

Real-Time Human Detection and Counting

Submitted in partial fulfilment of the requirements for the degree of

Bachelors of Technology
in
Computer Science and Engineering

by
Alokam Nikhitha
19BCE2555

Under the guidance of-
Prof. Gunavathi C.

School of Computer Science and Engineering (SCOPE)
VIT, Vellore



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

December, 2021

DECLARATION

I hereby declare that the thesis entitled “*Real-Time Human Detection and Counting*” submitted by me, for the award of the degree of Bachelors of Technology in Computer Science and Engineering to VIT is a record of bona fide work carried out by me under the supervision of Prof Gunavathi C.

I further declare that the work reported in this thesis 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.

Place : Vellore

Alokam Nikhitha

Date : 27/11/2021

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled “*Real-Time Human Detection and Counting*” submitted by Alokam Nikhitha (19BCE2555), School of Computer Science and Engineering (SCOPE), VIT, for the award of the degree of Bachelors of Technology in Computer Science and Engineering, is a record of bona fide work carried out by him under my supervision during the period, 01.08.2021 to 30.12.2021, as per the VIT code of academic and research ethics.

The contents of this report have 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. The thesis fulfils the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 27/11/2021

Signature of the Guide

Internal Examiner

External Examiner

Head of the Department

School of Computer Science and Engineering

ACKNOWLEDGEMENT

The project “*Real-Time Human Detection and Counting*” was made possible because of inestimable inputs from everyone involved, directly or indirectly. I would first like to thank my guide, *Prof. Gunavathi C*, who was highly instrumental in providing not only a required and innovative base for the project but also crucial and constructive inputs that helped me make my final product. Our professor guide has helped us perform research in the specified area and improve my understanding in the area of artificial intelligence and detection systems and I am very thankful for her support all throughout the project. I would also like to thank my team-mate *Harshit Mishra (19BCE0799)* for his tireless efforts that aided me to overcome technical inaccuracies in my code and helped me in completion of the final prototype.

Finally, I would like to thank *Vellore Institute of Technology*, for providing me with a flexible choice of the project and for supporting my work and implementation related to the project.

Alokam Nikhitha

19BCE2555

Executive Summary

The project “*Real-Time Human Detection and Counting*” works on the image recognition using Histogram of Oriented Gradient (HOG) model. The Histogram of Oriented Gradients is a feature descriptor that is used in Computer Vision and Image Processing for the purpose of object detection. The model works on recognising gradient orientations in localized portions of an image. The essential thought behind this HOG descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is concatenation of these histograms.

With this human detection done, we aim at using this *counting system* to study the traffic congestion patterns at major circles of a town or city. This will help in figuring out busy timings of a particular area and will aid the process of marking sparse location for faster travels. We can also use this model for picking of going-out-of-home locations, the examples of which include parks and restaurants. This use is an extension of studying traffic patterns and finding the sparse locations.

Table of Contents

CONTENTS	Page No.
Acknowledgement	i
Executive Summary	ii
Table of Contents	iii
List of Figures	iv
List of Tables	iv
Abbreviations	iv
Symbols and Notations	v
1. INTRODUCTION	1
1.1. OBJECTIVE	1
1.2 MOTIVATION	1
1.3 BACKGROUND	2
2. Applications of Object Detection	3
3. Challenges in Performing Real-Time Object Detection	3
4. Literature Review	4
5. Methodology	6
6. Tools and Requirements	6
6.1 Software Requirements	6
6.2 Dataset/Tools Used	6
7. IMPLEMENTATION	7
8. RESULTS AND CONCLUSION	13
9. REFERENCES	15

List of Figures

Figure	Page No
Fig 8.1 Human Detection while using Camera	13
Fig 8.2 Human Detection while using Image	14
Fig 8.3 Human Detection while using Video	14

List of Tables

Table	Page No
Table 1 Literature Review	5

Abbreviations

- HOG → Histogram of Gradients
- CV → Computer Vision
- WebCam → Web Camera
- GPS → Global Positioning System
- AI → Artificial Intelligence
- ML → Machine Learning
- DL → Deep Learning
- XML → extensible Markup Language
- env → Virtual Environment
- RCNN → Region based Convolutional Neural Networks
- CNN → Convolutional Neural Networks

Symbols and Notations

- $X \rightarrow$ x coordinate in 2d plane
- $Y \rightarrow$ y coordinate in 2d plane
- $W \rightarrow$ Width of rectangular box
- $H \rightarrow$ Height of rectangular box
- $-i \rightarrow$ image reading
- $-v \rightarrow$ video reading
- $-c \rightarrow$ web camera reading
- $-o \rightarrow$ storing output

1. INTRODUCTION

The project in Python requires one to have basic knowledge of python programming and OpenCV library. We will be using following libraries:

- **OpenCV:** A strong library for machine learning
- **Imutils:** Library used for image processing
- **NumPy:** Python library used for scientific computing and storing image in a numpy array.
- **Argparse:** Used to give input in command line.

1.1 OBJECTIVE:

The prime objective of this project is to study the human congestion and traffic patterns in major circles, parks or restaurants of a city or town. This would help us study congestion patterns and will help us device our plans such that we avoid being stuck in places with high density. This project can also be used in public places such as railway stations, bus stations, etc. to help direct people from sparse roots in order to avoid a situation like stampede.

The objective of our project is to provide a systematic approach in order to deal with situations of jams due to high population in certain places. It is similar to the function of GPS oriented traffic management systems that help provide faster paths due to high vehicle congestion on a road. But in our project, we aim at covering the indoor coverage of a particular place and provide similar function by detection human congestion in that place.

1.2 MOTIVATION:

The motivation for developing this project was the current scenario of pandemic in which social distancing has been one of the prominent tools to prevent the virus' spread. While devising the usage of this prototype, it was realized that this project cannot only be used for ensuring social distancing in public places, but could also be used for studying the patterns of high congestion in common places to suggest the users about when is the best time to go out. One of the other motivation behind this project was 2017 Mumbai stampede which, on record, left more than 23 people dead and more than 35 people injured due to poor management of human traffic. This project aims at minimizing these risks and provide better management of human traffic.

1.3 BACKGROUND:

In this python project, we are going to build the Human Detection and Counting System through Web Camera, or image/video input. This is an intermediate level deep learning project on computer vision, which helped us to grab theory basics of Artificial Intelligence, Deep Learning and Machine Learning.

The system will tell us how many people are present in the screen at any given time that is it can give real time analysis of human detection. Our project will be using basics of Deep Learning, through which we will be defining different sort of functions to capture the human body in real time and then following the capture we will be counting the number of humans detected.

We will be dividing the given image, video or live webcam feed to the model as an input which will then pass through some custom-made functions which were created through libraries like OpenCV, NumPy etc. These functions will be dividing the input into different frames to analyse and show the output.

The project uses Histogram of Oriented Gradients (HOG) model to identify human boundary and then finds the pixel density to identify if the identified boundary is actually human. In this way we will be able to identify for human like figures in an image. We will then use the number of humans in a particular frame to add upon and count them in subsequent frames of a video. This will help us enable our human counting system which is implemented by simple addition of number of humans in various frames split across a given time frame. The time frame delay is added to remove the redundant counting of same person (human) over multiple frames of same video.

KeyWords:

Face Detection, Face Recognition, OpenCV, Imutils, NumPy, Counting Systems.

2. Applications of Object Detection

Object Detection is being widely used in the industry right now. Anyone harbouring ambitions of working in computer vision should know these applications by heart.

The use cases of object detection range from personal security to automated vehicle systems. It is used in applications like Self-Driving Cars, Face Detection and Recognition, Action Recognition, Object Counting etc...

Some of the most commonly used algorithms for object detection include Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), Regional-based Convolutional Neural Networks (RCNN), Fast Regional-based Convolutional Neural Network (Fast RCNN), You Only Look Once (YOLO).

3. Challenges in Performing Real-Time Object Detection

Real-time object detection models should be able to sense the environment, parse the scene and finally react accordingly. The model should be able to identify what all types of objects are present in the scene. Once the type of objects has been identified, the model should locate the position of these objects by defining a bounding box around each object. The major challenge with this approach is to clearly define a human like boundary, which resembles very much like any other concrete object boundary. This issue of recognising human like boundary can also be resolved by studying gradient variations of the particular shape. This, in turn, raise another issue of quickly identifying these gradient variations in real time. The analysis of gradient variation adds computational complexity in our algorithm and hence using it in real-time becomes time expensive.

4. Literature Review

<i>Authors</i>	<i>Title (Study)</i>	<i>Methodology used/ Implementation</i>
1) Tejashree Dhawle, Urvashi Ukey, Rakshandha Choudante	Face Recognition using OpenCV and python	<p>The OpenCV helps in recognizing the frontal face of the person and also creates XML documents for several areas such as the parts of the body.</p> <p>Deep learning evolved lately in the process of the recognition systems. Hence deep learning along with the face recognition together work as the deep metric learning systems. In short deep learning in face detection and recognition will broadly work on two areas the first one being accepting the solitary input image or any other relevant picture and the second being giving the best outputs or the results of the image of the picture.</p>
2) Manav Bansal, Sohan Garg	Facial Detection & Recognition Using Open CV library	<p>For simplicity, the face recognition system presented in this paper is Eigenfaces using grayscale images.</p> <p>The paper demonstrates how easily it is to convert color images to grayscale (also called 'grayscale'), and then to apply Histogram Equalization as a very simple method of automatically standardizing the brightness and contrast of your facial images. For better results, you could use color face recognition (ideally with color histogram fitting in HSV or another color space instead of RGB), or apply more processing stages such as edge enhancement, contour detection, motion detection, etc</p>
3) Manav Bansal	FACE RECOGNITION IMPLEMENTATION	OpenCV accompanies a coach just as

	NTATIONON RASPBERRYPI USING OPENCV AND PYTHON	identifier. In the event that you need to prepare your own classifier for any article like vehicle, planes and so on you can utilize OpenCV to make one. Its full subtleties are given here: Cascade Classifier Training. Here we will manage detection. OpenCV as of now contains numerous pre-prepared classifiers for face, eyes, grin and so forth. Those XML records are put away in opencv/information/haarcascades/organizer. How about we make face and eye locator with OpenCV.
4) MAMATA S.KALAS	REAL TIME FACE DETECTION AND TRACKING USING OPENCV	Different methods and algorithms of face detection have been reviewed in this paper. The choice of a face detection method in any study should be based on the particular demands of the application. None of the current methods is the universal best for all applications. Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector. A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then
5) Mahila Khan Sudeshma Chakraborty	Face Detection and Recognition using Open CV	build a camera-based real-time face recognition system and set an algorithm by developing programming on Open CV, Haar Cascade, Eigenface, Fisher Face, LBPH, and Python

Table 1. Literature Review

5. Methodology

The **Histogram of Oriented Gradient (HOG)** is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient oriented in localized portions of an image. This method is similar to that of edge-oriented histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients of edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

6. Tools and Requirements

6.1 Software Requirements

1. Visual Studio Code for Python Programming.
2. Libraries used:

OpenCV: A strong library used for machine learning and image recognition

Imutils: Used for image processing

NumPy: Used for scientific computing as image is stored in a numpy array

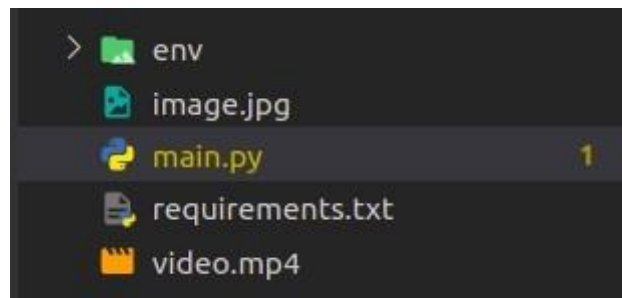
Argparse: Used to give input in command line.

6.2 Dataset/Tools Used

- a. No dataset is used in our project as this is a real-time project.
- b. Our project is inspired from a few GitHub repositories. One such link is: <https://github.com/saimj7/People-Counting-in-Real-Time>. Although, our project uses a very different approach, as we are using the Histogram of Oriented Gradients(HOG) but the repository mentioned above uses a SSD approach to solve the same problem.

7. IMPLEMENTATION

This is how the project directory looks like:



Now let's try to understand the codebase, these are the libraries which we are going to use

```
main.py > ...  
1  import cv2  
2  import imutils  
3  import numpy as np  
4  import argparse
```

Creating a model using HOGDescriptor with SVM to detect humans

```
110  HOGCV = cv2.HOGDescriptor()  
111  HOGCV.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())  
112
```

Here, the actual magic will happen.

Video: A video combines a sequence of images to form a moving picture. We call these images as Frame. So in general we will detect the person in the frame. And show it one after another that it looks like a video.

That is exactly what our Detect() method will do. It will take a frame to detect a person in it. Make a box around a person and show the frame and return the frame with person bounded by a green box.

```
6  def detect(frame):  
7      bounding_box_coordinates, weights = HOGCV.detectMultiScale(frame, winStride = (4, 4), padding = (8, 8), scale = 1.03)  
8  
9      person = 1  
10     for x,y,w,h in bounding_box_coordinates:  
11         cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)  
12         cv2.putText(frame, f'person {person}', (x,y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1)  
13         person += 1  
14  
15     cv2.putText(frame, 'Status : Detecting ', (40,40), cv2.FONT_HERSHEY_DUPLEX, 0.8, (255,0,0), 2)  
16     cv2.putText(frame, f'Total Persons : {person-1}', (40,70), cv2.FONT_HERSHEY_DUPLEX, 0.8, (255,0,0), 2)  
17     cv2.imshow('output', frame)  
18  
19     return frame
```

Everything will be done by detectMultiScale(). It returns 2-tuple.

1. List containing Coordinates of bounding Box of person. Coordinates are in form X, Y, W, H.

Where X,Y are starting coordinates of box and w, h are width and height of box respectively.

2. Confidence Value that it is a person.

Now, We have our detect method. Let's Create a Detector.

HumanDetector() method

There are two ways of getting Video.

1. Web Camera .
2. Path of file stored.

In this deep learning project, we can take images also. So our method will check if a path is given then search for the video or image in the given path and operate.

Otherwise, it will open the webCam.

```
79 def humanDetector(args):
80     image_path = args["image"]
81     video_path = args['video']
82     if str(args["camera"]) == 'true' : camera = True
83     else : camera = False
84
85     writer = None
86     if args['output'] is not None and image_path is None:
87         writer = cv2.VideoWriter(args['output'],cv2.VideoWriter_fourcc(*'MJPG'), 10, (600,600))
88
89     if camera:
90         print('[INFO] Opening Web Cam.')
91         detectByCamera(output_path,writer)
92     elif video_path is not None:
93         print('[INFO] Opening Video from path.')
94         detectByPathVideo(video_path, writer)
95     elif image_path is not None:
96         print('[INFO] Opening Image from path.')
97         detectByPathImage(image_path, args['output'])
```


DetectByCamera() method

```
49 def detectByCamera(writer):
50     video = cv2.VideoCapture(0)
51     print('Detecting people...')
52
53     while True:
54         check, frame = video.read()
55
56         frame = detect(frame)
57         if writer is not None:
58             writer.write(frame)
59
60         key = cv2.waitKey(1)
61         if key == ord('q'):
62             break
63
64     video.release()
65     cv2.destroyAllWindows()
```

cv2.VideoCapture(0) passing 0 in this function means we want to record from a webcam. **video.read()** read frame by frame. It returns a check which is True if this was able to read a frame otherwise False.

Now, For each Frame, we will call **detect()** method. Then we write the frame in our output file.

DetectByPathVideo() method

This method is very similar to the previous method except we will give a path to the Video.

First, we check if the video on the provided path is found or not.

Note – A full path must be given.

```

21 def detectByPathVideo(path, writer):
22
23     video = cv2.VideoCapture(path)
24     check, frame = video.read()
25     if check == False:
26         print('Video Not Found. Please Enter a Valid Path (Full path of Video Should be Provided).')
27         return
28
29     print('Detecting people...')
30     while video.isOpened():
31         #check is True if reading was successful
32         check, frame = video.read()
33
34         if check:
35             frame = imutils.resize(frame , width=min(800,frame.shape[1]))
36             frame = detect(frame)
37
38             if writer is not None:
39                 writer.write(frame)
40
41             key = cv2.waitKey(1)
42             if key== ord('q'):
43                 break
44         else:
45             break
46     video.release()
47     cv2.destroyAllWindows()

```

DetectByPathImage() method

This method is used if a person needs to be detected from an image.

```

67 def detectByPathImage(path, output_path):
68     image = cv2.imread(path)
69
70     image = imutils.resize(image, width = min(800, image.shape[1]))
71
72     result_image = detect(image)
73
74     if output_path is not None:
75         cv2.imwrite(output_path, result_image)
76
77     cv2.waitKey(0)
78     cv2.destroyAllWindows()

```

Argparse() method

The function argparse() simply parses and returns as a dictionary the arguments passed through your terminal to ourscript. There will be Three arguments within the Parser:

- 1.**Image:** The path to the image file inside your system

2.Video: The path to the Video file inside your system

3.Camera: A variable that if set to 'true' will call the cameraDetect() method.

```
99 def argsParser():
100     arg_parse = argparse.ArgumentParser()
101     arg_parse.add_argument("-v", "--video", default=None, help="path to Video File ")
102     arg_parse.add_argument("-i", "--image", default=None, help="path to Image File ")
103     arg_parse.add_argument("-c", "--camera", default=False, help="Set true if you want to use the camera.")
104     arg_parse.add_argument("-o", "--output", type=str, help="path to optional output video file")
105     args = vars(arg_parse.parse_args())
106
107     return args
```

Main function

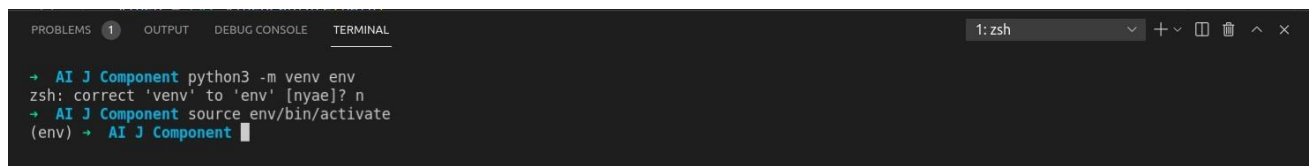
We have reached the end of our project.

Instead of declaring our model above, we can declare it in our main function.

```
109 if __name__ == "__main__":
110     HOGCV = cv2.HOGDescriptor()
111     HOGCV.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
112
113     args = argsParser()
114     humanDetector(args)
115
116
```

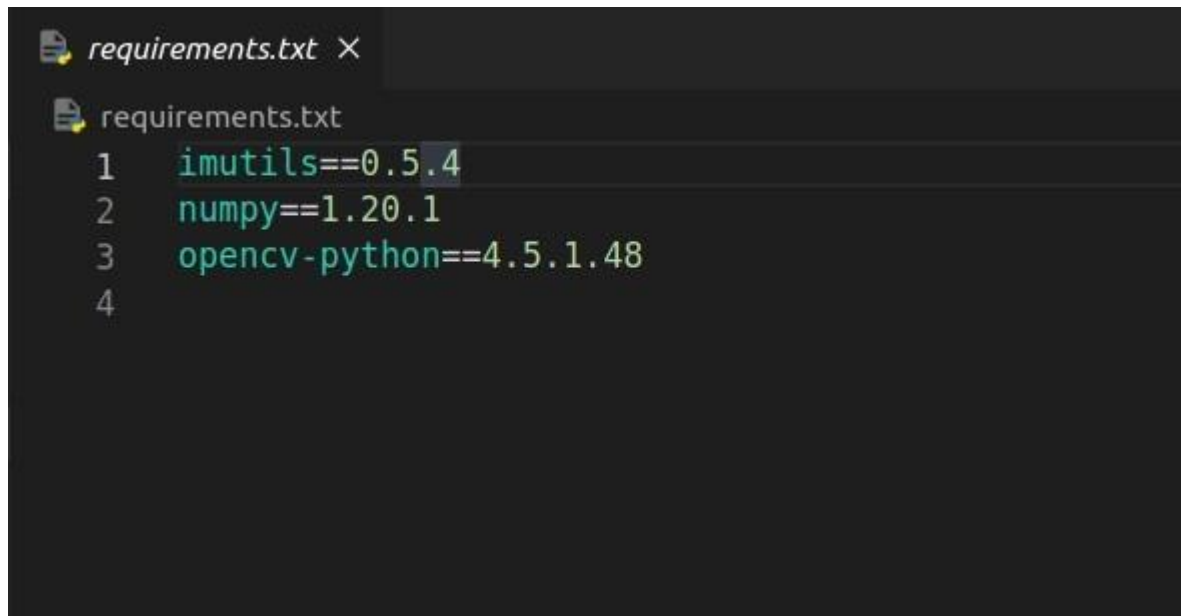
Running:

In order to run the project we have to first setup a virtual environment for the project and activate it:

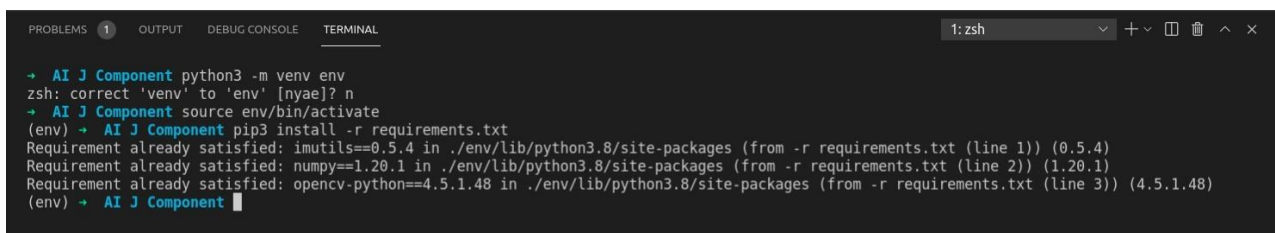


```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL 1: zsh
→ AI J Component python3 -m venv env
zsh: correct 'venv' to 'env' [nyae]? n
→ AI J Component source env/bin/activate
(env) → AI J Component
```

Now we have to download all the dependencies of the project mentioned in requirements.txt file:



```
requirements.txt
1 imutils==0.5.4
2 numpy==1.20.1
3 opencv-python==4.5.1.48
4
```



```
1: zsh
→ AI J Component python3 -m venv env
zsh: correct 'venv' to 'env' [nyae]? n
→ AI J Component source env/bin/activate
(env) → AI J Component pip3 install -r requirements.txt
Requirement already satisfied: imutils==0.5.4 in ./env/lib/python3.8/site-packages (from -r requirements.txt (line 1)) (0.5.4)
Requirement already satisfied: numpy==1.20.1 in ./env/lib/python3.8/site-packages (from -r requirements.txt (line 2)) (1.20.1)
Requirement already satisfied: opencv-python==4.5.1.48 in ./env/lib/python3.8/site-packages (from -r requirements.txt (line 3)) (4.5.1.48)
(env) → AI J Component
```

Now we are going to run the project, I will be using video to demonstrate how our code works:



```
1: python3
(env) → AI J Component python3 main.py -v video.mp4
[INFO] Opening Video from path.
Detecting people...
█
```

8. RESULTS AND CONCLUSION

While using camera:

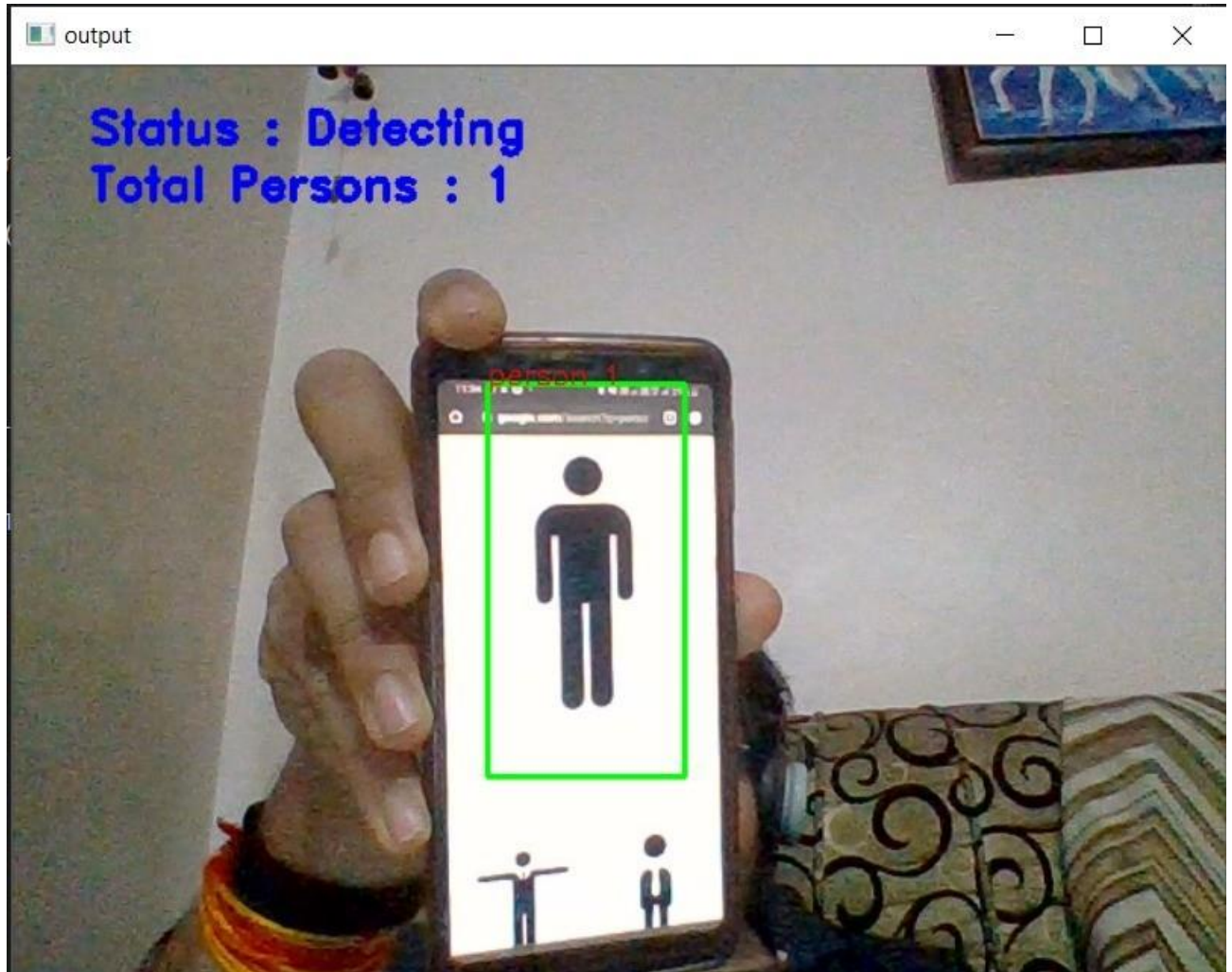


Fig 8.1 Human Detection while using Camera

While using image:



Fig 8.2 Human Detection while using Image

While using video:

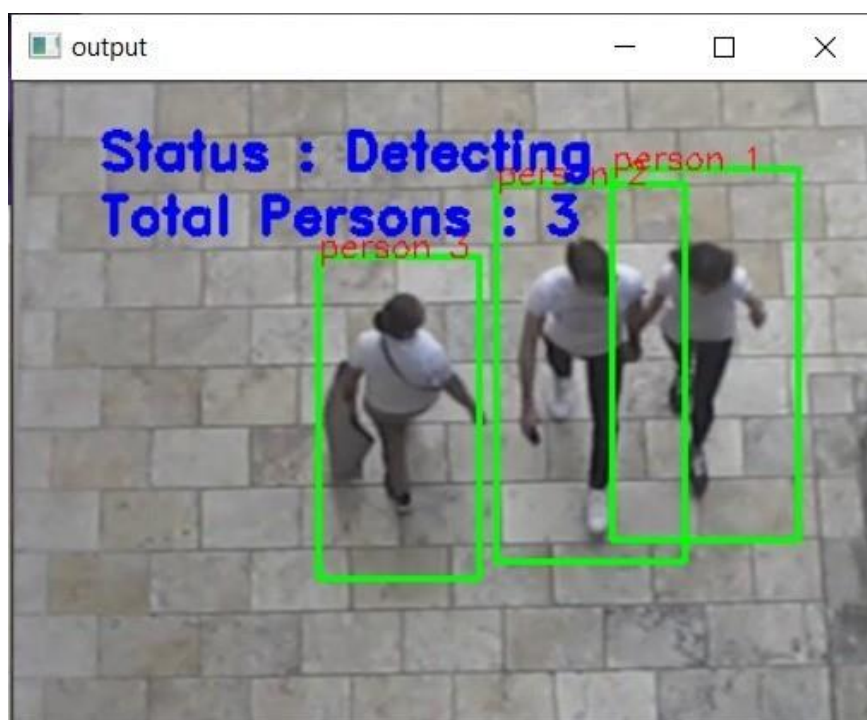


Fig 8.3 Human Detection while using Video

Expected Results

We expect to build the Human Detection and Counting System through Webcam or we can give your own video or images. This is an intermediate level deep learning project on computer vision, which will help us to master the concepts.

The system will tell how many people are present in the screen at any given time that is it can give real time analysis. We expect the system to be used in fields like proximity alert systems, intrusion alarm system, etc.

9 . REFERENCES

- [1] M. Khan, S. Chakraborty, R. Astya and S. Khepra, "Face Detection and Recognition Using OpenCV," 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 2019, pp. 116-119, doi: 10.1109/ICCCIS48478.2019.8974493.
- [2] Tejashree Dhawle, Urvashi Ukey, Rakshandha Choudante , “Face Detection and Recognition using OpenCV and Python”, Department of Computer Engineering, Dr. Babasaheb Ambedkar Technological University Raigad, India , 2020.
- [3] Manav Bansal , “Face Recognition Implementation on Raspberrypi Using Opencv and Python”, Sir Chhotu Ram Institute of Engineering and Technology,CCSU MEERUT , 2020.
- [4] MAMATA S.KALAS , “REAL TIME FACE DETECTION AND TRACKING USING OPENCV” , Department of IT, KIT’S College of Engg., Kolhapur, Volume-2, Issue-1, 2014.
- [5] P Y Kumbhar¹ , Mohammad Attaullah² , Shubham Dhere³ , Shivkumar Hipparagi , ” Real Time Face Detection and Tracking Using OpenCV”, Department of Electronics and Telecommunication Engineering, Walchand Institute of Technology, Solapur.

[6] Manav Bansal, Sohan Garg ,”Facial Detection & Recognition Using Open CV “library,
Sir Chhotu Ram Institute of Engineering and Technology,CCSU MEERUT , 2016