Configuration Manual

MSc Research Project
MSc Data Analytics

Sushaant Kanakaraj
Student ID: 19216360

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sushaant Kanakaraj
Student ID:	X19216360
Programme:	MSc in Data Analytics — **Year:** 2020- 2021
Module:	Research Project
Lecturer:	Mr. Hicham Rifai
Submission Due Date:	16 August 2021
Project Title:	Real-time Motorcyclists Helmet Detection and Vehicle License Plate Extraction using Deep Learning Techniques
Word Count:	1028 — **Page Count:** 13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sushaant Kanakaraj

Date: 16 August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sushaant Kanakaraj
Student ID: x19216360

1 Introduction

This Configuration manual contains the step-by-step information regarding the Storage, Setup, Software and Hardware requirements that are needed to implement the project "Real-time Motorcyclists Helmet Detection and Vehicle License Plate Extraction using Deep Learning Techniques"

2 Hardware and Software Specification

In this section we will discuss Hardware and Software specifications that are used while implementing this project

2.1 Local machine Hardware Specification

The local machine with the below specification is only used to run Jupyter notebook (Anaconda 3) for annotation purposes. Rest of the research is carried out in Google Colab.

Table 1: Hardware Specification

Hardware	Specification
Local machine	Dell Inspiron 5593
RAM	7.77 GB user available
SSD	256GB
CPU	Intel(R) Core (TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
GPU	Nvidia GeForce MX230

Figure 1: Harware specification of local machine

2.2 Google Colab Hardware Specification

Table 2: Google Colab Hardware Specification

Hardware	Specification
RAM	12.69GB
GPU (allocated based on runtime)	Tesla K80 / Tesla T4 (11.4GB)
Disk	107.72GB

2.3 Software Specification

Table 3: Software Hardware Specification

Software	Specification
OS	Windows 10 Home (64-bit)
Programming Language	Python 3.8
IDE	Google Colab, Jupyter Notebook
ML Libraries	PyTorch, TensorFlow, Darknet (NN framework)
Annotation tool	Labellmg
CUDA	11
CuDNN	7.6.5
Open CV	3.2
TensorFlow	2
Other tools	XmltoTxt converter

3 Data Pre-processing and Transformation

The opensource Labellmg is used for annotation of the images in .xml format.

Opening Labellmg

```
In [10]: !pip install --upgrade pyqt5 lxml

Requirement already satisfied: pyqt5 in c:\users\sushaant\anaconda3\lib\site-packages (5.15.4)
Requirement already satisfied: lxml in c:\users\sushaant\anaconda3\lib\site-packages (4.6.3)
Requirement already satisfied: PyQt5-sip<13,>=12.8 in c:\users\sushaant\anaconda3\lib\site-packages (from pyqt5) (12.9.0)
Requirement already satisfied: PyQt5-Qt5>=5.15 in c:\users\sushaant\anaconda3\lib\site-packages (from pyqt5) (5.15.2)

WARNING: You are using pip version 21.1.2; however, version 21.2.4 is available.
You should consider upgrading via the 'c:\users\sushaant\anaconda3\python.exe -m pip install --upgrade pip' command.

In [29]: !pip list

...

In [11]: LABELIMG_PATH = os.path.join('Tensorflow', 'labelimg')

In [12]: if not os.path.exists(LABELIMG_PATH):
             !mkdir {LABELIMG_PATH}
             !git clone https://github.com/tzutalin/labelImg {LABELIMG_PATH}

In [13]: !cd {LABELIMG_PATH} && pyrcc5 -o libs/resources.py resources.qrc

In [*]: !cd {LABELIMG_PATH} && python labelImg.py
```

Figure 2: Using Labellmg tool

The Labellmg tool is opened and the directory in which the images present are chosen.

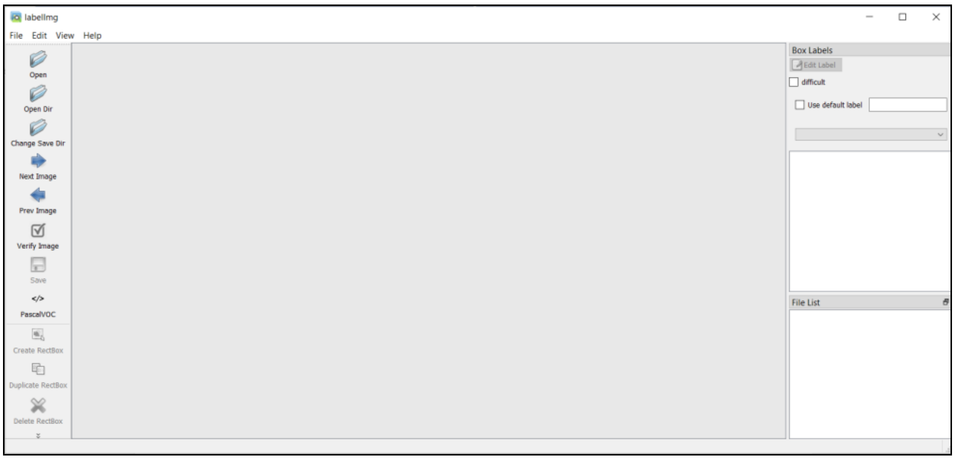

Figure 2: Labellmg tool window

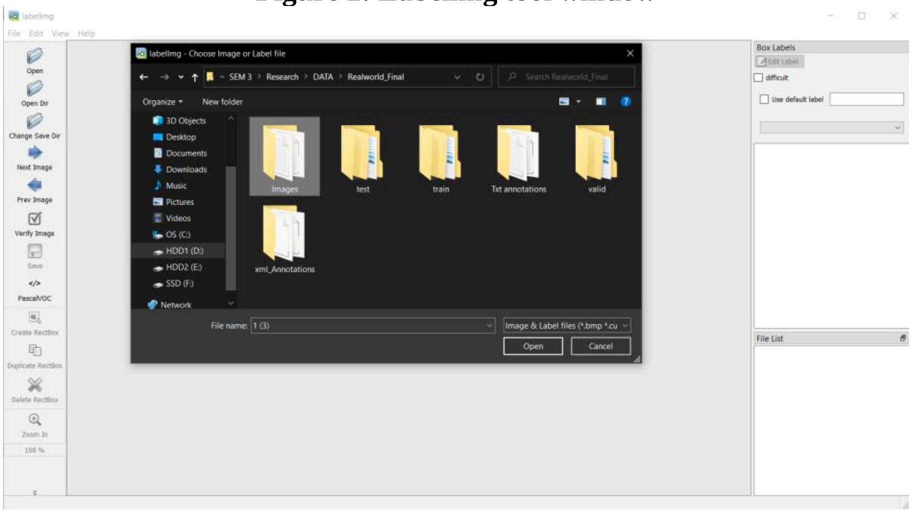

Figure 3: Opening directory in Labellmg tool

The Rectangular box is drawn to mark the area of interest and the class name is saved along with the annotation

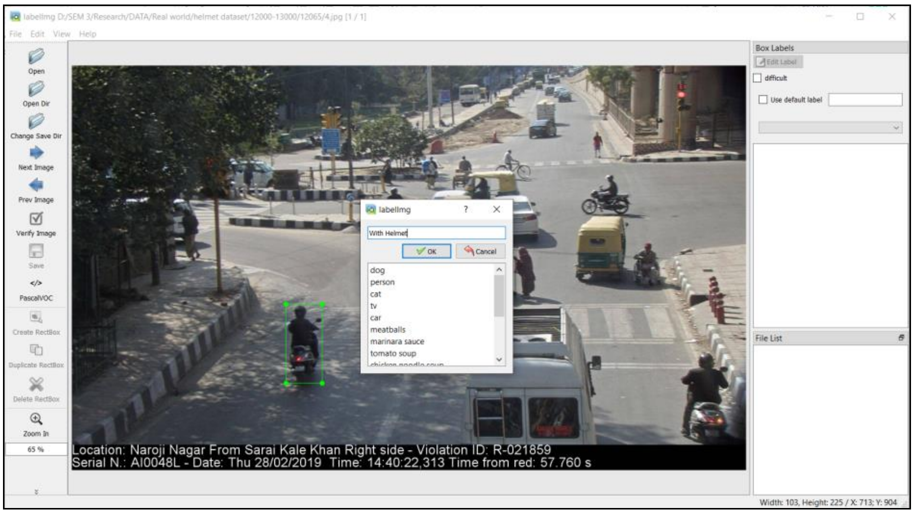

Figure 4: Drawing bounding boxes in Labellmg tool

The Resultant .xml file which contains the coordinates of the bounding boxes


```xml
<?xml version="1.0"?>
<annotation>
    <folder>Realworld_Final</folder>
    <filename>1 (3).jpg</filename>
    <path>D:\SEM 3\RIC\DATA\Realworld_Final\1 (3).jpg</path>
  - <source>
        <database>Unknown</database>
    </source>
  - <size>
        <width>4360</width>
        <height>2372</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
        <name>With Helmet</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
      - <bndbox>
            <xmin>894</xmin>
            <ymin>870</ymin>
            <xmax>1128</xmax>
            <ymax>1460</ymax>
        </bndbox>
    </object>
  - <object>
        <name>Without Helmet</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
      - <bndbox>
            <xmin>1197</xmin>
            <ymin>454</ymin>
            <xmax>1453</xmax>
            <ymax>1123</ymax>
        </bndbox>
    </object>
  - <object>
```


Figure 5: Annotated XML file

The .xml files are converted to .txt files using the opensource XmlToTxt converter

```
In [8]: # Clone the repository of XmlToTxt convertor
        !git clone https://github.com/Isabek/XmlToTxt.git

        Cloning into 'XmlToTxt'...

In [6]: # Check if New path exists
        if os.path.exists("F:/Research/Tensorflow Object Detection/TFODCourse") :
            # Change the current working Directory
            os.chdir("F:/Research/Tensorflow Object Detection/TFODCourse")
        else:
            print("Can't change the Current Working Directory")

In [7]: cwd = os.getcwd()
        cwd

Out[7]: 'F:\\Research\\Tensorflow Object Detection\\TFODCourse'

In [9]: # Enter to the directory XmlToTxt
        %cd XmlToTxt/

        F:\Research\Tensorflow Object Detection\TFODCourse\XmlToTxt

In [10]: # Install the requirements of XmlToTxt convertor
         !pip install declxml==0.9.1

         Requirement already satisfied: declxml==0.9.1 in c:\users\sushaant\anaconda3\lib\site-packages (0.9.1)

         WARNING: You are using pip version 21.1.2; however, version 21.1.3 is available.
         You should consider upgrading via the 'c:\users\sushaant\anaconda3\python.exe -m pip install --upgrade pip' command.

In [13]: !python xmltotxt.py -xml xml -out out
```

Figure 6: Using XML to TXT conversion tool

The resultant .txt files containing corrordinates of the bounding boxes.

```
0 0.757339 0.190135 0.050917 0.218381
0 0.436239 0.219857 0.061009 0.153879
0 0.266399 0.301433 0.056651 0.299325
0 0.185665 0.225759 0.061697 0.266020
1 0.747936 0.470067 0.063761 0.222597
0 0.811468 0.498524 0.058716 0.195194
0 0.940482 0.633642 0.061697 0.198988
```

Figure 7: Annotated TXT file

Google Colab runtime is changed to support GPU Hardware Acceleration

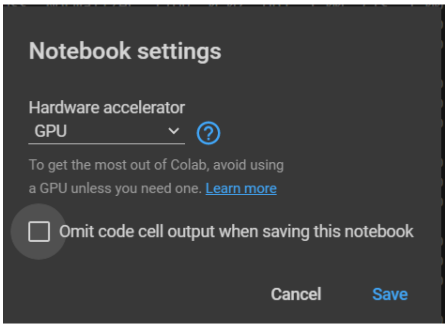

Figure 8: GPU harware acceleration in Google Colab

4 Implementation

4.1 YoloV4-Darknet

The Google drive is mounted to Google Colab. The drive path is optimized and the YoloV4-Darknet model is cloned. The GPU, OpenCV are enabled to take full advantage of GPU Hardware acceleration in Google Colab and the Darknet model building is initiated. The Files are copied from the darknet the Yolov4 folder.

```
[ ] # Mounting Google Drive
    %cd ..
    from google.colab import drive
    drive.mount('/content/gdrive')

    /
    Mounted at /content/gdrive

[ ] # Assigning simple path name
    !ln -s /content/gdrive/My\ Drive/ /mydrive

[ ] # Navigating to YoloV4 folder
    %cd /mydrive/yolov4

    /content/gdrive/My Drive/yolov4

[ ] #Cloning Darknet Yolov4 repository

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

    fatal: destination path 'darknet' already exists and is not an empty directory.

[ ] #Enbling GPU, OpenCV etc, to utilise colab GPU
    %cd darknet/
    !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
    !sed -i 's/GPU=0/GPU=1/' Makefile
    !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
    !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
    !sed -i 's/LIBSO=0/LIBSO=1/' Makefile
[ ] # Building Darknet
    !make

    g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null
    g++ -std=c++11 -shared -std=c++11 -fvisibility=hidden -DLIB_EXPORTS -Iinclude/ -I3rdparty/stb/include -DOPENCV
    In file included from src/yolo_v2_class.cpp:2:0:
```

Figure 9: YoloV4-darknet requiremets setup

```
[ ] # Copying files from Yolo folder to Darknet folder
    %cd data/
    !find -maxdepth 1 -type f -exec rm -rf {} \;
    %cd ..
    %rm -rf cfg/
    %mkdir cfg

    /content/gdrive/My Drive/yolov4/darknet/data
    /content/gdrive/My Drive/yolov4/darknet

[ ] # Extracting data
    !unzip /mydrive/yolov4/obj.zip -d data/
```

Figure 10: Extracting data

List of the names of the test and training data are created as .txt files. The YoloV4 weights are downloaded and the training of the model is initiated.

```
[ ] # create train and test text files
    !ls data/

    labels  obj  obj.data  obj.names  test.txt  train.txt

[ ] # Downloading yoloV4 weighs
    !wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137

[ ] # Training YoloV4-Darknet
    !./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show -map

[ ] !./darknet detector map data/obj.data cfg/yolov4-custom.cfg /mydrive/yolov4/training/yolov4-custom_last.weights -points 0
```

Figure 11: Training YoloV4-darknet

The Graph is generated which shows the Mean Average Precision and the loss. The model is then tested with an input video.

```
[ ] # Plotting mAP and Loss
    def imShow(path):
      import cv2
      import matplotlib.pyplot as plt
      %matplotlib inline
      image = cv2.imread(path)
      height, width = image.shape[:2]
      resized_image = cv2.resize(image,(3*width, 3*height), interpolation = cv2.INTER_CUBIC)
      fig = plt.gcf()
      fig.set_size_inches(18, 10)
      plt.axis("off")
      plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
      plt.show()

[ ] imShow('chart.png')
# Detecting from video
!./darknet detector demo data/obj.data cfg/yolov4-custom.cfg /mydrive/yolov4/training/yolov4-custom_best.weights

-dont_show /mydrive/yolov4/11276118.mp4 -thresh 0.1 -i 0 -out_filename /mydrive/yolov4/11276118_latest.avi
```

Figure 12: Plotting metrics

4.2 YoloV5s

The Google is Drive is mounted to the Google Colab. The paths are navigated to the YoloV5 directory and the model is cloned from the repository.

```
[ ] # Mounting Google drive
    %cd ..
    from google.colab import drive
    drive.mount('/content/gdrive')

    /
    Mounted at /content/gdrive

[ ] # Navigating path
    %cd /content/gdrive/MyDrive
    !pwd

    /content/gdrive/MyDrive
    /content/gdrive/MyDrive

[ ] # cloning YOLOv5 repository
    !git clone https://github.com/ultralytics/yolov5  # clone repo
    %cd yolov5
    !git reset --hard 886f1c03d839575afecb059accf74296fad395b6

    fatal: destination path 'yolov5' already exists and is not an empty directory.
    /content/gdrive/MyDrive/yolov5
    Checking out files: 100% (75/75), done.
    HEAD is now at 886f1c0 DDP after autoanchor reorder (#2421)
```

Figure 13: Setting up requirements for YoloV5s

The dependencies that are needed for running the model are installed and the Input data is extracted to be placed in the test/train and validation folders repectively.

```
[ ] # install dependencies as necessary
    !pip install -qr requirements.txt  # install dependencies (ignore errors)
    import torch

    from IPython.display import Image, clear_output  # to display images
    from utils.google_utils import gdrive_download  # to download models/datasets

    # clear_output()
    print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))

         |████████████████████████████████| 636 kB 5.3 MB/s
    Setup complete. Using torch 1.9.0+cu102 CPU

[ ] # Naivgating path
    %cd /content/gdrive/MyDrive/yolov5
    ! pwd

    /content/gdrive/MyDrive/yolov5
    /content/gdrive/MyDrive/yolov5

[ ] # Extracting input data
    !unzip /content/gdrive/MyDrive/obj_new.zip
```

Figure 14: Installing dependencies and extracting data

The training data with annotations are showcased and later the model is trained with the trianing data.

```
[ ] # Displaying training detections
    print("Training detections")
    Image(filename='/content/gdrive/MyDrive/yolov5/runs/train/yolov5s_results/train_batch2.jpg', width=900)

    # train yolov5s on custom data
    %%time
    %cd /content/gdrive/MyDrive/yolov5
    ! pwd
    !python train.py --img 416 --batch 16 --epochs 3000 --data './data.yaml' --cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results  --cache
```

Figure 15: Installing dependencies and extracting data

Detections are made from the test data and stored in YoloV5 → runs → detect → exp. The model is then tested with the desired video input.

```
[ ] # Detecting Test data

    import glob
    from IPython.display import Image, display

    for imageName in glob.glob('/content/gdrive/MyDrive/yolov5/runs/detect/exp9/*.jpg'): #assuming JPG
        display(Image(filename=imageName))
        print("\n")

[ ] # Detecting from Video
    !python detect.py --weights /content/gdrive/MyDrive/yolov5/runs/train/yolov5s_results/weights/best.pt --img 640 --conf 0.6 --source /content/gdrive/MyDrive/yolov5/video.mp4
```

Figure 16: Detection from Video

4.3 MobileNetV2 FPN lite and Optical Character Recognition

The Google drive is mounted to Colab and paths for the folder are assigned.

```
#!pip uninstall protobuf matplotlib -y
!pip install protobuf matplotlib==3.2
```

```
%tensorflow_version
```

```
Currently selected TF version: 2.x
Available versions:
* 1.x
* 2.x
```

```
import os
```

```
# Mounting google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Choosing TensorFlow2 model and path
CUSTOM_MODEL_NAME = 'my_ssd_mobilenet'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
```

```
#Assigning Paths

paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }
```

```
files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models', CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
```

```
# creating folders
for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            !mkdir -p {path}
        if os.name == 'nt':
            !mkdir {path}
```

Figure 17: Requiremetns for MobileNet V2

The TensorFlow2 model is cloned from the repository and the Object Detection algorithm dependencies are installed.

```
# Cloning Tensorflow model
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}
```

```
Cloning into 'Tensorflow/models'...
remote: Enumerating objects: 59546, done.
remote: Counting objects: 100% (1026/1026), done.
remote: Compressing objects: 100% (381/381), done.
remote: Total 59546 (delta 696), reused 929 (delta 627), pack-reused 58520
Receiving objects: 100% (59546/59546), 573.73 MiB | 34.51 MiB/s, done.
Resolving deltas: 100% (41322/41322), done.
```

```
# Install Tensorflow Object Detection
if os.name=='posix':
    !apt-get install protobuf-compiler
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && cp object_detection/packages/tf2/setup.py . && python -m pip install .

if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\\packages\\tf2\\setup.py setup.py && python setup.py builc
    !cd Tensorflow/models/research/slim && pip install -e .
```

```
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_test.py')
# Verify Installation
!python {VERIFICATION_SCRIPT}
```

Figure 18: Installing TensorFlow Object Detection

The lable map containint the information about the number of classes is created. As the Tensorflow predictions can only be done on Tensorflow record files, which is a binary file created from both the images and annotations.

```
[ ] # Creating Label map
    labels = [{'name':'Licence Plate', 'id':1}]

    with open(files['LABELMAP'], 'w') as f:
        for label in labels:
            f.write('item { \n')
            f.write('\tname:\'{}\'\n'.format(label['name']))
            f.write('\tid:{}\n'.format(label['id']))
            f.write('}\n')
```

```
[ ] # Archiving files for uploading in colab
    ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
    if os.path.exists(ARCHIVE_FILES):
      !tar -zxvf {ARCHIVE_FILES}
```

```
[ ] if not os.path.exists(files['TF_RECORD_SCRIPT']):
        !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}
```

```
[ ] #Create TensorFlow records file for test and train
    !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')}
    !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')}

    Successfully created the TFRecord file: Tensorflow/workspace/annotations/train.record
    Successfully created the TFRecord file: Tensorflow/workspace/annotations/test.record
```

Figure 19: Creating Label map and TensorFlow Records

The model is then trained and detections from the image is preformed.

```
[ ] # Train the model
    TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')

[ ] command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=10000".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])

[ ] print(command)

    python Tensorflow/models/research/object_detection/model_main_tf2.py --model_dir=Tensorflow/workspace/models/my_ssd_mobilenet --pipeline_config_path=Tensorflow/workspace

[ ] !{command}
```

Figure 20: Training the model

```
[ ] # Detect from an Image
    import cv2
    import numpy as np
    from matplotlib import pyplot as plt
    %matplotlib inline

[ ] category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])

[ ] IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', '12171.jpg')

[ ] img = cv2.imread(IMAGE_PATH)
    image_np = np.array(img)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
```

Figure 21: Performing Detections from images

Filering Region of interest and OCR from the images are perfomed

```
import easyocr

detection_threshold = 0.7

image = image_np_with_detections
scores = list(filter(lambda x: x> detection_threshold, detections['detection_scores']))
boxes = detections['detection_boxes'][:len(scores)]
classes = detections['detection_classes'][:len(scores)]

width = image.shape[1]
height = image.shape[0]
```

```
# Applying ROI filtering and OCR
for idx, box in enumerate(boxes):
    print(box)
    roi = box*[height, width, height, width]
    print(roi)
    region = image[int(roi[0]):int(roi[2]),int(roi[1]):int(roi[3])]
    reader = easyocr.Reader(['en'])
    ocr_result = reader.readtext(region)
    print(ocr_result)
    plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))

for result in ocr_result:
    print(np.sum(np.subtract(result[0][2],result[0][1])))
    print(result[1])
```

```
# OCR Filtering

region_threshold = 0.05

def filter_text(region, ocr_result, region_threshold):
    rectangle_size = region.shape[0]*region.shape[1]

    plate = []
    for result in ocr_result:
        length = np.sum(np.subtract(result[0][1], result[0][0]))
        height = np.sum(np.subtract(result[0][2], result[0][1]))

        if length*height / rectangle_size > region_threshold:
            plate.append(result[1])
    return plate

filter_text(region, ocr_result, region_threshold)
```

Figure 22: Extracting ROI and performing OCR