Project Name: Machine Learning Based Traffic Sign Detection and Classification

Anuj Kumar Tahlan

Student Number: 21252775

Session: 2021 – 2022

Maynooth University

Department of Computer Science

Head of Department: Dr.Joesph Timoney

Supervisor Name: Prof.Tim McCarthy

Date – 02 August 2022

Traffic Sign Recognition

- It is a process of detecting and classifying the traffic signs from real world scenarios.
- Consists of two parts :
    1. Traffic Sign Detection
    2. Traffic Sign Classification

This project contains both parts.

Why it is important ?

- ❖ Increasing number of vehicles in the world.
- ❖It is certainly possible to miss some traffic signs while driving on road.
- ❖ Provides Real time feedback to driver and thus assists.
- ❖Reduces traffic accidents.
- ❖A step towards reliable in driver less cars.

Some Methods for Recognition

- ❖Color Based
- ❖Shape Based
- ❖Histogram Oriented Graphics
- ❖AdaBoost Approach
- ❖Support Vector Machine Based
- ❖LIDAR based approach
- ❖Sliding Window
- ❖Region Proposal Algorithms
- ❖Machine Learning and Deep Learning Algorithms

and many more.

Challenges ?

- ❖Various Lighting conditions through out the day makes the color-based detection unreliable
- ❖Dynamic weather conditions
- ❖Degraded traffic signs
- ❖Fast processing for real time feedback

I have already trained the YOLOv4 model for Traffic Sign Detection and CNN model for Traffic Sign Classification. If you do not want to train the model and only want to test the already trained model please use the following Google Colab Jupyter notebooks.

Project - Traffic Sign Recognition (Contains two subparts: Detection and Classification)

Repository Link for all files - https://github.com/anuj-tahlan/Traffic_Sign_Recognition.git

Part 1 (a) of the Project - Traffic Sign Detection (Detecting traffic signs from real traffic scenes)

code file name - Traffic_Sign_Detection_Using_Yolo(Training and Testing).ipynb

Part 1(b) of the Project - Testing Traffic Sign Detection (Testing the Trained Model)

code file name - Testing_Model_Traffic_Sign_Detection_Using_Yolo.ipynb

Part 2(a) of the Project. It is called, Traffic Sign Classification (Complete Code : Training and Testing)

code file name - Traffic_Signal_Classification(Training and Testing).ipynb

Part 2(b) - Testing Traffic Sign Classification (Testing the trained model)

code file - Testing_Traffic_Sign_Classification_Model.ipynb

Other files link

- Some of the data files were big in size which cannot be uploaded to GITHUB, please use these links to download them. However, if you are using the Code notebooks then these files will be imported at runtime and no need to download.

For Detection Model

Required Files- https://www.dropbox.com/s/an1fp4qy9slgolg/Required_Files.zip

Trained weights for detection model - https://www.dropbox.com/s/pxcp2udk4mqqcin/yolov4_ts_train_7000.weights

Images for testing the detection model – https://www.dropbox.com/s/ttgnti691mr5tzg/TSD_Testing_Images_Videos.zip

Darknet folder - https://www.dropbox.com/s/sbyik9ci902l2qs/darknet.zip

For classification Model

Trained CNN Model - https://github.com/anuj-tahlan/Traffic-Sign-Classification/blob/492c1adb458642f4e2fe57c7532ae22e1c69ee92/TSC.h5

Testing Images - https://www.dropbox.com/s/ramoqqq2q35gbct/TestingImages.zip

Part 1 of the Project : Traffic Sign Detection

The following slides will describe about the part 1 of the project, which is Traffic Sign Detection.

In this part of the model, Traffic signs are detected from real time scenarios (I have used real traffic scene images and a video of actual driving scenes to test the model).

Top CNN Based Object Detection Methods ?

- ❖R-CNN - use selective search to extract just 2000 regions from the image, referred as region proposals.
- ❖Fast R-CNN
- ❖Faster-R-CNN
- ❖YOLO

Comparision

The COCO dataset makes object detection more challenging, as detectors often have substantially lower mAP. Here is a comparision of accuracy of different Object Detection Algorithm.

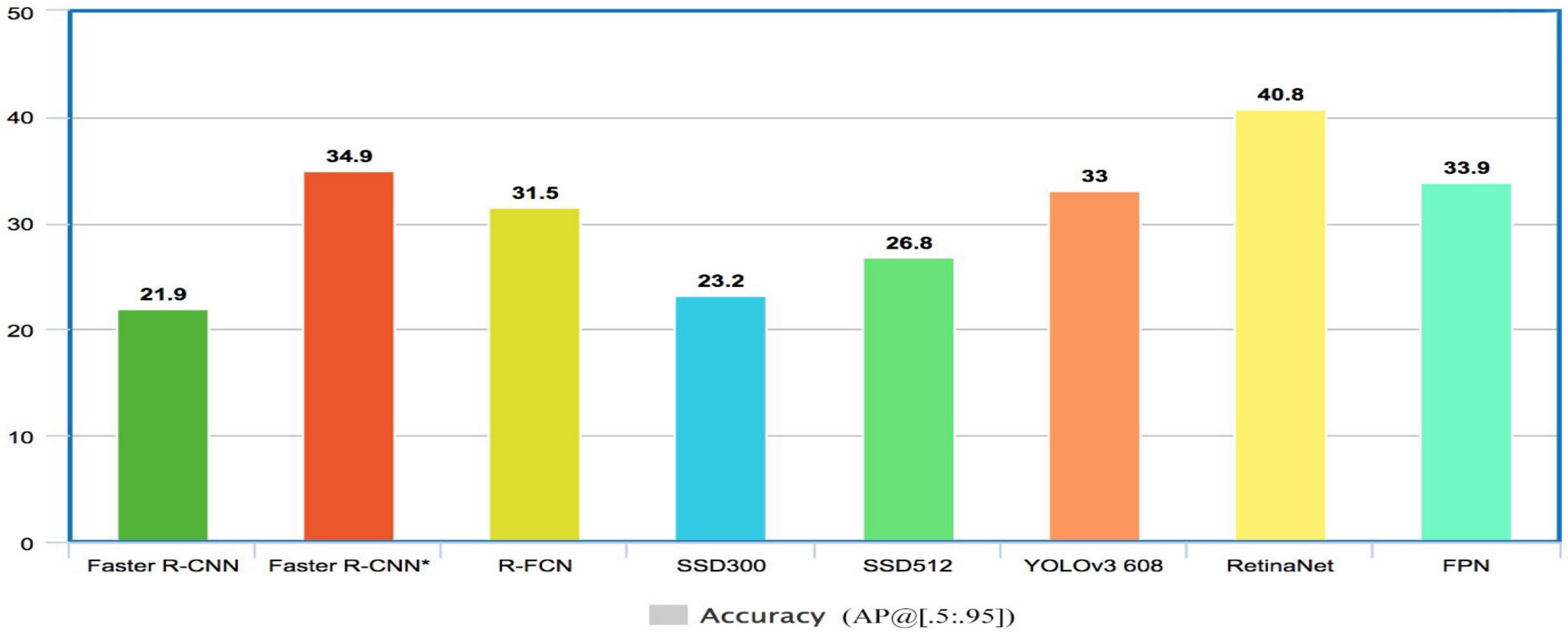


Figure 1. Hui, Jonathan. (2018). Accuracy . https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

What can be inferred from the comparision ?

- The comparision results doesn’t justify that one object detector is better than another, because these expirements are done under different settings. Certain factors such as Input image resolutions and feature extractors impact speed.

What can be inferred from the comparision ? Continue...

In general:

- ❖Region based detectors such as R-CNN are comparitively give slightly better accuracy when real-time speed is not required.
- ❖ While single shot detectors such as YOLO and its versions are best for real-time processing. However, the accuracy requirements must be verified for the application. If the model justify the accuracy requirements for the real-time processing than Yolo can be used.

Why I choose Yolov4 ?

Traffic Sign Detection – is a real-time requirement and the faster and accurate responses can help the driver to make decisions timely. The ability to reliably and quickly detect small objects at high speeds is crucial for Traffic Sign Detection (TSD).

This is the reason I have proposed Yolov4 object detector model for this project.

Dataset

❖The Dataset used for this part of the project is :

German Traffic Sign Detection Benchmark

❖ Download the data set from the official website:

https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/published-archive.html

Or

https://www.kaggle.com/datasets/safabouguezzi/german-traffic-sign-detection-benchmark-gtsdb

Steps

Re-formatting the dataset

- ❖Unzip the folder [FullIJCNN2013]
- ❖Inside the folder, 43 sub-folders named from 00 to 42. These sub-folders denote the class to which traffic sign belongs to. For example, folder 00 contains traffic speed signs for 20kmph, folder 19 contains traffic sign- Dangerous curve left. Similarly, each folder contains separate specific classes.
- ❖We will later
  categorise these classes into 4 main classes:
  (a) Prohibitory category: circular Traffic Signs with white background and red border line
  (b) Danger category :triangular Traffic Signs with white background and red border line
  (c) Mandatory category: circular Traffic Signs with blue background
  (d) Other category: Rest other signs.

Steps continued..

- ❖900 images [00000.ppm, 00001.ppm up to 00899.ppm] of real traffic signs.
- ❖The folder also contains A text file gt.txt which contains the ground truth annotations for test dataset.The text file contains lines of the form:

ImgNo.ppm;leftCol;topRow;rightCol;bottomRow;ClassID

for each traffic sign in the dataset.The first field refers to the image file the traffic sign is located in. Field 2 to 5 describe the region of interest (ROI) in that image. Finally, the ClassID is an integer number representing the kind of traffic sign.

Steps continued..
Converting given data to Yolo Format

- The raw data need to be converted in YOLO Format bounding box data. YOLO accept data in a specific format. The format identifies each image and the bounding box in that image. A supervised machine learning activity requires the labelling of data as a necessary step

Converting given data to Yolo Format...

- The process of labelling data involves a lot of manual work.There are various data labeling methods.Rectangular boxes called bounding boxes are used to specify where the target object is in the image.Polygon segmentation, Key-Point and Landmark,Lines and Splines.
- For YOLO, we will focus on Bounding boxes.Bounding boxes are rectangular boxes that are used to specify where the target object is to be placed.Bounding boxes are often represented by one coordinate (x1, y1) and the bounding box's width (w) and height (h), or by two coordinates (x1, y1) and (x2, y2).

Converting given data to Yolo Format...

- Create a new folder named ts (just an abbrevation for training set). You can use any name for this folder.
- You can create this folder wherever you want, but I will suggest to create it outside the FullIJCNN2013 folder. Now select all the .ppm format files in the,FullIJCNN2013 folder,and copy them to ts folder. Copy the gt.txt file. Now the ts folder will only contain 900, .ppm format image files and gt.txt file.

Converting given data to Yolo Format. continue...

gt.txt file includes coordinates of bounding boxes for all images. These must be converted into YOLO format.

To achieve this task, convert images from .ppm format to .jpg format. Also, next to every image file we need to create annotation text file with extension .txt, that has the same name as image file has. For example, 0001.ppm file then this must be converted to a 00001.jpg file and an annotation file with 00001.txt will be created.

Currently, the gt.txt annotation file has following bounding box coordinates : X min, Y min, X max,Y max.

Now to use YOLO for training,the training and testing data for bounding boxes must be in YOLO format.

For Yolo Format, Bounding boxes coordinates must be in the following format: [class][BoundingBox centre in x][Bounding Box centre in y][object width][object height].

Converting given data to Yolo Format. continue...

So, the tasks to be performed are as follows:
1. Convert .ppm files to .jpg files, using a python script.
2. Convert annotations of bounding boxes into Yolo Format To achieve this task I wrote some python code to convert the .ppm files to .jpg files, which I will explain next
3. Create txt files with annotations. These .txt files will be next to every image that will have the same names as images files have.

Converting given data to Yolo Format. continue...

Algorithm:

1. Setting up full path
2. Lists for categories
3. Loading data Frame with original annotations
4. Calculating numbers for YOLO format without normalization
5. Getting image’s real width and height
6. Normalizing numbers for YOLO format
7. Saving annotations in txt files
8. Saving images in jpg format

Result: txt files next to every image with annotations in YOLO format

Google Colab

For all the coding part for this project, Google Colaboratory (Google Colab) is used. Colab allows anybody to write and execute arbitrary python code through the browser and also has inbuilt required pandas, tensorflow etc. machine learning libraries and tools.

Converting given data to Yolo Format. continue...

- ❖ Download the file convert-annotations.ipynb from the GitHub source code of this project.
- ❖ The file is well commented to understand the steps.
- ❖ zip the ts folder which was downloaded earlier in dataset.

Note - When a folder is zipped in MacOS system there is additional folder created with name __MACOSX folder. This can create issues when extracting data on drive so to avoid this folder from creation, please follow the following steps:

Right click on the folder to be zipped and choose 'New Terminal at Folder' option. This will open the folder in MacOS terminal window. Now use the following command to zip the folder:

```
zip -r <foldername>.zip . -x ".*" -x "__MACOSX"
```

where <foldername> is the name of the folder to be zipped. After some time depending on the folder size it will be zipped. Now, when this folder is extracted there will be no __MACOSX folder present in it.

- ❖ Upload the zipped ts folder (ts.zip) on Google Drive inside Traffic_Sign_Detection_YOLOv4 folder.
- ❖ Now run the convert-annotations.ipynb file code step(cell) by step(cell) as mentioned in the file and it will convert the .ppm files to .jpg and with their .txt annotations bounding box file generated.

Create train.txt and test.txt files

❖Download the notebook:

create_train_test_text_file.ipynb

Run the cells off this notebook and train.txt and test.txt files will be created.

These files basically divide the dataset into training and testing.

Once created these files are inside ts folder

Darknet

- ❖ Darknet is a high-performance open-source neural network implementation framework. It is written in C and CUDA, it can be integrated with CPUs and GPUs. Darknet can be used for implementing advance neural network.
- ❖ There are two ways to use Darknet to either clone it directly into the Project root folder on Google drive, using the below command.

!git clone https://github.com/AlexeyAB/darknet.

The cloning is required only once when the code is executed for the first time.

In the Google Colab Jupyter
Notebook:

Traffic_Sign_Detection_Using_Yolo(Training and Testing).ipynb

Run the following command:

```
!git clone https://github.com/AlexeyAB/darknet
```

Second method is to download the Darknet folder using the same link from above, extract it and upload
that extracted folder on Google Drive in the project folder, Traffic Sign Detection YOLOv4. I have downloaded the folder manually to drive

Preparing Files for Training

- Prepare certain files needed for training in Darknet framework. These files are:
  1. ts data.data
  2. classes.names
  3. train.txt
  4. test.txt

Preparing Files for Training ts_data.data

Open any text editor and create a file with name ts data and save it with extension as .data. Hence, the name ts_data.data. This file must be uploaded to the cfg folder inside Darknet folder on google drive.

Content of ts data.data for this project:

```
classes = 4

train =/content/gdrive/MyDrive/Traffic Sign Detection YOLOv4/ts/train.txt

test = /content/gdrive/MyDrive/Traffic Sign Detection YOLOv4/ts/test.txt

names =/content/gdrive/MyDrive/Traffic Sign Detection YOLOv4/ts/classes.names

backup=/content/gdrive/MyDrive/Traffic Sign Detection YOLOv4/weights
```

Preparing Files for Training classes.names

Open any text editor and create a file with name ts data and save it with extension as .data. Hence, the name ts_data.data

Classes.names will contain the name of 4 classes in which we categorised our images during dataset preparation. It must be in the same order as the categorisation is done in convert-annotations.ipynb Where ‘p’ referred to prohibitory , ‘d’ for danger and so on.

Content of classes.names for this project:

```
prohibitory
danger
mandatory
other
```

Preparing Files for Training

- Train.txt and test.txt are already created and they must be present inside the ts folder which is present inside the project folder (Traffic_Sign_Detection_YOLOv4) on google drive.

Configuring the YOLOv4 for training

- Navigate to Darknet folder, inside this folder there are many sub-folders, we only have to work with configuration file:

yolov4-custom.cfg file.

- Create two new files with the names:
  yolov4 ts_train.cfg and yolov4 ts_test.cfg. Copy the contents of yolov4-custom.cfg into these files.

Updating yolov4 ts_train.cfg

In ts_train.cfg following changes are required to be made in the file for:

#Training section. Comment the default values for below lines and change them to as per mentioned below:

change batch = 32 # instead of 64
2. subdivisions = 16
3. width = 416 #instead of 608
4. height = 416 #instead of 608

Note different batch numbers can be used for training :

batch	subdivision
64	8 or 16 or 32
32	8 or 16
16	8 or 4
8	4 or 2

Updating yolov4 ts_test.cfg

❖ Updating yolov4 ts_test.cfg

In ts_test.cfg following changes are required to be made in the file for:

#Testing section. Comment the default values for below lines and change them to as per
mentioned below:

#Testing
1. change batch = 1
2. subdivisions = 1
3. width = 416 #instead of 608
4. height = 416 #instead of 608

- To update the number of iterations for training, change max batches that is total number of iterations for training and steps are used for updating learning rate. max batches are updated according to the number of classes. max batches = classes × 2000 , but it should not be less than 4000. steps are calculated as 80% and 90% from max batches.
  For this project, the number of classes is equal to 4, then:
  max batches=8000
  steps=6400,7200
  Update these values in yolov4 ts train.cfg and also in yolov4 ts test.cfg. 3.173.18
  Leave all other parameters to default
- Optimization
  • momentum=0.949 - hyperparameter for optimizer that defines how much history will influence
  further updating of weights
  • decay=0.0005 - decay the learning rate over the period of the training
  • learning rate=0.001 - initial learning rate for training

Update number of classes and CNN filters

- There are three Yolo layers in the configuration file. Find it with simple search for word YOLO. In each YOLO layer, change classes = 4 or and filters = 27 3.19 filters is calculated as, filters = (classes + 5) × masks.
  YOLO V4 predicts three bounding boxes for every cell of the feature maps hence the value of masks 3. As for this project classes = 4 So, filters = (4 + 5) × 3. is filters = 2

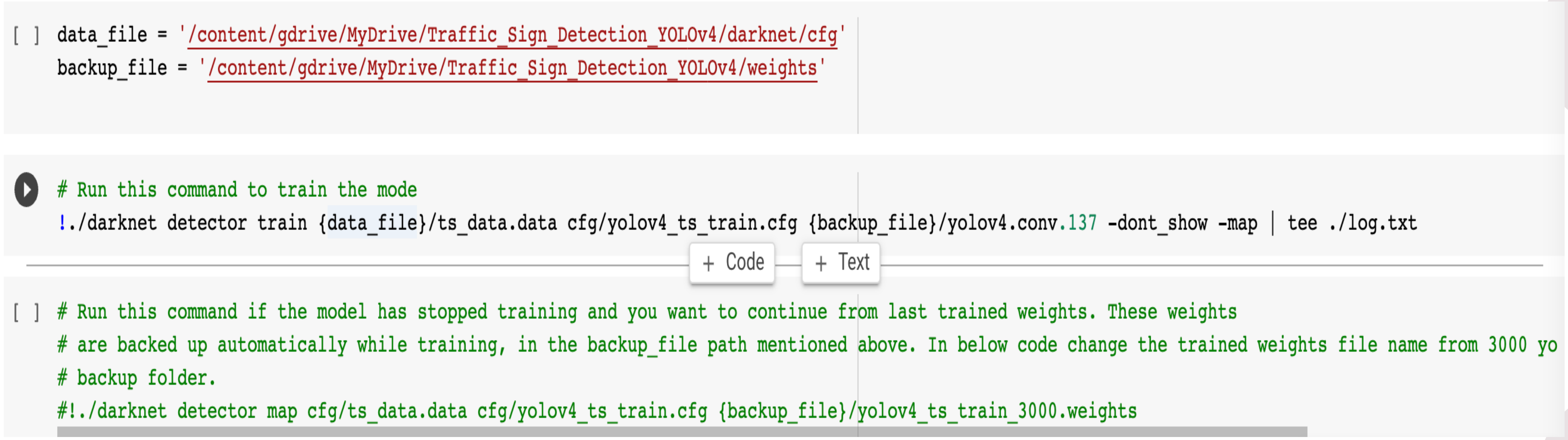
Create weights folder

❖Inside the Project root directory [Traffic Sign Detection YOLOv4] on Google Drive, create another
folder that has name, ”weights”.This folder will be used to store the training weights for the model.

Start training

To start training the detection model
Run the below command cells in the
Jupyter Notebook: Traffic_Sign_Detection_Using_Yolo(Training and Testing).ipynb

Note – Training will take significant time around 7 hours, but if short on time you can also stop it after 3000 weights because there is no significant increase in accuracy mAP results. (See training results slide)



```
data_file = '/content/gdrive/MyDrive/Traffic_Sign_Detection_YOLOv4/darknet/cfg'
backup_file = '/content/gdrive/MyDrive/Traffic_Sign_Detection_YOLOv4/weights'
```

```
# Run this command to train the mode
!./darknet detector train {data_file}/ts_data.data cfg/yolov4_ts_train.cfg {backup_file}/yolov4.conv.137 -dont_show -map | tee ./log.txt
```

```
# Run this command if the model has stopped training and you want to continue from last trained weights. These weights
# are backed up automatically while training, in the backup_file path mentioned above. In below code change the trained weights file name from 3000 yo
# backup folder.
#!./darknet detector map cfg/ts_data.data cfg/yolov4_ts_train.cfg {backup_file}/yolov4_ts_train_3000.weights
```

Training Results of Detection Model

❖ There are eight .weights files in the backup folder (weights folder) which are generated and stored during training time.

They are:

- 1000.weights
- 2000.weights
- 3000.weights
- 4000.weights
- 5000.weights
- 6000.weights
- 7000.weights
- 8000.weights

Selecting weights for testing the model

- Now the question arises which weight file should we provide as input while testing the model ? This depends on the -map values. start checking saved and trained weights from the end one by one calculating mean average precision (mAP).
- The goal is to find weights that have the biggest mAP.For example, if we consider Traffic Signs dataset, and if mAP for 7000 iterations is bigger than for 8000, then it is needed to check weights for 6000 iterations. Next, if mAP for 6000 iterations is already less than for 7000 iterations, then you can stop checking and use weights for 7000 iterations in detection tasks.

Command to find mAP values for each weight

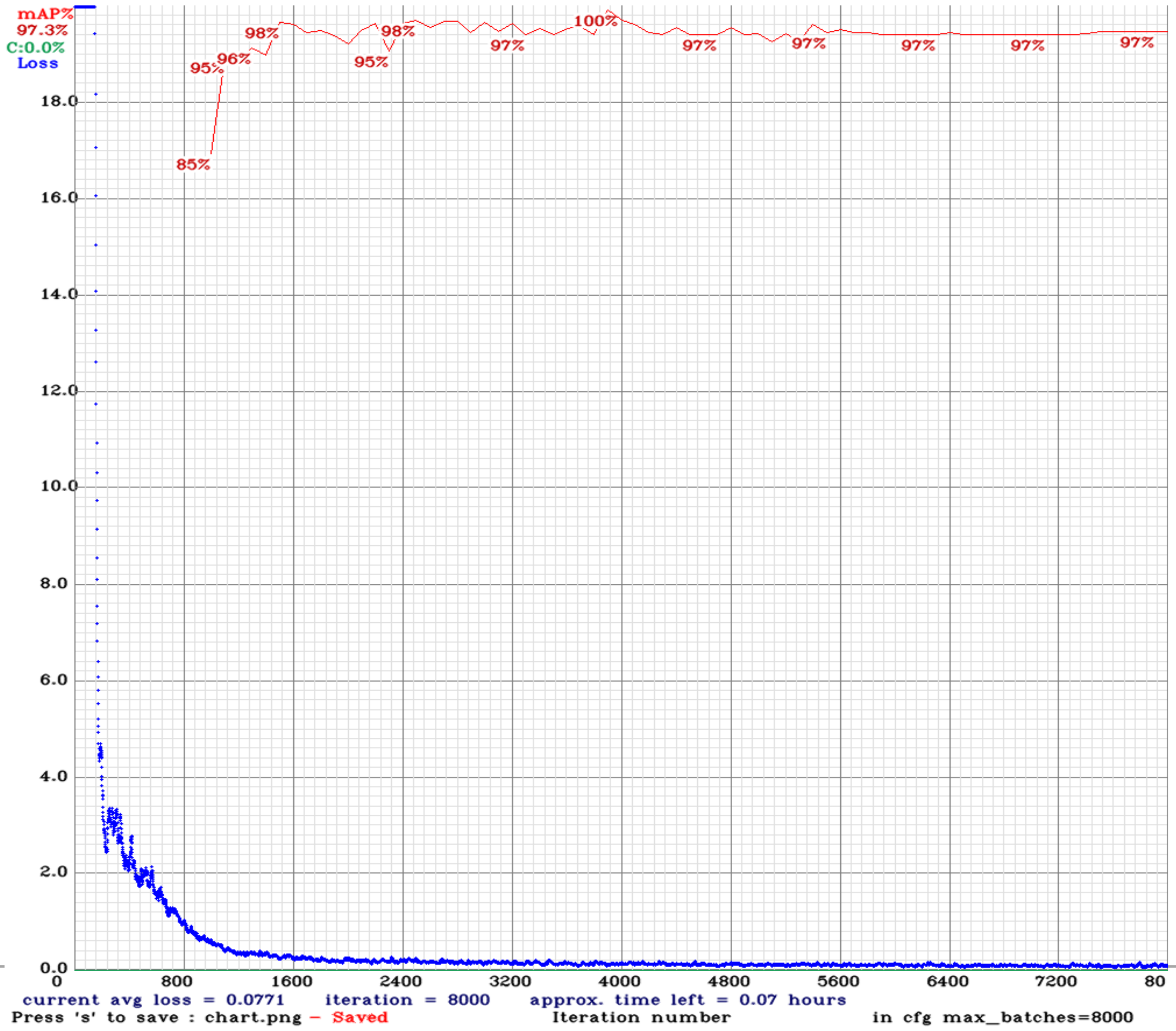
```
./darknet detector map cfg/ts_data.data cfg/yolov4_ts_train.cfg {backup_file}/yolov4_ts_train_8000.weights
```

For each weight file run the following command and the following results:

Calculating mAP for Traffic Signs dataset

weights	mAP=
1000.weights	84.58 %
2000.weights	96.98%
3000.weights	98.25%
4000.weights	98.41%
5000.weights	97.02%
6000.weights	96.99%
7000.weights	97.02%
8000.weights	97.26%

mAP results per 1000 batches



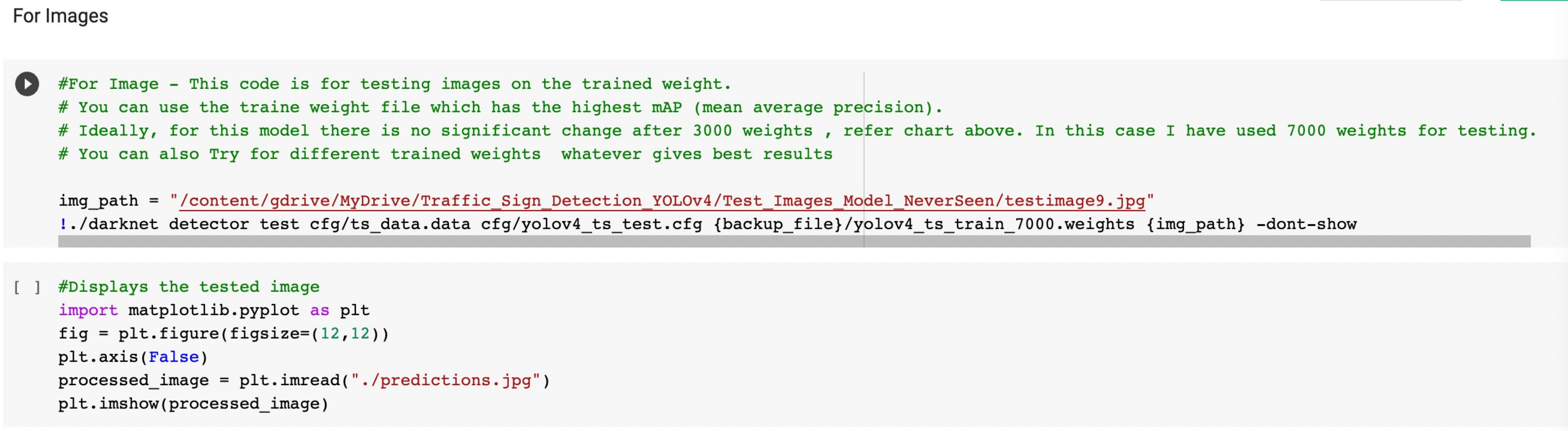
current avg loss = 0.0771 iteration = 8000 approx. time left = 0.07 hours

Press 's' to save : chart.png - Saved

in cfg max_batches=8000

Testing the Detection model on an image

- For testing an image which the model has never seen,to predict the category of traffic signs present in that image, the following command is used:



For Images

```
#For Image - This code is for testing images on the trained weight.
# You can use the traine weight file which has the highest mAP (mean average precision).
# Ideally, for this model there is no significant change after 3000 weights , refer chart above. In this case I have used 7000 weights for testing.
# You can also Try for different trained weights  whatever gives best results

img_path = "/content/gdrive/MyDrive/Traffic_Sign_Detection_YOLOv4/Test_Images_Model_NeverSeen/testimage9.jpg"
!./darknet detector test cfg/ts_data.data cfg/yolov4_ts_test.cfg {backup_file}/yolov4_ts_train_7000.weights {img_path} -dont-show
```

```
#Displays the tested image
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(12,12))
plt.axis(False)
processed_image = plt.imread("./predictions.jpg")
plt.imshow(processed_image)
```

Testing the Detection model on an image

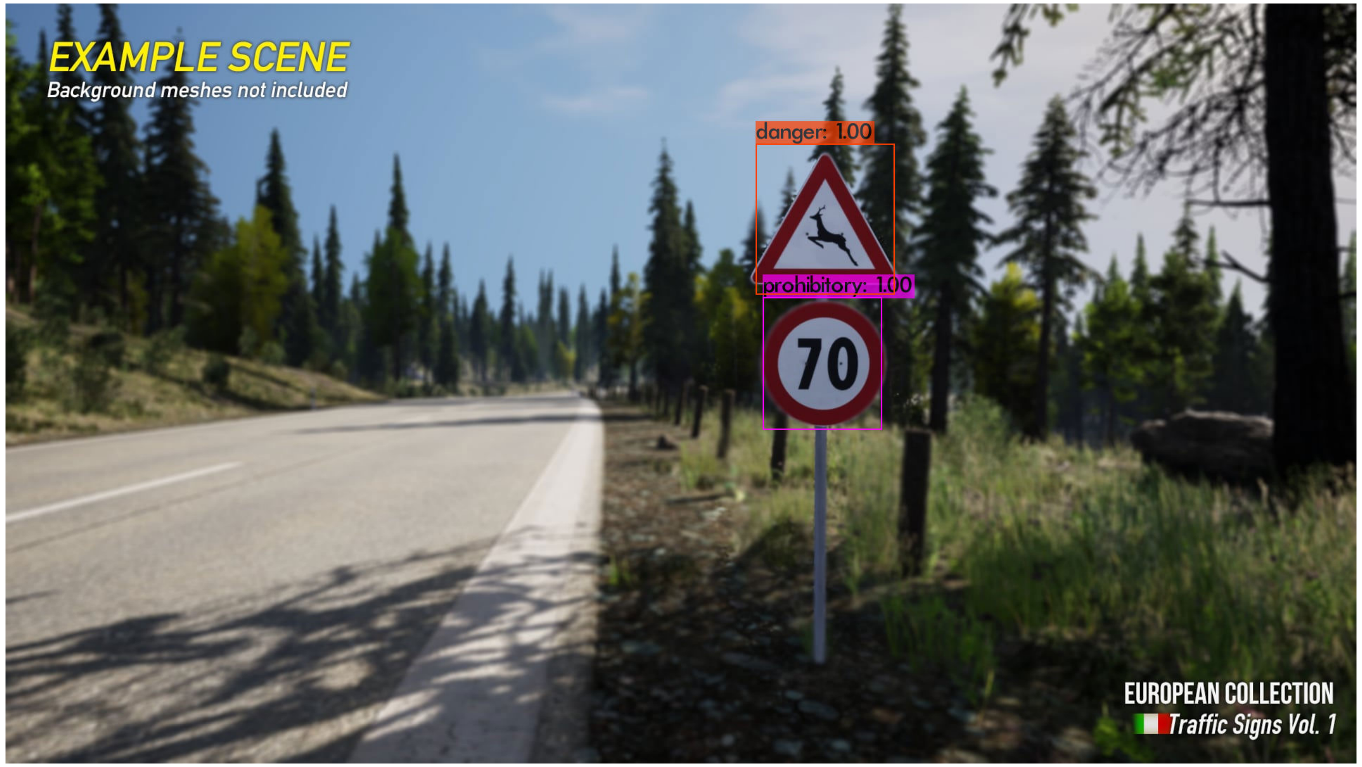
- There is a slight difference in the command for training and testing. For testing !./darknet detector test is used instead of !./darknet detector train. The ts data.data file remains same. For testing we will use testing configuration file (namely, yolov4 ts test.cfg), which was created during preparation of files.
  Now we have to provide the trained weights as input so {backup file}/yolov4 ts train 7000.weights .
  Once predictions is complete, use Matplotlib to print the predictions.jpg file that has the result.

Testing the Detection model on an image - Results

Original

EXAMPLE SCENE
Background meshes not included
70
EUROPEAN COLLECTION
Traffic Signs Vol. 1

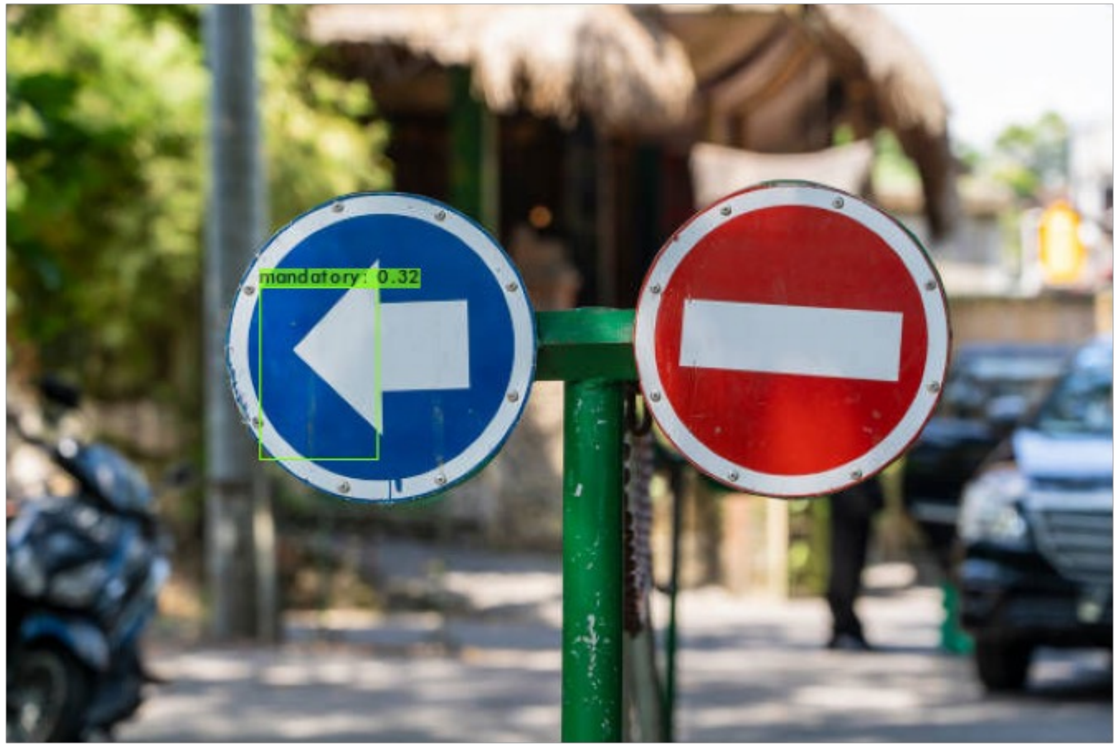
Predicted



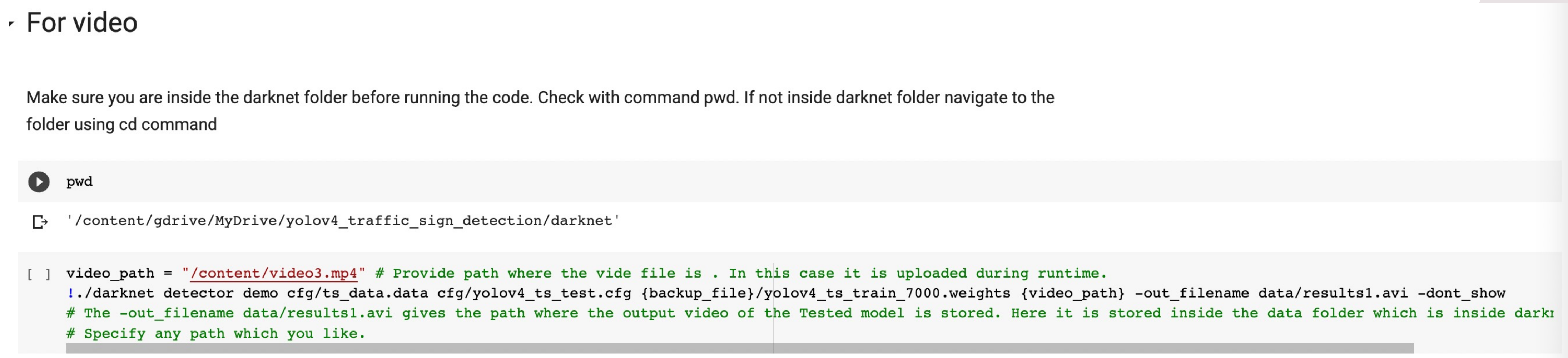
Testing the Detection model on an image - Results

Original

Predicted



Testing the Detection model on a video



For video

Make sure you are inside the darknet folder before running the code. Check with command pwd. If not inside darknet folder navigate to the folder using cd command

```
pwd
```

```
'/content/gdrive/MyDrive/yolov4_traffic_sign_detection/darknet'
```

```
[ ] video_path = "/content/video3.mp4" # Provide path where the vide file is . In this case it is uploaded during runtime.
    !./darknet detector demo cfg/ts_data.data cfg/yolov4_ts_test.cfg {backup_file}/yolov4_ts_train_7000.weights {video_path} -out_filename data/results1.avi -dont_show
    # The -out_filename data/results1.avi gives the path where the output video of the Tested model is stored. Here it is stored inside the data folder which is inside dark
    # Specify any path which you like.
```

Testing the Detection model on a video - Results

Testing Traffic Sign Detection Model on a video (drivers view):
Original Video without any sign detection - https://drive.google.com/file/d/1HpFBcFasKhWuGFHewO8tCOKaAgiWvLt-/view?usp=sharing

Video after Detected Signs - https://drive.google.com/file/d/1-58m87QD03rh52huLgr_09TNSzjfxaq2/view?usp=sharing

Part 2 of the Project : Traffic Sign Classification

This is part 2 of the Recognition Model.

The Detection model trained using Yolov4 model discussed in previous slides is used to detect traffic signs in a traffic scene . It classify the detected signs into one of the four classes:

- prohibitory
- danger
- mandatory
- Other

To further classify the signs into sub-categories of specific types (43 different classes) such as Speed Limit, Left Turn Ahead, Right Curve etc, this Traffic Sign Classification Model will be used.

This model uses CNN, keras/tensorflow and RELU Activation function for training the model.

Traffic Sign Classification

- The Detected traffic sign from the Traffic Scene must be cropped and then fed to the classification model for further sub-class classification in one of these 43 classes,
- However, in my Project, I have not developed the transition from Detection Model to Classification Model and has kept it as future work. So, for the classification model during training and testing the requirement is the input image must only contain a traffic sign and no other symbol. This manual step can be modelled as transition in future where both the Detection Model and Classification Model can be merged.

Some sample image requirements for Training and Testing Traffic Signs Classification

Traffic Sign Classification Dataset

- The dataset used for Classification model is:

German Traffic Sign Recognition Benchmark (GTSRB)

https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

Or

https://benchmark.ini.rub.de

Google Colab Jupyter Notebook for Classification Model

Download the Google Colab Notebook :

Traffic_Signal_Classification(Training and Testing).ipynb

Upload this file to your own google drive and open in Google Colab

The file is well commented for each step.

Steps

1. Import google drive using the same command
2. Inside Google drive create a folder Traffic Sign Recognition and another folder dataset inside this folder.
3. Download and arrange the dataset as explained
4. Unzip the Train.zip and Test.zip using the command specified Also please note, as the dataset is large, once unzipped it takes about 5-10 minutes time for all the images to show in their respective folders. As, it is only required to extract the images once, this is not a big ask.
5. The training dataset has folders from 0 to 42 which represent traffic sign classes,for each image inside the folders convert them to numpy arrays.
6. split the data into 80% training and 20% testing data.
7. Convert labels into one hot encoding.
8. Build the model
9. Compile the model
10. Train the model
11. Save the model
12. Test the model

Arranging the Data - GTSRB Dataset

- Download the GTSRB dataset from the links provided.
- Once downloaded extract the zip file, the folder contains many files but the relevant one are Train folder, Test folder and Test.csv. Create a zip file (compress) version of Train and Test (Train.zip and Test.zip). If using zip on MAC OS please follow the zip steps mentioned on slide 22. Converting given data to Yolo Format. continue...

Arranging the Data - GTSRB Dataset

- 1. Go to your google drive account
  2. Upload the Train.zip and Test.zip files inside the Traffic Sign Recognition/Dataset folder created earlier.

Open Test.csv file in excel and in column H [Path] , path entries are in the format

Test/00000.png change the path entries to:

/content/gdrive/MyDrive/Traffic Sign Classification/dataset/Test/00000.png .

In column 'I' enter the path /content/gdrive/MyDrive/Traffic Sign Classification/dataset/ and copy this path to all the rows in that column.

In column 'I' enter the path /content/gdrive/MyDrive/Traffic Sign Classification/dataset/ and copy this path to all the rows in that column.

In column J use the excel formula =I2&""&H2.
Now drop down this formula to all the rows in column J. Now copy the column J entries, right click and paste it as values in col J itself. Delete column H and column I. Now the column J entries are moved to column H. Name the column as Path.

Save the Test.csv file and then upload it to the "dataset" folder on Google Drive

Splitting the data and one hot encoding

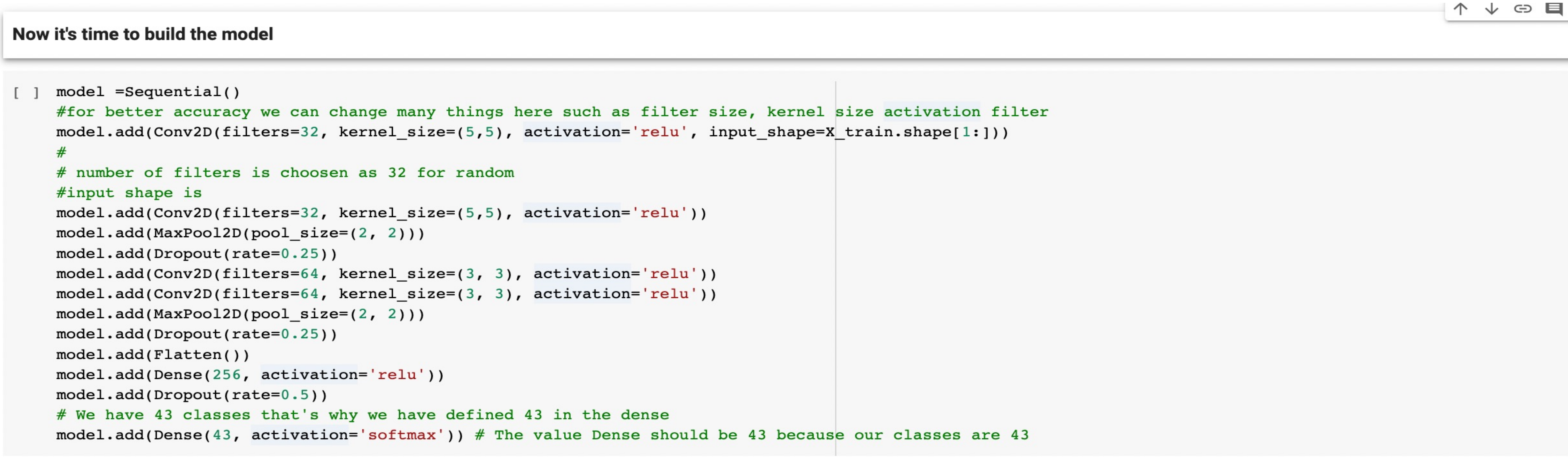
The whole dataset is required to be split into training and testing. The training data is 80% and 20% is testing data. Training and testing data are used to check that the model is working correctly. Testing images are the images which the model has not seen during training. However, this is not a set guideline to split the data into this percentage, the splitting ratio depends on data model size.

X_train, X_test, y_train, y_test = train_test_split(data_list, image_labels, test_size=0.2, random_state=0)

Why to convert labels to one hot encoding ?

- The data with variables is called categorical data. For example, ”dog” and ”cat” are the values for ”pet” variable. Machine learning algorithms cannot work with these kind of labels.The input and output variables required by Machine learning algorithms must be numeric. To convert the categorical data to numeric data there are two ways :
  1) Integer Encoding
  2) One-hot encoding
  Integer Encoding - each unique category value is assigned to integer value. for example ”dog” is 1, ”cat” is 2, ”horse” is 3.
- However, this encoding is reversible and machine learning algorithms can understand this and make it a pattern. This is not what is required here. For categorical data where there is no such one to one relation exist, integer encoding is not the right choice. It may results in poor performance and unexpected and incorrect results. One Hot Encoding - Integer variable encoding is replaced with binary variables. for example :
  dog cat horse
  1 0 0
  0 1 0
  0 0 1
  Following command will convert the text into one-hot encoding:
  y train = to categorical(y train, 43) for training data and y test = to categorical(y test, 43) for testing data
  We can do this one-hot encoding using Scikit-learn (Sklearn), but Keras has inbuilt to categorical method, which is used here.The simplest technique to build a model in Keras is sequential. It allows to build model layer by layer.

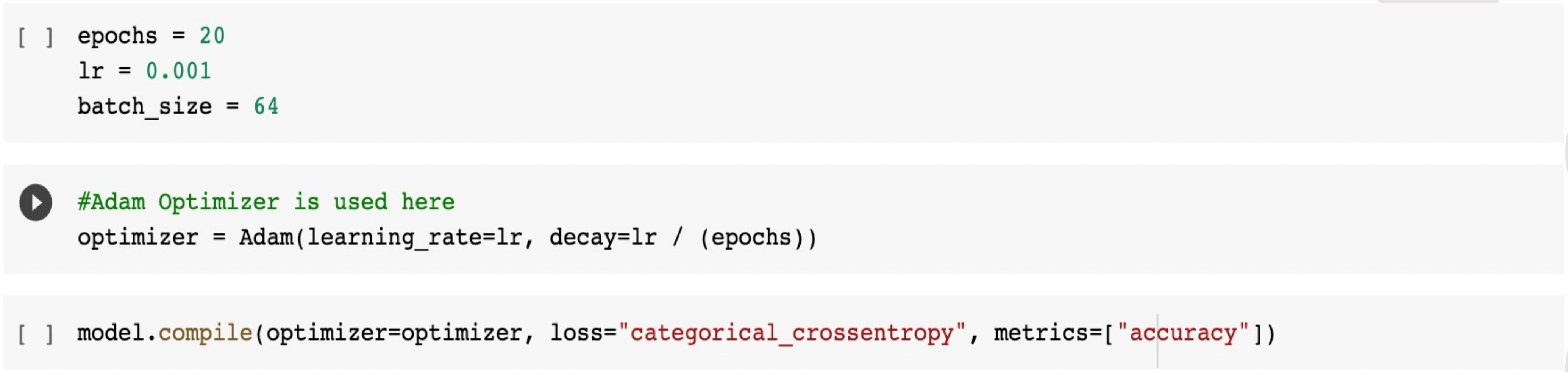
Building the classification model



Now it's time to build the model

```
[ ] model =Sequential()
    #for better accuracy we can change many things here such as filter size, kernel size activation filter
    model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
    #
    # number of filters is choosen as 32 for random
    #input shape is
    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))
    # We have 43 classes that's why we have defined 43 in the dense
    model.add(Dense(43, activation='softmax')) # The value Dense should be 43 because our classes are 43
```

Compile the Model



```
[ ] epochs = 20
    lr = 0.001
    batch_size = 64
```

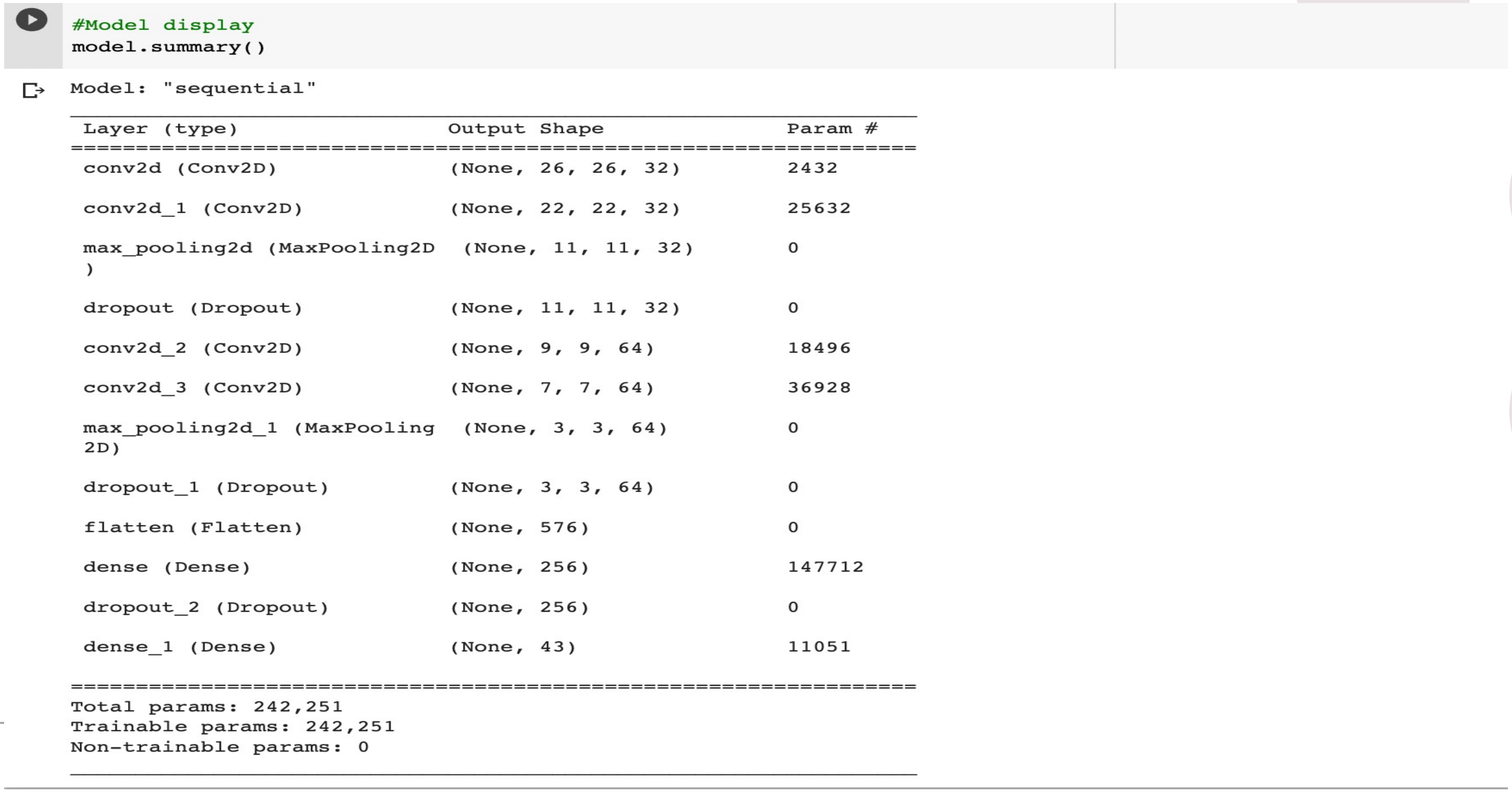
```
#Adam Optimizer is used here
optimizer = Adam(learning_rate=lr, decay=lr / (epochs))
```

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

Epochs , Batch size

- A single epoch occurs when the neural network only processes an entire dataset
  once, both forward and backward. But one Epoch is too big to feed to the computer.Thus, it is divided into batches. Also, if we feed the entire dataset once to the training model, it becomes underfit.
- To optimize the learning process, the entire dataset is passed to the same neural network multiple times. This is also important because to improve the learning and get better results from the gradient descent optimization algorithm.The neural network's weights are adjusted more frequently as the number of epochs rises, and
  the curve shifts from underfitting to optimum to overfitting.So, there are more than one epochs, an epoch is divided into batches and there is no fixed size of a batch it depends on the model size.

Model Summary



```
#Model display
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        2432

 conv2d_1 (Conv2D)           (None, 22, 22, 32)        25632

 max_pooling2d (MaxPooling2D  (None, 11, 11, 32)       0
 )

 dropout (Dropout)           (None, 11, 11, 32)        0

 conv2d_2 (Conv2D)           (None, 9, 9, 64)          18496

 conv2d_3 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_1 (MaxPooling  (None, 3, 3, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 3, 3, 64)          0

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 256)               147712

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 242,251
Trainable params: 242,251
Non-trainable params: 0
_________________________________________________________________
```

Train the Model

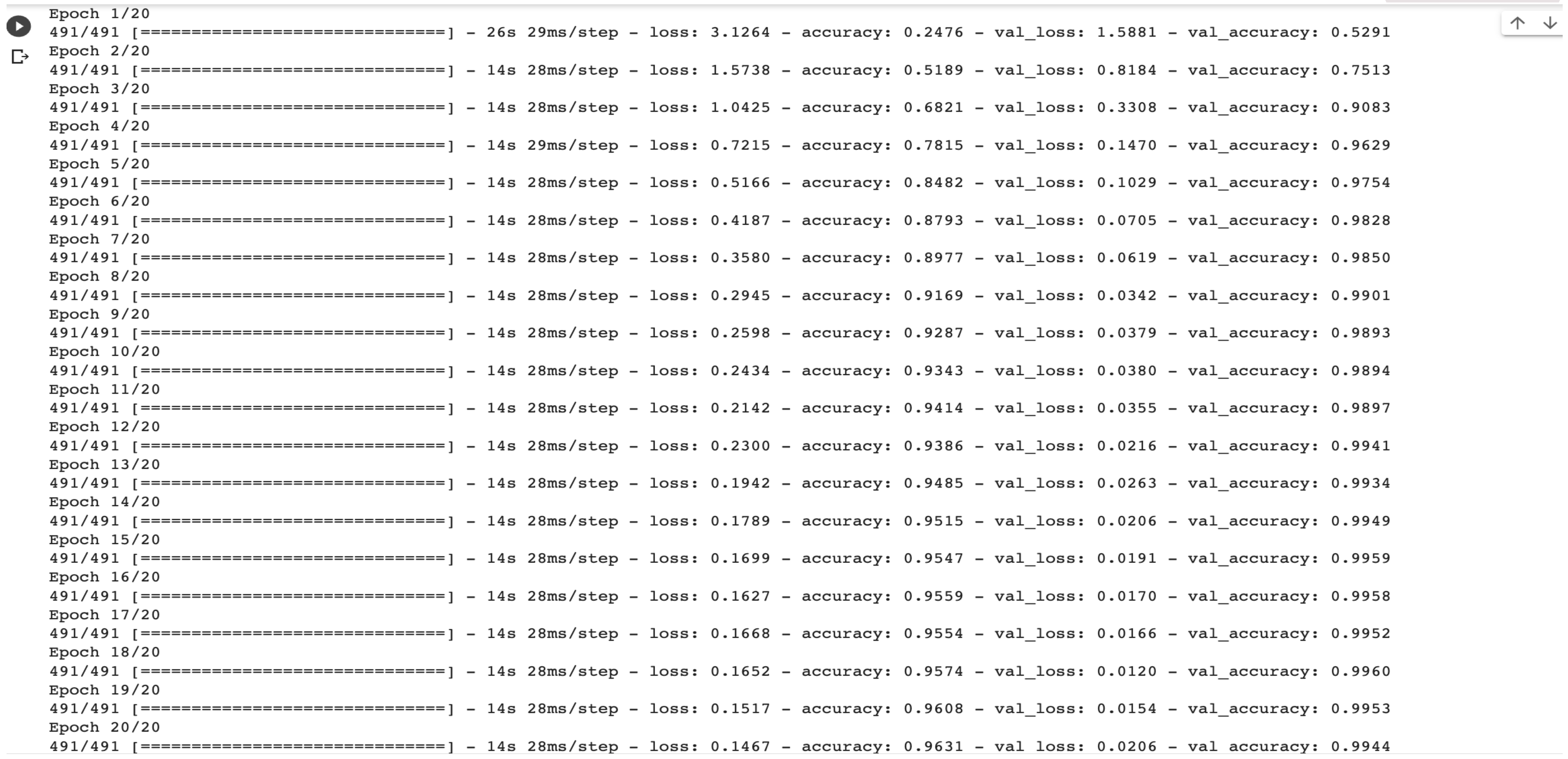
- Train the model following command is used:

```
H = model.fit(
datagen.flow(X_train, y_train, batch_size),
validation_data=(X_test, y_test),
epochs=epochs,
verbose=1)
```

For training the model, batch size is 64 and epochs are 20 and learning rate (lr) 0.001. Adam optimizer is used. For each batch there are 491 iterations. Once the training is done the model gave 99.44%accuracy

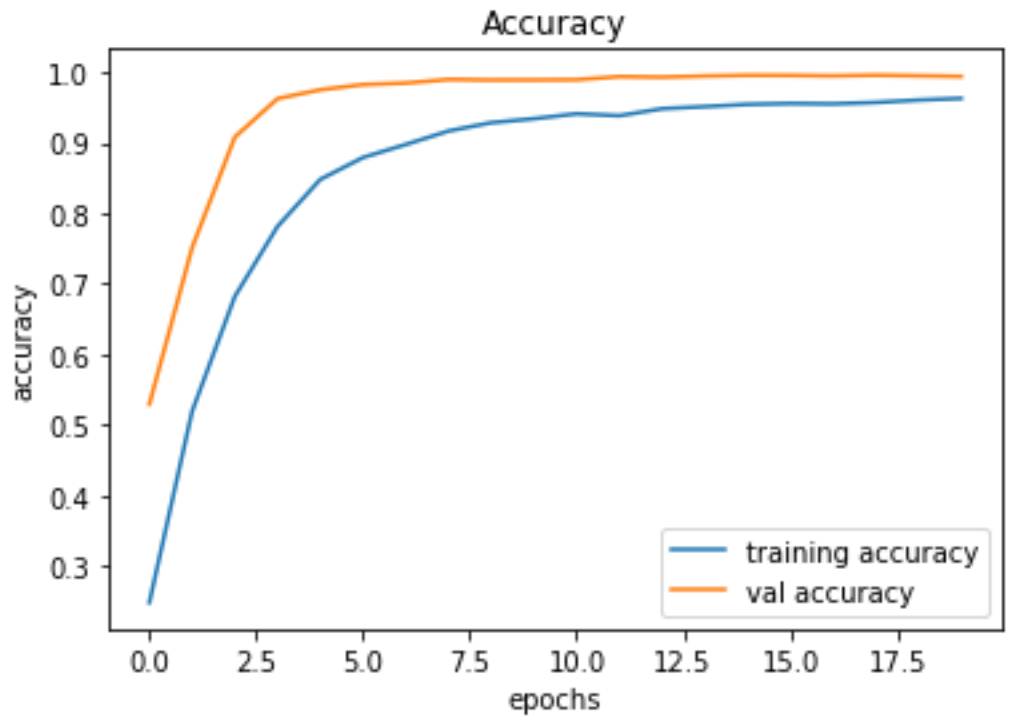
Also save the model. The code is provided in the notebook.

Training Result

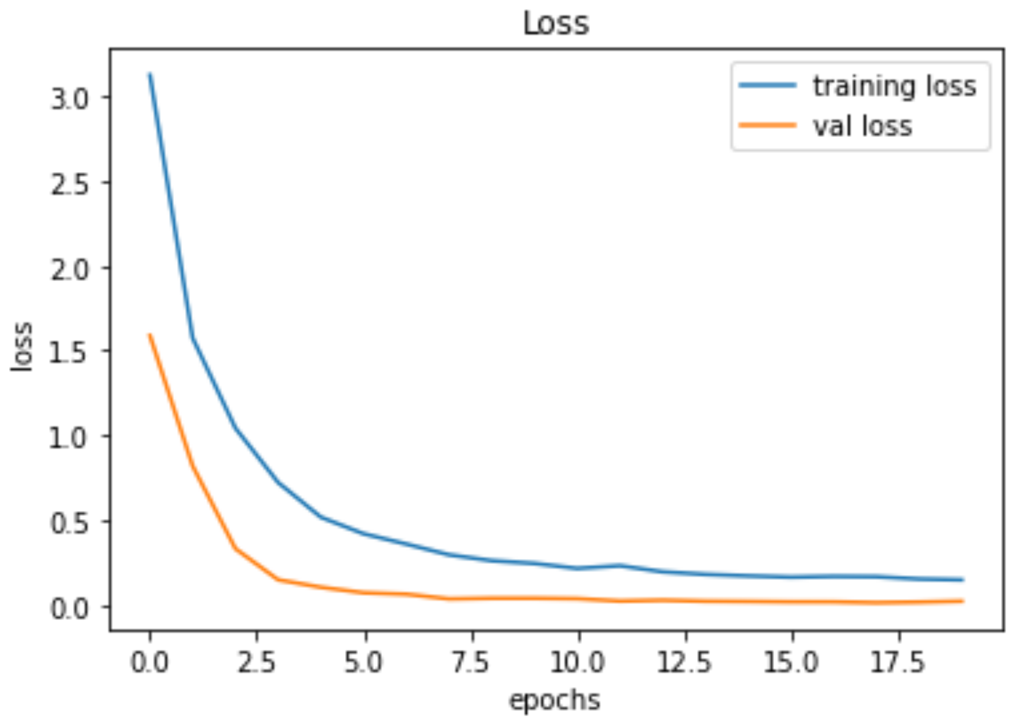


```
Epoch 1/20
491/491 [==============================] - 26s 29ms/step - loss: 3.1264 - accuracy: 0.2476 - val_loss: 1.5881 - val_accuracy: 0.5291
Epoch 2/20
491/491 [==============================] - 14s 28ms/step - loss: 1.5738 - accuracy: 0.5189 - val_loss: 0.8184 - val_accuracy: 0.7513
Epoch 3/20
491/491 [==============================] - 14s 28ms/step - loss: 1.0425 - accuracy: 0.6821 - val_loss: 0.3308 - val_accuracy: 0.9083
Epoch 4/20
491/491 [==============================] - 14s 29ms/step - loss: 0.7215 - accuracy: 0.7815 - val_loss: 0.1470 - val_accuracy: 0.9629
Epoch 5/20
491/491 [==============================] - 14s 28ms/step - loss: 0.5166 - accuracy: 0.8482 - val_loss: 0.1029 - val_accuracy: 0.9754
Epoch 6/20
491/491 [==============================] - 14s 28ms/step - loss: 0.4187 - accuracy: 0.8793 - val_loss: 0.0705 - val_accuracy: 0.9828
Epoch 7/20
491/491 [==============================] - 14s 28ms/step - loss: 0.3580 - accuracy: 0.8977 - val_loss: 0.0619 - val_accuracy: 0.9850
Epoch 8/20
491/491 [==============================] - 14s 28ms/step - loss: 0.2945 - accuracy: 0.9169 - val_loss: 0.0342 - val_accuracy: 0.9901
Epoch 9/20
491/491 [==============================] - 14s 28ms/step - loss: 0.2598 - accuracy: 0.9287 - val_loss: 0.0379 - val_accuracy: 0.9893
Epoch 10/20
491/491 [==============================] - 14s 28ms/step - loss: 0.2434 - accuracy: 0.9343 - val_loss: 0.0380 - val_accuracy: 0.9894
Epoch 11/20
491/491 [==============================] - 14s 28ms/step - loss: 0.2142 - accuracy: 0.9414 - val_loss: 0.0355 - val_accuracy: 0.9897
Epoch 12/20
491/491 [==============================] - 14s 28ms/step - loss: 0.2300 - accuracy: 0.9386 - val_loss: 0.0216 - val_accuracy: 0.9941
Epoch 13/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1942 - accuracy: 0.9485 - val_loss: 0.0263 - val_accuracy: 0.9934
Epoch 14/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1789 - accuracy: 0.9515 - val_loss: 0.0206 - val_accuracy: 0.9949
Epoch 15/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1699 - accuracy: 0.9547 - val_loss: 0.0191 - val_accuracy: 0.9959
Epoch 16/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1627 - accuracy: 0.9559 - val_loss: 0.0170 - val_accuracy: 0.9958
Epoch 17/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1668 - accuracy: 0.9554 - val_loss: 0.0166 - val_accuracy: 0.9952
Epoch 18/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1652 - accuracy: 0.9574 - val_loss: 0.0120 - val_accuracy: 0.9960
Epoch 19/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1517 - accuracy: 0.9608 - val_loss: 0.0154 - val_accuracy: 0.9953
Epoch 20/20
491/491 [==============================] - 14s 28ms/step - loss: 0.1467 - accuracy: 0.9631 - val loss: 0.0206 - val accuracy: 0.9944
```

Training Result

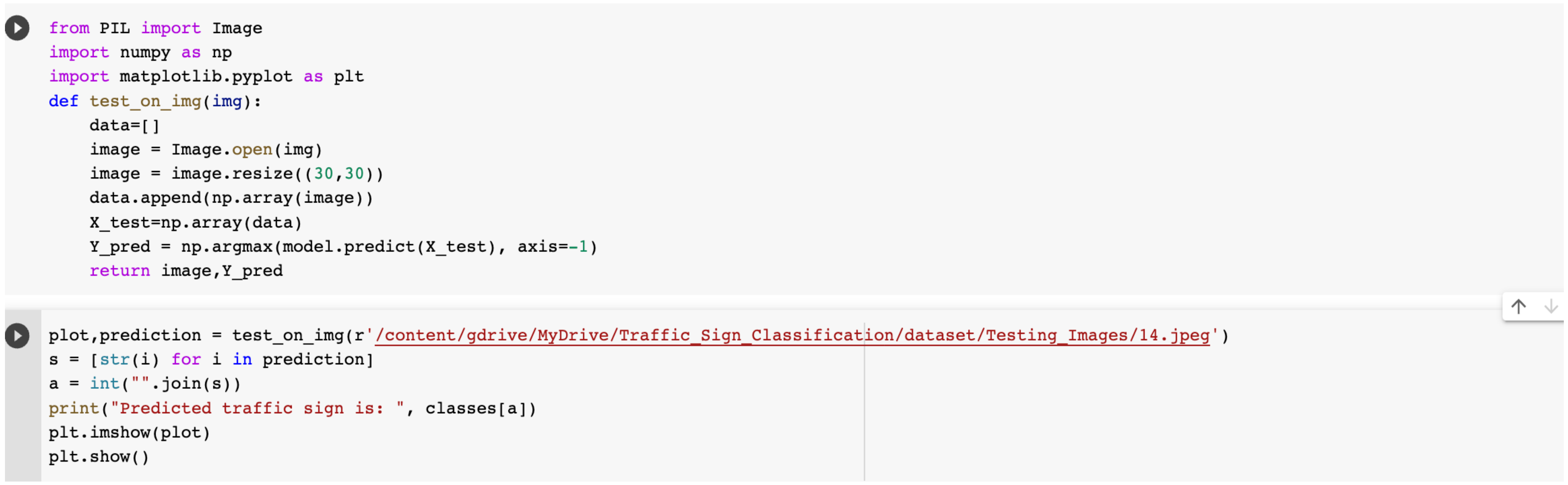


Training Result



Testing - Traffic Sign Classification Model

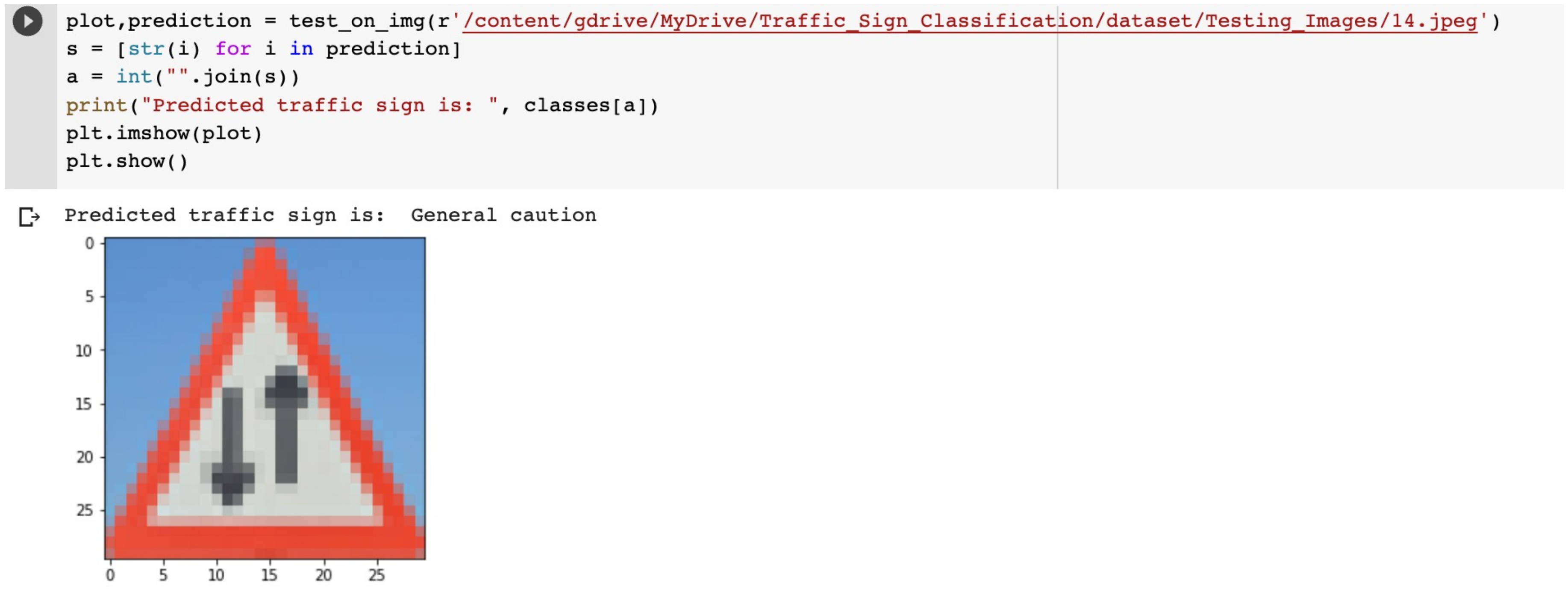
- For testing the model, first load the trained model which was saved earlier. Then run the following cells:



```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
def test_on_img(img):
    data=[]
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
    X_test=np.array(data)
    Y_pred = np.argmax(model.predict(X_test), axis=-1)
    return image,Y_pred
```

```
plot,prediction = test_on_img(r'/content/gdrive/MyDrive/Traffic_Sign_Classification/dataset/Testing_Images/14.jpeg')
s = [str(i) for i in prediction]
a = int("".join(s))
print("Predicted traffic sign is: ", classes[a])
plt.imshow(plot)
plt.show()
```

Testing - Traffic Sign Classification Model



```
plot,prediction = test_on_img(r'/content/gdrive/MyDrive/Traffic_Sign_Classification/dataset/Testing_Images/14.jpeg')
s = [str(i) for i in prediction]
a = int("".join(s))
print("Predicted traffic sign is: ", classes[a])
plt.imshow(plot)
plt.show()
```

```
Predicted traffic sign is:  General caution
```

Thank You