

Traffic Sign Recognition

Project Report

San Jose State University



**Project Report on
Traffic Sign Recognition
Subject: Data Mining(CMPE 255)
M.S in Software Engineering (Semester – 1)
(User Defined Project)**

**Submitted by:
Team Dragonites
Khushil Modi
(SJSU ID: 015923115)
Sarjak Patel
(SJSU ID:015945046)
Nevil Shah
(SJSU ID: 015964975)
Vishnu Yeruva
(SJSU ID: 015917551)**

**Guided by:
Prof. Vijay Eranti**

Academic Year: 2021-22

Traffic Sign Recognition

Acknowledgement

We would like to express our deep sense of gratitude and sincere appreciation to our respected and esteemed Prof. Vijay Eranti, Assistant Professor, Computer Engineering Department, San Jose State University (SJSU) for his constant support and encouragement regarding the project work throughout this semester .

Last but not the least, we express our sincere thanks to all our classmates and friends, who either directly or indirectly, have helped us in some ways in our project.

Yours sincerely,
Khushil Modi
Sarjak Patel
Nevil Shah
Vishnu Yeruva

Traffic Sign Recognition

<u>Chapter</u>	<u>Title</u>
1	Introduction
	1.1 Project Introduction
	1.2 Project Abstract
	1.3 Objective
	1.4 Approach
2	CNN Model
	2.1 Model Architecture
	2.2 Model Training
	2.3 Graphs
	2.3.1 Loss-Accuracy Graph
3	System development
	3.1 Hardware Requirements
	3.2 Software Requirements
	3.3 Tools Used
4	Implementation
	4.1 Dataset
	4.2 Detection & Classification
	4.2.1 Data Preprocessing

Traffic Sign Recognition

	4.2.2 Exploratory Data Analysis
	4.2.3 Model Training
	4.2.3.1 Image Augmentation
	4.2.3.2 Model Summary
	4.2.4 Model Testing
	4.3 User Interface & Deployment
5	Future Work & Conclusion
	5.1 Future Work
	5.2 Conclusion

Traffic Sign Recognition

Chapter 1: Introduction

Chapter 1.1 Introduction

Nowadays, more emphasis and research is given on the ability of a car to drive itself on a road. It means it has to make accurate decisions while driving on a road or a highway to avoid collisions. Hence, it is very important to keep the conditions on either side of the road in mind. And the most important characteristic of a driverless car is to detect and recognize various traffic signs on the road for maintaining the safety and security of the people inside and outside the vehicle.

The Roadway system has different characteristics whose principal aim is to regulate and control the flow of traffic smoothly and to ensure that drivers are bound by the rules so as to provide a safe environment to the nearby traffic to avoid road collisions.

Chapter 1.2 Project Abstract

Traffic sign Recognition is a software application whose aim is to recognize traffic signs as efficiently as possible. This report tries to solve a major problem of Road Safety which is to classify traffic signs. It also detects and filters the problem with the help of Convolution Neural Networks (CNNs) for image classification and delivers the accurate solution.

Traffic sign recognition software ensures that different types of traffic signs on both sides of the road are visible to the drivers while driving the car. This software can recognize different types of traffic signs like “Stop”, “School Ahead”, “Speed Limit”, “No parking” and many more.

We have designed and developed an application for traffic sign recognition which will play a pivotal role in driverless cars. After providing an image as an input, it recognizes the traffic sign image and gives the description of it based on the model trained and tested using the signs dataset. The model is trained using CNN and Flask framework for Front-End development

Chapter 1.3 Objective

The main goal of this project is to design and develop a web-based application which classifies the type of traffic sign for the image given as input. We are intending a system which is able to recognize the traffic signs on either side of the road to give applicable support to the user or the machine in the driverless cars.

Traffic Sign Recognition

Chapter 1.4 Approach

The approach consists of assembling a model with the help of Convolution Neural Networks(CNNs) for image classification by extracting different traffic signs from an image.

Traffic Sign Recognition

Chapter 2 CNN Model

Chapter 2.1 Model Architecture

The model is built using Convolution Neural Networks for traffic-sign recognition from images due to its high accuracy. CNNs builds a hierarchical model whose working is like a network of neurons and gives an output as a layer where all the neurons are connected with each other and thereafter, the output is given on the image given as input.

We have added following layers to the sequential model:

- 1) 2D convolution layer(to convert the images in order to gain some features)
- 2) Max Pooling(to reduce and highlight thre parameters in an image)
- 3) Batch Normalization(to normalize the inputs)
- 4)Flatten Layer(to convert in single dimension)
- 5) Dense Layer(For changing the dimensions of the given vector)
- 6) Dropout Layer(to prevent overfitting)

The above mentioned layers were added while training the model to classify the traffic signs as output. It uses “Adam’s” optimizer for training this model . Also, as we have multiple classes for categorization, we have used “Categorical Crossentropy” as the loss function.

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation='relu', input_shape=(IMAGE_HEIGHT, IMAGE_WIDTH, channels)),
    keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Flatten(),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.5),

    keras.layers.Dense(128, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.5),

    keras.layers.Dense(43, activation='softmax')
```

Traffic Sign Recognition

Chapter 2.2 Model Training

After constructing the architecture and compilation of the model, the next phase is Model training. The model gave better results with Batch Size 64. Also, it was found that the accuracy remained steady after training the model with 20 Epochs. In addition to that, the accuracy scored is approximately 94%.

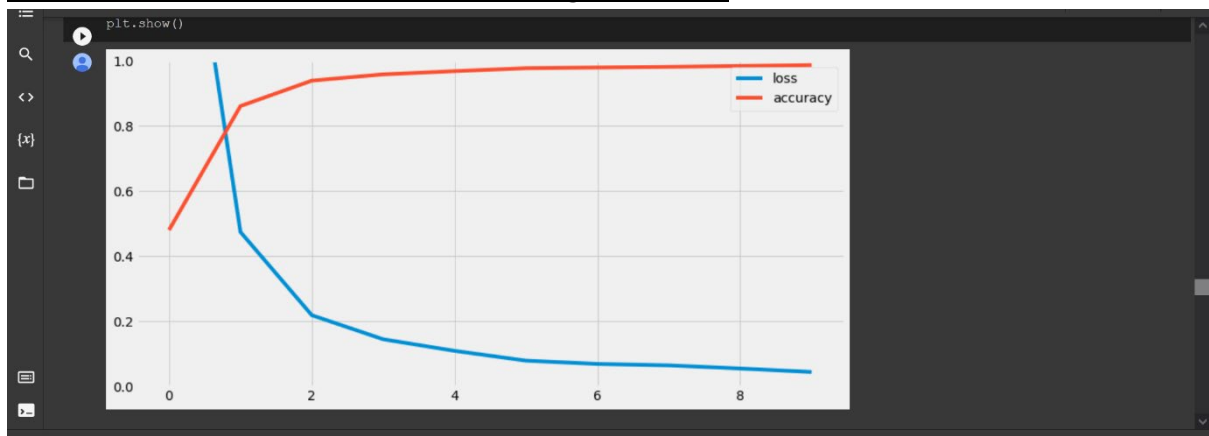
```
#image augmentation process starts here
aug = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.15,
    horizontal_flip=False,
    vertical_flip=False,
    fill_mode="nearest")

#the below given line will train the model at 20 Epochs with the training dataset.
history = model.fit(aug.flow(X_train, y_train, batch_size=32), epochs=epochs)
```

```
Epoch 1/20
981/981 [=====] - 31s 31ms/step - loss: 1.7309 - accuracy: 0.5318
Epoch 2/20
981/981 [=====] - 30s 31ms/step - loss: 0.4030 - accuracy: 0.8815
Epoch 3/20
981/981 [=====] - 30s 30ms/step - loss: 0.2113 - accuracy: 0.9395
Epoch 4/20
981/981 [=====] - 31s 32ms/step - loss: 0.1409 - accuracy: 0.9587
Epoch 5/20
981/981 [=====] - 30s 30ms/step - loss: 0.1081 - accuracy: 0.9681
Epoch 6/20
981/981 [=====] - 30s 30ms/step - loss: 0.0899 - accuracy: 0.9748
Epoch 7/20
981/981 [=====] - 30s 31ms/step - loss: 0.0709 - accuracy: 0.9792
Epoch 8/20
981/981 [=====] - 30s 30ms/step - loss: 0.0610 - accuracy: 0.9822
Epoch 9/20
981/981 [=====] - 30s 30ms/step - loss: 0.0599 - accuracy: 0.9826
Epoch 10/20
981/981 [=====] - 30s 30ms/step - loss: 0.0495 - accuracy: 0.9861
Epoch 11/20
981/981 [=====] - 29s 30ms/step - loss: 0.0423 - accuracy: 0.9873
Epoch 12/20
981/981 [=====] - 30s 30ms/step - loss: 0.0399 - accuracy: 0.9885
Epoch 13/20
981/981 [=====] - 29s 30ms/step - loss: 0.0377 - accuracy: 0.9892
```

Chapter 2.3 Graphs:

Chapter 2.3.1 Loss/Accuracy Graph



Traffic Sign Recognition

Chapter 3 System Development

Chapter 3.1 Hardware Requirements

- RAM: 8GB or more (16 GB Recommended)
- CPU: I7 processor recommended
- Storage: 256 Gigabytes SSD
- GPU: Nvidia NVS 310, GT, GTS, RTS (Recommended)

Chapter 3.2 Software Requirements

- Tensorflow
- OpenCV
- Access to Cloud platforms like Google Cloud platform, AWS Sagemaker.
- IDEs like Spyder, Jupyter Notebook, Google Colab
- Python v3.0.0 and above

Traffic Sign Recognition

Chapter 3.3 Tools Used

Sr. No.	Tool Name	Description
1	Anaconda Navigator	Used to provide environment space for python/Jupyter lab
2	Python 3 / Jupyter notebook	Used for Training the data and as language to work and for pre-required libraries to fetch and use
3	Flask	For front-end implementation
4	Mutiple Libraries	Numpy, pandas, tensorflow, sklearn,matplotlib,keras, openCV2
5	Pycharm	To provide IDE features and run the code with problem solving environment
6	Heroku	For software deployment

Traffic Sign Recognition

Chapter 4 Implementation

Chapter 4.1 Dataset

German Traffic Sign Recognition Benchmark(GTSRB) dataset

For traffic sign recognition, we looked for dataset which contains images for traffic signs. This dataset consists of multi-class, single image classification images. Furthermore, this dataset consists of more than 51k images and 43 classes.

Link for the dataset:- <https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>

Chapter 4.2 Detection & Classification

There are mainly 4 phases of this stage:

- 1)Data Preprocessing
- 2)Data Visualization
- 3)Model Training
- 4)Model Testing

Chapter 4.2.1 Data Preprocessing

- This is the first stage of the project. First and foremost, the necessary libraries are imported and the image dataset is loaded for further preprocessing.
- After loading the dataset, the images are resized to 30x30 image
- The images are thereafter converted into numpy array.

Traffic Sign Recognition

```
<>
{x}
image_data_list = []
image_labels_list = []

for i in range(NUMBER_CATEGORIES):
    path = data_dir + '/Train/' + str(i)
    images = os.listdir(path)

    for img in images:
        try:
            image = cv2.imread(path + '/' + img)
            image = Image.fromarray(image, 'RGB')
            image = image.resize((IMAGE_HEIGHT, IMAGE_WIDTH))#resizing the images to (30,30)
            image_data_list.append(np.array(image))
            image_labels_list.append(i)
        except:
            print("Error in the Image" + img)

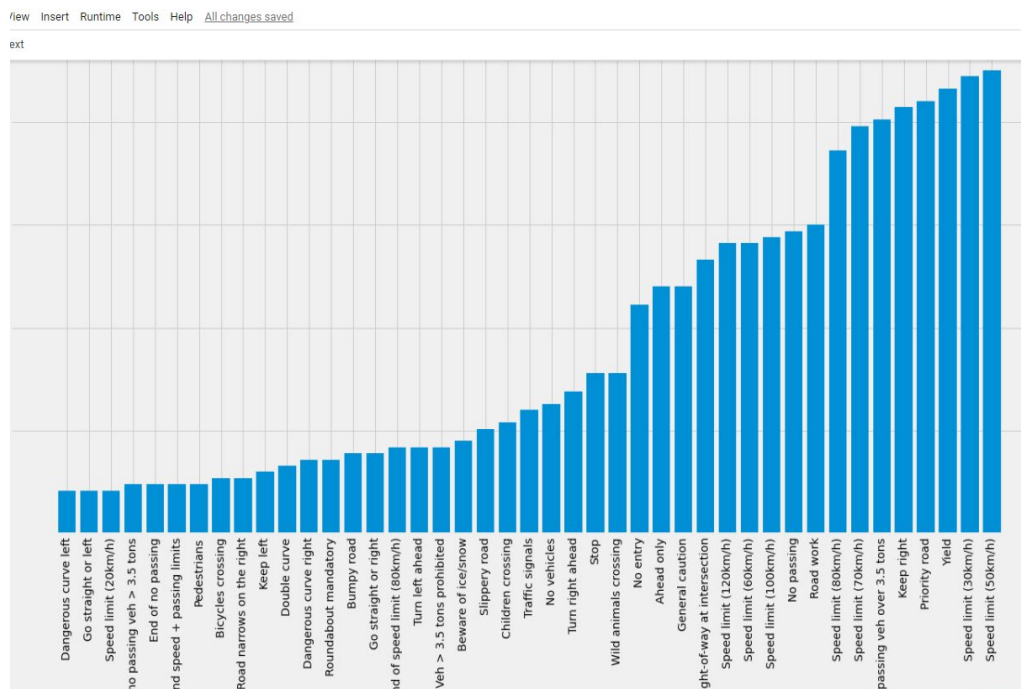
# Changing the list to numpy array
image_data_array = np.array(image_data_list)
image_labels_array = np.array(image_labels_list)

print(image_data_array.shape, image_labels_array.shape)
```

(39219, 30, 30, 3) (39219,)

Chapter 4.2.2 Exploratory Data Analysis

- **Exploratory data analysis (EDA)** is a process which conducts initial analysis and records its observations done with data sets.
- In this stage, we will order the 43 Classes of Traffic-Signs on the basis of number of images of each class given in the dataset.
- Speed limit(50kmph) has the highest number of images inside the given dataset whereas the images of dangerous curves has the least number.



Traffic Sign Recognition

Chapter 4.2.3 Model Training

- After EDA, the next stage is Model Training.
- The Dataset is, hereby, divided into training and testing datasets with number of records in the ratio 4:1.

```
Q From this point, we will divide the dataset(Ratio=4:1) into 2 parts :
<> 1) Training dataset(used for training the CNN model): 80%
(x) 2) Testing Dataset(used for testing the accuracy of the model):20%

[ ] X_train, X_test, y_train, y_test = train_test_split(image_data_array, image_labels_array, test_size=0.2, random_state=53, shuffle=True)
#now, we will bring the values in the range between 0 and 1.
X_train = X_train/255
X_test = X_test/255

print("X Train Shape", X_train.shape)
print("X Test Shape", X_test.shape)
print("y Train Shape", y_train.shape)
print("y Test Shape", y_test.shape)

X Train Shape (31375, 30, 30, 3)
X Test Shape (7844, 30, 30, 3)
y Train Shape (31375,)
y Test Shape (7844,)

[ ] y_train = keras.utils.to_categorical(y_train, NUMBER_CATEGORIES)
y_test = keras.utils.to_categorical(y_test, NUMBER_CATEGORIES)

print("y Train Shape", y_train.shape)
print("y Test Shape", y_test.shape)

y Train Shape (31375, 43)
y Test Shape (7844, 43)

So here, we have 31k records and 43 classes in the training data which will be given as input into the model.
And for the testing dataset, it contains about 8k records, which will be given as input to check the accuracy of the CNN trained model.
```

- Then a sequential model is defined and the different types of layers like 2D convolution, dense layer, MaxPool layer etc. are added and in this way, the architecture of the model is defined.
- Thereafter, the model is compiled where "Adam's optimizer" is used and the loss function defined here is categorical cross entropy.

```
#Here , we will add new layers inside a sequential model.
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=16, kernel_size=(3,3), activation='relu', input_shape=(IMAGE_HEIGHT, IMAGE_WIDTH, channels)),
    keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Flatten(),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.5),

    keras.layers.Dense(128, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.5),

    keras.layers.Dense(43, activation='softmax')
])
```

Traffic Sign Recognition

Chapter 4.2.3.1 Image Augmentation:

- Data Augmentation is a technique to create new training data from the existing data . This technique is used to increase the size of the training data to be given as input in the training model by performing minor modifications in the existing data.The Neural network will consider all the images as distinct images which will:
 - 1)Increase the overall performance of the model
 - 2) protect overfitting.

```
#image augmentation process starts here
aug = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.15,
    horizontal_flip=False,
    vertical_flip=False,
    fill_mode="nearest")

#the below given line will train the model at 20 Epochs with the training dataset.
history = model.fit(aug.flow(X_train, y_train, batch_size=32), epochs=epochs)
```

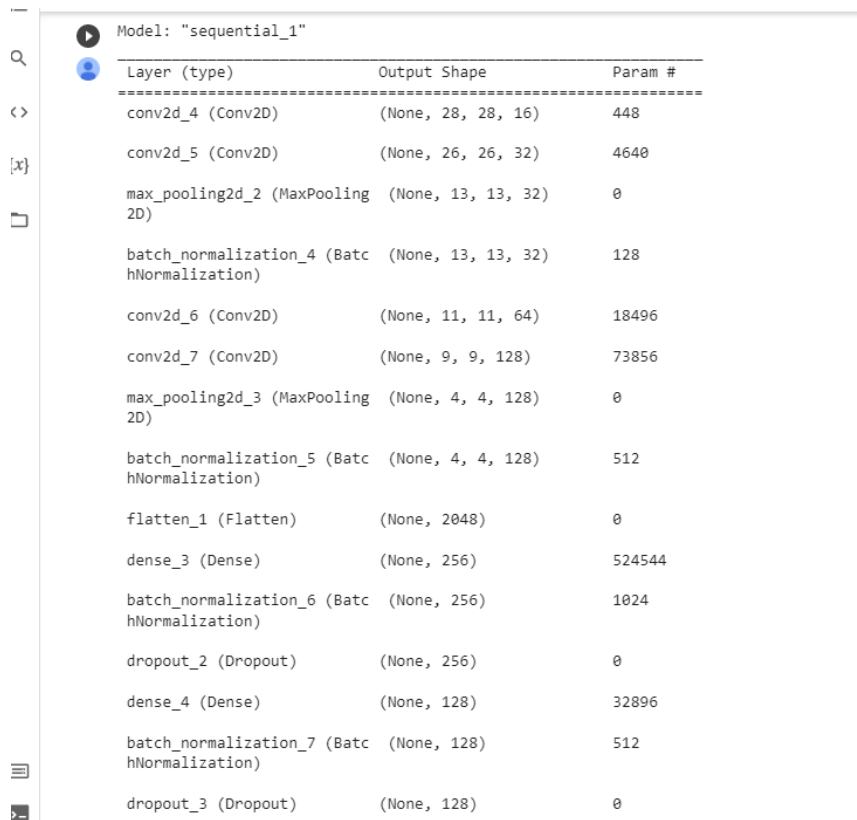
Epoch 1/20
981/981 [=====] - 31s 31ms/step - loss: 1.7309 - accuracy: 0.5318
Epoch 2/20
981/981 [=====] - 30s 31ms/step - loss: 0.4030 - accuracy: 0.8815
Epoch 3/20
981/981 [=====] - 30s 30ms/step - loss: 0.2113 - accuracy: 0.9395
Epoch 4/20
981/981 [=====] - 31s 32ms/step - loss: 0.1409 - accuracy: 0.9587
Epoch 5/20
981/981 [=====] - 30s 30ms/step - loss: 0.1081 - accuracy: 0.9681
Epoch 6/20
981/981 [=====] - 30s 30ms/step - loss: 0.0899 - accuracy: 0.9748
Epoch 7/20
981/981 [=====] - 30s 31ms/step - loss: 0.0709 - accuracy: 0.9792
Epoch 8/20
981/981 [=====] - 30s 30ms/step - loss: 0.0610 - accuracy: 0.9822
Epoch 9/20
981/981 [=====] - 30s 30ms/step - loss: 0.0599 - accuracy: 0.9826
Epoch 10/20
981/981 [=====] - 30s 30ms/step - loss: 0.0495 - accuracy: 0.9861
Epoch 11/20
981/981 [=====] - 29s 30ms/step - loss: 0.0423 - accuracy: 0.9873
Epoch 12/20
981/981 [=====] - 30s 30ms/step - loss: 0.0399 - accuracy: 0.9885
Epoch 13/20
981/981 [=====] - 29s 30ms/step - loss: 0.0377 - accuracy: 0.9892

- Afterwards, the model is fitted and trained over the traffic sign dataset using Epochs = 20 and accuracy obtained is about 99.36%.

Traffic Sign Recognition

Chapter 4.2.3.2 Model Summary:

- Here is the summary of the CNN built model.



The screenshot shows a Jupyter Notebook interface with a sidebar on the left containing icons for search, code, data, and file explorer. The main area displays the model summary for 'sequential_1'.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 28, 28, 16)	448
conv2d_5 (Conv2D)	(None, 26, 26, 32)	4640
max_pooling2d_2 (MaxPooling 2D)	(None, 13, 13, 32)	0
batch_normalization_4 (Batch Normalization)	(None, 13, 13, 32)	128
conv2d_6 (Conv2D)	(None, 11, 11, 64)	18496
conv2d_7 (Conv2D)	(None, 9, 9, 128)	73856
max_pooling2d_3 (MaxPooling 2D)	(None, 4, 4, 128)	0
batch_normalization_5 (Batch Normalization)	(None, 4, 4, 128)	512
flatten_1 (Flatten)	(None, 2048)	0
dense_3 (Dense)	(None, 256)	524544
batch_normalization_6 (Batch Normalization)	(None, 256)	1024
dropout_2 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32896
batch_normalization_7 (Batch Normalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0

Chapter 4.2.3 Model Testing

- After the model training using CNNs is completed, the accuracy of the model is tested over the testing data (remaining 20% of the dataset)

```
[ ] y_pred = model.predict(X_test)

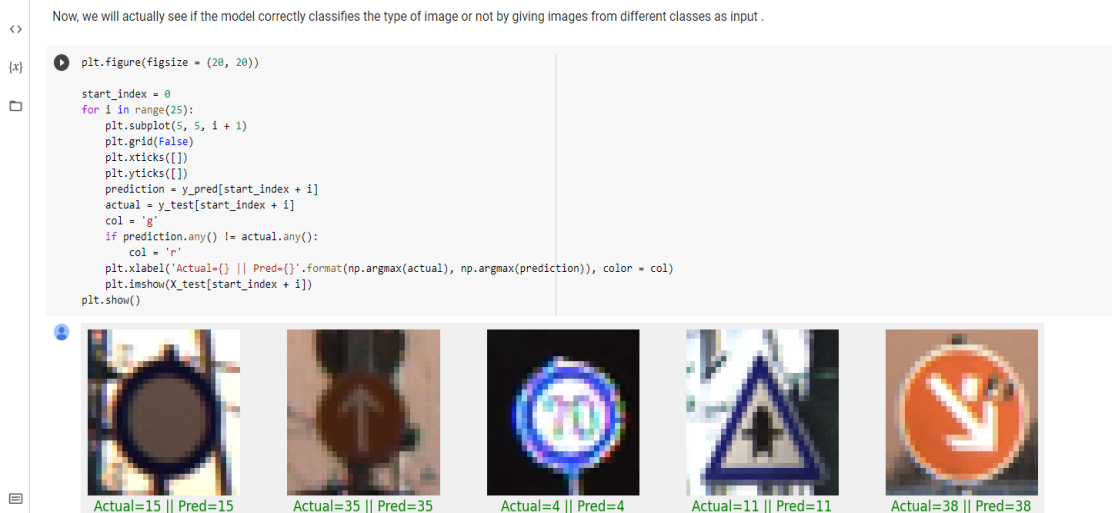
[ ] #Accuracy with the test data
    print('Test Data accuracy: ', accuracy_score(np.argmax(y_test,axis=1), np.argmax(y_pred, axis=1)))

Test Data accuracy: 0.9991075981642019
```

Note: After running the above cell, it is found that accuracy of the model over the test data is about 99.91%.

- When we observe the above figure closely, the accuracy of the model over the test dataset is also close to 99.90%.
- Now , we have given an image to classify which traffic sign it is.

Traffic Sign Recognition



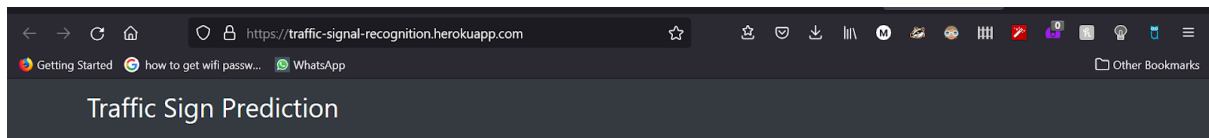
- The above image shows that the type of class predicted by the model is the same as the actual class.

Chapter 4.3 User Interface & Deployment

- For the user interface, we have used the Flask framework where users have to upload an image and the software will classify the type of traffic sign with the help of CNN model.
- With the aim of the software to operate entirely over cloud and accessible over the internet, we have deployed this software over Heroku which acts as a platform as a service(PaaS) .
- Here, the model can predict and classify the image given as input even if the image is blurry or have a darker shade than usual.

Image 1: Speed limit (distorted image)

Traffic Sign Recognition



Upload Traffic Signs

Upload...



Predicted Traffic Sign: Speed limit (30km/h)

Image 2: Road work(Blurry Image)



Upload Traffic Signs

Upload...



Predicted Traffic Sign: Road work

Traffic Sign Recognition

Image 3: Speed limit- 70 km/h (image of dark shade)

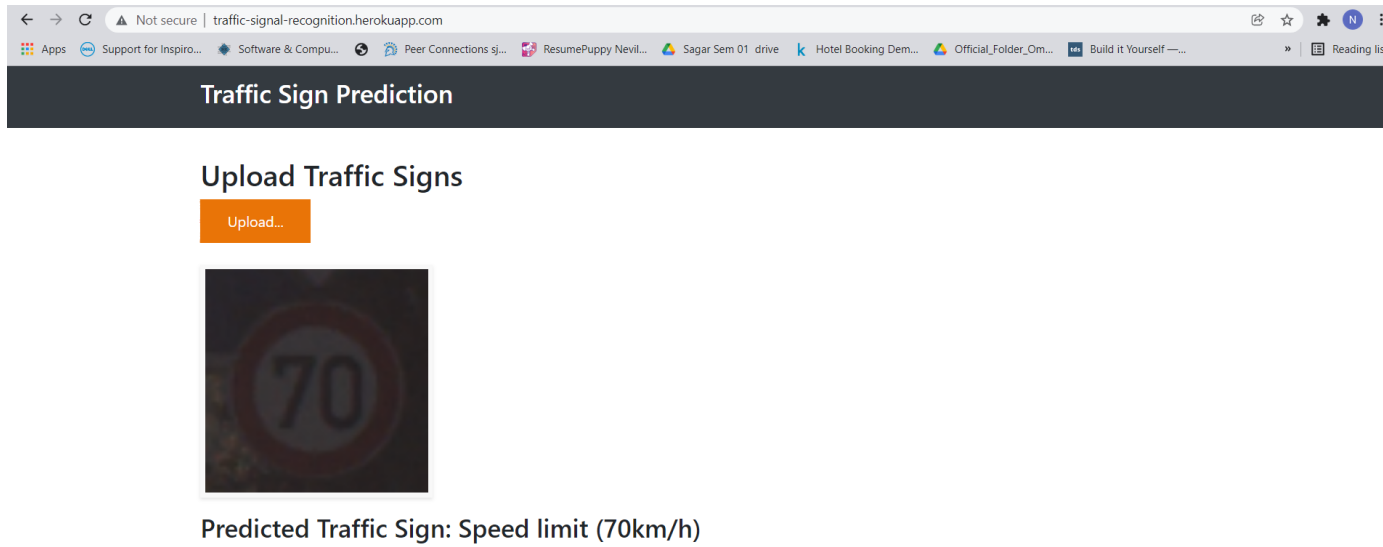
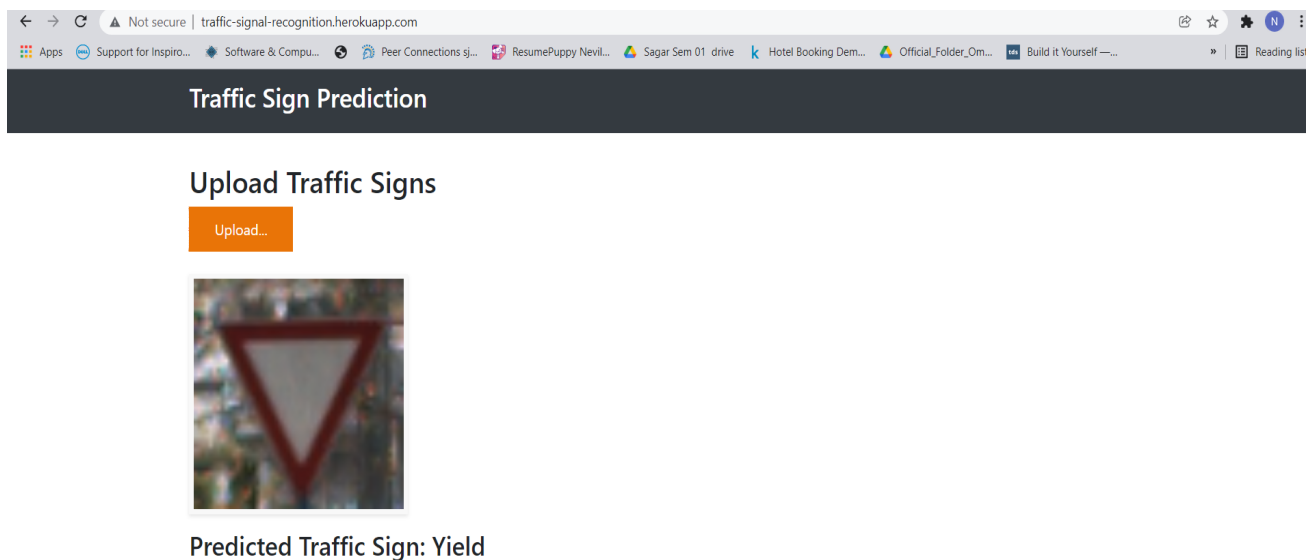


Image 4: Yield sign(very low clarity)



Traffic Sign Recognition

Chapter 5 Future Scope & Conclusion

Chapter 5.1 Future Scope

- A system that can be used in self-driving/driverless cars can be made.
- An alert and recommender driving system can also be made from this system for drivers.

Chapter 5.2 Conclusion

- A traffic sign recognition method on account of deep learning is proposed, where accuracy is about 99.8%
- By using traffic sign detection, image preprocessing, recognition and classification, this method can effectively detect and identify traffic signs.
- Even if the image given as input is distorted or with less clarity , the model predicts and classifies the correct type of traffic sign in most cases.

Traffic Sign Recognition

Chapter 6: References:

- <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- <https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?select=Train.csv>
- <https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>
- <https://devcenter.heroku.com/categories/deployment>
- <https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>
- <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>
- <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening>
- <https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/>
- <https://datascience.stackexchange.com/questions/24511/why-should-the-data-be-shuffled-for-machine-learning-tasks>
- <https://keras.io/>
- https://www.tensorflow.org/api_docs/python/tf/all_symbols
- <https://keras.io/api/models/model/>
- https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
- <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=What%20is%20the%20Adam%20optimization,iterative%20based%20in%20training%20data.>
- <https://docs.opencv.org/3.4/>
- <https://pillow.readthedocs.io/en/stable/reference/Image.html>