

Summer Industrial Internship Training Report

On

**LEARNING VARIOUS DETECTION ALGORITHMS IN OPEN-CV AND
TRAINING CUSTOM OBJECT DETECTOR FOR DETECTING HELIPORT “H”
SIGNS.**

At



DRDO- Young Scientist Laboratory-Asymmetric Technologies (DYSAL-AT)

(DEFENCE RESEARCH AND DEVELOPMENT ORGANISATION)

Velgalkunta , Telangana- 500043.

in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY (B.Tech)

IN

COMPUTER SCIENCE AND ENGINEERING



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

VELLORE INSTITUTE OF TECHNOLOGY ,VELLORE.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

MALLELA.SRI SAI MANIKANTA VISHNU SHASANK

18BCE2131

DECLARATION BY THE CANDIDATE

I hereby declare that the Industrial Internship report entitled
“LEARNING VARIOUS DETECTION ALGORITHMS IN OPEN-CV AND TRAINING CUSTOM OBJECT DETECTOR FOR DETECTING HELIPORT “H” SIGNS” submitted by me to Vellore Institute of Technology, Vellore in partial fulfillment of the requirement for the award of the degree of
Bachelor of Technology in Computer Science and Engineering is a record of bonafide industrial training undertaken by me under the supervision of **Mr.Saumyaranjan Mohanty, (SCIENTIST-C),DRDO-YSL-AT**. I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

NAME: M.SRI SAI MANIKANTA VISHNU SHASANK

REG.NO : 18BCE2131

(SCHOOL OF COMPUTER SCIENCE AND ENGINEERING)



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE :

दूरभाष - ०८४५८-२७३१०१
Tel No: 08458-273104
फैक्स- ०८४५८-२७३१००
Fax No: 08458- 273100
ई-मेल: director@dysl-at.drdo.in
Email: director@dysl-at.drdo.in



भारत सरकार, रक्षा मंत्रालय
Govt. of India, Ministry of Defence
रक्षा अनुसंधान एवं विकास संगठन
Defence Research & Development Orgn.
असममित प्रौद्योगिकी
Asymmetric Technologies
डीआरडीओ युवा वैज्ञानिक प्रयोगशाला
DRDO Young Scientists' Laboratories
ऑरेंज टेस्ट फैसिलिटी, डुंडीगल,
हैदराबाद - ५०००४३
Orange Test Facility, RCI, Dundigal,
Hyderabad-500043

Date: Jan-01-2021

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr. MSRI SAI MANIKANTA VISHNU SHASANK** , pursuing **B.Tech. in Computer Science Engineering in Vellore Institute of Technology at Vellore** has successfully completed his Summer internship for the project, title **Learning various Object-detection algorithms in Open-cv and Training Custom object detector for detecting heliport 'H' signs** , under the guidance of **Mr. Saumyaranjan Mohanty, Scientist** of this organization for the period from **01-December-2020 to 30-December-2020**. Duration of **4 weeks**.

During the training period his/her conduct at DYSL-AT was good.



(P .Shiva Prasad)

Director

P SHIVA PRASAD
Director Asymmetric Technologies (DYSL-AT)
DRDO Young Scientists' Laboratory
DRDO, Ministry of Defence, Govt. Of India
C/o Orange Facility, Dundigal, Hyderabad- 500 043

ACKNOWLEDGEMENT

I record my sincere thanks to the Director Mr.. P.Shiva Prasad , DRDO-Young Scientist Laboratory-Asymmetric Technologies for providing me an excellent opportunity to undergo training in his esteemed organization through which I can gain an exposure to Research and Development environment and getting acquainted with Object-detection Technologies.

I wish to express my deep sense of gratitude and obligation to , Mr Mohanty, Scientist-C, for his valuable guidance. I express my thanks to all members of DYSL-AT for providing me support and their constant encouragement during training.

I would like to thank all young scientists for allowing me access various technologies and valuable materials.

I would also express my gratitude to VIT University and all of my faculties till now who have guided me through my education to make me interested enough to take up object-detection development as a part of machine learning as a skill.

Last but not the least, I have nothing but the utmost respect and gratitude for my parents who have supported me immensely.

M.SRI SAI MANIKANTA VISHNU SHASANK

ORGANISATION PROFILE



DRDO Young Scientist Laboratories (DYSLs) are five specialised research laboratories located in five different cities of India, inaugurated by the Prime Minister of India on 2 January 2020. Each laboratory deals with a focused area of science - artificial intelligence, quantum technologies, cognitive technologies, asymmetric technologies and smart materials. The labs are located in Bengaluru, Mumbai, Chennai, Kolkata and Hyderabad.

Two of the directors chosen are scientists Parvathaneni Shiva Prasad of the Research Centre Imarat (RCI) and Ramakrishnan Raghavan of the Defence Metallurgical Research Laboratory (DMRL). They will function as completely independent directors of their respective labs. The directors have been finalised by a committee chaired by Principal Scientific Advisor K. Vijayaraghavan. Bengaluru, Mumbai, Chennai, Kolkata and Hyderabad.

DRDO is the R&D wing of Ministry of Defence, Govt of India, with a vision to empower India with cutting-edge defence technologies and a mission to achieve self-reliance in critical defence technologies and systems, while equipping our armed forces with state-of-the-art weapon systems and equipment in accordance with requirements laid down by the three Services. DRDO's pursuit of self-reliance and successful indigenous development and production of strategic systems and platforms such as Agni and Prithvi series of missiles; light combat aircraft, Tejas; multi-barrel rocket launcher, Pinaka; air defence system, Akash; a wide range of radars and electronic warfare systems; etc., have given quantum jump to India's military might, generating effective deterrence and providing crucial leverage.

Mission

To develop state of art advanced Asymmetric technologies.

Vision

Emphasis on incorporating Asymmetric technologies into existing System like classical radar, radio, surveillance systems to make them more agile to the environment.

TABLE OF CONTENTS :

CHAPTER NO.	TITLE	PAGE NO.	WEEK BREAKDOWN
	ORGANISATION PROFILE	06	
	LIST OF FIGURES	08	
1.	INTRODUCTION 1.OPEN CV BASICS	09	WEEK-1
	1.1 OPEN-CV 1.2 INSTALLATION 1.3 GUI OPENCV FEATURES 1.3.1 PROCESSING WITH IMAGES 1.3.2 PROCESSING WITH VIDEOS 1.3.3 PROCESSING WITH SHAPES 1.3.4 IMAGE TRANSLATIONS 1.3.5 IMAGE RESIZING 1.3.6 ANALYZING IMAGE HISTOGRAMS 1.3.7 IMAGE ROTATION BY ANY ANGLE		
2.	VARIOUS DETECTION ALGORITHMS IN OPEN-CV 2.1 HAAR- CASCADE FACE DETECTION 2.2 CANNY EDGE DETECTION 2.3 HARRIS-CORNER DETECTION 2.4 MEAN-SHIFT AND CAM - SHIFT FOR VIDEO OBJECT TRACKING	18	WEEK-2
3.	HELIPAD “H” SIGN DETECTION USING TENSORFLOW AND OPEN CV. 3.1 TENSORFLOW INSTALLATION 3.2 TRAINING CUSTOM DETECTOR 3.3 EXPORTING MODEL 3.4 USING OPENCV AND SAVED TF2 MODEL FOR DETECTION	31	WEEK-3,4

4.	DAY TO DAY BREAK-DOWN ACTIVITIES	46	
5.	COMPARISON OF COMPETENCY LEVELS BEFORE AND AFTER INTERNSHIP (SELF EVALUATION) :	51	
6.	CONCLUSION	52	
8.	APPENDICES	53	

LIST OF FIGURES :

1.1.1-READING IMAGE OUTPUT

1.1.2- PLOTTING IMAGE OUTPUT

1.1.3- VIDEO READING THROUGH WEBCAM

2.1.1- FACE DETECTION OUTPUT USING HAAR CASCADE

3.1.1 -OUTPUT TENSORFLOW INSTALLATION

3.1.2 -TENSORFLOW MODEL DIRECTORY SNAPSHOT

3.1.3- CREATION OF TENSORFLOW WORKSPACE SNAPSHOT

3.1.4-LABELIMG DIRECTORY

3.1.5 -LABELIMG OPENING WINDOW FOR ANNOTATION

3.1.6-DIRECTORY OF IMAGES COLLECTED FOR DATASET PREPARATION

3.1.7- LABEL MAP TEXT FILE

3.1.8 -ANNOTATION DIRECTORY HAVING TF RECORDS

3.1.9 – FOLDER OF PRE-TRAINED DOWNLOADED MODEL

3.1.10 – TENSORBOARD WINDOW OF TRAINING H PAD SIGNS ON PRE TRAINED MODEL

3.1.11- FOLDER OF SAVED MODEL AFTER EXPORTING

3.1.12 – PYTHON FILE CODE FOR RUNNING SAVED MODEL ON DETECTING IMAGES

3.1.13 – SCREENSHOT OF INFERENCE RUNNING

3.1.14- RESULT DETECTION SCREENSHOT.

INTRODUCTION:

1. OPEN-CV BASICS :

1.1 OpenCV-Python

Python is a general purpose programming language started by **Guido van Rossum**, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working in background) and second, it is very easy to code in Python. This is how OpenCV-Python works, it is a Python wrapper around original C++ implementation.

And the support of Numpy makes the task more easier. **Numpy** is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.

So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems.

1.2 INSTALLATION :

Installing OpenCV from prebuilt binaries

1. Below Python packages are to be downloaded and installed to their default locations.

- 1.1. [Python-2.7.x](#).

- 1.2. [Numpy](#).

- 1.3. [Matplotlib](#) (*Matplotlib is optional, but recommended since we use it a lot in our tutorials*).

2. Install all packages into their default locations. Python will be installed to **C:/Python27/**.
3. After installation, open Python IDLE. Enter `import numpy` and make sure Numpy is working fine.
4. Download latest OpenCV release from [sourceforge site](http://sourceforge.net) and double-click to extract it.
7. Goto **opencv/build/python/2.7** folder.
8. Copy **cv2.pyd** to **C:/Python27/lib/site-packages**.
9. Open Python IDLE and type following codes in Python terminal.
10. `>>> import cv2`
11. `>>> print cv2.__version__`

If the results are printed out without any errors, congratulations !!! You have installed OpenCV-Python successfully.

1.3 GUI -FEAUTERS IN OPEN-CV (BASICS) :

1. PROCESSING WITH IMAGES

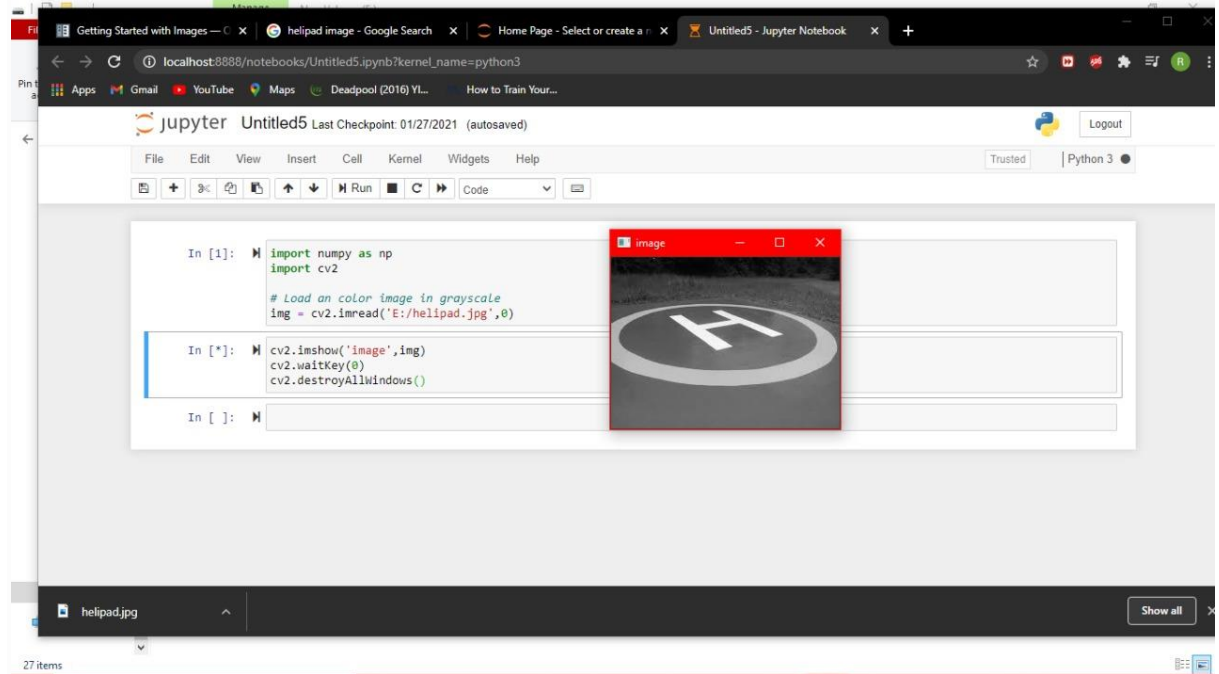
READ ,WRITE AND SAVE AN IMAGE :

```
import numpy as np
import cv2

img = cv2.imread('PATH_TO_IMAGE',0)
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:      # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

output :

figure-1.1.1



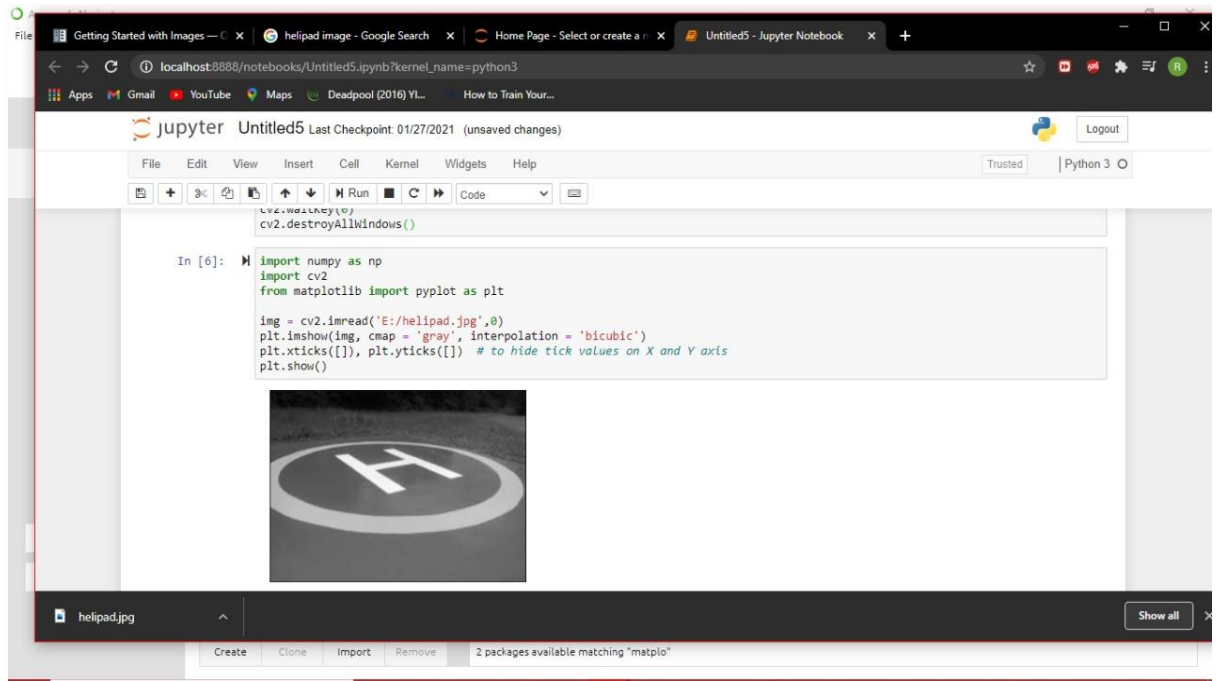
PLOTTING IMAGE :

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('messi5.jpg',0)
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
plt.xticks([], plt.yticks([]) # to hide tick values on X and Y axis
plt.show()
```

output :

figure-1.1.2



2. PROCESSING WITH VEDIOS :

Capture Video from Camera :

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

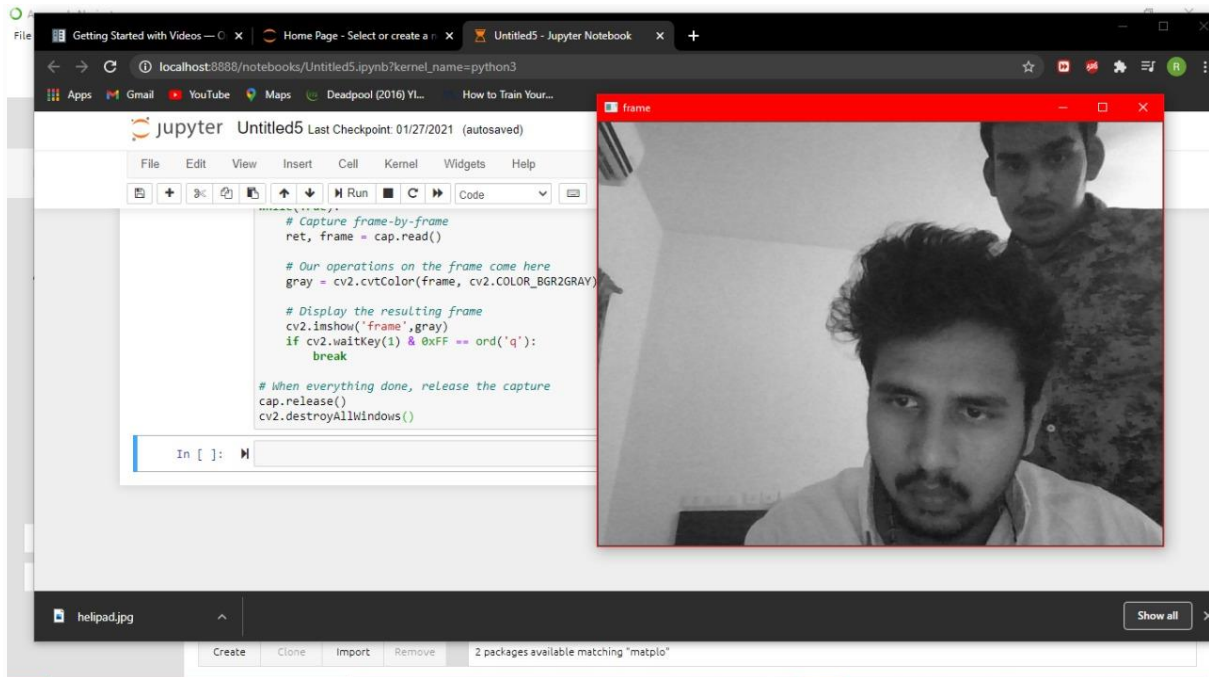
while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

output:
figure-2.1.1



3.DRAWING SHAPES :

1. LINE :

```
import numpy as np
import cv2

# Create a black image
img = np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px
img = cv2.line(img,(0,0),(511,511),(255,0,0),5)
```

2. RECTANGLE :

```
img = cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)
```

3. CIRCLE

```
img = cv2.circle(img,(447,63), 63, (0,0,255), -1)
```

1.3.4.- IMAGE TRANSLATIONS

CODE :

```
import cv2
import numpy as np

image = cv2.imread('C:\\gfg\\tomatoes.jpg')
```

```

# Store height and width of the image
height, width = image.shape[:2]

quarter_height, quarter_width = height / 4, width / 4

T = np.float32([[1, 0, quarter_width], [0, 1, quarter_height]])

# We use warpAffine to transform
# the image using the matrix, T
img_translation = cv2.warpAffine(image, T, (width, height))

cv2.imshow("Originalimage", image)
cv2.imshow("Translation", img_translation)
cv2.waitKey()

cv2.destroyAllWindows()

```

1.3.5- IMAGE RESIZING

CODE:

```

# import the necessary libraries
from skimage import transform
from skimage import filters
import cv2

# read the input image
img = cv2.imread('PATH TO IMAGE ')

# convert image from BGR
# to GRAY scale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# filter out the gradient representation
# image to further process for
# seam carving algorithm
# filters.sobel() is used to
# find edges in an image
filtered = filters.sobel(gray.astype("float"))

for i in range(20, 180, 20):

    # apply seam carve to the image,
    # iterating over the image for
    # multiple frames
    # transform.seam_carve() can transform
    # the seam of image vertically as
    # well as horizontally
    carved_image = transform.seam_carve(img,
                                         filtered,

```

```

        'vertical',
        i)

# show the original image
cv2.imshow("original", img)

# show the carved image
cv2.imshow("carved", carved_image)

# print shape of both images
print("Shape of original image ",
      img.shape)
print("Shape of Carved image ",
      carved_image.shape)

# wait
cv2.waitKey(0)

```

1.3.6-ANALYZING IMAGE HISTOGRAMS

CODE :

```

# load an image in grayscale mode
img = cv2.imread('D:/ex.jpg',0)

# calculate frequency of pixels in range 0-255
histg = cv2.calcHist([img],[0],None,[256],[0,256])

# importing required libraries of opencv
import cv2

# importing library for plotting
from matplotlib import pyplot as plt

# reads an input image
img = cv2.imread('ex.jpg',0)

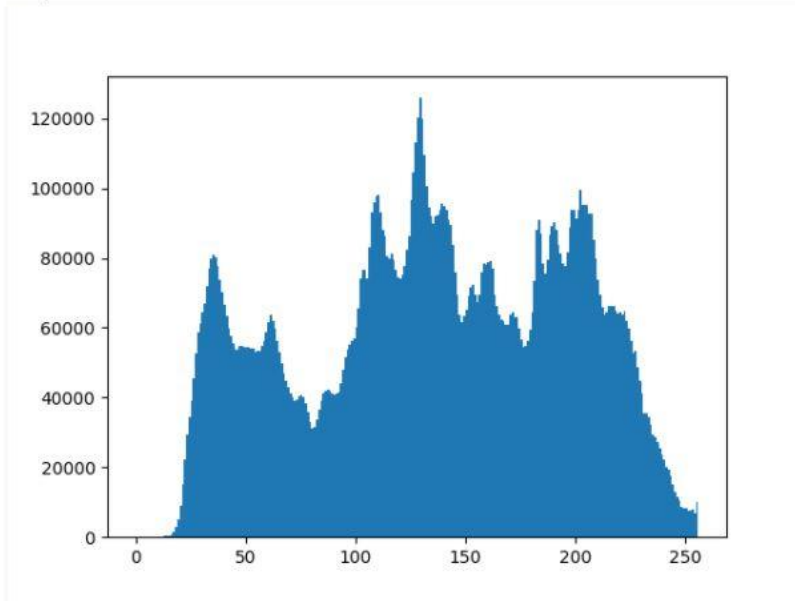
# find frequency of pixels in range 0-255
histr = cv2.calcHist([img],[0],None,[256],[0,256])

# show the plotting graph of an image
plt.plot(histr)
plt.show()

```

OUTPUT :

Output:



1.3.7-IMAGE ROTATION BY ANY ANGLE : **CODE:**

```
import cv2

import numpy as np

img = cv2.imread('IMAGE PATH')
num_rows, num_cols = img.shape[:2]

translation_matrix = np.float32([ [1,0,int(0.5*num_cols)], [0,1,int(0.5*num_rows)] ])
rotation_matrix = cv2.getRotationMatrix2D((num_cols, num_rows), 30, img_translation =
cv2.warpAffine(img, translation_matrix, (1)
img_rotation = cv2.warpAffine(img_translation, rotation_matrix, (2*num_cols, 2*num_rows))

cv2.imshow('Rotation', img_rotation)
cv2.waitKey()
```


OUTPUT:



2.VARIOUS DETECTION ALGORITHMS IN OPEN-CV :

2.1- DETECTION USING HAAR -CASCADE METHOD :

THEORY :

We will use the `cv::CascadeClassifier` class to detect objects in a video stream. Particularly, we will use the functions:

- `cv::CascadeClassifier::load` to load a .xml classifier file. It can be either a Haar or a LBP classifier
- `cv::CascadeClassifier::detectMultiScale` to perform the detection.

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels, all possible sizes and locations of each kernel are used to calculate lots of features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by **Adaboost**.

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. The authors have a good solution for that.

In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of **Cascade of Classifiers**. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is that plan!

The authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. (The two features in the above image are actually obtained as the best two features from Adaboost). According to the authors, on average 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read the paper for more details or check out the references in the Additional Resources section.

PROCESSING:

OpenCV comes with a trainer as well as detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one. Its full details are given here: [Cascade Classifier Training](#).

Here we will deal with detection. OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in `opencv/data/haarcascades/` folder. Let's create face and eye detector with OpenCV.

First we need to load the required XML classifiers. Then load our input image (or video) in grayscale mode.

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Now we find the faces in the image. If faces are found, it returns the positions of detected faces as Rect(x,y,w,h). Once we get these locations, we can create a ROI for the face and apply eye detection on this ROI (since eyes are always on the face !!!).

```

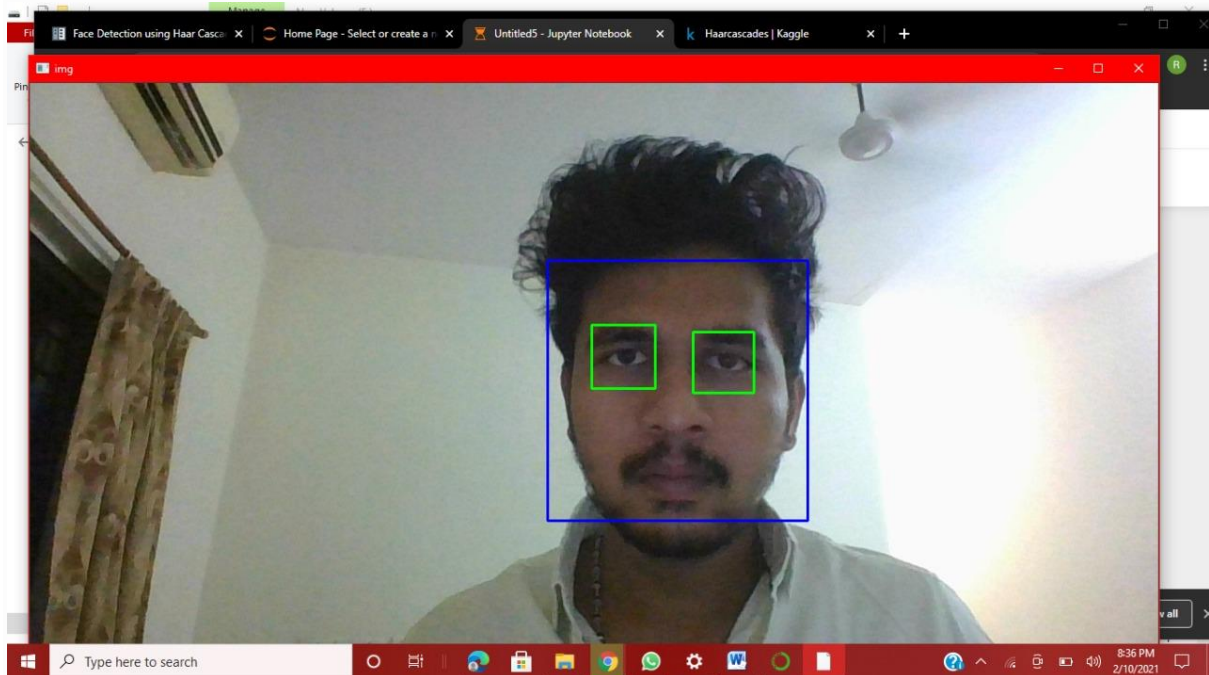
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

output :

figure-1.4 :



2.2- CANNY- EDGE DETECTION :

THEORY: Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages.

1. Noise Reduction

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

2. Finding Intensity Gradient of the Image

Smoothed image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

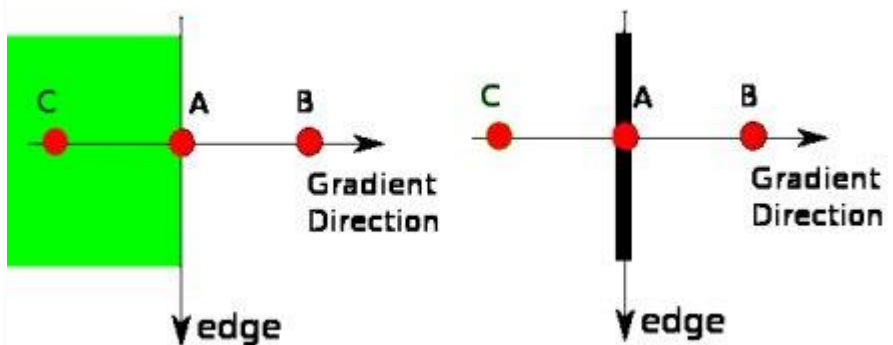
$$Edge_Gradient (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle (\theta) = \tan^{-1} \left(\frac{G_y}{G_x} \right)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

3. Non-maximum Suppression

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:

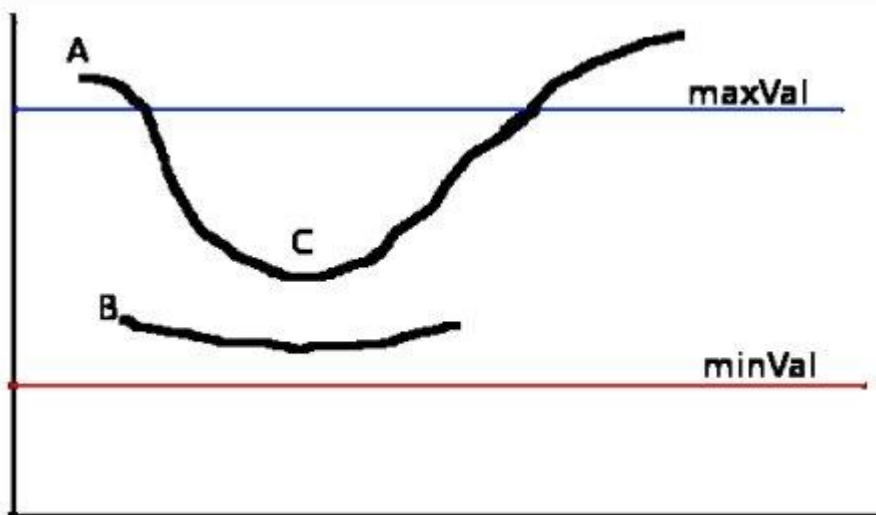


Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

In short, the result you get is a binary image with “thin edges”.

4. Hysteresis Thresholding

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, *minVal* and *maxVal*. Any edges with intensity gradient more than *maxVal* are sure to be edges and those below *minVal* are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:



The edge A is above the *maxVal*, so considered as “sure-edge”. Although edge C is below *maxVal*, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above *minVal* and is in same region as that of edge C, it is not connected to any “sure-edge”, so that is discarded. So it is very important that we have to select *minVal* and *maxVal* accordingly to get the correct result.

This stage also removes small pixels noises on the assumption that edges are long lines.

So what we finally get is strong edges in the image.

PROCESSING:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('IMAGE_PATH.jpg',0)
edges = cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([], plt.yticks([]))
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([], plt.yticks([]))
```

```
plt.show()
```

2.3- HARRIS -CORNER DETECTION :

THEORY:

In the last chapter, we saw that corners are regions in the image with large variation in intensity in all the directions. One early attempt to find these corners was done by **Chris Harris & Mike Stephens** in their paper **A Combined Corner and Edge Detector** in 1988, so now it is called the Harris Corner Detector. He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u,v) in all directions. This is expressed as below:

The window function is either a rectangular window or a Gaussian window which gives weights to pixels underneath.

We have to maximize this function $E(u,v)$ for corner detection. That means we have to maximize the second term. Applying Taylor Expansion to the above equation and using some mathematical steps (please refer to any standard text books you like for full derivation), we get the final equation as:

$$E(u,v) \approx [uv]M[uv]$$

where

$$M = \sum_{x,y} w(x,y) [I_x I_x I_x I_y I_y I_y]$$

Here, I_x and I_y are image derivatives in x and y directions respectively. (These can be easily found using [cv.Sobel\(\)](#)).

Then comes the main part. After this, they created a score, basically an equation, which determines if a window can contain a corner or not.

$$R = \det(M) - k(\text{trace}(M))^2$$

where

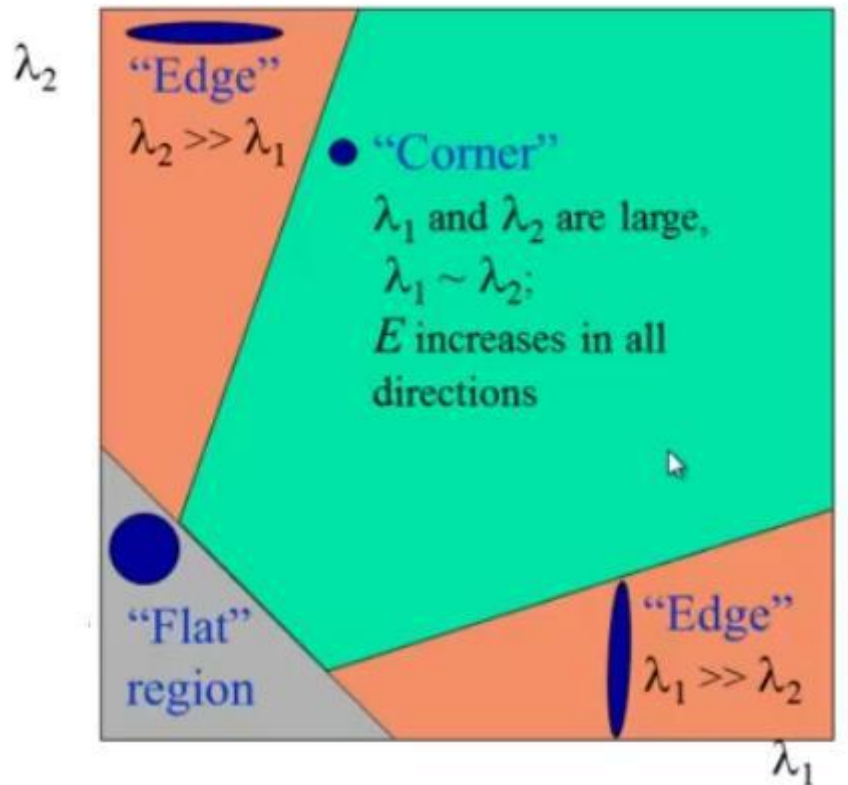
- $\det(M) = \lambda_1 \lambda_2$
- $\text{trace}(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigenvalues of M

So the magnitudes of these eigenvalues decide whether a region is a corner, an edge, or flat.

- When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.

- When $R < 0$, which happens when $\lambda_1 \gg \lambda_2$ or vice versa, the region is edge.
- When R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

It can be represented in a nice picture as follows:



image

So the result of Harris Corner Detection is a grayscale image with these scores. Thresholding for a suitable score gives you the corners in the image. We will do it with a simple image.

PROCESSING :

```
import cv2
import numpy as np

filename = 'PATH_OF_IMAGE.jpg'
img = cv2.imread(filename)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find Harris corners
gray = np.float32(gray)
dst = cv2.cornerHarris(gray, 2, 3, 0.04)
dst = cv2.dilate(dst, None)
ret, dst = cv2.threshold(dst, 0.01 * dst.max(), 255, 0)
dst = np.uint8(dst)
```



```

# find centroids
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)

# define the criteria to stop and refine the corners
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.001)
corners = cv2.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)

# Now draw them
res = np.hstack((centroids,corners))
res = np.int0(res)
img[res[:,1],res[:,0]]=[0,0,255]
img[res[:,3],res[:,2]] = [0,255,0]

cv2.imwrite('subpixel5.png',img)

```

2.4 -OPEN CV FOREGROUND EXTRACTION AND SEGMENTING :

```

# Python program to illustrate
# foreground extraction using
# GrabCut algorithm

# organize imports
import numpy as np
import cv2
from matplotlib import pyplot as plt

# path to input image specified and
# image is loaded with imread command
image = cv2.imread('image.jpg')

# create a simple mask image similar
# to the loaded image, with the
# shape and return type
mask = np.zeros(image.shape[:2], np.uint8)

# specify the background and foreground model
# using numpy the array is constructed of 1 row
# and 65 columns, and all array elements are 0
# Data type for the array is np.float64 (default)
backgroundModel = np.zeros((1, 65), np.float64)
foregroundModel = np.zeros((1, 65), np.float64)

# define the Region of Interest (ROI)
# as the coordinates of the rectangle
# where the values are entered as
# (startingPoint_x, startingPoint_y, width, height)
# these coordinates are according to the input image
# it may vary for different images
rectangle = (20, 100, 150, 150)

# apply the grabcut algorithm with appropriate

```

```

# values as parameters, number of iterations = 3
# cv2.GC_INIT_WITH_RECT is used because
# of the rectangle mode is used
cv2.grabCut(image, mask, rectangle,
            backgroundModel, foregroundModel,
            3, cv2.GC_INIT_WITH_RECT)

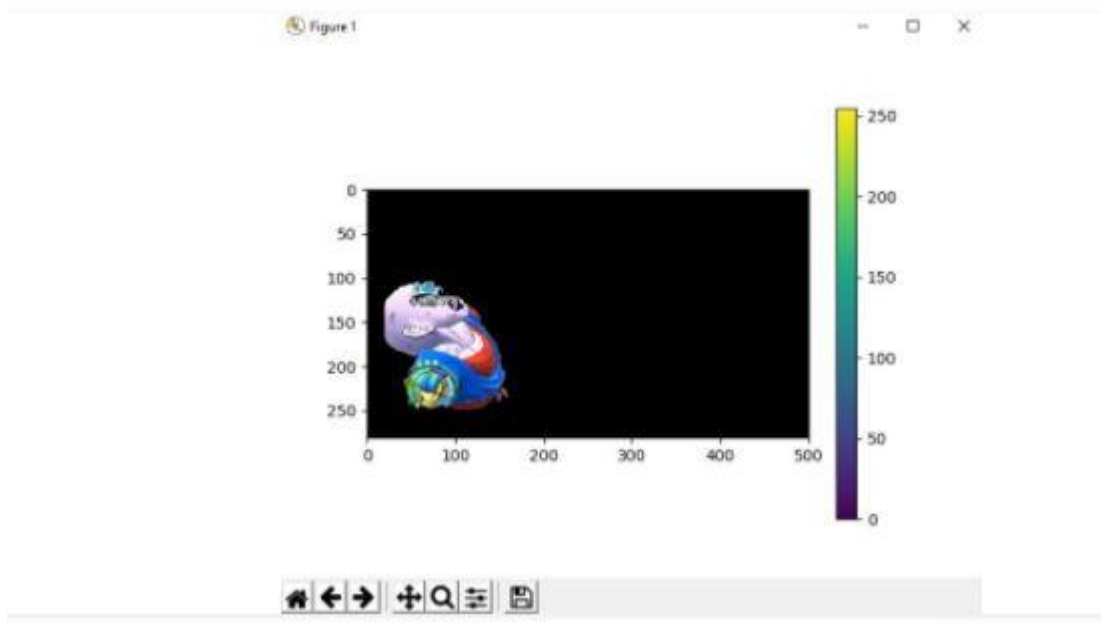
# In the new mask image, pixels will
# be marked with four flags
# four flags denote the background / foreground
# mask is changed, all the 0 and 2 pixels
# are converted to the background
# mask is changed, all the 1 and 3 pixels
# are now the part of the foreground
# the return type is also mentioned,
# this gives us the final mask
mask2 = np.where((mask == 2)|(mask == 0), 0, 1).astype('uint8')

# The final mask is multiplied with
# the input image to give the segmented image.
image = image * mask2[:, :, np.newaxis]

# output segmented image with colorbar
plt.imshow(image)
plt.colorbar()
plt.show()

```

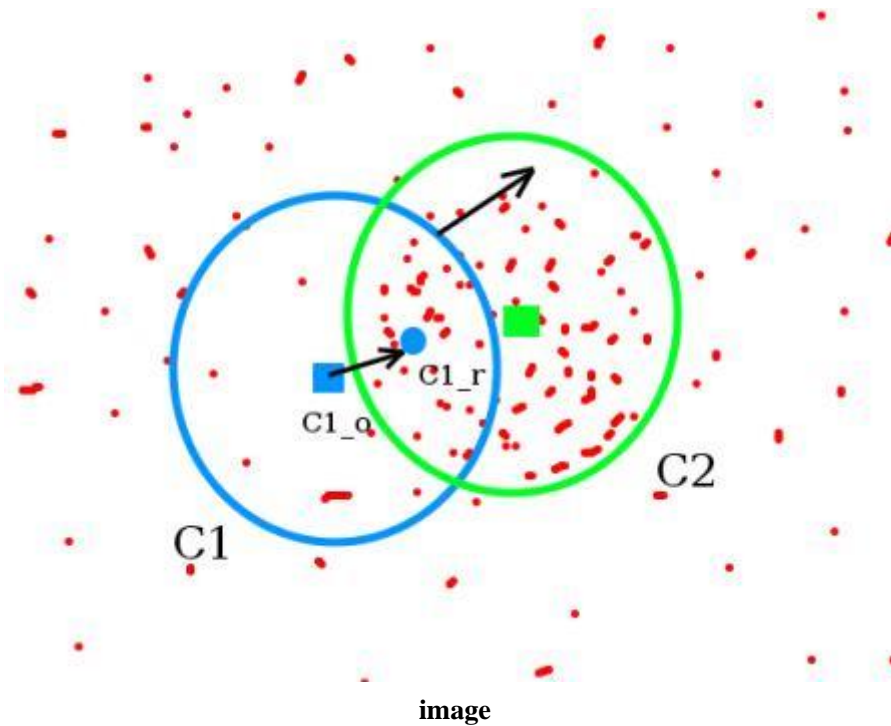
OUTPUT:



2.5- MEAN SHIFT AND CAM SHIFT FOR VIDEO OBJECT TRACKING:

MEAN-SHIFT:

The intuition behind the meanshift is simple. Consider you have a set of points. (It can be a pixel distribution like histogram backprojection). You are given a small window (may be a circle) and you have to move that window to the area of maximum pixel density (or maximum number of points). It is illustrated in the simple image given below:



The initial window is shown in blue circle with the name "C1". Its original center is marked in blue rectangle, named "C1_o". But if you find the centroid of the points inside that window, you will get the point "C1_r" (marked in small blue circle) which is the real centroid of the window. Surely they don't match. So move your window such that the circle of the new window matches with the previous centroid. Again find the new centroid. Most probably, it won't match. So move it again, and continue the iterations such that the center of window and its centroid falls on the same location (or within a small desired error). So finally what you obtain is a window with maximum pixel distribution. It is marked with a green circle, named "C2". As you can see in the image, it has maximum number of points.

So we normally pass the histogram backprojected image and initial target location. When the object moves, obviously the movement is reflected in the histogram backprojected image. As a result, the meanshift algorithm moves our window to the new location with maximum density.

CODE :

```
import numpy as np
import cv2 as cv
import argparse
parser = argparse.ArgumentParser(description="This sample demonstrates the meanshift algorithm. \
The example file can be downloaded from: \
VEDIO PATH )
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()
cap = cv.VideoCapture(args.image)
# take first frame of the video
ret,frame = cap.read()
# setup initial location of window
x, y, w, h = 300, 200, 100, 50 # simply hardcoded the values
track_window = (x, y, w, h)
# set up the ROI for tracking
roi = frame[y:y+h, x:x+w]
hsv_roi = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv.normalize(roi_hist,roi_hist,0,255,cv.NORM_MINMAX)
# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )
while(1):
    ret, frame = cap.read()
    if ret == True:
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv],[0],roi_hist,[0,180],1)
        # apply meanshift to get the new location
        ret, track_window = cv.meanShift(dst, track_window, term_crit)
        # Draw it on image
        x,y,w,h = track_window
        img2 = cv.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv.imshow('img2',img2)
        k = cv.waitKey(30) & 0xff
        if k == 27:
            break
        else:
            break
```

CAM- SHIFT :

Did you closely watch the last result? There is a problem. Our window always has the same size whether the car is very far or very close to the camera. That is not good. We need to adapt the window size with size and rotation of the target. Once again, the solution came from "OpenCV Labs" and it is called CAMshift (Continuously Adaptive Meanshift) published by Gary Bradsky in his paper "Computer Vision Face Tracking for Use in a Perceptual User Interface" in 1998 [28] .

It applies meanshift first. Once meanshift converges, it updates the size of the window as, $s = 2 \times M_{0.256} \sqrt{\quad}$. It also calculates the orientation of the best fitting ellipse to it. Again it applies the meanshift with new scaled search window and previous window location. The process continues until the required accuracy is met.

CODE :

```
import numpy as np
import cv2 as cv
import argparse
parser = argparse.ArgumentParser(description='This sample demonstrates the camshift algorithm. \
The example file can be downloaded from: \
VEDIO PATH')
parser.add_argument('image', type=str, help='path to image file')
args = parser.parse_args()
cap = cv.VideoCapture(args.image)
# take first frame of the video
ret, frame = cap.read()
# setup initial location of window
x, y, w, h = 300, 200, 100, 50 # simply hardcoded the values
track_window = (x, y, w, h)
# set up the ROI for tracking
roi = frame[y:y+h, x:x+w]
hsv_roi = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
roi_hist = cv.calcHist([hsv_roi],[0],mask,[180],[0,180])
cv.normalize(roi_hist,roi_hist,0,255,cv.NORM_MINMAX)
# Setup the termination criteria, either 10 iteration or move by atleast 1 pt
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )
while(1):
    ret, frame = cap.read()
    if ret == True:
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv],[0],roi_hist,[0,180],1)
        # apply camshift to get the new location
        ret, track_window = cv.CamShift(dst, track_window, term_crit)
        # Draw it on image
        pts = cv.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv.polylines(frame,[pts],True, 255,2)
        cv.imshow('img2',img2)
        k = cv.waitKey(30) & 0xff
        if k == 27:
            break
        else:
            break
```

3.HELIPD “H” SIGN DETECTION USING TENSORFLOW AND OPEN CV :

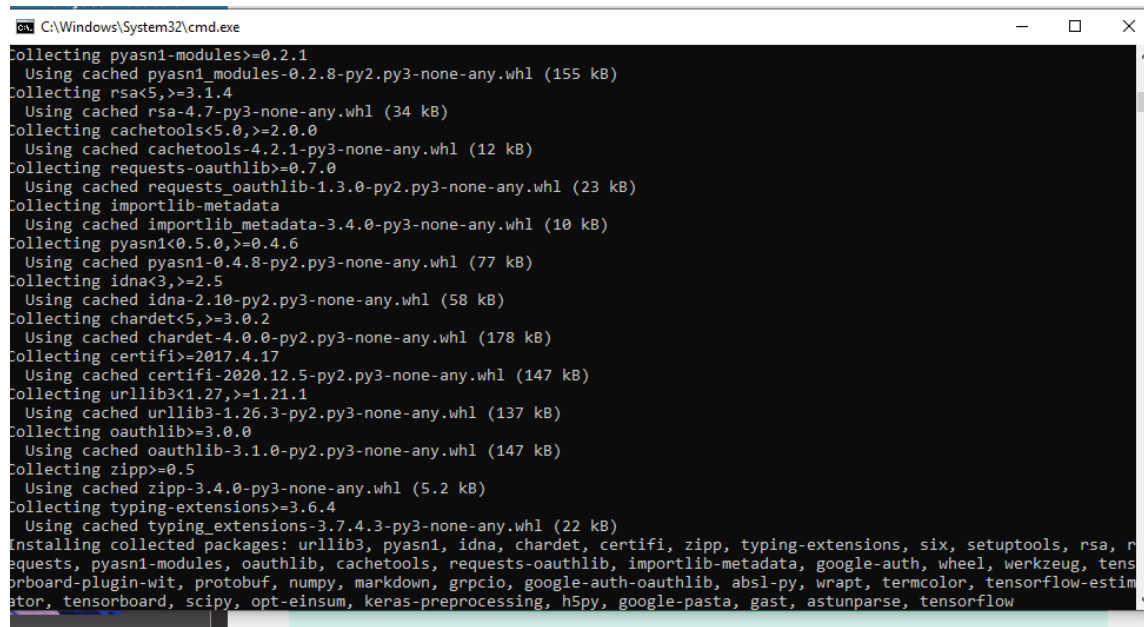
3.1 TENSORFLOW INSTALLATION :

3.1.1 TensorFlow Installation:

Run the following command in a *Terminal* window:

```
pip install --ignore-installed --upgrade tensorflow==2.2.0
```

output :



```
C:\Windows\System32\cmd.exe
Collecting pyasn1-modules<=0.2.1
  Using cached pyasn1_modules-0.2.8-py2.py3-none-any.whl (155 kB)
Collecting rsa<5,>=3.1.4
  Using cached rsa-4.7-py3-none-any.whl (34 kB)
Collecting cachetools<5.0,>=2.0.0
  Using cached cachetools-4.2.1-py3-none-any.whl (12 kB)
Collecting requests-oauthlib<=0.7.0
  Using cached requests_oauthlib-1.3.0-py2.py3-none-any.whl (23 kB)
Collecting importlib-metadata
  Using cached importlib_metadata-3.4.0-py3-none-any.whl (10 kB)
Collecting pyasn1<0.5.0,>=0.4.6
  Using cached pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)
Collecting idna<3,>=2.5
  Using cached idna-2.10-py2.py3-none-any.whl (58 kB)
Collecting chardet<5,>=3.0.2
  Using cached chardet-4.0.0-py2.py3-none-any.whl (178 kB)
Collecting certifi<=2017.4.17
  Using cached certifi-2020.12.5-py2.py3-none-any.whl (147 kB)
Collecting urllib3<1.27,>=1.21.1
  Using cached urllib3-1.26.3-py2.py3-none-any.whl (137 kB)
Collecting oauthlib<=3.0.0
  Using cached oauthlib-3.1.0-py2.py3-none-any.whl (147 kB)
Collecting zipp<=0.5
  Using cached zipp-3.4.0-py3-none-any.whl (5.2 kB)
Collecting typing-extensions<=3.6.4
  Using cached typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Installing collected packages: urllib3, pyasn1, idna, chardet, certifi, zipp, typing-extensions, six, setuptools, rsa, requests, pyasn1-modules, oauthlib, cachetools, requests-oauthlib, importlib-metadata, google-auth, wheel, werkzeug, tensorboard-plugin-wit, protobuf, numpy, markdown, grpcio, google-auth-oauthlib, absl-py, wrapt, termcolor, tensorflow-estimator, tensorboard, scipy, opt-einsum, keras-preprocessing, h5py, google-pasta, gast, astunparse, tensorflow
```

Verify your Installation :

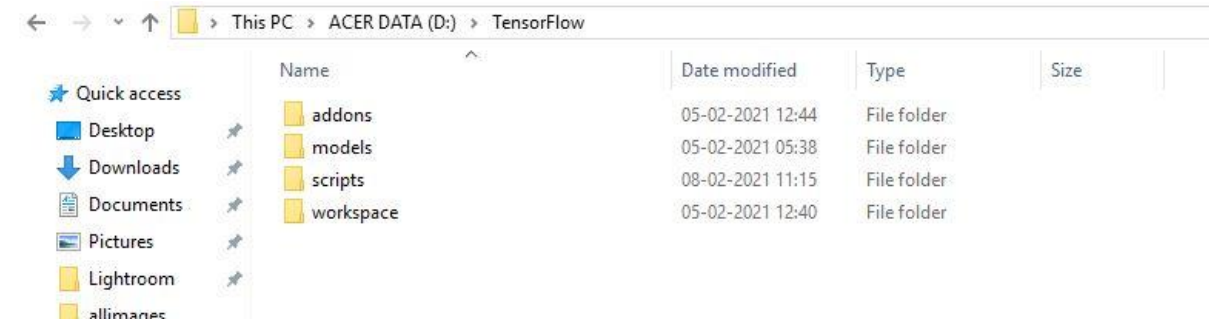
```
python -c "import tensorflow as tf;print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

TensorFlow Object Detection API Installation :

- Create a new folder under a path of your choice and name it `TensorFlow`.
(e.g. `D:\TensorFlow`).
- From your *Terminal* `cd` into the `TensorFlow` directory.
- To download the models you can either use [Git](#) to clone the [TensorFlow Models repository](#) inside the `TensorFlow` folder, or you can simply download it as a [ZIP](#) and extract its contents inside the `TensorFlow` folder. To keep things consistent, in the latter case you will have to rename the extracted folder `models-master` to `models`.

- You should now have a single folder named `models` under your `TensorFlow` folder, which contains another 4 folders as such:

OUTPUT :



Protobuf Installation/Compilation :

- Head to the [protoc releases page](#)
- Download the latest `protoc-*.zip` release (e.g. `protoc-3.12.3-win64.zip` for 64-bit Windows)
- Extract the contents of the downloaded `protoc-*.zip` in a directory `<PATH_TO_PB>` of your choice (e.g. `C:\Program Files\Google Protobuf`)
- Add `<PATH_TO_PB>` to your `Path` environment variable .
- In a new *Terminal* `1`, `cd` into `TensorFlow/models/research/` directory and run the following command:

- `# From within TensorFlow/models/research/
protoc object_detection/protos/*.proto --python_out=.`

NOTE : One must have installed Microsoft visual studio community edition to run setup perfectly.

COCO API installation :

```
pip install cython
pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI
```

Install the Object Detection API :

```
# From within TensorFlow/models/research/
python -m pip install .
```

copy `setup.py` file from `object_detection/packages/tf2/setup.py` to `research` folder and run above command. then it will download all packages and libraries useful for object detection api.

Test your Installation :

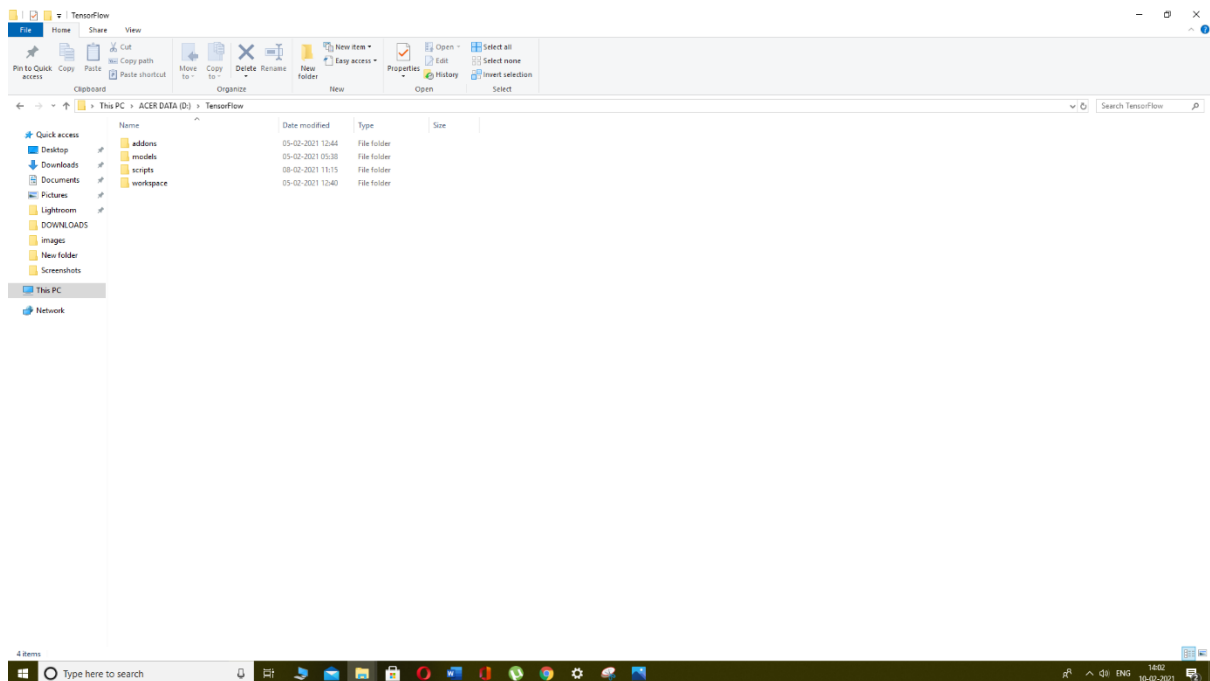
```
# From within TensorFlow/models/research/  
python object_detection/builders/model_builder_tf2_test.py
```

TRAINING CUSTOM DETECTOR :

1. Preparing the Workspace :

If you have followed the tutorial, you should by now have a folder Tensorflow, placed under <PATH_TO_TF> (e.g. D:/Tensorflow), with the following directory tree:

3.1.3 FIGURE:



- Now create a new folder under **TensorFlow** and call it **workspace**. It is within the **workspace** that we will store all our training set-ups. Now let's go under workspace and create another folder named **training_demo**. Now our directory structure should be as so:
- The **training_demo** folder shall be our *training folder*, which will contain all files related to our model training. It is advisable to create a separate training folder each time we wish to train on a different dataset. The typical structure for training folders is shown below.

Preparing the Dataset :

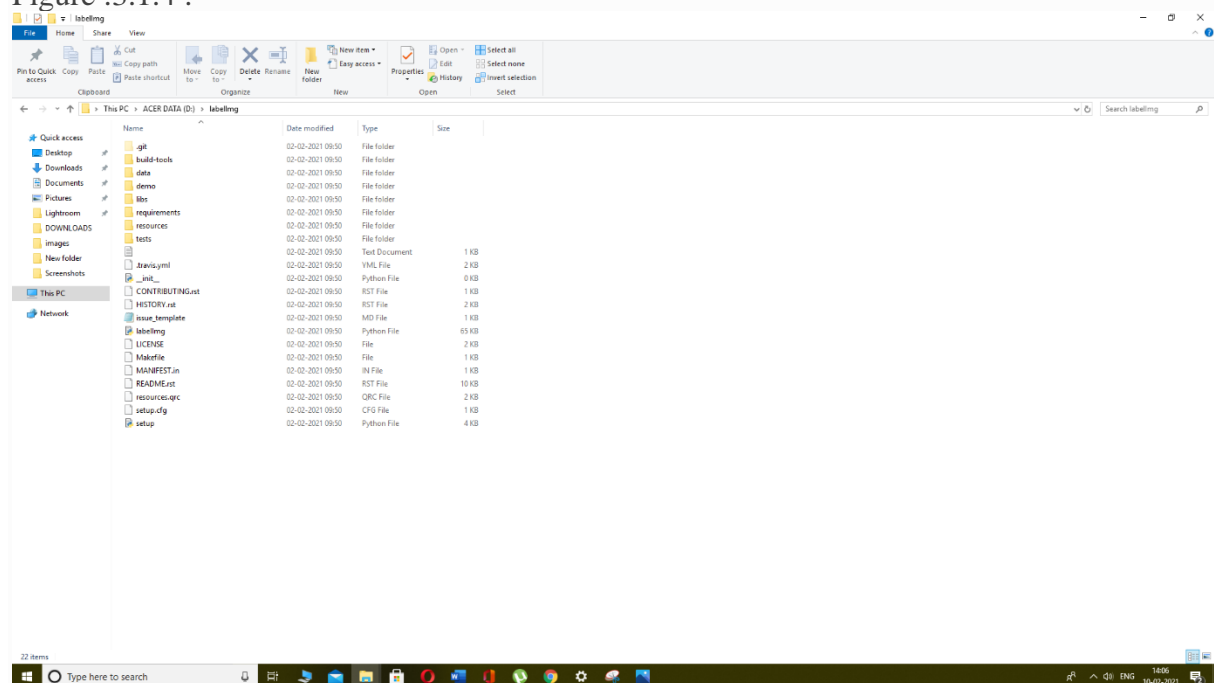
Annotate the Dataset :

1. Install labeling :

1. Download labelImg

- Inside your `TensorFlow` folder, create a new directory, name it `addons` and then `cd` into it.
- To download the package you can either use [Git](#) to clone the [labelImg repo](#) inside the `TensorFlow\addons` folder, or you can simply download it as a [ZIP](#) and extract its contents inside the `TensorFlow\addons` folder. To keep things consistent, in the latter case you will have to rename the extracted folder `labelImg-master` to `labelImg`. [1](#)

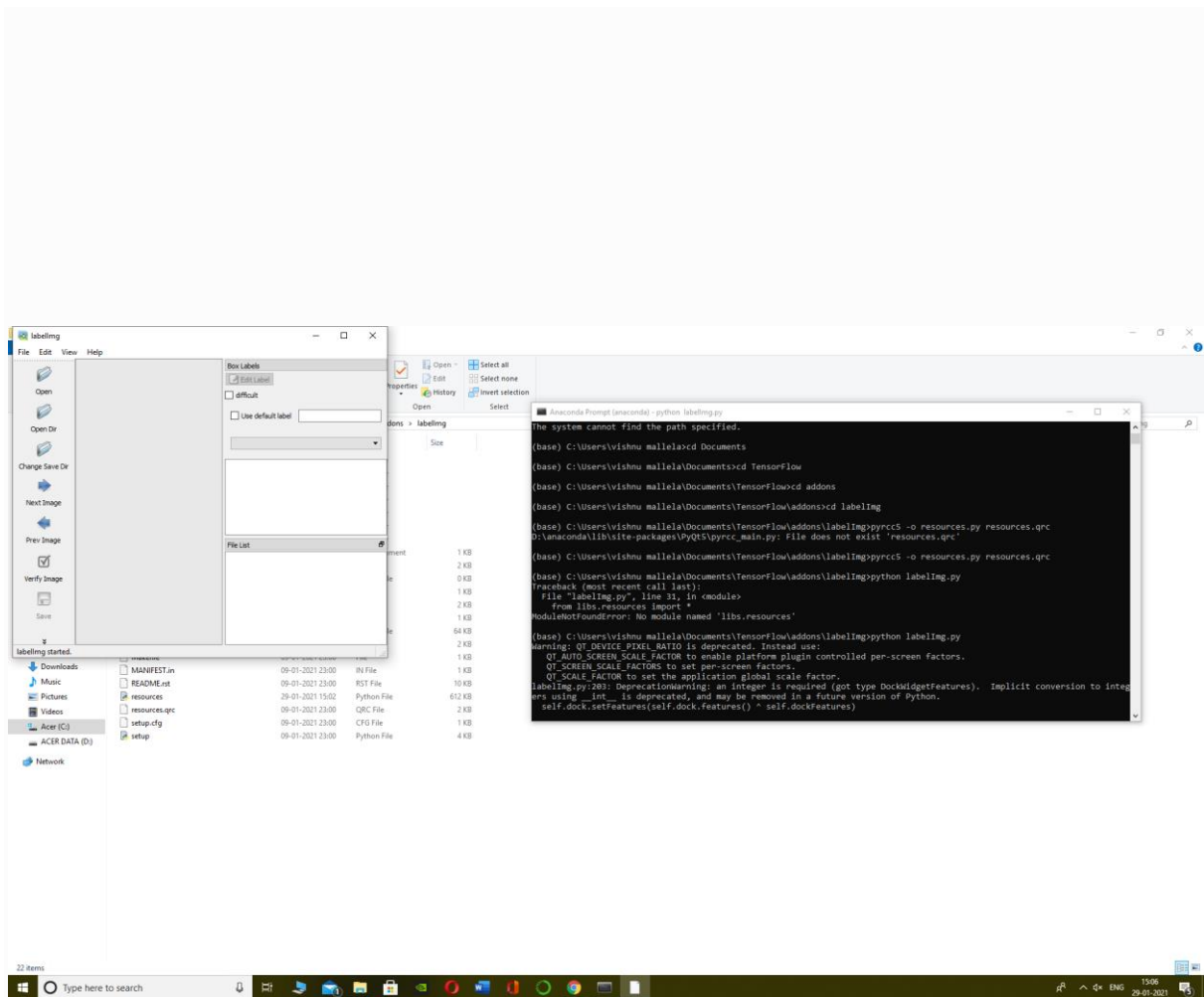
Figure :3.1.4 :



Annotate Images :

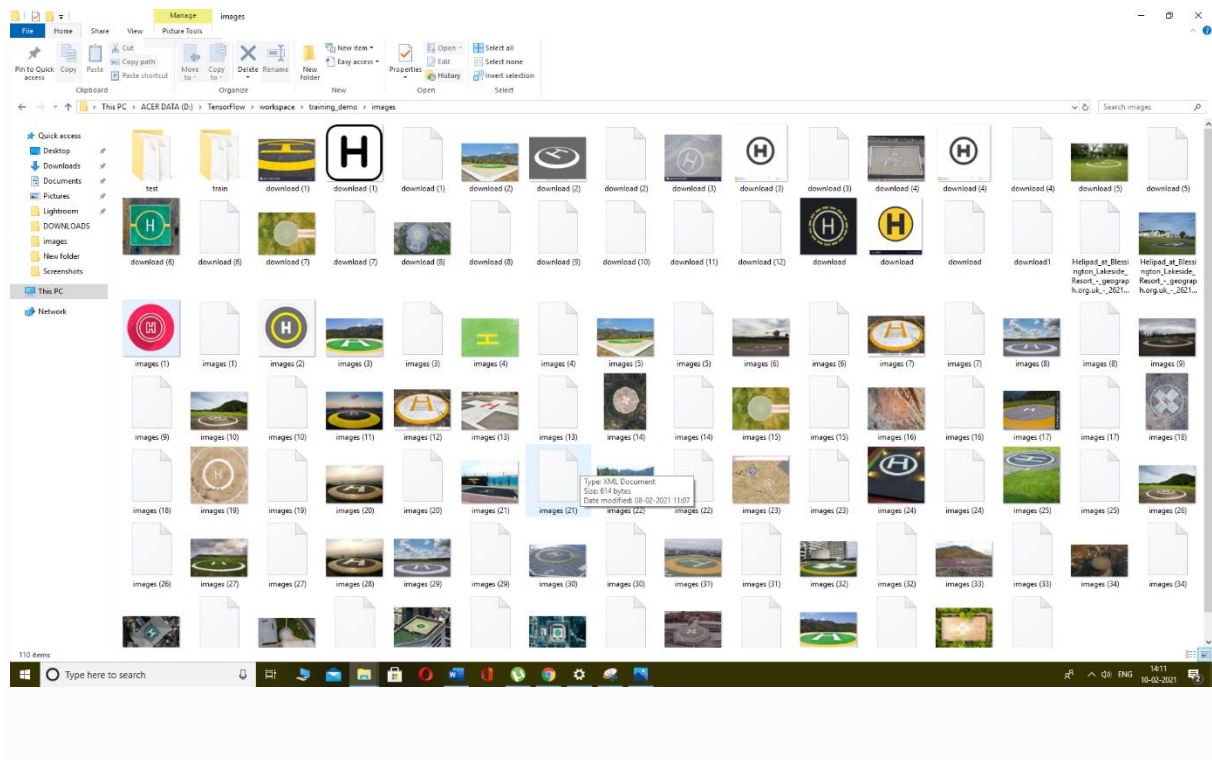
- Once you have collected all the H Sign images to be used to test your model (ideally more than 100 per class), place them inside the folder `training_demo/images`.
- Open a new *Terminal* window.
- Next go ahead and start `labelImg`, pointing it to your `training_demo/images` folder.

Figure :3.1.5



What is important is that once you annotate all your images, a set of new `*.xml` files, one for each image, should be generated inside your `training_demo/images` folder.

Figure :3.1.6 :



Partition the Dataset :

Typically, the ratio is 9:1, i.e. 90% of the images are used for training and the rest 10% is maintained for testing, bOnce you have decided how you will be splitting your dataset, copy all training images, together with their corresponding `*.xml` files, and place them inside the `training_demo/images/train` folder. Similarly, copy all testing images, with their `*.xml` files, and paste them inside `training_demo/images/test`.ut you can chose whatever ratio suits your needs.

Partition the Dataset :

Typically, the ratio is 9:1, i.e. 90% of the images are used for training and the rest 10% is maintained for testing, but you can chose whatever ratio suits your needs.

Once you have decided how you will be splitting your dataset, copy all training images, together with their corresponding `*.xml` files, and place them inside the `training_demo/images/train` folder. Similarly, copy all testing images, with their `*.xml` files, and paste them inside `training_demo/images/test`.

How to do that :

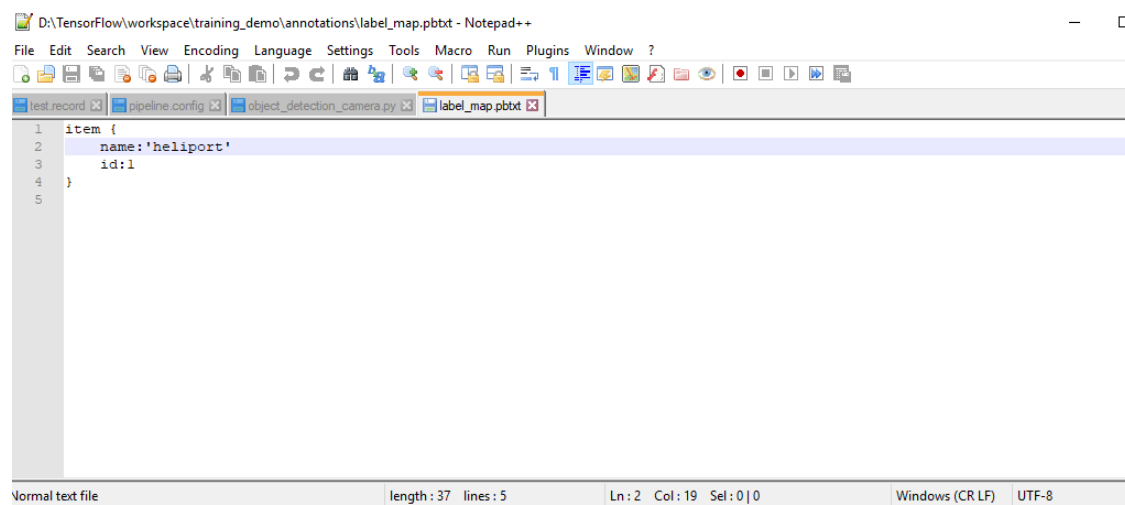
Then, `cd` into `TensorFlow/scripts/preprocessing` and run:

```
python partition_dataset.py -x -i D:Tensorflow/workspace/training_demo/images -r 0.1
```

Create Label Map :

Label map files have the extension `.pbtxt` and should be placed inside the `training_demo/annotations` folder.

Figure :3.1.7



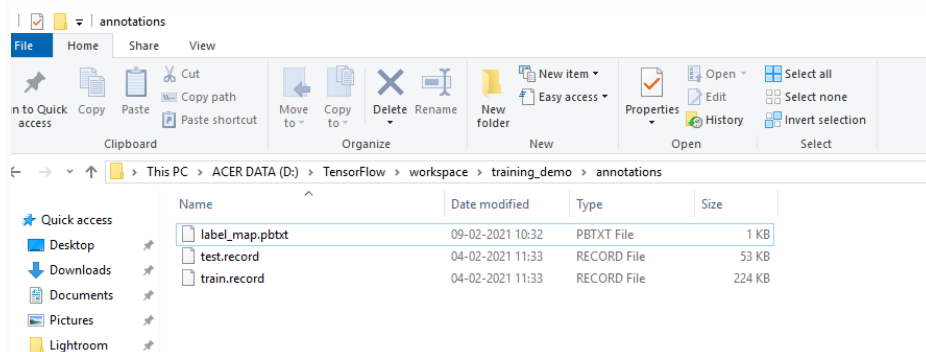
Create TensorFlow Records :

1. Install the `pandas` package:
2. Finally, `cd` into `TensorFlow/scripts/preprocessing` and run:

```
python generate_tfrecord.py -x D:/Tensorflow/workspace/training_demo/images/train -l
D:/Tensorflow/workspace/training_demo/annotations/label_map.pbtxt -o D:/
Tensorflow/workspace/training_demo/annotations/train.record
```

```
# python generate_tfrecord.py -x D:/Tensorflow/workspace/training_demo/images/test -l
D:/Tensorflow/workspace/training_demo/annotations/label_map.pbtxt -o
D:/Tensorflow/workspace/training_demo/annotations/test.record
```

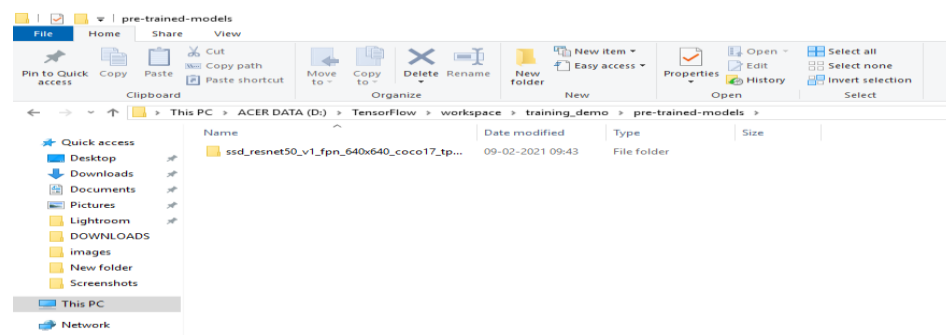
Figure :3.1.8:



Configuring a Training Job :

1. Download Pre-Trained Model

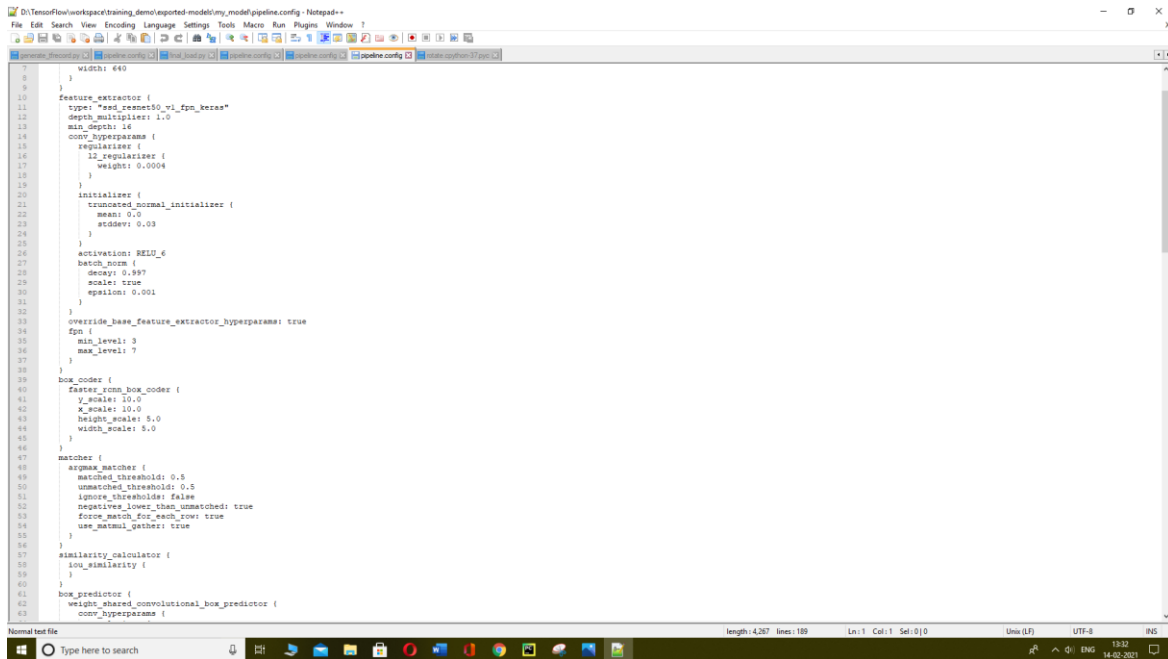
Figure :3.1.9 :



2. Configure the Training Pipeline :

MAKE THESE CHANGES IN CODE PIPELINE.CONFIG :

1. num_classes: 1 *# Set this to the number of different label classes*
2. batch_size: 8 *# Increase/Decrease this value depending on the available memory (Higher values require more memory and vice-versa)*
3. fine_tune_checkpoint: "D:/ tensorflow/workspace/training_demo/pre-trained-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint/ckpt-0" *# Path to checkpoint of pre-trained model*
4. label_map_path: "D:/tensorflow/workspace/training_demo/annotations/label_map.pbtxt"

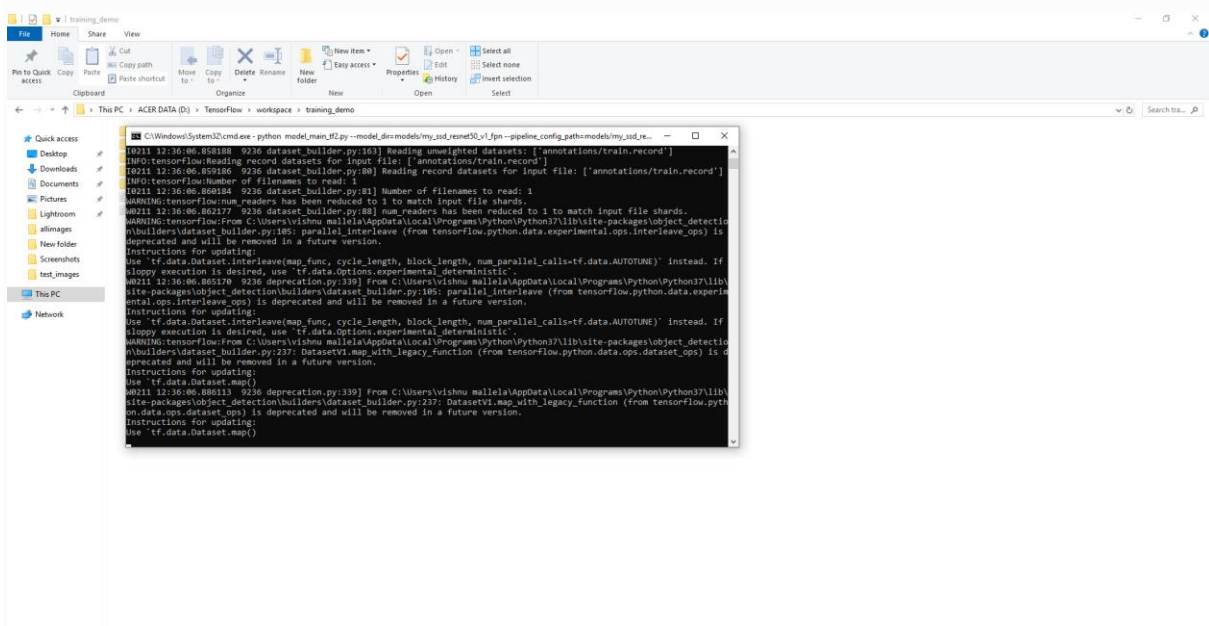


```
7 width: 640
8 }
9 }
10 feature_extractor {
11   type: "ssd_resnet50_v1_fpn_keras"
12   depth_multipliers: 1.0
13   min_depth: 16
14   conv_hypersparams {
15     regularizer {
16       l2_regularizer {
17         weight: 0.0004
18       }
19     }
20     initializer {
21       truncated_normal_initializer {
22         mean: 0.0
23         stddev: 0.03
24       }
25     }
26     activation: RELU_6
27     batch_norm {
28       decay: 0.997
29       scale: true
30       epsilon: 0.001
31     }
32   }
33   override_base_feature_extractor_hypersparams: true
34   fpn {
35     min_level: 3
36     max_level: 7
37   }
38 }
39 box_coder {
40   faster_rcnn_box_coder {
41     y_scale: 10.0
42     x_scale: 10.0
43     height_scale: 5.0
44     width_scale: 5.0
45   }
46 }
47 matcher {
48   argmax_matcher {
49     matched_threshold: 0.5
50     unmatched_threshold: 0.5
51     ignore_thresholds: false
52     negative_lower_than_unmatched: true
53     force_match_for_each_row: true
54     use_matmul_gather: true
55   }
56 }
57 similarity_calculator {
58   iou_similarity {
59   }
60 }
61 box_predictor {
62   weight_shared_convolutional_box_predictor {
63     conv_hypersparams {
```

TRAINING MODEL :

Before we begin training our model, let's go and copy the `TensorFlow/models/research/object_detection/model_main_tf2.py` script and paste it straight into our `training_demo` folder. We will need this script in order to train our model.

Now, to initiate a new training job, open a new *Terminal*, `cd` inside the `training_demo` folder and run the following command:



```
python model_main_tf2.py --model_dir=models/my_ssd_resnet50_v1_fpn --  
pipeline_config_path=models/my_ssd_resnet50_v1_fpn/pipeline.config
```

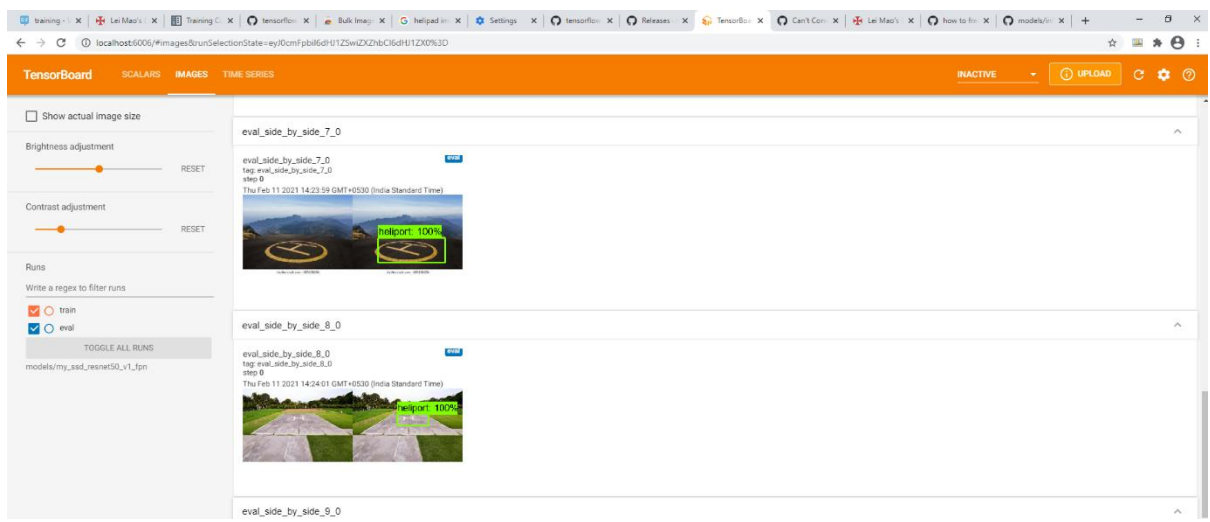
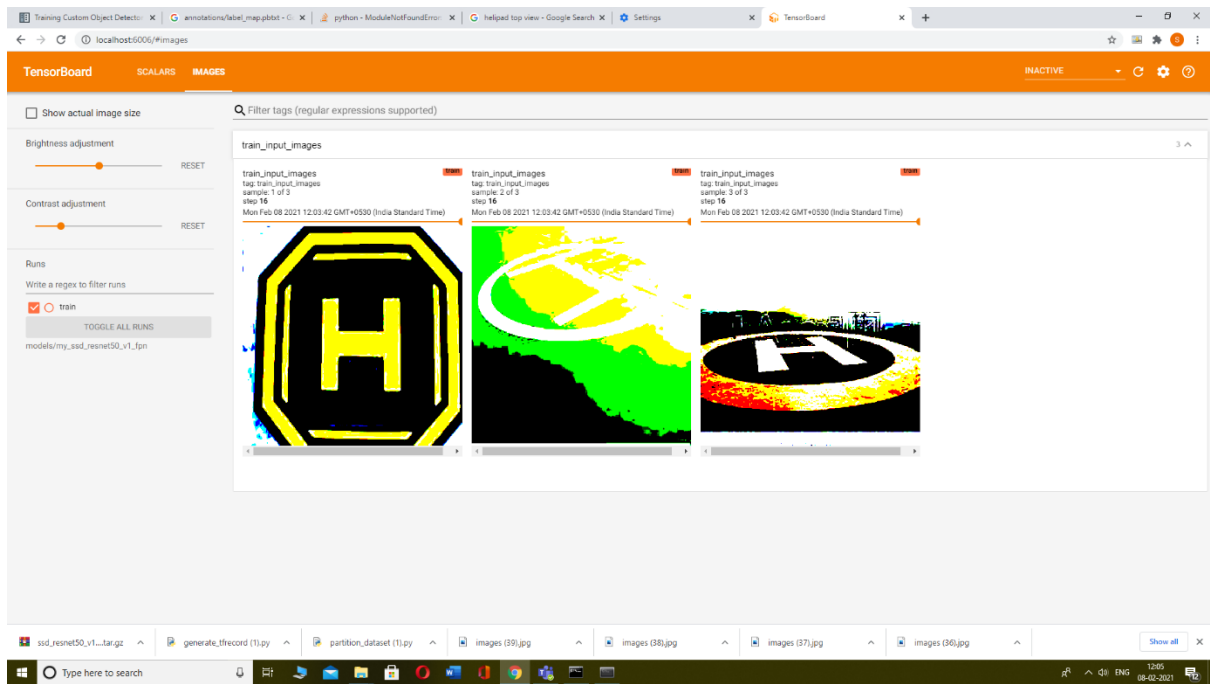
MONITORING TRAINING:

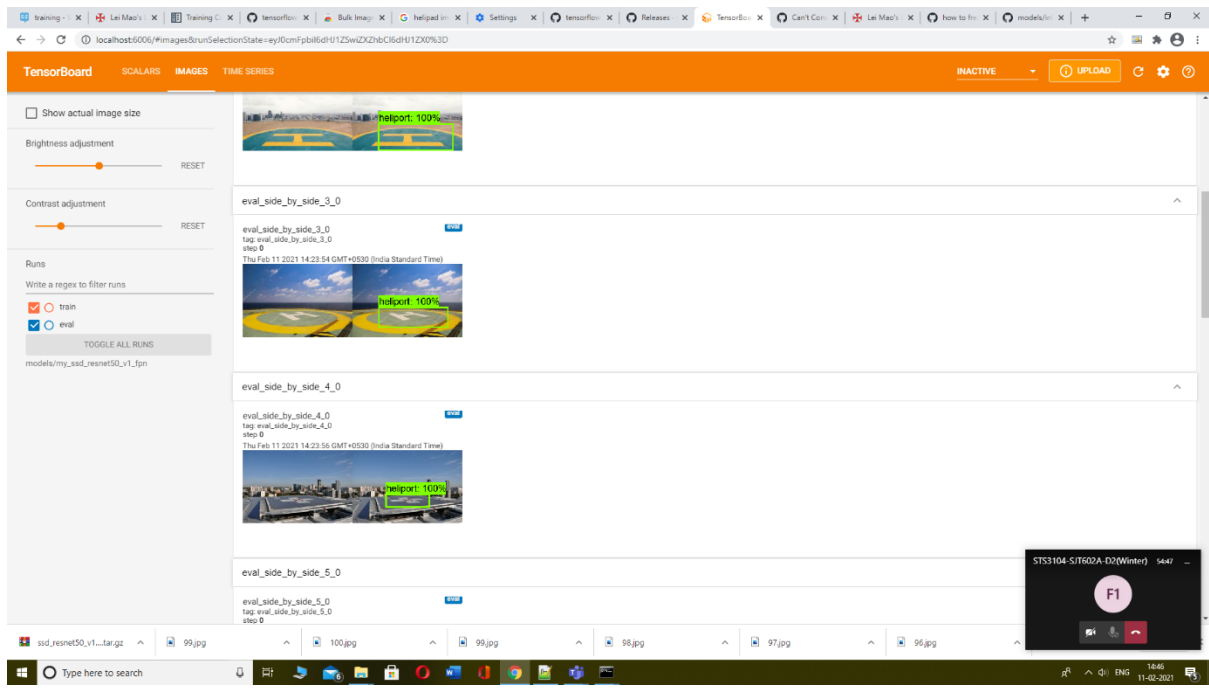
- `cd` into the `training_demo` folder.
- Run the following command:

```
tensorboard --logdir=models/my_ssd_resnet50_v1_fpn
```

OUTPUT :

Figure 3.1.10



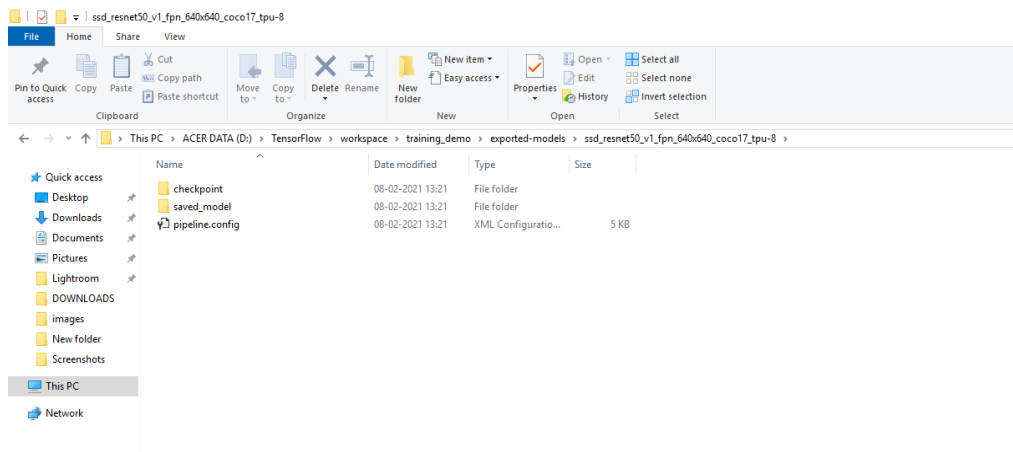


3.2 Exporting a Trained Model :

- Copy the `TensorFlow/models/research/object_detection/exporter_main_v2.py` script and paste it straight into your `training_demo` folder.
- Now, open a *Terminal*, `cd` inside your `training_demo` folder, and run the following command:

```
python .\exporter_main_v2.py --input_type image_tensor --pipeline_config_path
.\models\my_efficientdet_d1\pipeline.config --trained_checkpoint_dir .\models\my_efficientdet_d1\ --output_directory
.\exported-models\my_model
```

Figure 3.1.11
output :



In saved_model we have , .pb file which we use for inference further.

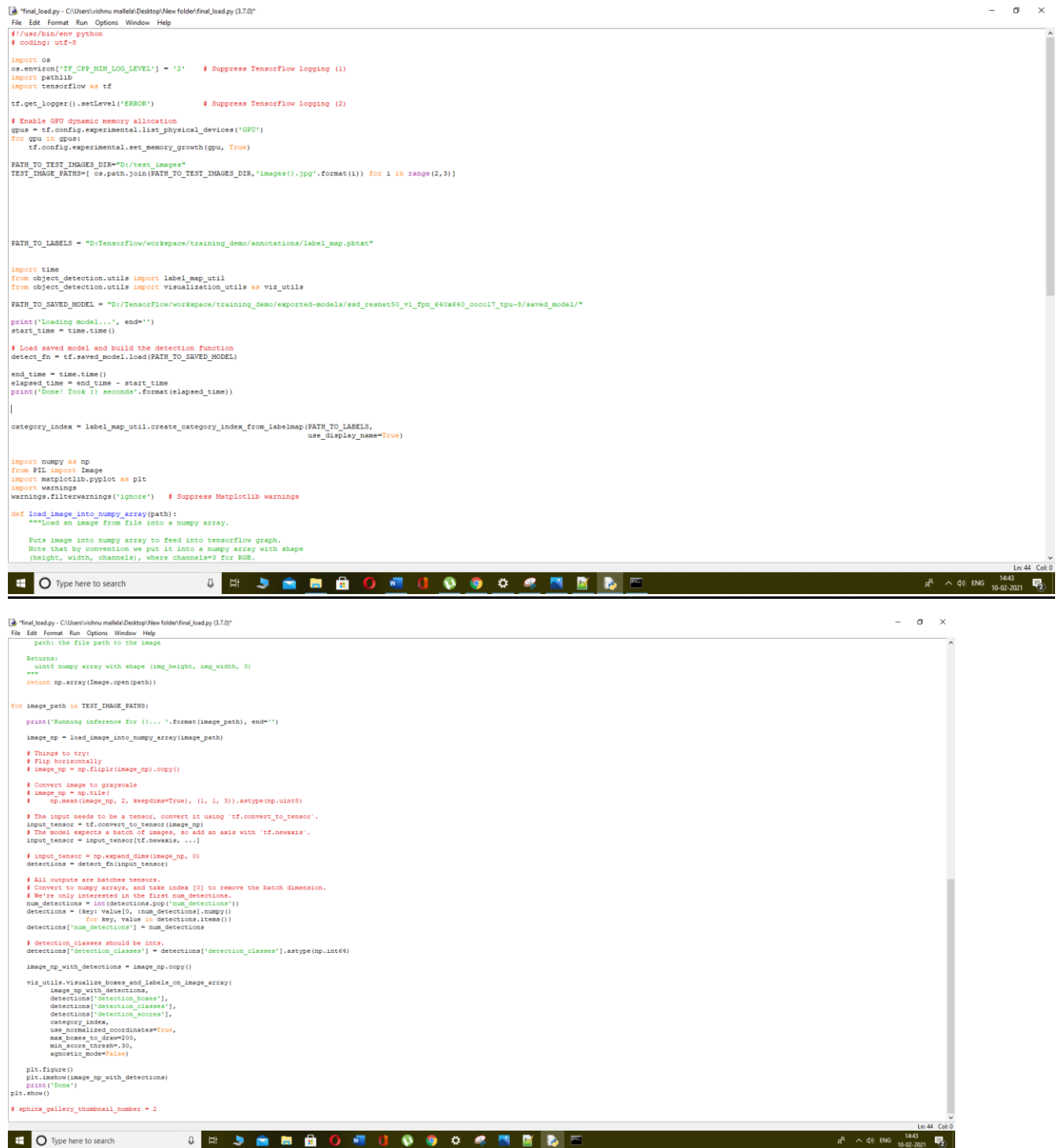
Now our model is ready , no we use open cv-python code to detect helipad sign :

4. DETECTION USING SAVED TF2 MODEL AND OPENCV-PYTHON :

For testing this I created a python code called load_final.py .

CODE:

Figure :3.1.12



```
#!/usr/bin/env python
# coding: utf-8

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # Suppress TensorFlow logging (1)
import pathlib
import tensorflow as tf

tf.get_logger().setLevel('ERROR') # Suppress TensorFlow logging (2)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

PATH_TO_TEST_IMAGES_DIR = "D:/test_images"
TEST_IMAGE_PATHS = [os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(2,3)]

PATH_TO_LABELS = "D:/TensorFlow/workspace/training_demo/annotations/label_map.pbtxt"

import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

PATH_TO_SAVED_MODEL = "D:/TensorFlow/workspace/training_demo/exported-models/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/saved_model/"
print('Loading model...', end='')
start_time = time.time()

# Load saved model and build the detection function
detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)

end_time = time.time()
elapsed_time = end_time - start_time
print('Done! Took {} seconds'.format(elapsed_time))

|

category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_name=True)

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore') # Suppress Matplotlib warnings

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.
    Put image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.
    """
    path = the file path to the image
    Returns:
        uint8 numpy array with shape (img_height, img_width, 3)
    """
    return np.array(Image.open(path))

for image_path in TEST_IMAGE_PATHS:
    print('Running inference for {}...'.format(image_path), end='')
    image_np = load_image_into_numpy_array(image_path)

    # Things to try:
    # Flip horizontally
    # image_np = np.flip(image_np, copy())

    # Convert image to grayscale
    # image_np = np.tile(
    #     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

    # The input needs to be a tensor, convert it using 'tf.convert_to_tensor'.
    input_tensor = tf.convert_to_tensor(image_np)
    # The model expects a batch of images, so add an axis with 'tf.newaxis'.
    input_tensor = input_tensor[tf.newaxis, ...]

    # input_tensor = np.expand_dims(image_np, 0)
    detections = detect_fn(input_tensor)

    # All outputs are batches tensors.
    # Convert to numpy arrays, and take index [0] to remove the batch dimension.
    # We're only interested in the first num_detections.
    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)

    image_np_with_detections = image_np.copy()

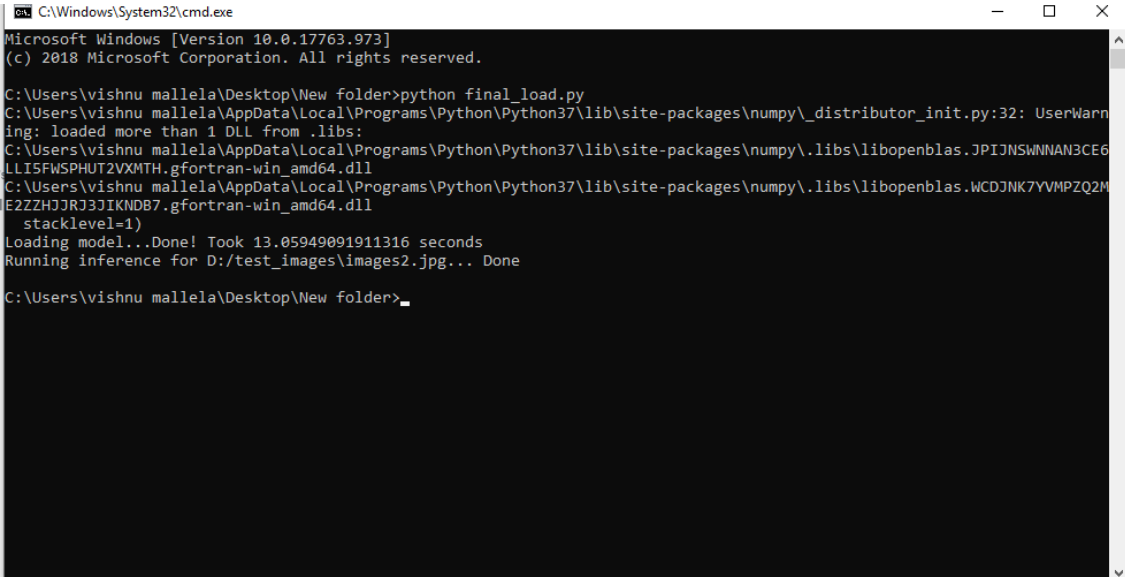
    viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes'],
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=200,
        min_score_thresh=.50,
        agnostic_mode=False)

    plt.figure()
    plt.imshow(image_np_with_detections)
    print('Done')
    plt.show()

# sphinx_gallery_thumbnail_number = 2
```

OUTPUT :

Figure 3.1.13



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\vishnu mallela\Desktop\New folder>python final_load.py
C:\Users\vishnu mallela\AppData\Local\Programs\Python\Python37\lib\site-packages\numpy\_distributor_init.py:32: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\vishnu mallela\AppData\Local\Programs\Python\Python37\lib\site-packages\numpy\.libs\libopenblas.JPIJNSWINNAN3CE6LLI5FWSPHUT2VXXMTH.gfortran-win_amd64.dll
C:\Users\vishnu mallela\AppData\Local\Programs\Python\Python37\lib\site-packages\numpy\.libs\libopenblas.WCDJNK7YVMPZQ2ME2ZZHJ3RJ33IKND87.gfortran-win_amd64.dll
  stacklevel=1)
Loading model...Done! Took 13.05949091911316 seconds
Running inference for D:/test_images/images2.jpg... Done

C:\Users\vishnu mallela\Desktop\New folder>
```

Figure.3.1.14



5.DAY TO DAY ACTIVITIES:

(01-12-2020 TO 30-12-2020) - 4 WEEKS:

01-12-2020:

I VISITED DYSL-AT LABORATORIES ON TUESDAY AND I DISCUSSED ABOUT SCIENTIST THERE AND GOT KNOWLEDGE ABOUT VARIOUS ACTIVITIES HAPPENING THERE AND INTRODUCING MYSELF TO THEM AND LETTING THEM KNOW ABOUT THE KNOWLEDGE I KNOW REGARDING AI AND MACHINE LEARNING.

02-12-2020:

PARTICIPATING IN THEIR ORIENTATION PROGRAMME AND KNOWING ABOUT VARIOUS TECHNOLOGIES THEY ARE USING AND TECHNOLOGIES THAT ARE USEFUL TO KNOW ABOUT IN FUTURE . AND THEY HAVE ASSIGNED US A CHAMBER AND PROVIDED US WITH A SYSTEM.

03-12-2020:

ASKING THE OBJECTIVE OF OUR INTERSHIP WITH MR.MOHANTY SCIENTIST'C' AT ORGANISATION. HE WAS ASSIGNED TO US MONITOR/ MENTOR US.AND HE TOLD US TO GO THROUGH THE ARTICLES WHERE OBJECT DETECTION BASED CONCEPTS ARE THERE AND TOLD US TO TAKE A LOOK AT THEM AND HE SAID AFTER GOING THROUGH THEM HE WILL BE ASSIGNING US TO WORK ON SOMETHING.

04-12-2020:

AND AFTER LETTING HIM KNOW THE PROCESS OF OUR UNDERSTANDING ABOUT OBJECT DETECTION . HE TOLD US TO GO THROUGH OPEN-CV TUTORIALS AND LEARN THEM AND UNDERSTAND VARIOUS OBJECT DETECTION ALGORITHMS IN OPENCV.

05-12-2020 :

REVISING THE WORK DONE THROUGHOUT WEEK AND LEARNING AND SOLVING ERRORS.

06-12-2020:

WEEKEND.

07-12-2020:

GETTING STARTED WITH OPENCV AND LEARNING THE BASICS ABOUT IMAGE PROCESSING AND VARIOUS LIKE IMAGE PLOTTING , READING , SAVING AND PRACTING THEM.

08-12-2020 :

GETTING STARTED WITH OPENCV VIDEO PROCESSING LIKE READING , WRITING AND SAVING IMAGE AND ACCESSING WEBCAM FOR VIDEO RECORDING AND SAVING .

09-12-2020 :

GETTING STARTED WITH SHAPES IN OPENCV AND KNOW HOW TO DRAW RECTANGLES ,SQUARES CIRCLES AND VARIOUS SHAPES ON AN IMAGE IN OPENCV.

10-12-2020 :

LEARNING VARIOUS IMAGE TRANSLATIONS AND RESIZING AND LEARNING ROTATIONS OF AN IMAGE BY ANY DEGREE AND IMAGE HISTOGRAMS .

11-12-2020:

LEARNING VARIOUS ARTICLES IN PYIMAGE SEARCH WEBSITE FOR GETTING HOW TO USE THEM AND HOW ABOVE IMAGE AND VIDEO PROCESSING ARE USEFUL IN OBJECT DETECTION .

12-12-2020:

REVISING THE WORK DONE THROUGHOUT WEEK AND LEARNING AND SOLVING ERRORS.

13-12-2020:

WEEKEND.

14-12-2020:

AGAIN ON MONDAY , WE MET MR.MOHANTY AND DISCUSSED ABOUT OUR PROGRESS AND LETTING HIM KNOW WHATEVER WE GONE THROUGH AND HE TOLD US TO GO ON FURTHER WITH VARIOUS DETECTION ALGORITHMS IN OPENCV .AND AGAIN WE HAD MEETING WITH DIRECTOR MR.SHIVA PRASAD SIR AND AGAIN HE CONDUCTED A MEETING FOR ALL OUR INTERNS AND DISCUSSED ABOUT THE VARIOUS PROJECTS HAPPENING THERE AND ETC.

15-12-2020:

WE STARTED LOOKING OVER VARIOUS OBJECT DETECTION ALGORITHMS IN OPENCV AND DICIDED TO LEARN POSSIBLE USEFUL ALGORITHMS .AND WE STARTED LEARNING HAAR-CASCADE OBJECT DETECTION AND LEARNING HOW IT SAMPLES POSITIVE AND NEGATIVE IMAGES FOR DETECTION .

16-12-2020:

LEARNT CANNY EDGE -DETECTION AND GETTING TO KNOW ABOUT HOW IT IS USEFUL IN DETECTION EDGES AND GETTING AN IDEA HOW IT WORKS IF EDGE TO BE DETECTED AND PRACTICING IT ON VARIOUS IMAGES.

17-12-2020:

LEARNING HARRY-CORNER DETECTION AND FOREGROUND-EXTRACTION AND PRACTICING THEM.

18-12-2020:

MEAN SHIFT AND CAM-SHIFT ARE IMPORTANT AND USEFUL ALGORITHMS IN OBJECT VIDEO TRACKING SO UNDERSTANDING THESE SHIFT ALGORITHM PROCESSES AND PRACTICING THEM .

19-12-2020 :

REVISING THE WORK DONE THROUGHOUT WEEK AND LEARNING AND SOLVING ERRORS.

20-12-2020:

WEEKEND.

21-12-2020:

SHOWING OUR PROGRESS TO MR.MOHANTY ON OPENCV LEARNING ASKING HIM ABOUT DETAILS OF OUR FURTHER PROCESS AND HE ASSIGNED US TO DO A CUSTOM OBJECT DETECTOR FOR DETECTING “H” HELIPAD SIGNS WHICH CAN BE USEFUL FURTHER FOR LANDING DRONES ON THEM.

22-12-2020:

GOING THROUGH VARIOUS ARTICLES AND WEBSITES ON HOW TO MAKE A CUSTOM DETECTOR AND DECIDED TO MAKE CUSTOM DETECTOR ON TENSORFLOW PLATFORM BECAUSE IT PROVIDES A LOT OF EFFICIENT PRE-TRAINED MODELS TO TRAIN UPON OUR NEEDS. AND WE HAVE KNOWN HOW TO GO ON FURTHER WITH TENSORFLOW AND INSTALLING THE TENSORFLOW API .

23-12-2020:

INSTALLING TENSORFLOW AND INSTALLING VARIOUS PACKAGES AND DEPENDENCIES FOR PROCESSING WITH TENSORFLOW.CREATING A TENSORFLOW API FOR TRAINING OUR CUSTOM DETECTOR .

24-12-2020:

FOR TRAINING , WE NEED DATA SET SO WE COLLECTED A BATCH OF IMAGES OF H SIGNS FOR DATASET PREPARATION AND ANNOTATED THEM TO COVERT INTO XML FILES AND THEY ARE CONVERTED LATER TO TENSORFLOW RECORD FILES .

25-12-2020:

FOR CONFIGURING A TRAINING JOB WE HAVE DOWNLOADED MANY PRE TRAINED MODELS AND TRAINED THEM FOR DETECTING “ H” SIGNS , WE FAILED WHILE TRAINING THEM BECAUSE OF SOME SYSTEM REQUIREMENT ERRORS AND AFTER KNOWING ERORRS AND KNOWING HOW TO CONFIGURE CHECKPONT AND HOW TO MAKE IT WORK.

26-12-2020:

REVISING THE WORK DONE THROUGHOUT WEEK AND LEARNING AND SOLVING ERRORS.

27-12-2020 :

WEEKEND.

28-12-2020:

TRAINED ON VARIOUS PRE- TRAINED MODELS BUT GOT SOME BAD RESULTS WITH FEW AND GOOD WITH SOME AFTER ALL THESE WE FIXED OUR MODEL AS SSD_RESNET.

29-12-2020:

EXPORTING THE TRAINED MODEL AND USING THIS MODEL TO DETECT H SIGNS AND CREATING A CODE TO TAKE INPUT FROM SYSTEM AND DETECT THEM USING THIS SAVED MODEL AND OPENCV .

GOT SOME FRUITFUL RESULTS BUT TO MAKE AN VERY ACCURATE DETECTOR SHOULD BE TRAINED ON A BIG DATA SET AND ALSO TO BE TRAINED ON MORE NEGATIVE IMAGES .

30-12-2020:

FINALLY SPEAKING WITH MR.MOHANTY AND REPORTING HIM THE RESULTS AND HE ASKED TO MEET DIRECTOR OF DYSL AND SHOW HIM THE PROCESS THAT WE HAVE DONE AND HE APPRICIATED OUR WORK AND HANDED OVER CERTIFICATE TO ME .

COMPARISON OF COMPETENCY LEVELS BEFORE AND AFTER INTERNSHIP(SELF EVALUATION) :

THE SKILLSET OF THE STUDENT BEFORE TRAINING:

I'm a fast learner adept in the art of technology as well as firm grasp over the subjects, I have studied with regard to the concepts taught. Before the training program, I was not exposed to any sort of knowledge about machine learning or detection techniques, nor was I competent in any of the use detection methods. I was unaware of any machine learning concepts or detecting concepts but only knew the very basics (on reading blogs before my internship). I was not exposed to how detection algorithms work or what the algorithms used were.

SKILLS ACQUIRED AFTER TRAINING:

- 1.GOT TO KNOW VARIOUS OPEN-CV PROCESSING AND TECHNIQUES .**
- 2. LEARNT VARIOUS OBJECT DETECTION ALGORITHMS IN OPENCV AND KNOW HOW TO APPLY THEM.**
- 3.GOT SOME GOOD KNOWLEDGE ON OPEN-CV PROCESSING.**
- 4. LEARNT HOW TO MAKE A CUSTOM DETECTOR WHICH IS A PART OF LEARNING IN MACHINE LEARNING.**

CONCLUSION:

MY 4-WEEK INTERNSHIP WITH DRDO-DYSL-AT WAS A HUGE SUCCESS AND A GREAT TIME SPENT FOR THE ACQUISITION OF KNOWLEDGE AND SKILLS .THROUGH MY TRAINING I WAS EXPOSED TO APPLICATIONS AND FIELDS OF IMPLEMENTATION ON OBJECT DETECTION PROCESSING . THROUGH MY TRAINING I WAS ABLE TO APPRECIATE MU CHOSE FIELD OF STUDY EVEN MORE,BECAUSE I WAS ABLE TO BLEND MY THEORETICAL KNOWLEDGE TO A HANDS ON PRACTICAL APPROACH. MY TRAINING HERE HAS GIVEN ME A BIRD'S EYE OF THE VAIRABILITY OF THE SCOPE OF COMPUTER SCIENCE ENGINEERING AND ENCOURAGED ME TO PURSUE THE VERY

SAME WHOLE HEARTEDLY.I HAVE ALSO BENN ABLE TO IMPROVE MY INTERPERSONAL SKILLS AND HAVE BEEN ABLE TO GAUGE WHAT A LIFE AFTER COLLEGE WOULD LOOK LIKE WHILE DEALING WITH CLIENTS IN THE REAL WORLD.

IN CONCLUSION, I GOT GREAT KNOWLEDGE ON OPEN-CV PROCESSING AND DEVELOPED CUSTOM OBJECT DETECTOR FOR DETECTING HELIPAD SIGNS.

APPENDICES :

- 1.https://docs.opencv.org/master/d9/df8/tutorial_root.html
2. <https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/>
3. <https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/>
4. <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/>
5. <https://github.com/nicknochnack/RealTimeObjectDetection>

