Automatic Number Plate Recognition (Image and Real-time)

Design Project report in compliance with the necessities

of B.Tech degree. (COE)

by

Pavendhan N
(Roll: COE18B041)

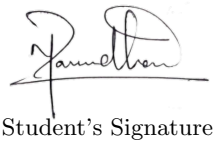
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY,
DESIGN AND MANUFACTURING, KANCHEEPURAM

December 2021

Certificate

I, **Pavendhan N**, with Roll No: **COE18B041** state that the report submitted in the Design Project Report entitled **Automatic Number Plate Recognition (Image and Real-time)** represents the original work done by me in the **CSE Department** at **IIITDM, Kancheepuram** in the year **2021**. I certify that:

- I haven't tampered with any data.
- I did not commit intellectual property plagiarism. I have mentioned the contributions of other people.
- I have recognized all investigations and collaborations.
- I certify that any false declaration shall lead to a serious disciplinary action.
- I am aware that the project report can be checked for academic misconduct.



Date: **7/12/2021** Student's Signature

As a supervisor of the aforementioned work, I confirm that the project details described in this report has been worked out under my supervision and is worthy of consideration for the necessities of the project planning during the period **November 2021** to **December 2021**.

Advisor: **Dr. Ram Prasad Padhy** Advisor's Signature

Abstract

This report primarily covers the algorithm implemented in the design project which automatically detects the Number plate using various existing ML theory and models.
It is capable of recognising the Number plate, both by image and in real-time using tensorflow and ocr.

Acknowledgements

First and foremost, I sincerely thank **IIITDM Kancheepuram**, for enabling me with the opportunity to work under a guide and for the appropriate time period for design project development. Also I would like to thank my guide, **Dr. Ram Prasad Padhy** for supporting and encouraging me to work on this project, under his guidance.

During this period I explored various Models and methods to develop this project.

Additionally, I sincerely thank my parents and friends for being supportive and helpful.

Contents

List of Figures

Abbreviations

ANPR	**A**utomatic **N**umber **P**late **R**ecognition
OCR	**O**ptical **C**haracter **R**ecognition
SSD	**S**ingle **S**hot **D**etection
RPN	**R**egional **P**roposal **N**etwork
CNN	**C**onvolution **N**eural **N**etwork
NLP	**N**atural **L**anguage **P**rocessing
ROI	**R**egion **O**f **I**nterest
API	**A**pplication **P**rogramming **I**nterface
CV	**C**omputer **V**ision
DL	**D**eep **L**earning
ML	**M**achine **L**earning

Chapter 1

An Introduction

1.1 Background and Motivation

Number plates are used for identification that are often present in easily visible positions of on-road vehicles to provide every vehicle its own unique combination of set of characters to identify a particular vehicle. These combinations are available in different formats around the world for each country, and in some cases the combination is different in several parts of a country. The entire purpose is to provide every vehicle an identity.

The purpose of these number plates is to connect back a vehicle which displays the registration to a central database, which can provide additional details of the vehicle, such as model of the car, make of the car, it's engine size and color of the body, owner's address for an instance. Registration number attained is the key to access additional details, which might be required by law for situations like car accidents, stolen vehicle, speeding tickets or breaking traffic rules.

Number plates are a legal thing that can solely be shown by the owner, and it can't be easily created. They are provided by authorities present in each country.

1.2 About ANPR

Automatic Number Plate Recognition, is a CV tech that can identify vehicle's number plates from images captured with machines. Recently, it has become prime need because of three reasons: the ever-growing number of cars, the swift growth of image processing tech and abilities and the enormous amount of real-life applications that the tech provides.

ANPR can be used for various purposes like Real-time no insurance detection, find a stolen or suspected vehicle, Segregate inside and outside vehicles in important places like Banks and Museums, Toll gate, etc.

But, ANPR systems development is not such an easy work, since it faces lots of challenges like environmental and number plate format variations. Environmentally, inconsistent lighting or background largely influence p late r ecognition. I n f act, t hey c an r educe the quality of image of the car and background add unnecessary noise and add burden to the number plate location detection.

These problems seems solvable easily if the recognition is done is passive mode i.e. capturing images and recognising them later on, but it's hard when it is done in Real-time.

1.3 Project Objectives

The primary objectives of the project is to make ANPR system using existing technologies with good accuracy.

- Usage of SSD model
- Usage of pre-trained Tensor-flow Object Detection from TF ZOO
- Leverage GPU over CPU for this system
- Usage of Data set available in Kaglle
- Training Data set using SSD for better results in regard to number plate recognition (Bounding box)
- Usage of Easy OCR for character recognition
- Creating both Image and Real-time Detection modules
- Exporting results in CSV format along with uniquely generated images names of the images recognised exported to local directory

Chapter 2

Methodology

2.1 TensorFlow Object Detection API

TensorFlow Object Detection API is an open source framework assembled using the TensorFlow which makes it lot less challenging to build, train and apply object detection models. It tries to localize and identify multiple countable objects in an image with good accuracy.

There are pre-trained models in this framework which are referred to as Model Zoo from which pre-trained Object Detection models are imported for this project. These are constructed and trained on the COCO data set, the KITTI data set, and the Open Images Data set. These models are be utilized for inference if there are topics of interest in categories which can makes use of them.

FIGURE 2.1: TensorFlow

2.1.1 General Framework of Object Detection

- **Object Localization:** Primarily, a DL model is utilized to produce a big set of bounding boxes which spans the complete the image.

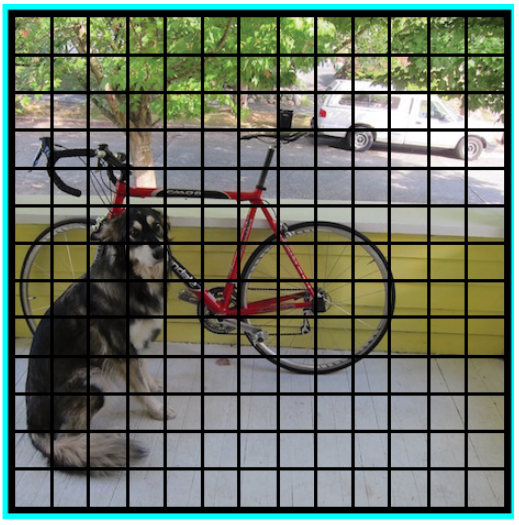


FIGURE 2.2: Object Localization

- **Object Classification:** Now, Visual features present in the image are extracted for each bounding box. These are assessed one by one and is decided if and which objects are visible in each of the bounding boxes using the features.

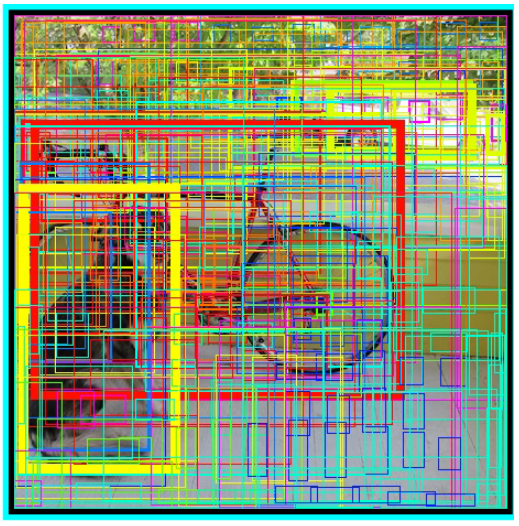


FIGURE 2.3: Object Classification

- **Non-Maximum Suppression:** In the final step, post-processing of the classification happens where overlapping bounding boxes are merged into a single box.

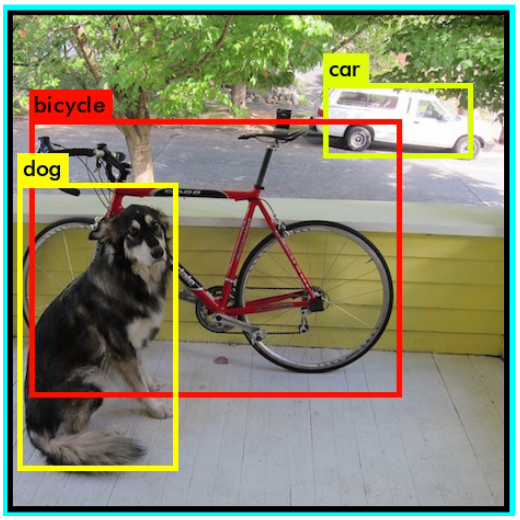


FIGURE 2.4: Suppression

2.1.2 SSD MobileNet Architecture

The **Single Shot Detection (SSD)** architecture is a single convolution network (CN) that has capability to learn and detect bounding box locations in an image and allocate these bounding box locations predicted in a single pass. Thus, SSD perhaps be trained end to end efficiently. SSD MobileNet utilizes a base architecture which comes before various convolution layers which comes after it. It works using feature maps to detect the locations of boxes in the image.

2.1.3 Why SSD?

When SSD is utilized, only one single shot is needed to be taken to detect multiple objects in the image, whilst RPN type of methods, R-CNN series for an instance, which need two shots to be taken to detect the objects, one to generate region proposals, and another to detect the object of each proposal. Hence, SSD architecture is much quicker in contrast to two-shot RPN-based methods.

2.1.4 MobileNet SSD Loss

When the matched bounding boxes are ready, the loss can be computed as:

L = 1/N (L class + L box)

N: Total number of matched boxes. 'L class': soft-max loss of the classification and 'L box': L1 smooth loss representing errors. L1 smooth loss is more robust to outliers than L1 Loss. If N=0, then Loss=0.

2.2 Optical Character Recognition (OCR)

OCR is a process which is utilized by special software to convert text images scanned to electronic text, therefore data in digitized format can be searched, indexed and retrieved. OCR engines are developed and optimized for many real world applications namely, checks, passports, bank statements, insurance documents, **license plates**, etc. Each application requires data set processing which has several hundreds of thousands scanned documents to train and optimize the OCR engines.

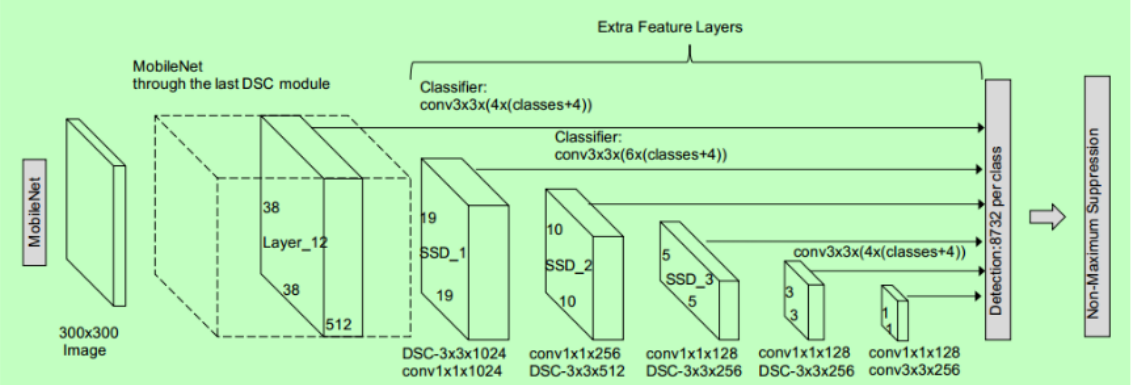


FIGURE 2.5: SSD MobileNet Layered Architecture

2.2.1 OCR Attributes

- **Density of Texts:** On every printed/hand-written paper, text is dense. But, when an image of a street with a single street sign is given, text is sparse.

- **Structure of Texts:** Texts on a paper is structured, most probably in strict rows, while texts in the wild may be scattered everywhere and in random rotations.

- **Fonts:** Printed fonts are easier to read and understand, since they are better structured than the less structured and noisy hand-written characters.

- **Character type:** Text come in different languages which are very different from each other.

- **Location:** Some text are present in center, while some text are be located in random locations in the image.

FIGURE 2.6: OCR Example

2.2.2 EasyOCR

EasyOCR is actually a python package that holds PyTorch as a back-end handler, also leveraging GPU which allows CV developers to efficiently make usage of OCR. In the case of OCR, EasyOCR is definitely the most straightforward method to apply OCR which utilizes high end deep learning library (PyTorch), supporting it in the back-end, that makes its accuracy more credible.
EasyOCR supports 70+ languages for detection purposes.

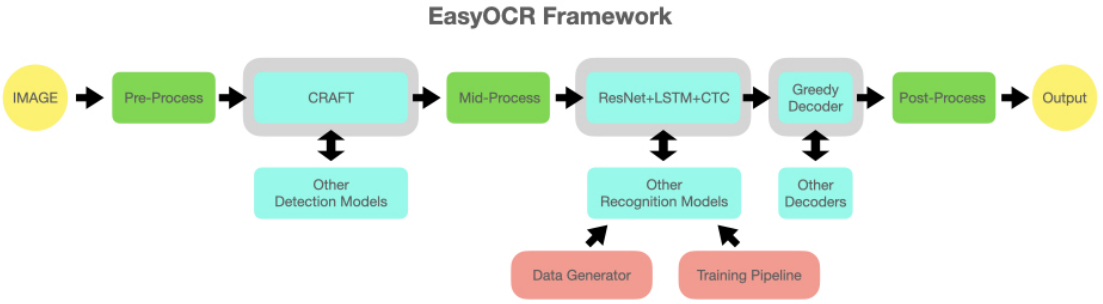


FIGURE 2.7: EasyOCR Framework

2.3 OpenCV

OpenCV (Open Source CV Library) is an open-source library which includes hundreds of CV algorithms especially aimed at real-time CV. The report uses OpenCV 2.x API, which is essentially a C++ API, as compared to the C-based OpenCV 1.x API. OpenCV has a modular structure, which signifies the package consists of various shared or static libraries.

Chapter 3

Project Work

The entire Project is completed under a virtual environment using **venv** by python and using **venv kernel** instead of default python kernel.

3.1 Setup Paths

First and foremost in this project, several requirements are to be achieved so that later there is no error and the process goes on without any hindrance. This is done using OS library.

- **Macros:** Several variables are defined which will be used later in the code, which carries names of some files, model name and URL.

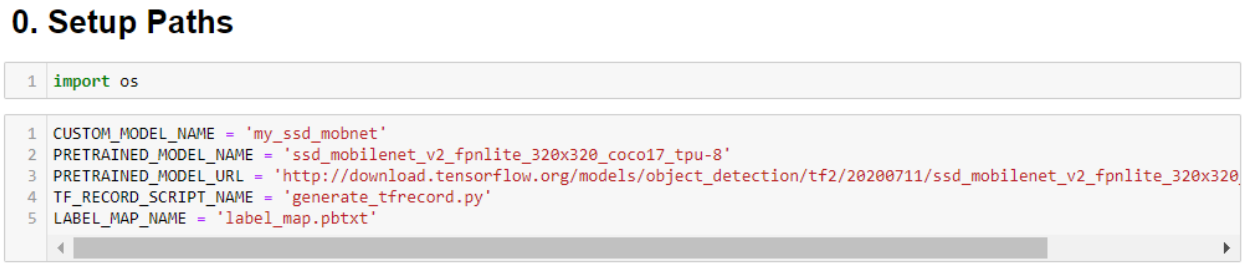


FIGURE 3.1: Macros

- **Paths:** Numerous paths are defined under the list paths w.r.t. the TensorFlow model which will be imported later on.

```
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')
 }
```

FIGURE 3.2: Paths

- **Files:** Three files required are defined with relative paths.

```
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)
}
```

FIGURE 3.3: Files

- **Path Creation:** The path values defined in the list path are created in the root folder w.r.t. OS used.

```
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 3.4: Path Creation

3.2 Download and Install TensorFlow models

TensorFlow provides various pre trained models under TensorFlow Model ZOO which cover broad range of categories.

3.2.1 Import TensorFlow Models from TensorFlow Model Z00

The models are imported using git clone from the TensorFlow ZOO GitHub repository.

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

FIGURE 3.5: Import

3.2.2 Install TensorFlow Object Detection

Now that all models are imported into root folder, object detection model is installed in the next step.

```
# 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/

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\\packag
    !cd Tensorflow/models/research/slim && pip install -e .
```

FIGURE 3.6: Installation TFOD

After this, all the necessary libraries are installed by using the **Verification Script** until **OK** appears in the output.

```
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_te
# Verify Installation
!python {VERIFICATION_SCRIPT}
```

FIGURE 3.7: Verification

```
Ran 24 tests in 66.637s

OK (skipped=1)
```

FIGURE 3.8: Status: OK

3.2.3 Strict Requirements in terms of Version

If these requirments are not satisfied, then the dedicated GPU cannot be used.

- **tensorflow:** 2.4.1
- **tensorflow-gpu:** 2.4.1
- **CUDA:** 11.0
- **cudnn: 8.0.5**

3.3 Training the Data Set

The data set used for this project is downloaded from Kaggle which provides **432** car images and its information in XML format.

- **TRAIN:** 410 images
- **TEST:** 411-432 i.e. 32 images

3.3.1 The XML file

The XML files corresponding to each car images provides the following details:

- Folder name
- File name
- **Size:** Width, Height, Depth
- Segmentation
- **Object:** Name (Label), Pose, Truncation, Occlusion, Difficulty
- **Bounding Box:** xmin ymin xmax ymax (This denotes the box location of the plate in the image)

```
<annotation>
    <folder>images</folder>
    <filename>Cars411.png</filename>
    <size>
        <width>400</width>
        <height>268</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>licence</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>160</xmin>
            <ymin>153</ymin>
            <xmax>249</xmax>
            <ymax>197</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 3.9: XML

3.3.2 Label Map

Since the only Object that is going to be detected by SSD is Number Plate, only one label is required with matches with Object — Name in the XML file. Hence a Label map is created which is required for training.

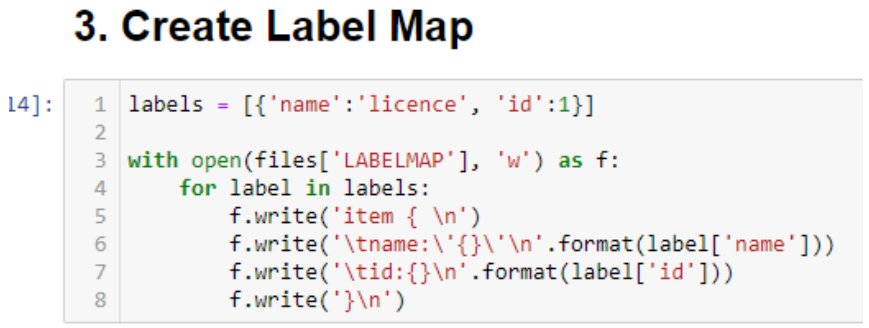


FIGURE 3.10: Label Map Code

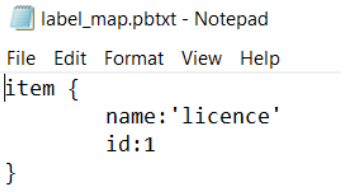


FIGURE 3.11: Label Map

3.3.3 Create TF Records

TensorFlow Object Detection requires the details present in the images and the annotations in the XML file in the single entity called TF record. It is created separately for train and test as **.RECORD** file type.

FIGURE 3.12: TF Record creation

3.3.4 Transfer Learning: Copy and Update the Model Config

Transfer learning is a ML algorithm where a model is developed for a work, is re-utilized as the beginning point for a model on a second work.
It is a well-known method in DL where pre-trained models are re-utilized as the beginning point on CV algorithms and NLP tasks, provided the large compute and time resources needed to develop neural network model.

The pretrained model config file available in the TF ZOO is copied to required folder and several statements which point to annotations recorded previously in record files are modified to point to the newly created ones for training using the macros, paths and files defined in the beginning.

5. Copy Model Config to Training Folder

```
if os.name =='posix':
    !cp {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKP
if os.name == 'nt':
    !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHEC
```

6. Update Config For Transfer Learning

```
config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
```

```
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
```

```
pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'che
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train.rec
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'test.re
```

```
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)
```

FIGURE 3.13: Model Config

3.3.5 Train the model

After meeting the requirements explained above, training command line is generated to run the command externally in command prompt in the root directory, so that the results / Checkpoints will be saved and will be usable for future detection.

Number of Steps: 10000

Time taken: 4 hours

7. Train the model (Generate command to run externally to save the output)

```
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['CHECKP
```

```
print(command)
```

```
python Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ssd_mobnet --pi
peline_config_path=Tensorflow\workspace\models\my_ssd_mobnet\pipeline.config --num_train_steps=10000
```

FIGURE 3.14: Train Command line

3.4 Split GPU RAM

For this project, TensorFlow, EasyOCR and Real-time detection leverages dedicated GPU, hence limiting RAM for TensorFlow enables other two to run without running out of memory.

Memory Limited: 1024MB

```
# Prevent GPU complete consumption
# gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_virtual_device_configuration(
            gpus[0], [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=1024)])
    except RuntimeError as e:
        print(e)
```

FIGURE 3.15: GPU Limit

3.5 Load the Trained Model

The training creates several checkpoints throughout the training process. The latest checkpoint (here 11th chkpnt) is loaded for detection later on. As soon it is loaded, TensorFlow makes use of the allotted GPU RAM stated above.

8. Load Trained Model From Checkpoint

```
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-11')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

FIGURE 3.16: Load Trained Model

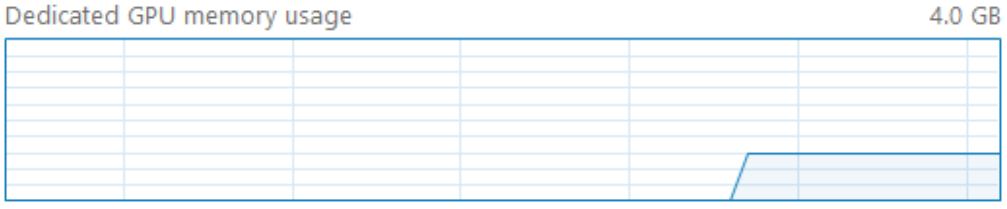


FIGURE 3.17: GPU Used

3.6 Detection from an Image

- First, category index is created using the label map
- New Image path is defined to traverse inside the **test** folder.
- The image mentioned in path is opened by openCV, and converted to tensor. Then the detection algorithm works, where it detect the number plate and marks it with a green box on the image and is displayed using matplotlib.

- The detection id done stored in int64 format as an numpy array.

- **Detections** list consists of Detection boxes, detection boxes and detection scores. After attaining a minimum threshold score of **0.8**, the results are displayed.

9. Detection from an Image

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

```
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'test.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'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.8,
            agnostic_mode=False)

plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```

FIGURE 3.18: Detection algorithm

3.7 Apply EasyOCR

Now that the number plate is recognised using the trained model, next step in process is the recognise the characters on the number plate. For this, EasyOCR is used which fast and straight-forward. The process is as follows:

- First, **Detection Threshold** is defined, which filters results from detection scores present is detections list above certain point. (here 0.7)
- W.r.t. to the new filtered scores, detection boxes and classes are filtered as well.
- Next, Image width and height are extracted, which is used to calculate the **ROI:** Region of Interest.
- The ROI is extracted from the images (the number plate) to remove unnecessary information and OCR is applied on it
- One thing to note is some number plates have information other required numbers in small fonts. To filter that out, **Region Threshold** is defined.
- In filtering algorithm, the area of ROI is calculated and looped through the **OCR result**. The part of result which crosses the region threshold is considered as the Data on the number Plate.

OCR Filtering

```
1  def filter_text(region, ocr_result, region_threshold):
2      rectangle_size = region.shape[0]*region.shape[1]
3
4      plate = []
5      for result in ocr_result:
6          length = np.sum(np.subtract(result[0][1], result[0][0]))
7          height = np.sum(np.subtract(result[0][2], result[0][1]))
8
9          if length*height / rectangle_size > region_threshold:
10             plate.append(result[1])
11     return plate
```

FIGURE 3.19: Filtering algorithm

Apply OCR

```
detection_threshold = 0.7
region_threshold = 0.45
```

```
def apply_ocr(image, detections, detection_threshold, region_threshold):

    # Scores, boxes and classes above threhold
    scores = list(filter(lambda x: x> detection_threshold, detections['detection_scores']))
    boxes = detections['detection_boxes'][:len(scores)]
    classes = detections['detection_classes'][:len(scores)]

    # Full image dimensions
    width = image.shape[1]
    height = image.shape[0]

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

        text = filter_text(region, ocr_result, region_threshold)

        plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
        plt.show()
        print(text)
        return text, region
```

```
text, region = apply_ocr(image_np_with_detections, detections, detection_threshold, region_threshold)
```

FIGURE 3.20: OCR Application

3.8 Real Time Recognition

In Real time recognition, due to Low-end GPU, the detection faces problems like heavy lag which gives multiple results, some of which have good accuracy. Here OpenCV is used and while the camera is turned on / functioning, the **Frame** is read as numpy array and the process explained above is followed.

```
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    ret, frame = cap.read()
    image_np = np.array(frame)

    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'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.8,
                agnostic_mode=False)

    try:
        text, region = apply_ocr(image_np_with_detections, detections, detection_threshold, region_threshold)
        save_results(text, region, 'realtimeresults.csv', 'Detection_Images')
    except:
        pass

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

FIGURE 3.21: Real Time Recognition

3.9 Export Results

In the Final step, the results acquired in the algorithm **ROI-Image** and **Characters Recognised** are stored in root directory.

- Using **UUID** library from python, an unique image name is generated for image detected and saved with that name.
- The Images detected using both image and real time recognition, are stored in **Detection_Images** folder.

- Using **CSV** library from python, the filename and the corresponding result is exported in the csv format.
- Results from image Recognition are stored in **detection_results.csv**
- Results from Real Time Recognition are stored in **realtimeresults.csv**

11. Save Results

```
# '{}.jpg'.format(uuid.uuid1())
```

```
def save_results(text, region, csv_filename, folder_path):
    img_name = '{}.jpg'.format(uuid.uuid1())

    cv2.imwrite(os.path.join(folder_path, img_name), region)

    with open(csv_filename, mode='a', newline='') as f:
        csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        csv_writer.writerow([img_name, text])
```

```
# region
```

```
save_results(text, region, 'detection_results.csv', 'Detection_Images')
```

FIGURE 3.22: Image Recognition: Save

```
save_results(text, region, 'realtimeresults.csv', 'Detection_Images')
```

FIGURE 3.23: Real Time Recognition: Save

Chapter 4

Results

This chapter covers all the Results / Outputs from the ANPR project for both image ad real time recognition.

4.1 GPU Consumption

This project has leveraged the dedicated GPU for various process and the consumption has been controlled in the process.

- **Only TensorFlow:**

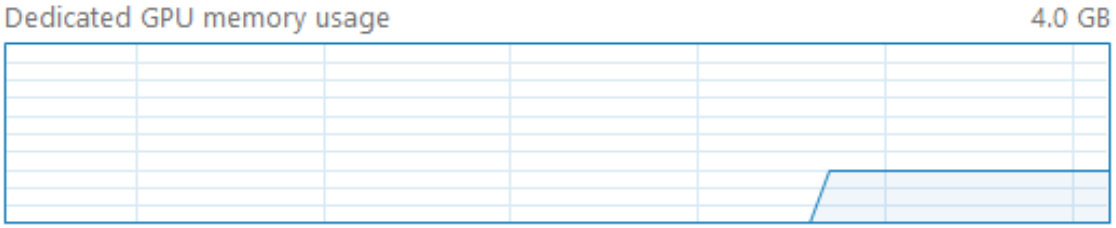


Figure 4.1: TF

- **TensorFlow + OCR**

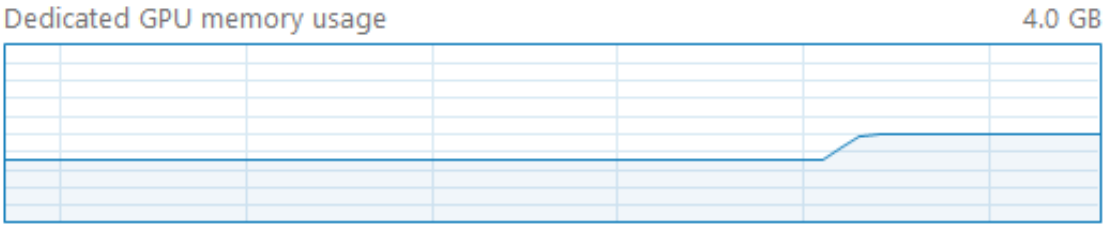


Figure 4.2: TF + OCR

- **TensorFlow + OpenCV**

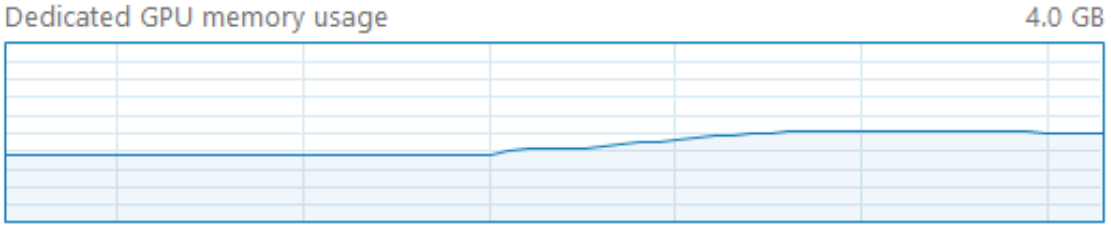


Figure 4.3: TF + CV

- **TensorFlow + OCR + OpenCV**

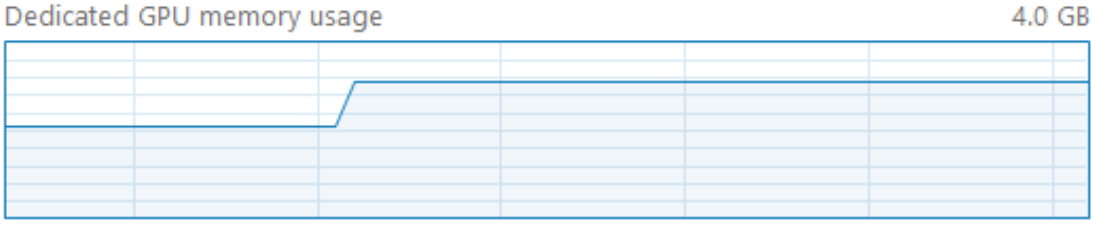


Figure 4.4: TF + OCR + CV

4.2 Detetion from an Image

This section displays output from the image detection explained in previous chapter.

- **Number Plate Recognition:** Using SSD model.

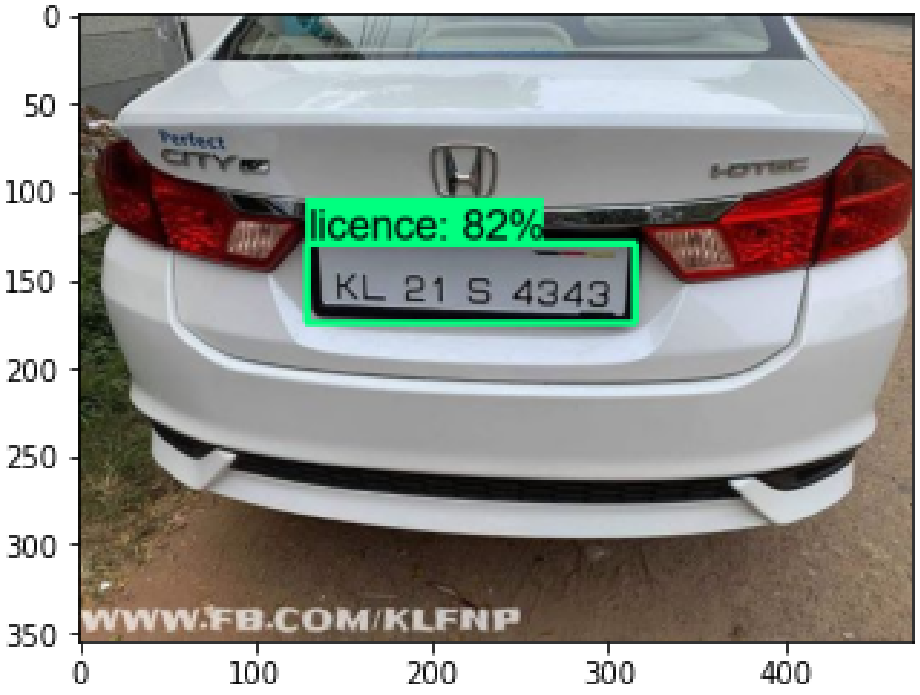


FIGURE 4.5: Number Plate Detection

- **Region of Interest and Character Detection:**

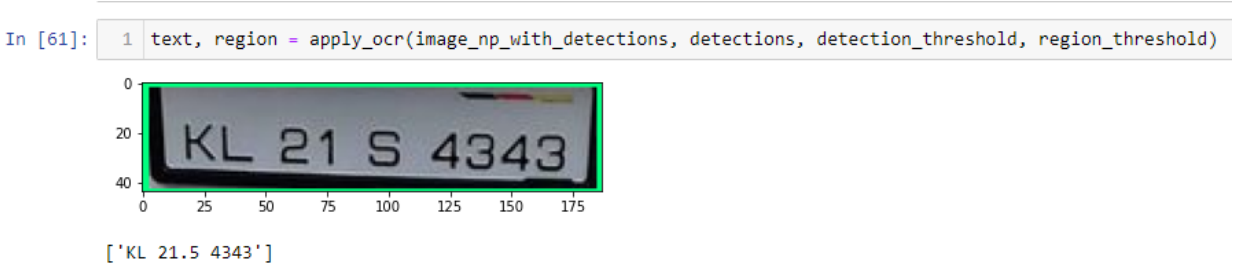


FIGURE 4.6: ROI + OCR

- **Image saved in root directory:**

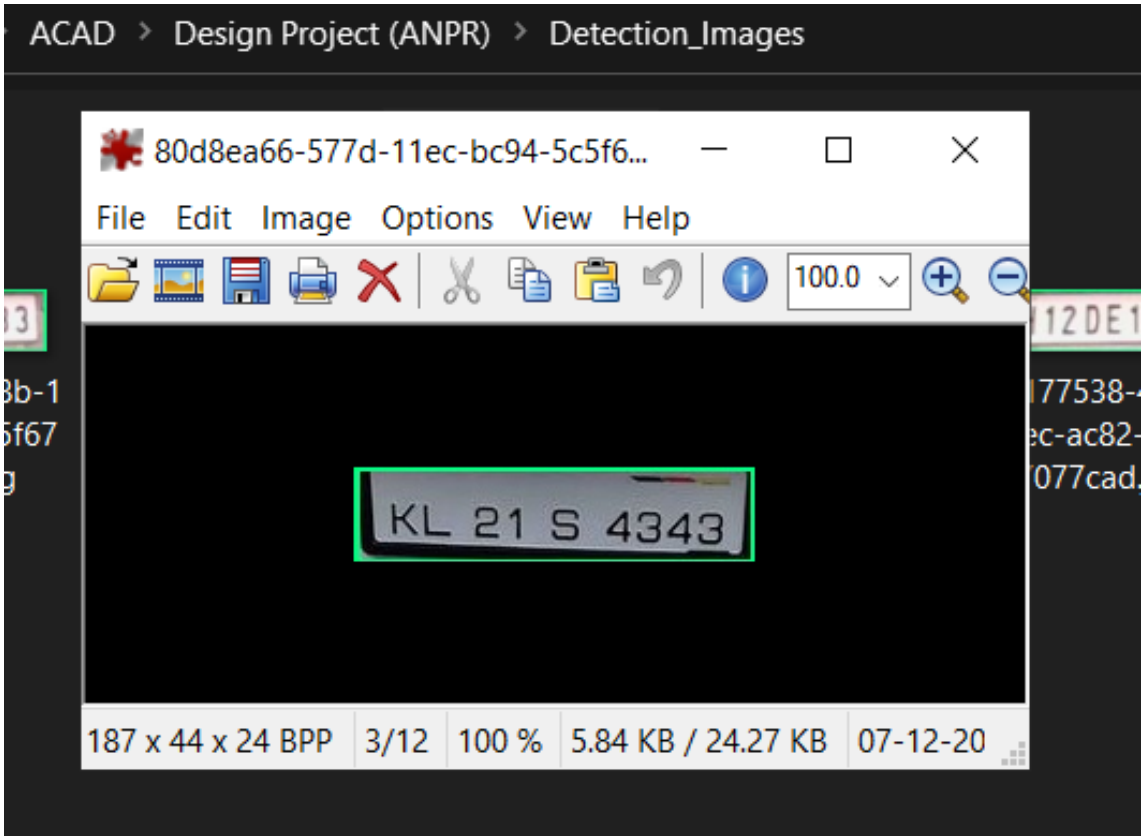


FIGURE 4.7: Save

- **Number Plate Recorded in CSV:**

A	B	C
77ef19b1-4b8a-11ec-af90-5c5f67077cad.jpg	['SH335c0']	
87f7fb13-5516-11ec-ac59-5c5f67077cad.jpg	['ISKIPGAs']	
80d8ea66-577d-11ec-bc94-5c5f67077cad.jpg	['KL 21.5 4343']	

FIGURE 4.8: Record

4.3 Real Time Recognition

This section displays output from the Real Time Recognition explained in previous chapter.

- **Number Plate Recognition:** Using SSD model and OpenCV

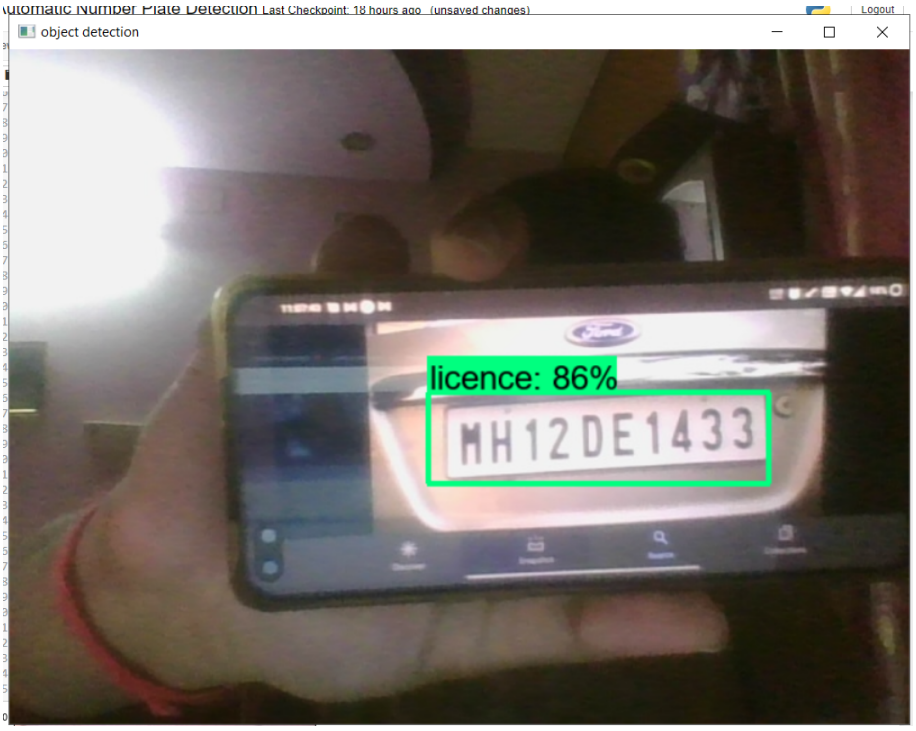


FIGURE 4.9: Number Plate Detection

- **Region of Interest and Character Detection:**


```
['mHI2 DE1433']
```

FIGURE 4.10: ROI + OCR

- **Image saved in root directory:**

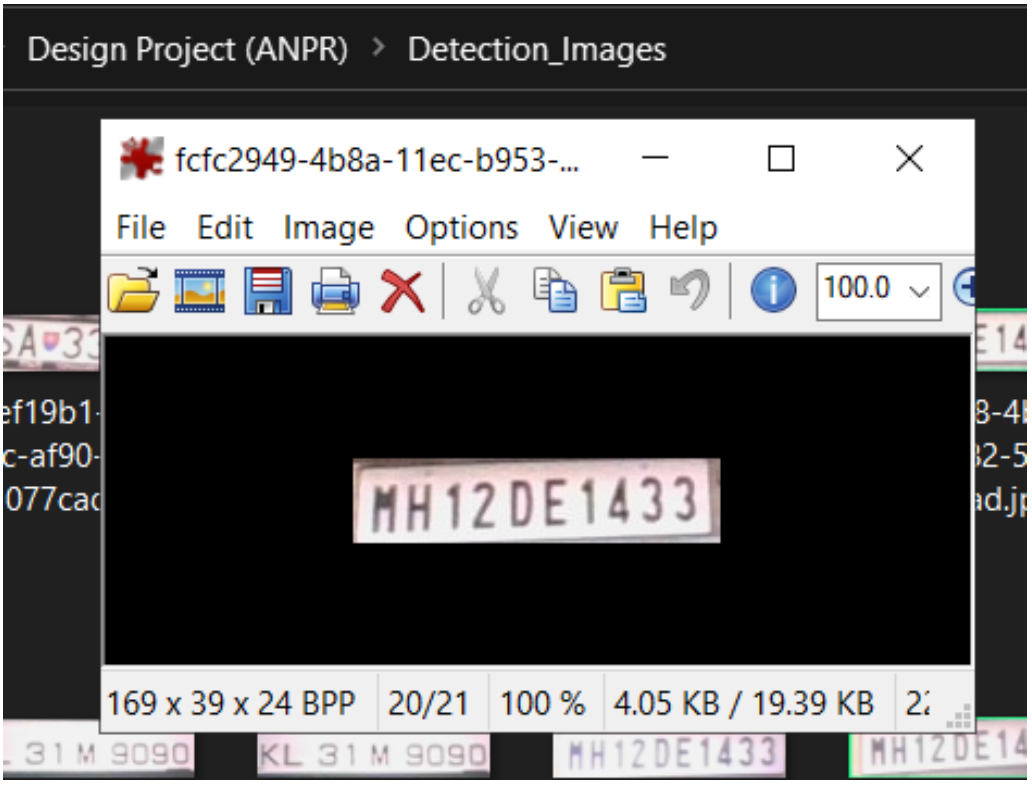


FIGURE 4.11: Save

- **Number Plate Recorded in CSV:**

fcfc2949-4b8a-11ec-b953-5c5f67077cad.jpg	['MH12 DE1433']

FIGURE 4.12: Record

Chapter 5

Conclusions and Future Sights

After Successful completion of this project, I have learnt various things especially the coding section. With the newly gained Experience and knowledge

- I would like to develop this project further more in terms of **Accuracy** and **Capability to work on Variety of Data.**
- I would like to fine tune this project further more, so that this can be used as part of bigger projects.

Bibliography

[1] “Tensorflow api.” [Online]. Available: https://www.tensorflow.org/api_docs

[2] “Tensorflow object detection api.” [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection

[3] “Tensorflow model zoo.” [Online]. Available: https://github.com/tensorflow/models

[4] “Stackoverflow.” [Online]. Available: https://stackoverflow.com/

[5] “Object detection using tf and ssd.” [Online]. Available: https://www.analyticsvidhya.com/blog/2020/04/build-your-own-object-detection-model-using-tensorflow-api/