A

PROJECT REPORT ON TRAFFIC SIGN RECOGNITION USING CNN

SYMBIOSIS INSTITUTE OF GIO-INFORMATICS

(SYMBIOSIS INTERNATIONAL UNIVERSITY)

5th Floor, Atur Centre, Gokhale Cross Road Model Colony, Pune - 411016

GUIDED BY:	**SUBMITTED BY:**
Mr. Devawrat Bhave	**Sahil Abbas Naqvi**
(Professor)	**M.sc DS&SA**
	PRN: 20070243026

ABSTRACT

The project is undertaken at Symbiosis Institute of Geoinformatics, Pune has the potential of giving a visually better and more clear understanding of Traffic Signs Classification.

The project gives us knowledge about different traffic signs. The main objective of this project was to develop a CNN model which correctly classify the different traffic signs and correctly predict each one of them.

PREFACE

You must have heard about the self-driving cars in which the passenger can fully depend on the car for traveling. But to achieve that level of autonomous driving, it is necessary for vehicles to understand and follow all traffic rules.

In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly.

What is Traffic Signs Recognition?

There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to.

Traffic Signs Recognition – About the Project

In this Python project example, we will build a deep neural network model that can classify traffic signs present in the image into different categories. With this model, we can read and understand traffic signs which are a very important task for all autonomous vehicles.

ACKNOWLEDGMENT

I am thankful to receive assistance from one person in making this project a success.

I take this opportunity to express my deep regards and gratitude to **Mr. Devawrat Bhave (Professor)** for supporting me throughout the completion of the project.

Without your help, I surely will not be able to complete this project. Thank you so much sir for helping me during rough and tough times.

MATERIAL AND METHOD USED

1. **PROGRAMING LANGUAGE USED:**
   The main language used in this project is **PYTHON.** I have used different libraries for better visualization and interpretation of data. Here are some of them.

   a) **TensorFlow**: TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

   b) **Scikit-learn:** Scikit-learn (**Sklearn**) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering, and dimensionality reduction via a consistence interface in Python.

   c) **Keras:** Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks.

   d) **OpenCV:** OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc. In OpenCV, the CV is an abbreviation form of a computer vision, which is defined as a field of study that helps computers to understand the content of the digital images such as photographs and videos. The purpose of computer vision is to understand the content of the images. It extracts the description from the pictures, which may be an object, a text description, and three-dimension model, and so on. For example, cars can be facilitated with computer vision, which will be able to identify and different objects around the road, such as traffic lights, pedestrians, traffic signs, and so on, and acts accordingly.

   e) **NumPy:** NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more. NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries. The core of NumPy is well-optimized C code.

f) **Pandas:** Pandas is a software library written for the Python programming language for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series.

g) **OS:** The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing, and identifying the current directory**,** etc. You first need to import the os module to interact with the underlying operating system. So, import it using the import os statement before using its functions.

2. DATA COLLECTION:

GTSRB (German Traffic Sign Recognition Benchmark):

The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. We cordially invite researchers from relevant fields to participate: The competition is designed to allow for participation without special domain knowledge. Our benchmark has the following properties:

- Single-image, multi-class classification problem
- More than 40 classes
- More than 50,000 images in total
- Large, lifelike database

The dataset contains more than 50,000 images of different traffic signs. It is further classified into 43 different classes. The dataset is quite varying, some of the classes have many images while some classes have few images. The size of the dataset is around 300 MB. The dataset has a train folder which contains images inside each class and a test folder which we will use for testing our model.

3. LOADING DATA:

First, I create two empty list one for DATA and another for LABELS. Then I try to Retrieve the images and their labels. For this process I ran a for loop inside my working directory and iterate through every class and then retrieve all the images and their labels. Then I resize all my images in (30,30) and then place all images in NumPy array and after that I append all the images in my DATA list and append my labels list.

STEPS TO BUILD THE PROJECT

To get started with the project, first you need to download the dataset and extract all the files into a single folder such that you will have all the different folders train, test in it.

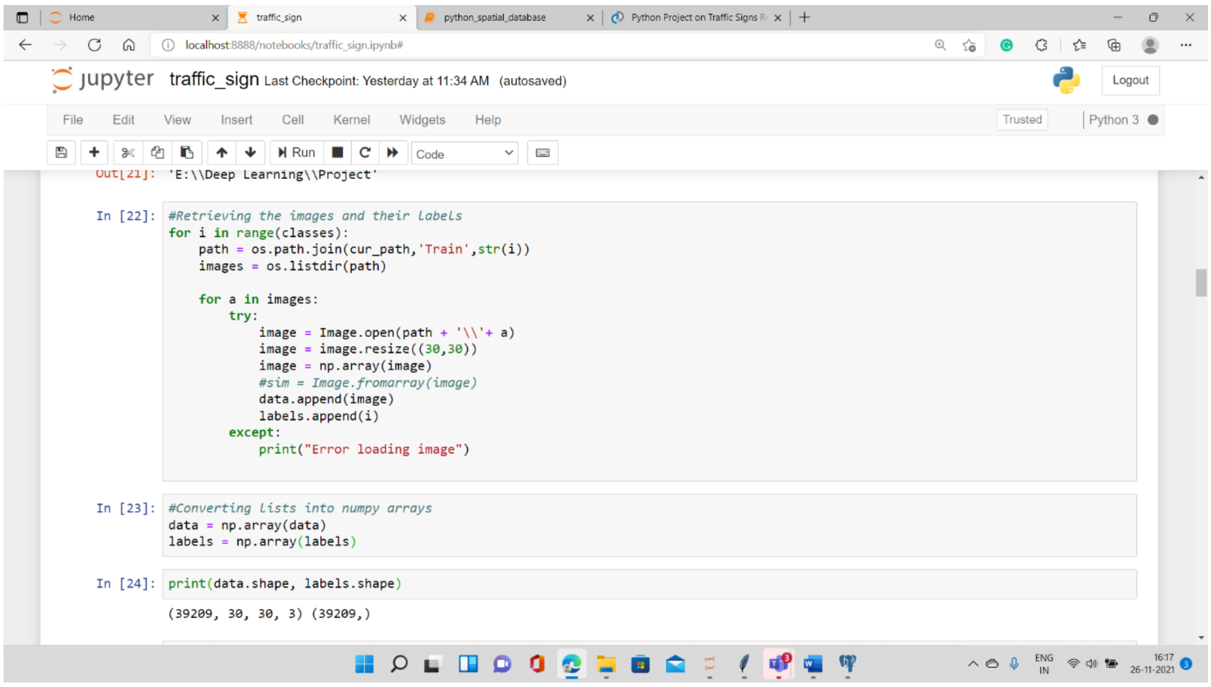
Our approach to building this traffic sign classification model is discussed in four steps:

- Explore the dataset
- Build a CNN model
- Train and validate the model
- Test the model with test dataset

Step 1: Explore the dataset

Our 'train' folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42. With the help of the OS module, we iterate over all the classes and append images and their respective labels in the data and labels list.

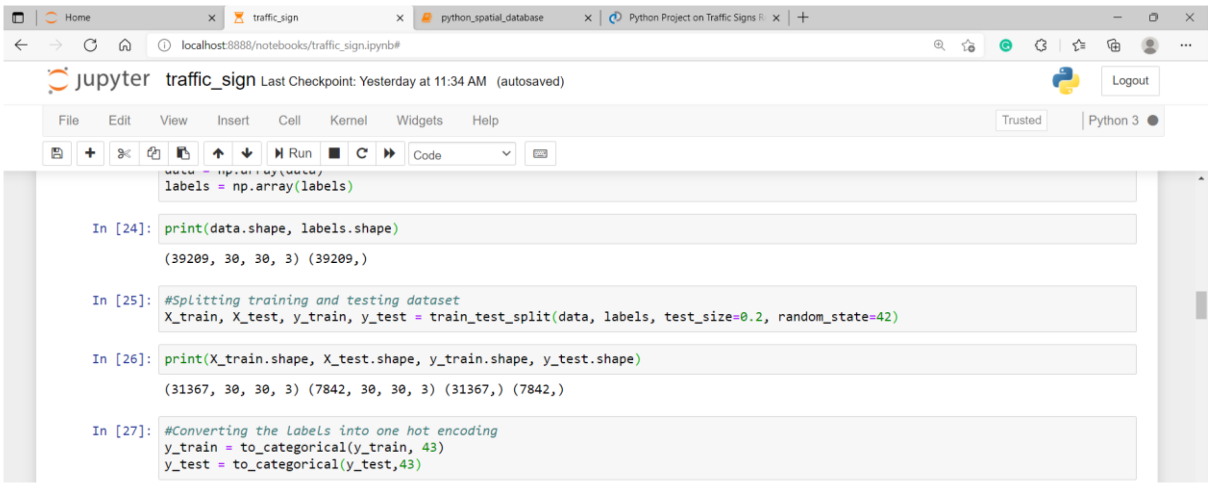
The PIL library is used to open image content into an array.


Home
traffic_sign
python_spatial_database
Python Project on Traffic Signs R
localhost:8888/notebooks/traffic_sign.ipynb#
jupyter
traffic_sign Last Checkpoint: Yesterday at 11:34 AM (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
ENG IN
16:17 26-11-2021

```
Out[21]: 'E:\\Deep Learning\\Project'

In [22]: #Retrieving the images and their labels
         for i in range(classes):
             path = os.path.join(cur_path,'Train',str(i))
             images = os.listdir(path)

             for a in images:
                 try:
                     image = Image.open(path + '\\'+ a)
                     image = image.resize((30,30))
                     image = np.array(image)
                     #sim = Image.fromarray(image)
                     data.append(image)
                     labels.append(i)
                 except:
                     print("Error loading image")

In [23]: #Converting lists into numpy arrays
         data = np.array(data)
         labels = np.array(labels)

In [24]: print(data.shape, labels.shape)

         (39209, 30, 30, 3) (39209,)
```

Finally, we have stored all the images and their labels into lists (data and labels).

We need to convert the list into NumPy arrays for feeding to the model. The shape of data is (39209, 30, 30, 3) which means that there are 39,209 images of size 30×30 pixels and the last 3 means the data contains coloured images (RGB value). With the sklearn package, we use the train_test_split() method to split training and testing data. From the keras.utils package, we use to_categorical method to convert the labels present in y_train and t_test into one-hot encoding.


Home
traffic_sign
python_spatial_database
Python Project on Traffic Signs R
localhost:8888/notebooks/traffic_sign.ipynb#
jupyter traffic_sign Last Checkpoint: Yesterday at 11:34 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
         labels = np.array(labels)

In [24]: print(data.shape, labels.shape)

         (39209, 30, 30, 3) (39209,)

In [25]: #Splitting training and testing dataset
         X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

In [26]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

         (31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)

In [27]: #Converting the labels into one hot encoding
         y_train = to_categorical(y_train, 43)
         y_test = to_categorical(y_test,43)
```

Step 2: Build a CNN model

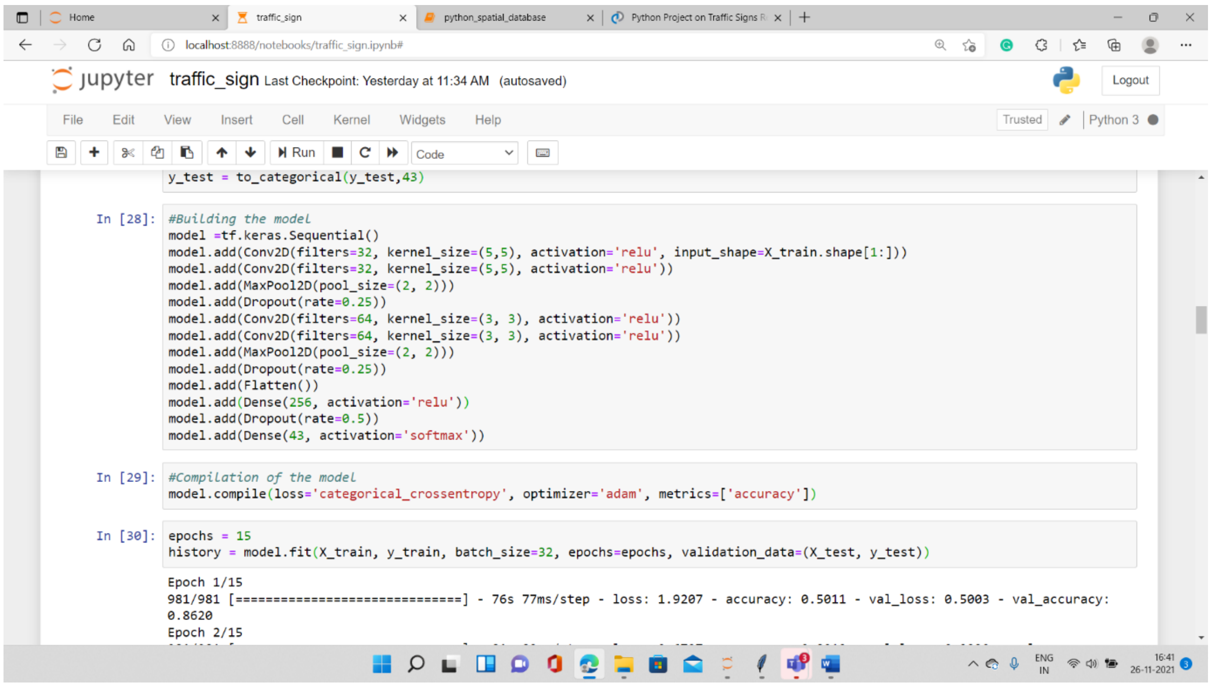
To classify the images into their respective categories, we will build a CNN model (**CONVOLUTIONAL NEURAL NETWORK)**. CNN is best for image classification purposes.

The architecture of our model is:

- 2 Conv2D layer (filter=32, kernel_size=(5,5), activation=”relu”)
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- MaxPool2D layer (pool_size=(2,2))
- 2 Conv2D layer (filter=64, kernel_size=(3,3), activation=”relu”)

- Dropout layer (rate=0.25)
- Flatten layer to squeeze the layers into 1 dimension
- Dense Fully connected layer (256 nodes, activation=”relu”)
- Dropout layer (rate=0.5)
- Dense layer (43 nodes, activation=”softmax”)

We compile the model with Adam optimizer which performs well, and loss is “categorical_crossentropy” because we have multiple classes to categorise

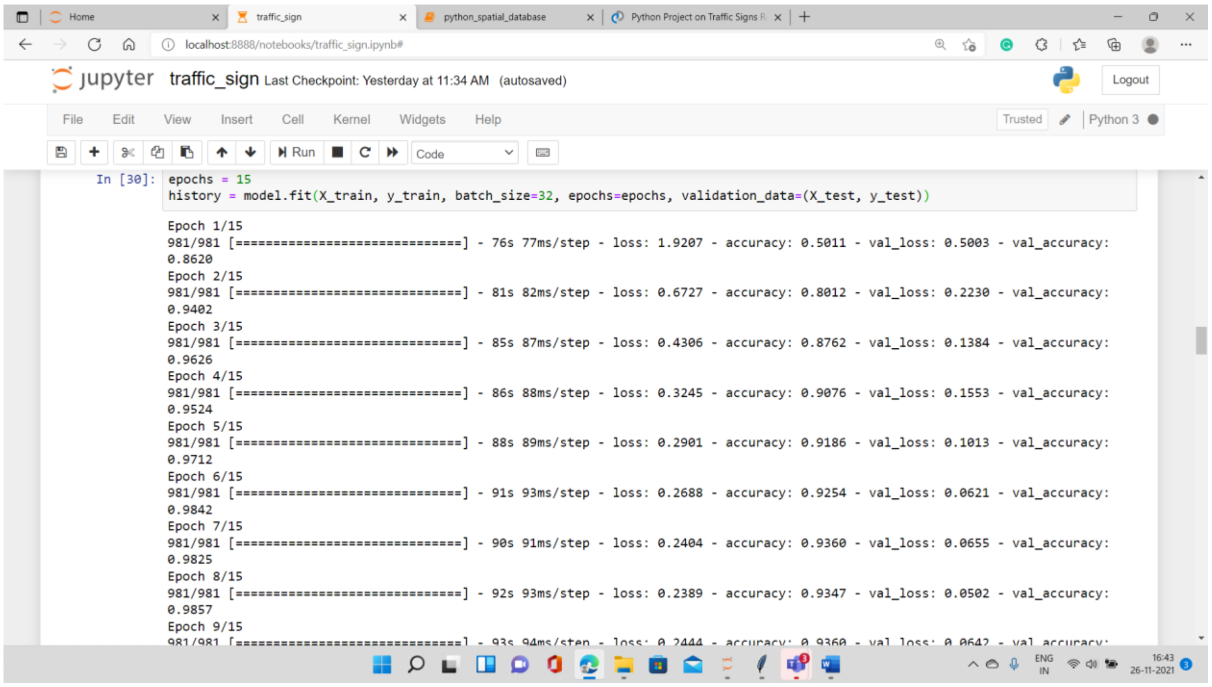


```
y_test = to_categorical(y_test,43)
```

```
In [28]: #Building the model
         model =tf.keras.Sequential()
         model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
         model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
         model.add(MaxPool2D(pool_size=(2, 2)))
         model.add(Dropout(rate=0.25))
         model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
         model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
         model.add(MaxPool2D(pool_size=(2, 2)))
         model.add(Dropout(rate=0.25))
         model.add(Flatten())
         model.add(Dense(256, activation='relu'))
         model.add(Dropout(rate=0.5))
         model.add(Dense(43, activation='softmax'))

In [29]: #Compilation of the model
         model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

In [30]: epochs = 15
         history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))

         Epoch 1/15
         981/981 [==============================] - 76s 77ms/step - loss: 1.9207 - accuracy: 0.5011 - val_loss: 0.5003 - val_accuracy:
         0.8620
         Epoch 2/15
```

Step 3: Train and validate the model

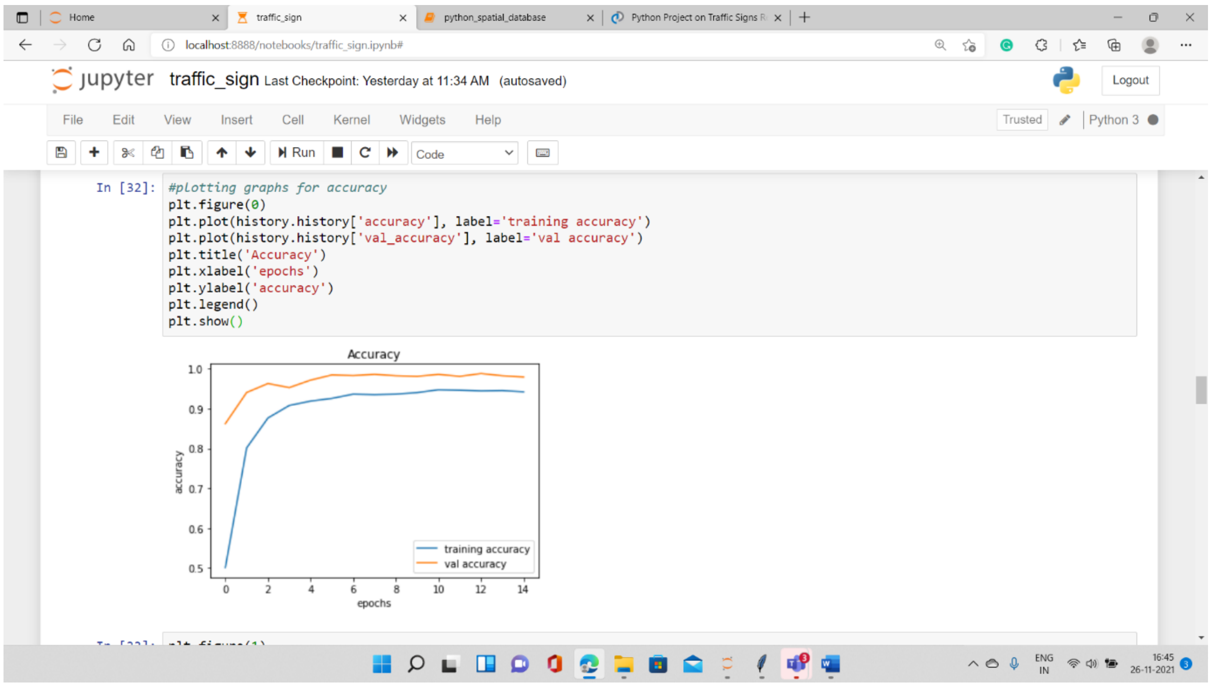
After building the model architecture, we then train the model using model.fit(). I tried with batch size 32 and 64. Our model performed better with 64 batch size. And after 15 epochs the accuracy was stable.



```
In [30]: epochs = 15
         history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))

         Epoch 1/15
         981/981 [==============================] - 76s 77ms/step - loss: 1.9207 - accuracy: 0.5011 - val_loss: 0.5003 - val_accuracy:
         0.8620
         Epoch 2/15
         981/981 [==============================] - 81s 82ms/step - loss: 0.6727 - accuracy: 0.8012 - val_loss: 0.2230 - val_accuracy:
         0.9402
         Epoch 3/15
         981/981 [==============================] - 85s 87ms/step - loss: 0.4306 - accuracy: 0.8762 - val_loss: 0.1384 - val_accuracy:
         0.9626
         Epoch 4/15
         981/981 [==============================] - 86s 88ms/step - loss: 0.3245 - accuracy: 0.9076 - val_loss: 0.1553 - val_accuracy:
         0.9524
         Epoch 5/15
         981/981 [==============================] - 88s 89ms/step - loss: 0.2901 - accuracy: 0.9186 - val_loss: 0.1013 - val_accuracy:
         0.9712
         Epoch 6/15
         981/981 [==============================] - 91s 93ms/step - loss: 0.2688 - accuracy: 0.9254 - val_loss: 0.0621 - val_accuracy:
         0.9842
         Epoch 7/15
         981/981 [==============================] - 90s 91ms/step - loss: 0.2404 - accuracy: 0.9360 - val_loss: 0.0655 - val_accuracy:
         0.9825
         Epoch 8/15
         981/981 [==============================] - 92s 93ms/step - loss: 0.2389 - accuracy: 0.9347 - val_loss: 0.0502 - val_accuracy:
         0.9857
         Epoch 9/15
         981/981 [==============================] - 93s 94ms/step - loss: 0.2444 - accuracy: 0.9360 - val_loss: 0.0642 - val_accuracy:
```

Our model got a 95% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy and the loss.

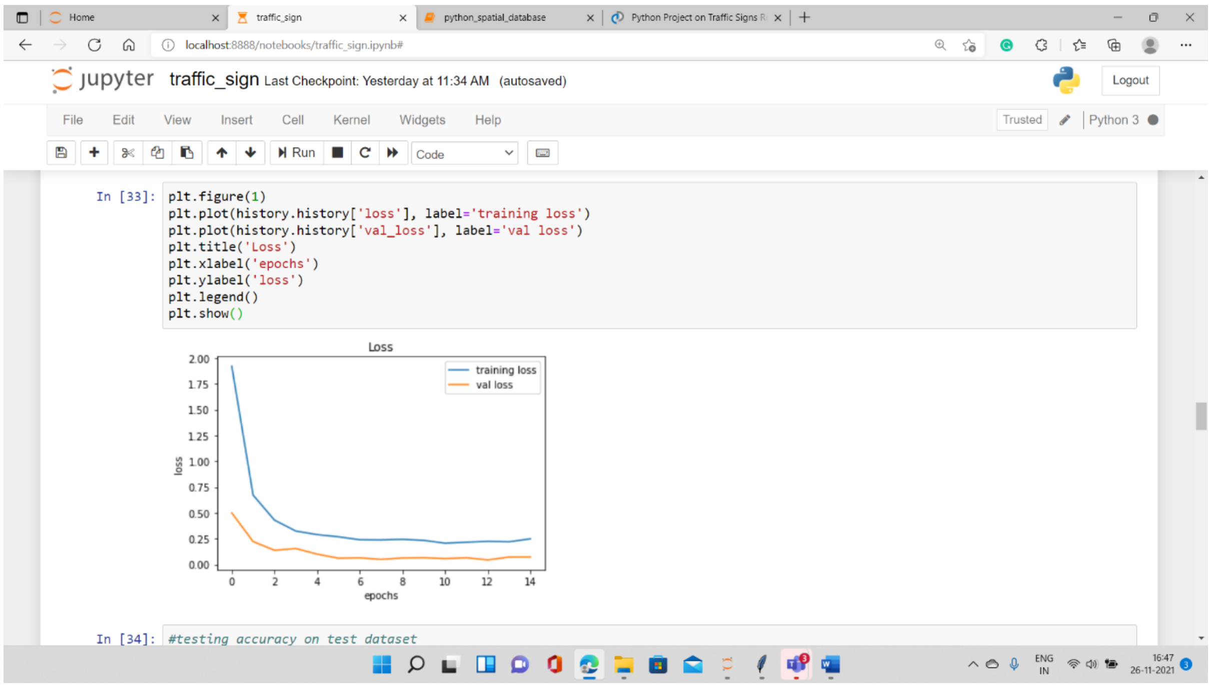
PLOT FOR TRAINING AND VALIDATION ACCURACY


Home
traffic_sign
python_spatial_database
Python Project on Traffic Signs R
localhost:8888/notebooks/traffic_sign.ipynb#
jupyter traffic_sign Last Checkpoint: Yesterday at 11:34 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Accuracy
accuracy
epochs
training accuracy
val accuracy
ENG IN
16:45
26-11-2021

```
In [32]: #plotting graphs for accuracy
         plt.figure(0)
         plt.plot(history.history['accuracy'], label='training accuracy')
         plt.plot(history.history['val_accuracy'], label='val accuracy')
         plt.title('Accuracy')
         plt.xlabel('epochs')
         plt.ylabel('accuracy')
         plt.legend()
         plt.show()
```

```
In [33]: plt.figure(1)
```

PLOT FOR TRAINING AND VALIDATION ACCURACY

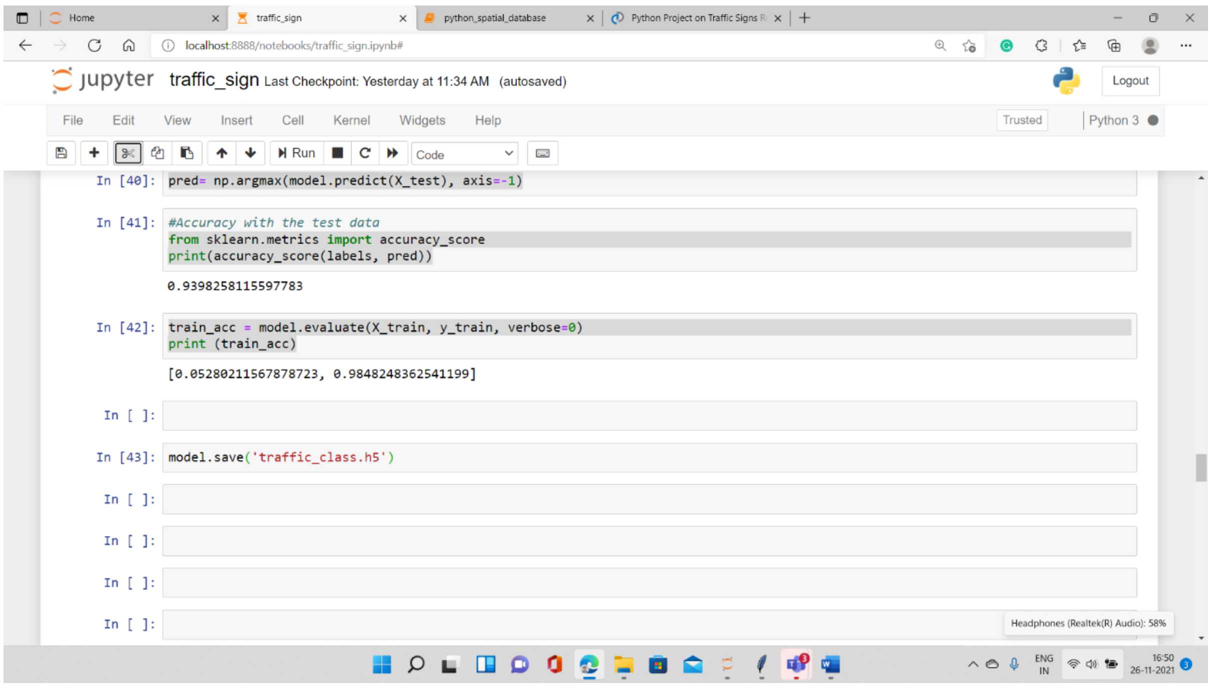

Home
traffic_sign
python_spatial_database
Python Project on Traffic Signs R
localhost:8888/notebooks/traffic_sign.ipynb#
jupyter traffic_sign Last Checkpoint: Yesterday at 11:34 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Loss
loss
epochs
training loss
val loss
ENG IN
16:47
26-11-2021

```
In [33]: plt.figure(1)
         plt.plot(history.history['loss'], label='training loss')
         plt.plot(history.history['val_loss'], label='val loss')
         plt.title('Loss')
         plt.xlabel('epochs')
         plt.ylabel('loss')
         plt.legend()
         plt.show()
```

```
In [34]: #testing accuracy on test dataset
```

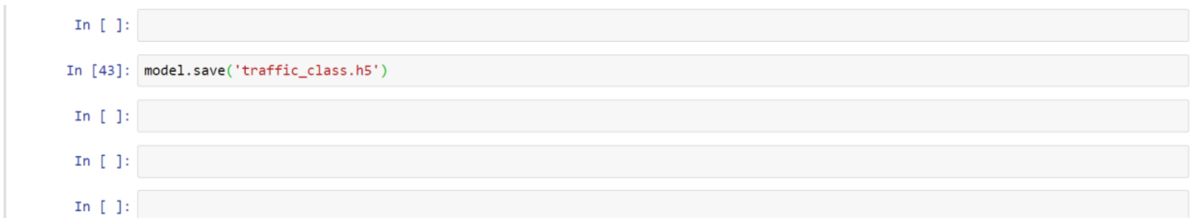
Step 4: Test our model with test dataset

Our dataset contains a test folder and in a test.csv file, we have the details related to the image path and their respective class labels. We extract the image path and labels using pandas. Then to predict the model, we must resize our images to 30×30 pixels and make a NumPy array containing all image data. From the sklearn.metrics, we imported the accuracy_score and observed how our model predicted the actual labels. We achieved a 95% accuracy in this model.



```
In [40]: pred= np.argmax(model.predict(X_test), axis=-1)

In [41]: #Accuracy with the test data
         from sklearn.metrics import accuracy_score
         print(accuracy_score(labels, pred))

         0.9398258115597783

In [42]: train_acc = model.evaluate(X_train, y_train, verbose=0)
         print (train_acc)

         [0.05280211567878723, 0.9848248362541199]

In [ ]:

In [43]: model.save('traffic_class.h5')

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```

In the end, we are going to save the model that we have trained using the Keras model.save() function.

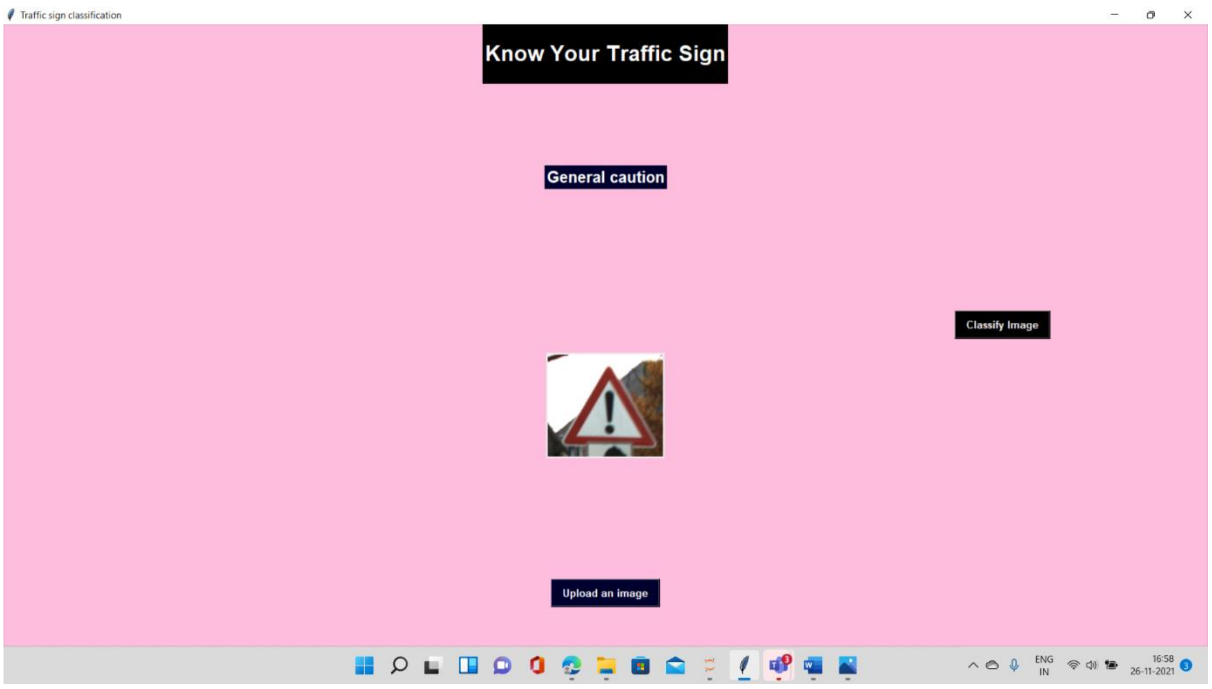


```
In [ ]:

In [43]: model.save('traffic_class.h5')

In [ ]:

In [ ]:

In [ ]:
```

TRAFFIC SIGN CLASSIFIER GUI

Now we are going to build a graphical user interface for our traffic signs classifier with Tkinter. Tkinter is a GUI toolkit in the standard python library. Make a new file in the project folder and copy the below code. Save it as gui.py and you can run the code by typing python gui.py in the command line.

In this file, we have first loaded the trained model 'traffic_classifier.h5' using Keras. And then we build the GUI for uploading the image and a button is used to classify which calls the classify() function. The classify() function is converting the image into the dimension of shape (1, 30, 30, 3). This is because to predict the traffic sign, we must provide the same dimension we have used when building the model. Then we predict the class, the model.predict_classes(image) returns us a number between (0-42) which represents the class it belongs to. We use the dictionary to get the information about the class. Here's the code for the gui.py file.



SUMMARY

In this Project, we have successfully classified the traffic signs classifier with 95% accuracy and visualized how our accuracy and loss changes with time, which is pretty good from a simple CNN model.