Python for Deep. Learning Project Increment-2

Traffic Signs Recognition

Team 7 contribution:

Avinash Ganguri (ID – 6), *Model Evaluation with different hyperparameters*
Sri Sai Nikhil Kantipudi (ID – 10), *Model Analysis with callback functions and accuracy*
Dileep Reddy Peddakam (ID – 19), *Plotting and creating GUI in Tkinter*

Done so far:

Increment -1

1. *Preprocessed the Data,*

- Resizing the images
- Scaling the data
- One hot encoding using to_categorial

2. *Building our CNN Model using keras.*

3. *Plotted the loss and accuracy using history object.*

4. *Displayed a sample image from the test data set*

5. *Predicting the accuracy of our model with test labels data file*

Increment -2

6. *Modifying our CNN Model with different hyperparameters to improve the accuracy.*

7. *Implemented callback functions such as Early stopping in our model*

8. *Created a Desktop Application using Tkinter python library to upload the image and classify it.*

Remaining Parts to be Done for Final Increment:

- Make sure to classify the images on real world images too, other than from the test data set.

- Creating a Web application using flask and will classify the images. If possible, will try to host the site so that it can be accessible to the public.

Challenges faced:

- Running the environment in local machine is quite hard with package dependency issues.
- Even with Google Co-Lab's gpu and tpu, to train the large dataset is tedious and time taking as it requires the Colab pro, so we preferred running it on Jupyter Notebook instead.
- Running more epochs again and again when changing the model features is such a drag.

Screenshots:

Importing the Libraries,

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
from PIL import Image
import os
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
from keras.utils import to_categorical
from sklearn.metrics import accuracy_score
```

Loading the data and resizing it,

```
data=[]
labels=[]

height = 30
width = 30
channels = 3
classes = 43
n_inputs = height * width * channels

for i in range(classes) :
    path = "/Users/avi/Documents/UMKC/PythonDL/project/data/Train/{0}/".format(i)
    print(path)
    Class=os.listdir(path)
    for a in Class:
        try:
            image=cv2.imread(path+a)
            image_from_array = Image.fromarray(image, 'RGB')
            size_image = image_from_array.resize((height, width))
            data.append(np.array(size_image))
            labels.append(i)
        except AttributeError:
            print(" ")

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

#Randomizing the order of the input images
s=np.arange(data.shape[0])
np.random.seed(43)
np.random.shuffle(s)
data=data[s]
labels=labels[s]
```

```
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/0/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/1/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/2/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/3/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/4/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/5/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/6/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/7/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/8/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/9/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/10/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/11/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/12/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/13/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/14/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/15/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/16/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/17/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/18/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/19/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/20/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/21/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/22/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/23/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/24/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/25/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/26/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/27/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/28/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/29/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/30/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/31/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/32/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/33/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/34/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/35/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/36/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/37/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/38/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/39/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/40/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/41/
/Users/avi/Documents/UMKC/PythonDL/project/data/Train/42/
```

Preprocessing the data,

- ***Splitting and scaling the data***
- ***One hot encoding***

```
#Preprocessing the Dataset
#Spliting the images into train and validation sets
(X_train,X_val)=data[(int)(0.2*len(labels)):],data[:(int)(0.2*len(labels))]
X_train = X_train.astype('float32')/255
X_val = X_val.astype('float32')/255
(y_train,y_val)=labels[(int)(0.2*len(labels)):],labels[:(int)(0.2*len(labels))]

#Using one hot encoding for the train and validation labels with to_categorial
y_train = to_categorical(y_train, 43)
y_val = to_categorical(y_val, 43)
```

Building the CNN Model and Compiling it,

For Increment 1:

```
#Building our CNN Model

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='tanh', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='tanh'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='tanh'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='tanh'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='sigmoid'))

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

Training the model with 10 epochs,

```
#Training the model
epochs = 10
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_val, y_val))
```

```
Train on 31368 samples, validate on 7841 samples
Epoch 1/10
31368/31368 [==============================] - 72s 2ms/step - loss: 1.1712 - accuracy: 0.6780 - val_loss: 0.1673 - v
al_accuracy: 0.9589
Epoch 2/10
31368/31368 [==============================] - 72s 2ms/step - loss: 0.2338 - accuracy: 0.9364 - val_loss: 0.0720 - v
al_accuracy: 0.9832
Epoch 3/10
31368/31368 [==============================] - 71s 2ms/step - loss: 0.1427 - accuracy: 0.9626 - val_loss: 0.0439 - v
al_accuracy: 0.9898
Epoch 4/10
31368/31368 [==============================] - 72s 2ms/step - loss: 0.1076 - accuracy: 0.9714 - val_loss: 0.0383 - v
al_accuracy: 0.9908
Epoch 5/10
31368/31368 [==============================] - 77s 2ms/step - loss: 0.0881 - accuracy: 0.9755 - val_loss: 0.0309 - v
al_accuracy: 0.9922
Epoch 6/10
31368/31368 [==============================] - 89s 3ms/step - loss: 0.0831 - accuracy: 0.9750 - val_loss: 0.0279 - v
al_accuracy: 0.9939
Epoch 7/10
31368/31368 [==============================] - 80s 3ms/step - loss: 0.0722 - accuracy: 0.9794 - val_loss: 0.0273 - v
al_accuracy: 0.9925
Epoch 8/10
31368/31368 [==============================] - 89s 3ms/step - loss: 0.0709 - accuracy: 0.9790 - val_loss: 0.0281 - v
al_accuracy: 0.9929
Epoch 9/10
31368/31368 [==============================] - 82s 3ms/step - loss: 0.0652 - accuracy: 0.9811 - val_loss: 0.0252 - v
al_accuracy: 0.9943
Epoch 10/10
31368/31368 [==============================] - 83s 3ms/step - loss: 0.0681 - accuracy: 0.9793 - val_loss: 0.0261 - v
al_accuracy: 0.9943
```

For Increment 2:

Modified the CNN Model with different Hyperparameters,

Model 1

```
#Building our CNN Model

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(X_train.shape[1:]), padding='same', activation='relu'))
model.add(Dropout(0.5))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(64, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.3))
model.add(Dense(43, activation='softmax'))

#Compilation of the model
epochs = 10
lrate = 0.001
decay = lrate/epochs
sgd = Adam(lr=lrate)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

Training Model 1

```
#Training the model
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_val, y_val))
```

```
Train on 31368 samples, validate on 7841 samples
Epoch 1/10
31368/31368 [==============================] - 208s 7ms/step - loss: 1.8806 - accuracy: 0.4699 - val_loss: 0.6332 - v
al_accuracy: 0.8445
Epoch 2/10
31368/31368 [==============================] - 136s 4ms/step - loss: 0.9383 - accuracy: 0.7000 - val_loss: 0.4611 - v
al_accuracy: 0.8980
Epoch 3/10
31368/31368 [==============================] - 123s 4ms/step - loss: 0.7508 - accuracy: 0.7580 - val_loss: 0.2927 - v
al_accuracy: 0.9449
Epoch 4/10
31368/31368 [==============================] - 112s 4ms/step - loss: 0.6349 - accuracy: 0.7922 - val_loss: 0.2586 - v
al_accuracy: 0.9517
Epoch 5/10
31368/31368 [==============================] - 112s 4ms/step - loss: 0.5593 - accuracy: 0.8176 - val_loss: 0.2005 - v
al_accuracy: 0.9699
Epoch 6/10
31368/31368 [==============================] - 114s 4ms/step - loss: 0.5133 - accuracy: 0.8318 - val_loss: 0.1615 - v
al_accuracy: 0.9680
Epoch 7/10
31368/31368 [==============================] - 126s 4ms/step - loss: 0.4912 - accuracy: 0.8381 - val_loss: 0.1651 - v
al_accuracy: 0.9762
Epoch 8/10
31368/31368 [==============================] - 116s 4ms/step - loss: 0.4605 - accuracy: 0.8480 - val_loss: 0.1514 - v
al_accuracy: 0.9733
Epoch 9/10
31368/31368 [==============================] - 114s 4ms/step - loss: 0.4359 - accuracy: 0.8578 - val_loss: 0.1425 - v
al_accuracy: 0.9778
Epoch 10/10
31368/31368 [==============================] - 116s 4ms/step - loss: 0.4094 - accuracy: 0.8652 - val_loss: 0.1194 - v
al_accuracy: 0.9770
```

The Accuracy is 86.5% and the Validation Accuracy is 97.7% for Model 1.

Model 2

```
#Definition of the DNN model

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(Dropout(rate=0.5))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(rate=0.5))
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(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']
)
```

Training Model 2

```
#Training the model
epochs = 10
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_val, y_val))
```

```
WARNING:tensorflow:From //anaconda/envs/pythonproject/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py
:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 31368 samples, validate on 7841 samples
Epoch 1/10
31368/31368 [==============================] - 139s 4ms/step - loss: 1.5208 - accuracy: 0.5640 - val_loss: 0.4277 - v
al_accuracy: 0.9329
Epoch 2/10
31368/31368 [==============================] - 125s 4ms/step - loss: 0.3901 - accuracy: 0.8742 - val_loss: 0.1962 - v
al_accuracy: 0.9754
Epoch 3/10
31368/31368 [==============================] - 129s 4ms/step - loss: 0.2489 - accuracy: 0.9188 - val_loss: 0.1218 - v
al_accuracy: 0.9816
Epoch 4/10
31368/31368 [==============================] - 131s 4ms/step - loss: 0.2001 - accuracy: 0.9353 - val_loss: 0.0712 - v
al_accuracy: 0.9871
Epoch 5/10
31368/31368 [==============================] - 135s 4ms/step - loss: 0.1633 - accuracy: 0.9482 - val_loss: 0.0637 - v
al_accuracy: 0.9916
Epoch 6/10
31368/31368 [==============================] - 132s 4ms/step - loss: 0.1468 - accuracy: 0.9533 - val_loss: 0.0774 - v
al_accuracy: 0.9860
Epoch 7/10
31368/31368 [==============================] - 150s 5ms/step - loss: 0.1311 - accuracy: 0.9587 - val_loss: 0.0557 - v
al_accuracy: 0.9911
Epoch 8/10
31368/31368 [==============================] - 160s 5ms/step - loss: 0.1287 - accuracy: 0.9599 - val_loss: 0.0316 - v
al_accuracy: 0.9939
Epoch 9/10
31368/31368 [==============================] - 146s 5ms/step - loss: 0.1199 - accuracy: 0.9617 - val_loss: 0.0337 - v
al_accuracy: 0.9944
Epoch 10/10
31368/31368 [==============================] - 213s 7ms/step - loss: 0.1252 - accuracy: 0.9619 - val_loss: 0.0264 - v
al_accuracy: 0.9950
```

The Accuracy is 96.1% and the Validation Accuracy is 99.5% for Model 2.

Model 3

```
#Building our CNN Model

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='tanh', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='tanh'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='tanh'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='tanh'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='sigmoid'))

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

Training Model 3

```
#Training the model
epochs = 10
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_val, y_val))
```

```
Train on 31368 samples, validate on 7841 samples
Epoch 1/10
31368/31368 [==============================] - 72s 2ms/step - loss: 1.1712 - accuracy: 0.6780 - val_loss: 0.1673 - va
l_accuracy: 0.9589
Epoch 2/10
31368/31368 [==============================] - 72s 2ms/step - loss: 0.2338 - accuracy: 0.9364 - val_loss: 0.0720 - va
l_accuracy: 0.9832
Epoch 3/10
31368/31368 [==============================] - 71s 2ms/step - loss: 0.1427 - accuracy: 0.9626 - val_loss: 0.0439 - va
l_accuracy: 0.9898
Epoch 4/10
31368/31368 [==============================] - 72s 2ms/step - loss: 0.1076 - accuracy: 0.9714 - val_loss: 0.0383 - va
l_accuracy: 0.9908
Epoch 5/10
31368/31368 [==============================] - 77s 2ms/step - loss: 0.0881 - accuracy: 0.9755 - val_loss: 0.0309 - va
l_accuracy: 0.9922
Epoch 6/10
31368/31368 [==============================] - 89s 3ms/step - loss: 0.0831 - accuracy: 0.9750 - val_loss: 0.0279 - va
l_accuracy: 0.9939
Epoch 7/10
31368/31368 [==============================] - 80s 3ms/step - loss: 0.0722 - accuracy: 0.9794 - val_loss: 0.0273 - va
l_accuracy: 0.9925
Epoch 8/10
31368/31368 [==============================] - 89s 3ms/step - loss: 0.0709 - accuracy: 0.9790 - val_loss: 0.0281 - va
l_accuracy: 0.9929
Epoch 9/10
31368/31368 [==============================] - 82s 3ms/step - loss: 0.0652 - accuracy: 0.9811 - val_loss: 0.0252 - va
l_accuracy: 0.9943
Epoch 10/10
31368/31368 [==============================] - 83s 3ms/step - loss: 0.0681 - accuracy: 0.9793 - val_loss: 0.0261 - va
l_accuracy: 0.9943
```

The Accuracy is 97.9% and the Validation Accuracy is 99.4% for Model 2.

Model 4

```
#Definition of the DNN model

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
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(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']
)
```

Training Model 4

- *With Early Stopping as Callback Function*

```
#using 20 epochs for the training and earlystopping
epochs = 20
monitor=EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto',restore_best_weights=True)
history = model.fit(X_train, y_train, callbacks=[monitor], batch_size=32, epochs=epochs,
validation_data=(X_val, y_val))
```

When the model is trained with 20 Epochs, it is observed that Epoch 16 is the best Epoch and it stopped further and saved it the best accuracy to get.

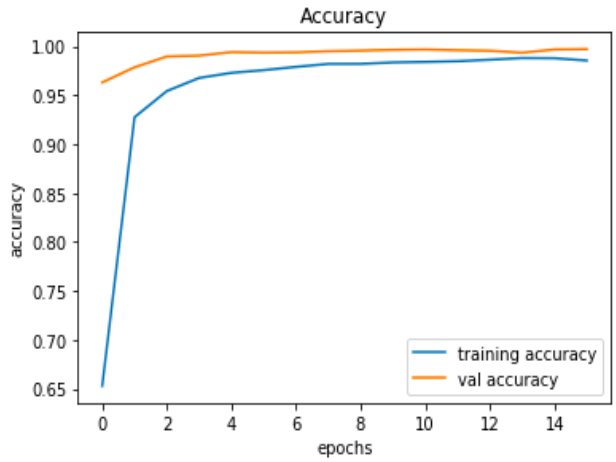
```
Epoch 12/20
31368/31368 [==============================] - 102s 3ms/step - loss: 0.0486 - accuracy: 0.9848 - val_loss: 0.0179 - v
al_accuracy: 0.9960
Epoch 13/20
31368/31368 [==============================] - 207s 7ms/step - loss: 0.0478 - accuracy: 0.9864 - val_loss: 0.0222 - v
al_accuracy: 0.9955
Epoch 14/20
31368/31368 [==============================] - 104s 3ms/step - loss: 0.0391 - accuracy: 0.9881 - val_loss: 0.0277 - v
al_accuracy: 0.9935
Epoch 15/20
31368/31368 [==============================] - 101s 3ms/step - loss: 0.0404 - accuracy: 0.9879 - val_loss: 0.0213 - v
al_accuracy: 0.9968
Epoch 16/20
31368/31368 [==============================] - 111s 4ms/step - loss: 0.0485 - accuracy: 0.9856 - val_loss: 0.0180 - v
al_accuracy: 0.9972
Restoring model weights from the end of the best epoch.
Epoch 00016: early stopping
```

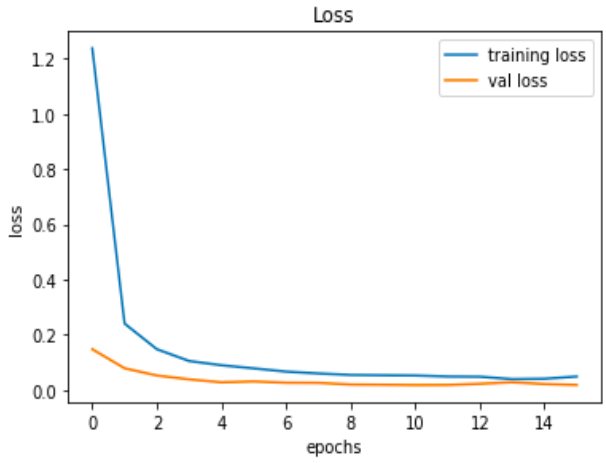
The Accuracy is 98.5% and the Validation Accuracy is 99.7% for Model 4.

After trying out different Hyperparameters and changing several combinations of layers and activation functions with different optimizers, we preferred **Model 4** as our best fit.

Plotting the Loss and Accuracy using the History Object,

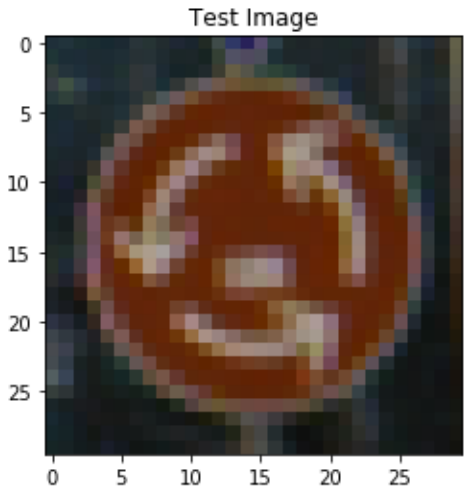
```
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

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()
```





Displaying a Test Image,

```
plt.imshow(X_test[1123,:,:],cmap='gray')
plt.title('Test Image')
plt.show()
```



Predicting the model with Test data file and displaying the accuracy of the model,

```
#Predicting with the test data
y_test=pd.read_csv("/Users/avi/Documents/UMKC/PythonDL/project/data/Test.csv")
labels=y_test['Path'].to_numpy()
y_test=y_test['ClassId'].values

data=[]

for f in labels:
    image=cv2.imread('/Users/avi/Documents/UMKC/PythonDL/project/data/Test/'+f.replace('Test/', ''))
    image_from_array = Image.fromarray(image, 'RGB')
    size_image = image_from_array.resize((height, width))
    data.append(np.array(size_image))

X_test=np.array(data)
X_test = X_test.astype('float32')/255
pred = model.predict_classes(X_test)
```

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

```
0.9742676167854315
```

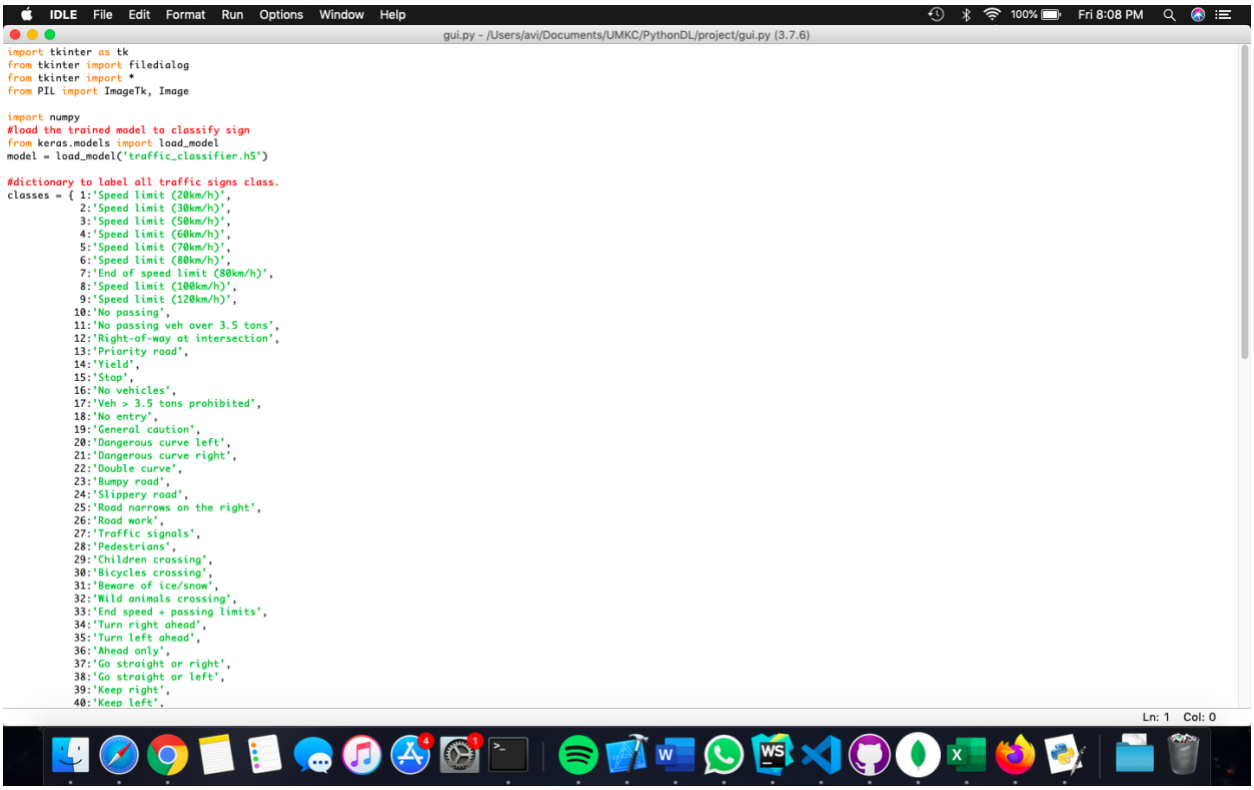
The Accuracy with the test data is 97.4%

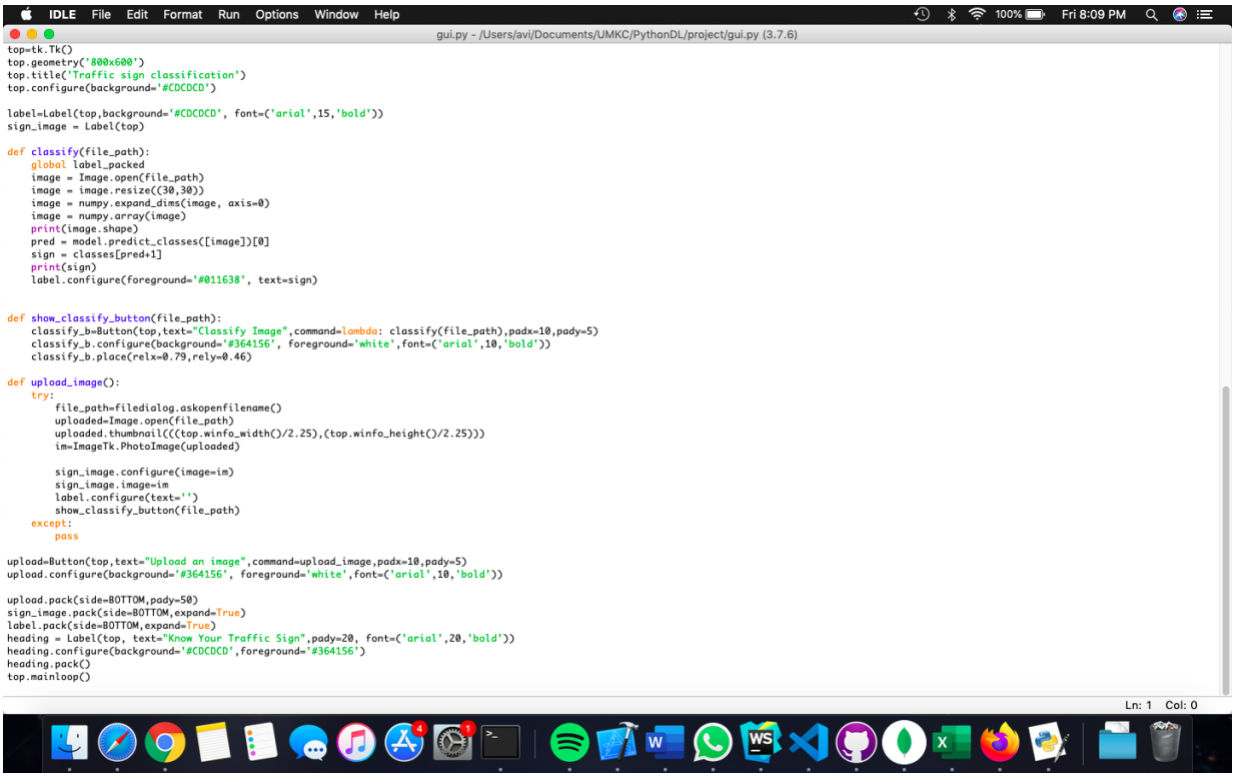
Saving the Model

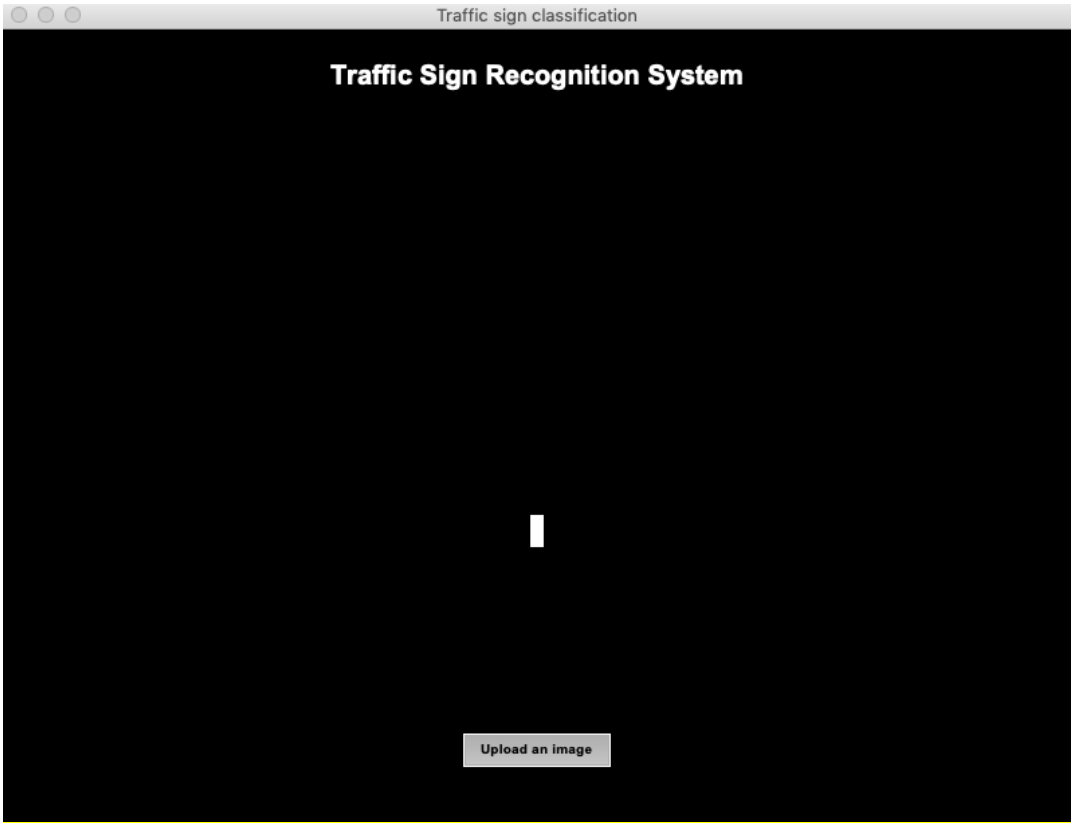
```
model.save('/Users/avi/Documents/UMKC/PythonDL/project/model.h5')
```

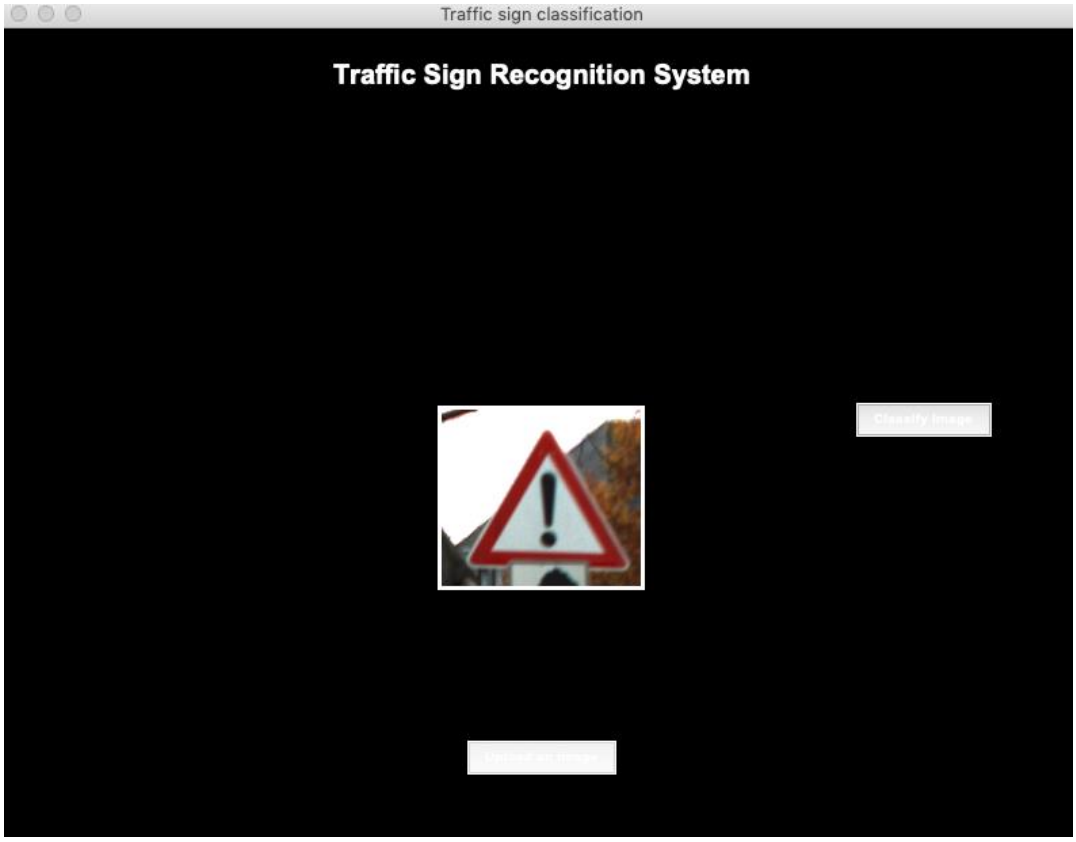
Creating GUI (Desktop Application)

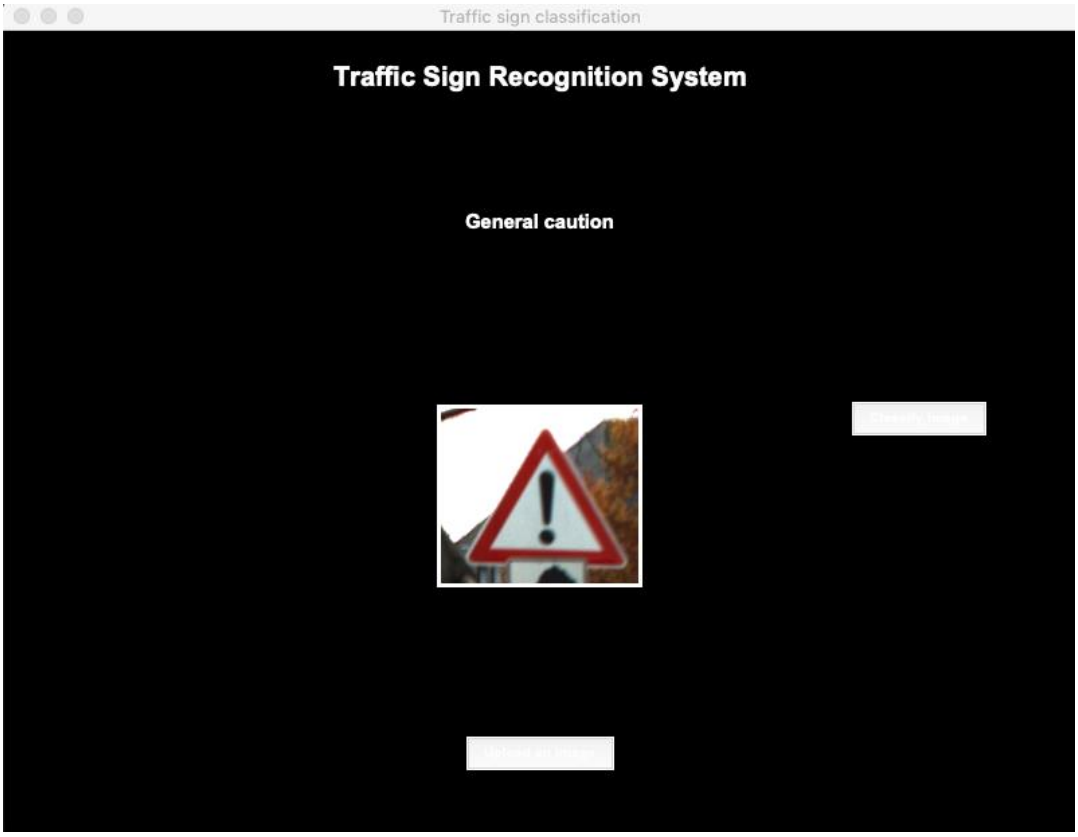
- *Using Tkinter Python Library*
- *Loading the 'model.h5' binary file to the Python script to classify*




IDLE File Edit Format Run Options Window Help
100% Fri 8:09 PM
gui.py - /Users/avi/Documents/UMKC/PythonDL/project/gui.py (3.7.6)
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)
print(image.shape)
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()
Ln: 1 Col: 0


Traffic sign classification
Traffic Sign Recognition System
Upload an image


Traffic sign classification
Traffic Sign Recognition System
Classify Image
Upload an Image


Traffic sign classification
Traffic Sign Recognition System
General caution
Classify Image
Upload an Image