OZLEM KILICKAYA

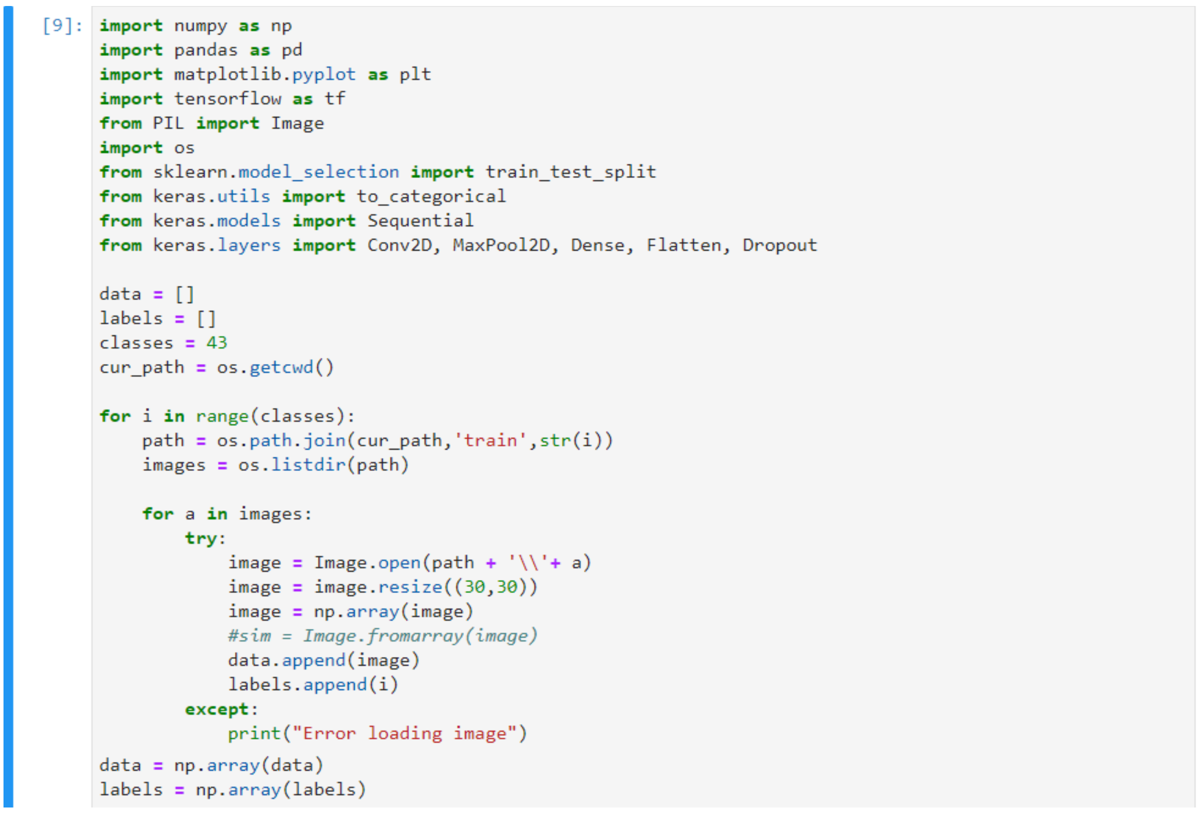
Traffic Sign Recognition Project

Step by step

Step-1: Explore the dataset

The ‘train’ folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42. Thanks to OS module, all the classes are iterated and images are appended and their respective labels in the data and labels list.

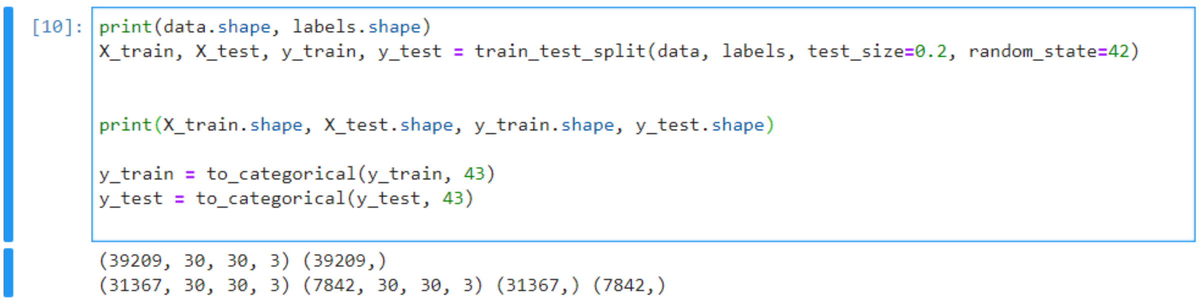
The PIL library is used to open image content into an array. All the images and their labels into lists (data and labels) are stored.



```
[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)
```

Lists must be turned into numpy arrays in order to feed 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). From sklearn package, use train_test_split() method to split training and testing data. From the keras.utils package, 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

CNN is best algorithm for image classification purposes.

The architecture of this model is given as follows:

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)

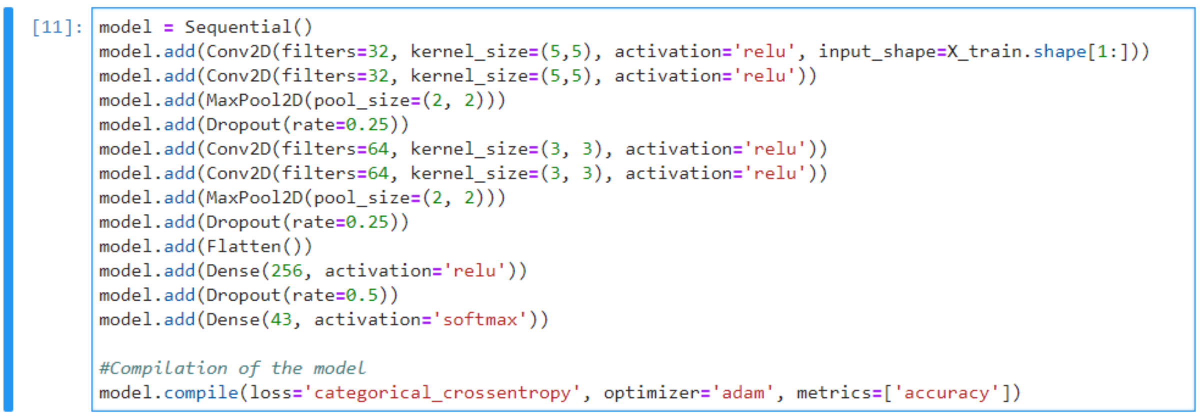
Flatten layer to squeeze the layers into 1 dimension

Dense Fully connected layer (256 nodes, activation=”relu”)

Dropout layer (rate=0.5)

Dense layer (43 nodes, activation=”softmax”)

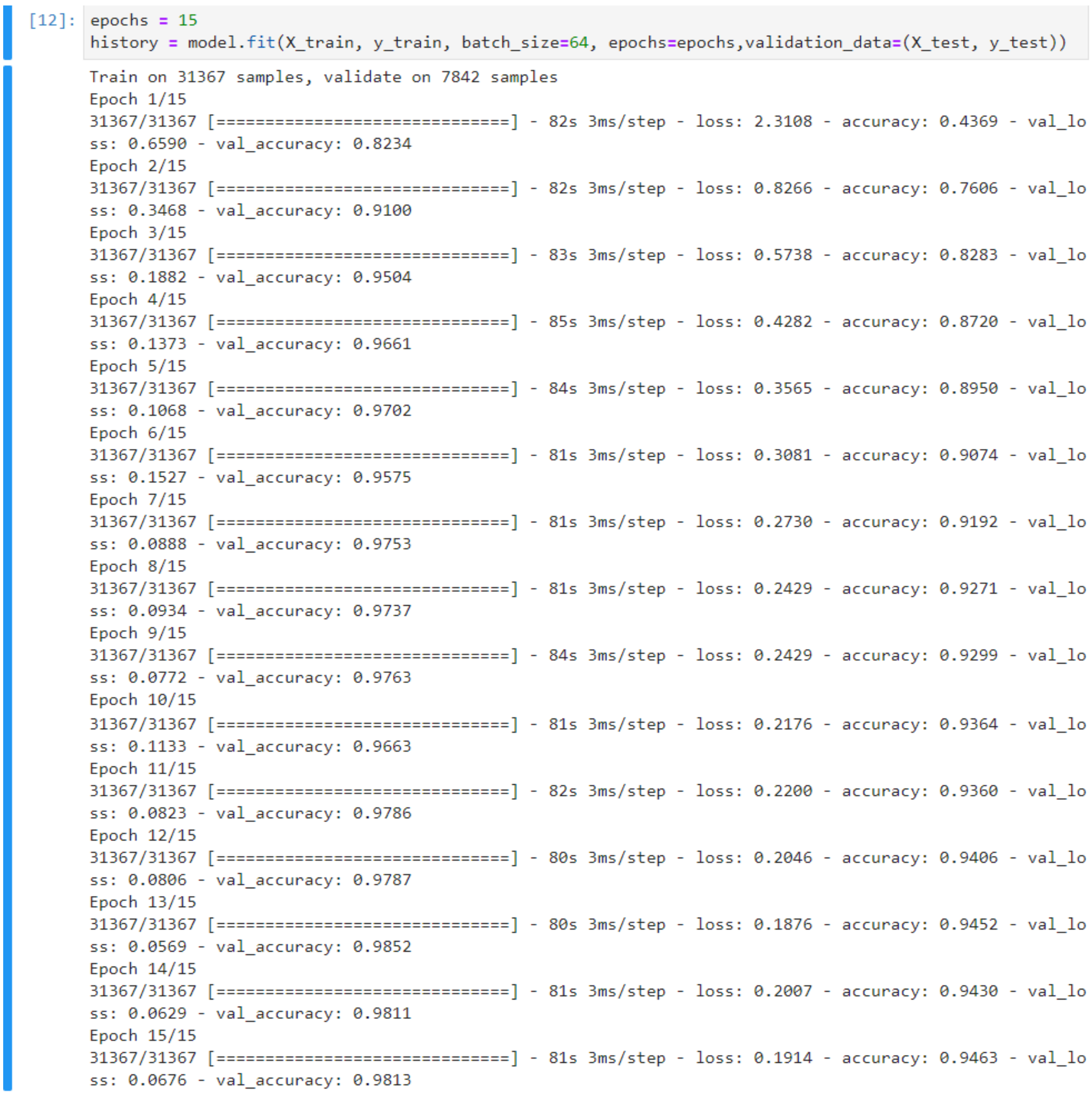
Adam optimizer is used to compile the model. Because this model performs well and this model contains multiple classes to categorize.



```
[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'])
```

Step 3: Train and validate the model

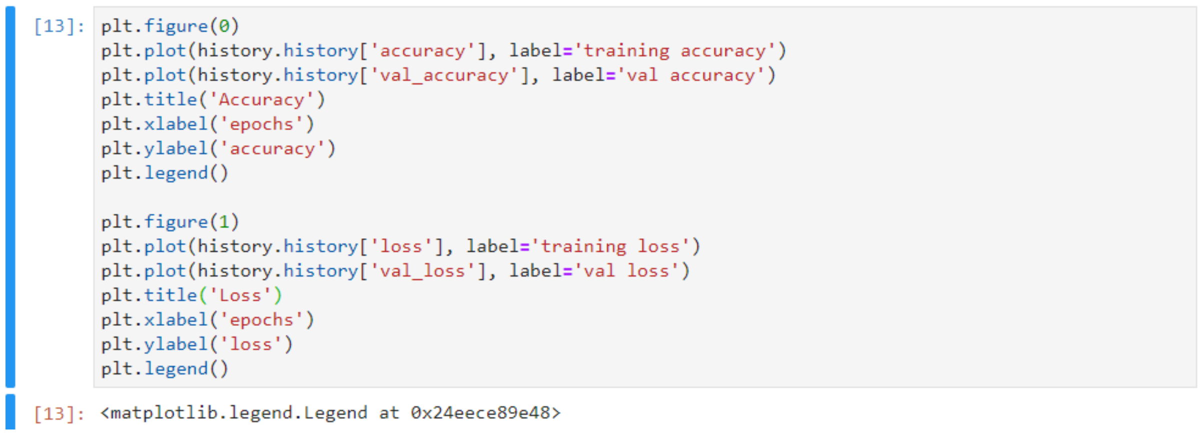
Model architecture was built. Then model must be trained by using model.fit(). Different batch sizes are tried. These are 32 and 64. This model performed better with 64 batch size. Accuracy became stable after 15 epochs.



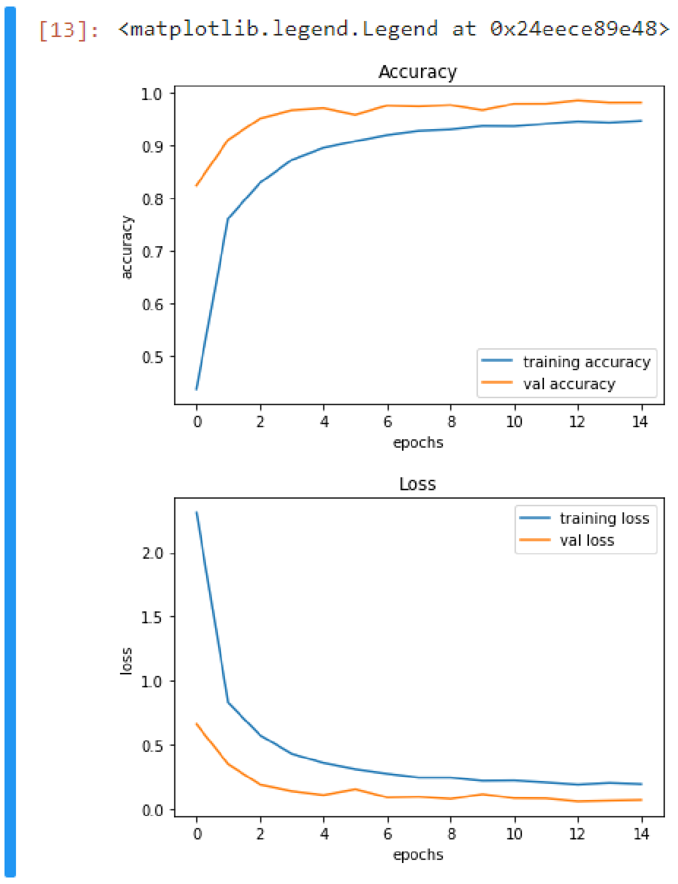
```
[12]: epochs = 15
      history = model.fit(X_train, y_train, batch_size=64, epochs=epochs,validation_data=(X_test, y_test))
```

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

This model achieved a 95% accuracy on the training dataset. I used matplotlib, the graph was plotted for accuracy and the loss.



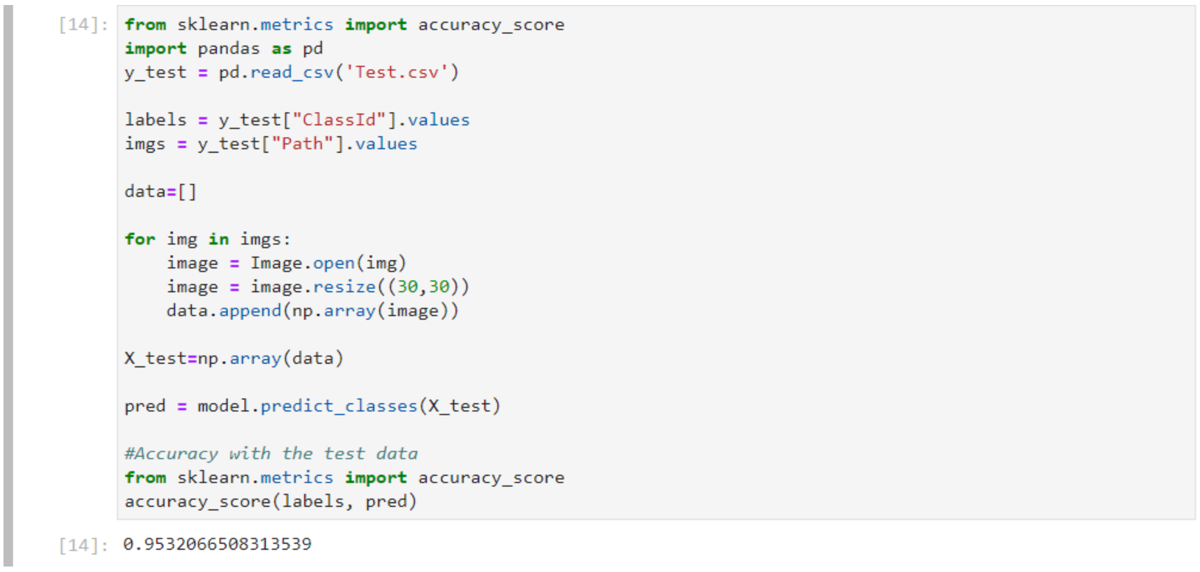
```
[13]: plt.figure(0)
      plt.plot(history.history['accuracy'], label='training accuracy')
      plt.plot(history.history['val_accuracy'], label='val accuracy')
      plt.title('Accuracy')
      plt.xlabel('epochs')
      plt.ylabel('accuracy')
      plt.legend()

      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>
```



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

Step 4: Test the model with test dataset

This dataset contains a test folder and in a test.csv file. It contains details related to the image path and their respective class labels. The image path and labels are extracted by using pandas. Then to predict the model, images are resized to 30×30 pixels and make a numpy array containing all image data. From the sklearn.metrics, the accuracy_score was imported. Then this model has observed. 95% accuracy has achieved with this model.

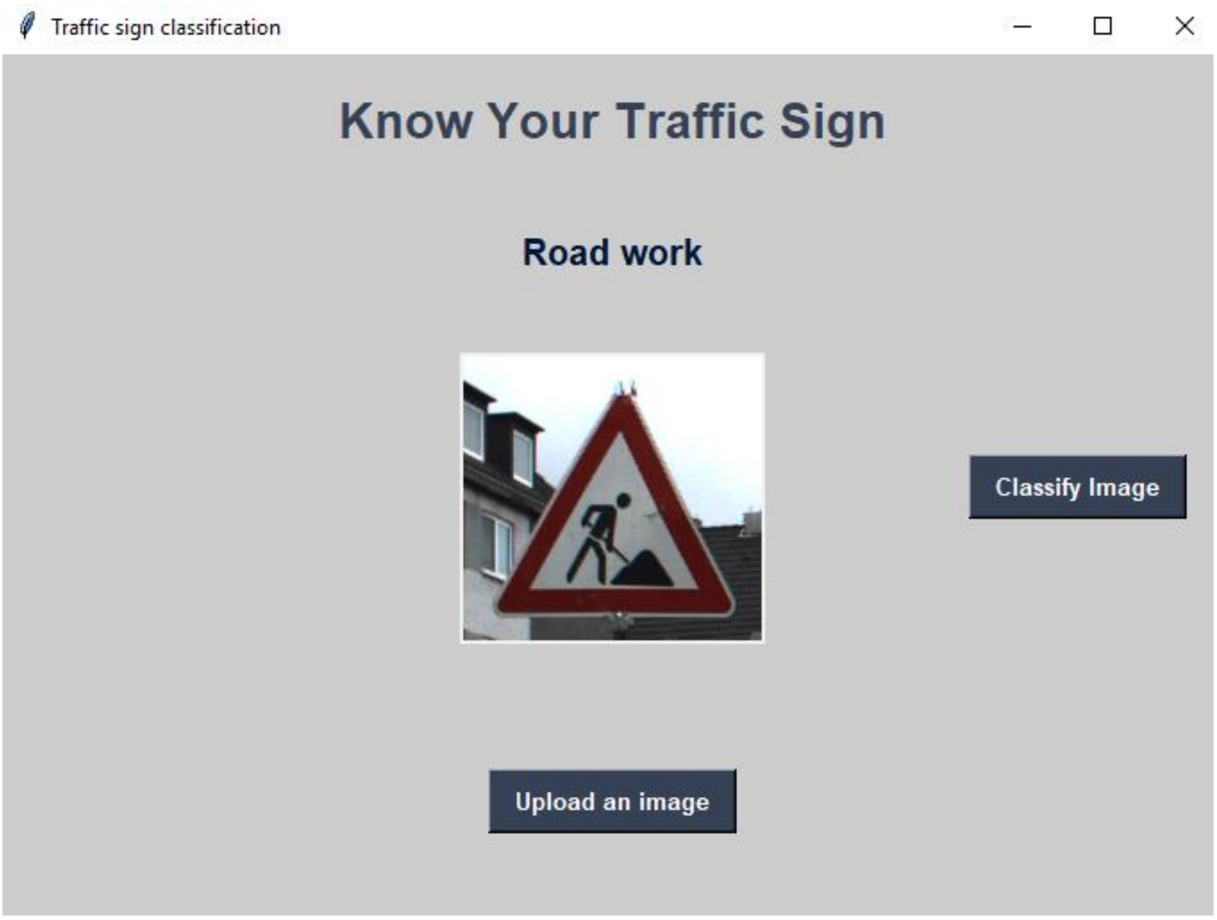


```
[14]: from sklearn.metrics import accuracy_score
      import pandas as pd
      y_test = pd.read_csv('Test.csv')

      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
```

Graphical User Interface for Traffic Signs Classifier

Traffic signs classifier graphical user interface was built with Tkinter. Tkinter is a GUI toolkit in the standard python library. Make a new file in the project folder. Then use the code given in gui.py within this project. Save it as gui.py and you can run the code by typing python gui.py in the command line.

In this file, the trained model 'traffic_classifier.h5' was loaded by using Keras. Then we build the GUI for uploading the image and a button is used to classify which calls the classify() function. The classify() function converts the image into the dimension of shape (1, 30, 30, 3). This is because to predict the traffic sign it must match with the same dimension that it was used when building the model. Then the class predicted by using the model.predict_classes(image) returns us a number between (0-42) which represents the class it belongs to. Dictionary has used to get the information about the class. Code has given in the gui.py file. Output for the graphical user interface is given as follows.


Traffic sign classification
Know Your Traffic Sign
Road work
Classify Image
Upload an image