Department of Electronics & Communication Engineering

MANIT Bhopal

Minor Project Report (EC-326)

Group No. 26

Project Title: Real Time Human Detection and Counting

PROJECT MEMBERS

S. No.	Scholar No.	Name	Signature
1	181114001	Lakshta Kori	[illegible]
2	181114007	Toshika Choyal	[illegible]
3	181114013	Anushree Malviya	[illegible]
4	181114030	Abhishek Garg	Abhishek Garg
5	181114032	Utkarsh Gupta	[illegible]

MENTOR: Dr. Bhavna Shrivastava

INDEX

INTRODUCTION

- Over the recent years, detecting human beings in a video scene of a surveillance system is attracting more attention due to its wide range of applications in abnormal event detection, human gait characterization, person counting in a dense crowd etc.

Human detection is the key technology of intelligent video surveillance, especially for static images. Despite various difficulties, the development of human detection has made a number of achievements over the years.

The detection process generally occurs in two steps: object detection and object classification. Object detection could be performed by background subtraction, optical flow and spatio-temporal filtering.

Background subtraction is a popular method for object detection where it attempts to detect moving objects from the difference between the current frame and a background frame in a pixel-by-pixel or block-by-block fashion.

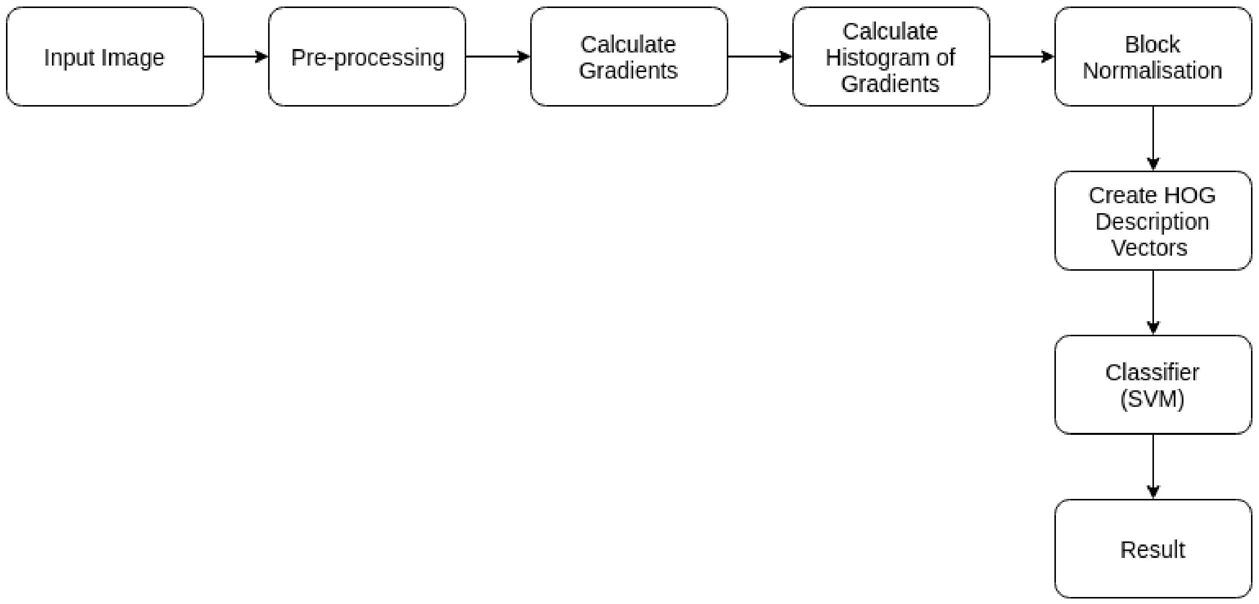
The object classification methods could be divided into three categories: shape-based, motion-based and texture-based. Shape-based approaches first describe the shape information of moving regions such as points, boxes and blobs. Then, it is commonly considered as a standard template-matching issue.

However, the articulation of the human body and the differences in observed viewpoints lead to a large number of possible appearances of the body, making it difficult to accurately distinguish a moving human from other moving objects using the shape-based approach. This challenge could be overcome by applying part-based template matching. Texture-based methods such as histograms of oriented gradients (HOG) use high dimensional features based on edges and use support vector machines (SVM) to detect human regions. This technique counts the occurrences of gradient orientation in localized portions of an image, is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

OBJECTIVE

To build the Human Detection and Counting System using Histogram of Oriented Gradients (HOG) and Pretrained Support Vector Machine (SVM) in Python.

BLOCK DIAGRAM



- Input in our case is the Image/Video/Webcam Feed which is taken as input through the terminal when executing the script. The input is passed as arguments with the help of **argParse** library.

- Pre-processing the input is very crucial before passing it into our functions. Pre-processing includes resizing, gray scaling, blurring (if necessary). For blurring we used Median Blur Filter which is better than Gaussian Blur and gave us better results, but is a bit slower.

- Next, we need to define our region of interest which will be a fixed 1:2 aspect ratio sliding window. We calculate the horizontal and vertical gradient with magnitude and direction.

- The next step is to create a histogram of gradients in these cells. The histogram contains 9 bins corresponding to angles 0,20, 40 ….160. A bin is selected based on the direction. If the magnitude of current gradient comes between two bins, then we split the magnitude evenly between the two bins.

- Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations.

- To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector. And next we will calculate the size of the vector.

- This final feature vector is then passed into the binary classifier Support Vector Machine (SVM) which is pretrained for human detection.

- SVM returns us the coordinates of all the blocks which are tested positive and then we can easily mark the detection with boxes and create an algorithm for counting each detection.

FEATURE DESCRIPTOR

➢ Introduction

A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. Examples are HOG, SURF, SIFT. Typically, a feature descriptor converts an image of size width x height x 3 (channels) to a feature vector / array of length n.

In the case of the HOG feature descriptor, the input image is of size 64 x 128 x 3 and the output feature vector is of length 3780.

In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions.

➢ What are useful features?

Clearly, the feature vector is not useful for the purpose of viewing the image. But it is very useful for tasks like image recognition and object detection. The feature vector produced by these algorithms when fed into an image classification algorithm like Support Vector Machine (SVM) produce good results.

➢ How is Histogram of Oriented Gradients Calculated?

Step 1: Preprocessing

As mentioned earlier HOG feature descriptor used for pedestrian detection is calculated on a 64×128 patch of an image. Of course, an image may be of any size. Typically patches at multiple scales are analyzed at many image locations. The only constraint is that the patches being analyzed have a fixed aspect ratio. In our case, the patches need to have an aspect ratio of 1:2. For example, they can be 100×200, 128×256, or 1000×2000 but not 101×205.

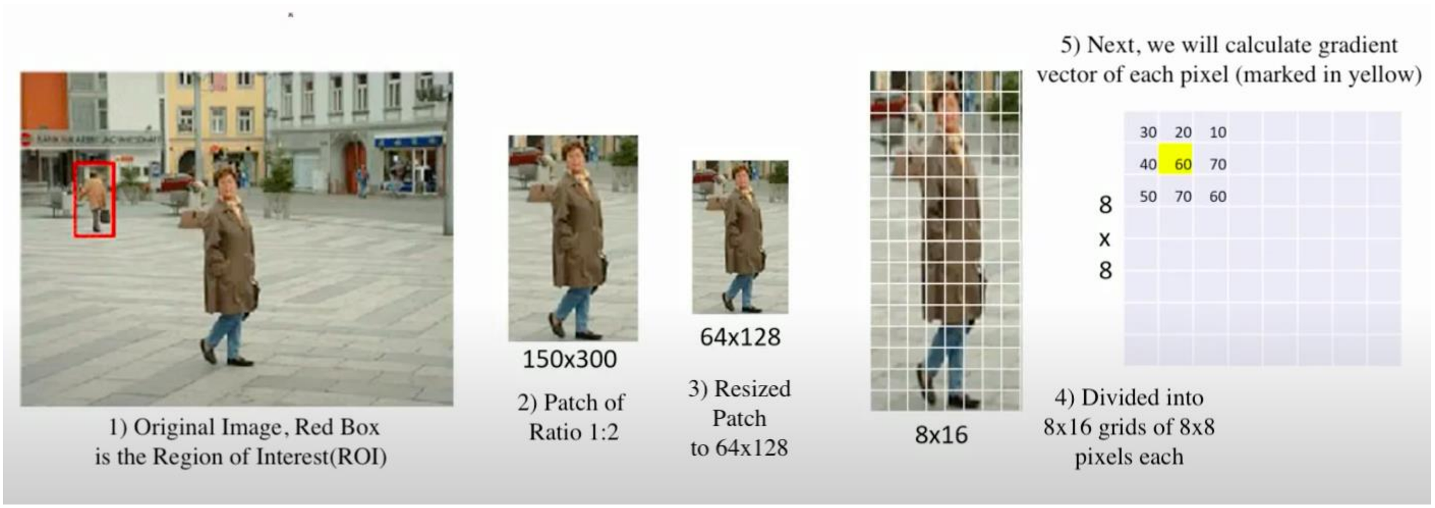

5) Next, we will calculate gradient vector of each pixel (marked in yellow)
30 20 10
40 60 70
50 70 60
8 x 8
1) Original Image, Red Box is the Region of Interest(ROI)
150x300
2) Patch of Ratio 1:2
64x128
3) Resized Patch to 64x128
8x16
4) Divided into 8x16 grids of 8x8 pixels each

Fig. 2 Steps to find Gradient

Step 2: Calculate the Gradient Images

To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients, after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.

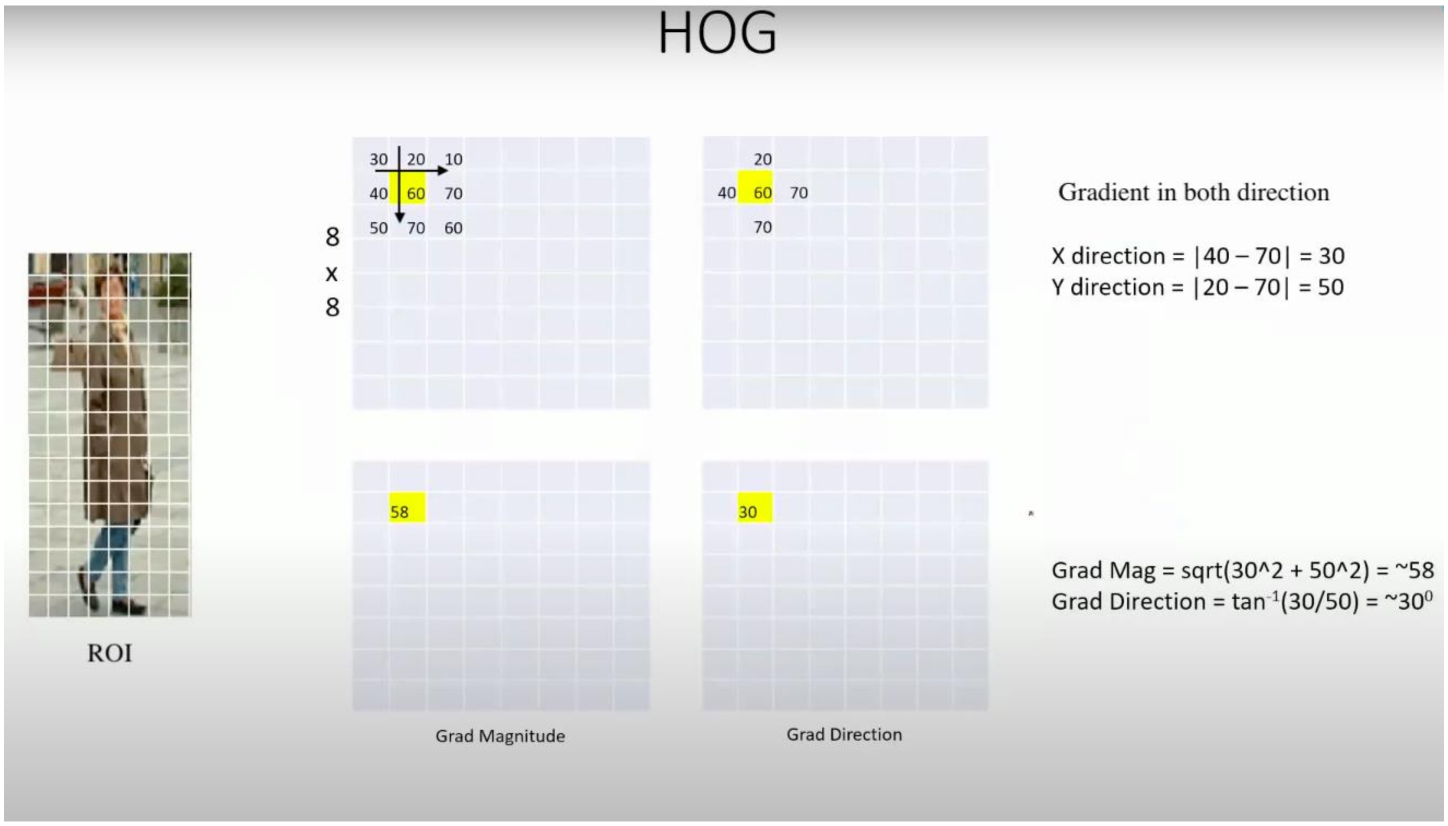
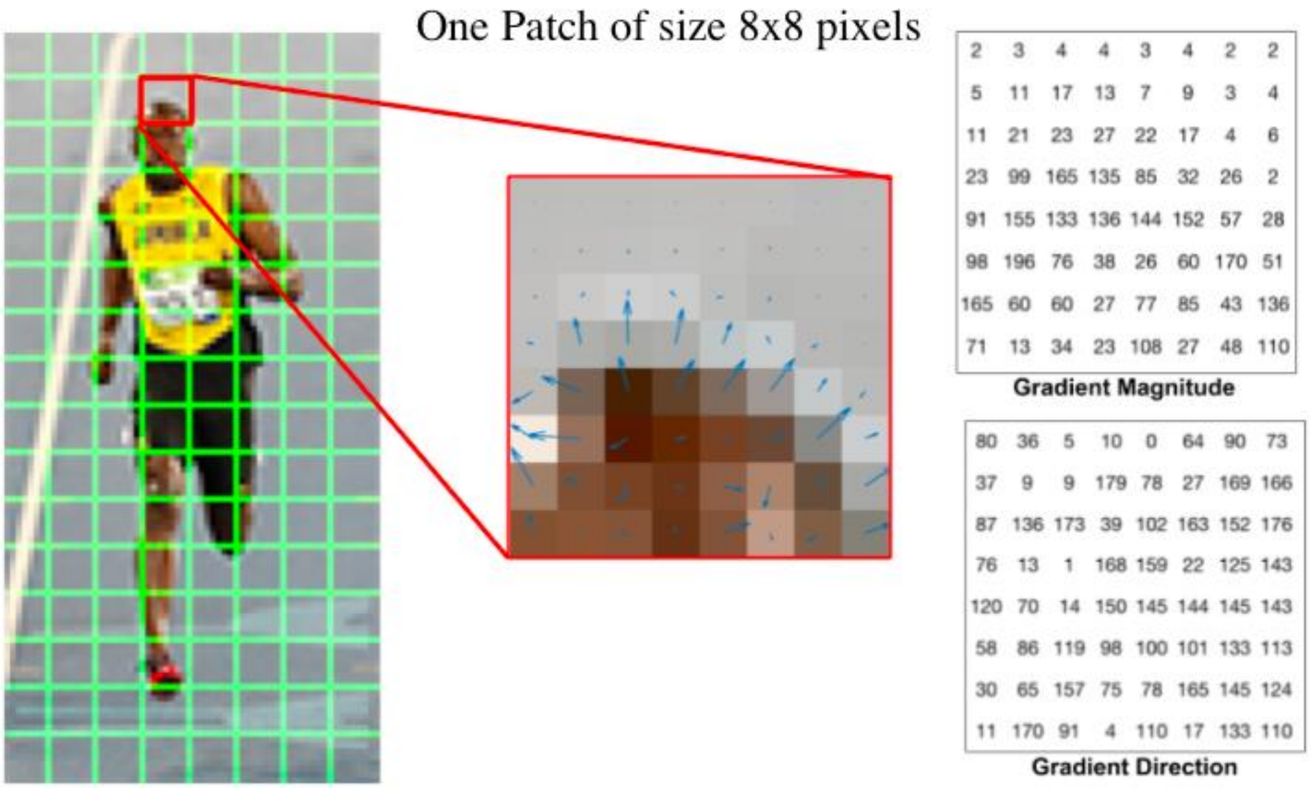

HOG
8 x 8
30 20 10
40 60 70
50 70 60
20
40 60 70
70
Gradient in both direction
X direction = |40 – 70| = 30
Y direction = |20 – 70| = 50
58
30
Grad Mag = sqrt(30^2 + 50^2) = ~58
Grad Direction = tan-1(30/50) = ~30°
ROI
Grad Magnitude
Grad Direction

Fig. 3 Gradient Magnitude and Direction Calculation

Step 3: Calculate Histogram of Gradients in 8×8 cells

- One of the important reasons to use a feature descriptor to describe a patch of an image is that it provides a compact representation. An 8×8 image patch contains 8x8x3 = 192-pixel values. The gradient of this patch contains 2 values (magnitude and direction) per pixel which adds up to 8x8x2 = 128 numbers.

- HOG was initially used for Pedestrian detection only. That is why 8×8 cells in a photo of a pedestrian scaled to 64×128 are big enough to capture interesting features (e.g., the face, the top of the head etc.)



Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Direction

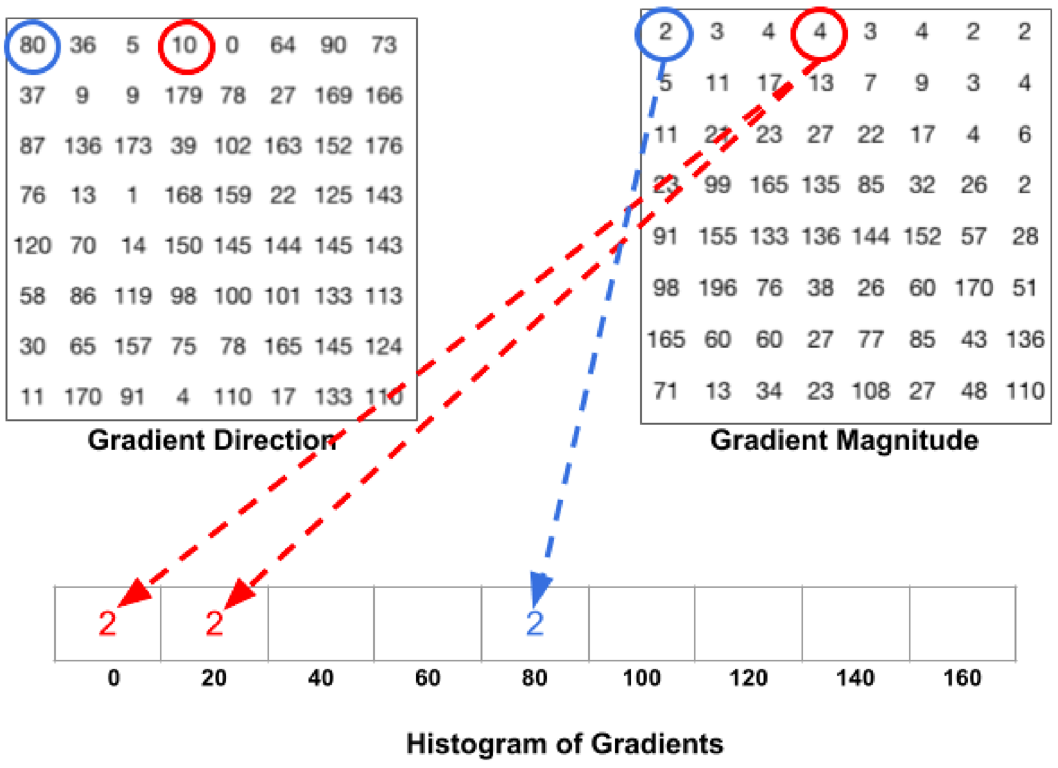
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Fig 4 Example of calculating gradient on a patch of image

- The image above shows the patch of the image overlaid with arrows showing the gradient the arrow shows the direction of gradient and its length shows the magnitude. The direction of arrows points to the direction of change in intensity and the magnitude shows how big the difference is.

- The Gradient Direction matrix stores the vector angle of each pixel (0 to 180). The next step is to create a histogram of gradients in these 8×8 cells. The Histogram contains 9 bins corresponding to angles 0, 20, 40 … 160. The following figure illustrates the process. We are looking at magnitude and direction of the gradient of the same 8×8 patch as in the previous figure.

- A bin is selected based on the direction. If the magnitude of current gradient comes between two bins, then we split the magnitude evenly between the two bins.



80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

0	20	40	60	80	100	120	140	160
2	2			2				

Histogram of Gradients

Fig 5 Creating Histogram with Gradient Matrix

HISTOGRAM CREATED BY THESE GRADIENT VECTORS IS:

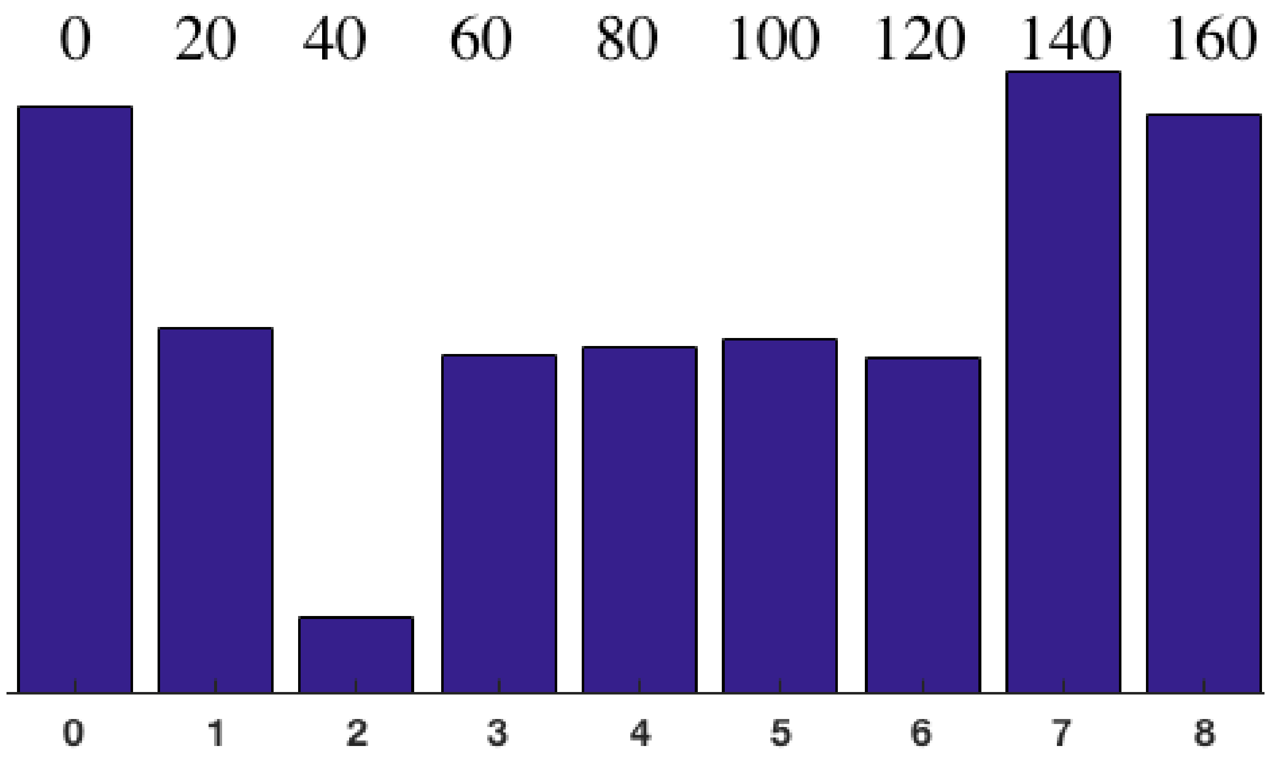


Fig 6 Histogram

Step 4: 16×16 Block Normalization

- In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half.

- Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram so they are not affected by lighting variations.

Example given below shows normalization of a vector [0, 3, -4]

$$\begin{aligned}\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|} &= \frac{1}{\sqrt{(0)^2 + (3)^2 + (-4)^2}} \begin{bmatrix} 0 \\ 3 \\ -4 \end{bmatrix} \\ &= \frac{1}{\sqrt{25}} \begin{bmatrix} 0 \\ 3 \\ -4 \end{bmatrix} \\ &= \frac{1}{5} \begin{bmatrix} 0 \\ 3 \\ -4 \end{bmatrix}\end{aligned}$$

- A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector and it can be normalized just the way a 3×1 vector is normalized.

- The window is then moved by 8 pixels and a normalized 36×1 vector is calculated over this window and the process is repeated.

Step 5: Calculate the HOG feature vector

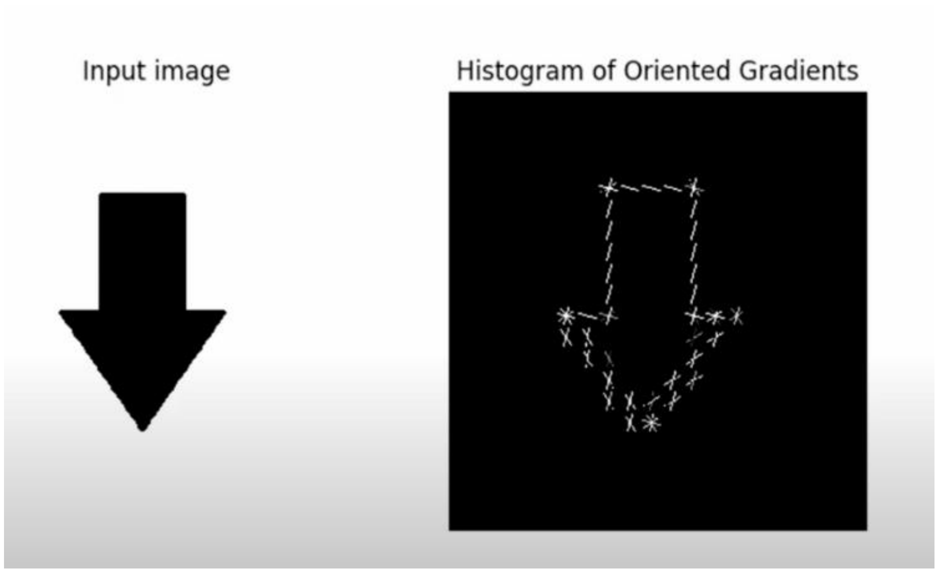
- To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector.
  Let us calculate the size of the vector

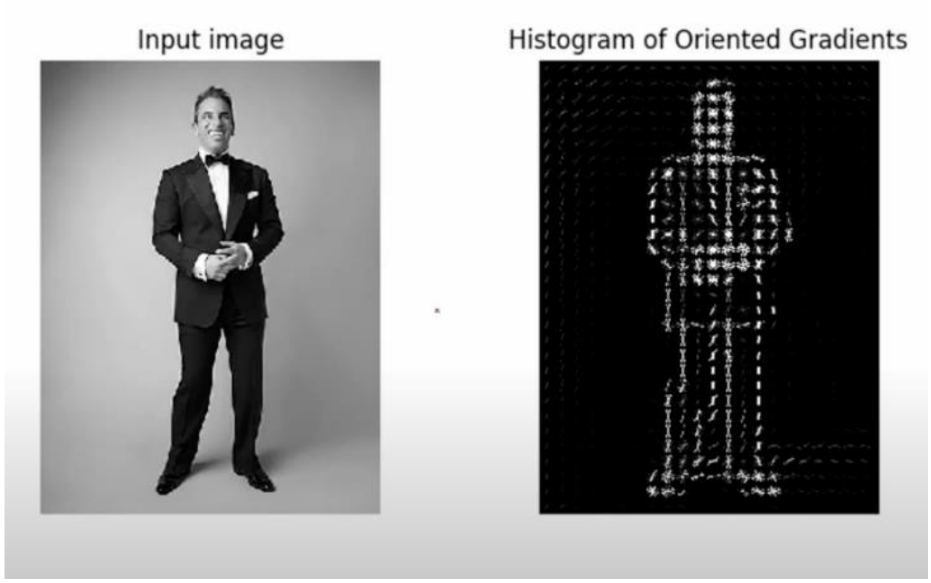
- How many positions of the 16×16 blocks do we have? There are 7 horizontals and 15 vertical positions making a total of 7 x 15 = 105 positions.

- Each 16×16 block is represented by a 36×1 vector. So, when we concatenate them all into one giant vector, we obtain a 36×105 = 3780-dimensional vector.

- For an 8x16 Patch we will end up with a 7x15 Feature Vector each having size 36.

Below are the outputs of HOG performed on different types of images:





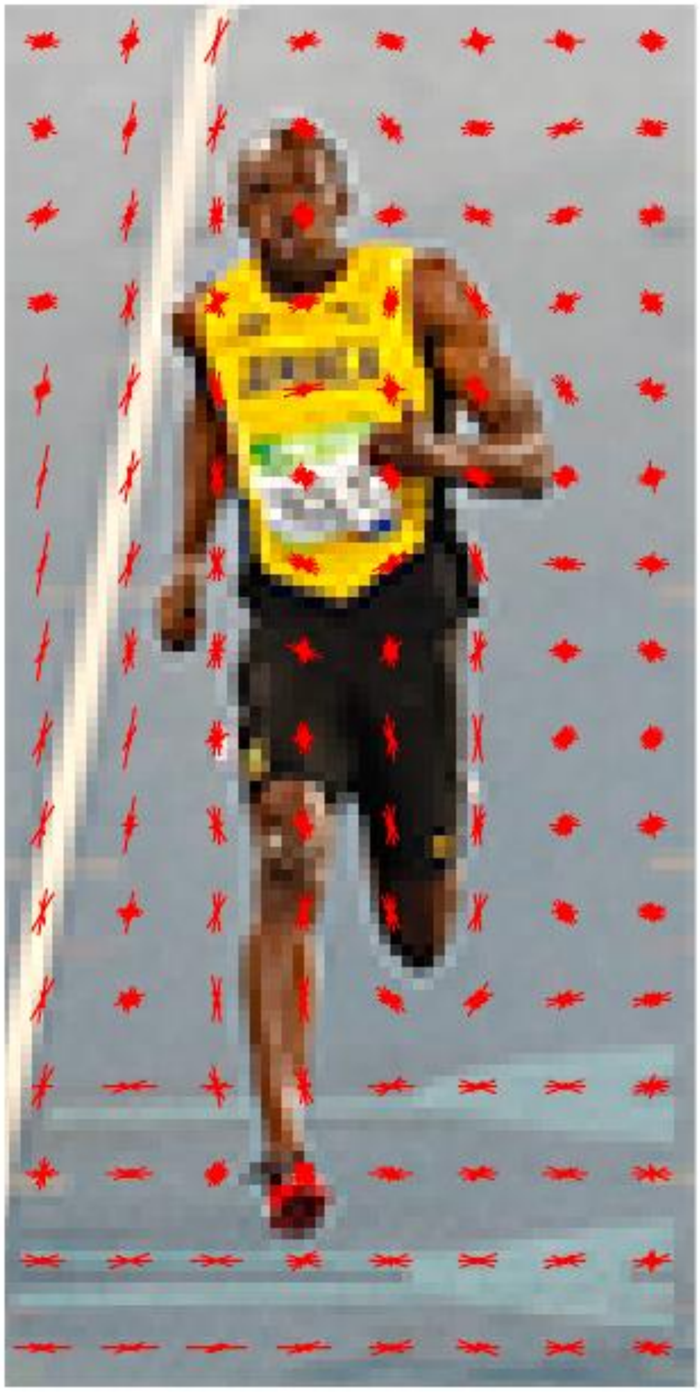


Fig 7 Histogram elements

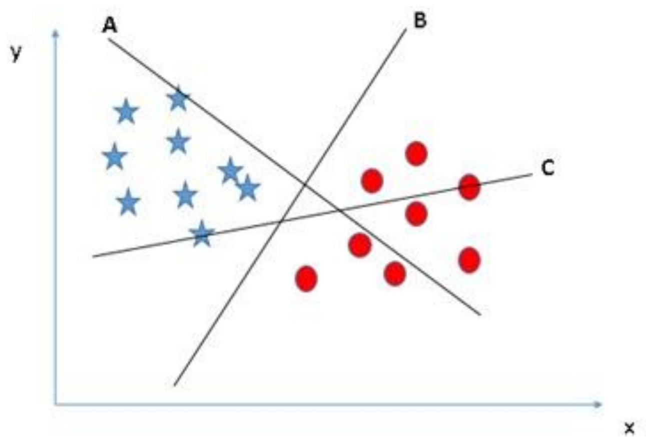
SVM

➢ Introduction

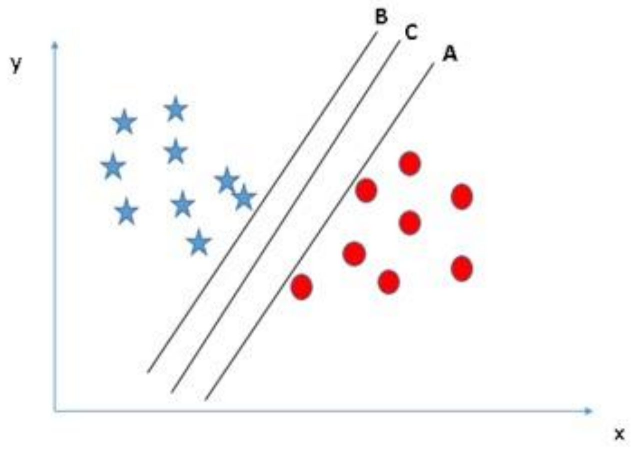
A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text.

➢ Working of SVM

The basics of Support Vector Machines and how it works are best understood with a simple example. Let's imagine we have two tags: red and blue, and our data has two features: x and y. We want a classifier that, given a pair of (x, y) coordinates, outputs if it's either red or blue. We plot our already labeled training data on a plane:



Our Labelled Data



In 2D, the best hyperplane is simply a line

➢ A support vector machine takes these data points and outputs the hyperplane (which in two dimensions it's simply a line) that best separates the tags. This line is the decision boundary: anything that falls to one side of it we will classify as blue, and anything that falls to the other as red.

- For SVM, it's the one that maximizes the margins from both tags. In other words: the hyperplane whose distance to the nearest element of each tag is the largest.

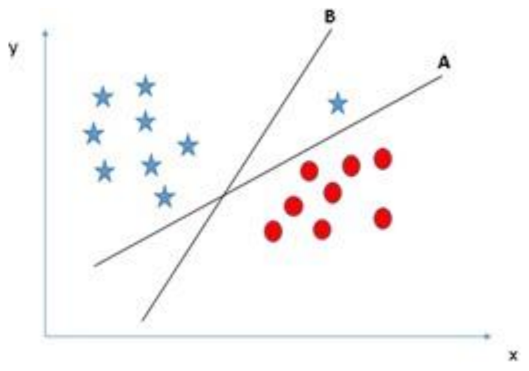
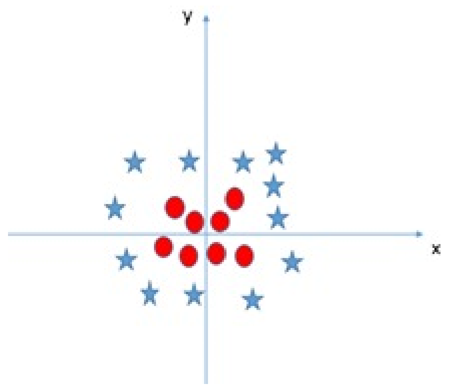


Fig : Not all Hyperplanes created equal

Nonlinear data

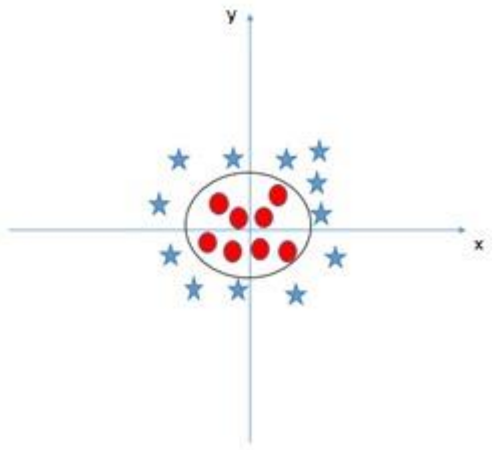
Since clearly the data was linearly separable we could draw a straight line to separate red and blue. Sadly, usually things aren't that simple.



A more complex Data Set

It's pretty clear that there's not a linear decision boundary (a single straight line that separates both tags). However, the vectors are very clearly segregated and it looks as though it should be easy to separate them.

So, here's what we'll do: we will add a 3rd dimension. Up until now we had two dimensions: x and y. We create a new z dimension, and we rule that it be calculated a certain way that is convenient for us: $z = x^2 + y^2$ (Circle)



➢ TRAINING

- Since SVM is a supervised learning algorithm we need to train it.
- For training we need 1000 images of humans and 1000 images without humans.
- Next step is to process them and find their feature vector using Histogram of Oriented Gradients.
- The next step is to train our SVM classifier model using the result of previous line.

USING OUTPUT OF HOG AS INPUT OF SVM

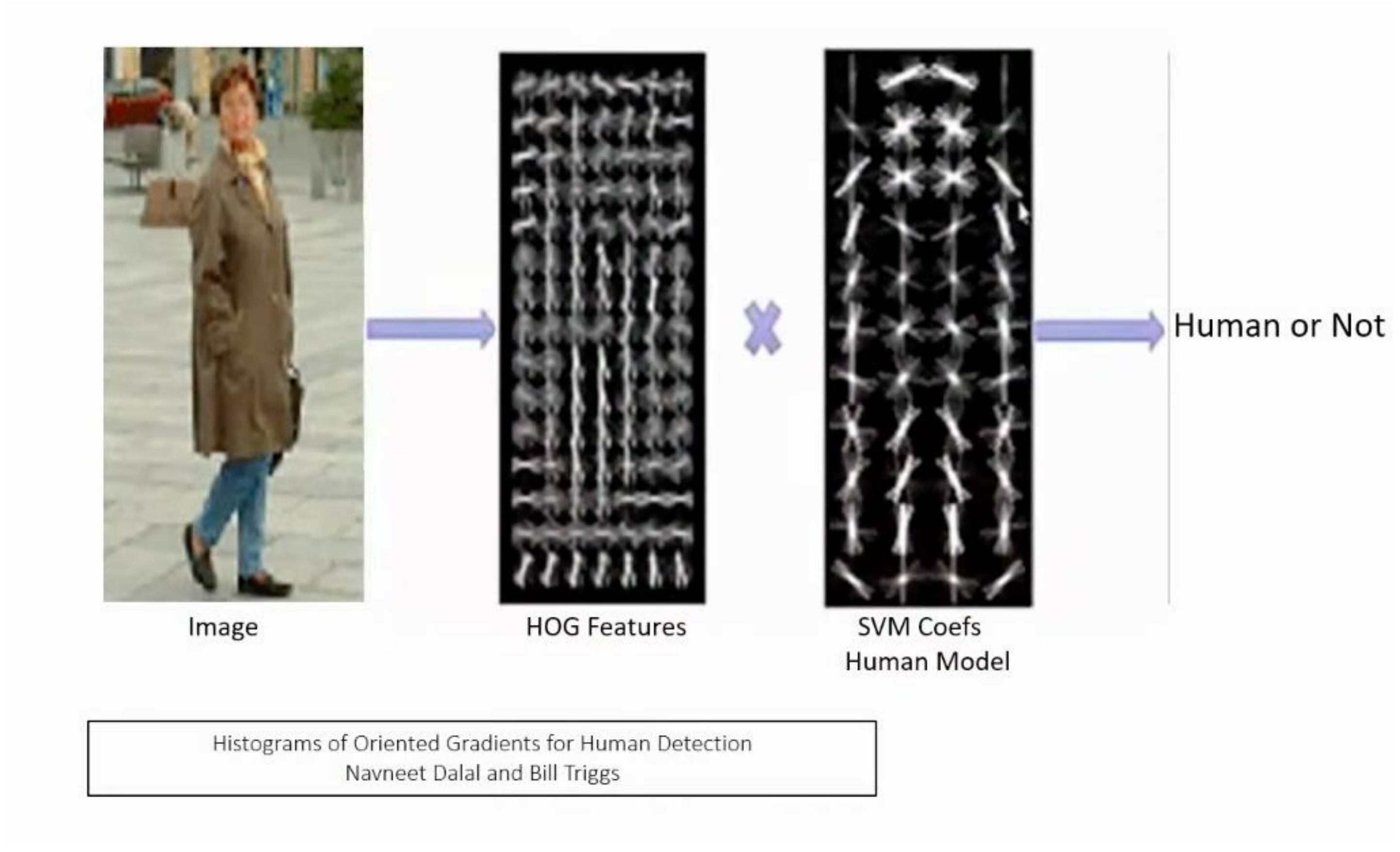


Image
HOG Features
SVM Coefs
Human Model
Human or Not
Histograms of Oriented Gradients for Human Detection
Navneet Dalal and Bill Triggs

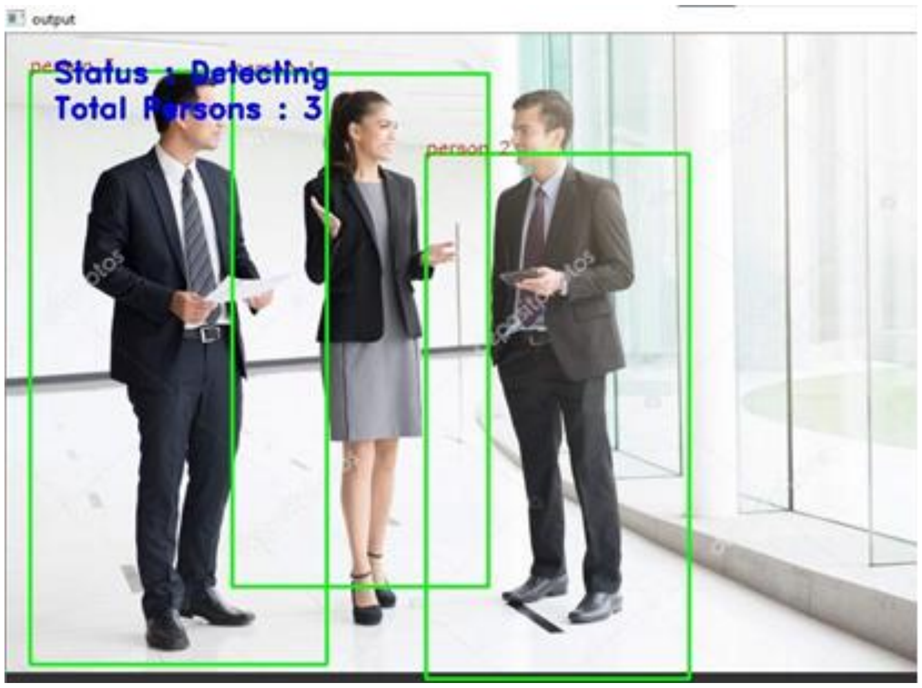
After extracting the HOG features which will be of length 3780. We multiply it's SVM Coefficients for Human Model which is also of length 3780 and then we add the bias term. Which then gives us the result (+ve or –ve).

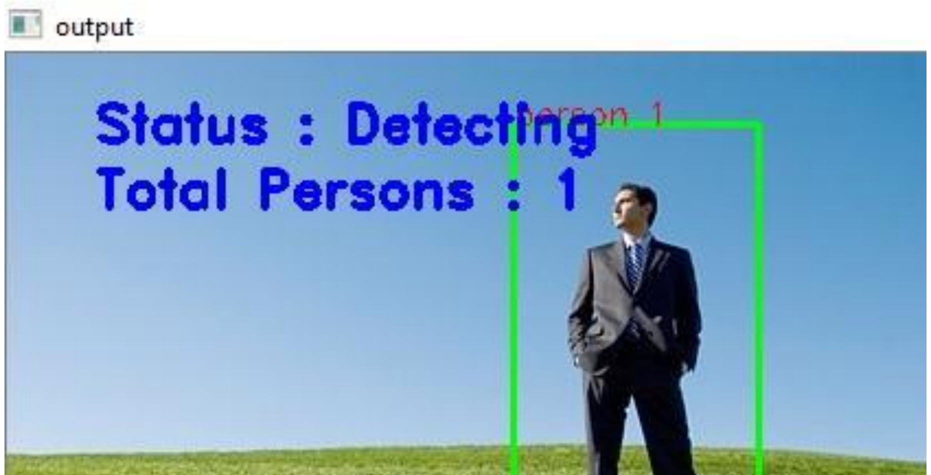
SOFTWARE USED

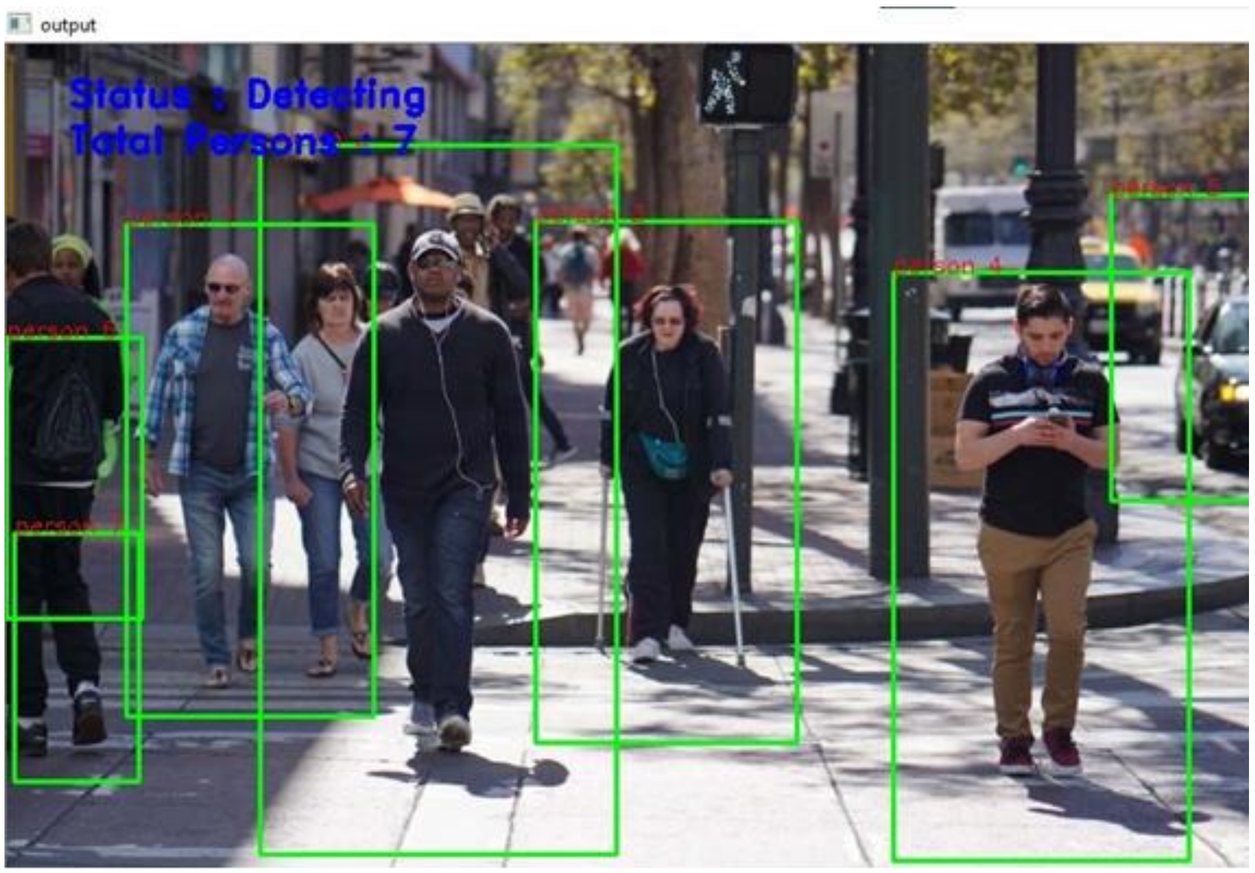
- PyCharm (Python IDE)
- OpenCV Library
- NumPy Library
- ImUtils Library

RESULTS

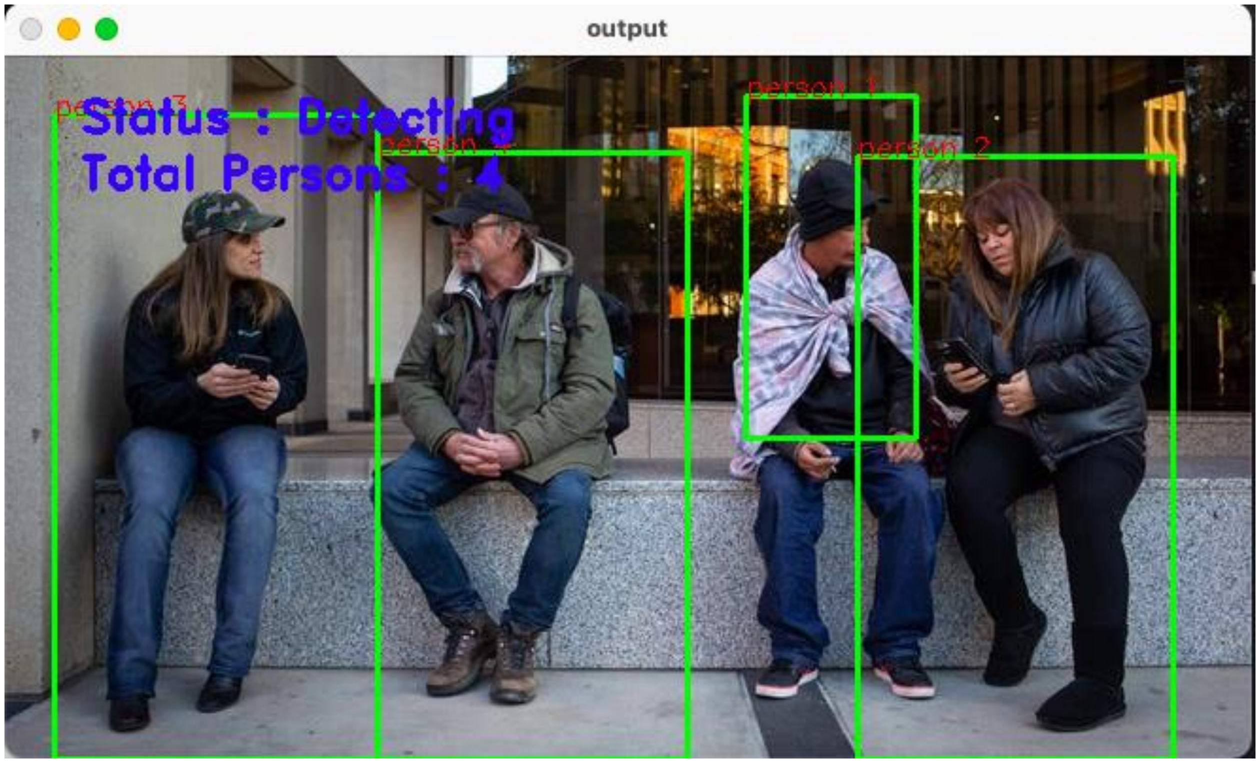
Through this project, we have successfully created an efficient people counter, where one can give in video, image, or even live camera feed as input and then detect human beings accurately.







```
Status : Detecting
Total Persons : 7
```



```
Status : Detecting
Total Persons : 4
```

OUTPUT VIDEO – Link:

https://drive.google.com/file/d/1gkx--e8WR65_nPzZBLH77OEy7gzmLoPD/view?usp=sharing

Python Code for the above video – Link:

https://drive.google.com/file/d/1vaagUhYpLoVvnLTjWkd44ITTxzl0-hp6/view?usp=sharing

CONCLUSION

- We concluded in this project that detecting humans in images and videos is a challenging problem owing to the motion of the subjects, the camera and the background and to variations in pose, appearance, clothing, illumination and background clutter.

- We developed a detector for standing and moving people in videos, testing several different motions and coding schemes and showing empirically that orientated histograms give the best overall performance.

- The resulting detector is tested on several databases including a challenging test set taken from CCTV footage and containing wide ranges of pose, motion and background variations, including moving cameras and backgrounds. Experimental results from real time video are provided.

- In this context we concentrate on the most successful and popular vector-form feature: **Histograms of Oriented Gradients (HOG).**

- The ultimate goal of this project was to established an efficient, robust and user-friendly system to detect human. Our achievement from this project is satisfactory. We aspire to do more research in the same field.

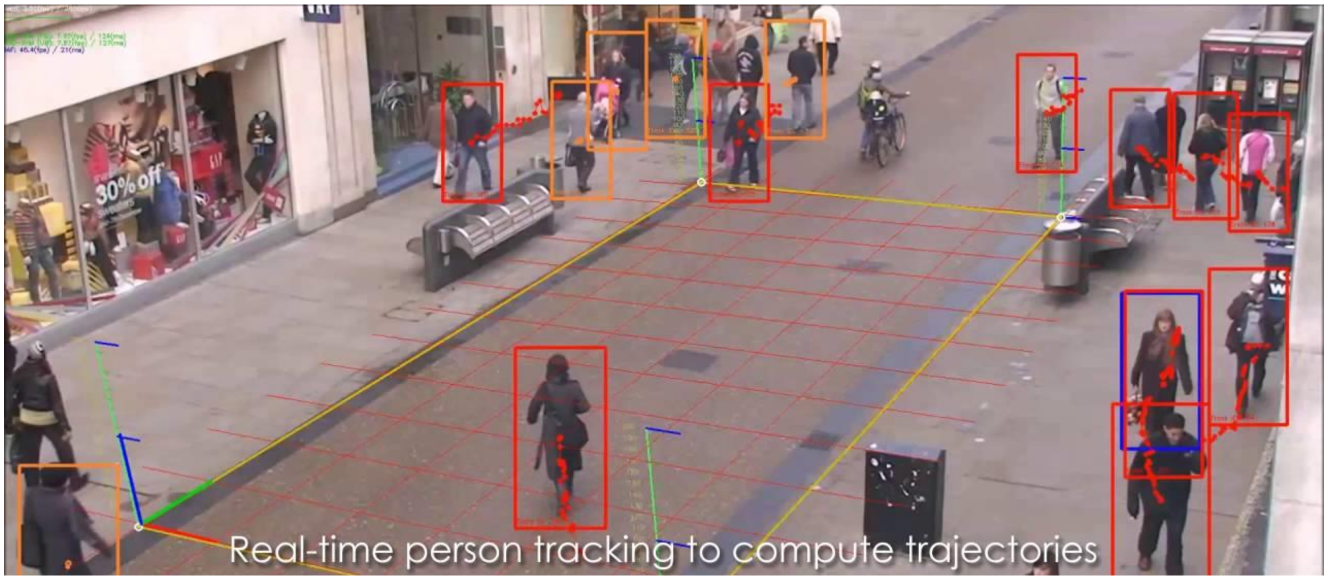
FUTURESCOPE

1. Surveillance systems

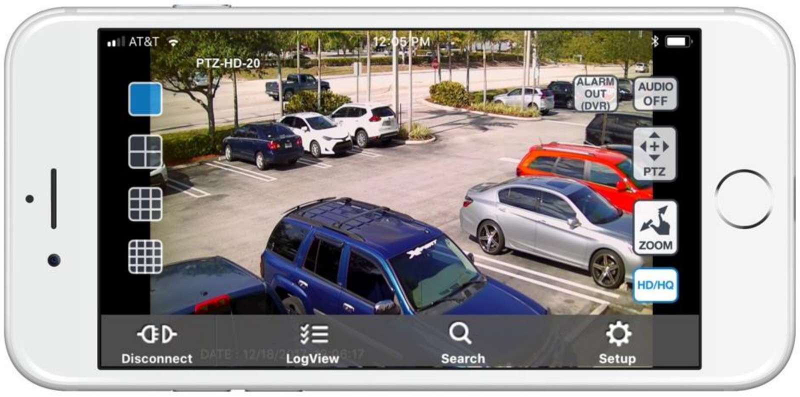
- Detecting human beings in a video scene of a surveillance system is attracting more attention due to its wide range of applications in abnormal event detection.

- Due to the limitations in the human capability to monitor simultaneous events in surveillance displays, we can use this human detection system as an automated video surveillance for tracking and classifying the objects for High level Analysis.

- It can act as the major backbone for the Computer Vision and real time pattern recognition systems.



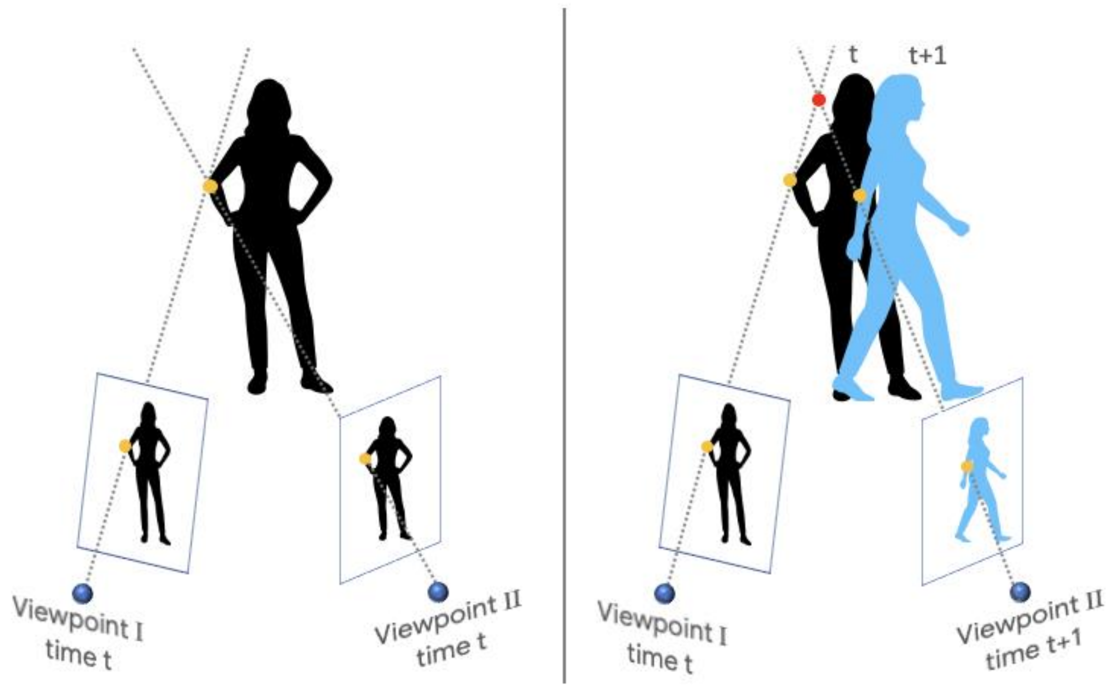
2. Remote Camera View:



➢ It refers to the relay of the captured information to a remote device such as a mobile phone over the internet through an App.

➢ As our security camera and human detection system comes with a built in web server that is always functional when the recorder is power up which will allow viewing over Local Area Network (LAN).

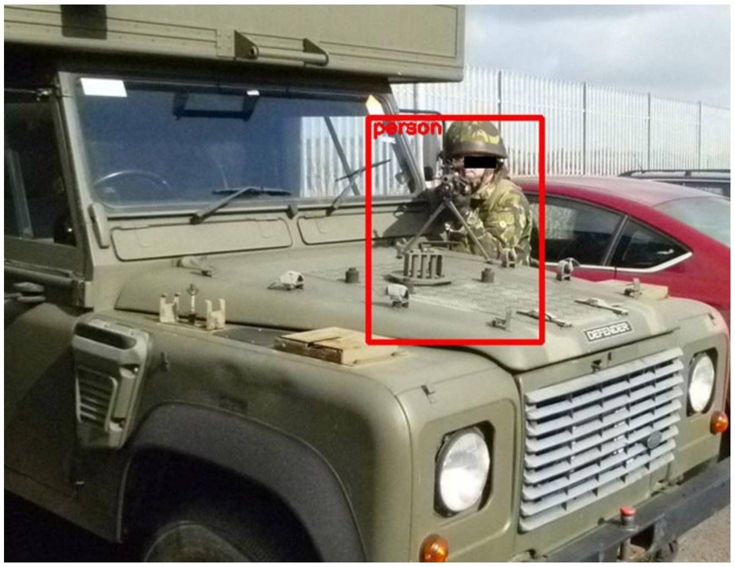
3. Computer Vision Systems



- In future much of our work will be going to be done by the robots and computers, so as to make computer vision close to human vision, it can only be done by using stereo cameras and detection systems.

- In this detection system the computer vision system is developed using the stereo camera system for measuring object distances.

4. A Military Equipment



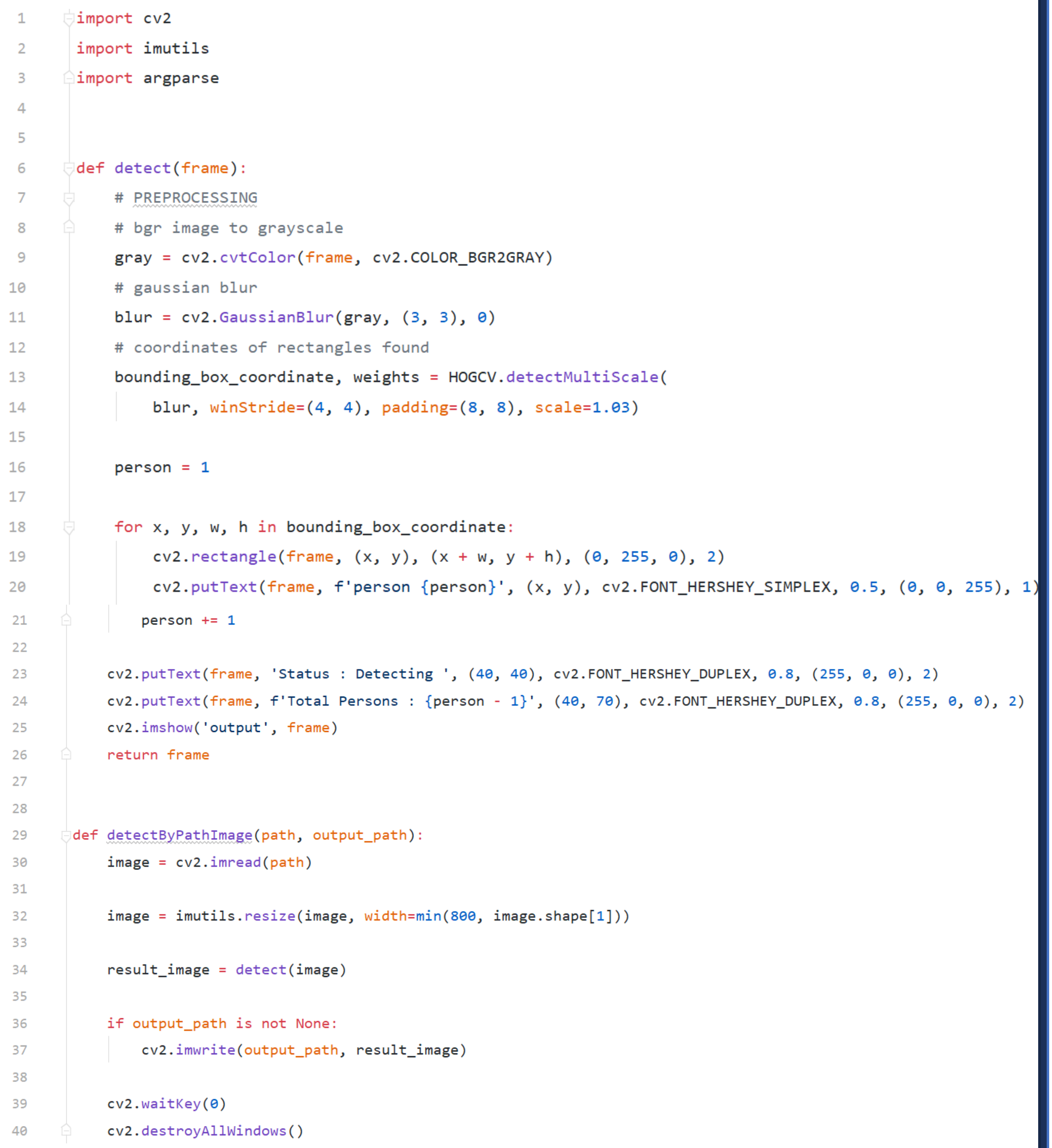
- This project can also be used for military purpose of threat detection kit which is used to identify and detection unwanted people.

- Along with this it can also be used by military satellites to detect unusual objects on the ground from the space using this detection system.

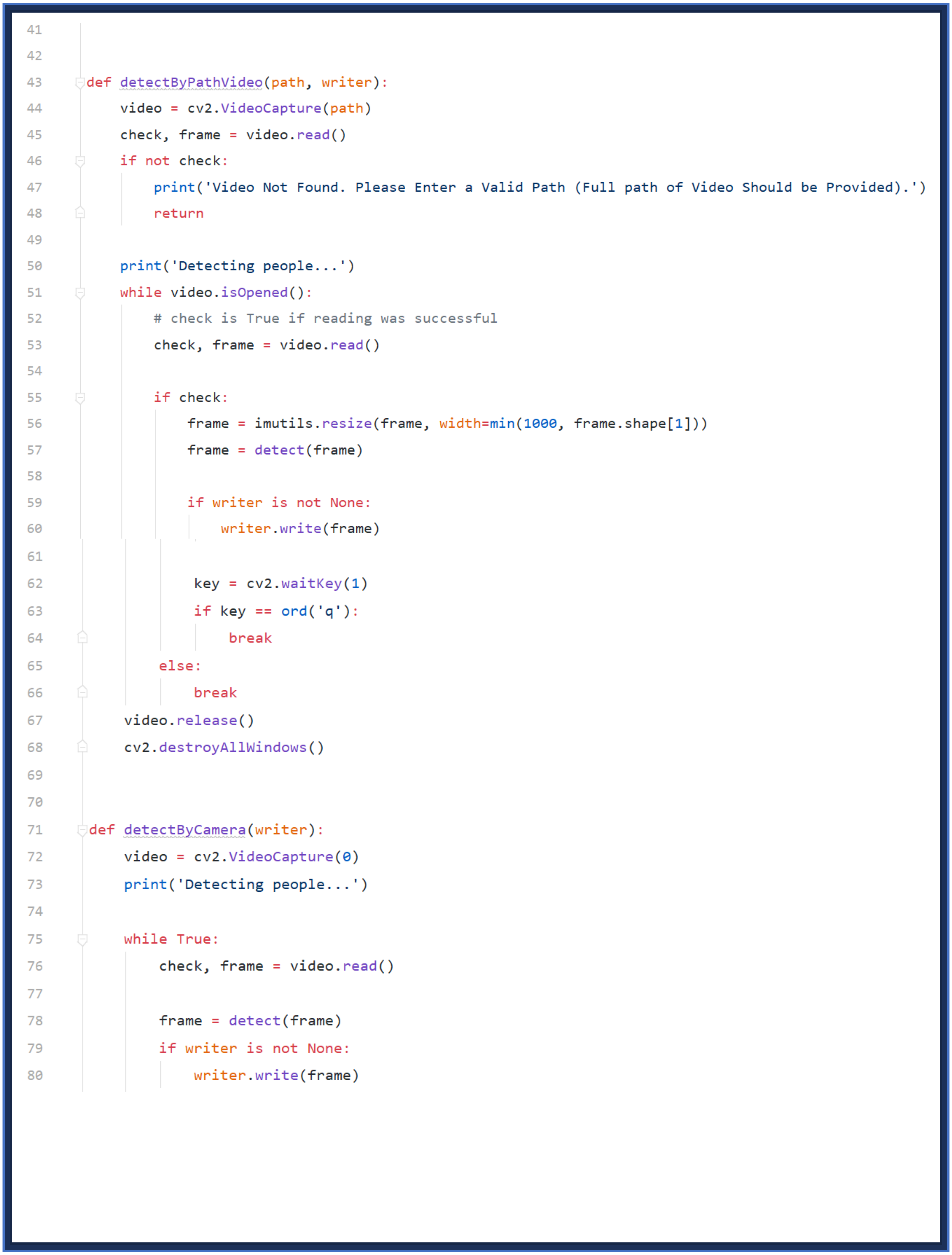
CODE

Link:

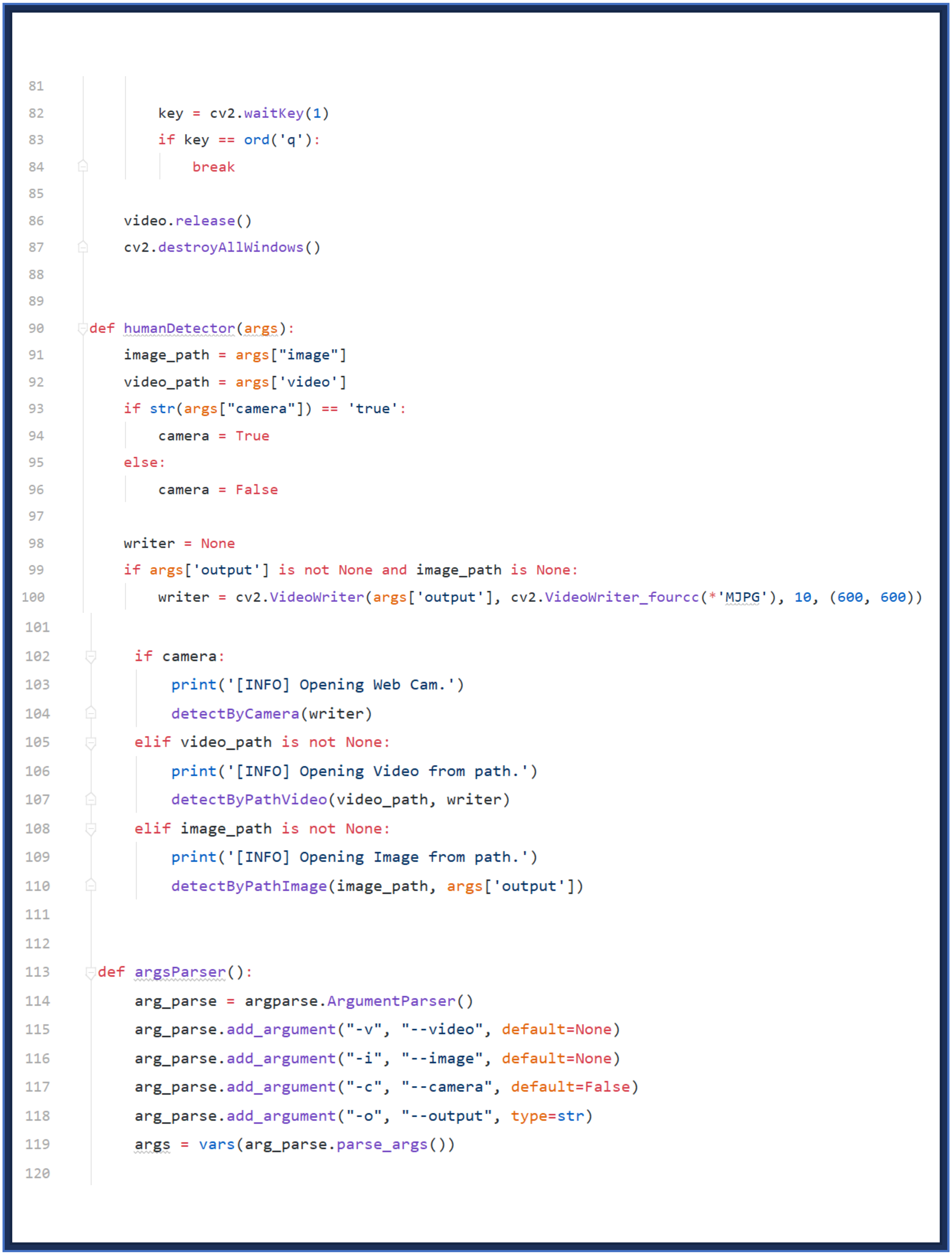
https://drive.google.com/file/d/1vaagUhYpLoVvnLTjWkd44ITTxzl0-hp6/view?usp=sharing



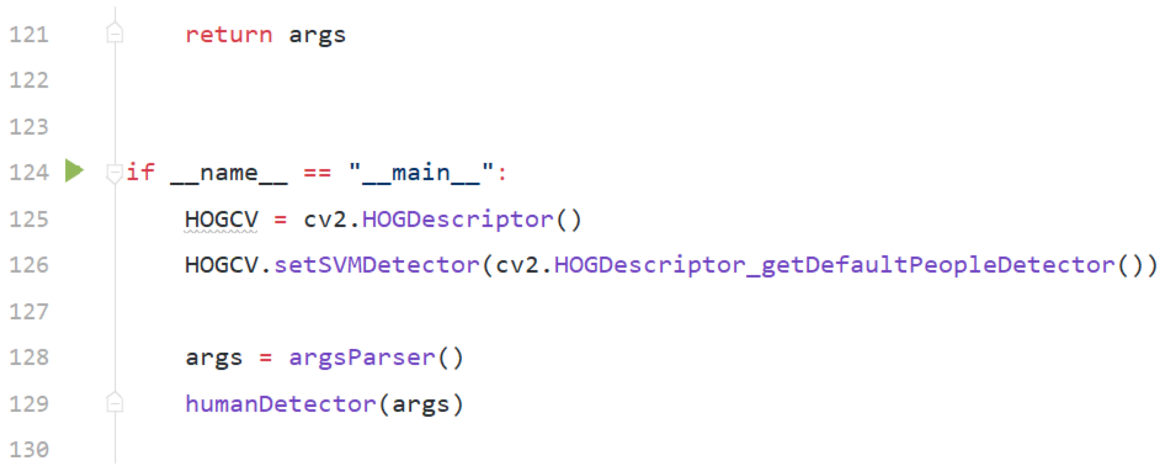
```python
import cv2
import imutils
import argparse


def detect(frame):
    # PREPROCESSING
    # bgr image to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # gaussian blur
    blur = cv2.GaussianBlur(gray, (3, 3), 0)
    # coordinates of rectangles found
    bounding_box_coordinate, weights = HOGCV.detectMultiScale(
        blur, winStride=(4, 4), padding=(8, 8), scale=1.03)

    person = 1

    for x, y, w, h in bounding_box_coordinate:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, f'person {person}', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
        person += 1

    cv2.putText(frame, 'Status : Detecting ', (40, 40), cv2.FONT_HERSHEY_DUPLEX, 0.8, (255, 0, 0), 2)
    cv2.putText(frame, f'Total Persons : {person - 1}', (40, 70), cv2.FONT_HERSHEY_DUPLEX, 0.8, (255, 0, 0), 2)
    cv2.imshow('output', frame)
    return frame


def detectByPathImage(path, output_path):
    image = cv2.imread(path)

    image = imutils.resize(image, width=min(800, image.shape[1]))

    result_image = detect(image)

    if output_path is not None:
        cv2.imwrite(output_path, result_image)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```



```python


def detectByPathVideo(path, writer):
    video = cv2.VideoCapture(path)
    check, frame = video.read()
    if not check:
        print('Video Not Found. Please Enter a Valid Path (Full path of Video Should be Provided).')
        return

    print('Detecting people...')
    while video.isOpened():
        # check is True if reading was successful
        check, frame = video.read()

        if check:
            frame = imutils.resize(frame, width=min(1000, frame.shape[1]))
            frame = detect(frame)

            if writer is not None:
                writer.write(frame)

            key = cv2.waitKey(1)
            if key == ord('q'):
                break
        else:
            break
    video.release()
    cv2.destroyAllWindows()


def detectByCamera(writer):
    video = cv2.VideoCapture(0)
    print('Detecting people...')

    while True:
        check, frame = video.read()

        frame = detect(frame)
        if writer is not None:
            writer.write(frame)
```



```python

        key = cv2.waitKey(1)
        if key == ord('q'):
            break

    video.release()
    cv2.destroyAllWindows()


def humanDetector(args):
    image_path = args["image"]
    video_path = args['video']
    if str(args["camera"]) == 'true':
        camera = True
    else:
        camera = False

    writer = None
    if args['output'] is not None and image_path is None:
        writer = cv2.VideoWriter(args['output'], cv2.VideoWriter_fourcc(*'MJPG'), 10, (600, 600))

    if camera:
        print('[INFO] Opening Web Cam.')
        detectByCamera(writer)
    elif video_path is not None:
        print('[INFO] Opening Video from path.')
        detectByPathVideo(video_path, writer)
    elif image_path is not None:
        print('[INFO] Opening Image from path.')
        detectByPathImage(image_path, args['output'])


def argsParser():
    arg_parse = argparse.ArgumentParser()
    arg_parse.add_argument("-v", "--video", default=None)
    arg_parse.add_argument("-i", "--image", default=None)
    arg_parse.add_argument("-c", "--camera", default=False)
    arg_parse.add_argument("-o", "--output", type=str)
    args = vars(arg_parse.parse_args())

```



```
    return args


if __name__ == "__main__":
    HOGCV = cv2.HOGDescriptor()
    HOGCV.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

    args = argsParser()
    humanDetector(args)
```

Google drive link for the Code:

https://drive.google.com/file/d/1vaagUhYpLoVvnLTjWkd44ITTxzl0-hp6/view?usp=sharing

REFERENCES

- Original Citation "Histograms of Oriented Gradients for Human Detection"
- OpenCV HOG + SVM Official Documentation
- Gradient Vector and Normalization
- Human Detection Dataset for SVM