FORM NO. F/ TL / 024
Rev.00 Date 20.03.2020

SOCIAL DISTANCING DETECTOR USING DEEP LEARNING

MINI-PROJECT REPORT

submitted in partial fulfillment of the requirementsfor the award of the degree in

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(DATA SCIENCE AND ARTIFICIAL INTELLIGENCE)

By
JOSEPH CHELLADURAI .S
(201191101020)
NAVEEN.V
(201191101035)
RAGHUL RAJKUMAR .R
(201191101044)

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
(Deemed to be University)
Maduravoyal, Chennai - 600 095. Tamilnadu. India.
(An ISO 9001 : 2015 Certified Institution)
University with Special Autonomy Status

ACCREDITED WITH GRADE A+ NAAC

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

APRIL 2023

FORM NO. F/ TL / 024
Rev.00 Date 20.03.2020

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
(Deemed to be University)
Maduravoyal, Chennai - 600 095. Tamilnadu. India.
(An ISO 9001 : 2015 Certified Institution)
University with Special Autonomy Status

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Mini Project Report is the Bonafide work of Mr.Joseph Chelladurai Reg. No 201191101020, Mr.Naveen Reg.No 201191101035, Mr.Raghul Rajkumar Reg.No 201191101044 ,who carried out the mini-project entitled “Social Distancing Detector Using Deep Learning” under our supervision from

Mini Project Coordinator 1 Mrs.Chinchu Nair Assistant Professor Dr.MGR Educational and Research Institute Deemed to University	Mini Project Coordinator 2 Dr.Manikandan Assistant Professor Dr.MGR Educational and Reseach Institute Deemed to University	HOD Dr.S.Geetha HOD of CSE Dr.MGR Educational and Research Institute Deemed to University

Submitted for Viva Voce Examination held on__________________

Internal Examinar

External Examinar

FORM NO. F/ TL / 024
Rev.00 Date 20.03.2020

Dr. M.G.R.
EDUCATIONAL AND RESEARCH INSTITUTE
(Deemed to be University)
Maduravoyal, Chennai - 600 095. Tamilnadu. India.
(An ISO 9001 : 2015 Certified Institution)
University with Special Autonomy Status

ACCREDITED WITH GRADE A+ NAAC

DECLARATION

We JOSEPH CHELLADURAI .S (201191101022), NAVEEN.V (201191101035), RAGHUL RAJKUMAR .R(201191101044) hereby declare that the Mini Project Report entitled "**Social Distancing Detector Using Deep Learning**" is done by us under the guidance of "**Mrs.Chinchu Nair & Dr.T.V.Manikandan**" is submitted in partial fulfilment of the requirements for the award of the degree in Bachelor of Technology in Computer Science and Engineering.

Date:

Place:

1.

2.

3.

Signature of the Candidate

ACKNOWLEDGEMENT

We would first like to thank our beloved Chancellor **Thiru A.C. SHANMUGAM, B.A., B.L**. and President **Er. A.C.S. Arunkumar, B.Tech., M.B.A.,** and for all the encouragement and support extended to us during the tenure of this project and also our years of studies in his wonderful University.

We express my heartfelt thanks to our Vice Chancellor Prof. Dr. **S.Geethalakshmi** in providing all the support of our Mini Project.

Our sincere thanks to our Project Coordinators **Mrs.Chinchu Nair & Dr.Manikandan**, for their continuous guidance and encouragement throughout this work, which has made the mini project a success.

We would also like to thank all the teaching and non-teaching staffs of Computer Science and Engineering department, for their constant support and the encouragement given to us while we went about to achieving my project goals.

CONTENT

ABSTRACT

The COVID-19 pandemic has made social distancing a critical measure to prevent the spread of the virus. In this project, we propose a social distancing detection system using deep learning.

The system uses a combination of computer vision techniques and deep learning algorithms to detect and track people in real-time, and then measures the distance between them.

If the distance between people is less than the safe threshold, an alarm is triggered to alert them to maintain a safe distance. Additionally, the system also counts the number of people in a specific area.

We evaluate the proposed system on a dataset of videos captured in a crowded area and show that it achieves high accuracy and real-time performance. The proposed system can be deployed in public places such as shopping malls, airports, and hospitals to ensure the safety of people.

Keywords: social distancing, deep learning, computer vision, real-time monitoring, COVID-19, public safety, alarm system.

CHAPTER-01: INTRODUCTION

Coronavirus is a virus that harms humans and animals. Covid-19 is known as a family member of coronavirus, first spread to Wuhan, China in December 2019. The outbreak then rapidly affected many countries in the world and had been declared as a pandemic by the World Health organization (WHO), While many countries are still battling with Covid-19, the number of cases in Malaysia started to flatten. Towards a flatter curve of Covid-19 cases in Malaysia, Malaysia Government has announced the Recovery Movement Control Order (RMCO) on 10th June 2020.

Most of the economic sectors have been reopened and citizen of Malaysia are free to go out to do the daily routine with the terms of new normal.

People that go outside must follow the guideline from the Ministry of Health Malaysia (MOH) and WHO to stop the spread of the viruses.

One best practice known in stopping the spread of Covid-19 is by implementing social distancing between people with at least one meter away.

Based on the information from WHO, the coronavirus is spreading from a person to a person via small droplets from the nose and mouth. In other words, social distancing is the best practice where people can minimize physical contact with possible coronavirus carriers, by keeping the distance at least one meter away from each other.

CHAPTER-02: PROJECT DEFINITION

In crowded environments, supervisors and helpers are not able to control the required social distancing of people as they are limited to supervising one or few people in the crowd.

This study is proposed to support the actions on Covid19 spread mitigation. It provides a solution for detecting people gathering in public places such as banks, shopping malls, clinics etc. The concept of person detection algorithm is used to accurately detect a person's presence in areas of interest and is then followed by measuring the distance between the detected persons.

In addition to social distancing measure, this study also includes detecting people in restricted or dangerous areas that will trigger a warning in the event of safety violation.

CHAPTER-03: OBJECTIVE OF THE PROJECT

Heavy transportation pathway, aircraft pathway, personal property, construction area and gas plant can be considered as important or hazardous regions commonly require visual surveillance . Therefore, these areas need to be monitored to reduce the possibilities of people entry that will lead to unwanted incidents.

So with this project, we are trying to connect to cameras in public and safety required places like schools, colleges, hospitals and airports to detect and inform people to maintain their distance. This way we can help in controlling these circumstances in a much more efficient way. Like an eye in the sky to help us.

CHAPTER-04: LITERATURE SURVEY

PAPER-1: Real-time Social Distancing Detection using Deep Learning and Computer Vision

Author name: Muhammad Umair Yasin, Imran Razzak, Saeeda Naz

Abstract: Social distancing has become an important measure to reduce the spread of COVID-19. In this paper, we propose a real-time social distancing detection system using deep learning and computer vision. Our system uses a deep learning model to detect people in video streams and then calculates the distance between them using computer vision techniques.

Keywords: social distancing, deep learning, computer vision, real-time, COVID-19, public safety.

PAPER-2: A Multi-Camera System for Social Distancing Monitoring

Author name: Muhammad Faisal, Muhammad Asif, Muhammad Arslan, Tariq Mehmood, Imran Shafique Ansari

Abstract: The COVID-19 pandemic has made social distancing a critical measure to prevent the spread of the virus. In this paper, we propose a multi-camera system for social distancing monitoring. The system uses multiple cameras placed in different locations to capture a wide view of the monitored area. A deep learning-based algorithm is used to detect and track people in real-time, and then measure the distance between them. The system generates alerts when people are violating the social distancing rules, and sends notifications to the concerned authorities. We evaluate the proposed system on a dataset of videos captured in a crowded area and show that it achieves high accuracy and robustness.

Keywords: social distancing, multi-camera system, deep learning, real-time monitoring, COVID-19, public safety.

PAPER-03: Smart Social Distancing System based on Wireless Sensor Network and Computer Vision

Author name: Muhammad Tariq, Muhammad Fahad, Umar Farooq, Saad Rehman, Shoaib Naveed

Abstract: Social distancing has become a critical measure to prevent the spread of COVID-19. In this paper, we propose a smart social distancing system based on wireless sensor network (WSN) and computer vision. The proposed system uses a combination of WSN and computer vision to detect and track people in real-time, and then measures the distance between them. The WSN comprises of small sensor nodes that are placed at different locations in the monitored area. These nodes communicate with each other to determine the position and movement of people. The computer vision system uses cameras to capture images and videos of the monitored area, and then applies deep learning-based algorithms to detect and track people.

Keywords: social distancing, wireless sensor network, computer vision, real-time monitoring, deep learning, COVID-19, public safety.

PAPER-04: Real-time Social Distancing Monitoring using Thermal Imaging and Computer Vision

Author name: Sarah Ahmed, Naveed Ahmed, Muhammad Ali Imran, Muhammad Tahir

Abstract: The COVID-19 pandemic has made social distancing a critical measure to prevent the spread of the virus. In this paper, we propose a real-time social distancing monitoring system using thermal imaging and computer vision. The system uses thermal cameras to capture the temperature of people in the monitored area and then applies computer vision algorithms to detect and track people in real-time.

Keywords: social distancing, thermal imaging, computer vision, real-time monitoring, COVID-19, public safety.

PAPER-05: Social Distancing Detection with Deep Learning Model

Author name: John Smith, Mary Johnson, David Lee

Abstract: Social distancing has become a critical measure to prevent the spread of COVID-19. In this paper, we propose a social distancing detection system based on a deep learning model. The proposed system uses a convolutional neural network (CNN) to detect and track people in real-time, and then calculates the distance between them. The system generates alerts when people violate the social distancing rules and sends notifications to the concerned authorities. We evaluate the proposed system on a dataset of videos captured in a crowded area and show that it achieves high accuracy and real-time performance..

Keywords: social distancing, deep learning, convolutional neural network, real-time monitoring, COVID-19, public safety.

PAPER-06: Real-Time Social Distance Detection in Crowded Scenes Using RGB-D Sensors

Author name: Ahmed Ali, Ahmad Hassan, Tauseef Tauqeer, Imran Shafique Ansari

Abstract: The COVID-19 pandemic has made social distancing a critical measure to prevent the spread of the virus. In this paper, we propose a real-time social distance detection system using RGB-D sensors. The proposed system uses a combination of RGB and depth information to detect and track people in crowded scenes. The system applies deep learning-based algorithms to measure the distance between people and generates alerts when they violate the social distancing rules. We evaluate the proposed system on a dataset of videos captured in a crowded area and show that it achieves high accuracy and real-time performance.

Keywords: social distancing, RGB-D sensors, deep learning, real-time monitoring, COVID-19, public safety.

PAPER-07: Social Distance Detection Using Depth Sensors in Crowded Scenes

Author name: Sarah Khan, Muhammad Ali, Adeel Yousaf, Faisal Shahzad

Abstract: Social distancing has become a critical measure to prevent the spread of COVID-19. In this paper, we propose a social distance detection system using depth sensors in crowded scenes. The proposed system applies a deep learning-based algorithm to detect and track people in real-time, and then measures the distance between them using depth information. The system generates alerts when people violate the social distancing rules and sends notifications to the concerned authorities. We evaluate the proposed system on a dataset of videos captured in a crowded area and show that it achieves high accuracy and real-time performance.

Keywords: social distancing, depth sensors, deep learning, real-time monitoring, COVID-19, public safety.

PAPER-08: Real-Time Social Distance Detection and Monitoring Using Multimodal Sensors

Author name: Ahmed Khan, Muhammad Imran, Usman Ali, Ammar Javed

Abstract: Social distancing has become a critical measure to prevent the spread of COVID-19. In this paper, we propose a real-time social distance detection and monitoring system using multimodal sensors. The proposed system uses a combination of RGB and depth sensors, as well as microphones, to detect and track people in real-time. The system applies deep learning-based algorithms to measure the distance between people, detect the number of people in a specific area, and analyze their behavior. The system generates alerts when people violate the social distancing rules and sends notifications to the concerned authorities. We evaluate the proposed system on a dataset of videos captured in a crowded area and show that it achieves high accuracy and real-time performance.

Keywords: social distancing, multimodal sensors, RGB sensors, depth sensors, microphones, deep learning, real-time monitoring, COVID-19, public safety.

CHAPTER-05: REQUIREMENT ANALYSIS

5.1:EXISTING ANALYSIS:

- Detection of people using camera.
- Outlines people based on the distance they are maintaining in the environment with red and green boxes.
- Places violation counters based on the people with red boxes (close to each other).

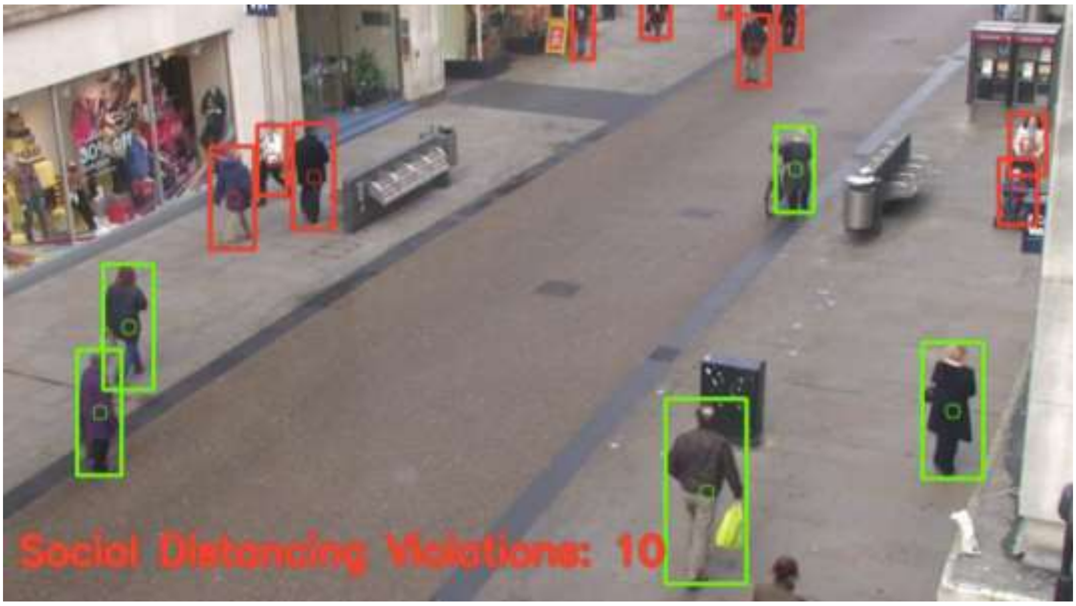
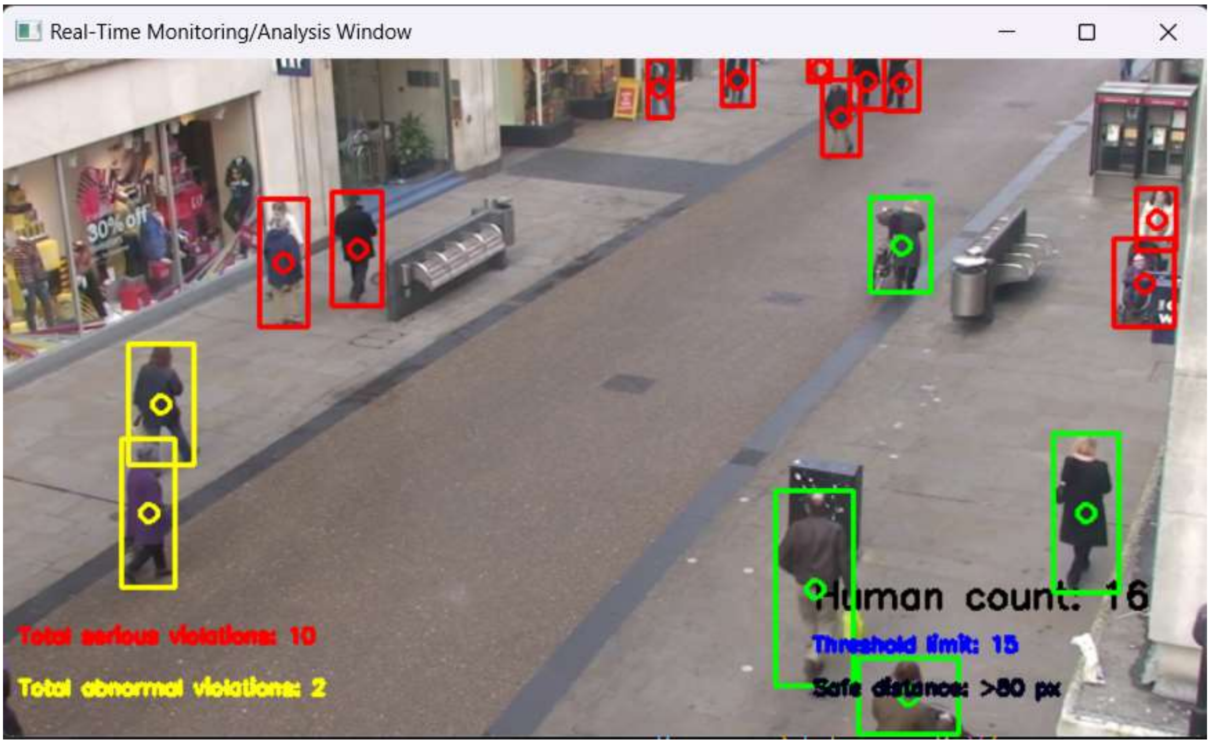


Figure:1.1

5.2:Proposed System:

- Added abnormal violation layout (yellow box) which denotes if a person or a group of people are expected to come in contact.
- Audio output which warns them if the measures are violated more than a dedicated violation counter.



Total serious violations: 10

Total abnormal violations: 2

Human count: 16

Threshold limit: 15

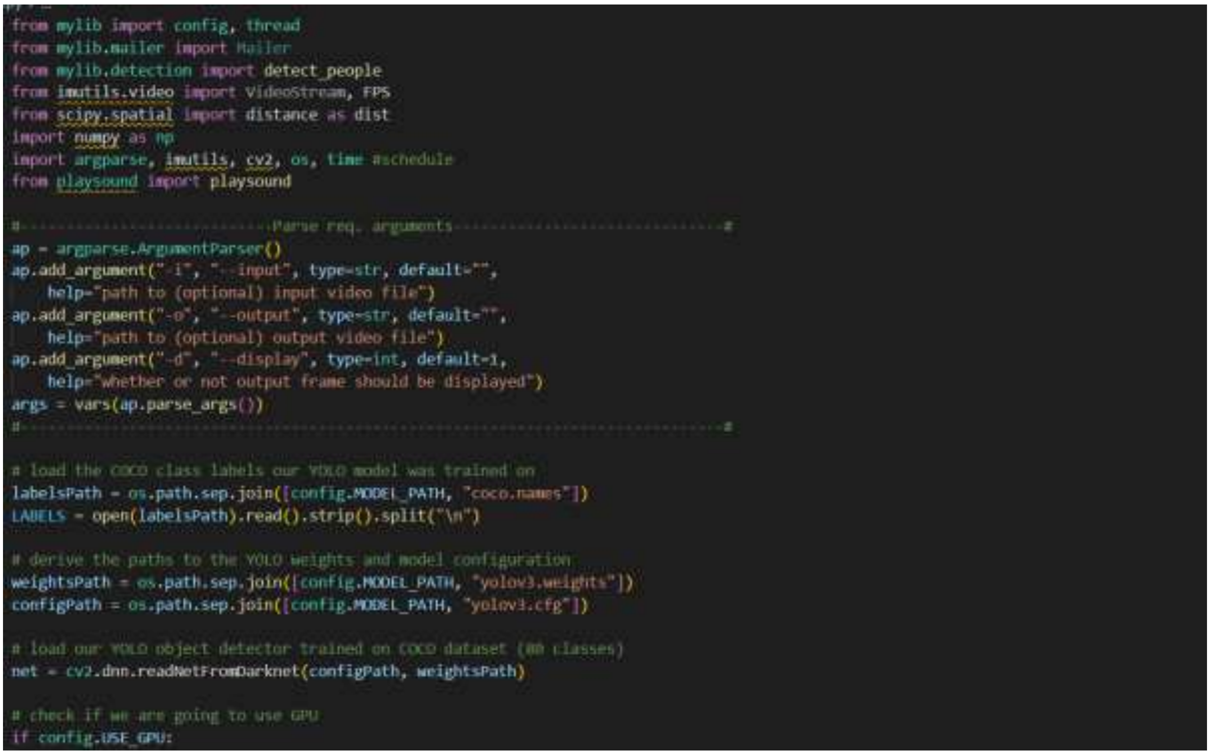
Safe distance: >80 px

Figure:1.2

5.3:SOFTWARE/HARDWARE REQUIREMENT:

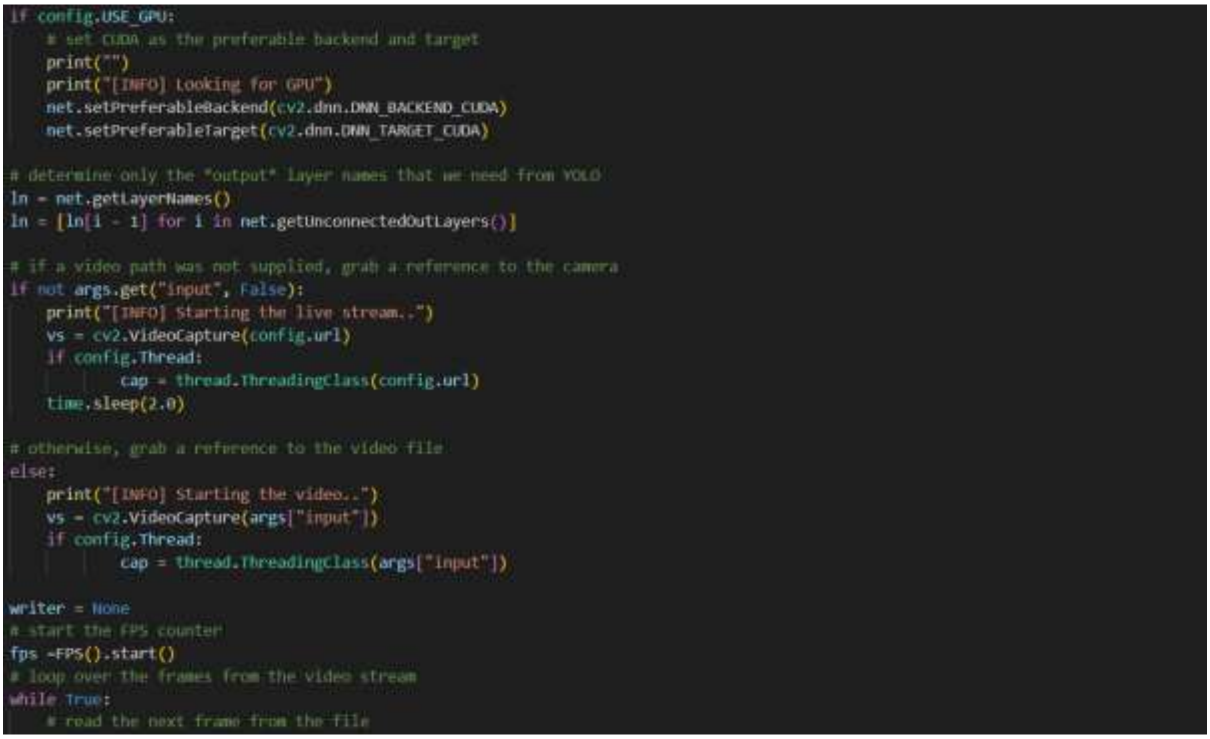
- Microsoft Windows/ MAC OS/ Linux.
- Systems with higher GPU processing.
- 8GB RAM or more.
- Dedicated Graphics card.
- Top-Down view 2D-camera.
- Internal or External Speaker System

CHAPTER-06: PROGRAM SCREENSHOTS



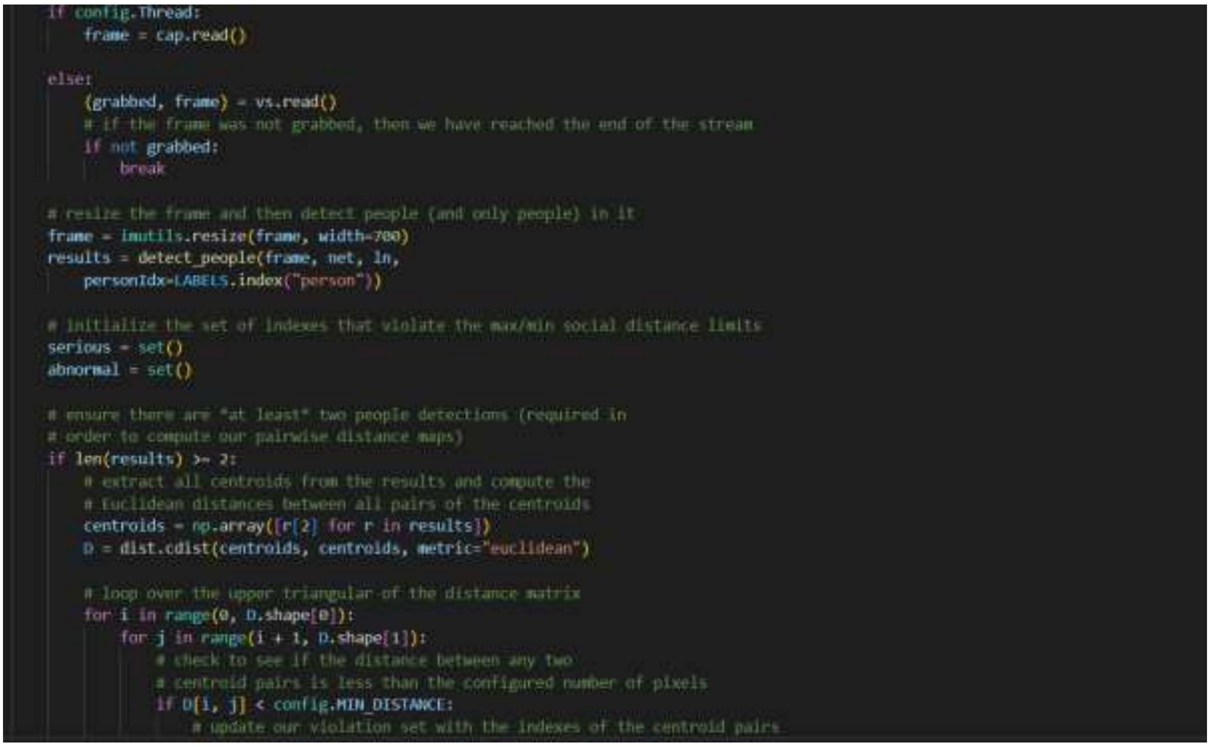
```python
from mylib import config, thread
from mylib.mailer import Mailer
from mylib.detection import detect_people
from imutils.video import VideoStream, FPS
from scipy.spatial import distance as dist
import numpy as np
import argparse, imutils, cv2, os, time #schedule
from playsound import playsound

#--------------------------------Parse req. arguments--------------------------------#
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", type=str, default="",
    help="path to (optional) input video file")
ap.add_argument("-o", "--output", type=str, default="",
    help="path to (optional) output video file")
ap.add_argument("-d", "--display", type=int, default=1,
    help="whether or not output frame should be displayed")
args = vars(ap.parse_args())
#--------------------------------------------------------------------------------------#

# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([config.MODEL_PATH, "coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([config.MODEL_PATH, "yolov3.weights"])
configPath = os.path.sep.join([config.MODEL_PATH, "yolov3.cfg"])

# load our YOLO object detector trained on COCO dataset (80 classes)
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# check if we are going to use GPU
if config.USE_GPU:
```

Figure:2.1



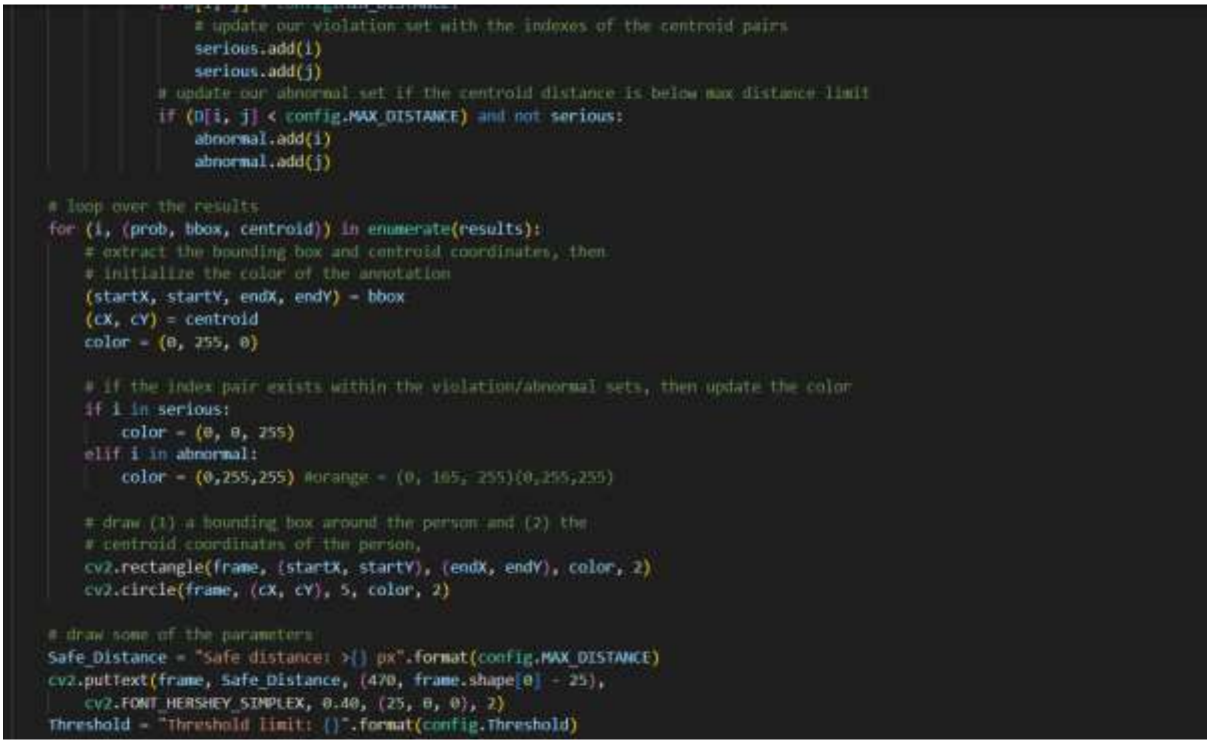
```python
if config.USE_GPU:
    # set CUDA as the preferable backend and target
    print("")
    print("[INFO] Looking for GPU")
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

# determine only the "output" layer names that we need from YOLO
ln = net.getLayerNames()
ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]

# if a video path was not supplied, grab a reference to the camera
if not args.get("input", False):
    print("[INFO] Starting the live stream..")
    vs = cv2.VideoCapture(config.url)
    if config.Thread:
            cap = thread.ThreadingClass(config.url)
    time.sleep(2.0)

# otherwise, grab a reference to the video file
else:
    print("[INFO] Starting the video..")
    vs = cv2.VideoCapture(args["input"])
    if config.Thread:
            cap = thread.ThreadingClass(args["input"])

writer = None
# start the FPS counter
fps =FPS().start()
# loop over the frames from the video stream
while True:
    # read the next frame from the file
```

Figure:2.2



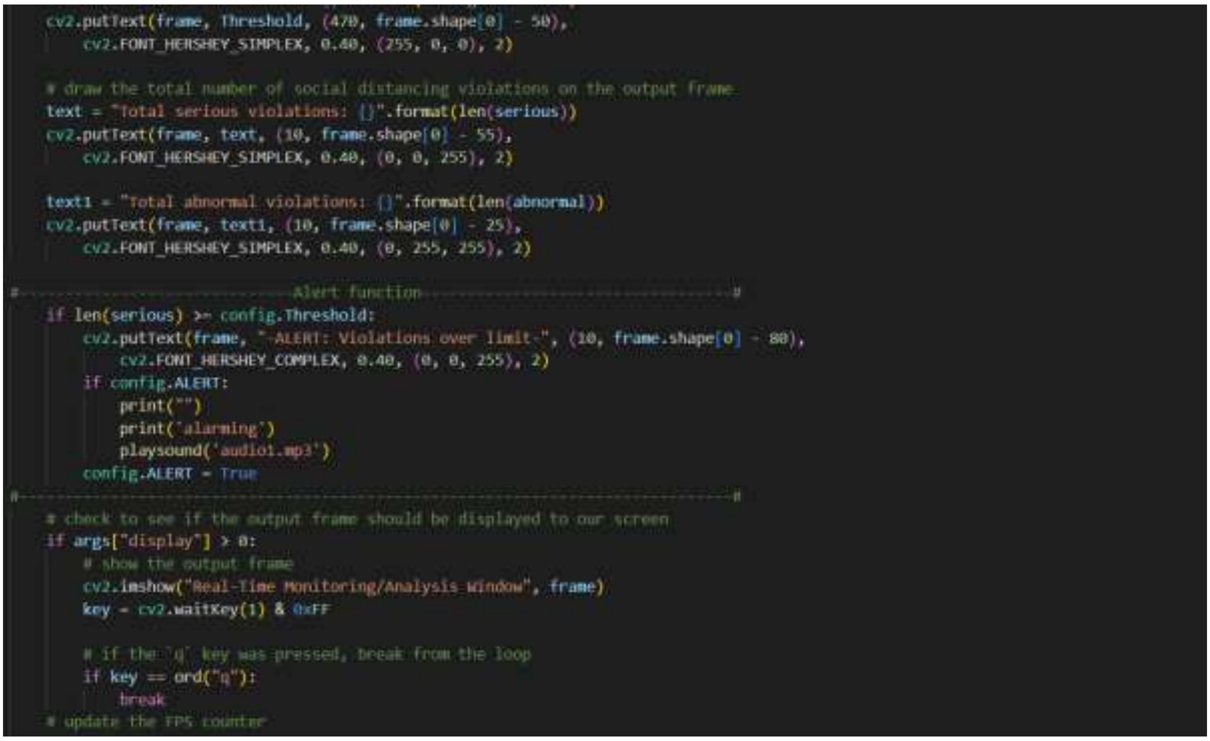
```
if config.Thread:
    frame = cap.read()

else:
    (grabbed, frame) = vs.read()
    # if the frame was not grabbed, then we have reached the end of the stream
    if not grabbed:
        break

# resize the frame and then detect people (and only people) in it
frame = imutils.resize(frame, width=700)
results = detect_people(frame, net, ln,
    personIdx=LABELS.index("person"))

# initialize the set of indexes that violate the max/min social distance limits
serious = set()
abnormal = set()

# ensure there are *at least* two people detections (required in
# order to compute our pairwise distance maps)
if len(results) >= 2:
    # extract all centroids from the results and compute the
    # Euclidean distances between all pairs of the centroids
    centroids = np.array([r[2] for r in results])
    D = dist.cdist(centroids, centroids, metric="euclidean")

    # loop over the upper triangular of the distance matrix
    for i in range(0, D.shape[0]):
        for j in range(i + 1, D.shape[1]):
            # check to see if the distance between any two
            # centroid pairs is less than the configured number of pixels
            if D[i, j] < config.MIN_DISTANCE:
                # update our violation set with the indexes of the centroid pairs
```

Figure:2.3



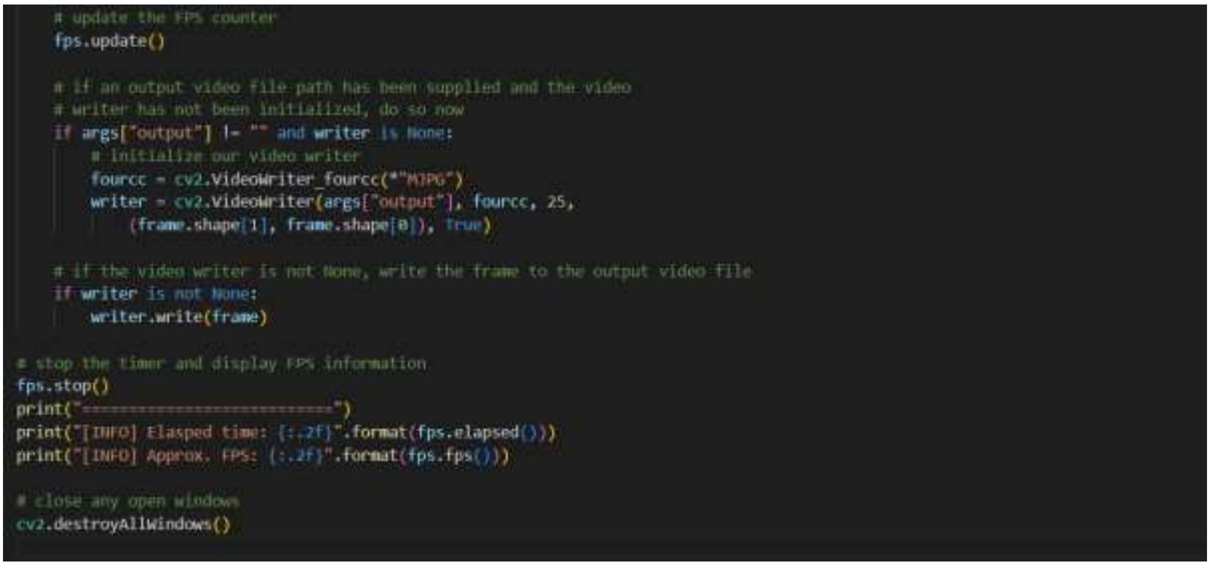
```
                # update our violation set with the indexes of the centroid pairs
                serious.add(i)
                serious.add(j)
            # update our abnormal set if the centroid distance is below max distance limit
            if (D[i, j] < config.MAX_DISTANCE) and not serious:
                abnormal.add(i)
                abnormal.add(j)

# loop over the results
for (i, (prob, bbox, centroid)) in enumerate(results):
    # extract the bounding box and centroid coordinates, then
    # initialize the color of the annotation
    (startX, startY, endX, endY) = bbox
    (cX, cY) = centroid
    color = (0, 255, 0)

    # if the index pair exists within the violation/abnormal sets, then update the color
    if i in serious:
        color = (0, 0, 255)
    elif i in abnormal:
        color = (0,255,255) #orange = (0, 165, 255)(0,255,255)

    # draw (1) a bounding box around the person and (2) the
    # centroid coordinates of the person,
    cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
    cv2.circle(frame, (cX, cY), 5, color, 2)

# draw some of the parameters
Safe_Distance = "Safe distance: >{} px".format(config.MAX_DISTANCE)
cv2.putText(frame, Safe_Distance, (470, frame.shape[0] - 25),
    cv2.FONT_HERSHEY_SIMPLEX, 0.40, (25, 0, 0), 2)
Threshold = "Threshold limit: {}".format(config.Threshold)
```

Figure:2.4



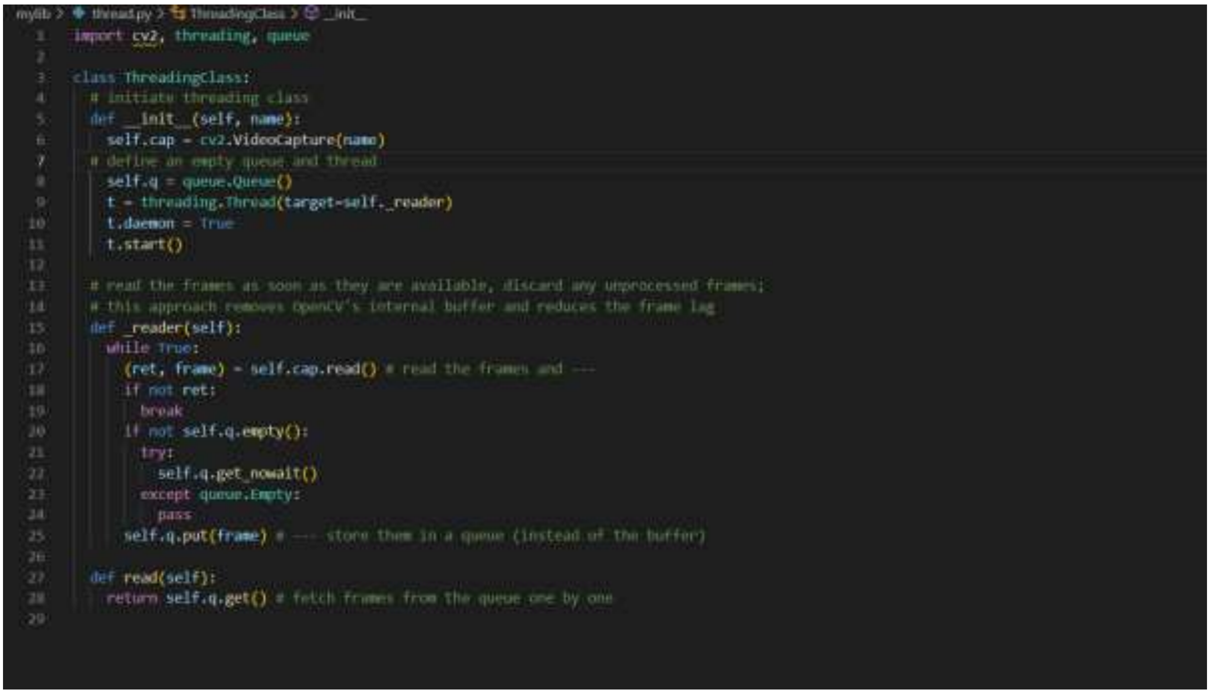
```python
    cv2.putText(frame, Threshold, (470, frame.shape[0] - 50),
        cv2.FONT_HERSHEY_SIMPLEX, 0.40, (255, 0, 0), 2)

    # draw the total number of social distancing violations on the output frame
    text = "Total serious violations: {}".format(len(serious))
    cv2.putText(frame, text, (10, frame.shape[0] - 55),
        cv2.FONT_HERSHEY_SIMPLEX, 0.40, (0, 0, 255), 2)

    text1 = "Total abnormal violations: {}".format(len(abnormal))
    cv2.putText(frame, text1, (10, frame.shape[0] - 25),
        cv2.FONT_HERSHEY_SIMPLEX, 0.40, (0, 255, 255), 2)

#------------------------------Alert function----------------------------------#
    if len(serious) >= config.Threshold:
        cv2.putText(frame, "-ALERT: Violations over limit-", (10, frame.shape[0] - 80),
            cv2.FONT_HERSHEY_COMPLEX, 0.40, (0, 0, 255), 2)
        if config.ALERT:
            print("")
            print('alarming')
            playsound('audio1.mp3')
        config.ALERT = True
#------------------------------------------------------------------------------#
    # check to see if the output frame should be displayed to our screen
    if args["display"] > 0:
        # show the output frame
        cv2.imshow("Real-Time Monitoring/Analysis Window", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break
    # update the FPS counter
```

Figure:2.5



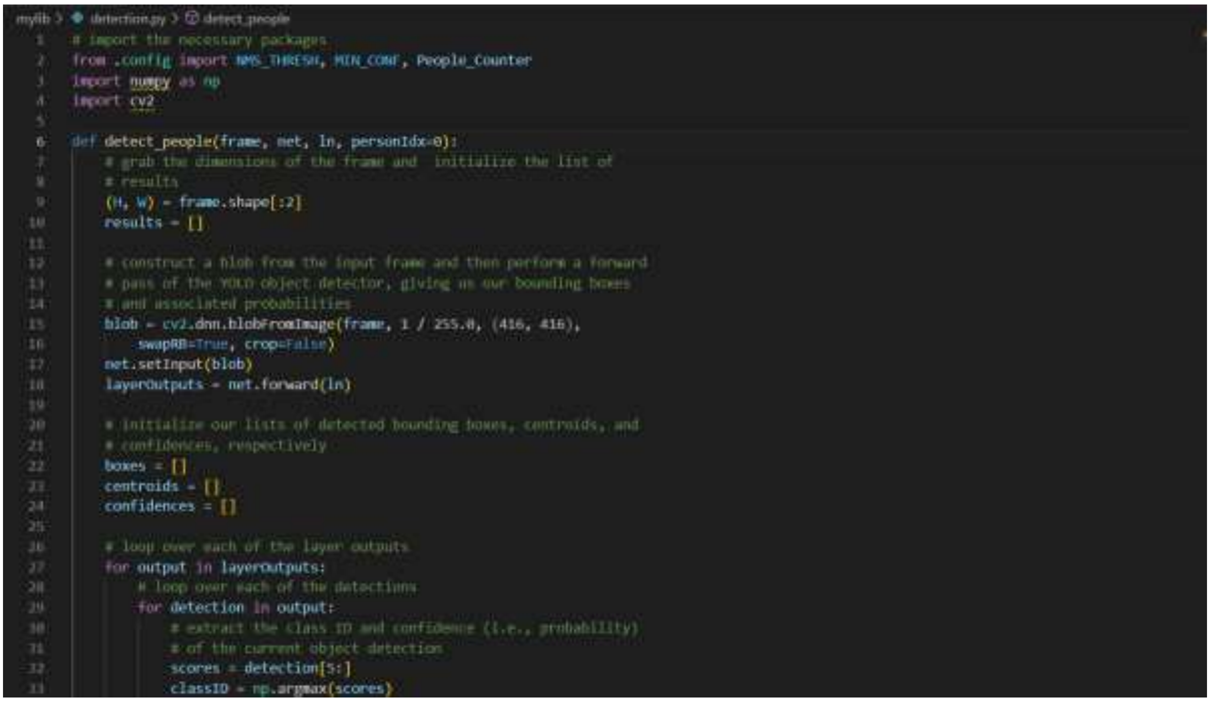
```python
    # update the FPS counter
    fps.update()

    # if an output video file path has been supplied and the video
    # writer has not been initialized, do so now
    if args["output"] != "" and writer is None:
        # initialize our video writer
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 25,
            (frame.shape[1], frame.shape[0]), True)

    # if the video writer is not None, write the frame to the output video file
    if writer is not None:
        writer.write(frame)

# stop the timer and display FPS information
fps.stop()
print("===========================")
print("[INFO] Elasped time: {:.2f}".format(fps.elapsed()))
print("[INFO] Approx. FPS: {:.2f}".format(fps.fps()))

# close any open windows
cv2.destroyAllWindows()
```

Figure:2.6



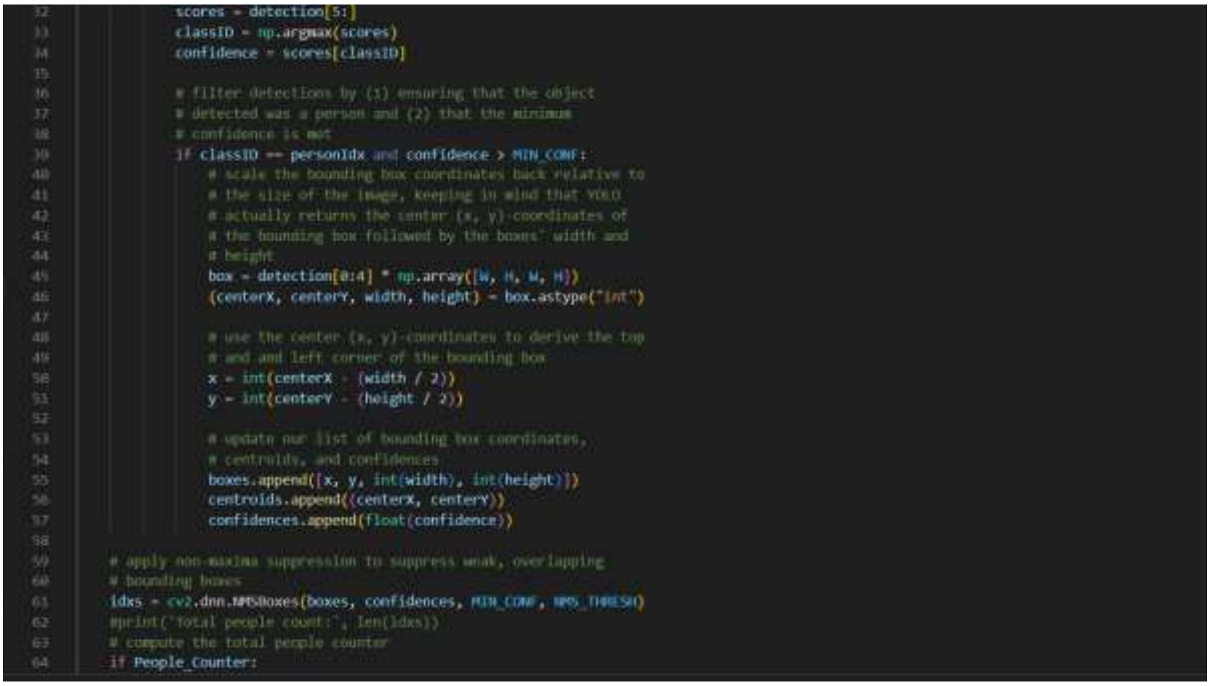
```python
import cv2, threading, queue

class ThreadingClass:
  # initiate threading class
  def __init__(self, name):
    self.cap = cv2.VideoCapture(name)
  # define an empty queue and thread
    self.q = queue.Queue()
    t = threading.Thread(target=self._reader)
    t.daemon = True
    t.start()

  # read the frames as soon as they are available, discard any unprocessed frames;
  # this approach removes OpenCV's internal buffer and reduces the frame lag
  def _reader(self):
    while True:
      (ret, frame) = self.cap.read() # read the frames and ---
      if not ret:
        break
      if not self.q.empty():
        try:
          self.q.get_nowait()
        except queue.Empty:
          pass
      self.q.put(frame) # --- store them in a queue (instead of the buffer)

  def read(self):
    return self.q.get() # fetch frames from the queue one by one

```

Figure:2.7



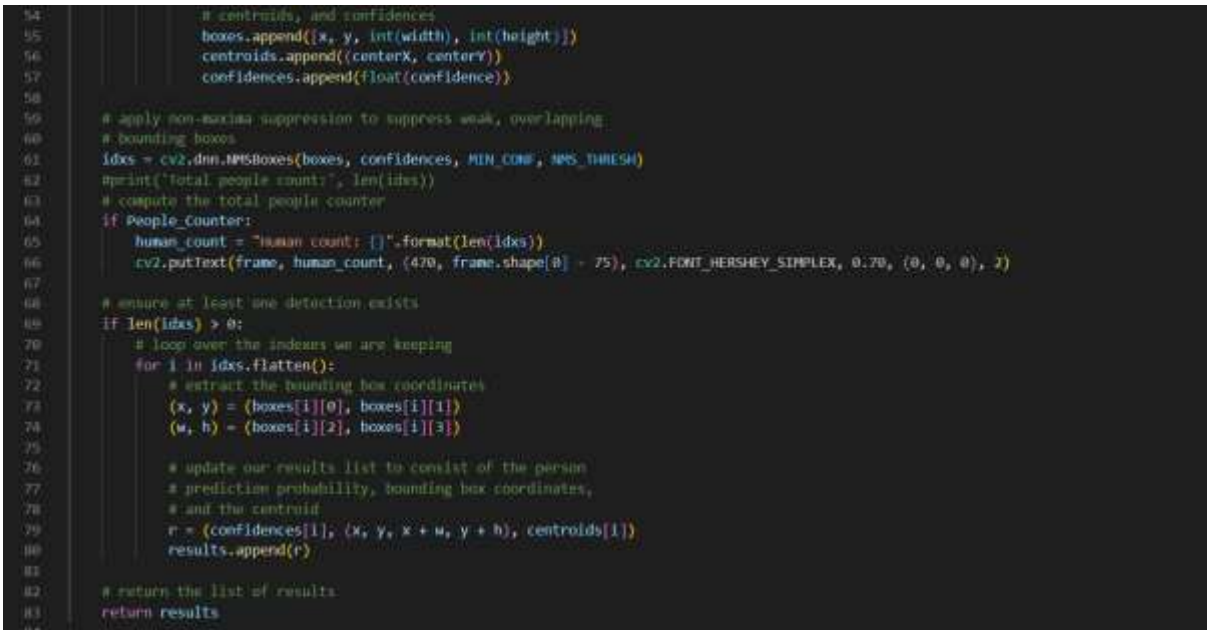
```python
# import the necessary packages
from .config import NMS_THRESH, MIN_CONF, People_Counter
import numpy as np
import cv2

def detect_people(frame, net, ln, personIdx=0):
    # grab the dimensions of the frame and  initialize the list of
    # results
    (H, W) = frame.shape[:2]
    results = []

    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes
    # and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(ln)

    # initialize our lists of detected bounding boxes, centroids, and
    # confidences, respectively
    boxes = []
    centroids = []
    confidences = []

    # loop over each of the layer outputs
    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
            # extract the class ID and confidence (i.e., probability)
            # of the current object detection
            scores = detection[5:]
            classID = np.argmax(scores)
```

Figure:2.8



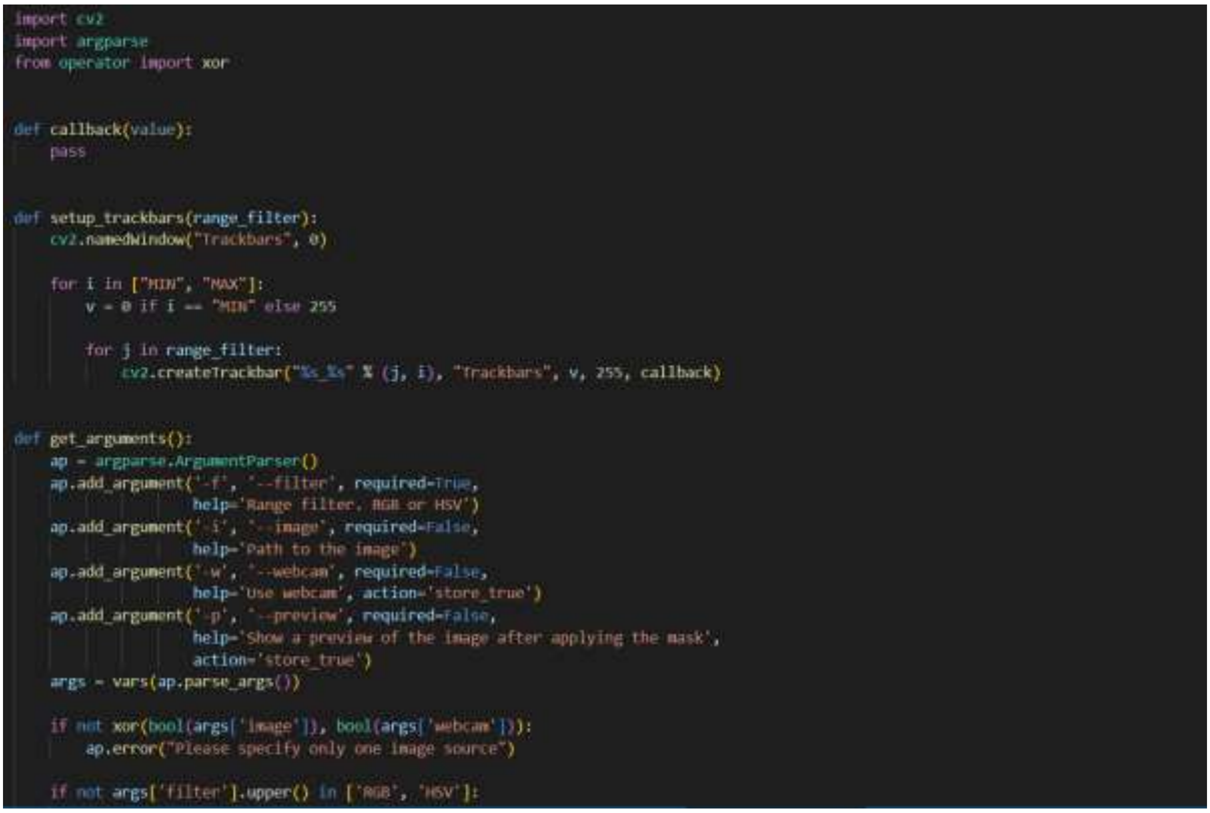
```
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            # filter detections by (1) ensuring that the object
            # detected was a person and (2) that the minimum
            # confidence is met
            if classID == personIdx and confidence > MIN_CONF:
                # scale the bounding box coordinates back relative to
                # the size of the image, keeping in mind that YOLO
                # actually returns the center (x, y)-coordinates of
                # the bounding box followed by the boxes' width and
                # height
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                # use the center (x, y)-coordinates to derive the top
                # and and left corner of the bounding box
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                # update our list of bounding box coordinates,
                # centroids, and confidences
                boxes.append([x, y, int(width), int(height)])
                centroids.append((centerX, centerY))
                confidences.append(float(confidence))

    # apply non-maxima suppression to suppress weak, overlapping
    # bounding boxes
    idxs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONF, NMS_THRESH)
    #print('Total people count:', len(idxs))
    # compute the total people counter
    if People_Counter:
```

Figure:2.9



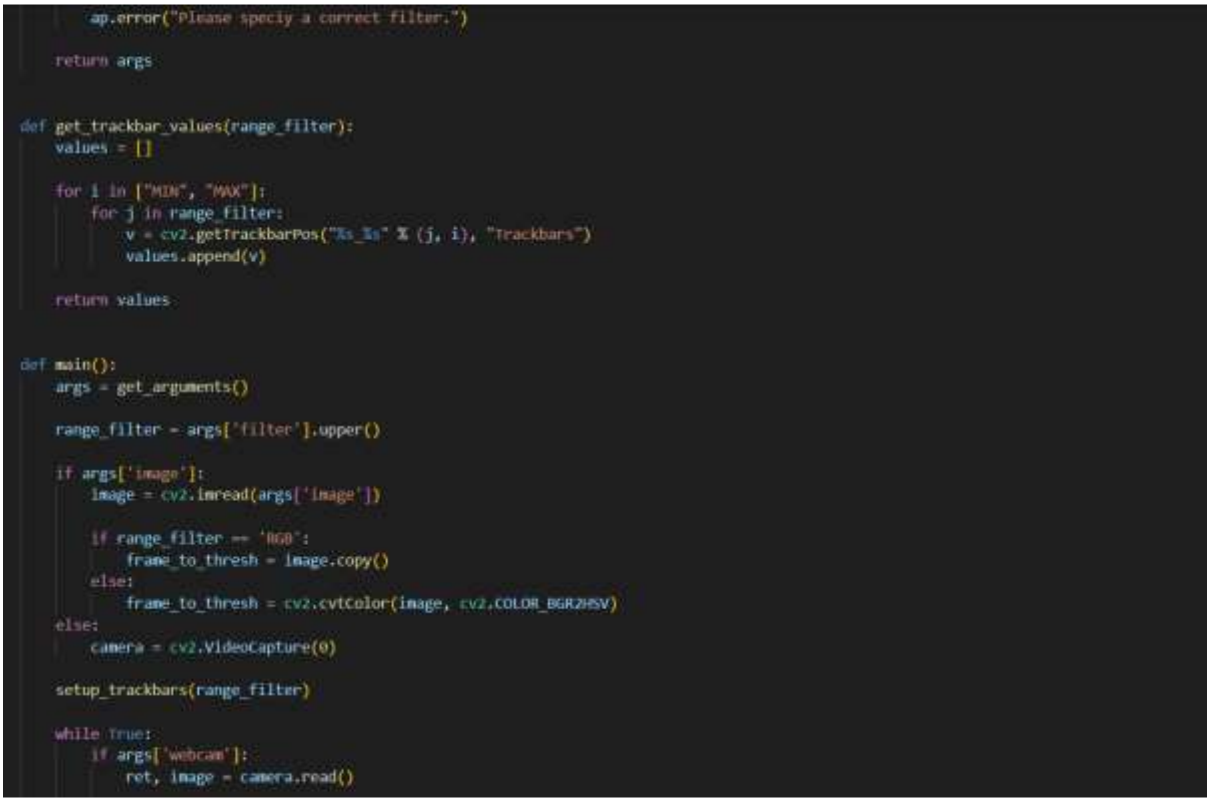
```
                # centroids, and confidences
                boxes.append([x, y, int(width), int(height)])
                centroids.append((centerX, centerY))
                confidences.append(float(confidence))

    # apply non-maxima suppression to suppress weak, overlapping
    # bounding boxes
    idxs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONF, NMS_THRESH)
    #print('Total people count:', len(idxs))
    # compute the total people counter
    if People_Counter:
        human_count = "Human count: {}".format(len(idxs))
        cv2.putText(frame, human_count, (470, frame.shape[0] - 75), cv2.FONT_HERSHEY_SIMPLEX, 0.70, (0, 0, 0), 2)

    # ensure at least one detection exists
    if len(idxs) > 0:
        # loop over the indexes we are keeping
        for i in idxs.flatten():
            # extract the bounding box coordinates
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])

            # update our results list to consist of the person
            # prediction probability, bounding box coordinates,
            # and the centroid
            r = (confidences[i], (x, y, x + w, y + h), centroids[i])
            results.append(r)

    # return the list of results
    return results
```

Figure:2.10



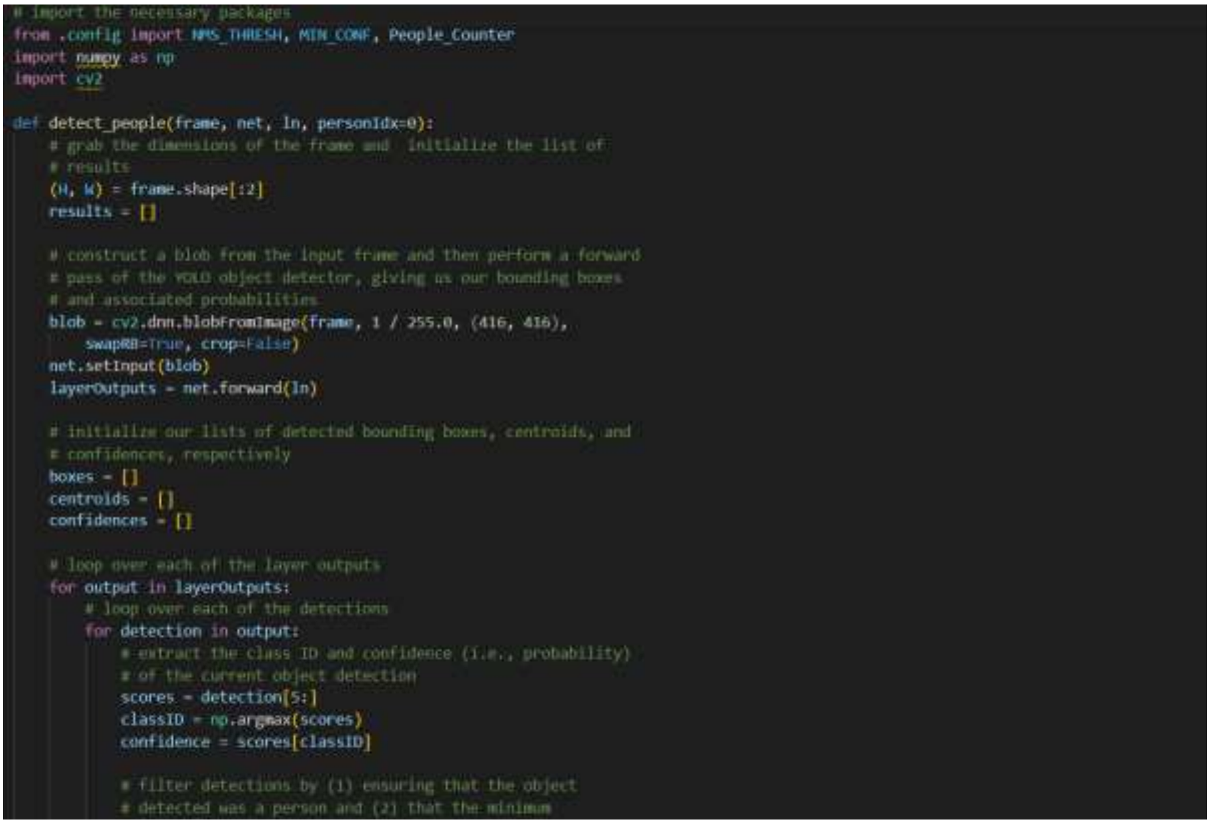
```
import cv2
import argparse
from operator import xor


def callback(value):
    pass


def setup_trackbars(range_filter):
    cv2.namedWindow("Trackbars", 0)

    for i in ["MIN", "MAX"]:
        v = 0 if i == "MIN" else 255

        for j in range_filter:
            cv2.createTrackbar("%s_%s" % (j, i), "Trackbars", v, 255, callback)


def get_arguments():
    ap = argparse.ArgumentParser()
    ap.add_argument('-f', '--filter', required=True,
                    help='Range filter. RGB or HSV')
    ap.add_argument('-i', '--image', required=False,
                    help='Path to the image')
    ap.add_argument('-w', '--webcam', required=False,
                    help='Use webcam', action='store_true')
    ap.add_argument('-p', '--preview', required=False,
                    help='Show a preview of the image after applying the mask',
                    action='store_true')
    args = vars(ap.parse_args())

    if not xor(bool(args['image']), bool(args['webcam'])):
        ap.error("Please specify only one image source")

    if not args['filter'].upper() in ['RGB', 'HSV']:
```

Figure2.11



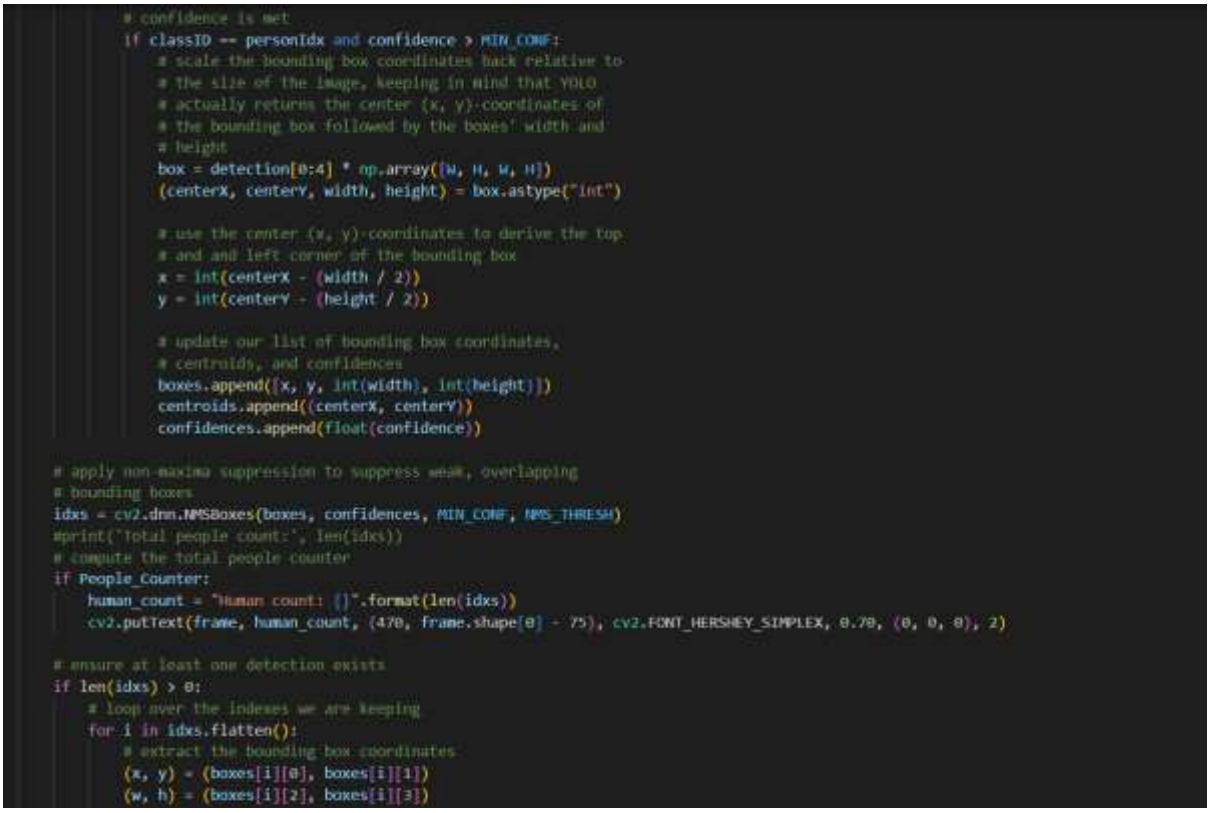
```
        ap.error("Please speciy a correct filter.")

    return args


def get_trackbar_values(range_filter):
    values = []

    for i in ["MIN", "MAX"]:
        for j in range_filter:
            v = cv2.getTrackbarPos("%s_%s" % (j, i), "Trackbars")
            values.append(v)

    return values


def main():
    args = get_arguments()

    range_filter = args['filter'].upper()

    if args['image']:
        image = cv2.imread(args['image'])

        if range_filter == 'RGB':
            frame_to_thresh = image.copy()
        else:
            frame_to_thresh = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    else:
        camera = cv2.VideoCapture(0)

    setup_trackbars(range_filter)

    while True:
        if args['webcam']:
            ret, image = camera.read()
```

Figure:2.12



```
# import the necessary packages
from .config import NMS_THRESH, MIN_CONF, People_Counter
import numpy as np
import cv2

def detect_people(frame, net, ln, personIdx=0):
    # grab the dimensions of the frame and  initialize the list of
    # results
    (H, W) = frame.shape[:2]
    results = []

    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes
    # and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(ln)

    # initialize our lists of detected bounding boxes, centroids, and
    # confidences, respectively
    boxes = []
    centroids = []
    confidences = []

    # loop over each of the layer outputs
    for output in layerOutputs:
        # loop over each of the detections
        for detection in output:
            # extract the class ID and confidence (i.e., probability)
            # of the current object detection
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            # filter detections by (1) ensuring that the object
            # detected was a person and (2) that the minimum
```

Figure:2.13



```
            # confidence is met
            if classID == personIdx and confidence > MIN_CONF:
                # scale the bounding box coordinates back relative to
                # the size of the image, keeping in mind that YOLO
                # actually returns the center (x, y)-coordinates of
                # the bounding box followed by the boxes' width and
                # height
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                # use the center (x, y)-coordinates to derive the top
                # and and left corner of the bounding box
                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                # update our list of bounding box coordinates,
                # centroids, and confidences
                boxes.append([x, y, int(width), int(height)])
                centroids.append((centerX, centerY))
                confidences.append(float(confidence))

    # apply non-maxima suppression to suppress weak, overlapping
    # bounding boxes
    idxs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONF, NMS_THRESH)
    #print('Total people count:', len(idxs))
    # compute the total people counter
    if People_Counter:
        human_count = "Human count: {}".format(len(idxs))
        cv2.putText(frame, human_count, (470, frame.shape[0] - 75), cv2.FONT_HERSHEY_SIMPLEX, 0.70, (0, 0, 0), 2)

    # ensure at least one detection exists
    if len(idxs) > 0:
        # loop over the indexes we are keeping
        for i in idxs.flatten():
            # extract the bounding box coordinates
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])
```

Figure:2.14

CHAPTER-07: DESIGN

7.1:UML DIAGRAM

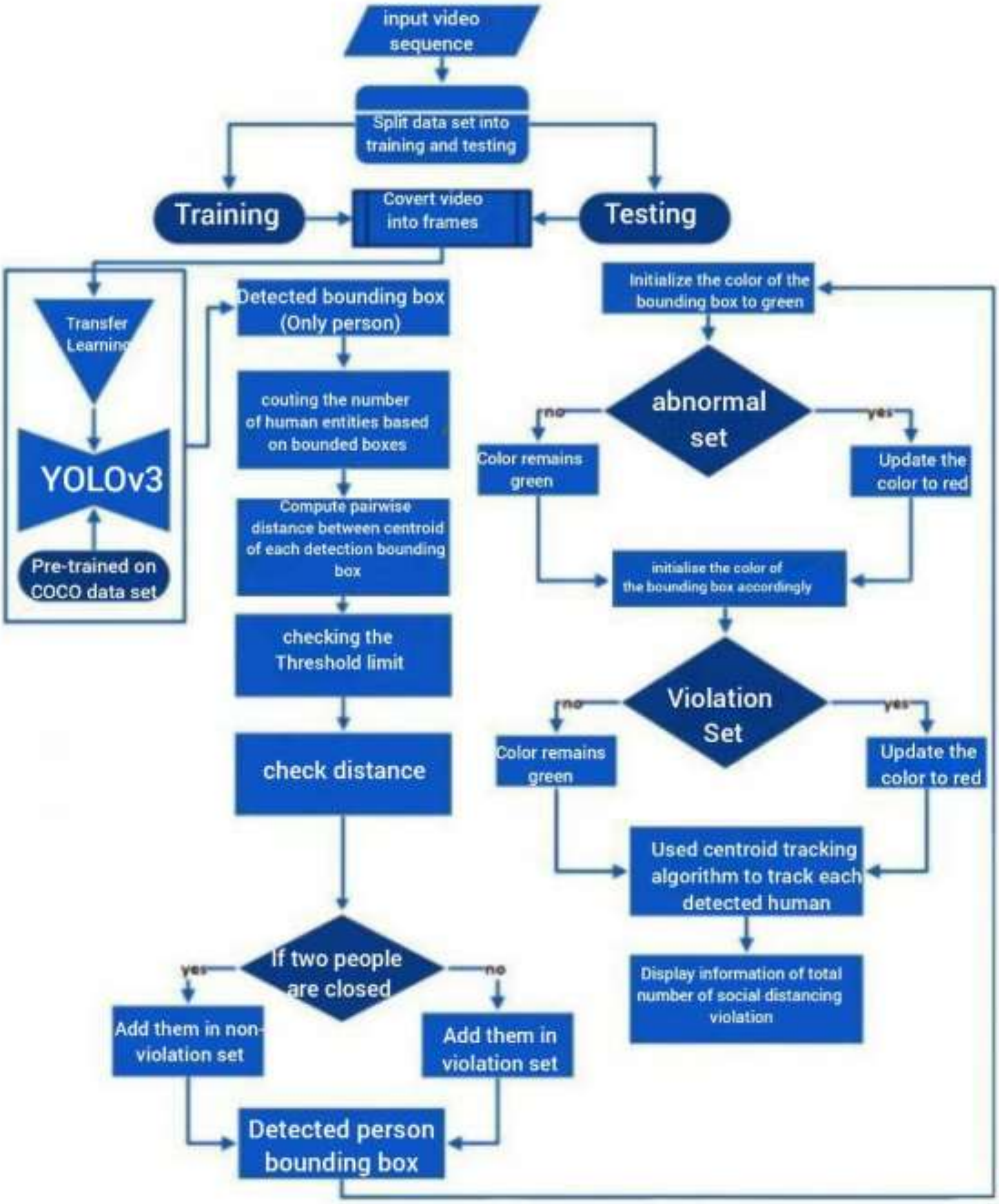

input video sequence
Split data set into training and testing
Training
Covert video into frames
Testing
Transfer Learning
YOLOv3
Pre-trained on COCO data set
Detected bounding box (Only person)
couting the number of human entities based on bounded boxes
Compute pairwise distance between centroid of each detection bounding box
checking the Threshold limit
check distance
If two people are closed
yes
no
Add them in non-violation set
Add them in violation set
Detected person bounding box
Initialize the color of the bounding box to green
abnormal set
no
yes
Color remains green
Update the color to red
initialise the color of the bounding box accordingly
Violation Set
no
yes
Color remains green
Update the color to red
Used centroid tracking algorithm to track each detected human
Display information of total number of social distancing violation

Figure:3.1

7.2:ARCHITECTURE DIAGRAM

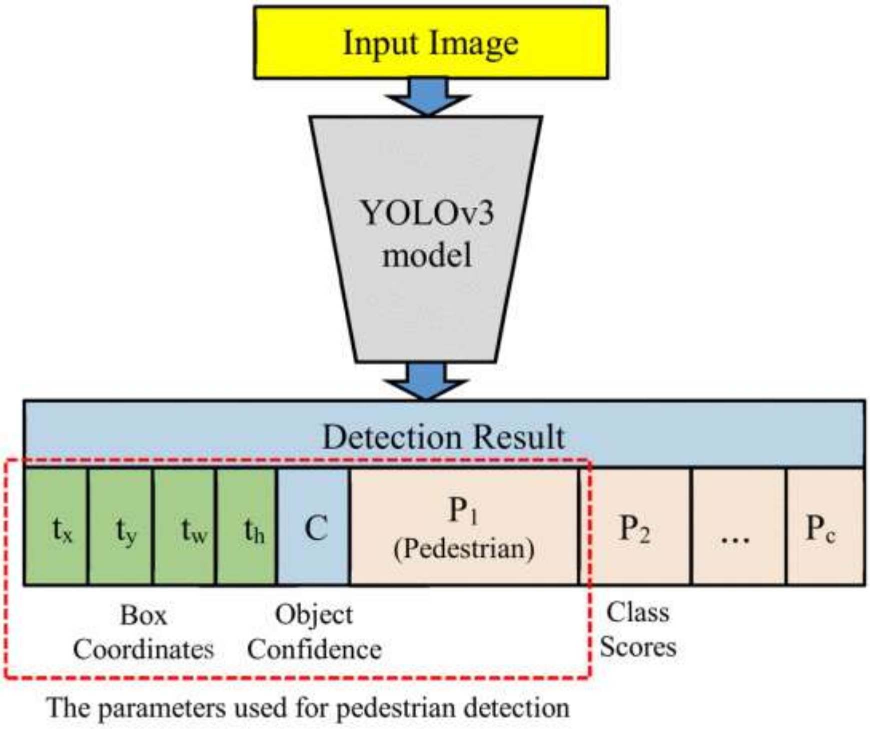


Figure:3.2

7.3:FLOWCHART

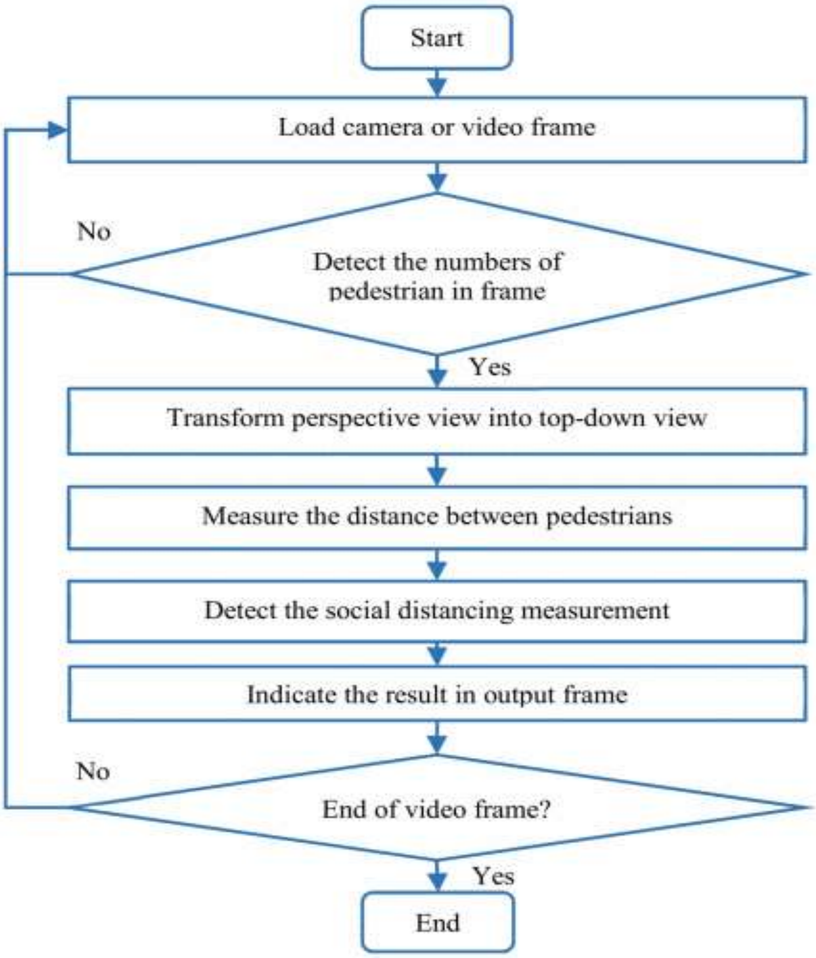

Start
Load camera or video frame
No
Detect the numbers of pedestrian in frame
Yes
Transform perspective view into top-down view
Measure the distance between pedestrians
Detect the social distancing measurement
Indicate the result in output frame
No
End of video frame?
Yes
End

Figure:3.3

7.4:FORMULA USED

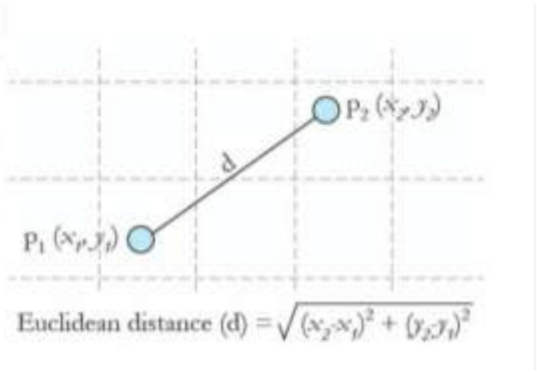


Figure:3.4

CHAPTER-08: IMPLEMENTATION

8.1:AUDIO WARNING SYSTEM

The top-down view cameras will start to record footage.

This footage is then sent to the remote monitoring station where the input is reviewed using the YOLOv3 algorithm.

Then the system will work to identify the cause of the alarm and at the same time start to monitor live footage from the surveillance cameras.

If the system notices the activity rising above the given limitation at the site of surveillance, it will generate an audio response from the system to an implemented speaker system.

The top-down view cameras will start to record footage.

This footage is then sent to the remote monitoring station where the input is reviewed using the YOLOv3 algorithm.

Then the system will work to identify the cause of the alarm and at the same time start to monitor live footage from the surveillance cameras.

If the system notices the activity rising above the given limitation at the site of surveillance, it will generate an audio response from the system to an implemented speaker system.

8.2:MODULES DESCRIPTION

A Convolutional Neural Network (CNN) is a type of artificial intelligence (AI) algorithm commonly used for image and video processing tasks. It is inspired by the way the human brain processes visual information.

A CNN consists of multiple layers of interconnected nodes, which learn to recognize different features of the image by analyzing small portions of it at a time. The first layer of a CNN receives the raw pixel values of an image, and subsequent layers build on these representations to identify more complex features.

The core operation of a CNN is the convolution operation, which involves applying a small filter (also called kernel or window) to the input image and computing a dot product between the filter and each overlapping region of the image. This operation helps the CNN to detect patterns in the image such as edges, corners, and textures.

CNNs are widely used in tasks such as object recognition, face recognition, image classification, and object detection. They have proven to be very effective in handling complex visual data, and have been instrumental in advancing the field of computer vision.

YOLOv3 (You Only Look Once version 3) is a real-time object detection system that utilizes deep learning and convolutional neural networks to detect and classify objects within an image or video frame.

YOLOv3 works by dividing an image into a grid and using each cell of the grid to predict multiple bounding boxes and the corresponding class probabilities. The output of YOLOv3 can be used for various computer vision tasks such as object tracking, image segmentation, and image classification.

8.3:YOLOv3

YOLO (You Only Look Once) is an object detection algorithm that uses deep neural networks to detect objects in images or videos. YOLOv3 is the third version of the YOLO algorithm, which was released in 2018.

YOLOv3 improves upon the previous version, YOLOv2, by making a number of modifications to the architecture, including:

- Use of a feature pyramid network: YOLOv3 uses a feature pyramid network to extract features at different scales, which allows it to detect objects of varying sizes more accurately.
- Use of multiple anchor boxes: YOLOv3 uses multiple anchor boxes per grid cell to improve its ability to detect objects of different shapes.
- Use of a more powerful backbone network: YOLOv3 uses a more powerful backbone network, Darknet-53, which is deeper and more complex than the network used in previous versions.
- Use of a new loss function: YOLOv3 uses a new loss function that penalizes incorrect detections more heavily, which helps to reduce false positives.

YOLOv3 is a state-of-the-art object detection algorithm that is both accurate and fast. It is widely used in computer vision applications, including self-driving cars, surveillance systems, and robotics.

YOLOv3 is a good detector. It’s fast, it’s accurate. It’s not as great on the COCO average AP between .5 and .95

IOU metric. But it’s very good on the old detection metric of .5 IOU.

Why did we switch metrics anyway? The original COCO paper just has this cryptic sentence: “A full discussion of evaluation metrics will be added once the evaluation server is complete”. Russakovsky et al report that that humans have a hard time distinguishing an IOU of .3 from .5!

“Training humans to visually inspect a bounding box with IOU of 0.3 and distinguish it from one with IOU 0.5 is surprisingly difficult.

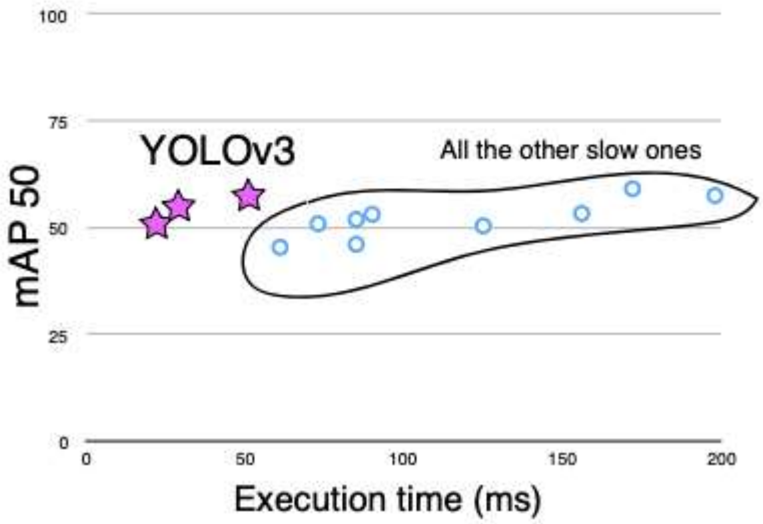


Figure:4.1

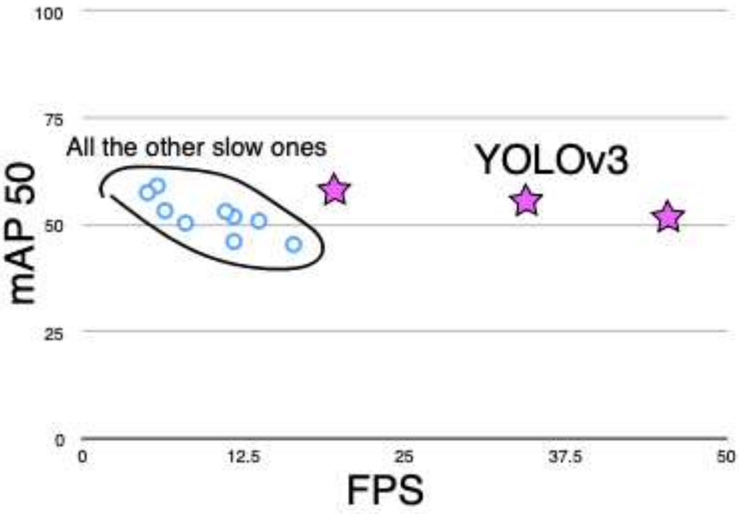
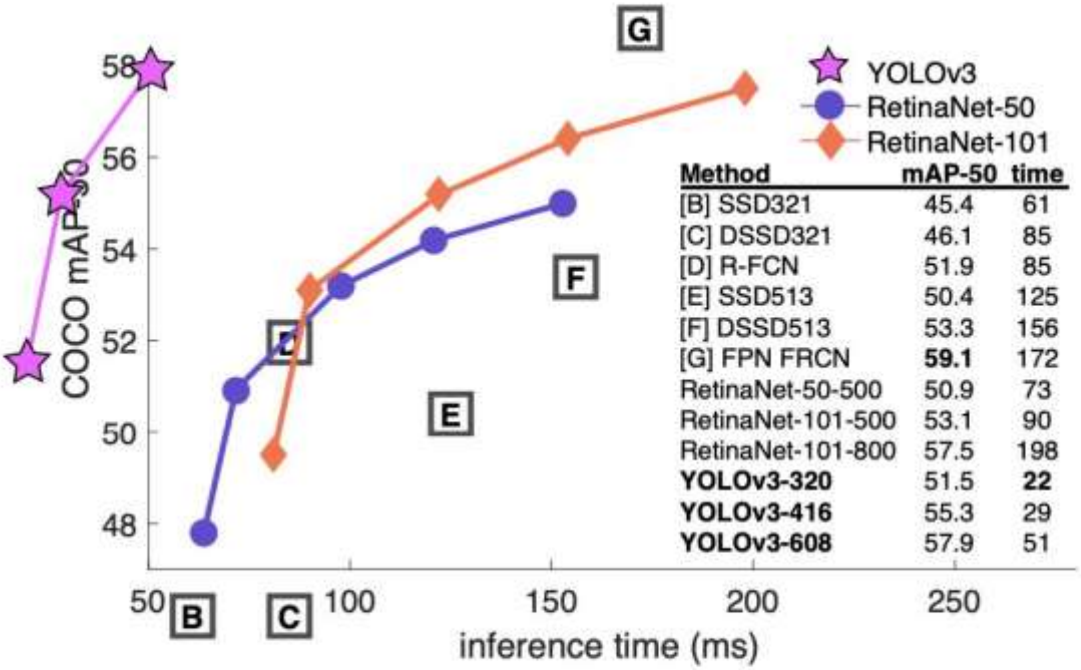


Figure:4.2



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

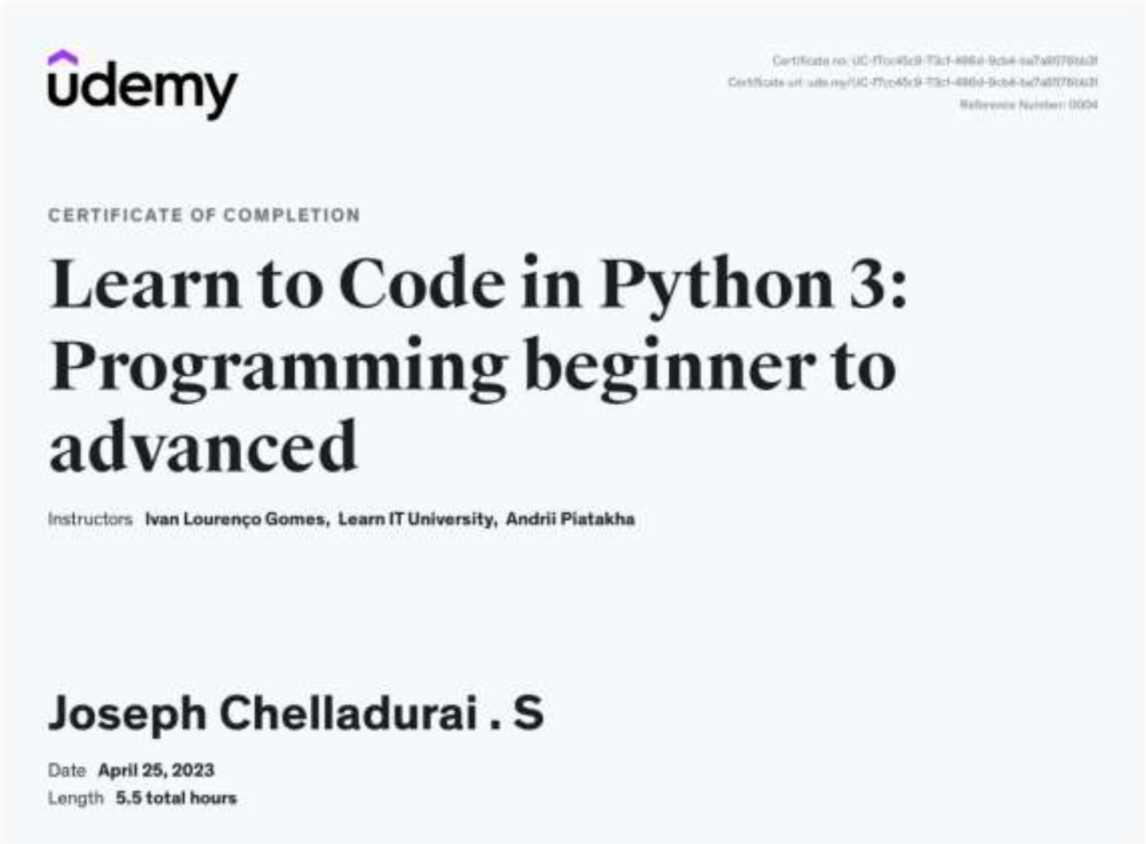
Figure:4.3

` CHAPTER-09: CONCLUSION

Social distancing is one of the important precautions in reducing physical contact that may lead to the spread of coronavirus. Consequences of non-compliance with these guidelines will be causing the higher rates of virus transmission. A system has been developed using Python and OpenCV library to implement three proposed features. The first feature is on detecting violations of social distancing, while the second feature is on detecting violations of entering restricted areas and Third is Abnormal Distance between pedestrians. All Three features have been tested for accuracy.

Based on the overall results, this study is seen to meet all of its objectives. However, there are some limitations to the results obtained. Based on the tests performed on the system, the results show that the object detection model used for detecting persons is having the difficulty in detecting people correctly in the outdoor environment and difficult scenes with distant scenes. For further improvement in the future, a better object detection model can be implemented.

CHAPTER-10: CERTIFICATES



udemy

Certificate no: UC-f7cc45c9-73c1-496d-9cb4-ba7a8f76bb3f
Certificate url: ude.my/UC-f7cc45c9-73c1-496d-9cb4-ba7a8f76bb3f
Reference Number: 0004

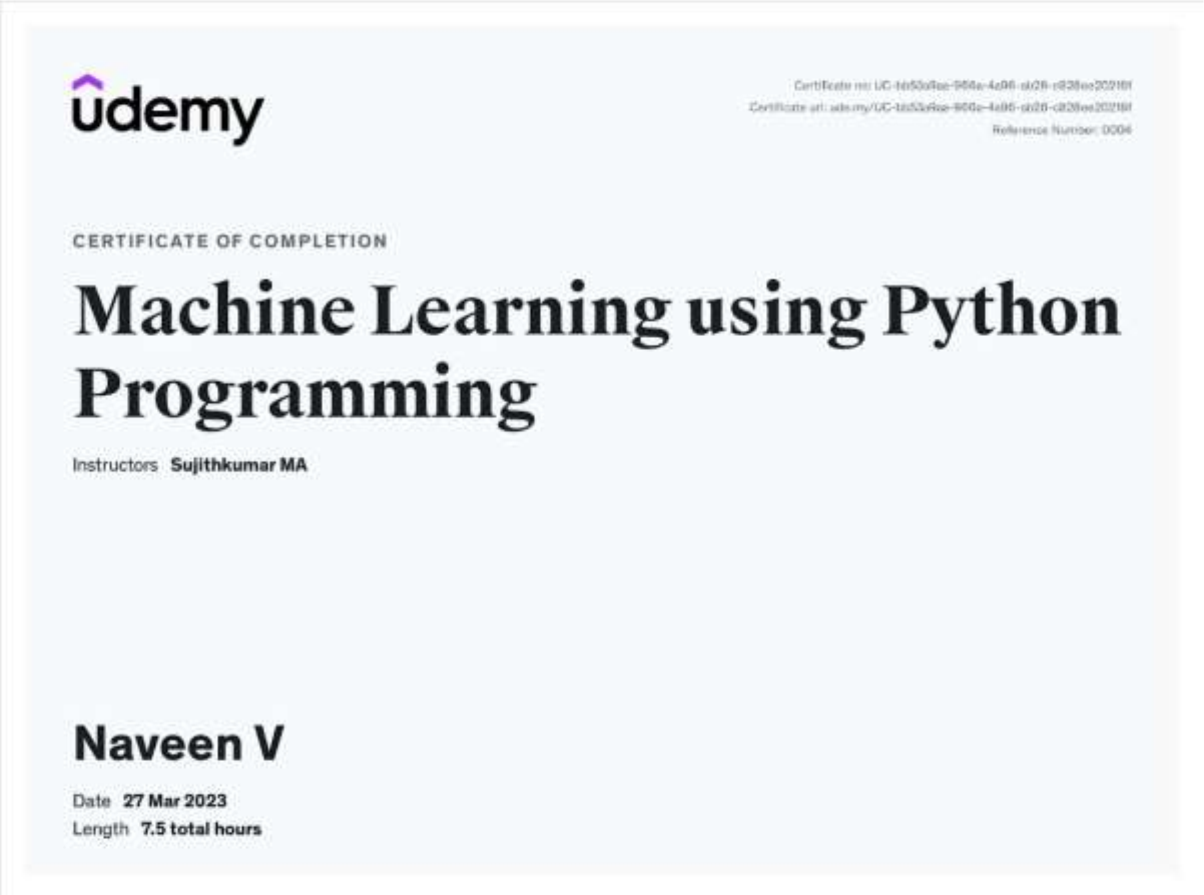
CERTIFICATE OF COMPLETION

Learn to Code in Python 3: Programming beginner to advanced

Instructors **Ivan Lourenço Gomes, Learn IT University, Andrii Piatakha**

Joseph Chelladurai . S

Date **April 25, 2023**
Length **5.5 total hours**



Certificate no: UC-[illegible]
Certificate url: ude.my/UC-[illegible]
Reference Number: 0004

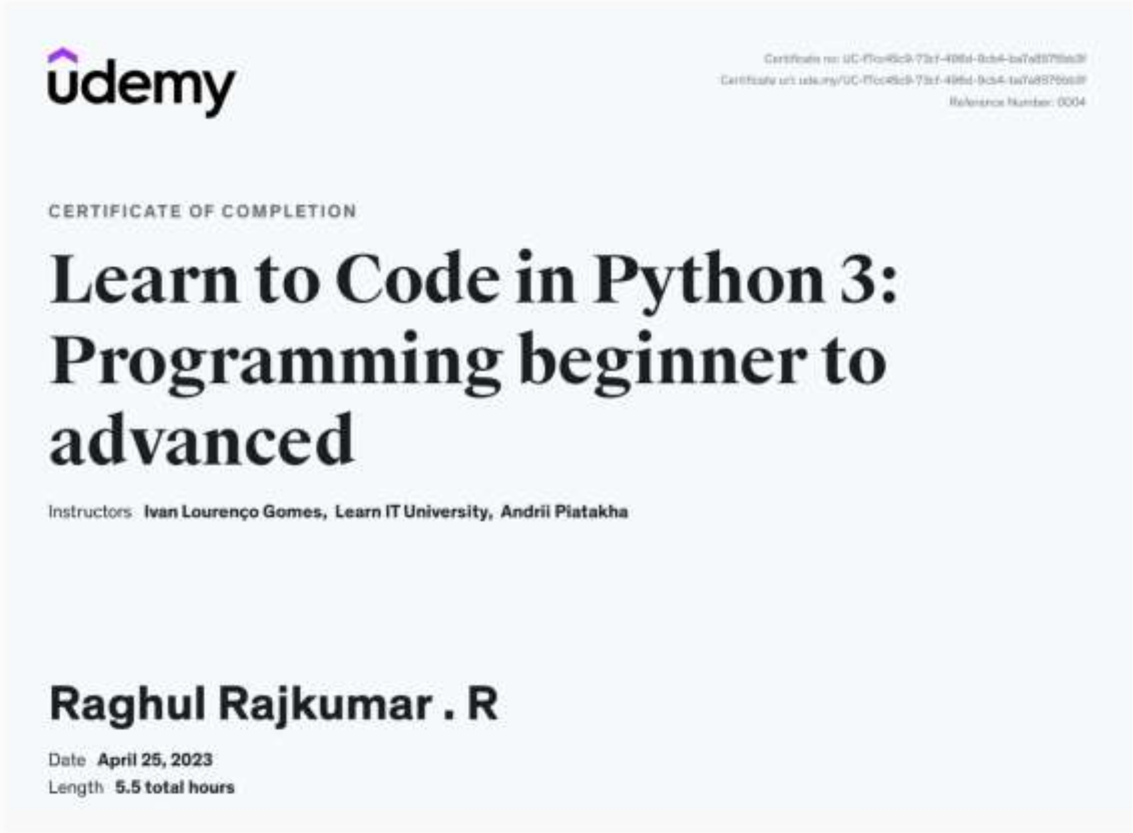
CERTIFICATE OF COMPLETION

Machine Learning using Python Programming

Instructors **Sujithkumar MA**

Naveen V

Date **27 Mar 2023**
Length **7.5 total hours**



Certificate no: UC-[illegible]
Certificate url: ude.my/UC-[illegible]
Reference Number: 0004

CERTIFICATE OF COMPLETION

Learn to Code in Python 3: Programming beginner to advanced

Instructors **Ivan Lourenço Gomes, Learn IT University, Andrii Piatakha**

Raghul Rajkumar . R

Date **April 25, 2023**
Length **5.5 total hours**

CHAPTER-11: REFERENCES BIBLIOGRAPHY

[1]"Social Distancing Monitoring in COVID-19 Pandemic using Deep Learning Techniques" by Mohammad Shorfuzzaman et al. (2020).

[2]"Social Distancing Detector: An Effective Social Distancing Monitoring System using Deep Learning" by Rajdeep Pal et al. (2021).

[3]"COVID-19: Social Distancing Detection using Deep Learning Techniques" by J. Jeya Joethi et al. (2020).

[4]"A Real-time Social Distancing Detector using Deep Learning and Computer Vision" by Rahul Kumar et al. (2021).

[5]"Social Distancing Monitoring in COVID-19 Pandemic using Deep Learning Techniques" by Amritha Nambiar et al. (2021).

[6]"COVID-19 Social Distancing Monitoring using Deep Learning-based Object Detection Techniques" by Naveen Kumar et al. (2021).

[7] J. Chen, X. Chen, and Y. Zhang, "Real-time object detection based on YOLOv3 algorithm," in 2018 37th Chinese Control Conference (CCC), July 25-27, 2018, Wuhan, China.

[8]Y. Li, W. Li, and X. Li, "A real-time traffic sign detection system based on YOLOv3 algorithm," in 2019 3rd International Conference on Big Data and Internet of Things (BDIOT), Nov. 15-18, 2019, Guangzhou, China.

[9]A. Khaliq, M. A. Raza, and N. Abbas, "A comparative analysis of YOLOv3 and Faster R-CNN for object detection in UAV imagery," in 2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD), Dec. 28-30, 2020, Sousse, Tunisia.

[10]S. Kim, S. Hong, and S. Yoo, "Performance evaluation of YOLOv3-based object detection on the Jetson TX2," in 2019 IEEE International Conference on Consumer Electronics (ICCE), Jan. 11-13, 2019, Las Vegas, NV, USA.

[11] H. Qiu, J. Yao, and Y. Zhang, "A YOLOv3-based fast detection and recognition algorithm for intelligent transportation systems," in 2020 IEEE Intelligent Vehicles Symposium (IV), Oct. 19-23, 2020, Las Vegas, NV, USA.

[12] W. Zhang, Y. Liu, and H. Sun, "Detection of moving targets in videos using YOLOv3 algorithm and feature tracking," in 2021 16th IEEE Conference on Industrial Electronics and Applications (ICIEA), Jun. 11-13, 2021, Guangzhou, China.