

Project on Traffic Signs Recognition with 95% Accuracy using CNN & Keras

Project – Traffic Signs Recognition

You must have heard about the self-driving cars in which the passenger can fully depend on the car for traveling. But to achieve level 5 autonomous, it is necessary for vehicles to understand and follow all traffic rules.

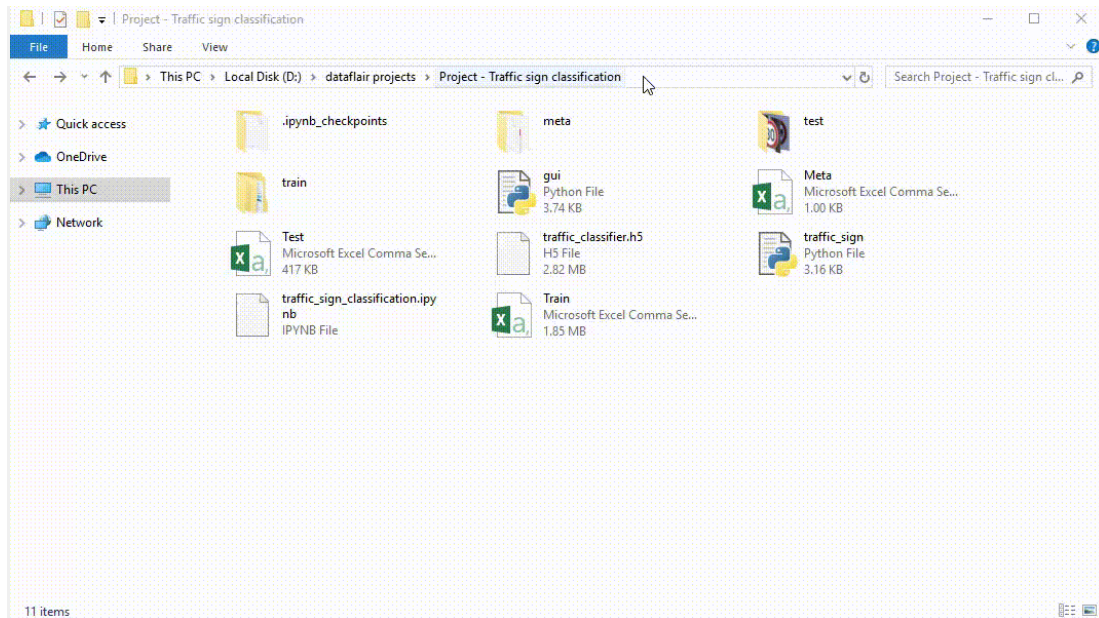
In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be 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 Python 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 are able to read and understand traffic signs which are a very important task for all autonomous vehicles.



The Dataset of Python Project

For this project, we are using the public dataset available at Kaggle:

[Traffic Signs Dataset](#)

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.

ELON MUSK : I'm gonna tell my kids that we used to drive cars.



Prerequisites

This project requires prior knowledge of Keras, Matplotlib, Scikit-learn, Pandas, PIL and image classification.

To install the necessary packages used for this Python data science project, enter the below command in your terminal:

```
pip install tensorflow keras sklearn matplotlib pandas pil
```

Steps to Build the Python Project

To get started with the project, unzip the file attached.

And extract the files into a folder such that you will have a train, test and a meta folder.

is PC > Local Disk (D:) > dataflair projects > Project - Traffic sign classification >

Name	Date modified	Type	Size
meta	04-Oct-19 12:52 PM	File folder	
test	04-Oct-19 12:52 PM	File folder	
train	04-Oct-19 12:55 PM	File folder	
Meta	25-Nov-18 6:13 PM	Microsoft Excel C...	2 KB
Test	25-Nov-18 6:13 PM	Microsoft Excel C...	418 KB
Train	25-Nov-18 6:13 PM	Microsoft Excel C...	1,896 KB

Create a Python script file and name it traffic_signs.py in the project folder.

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.

```
[9]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

data = []
labels = []
classes = 43
cur_path = os.getcwd()

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")

data = np.array(data)
labels = np.array(labels)
```

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 colored 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.

```

[10]: print(data.shape, labels.shape)
      X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

      print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

      y_train = to_categorical(y_train, 43)
      y_test = to_categorical(y_test, 43)

      (39209, 30, 30, 3) (39209,)
      (31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
  
```

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)
- 2 Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- 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.

```

[11]: model = 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'))

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

Steps 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.

```
[12]: epochs = 15
      history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_test, y_test))

Train on 31367 samples, validate on 7842 samples
Epoch 1/15
31367/31367 [=====] - 82s 3ms/step - loss: 2.3108 - accuracy: 0.4369 - val_loss: 0.6590 - val_accuracy: 0.8234
Epoch 2/15
31367/31367 [=====] - 82s 3ms/step - loss: 0.8266 - accuracy: 0.7606 - val_loss: 0.3468 - val_accuracy: 0.9100
Epoch 3/15
31367/31367 [=====] - 83s 3ms/step - loss: 0.5738 - accuracy: 0.8283 - val_loss: 0.1882 - val_accuracy: 0.9504
Epoch 4/15
31367/31367 [=====] - 85s 3ms/step - loss: 0.4282 - accuracy: 0.8720 - val_loss: 0.1373 - val_accuracy: 0.9661
Epoch 5/15
31367/31367 [=====] - 84s 3ms/step - loss: 0.3565 - accuracy: 0.8950 - val_loss: 0.1068 - val_accuracy: 0.9702
Epoch 6/15
31367/31367 [=====] - 81s 3ms/step - loss: 0.3081 - accuracy: 0.9074 - val_loss: 0.1527 - val_accuracy: 0.9575
Epoch 7/15
31367/31367 [=====] - 81s 3ms/step - loss: 0.2730 - accuracy: 0.9192 - val_loss: 0.0888 - val_accuracy: 0.9753
Epoch 8/15
31367/31367 [=====] - 81s 3ms/step - loss: 0.2429 - accuracy: 0.9271 - val_loss: 0.0934 - val_accuracy: 0.9737
Epoch 9/15
31367/31367 [=====] - 84s 3ms/step - loss: 0.2429 - accuracy: 0.9299 - val_loss: 0.0772 - val_accuracy: 0.9763
Epoch 10/15
31367/31367 [=====] - 81s 3ms/step - loss: 0.2176 - accuracy: 0.9364 - val_loss: 0.1133 - val_accuracy: 0.9663
Epoch 11/15
31367/31367 [=====] - 82s 3ms/step - loss: 0.2200 - accuracy: 0.9360 - val_loss: 0.0823 - val_accuracy: 0.9786
Epoch 12/15
31367/31367 [=====] - 80s 3ms/step - loss: 0.2046 - accuracy: 0.9406 - val_loss: 0.0806 - val_accuracy: 0.9787
Epoch 13/15
31367/31367 [=====] - 80s 3ms/step - loss: 0.1876 - accuracy: 0.9452 - val_loss: 0.0569 - val_accuracy: 0.9852
Epoch 14/15
31367/31367 [=====] - 81s 3ms/step - loss: 0.2007 - accuracy: 0.9430 - val_loss: 0.0629 - val_accuracy: 0.9811
Epoch 15/15
31367/31367 [=====] - 81s 3ms/step - loss: 0.1914 - accuracy: 0.9463 - val_loss: 0.0676 - val_accuracy: 0.9813
```

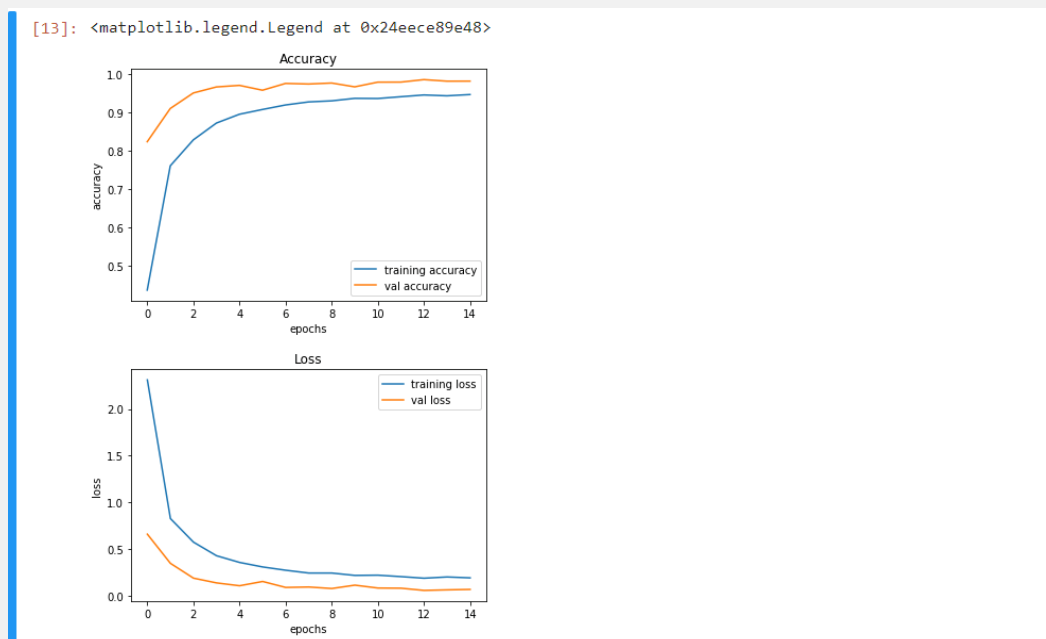
Our model got a 95% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy and the loss.

```
[13]: 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.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()
```

[13]: <matplotlib.legend.Legend at 0x24eece89e48>

Plotting Accuracy



Accuracy and Loss Graphs

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 have to 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.


```
[14]: from sklearn.metrics import accuracy_score
import pandas as pd
y_test = pd.read_csv('Test.csv')

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

data=[]

for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))

X_test=np.array(data)

pred = model.predict_classes(X_test)

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

```
[14]: 0.9532066508313539
```

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

```
model.save('traffic_classifier.h5')
```

Full Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
data = []
labels = []
classes = 43
cur_path = os.getcwd()
#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")
#Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)
#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)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
#Converting the labels into one hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
#Building the model
model = 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'))
#Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")
#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()
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()
#testing accuracy on test dataset
from sklearn.metrics import accuracy_score
y_test = pd.read_csv("Test.csv")
labels = y_test["ClassId"].values
imgs = y_test["Path"].values
data=[]
for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
X_test=np.array(data)
pred = model.predict_classes(X_test)
#Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
model.save('traffic_classifier.h5')
```

Traffic Signs 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 have to 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.

Code:

```
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy

#load the trained model to classify sign
from keras.models import load_model
model = load_model('traffic_classifier.h5')
#dictionary to label all traffic signs class.
classes = { 1:'Speed limit (20km/h)',
2:'Speed limit (30km/h)',
3:'Speed limit (50km/h)',
4:'Speed limit (60km/h)',
5:'Speed limit (70km/h)',
6:'Speed limit (80km/h)',
7:'End of speed limit (80km/h)',
8:'Speed limit (100km/h)',
9:'Speed limit (120km/h)',
10:'No passing',
11:'No passing veh over 3.5 tons',
12:'Right-of-way at intersection',
13:'Priority road',
14:'Yield',
15:'Stop',
16:'No vehicles',
17:'Veh > 3.5 tons prohibited',
18:'No entry',
19:'General caution',
20:'Dangerous curve left',
21:'Dangerous curve right',
22:'Double curve',
23:'Bumpy road',
24:'Slippery road',
25:'Road narrows on the right',
26:'Road work',
27:'Traffic signals',
28:'Pedestrians',
29:'Children crossing',
```

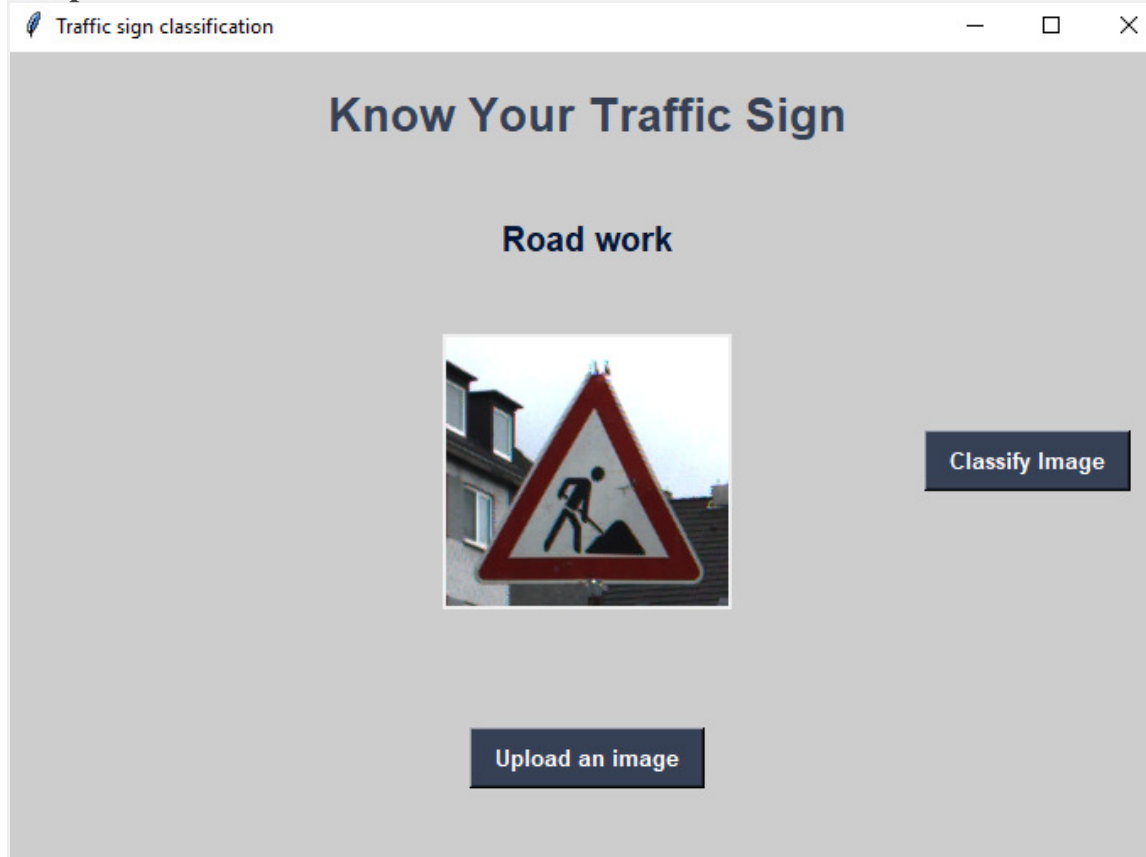
```

30:'Bicycles crossing',
31:'Beware of ice/snow',
32:'Wild animals crossing',
33:'End speed + passing limits',
34:'Turn right ahead',
35:'Turn left ahead',
36:'Ahead only',
37:'Go straight or right',
38:'Go straight or left',
39:'Keep right',
40:'Keep left',
41:'Roundabout mandatory',
42:'End of no passing',
43:'End no passing veh > 3.5 tons' }
#initialise GUI
top=tk.Tk()
top.geometry('800x600')
top.title("Traffic sign classification")
top.configure(background='#CDCDCD')
label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30,30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred = model.predict_classes([image])[0]
    sign = classes[pred+1]
    print(sign)
    label.configure(foreground='#011638', text=sign)
def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",command=lambda: classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)
def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im

```

```
label.configure(text="")
show_classify_button(file_path)
except:
pass
upload=Button(top,text="Upload an image",command=upload_image,padx=10,pady=5)
upload.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Know Your Traffic Sign",pady=20, font=('arial',20,'bold'))
heading.configure(background='#CDCDCD',foreground='#364156')
heading.pack()
top.mainloop()
```

Output:



Summary

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