.fit_generator(dataGen.flow(X_train, y_train, batch_size=batch_size),
                              steps_per_epoch=steps_per_epoch, epochs=epochs,
validation_data=(X_validation, y_validation), shuffle=1)
```

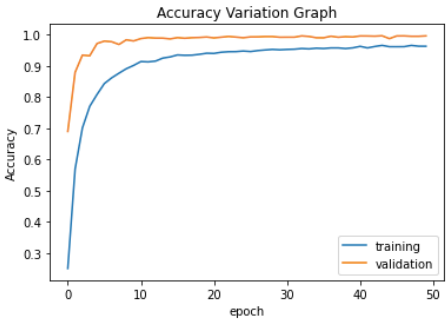
6.12 PLOTTING THE LOSS AND VARIATION GRAPH

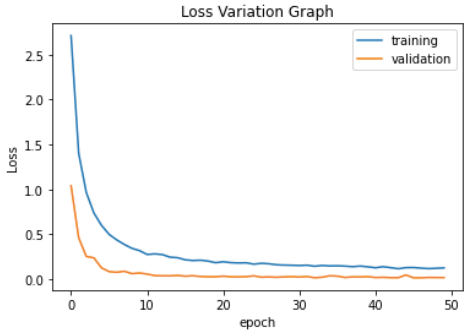
The graph of how the accuracy and loss varies with each of the epoch is plotted.

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

Output

```
Test Score: 0.012194344773888588
Test Accuracy: 0.995976984500885
```





6.13 SAVING THE MODEL

The model is saved to the disk in the form of JSON and HDF5 for future operations.

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

6.14 CREATING THE GUI FOR THE PRACTICAL APPLICATION OF THE MODEL

We are using eel python library to write the GUI code for the application using HTML, CSS and JavaScript.

We use the bootstrap framework to create the container, div and cards and custom CSS for animation rendering. The GUI also points to the open sourced code for this particular application of the model stored in GitHub.

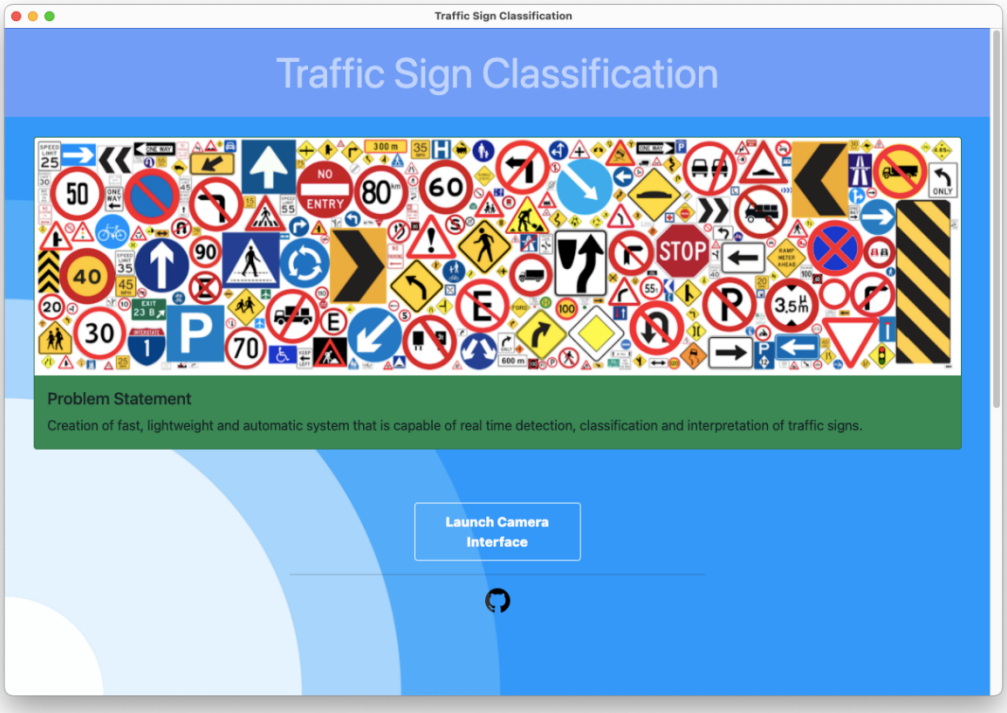


Figure 5 Welcome Screen

6.15 CREATING THE OPENCV INTERFACE FOR THE PRACTICAL APPLICATION OF THE MODEL

We also created a OpenCV webcam interface for the model to identify the traffic signs. Our practical demonstration also has a logging system that will log the identified signs along with the probability and identified time for review and safety purposes.

Figure 6 OpenCV Interface

The contents of the log file:

```
No: 11 Time: 20:48:04
[14] Stop 97.35%
-----------------------------------------------------

No: 12 Time: 20:48:05
[12] Priority road 95.93%
-----------------------------------------------------
No: 13 Time: 20:48:05
[14] Stop 92.57%
-----------------------------------------------------
No: 14 Time: 20:48:05
[14] Stop 99.61%
-----------------------------------------------------
No: 15 Time: 20:48:05
[12] Priority road 93.11%
-----------------------------------------------------
```

RESULTS AND CONCLUSION

7.1 RESULTS

1. The dataset obtained has been cleaned before processing and made error free.
2. The distribution of images across different classes of traffic signs has been deemed satisfactory and has been optimized to provide higher accuracy.
3. The CNN model was created, trained and further refined to achieve the theoretical accuracy as per the intended objective.
4. Serialization of the model was done successfully using JSON for the model and HDF5 for the weights.
5. Given ideal conditions, the practical application of the model provided optimal outputs, and has been recorded to be highly accurate.
6. All the objectives set for the model have been satisfied within the stipulated time with minimal errors.
7. The logging system adopted has shown to work well under the given conditions, with limiters set to achieve precision readings.

7.1.1 Comparison with other systems

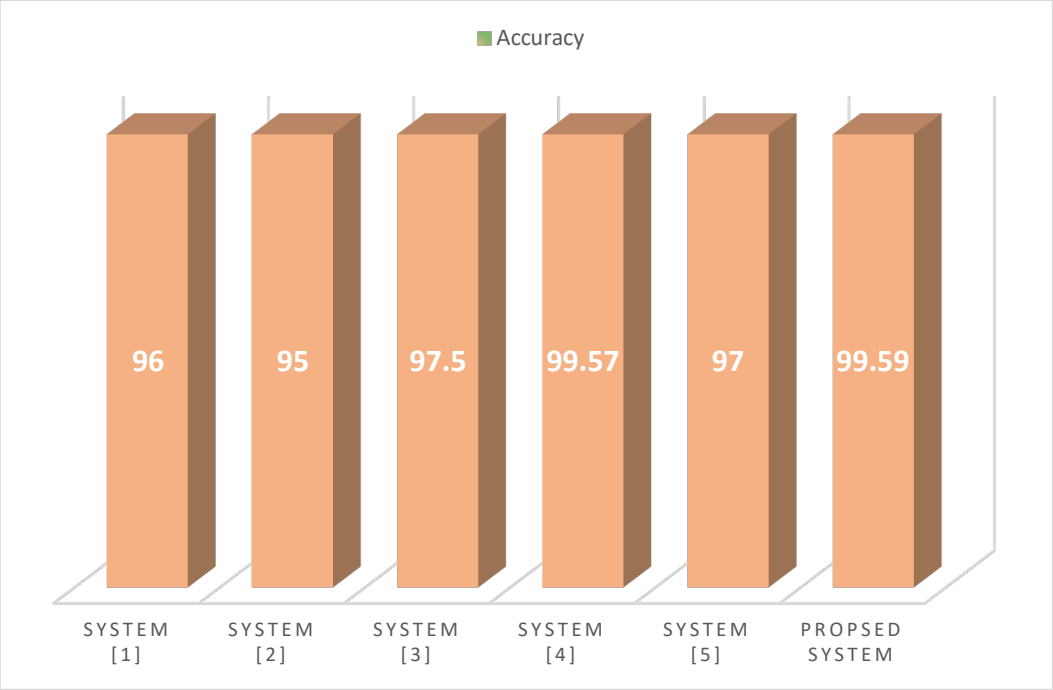
Comparing the different features of the systems seen in the literature survey with our proposed system.

Features	**System [1]**	**System [2]**	**System [3]**	**System [4]**	**System [5]**	**Proposed System**
Signs recognition	Detects only circular signs	Doesn't detect octagonal signs	Detects all signs	Detects all signs	Detects all signs	Detects all signs

Training Time	Less	Less	Very high	High	Less	Less
Tilted Board recognition	No	No	Yes	Yes	Yes	Yes
Occluded Boards recognition	No	No	Yes	Yes	Yes	Yes
Colour Independent Recognition	No	No	Yes	Yes	No	Yes

This shows that our proposed system is competent with the already existing systems and is able to be ahead of some of them.

Comparing the accuracy of all the systems also shows that the proposed system is one of the most competent of the bunch.



CHAPTER - 8

CONCLUSION AND FUTURE WORKS

8.1 CONCLUSION

The idea of application of convolution neural networks to solve the problem pertaining to traffic sign classification has proven to provide a feasible solution, and the results obtained on deploying this model have been satisfactory. Over time, this technique can be refined and can be used in real life to great effect.

The model designed with necessary tweaking and improvements with new technology can be incorporated into vehicles to enable heightened automation. The system can be optimized to work under different weather scenarios. Varying visibility of the road can also be taken into account when upgrading the model. The logging system can be optimized to provide more details regarding the driving experience. Road sign frequency and driver response can be recorded to make the model learn the driver better, and thereby force instructions if the driver tries to ignore the traffic signs.

8.2 FUTURE WORK

8.2.1 Dataset

The dataset used during the development stages of this model is heavily reliant on GTSRB. Although the dataset is a benchmark, it also misses out on newer traffic signs that have come into existence over the last few years. Using these new signs and their corresponding sign classes is important to make sure the system stays relevant.

8.2.2 Refinement

There is periodic refinement required when a model is designed to capture real life scenarios. Same can be considered in terms of this model. Constant refinement will help improve the accuracy of the model.

8.2.3 Data Logging

A late addition to this model was the logging system, wherein the model logs in all the traffic signs encountered in a given time period. This system works well given the circumstances, but further improvements such as multiple logging data, decision chart and other graphical metrics can be drawn to better enhance the system's applicability.

8.2.4 Other Implementation

The model designed is exclusive for traffic signs. But it's base idea can be used to design other equally important models. An example would be that of car dashboard. A model can be designed which reacts to the car dashboard, notifying the driver of the current condition of the car, and also providing quick solutions or other info regarding it.

CHAPTER - 9

REFERENCES

9.1 Published Papers:

[1]. **Towards Reliable Traffic Sign Recognition**
Benjamin Hoferlin - Intelligent Systems Group Universitat Stuttgart Stuttgart, Germany benjamin.hoeferlin@vis.uni-stuttgart.de
Klaus Zimmermann - European Technology Center (ESTEC) Sony Deutschland GmbH Stuttgart, Germany klaus.zimmermann@sony.de

[2]. **Detection And Recognition Of Indian Traffic Signs**
Pritika Priya, Dhara Modha, Mansi Agrawal Department Of Information Technology, Bharati Vidyapeeth College Of Engineering For Women, Pune-43

[3]. **Deep Learning for Large-Scale Traffic-Sign Detection and Recognition**
Domen Tabernik and Danijel Skocaj Faculty of Computer and Information Science, University of Ljubljana Vecna pot 113, 1000 Ljubljana {domen. tabernik,danijel.skocaj}@fri.uni-lj.si

[4]. **Traffic Sign Classification Using Deep Inception Based Convolutional** Networks Mrinal Haloi IIT Guwahati1, mrinal.haloi11@gmail.com

[5]. **Indian Traffic Sign Detection and Classification Using Neural Networks**
Arun Nandewal - CSE Department, arunnandewal@gmail.com
Abhishek Tripathi - IT Department, abhishek.tripathi2421@gmail.com
Satyam Chandra - EEE Department, satyam9871@gmail.com
NITK Surathkal

9.2 Books, Videos and Study Materials:

[6]. **Saha, S.** (2018). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.

[7]. **Stanford University** (2018). Introduction to Convolutional Neural Networks.

[8]. **Visin, V.D., Francesco** (2016). English: Animation of a variation of the convolution operation. Blue maps are inputs, and cyan maps are outputs. [online] Wikimedia Commons.

[9]. **Bonner, A.** (2019). The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks.