Traffic Signs Recognition

Deep learning CNN Project

Contents

- What is Traffic Signs Recognition?
- Problem Statement
- Data source
- Setting up tensorflow GPU
- Importing all necessary libraries
- Importing data and preprocessing
- Converting image and label lists into numpy arrays
- Splitting data and labels into training and testing dataset
- CNN Model Building
- Compilation of the model
- fitting of the model

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.

- We have to build a Deep Neural Network model that can classify traffic signs present in the image into different categories.

- With this model, we should be able to read and understand traffic signs which are a very important task for all autonomous vehicles.

Problem Statement

- We 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.
- you have to build a Deep Neural Network model that can classify traffic signs present in the image into different categories.
- With this model, we should be able to read and understand traffic signs which are a very important task for all autonomous vehicles

Data source

- The dataset contains more than 50,000 images of different traffic signs. It is further classified into 43 different classes.
- 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 you will use for testing your model.
- The ‘train’ folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42.
- You have to explore the dataset and then build a CNN model. You can use train data for training the model and test the model with the test dataset.

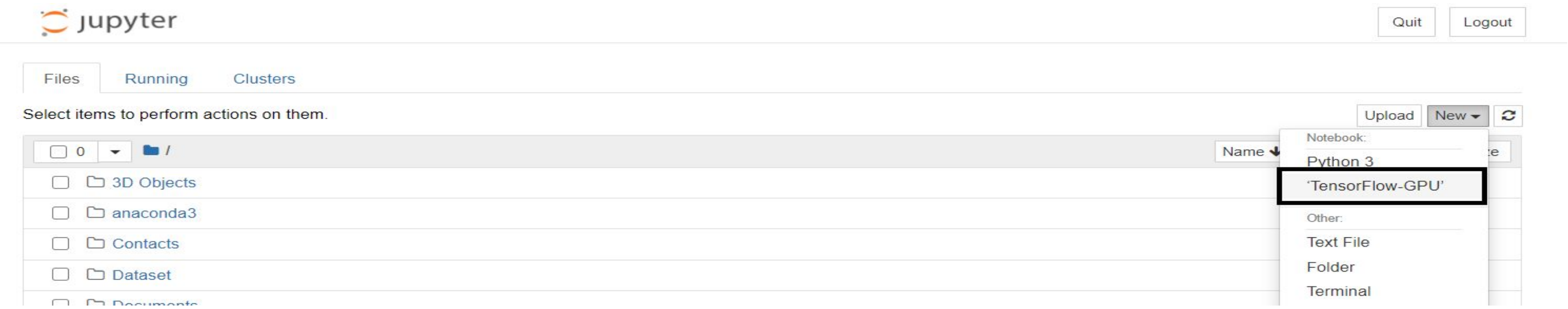
Setting up tensorflow GPU

Tensorflow GPU is not ready available in jupyter notebook. It must be set-up through anaconda prompt and activate it .
After set-up choose tensorflow GPU as shown

Anaconda Prompt (anaconda3) - jupyter notebook

```
(base) C:\Users\sai>activate tensorflow

(tensorflow) C:\Users\sai>jupyter notebook
[I 21:03:27.775 NotebookApp] The port 8888 is already in use, trying another port.
[I 21:03:27.781 NotebookApp] Serving notebooks from local directory: C:\Users\sai
[I 21:03:27.781 NotebookApp] Jupyter Notebook 6.1.6 is running at:
[I 21:03:27.781 NotebookApp] http://localhost:8889/?token=8fcb3c6068fc1802d6aa2165d51df497bc2a4f7a643e4baf
[I 21:03:27.782 NotebookApp]  or http://127.0.0.1:8889/?token=8fcb3c6068fc1802d6aa2165d51df497bc2a4f7a643e4baf
[I 21:03:27.782 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 21:03:27.868 NotebookApp]
```



Importing all necessary libraries

Importing required libraries for CNN project

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

These are the libraries required for traffic sign recognition CNN project.

Importing data and preprocessing

- 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.

```
data = []
labels = []
categories = 43
height = 30
width = 30
curr_path = os.getcwd()                    # The method os. getcwd() in Python returns the current working directory of a process.

for i in range(categories):
    path = os.path.join(curr_path,'Dataset/train',str(i))       #joins one or more path components
    images = os.listdir(path)                                   #lists files and directories in the given path
    for a in images:
        try:
            image = Image.open(path + '//'+ a)                  #opening other image file
            image = image.resize((height,width))                #resizing the image to maintain uniform
            image = np.array(image)                             #getting array of images using numpy
            data.append(image)                                  #appending images data to data list
            labels.append(i)                                    #appending the labels list
        except:
            print("Error loading image")
```

Converting image and label lists into numpy arrays

We need to convert the data and label list into numpy arrays for feeding to the model.

Converting image and label lists into numpy arrays

We need to convert the data and label list into numpy arrays for feeding to the model.

```
data = np.array(data)                    #data list into numpy array
labels = np.array(labels)                #label list into numpy array
print(data.shape, labels.shape)          #shape of data and label
```

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

The shape of data is (39209, 30, 30, 3)

which means that there are 39209 images

image size 30×30 pixels

The last 3 defines the data contains colored images i,e (RGB value).

Splitting data and labels into training and testing dataset

Splitting data and labels into training and testing dataset

```
X_train, X_test, y_train, y_test = train_test_split(data,          #train_test_split() method to split training and testing data
                                                    labels,
                                                    test_size=0.2,
                                                    random_state=30)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)    # shape of training and testing
```

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

test size is 0.2 i,e 20 % data

For training

The shape of data is (31367, 30, 30, 3)

which means that there are 31367 images

image size 30×30 pixels

The last 3 defines the data contains colored images i,e (RGB value).

for testing

The shape of data is (7842, 30, 30, 3)

which means that there are 7842 images

image size 30×30 pixels

The last 3 defines the data contains colored images i,e (RGB value).

CNN Model Building

To classify the images into their respective categories, we will build a CNN model. CNN is best for image classification purposes.

```
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=32, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.3))
model.add(Flatten())
model.add(Dense(190, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
```

Compilation of the model

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

Compile defines the loss function, the optimizer and the metrics

Compilation of the model

Compile defines the loss function, the optimizer and the metrics

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

Fitting of 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.
- Our model got a 93% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy and the loss.

Trains the model for a fixed number of epochs

```
model.fit(X_train, y_train, batch_size=64,
          epochs=15,
          validation_data=(X_test, y_test))
```

```
Epoch 1/15
491/491 [==============================] - 90s 181ms/step - loss: 4.2762 - accuracy: 0.1757 - val_loss: 0.9082 - val_accuracy:
0.7886
Epoch 2/15
491/491 [==============================] - 91s 186ms/step - loss: 1.0949 - accuracy: 0.6931 - val_loss: 0.2434 - val_accuracy:
0.9436
Epoch 3/15
491/491 [==============================] - 85s 174ms/step - loss: 0.5752 - accuracy: 0.8376 - val_loss: 0.2187 - val_accuracy:
0.9555
Epoch 4/15
491/491 [==============================] - 89s 182ms/step - loss: 0.4539 - accuracy: 0.8788 - val_loss: 0.1211 - val_accuracy:
0.9719
Epoch 5/15
491/491 [==============================] - 91s 186ms/step - loss: 0.3086 - accuracy: 0.9132 - val_loss: 0.0856 - val_accuracy:
0.9781
Epoch 6/15
491/491 [==============================] - 85s 174ms/step - loss: 0.3135 - accuracy: 0.9177 - val_loss: 0.0774 - val_accuracy:
0.9820
Epoch 7/15
491/491 [==============================] - 82s 168ms/step - loss: 0.2571 - accuracy: 0.9311 - val_loss: 0.0788 - val_accuracy:
0.9805
Epoch 8/15
491/491 [==============================] - 83s 168ms/step - loss: 0.2405 - accuracy: 0.9328 - val_loss: 0.0590 - val_accuracy:
0.9867
```

Conclusion

- With the provided dataset we were able to build a model without any hurdles. Data provided by organizations is pretty huge.
- In Data Science the weightage of the data plays a huge role in the building model.
- Finally we are able to gain an accuracy of 93% in this traffic sign recognition CNN model.

- Limitations of this work and Scope for Future Work

  What are the limitations of this solution provided, the future scope?

  Preprocessing data was a bit of a complex task in this project.

  Training the model was very time consuming.