PROJECT COMPUTER VISION
"Social Distancing Analyzer Using Computer Vision and Deep Learning"

Bram Marcellino Aritonang (001201900078)
Fitri Anggraini (001201900071)
Vicky Jendra (001201900132)

1. Introduction.

- What is the program about?

  → This program presents a methodology for detecting social distance using deep learning and computer vision between people to control the spread of covid-19. This application gives alerts to people for maintaining social distance in crowded places. By using pre-recorded video as input and the open-source object detection pretrained model using the YOLOv3 (You only look once) algorithm We can tell if people are following social distancing or not and based on that we are creating red or green bounding boxes over it. It is also working on web cameras, CCTV, etc, and can detect people in real-time. This may help authorities to redesign the layout of public places or to take precautionary actions to mitigate high-risk zones.

- In what language is the program implemented?

  → The program that we created uses the Python language to implement the program. Python is a high-level general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

  → Detect humans in the frame Using yolov3. YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. YOLO uses features learned by a deep convolutional neural network to detect an object. For simplicity, we using an open-source pedestrian detection network based on the Yolo v3 architecture. To clean up the output bounding boxes, we apply minimal post-processing such as non-max suppression (NMS) and various rule-based heuristics, so as to minimize the risk of overfitting.

  → Process images and videos to identify objects, faces of a human using OpenCV. OpenCV is a great tool for image processing and performing computer vision tasks.

  → We are using the coco dataset here which is trained on 80 layers but we are using person class here so from 80 layers we are using only person class. COCO (Common Objects in Context) as the image dataset was created with the goal of advancing image recognition.

2. Methodology.

- Input Collection.

  → The video recorded by the CCTV camera is given as the input. The camera is set up in a way it captures at a fixed angle and the video frame's view was changed into a 2D bird's view to accurately estimate the distance between each person. It is taken that the people within the frame are leveled on the horizontal plane. Then, four points from the horizontal plane are chosen, then it is changed into the bird's view. Now the position of each person can be calculated based on the bird's view.

- Calibrating the camera.

  → The region of interest (ROI) of an image or a video frame focused on the person who is walking was captured using a CCTV camera was then changed into a two-dimensional bird's view. The changed view's dimension is 480 pixels on all sides. The calibration is done by transforming the view frame captured into a two-dimensional bird's view. The camera calibration is done straightforwardly using OpenCV. The transformation of view is done using a calibration function that selects 4 points in the input image/video frame and then mapping each point to the edges of the rectangular two-dimensional image frame. Now the interval of each person in the frame can be calculated easily as it corresponds to the total pixels present in between each person in the changed bird's view.

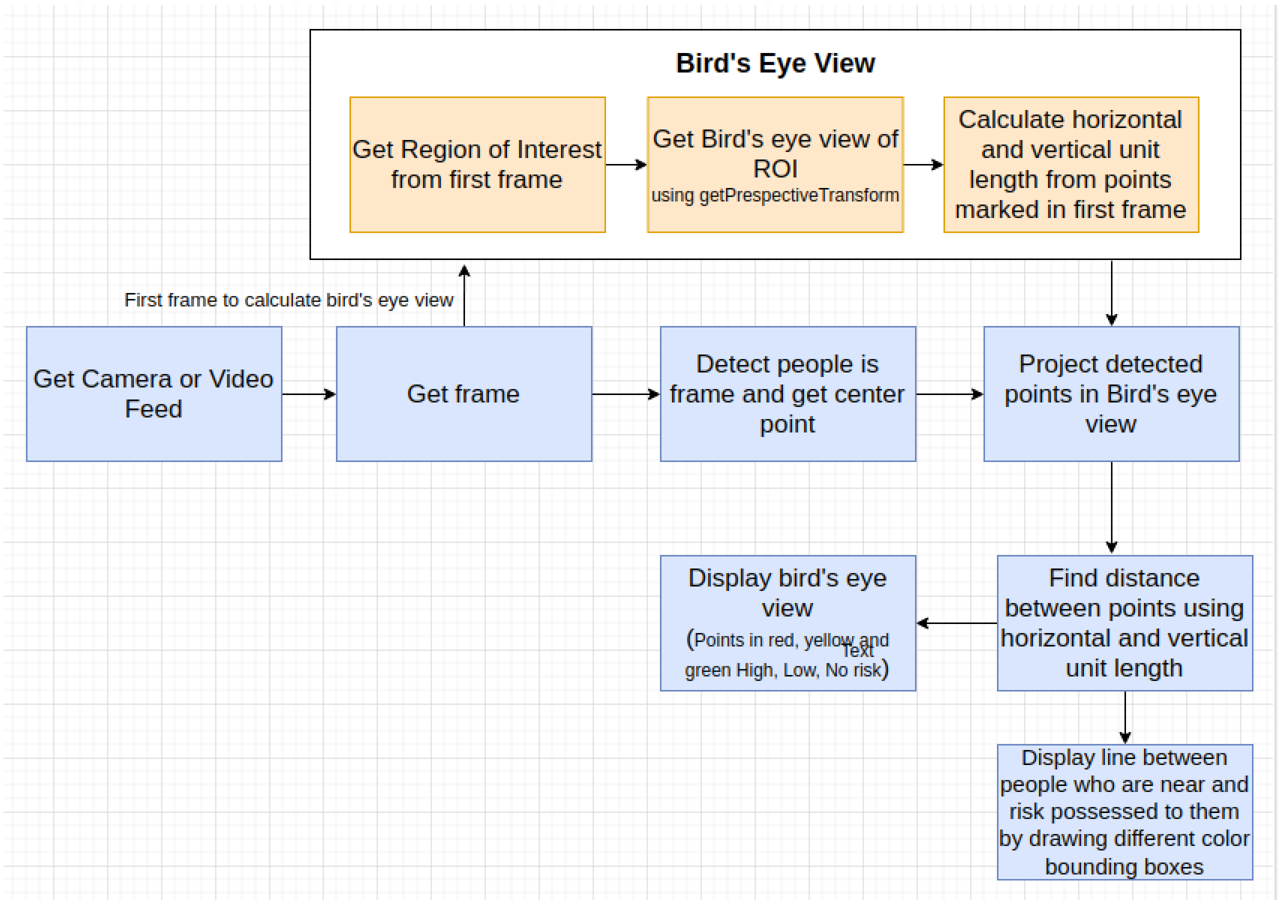
- Detection of pedestrians

  → Deep Convolutional Neural Networks model is a simple and efficient model for object detection. This model considers the region which contains only "Person" class and discards the regions that are not likely to contain any object. The object detection approach used in the Social distancing analyzer model reduces the computational complexity issues. It is done by formulating the detection of objects with the help of a single regression problem. In object detection models based on deep learning, the You Only Look Once (YOLO) model. This model is suitable for real-time applications and it is faster and provides accurate results. The YOLOv3 is an object detection model that takes an image or a video as an input and can simultaneously learn and draw bounding box coordinates, corresponding class label probabilities, and object confidence. The YOLOv3 is an already trained model on the Common Objects in Context dataset (COCO dataset). This dataset consists of 80 labels including a human class known as pedestrian class.

- Measurement Of Distance

→ The precision of the distance measurement between pedestrians is also affected by the pedestrian detection algorithm. We need to estimate person location in frame. We can take bottom center point of bounding box as person location in frame. Then we estimate (x,y) location in bird's eye view by applying transformation to the bottom center point of each person's bounding box, resulting in their position in the bird's eye view.

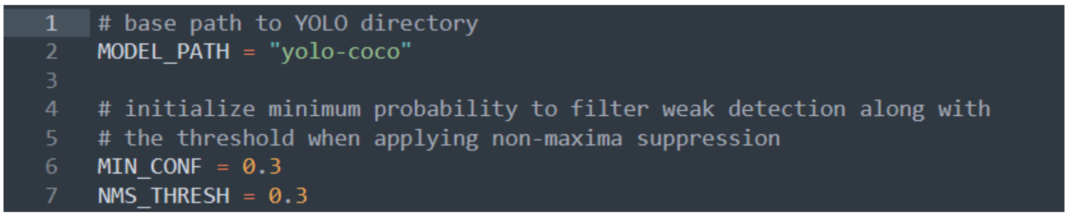
- Code Flow.


Bird's Eye View
Get Region of Interest from first frame
Get Bird's eye view of ROI using getPrespectiveTransform
Calculate horizontal and vertical unit length from points marked in first frame
First frame to calculate bird's eye view
Get Camera or Video Feed
Get frame
Detect people is frame and get center point
Project detected points in Bird's eye view
Display bird's eye view (Points in red, yellow and green High, Low, No risk)
Text
Find distance between points using horizontal and vertical unit length
Display line between people who are near and risk possessed to them by drawing different color bounding boxes

3. Explanation code

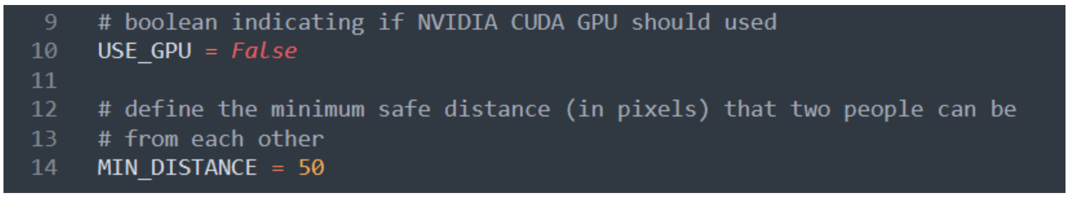
   - social_distancing_config.py

     - To help keep code organized, we use a configuration file to store important variables.



```
1  # base path to YOLO directory
2  MODEL_PATH = "yolo-coco"
3
4  # initialize minimum probability to filter weak detection along with
5  # the threshold when applying non-maxima suppression
6  MIN_CONF = 0.3
7  NMS_THRESH = 0.3
```

→ Here, we have the path to the YOLO object detection model (Line 2). We also define the minimum object detection confidence and non-maxima suppression threshold.



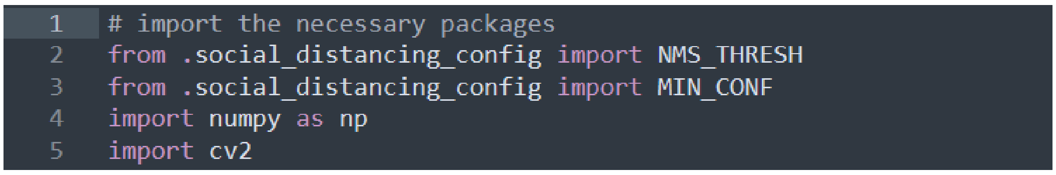
```
9  # boolean indicating if NVIDIA CUDA GPU should used
10 USE_GPU = False
11
12 # define the minimum safe distance (in pixels) that two people can be
13 # from each other
14 MIN_DISTANCE = 50
```

→ The USE_GPU boolean on Line 10 indicates whether your NVIDIA CUDA-capable GPU will be used to speed up inference (requires that OpenCV's "dnn" module be installed with NVIDIA GPU support).

→ Line 14 defines the minimum distance (in pixels) that people must stay from each other in order to adhere to social distancing protocols.

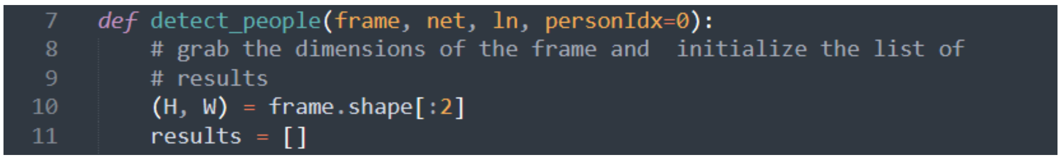
   - detection.py

     - Using YOLO with OpenCV requires a bit more output processing than other object detection methods, implementation of a detect_people function that encapsulates any YOLO object detection logic.



```
1  # import the necessary packages
2  from .social_distancing_config import NMS_THRESH
3  from .social_distancing_config import MIN_CONF
4  import numpy as np
5  import cv2
```

→ We begin with imports, including those needed from our configuration file on Lines 2 and 3 — the NMS_THRESH and MIN_CONF. We also take advantage of NumPy and OpenCV in this script (Lines 4 and 5).



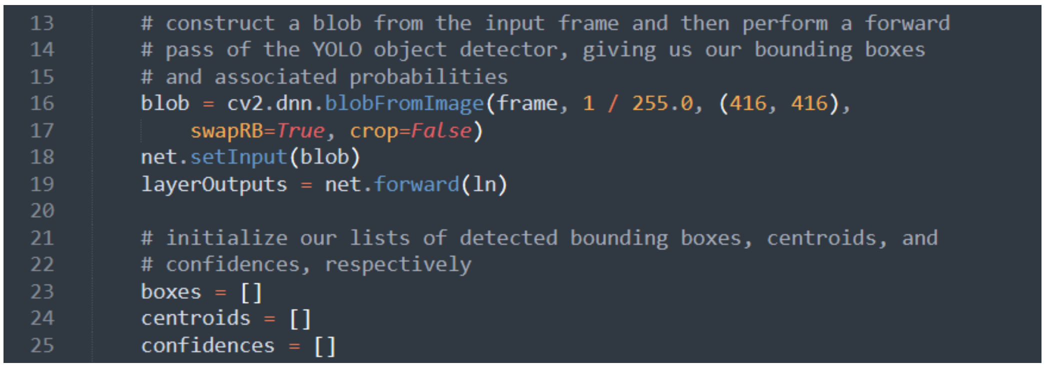
```
 7  def detect_people(frame, net, ln, personIdx=0):
 8      # grab the dimensions of the frame and  initialize the list of
 9      # results
10      (H, W) = frame.shape[:2]
11      results = []
```

→ Beginning on Line 7, we define detect_people; the function accepts four parameters:

- frame: The frame from our video file or directly from webcam
- net: The pre-initialized and pre-trained YOLO object detection model
- ln: The YOLO CNN output layer names
- personIdx: The YOLO model can detect many types of objects; this index is specifically for the person class, as we won't be considering other objects

→ Line 10 grabs the frame dimensions for scaling purposes. Then initialize our results list, which the function ultimately returns. The results consist of

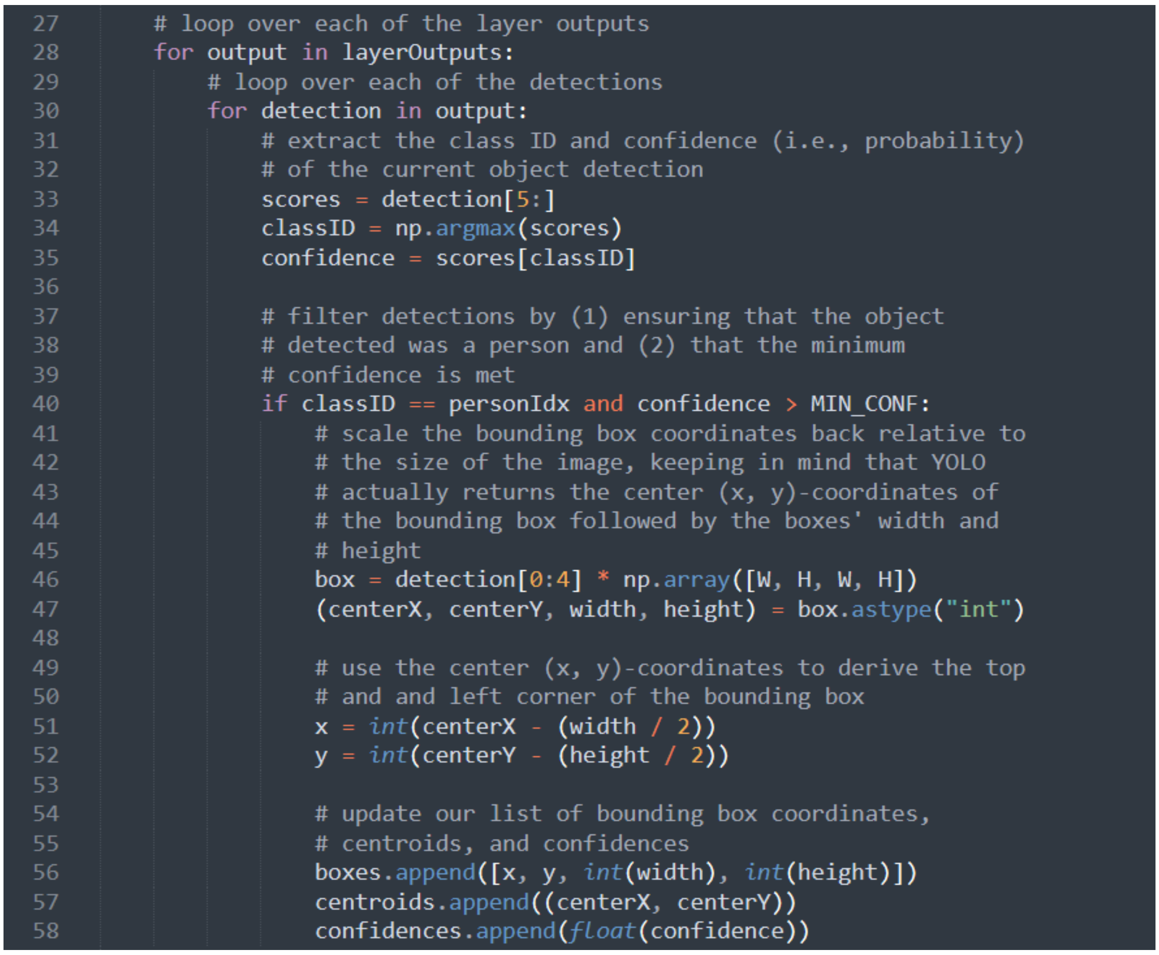
1) the person prediction probability,
2) bounding box coordinates for the detection, and
3) the centroid of the object.



```
13      # construct a blob from the input frame and then perform a forward
14      # pass of the YOLO object detector, giving us our bounding boxes
15      # and associated probabilities
16      blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
17          swapRB=True, crop=False)
18      net.setInput(blob)
19      layerOutputs = net.forward(ln)
20
21      # initialize our lists of detected bounding boxes, centroids, and
22      # confidences, respectively
23      boxes = []
24      centroids = []
25      confidences = []
```

→ Pre-processing our frame requires that we construct a blob (Lines 16 and 17). From there, we are able to perform object detection with YOLO and OpenCV (Lines 18 and 19).

→ Lines 23-25 initialize lists that will soon hold our bounding boxes, object centroids, and object detection confidences.



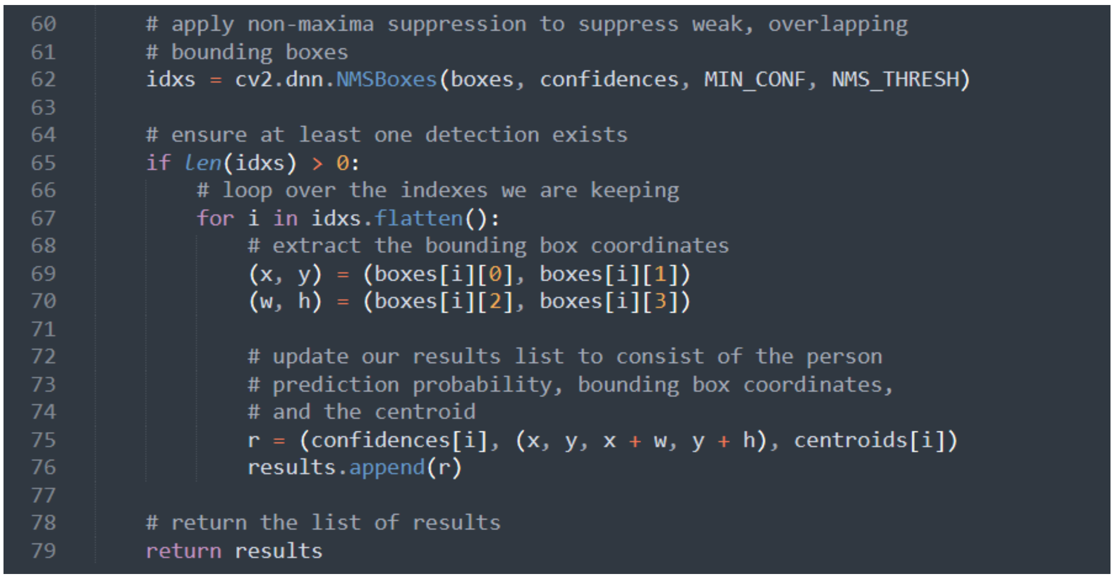
```
27        # loop over each of the layer outputs
28        for output in layerOutputs:
29            # loop over each of the detections
30            for detection in output:
31                # extract the class ID and confidence (i.e., probability)
32                # of the current object detection
33                scores = detection[5:]
34                classID = np.argmax(scores)
35                confidence = scores[classID]
36
37                # filter detections by (1) ensuring that the object
38                # detected was a person and (2) that the minimum
39                # confidence is met
40                if classID == personIdx and confidence > MIN_CONF:
41                    # scale the bounding box coordinates back relative to
42                    # the size of the image, keeping in mind that YOLO
43                    # actually returns the center (x, y)-coordinates of
44                    # the bounding box followed by the boxes' width and
45                    # height
46                    box = detection[0:4] * np.array([W, H, W, H])
47                    (centerX, centerY, width, height) = box.astype("int")
48
49                    # use the center (x, y)-coordinates to derive the top
50                    # and and left corner of the bounding box
51                    x = int(centerX - (width / 2))
52                    y = int(centerY - (height / 2))
53
54                    # update our list of bounding box coordinates,
55                    # centroids, and confidences
56                    boxes.append([x, y, int(width), int(height)])
57                    centroids.append((centerX, centerY))
58                    confidences.append(float(confidence))
```

→ Looping over each of the layerOutputs and detections (Lines 28-30), we first extract the classID and confidence (i.e., probability) of the current detected object (Lines 33-35). From there, we verify that

1) the current detection is a person and
2) the minimum confidence is met or exceeded (Line 40).

→ Assuming so, we compute bounding box coordinates and then derive the center (i.e., centroid) of the bounding box (Lines 46 and 47). Notice how we scale (i.e., multiply) our detection by the frame dimensions we gathered earlier.

→ Using the bounding box coordinates, Lines 51 and 52 then derive the top-left coordinates for the object. We then update each of our lists (boxes, centroids, and confidences) via Lines 56-58.



```
60      # apply non-maxima suppression to suppress weak, overlapping
61      # bounding boxes
62      idxs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONF, NMS_THRESH)
63
64      # ensure at least one detection exists
65      if len(idxs) > 0:
66          # loop over the indexes we are keeping
67          for i in idxs.flatten():
68              # extract the bounding box coordinates
69              (x, y) = (boxes[i][0], boxes[i][1])
70              (w, h) = (boxes[i][2], boxes[i][3])
71
72              # update our results list to consist of the person
73              # prediction probability, bounding box coordinates,
74              # and the centroid
75              r = (confidences[i], (x, y, x + w, y + h), centroids[i])
76              results.append(r)
77
78      # return the list of results
79      return results
```

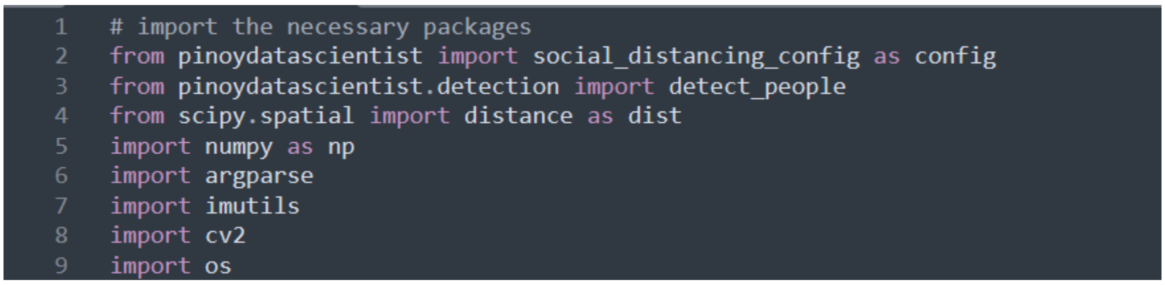
→ Next, we apply non-maxima suppression. The purpose of non-maxima suppression is to suppress weak, overlapping bounding boxes. Line 62 applies this method (it is built-in to OpenCV) and results in the idxs of the detections.

→ Assuming the result of NMS yields at least one detection (Line 65), we loop over them, extract bounding box coordinates, and update our results list consisting of the:

1) Confidence of each person detection
2) Bounding box of each person
3) Centroid of each person

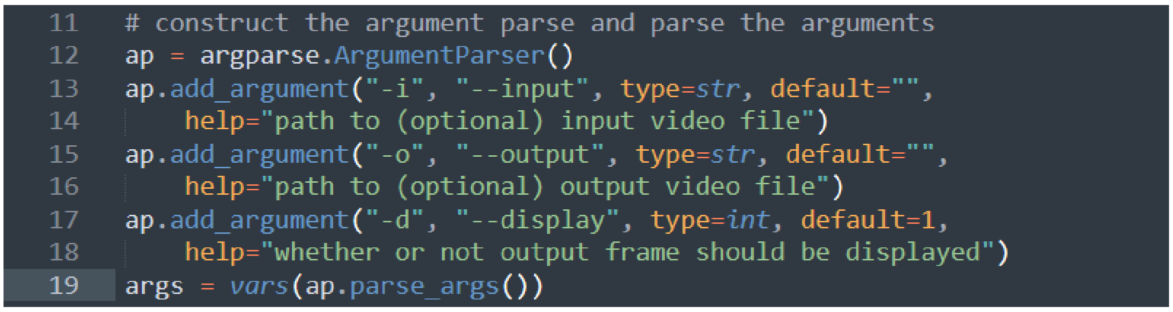
Finally, we return the results to the calling function.

- social_distance_detector.py



```
1  # import the necessary packages
2  from pinoydatascientist import social_distancing_config as config
3  from pinoydatascientist.detection import detect_people
4  from scipy.spatial import distance as dist
5  import numpy as np
6  import argparse
7  import imutils
8  import cv2
9  import os
```

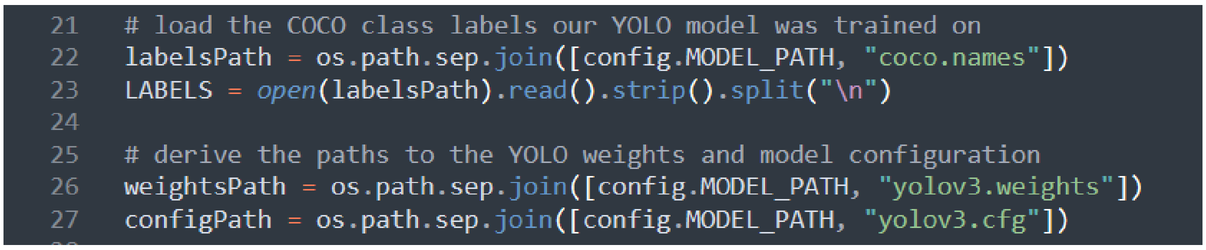
→ The most notable imports on Lines 2-9 include our config, our detect_people function, and the Euclidean distance metric (shortened to dist and to be used to determine the distance between centroids).



```
11  # construct the argument parse and parse the arguments
12  ap = argparse.ArgumentParser()
13  ap.add_argument("-i", "--input", type=str, default="",
14      help="path to (optional) input video file")
15  ap.add_argument("-o", "--output", type=str, default="",
16      help="path to (optional) output video file")
17  ap.add_argument("-d", "--display", type=int, default=1,
18      help="whether or not output frame should be displayed")
19  args = vars(ap.parse_args())
```

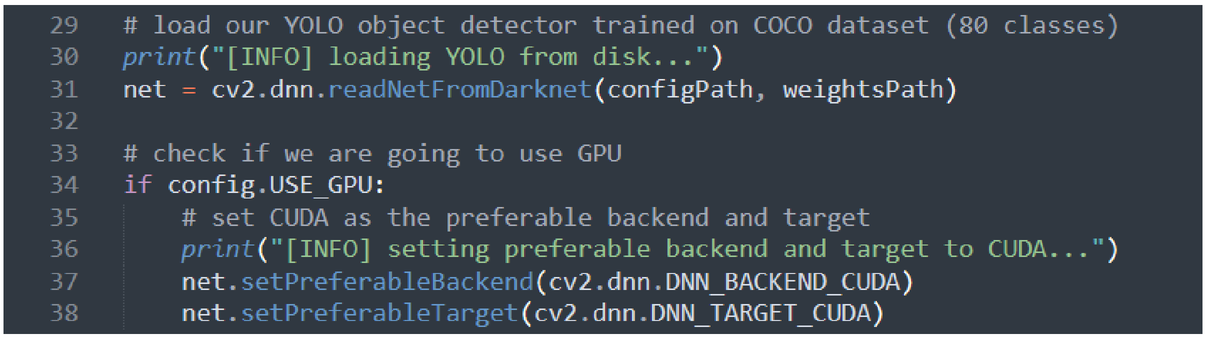
→ This script requires the following arguments to be passed via the command line/terminal:

- input: The path to the optional video file. If no video file path is provided, your computer's first webcam will be used by default.
- output: The optional path to an output (i.e., processed) video file. If this argument is not provided, the processed video will not be exported to disk.
- display: By default, we'll display our social distance application on-screen as we process each frame. Alternatively, you can set this value to 0 to process the stream in the background.



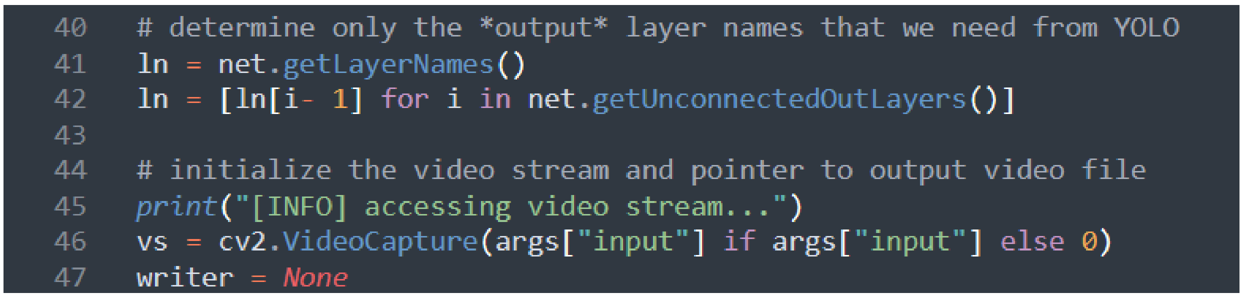
```
21  # load the COCO class labels our YOLO model was trained on
22  labelsPath = os.path.sep.join([config.MODEL_PATH, "coco.names"])
23  LABELS = open(labelsPath).read().strip().split("\n")
24
25  # derive the paths to the YOLO weights and model configuration
26  weightsPath = os.path.sep.join([config.MODEL_PATH, "yolov3.weights"])
27  configPath = os.path.sep.join([config.MODEL_PATH, "yolov3.cfg"])
```

→ Here, we load our load COCO labels (Lines 22 and 23) as well as define our YOLO paths (Lines 26 and 27).



```
29  # load our YOLO object detector trained on COCO dataset (80 classes)
30  print("[INFO] loading YOLO from disk...")
31  net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
32
33  # check if we are going to use GPU
34  if config.USE_GPU:
35      # set CUDA as the preferable backend and target
36      print("[INFO] setting preferable backend and target to CUDA...")
37      net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
38      net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
```

→ Using OpenCV's DNN module, we load our YOLO net into memory (Line 31). We have the USE_GPU option set in the config, then the backend processor is set to be our NVIDIA CUDA-capable GPU. because I don't have a CUDA-capable GPU, ensure that the configuration option is set to False so that my CPU is the processor used.

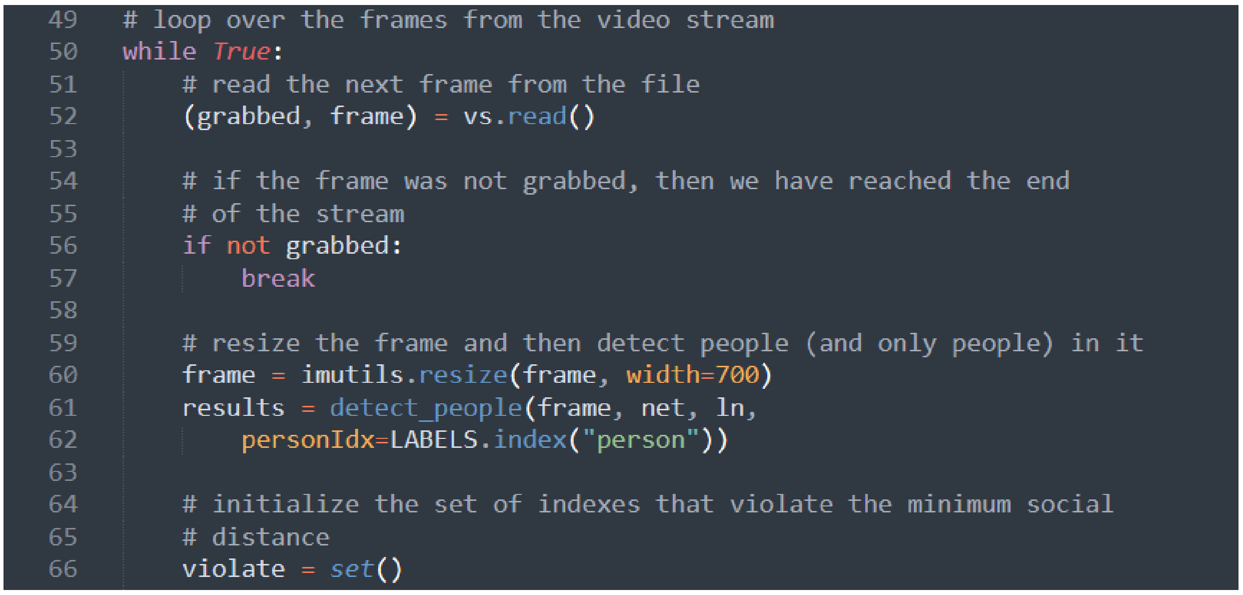


```
40  # determine only the *output* layer names that we need from YOLO
41  ln = net.getLayerNames()
42  ln = [ln[i- 1] for i in net.getUnconnectedOutLayers()]
43
44  # initialize the video stream and pointer to output video file
45  print("[INFO] accessing video stream...")
46  vs = cv2.VideoCapture(args["input"] if args["input"] else 0)
47  writer = None
```

→ Here, Lines 41 and 42 gather the output layer names from YOLO; we'll need them in order to process our results.

→ We then start our video stream (either a video file via the --input command line argument or a webcam stream) Line 46.

→ For now, we initialize our output video writer to None. Further setup occurs in the frame processing loop.



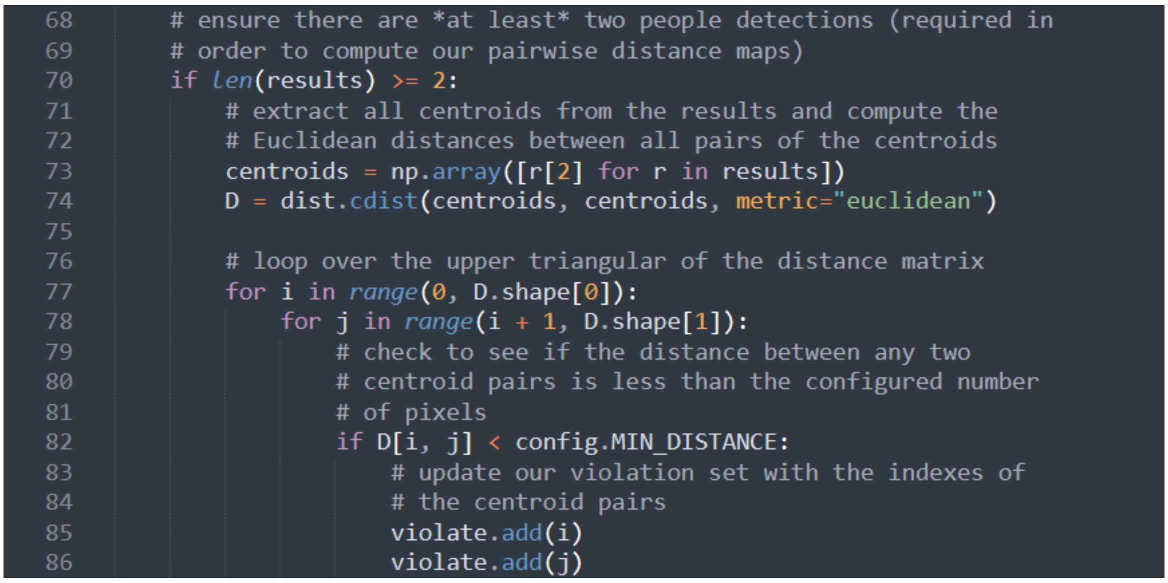
```
49  # loop over the frames from the video stream
50  while True:
51      # read the next frame from the file
52      (grabbed, frame) = vs.read()
53
54      # if the frame was not grabbed, then we have reached the end
55      # of the stream
56      if not grabbed:
57          break
58
59      # resize the frame and then detect people (and only people) in it
60      frame = imutils.resize(frame, width=700)
61      results = detect_people(frame, net, ln,
62          personIdx=LABELS.index("person"))
63
64      # initialize the set of indexes that violate the minimum social
65      # distance
66      violate = set()
```

→ Lines 50-52 begin a loop over frames from our video stream.

→ The dimensions of our input video for testing are quite large, so we resize each frame while maintaining the aspect ratio (Line 60).

→ Using our detect_people function implemented in the previous section, we grab the results of YOLO object detection (Lines 61 and 62).

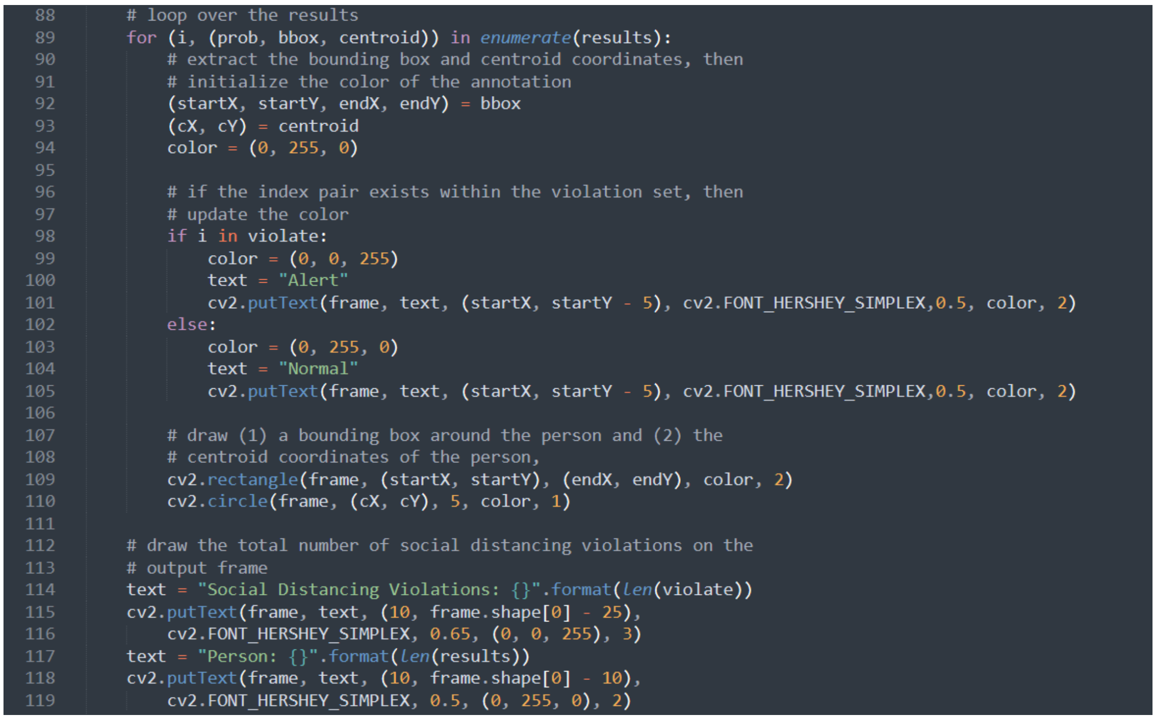
→ We then initialize our violate set on Line 66; this set maintains a listing of people who violate social distance regulations set forth by public health professionals.



```python
68      # ensure there are *at least* two people detections (required in
69      # order to compute our pairwise distance maps)
70      if len(results) >= 2:
71          # extract all centroids from the results and compute the
72          # Euclidean distances between all pairs of the centroids
73          centroids = np.array([r[2] for r in results])
74          D = dist.cdist(centroids, centroids, metric="euclidean")
75
76          # loop over the upper triangular of the distance matrix
77          for i in range(0, D.shape[0]):
78              for j in range(i + 1, D.shape[1]):
79                  # check to see if the distance between any two
80                  # centroid pairs is less than the configured number
81                  # of pixels
82                  if D[i, j] < config.MIN_DISTANCE:
83                      # update our violation set with the indexes of
84                      # the centroid pairs
85                      violate.add(i)
86                      violate.add(j)
```

→ Assuming that at least two people were detected in the frame (Line 70), we proceed to:

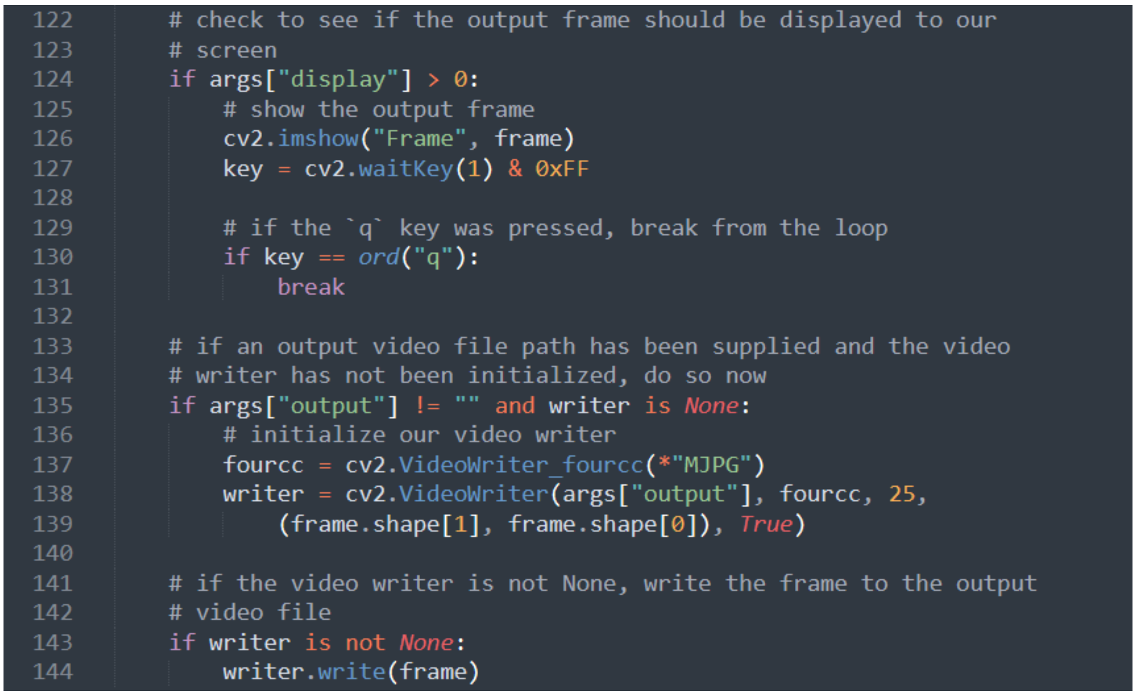
- Compute the Euclidean distance between all pairs of centroids (Lines 73 and 74)
- Loop over the upper triangular of distance matrix (since the matrix is symmetrical) beginning on Lines 77 and 78
- Check to see if the distance violates our minimum social distance set forth by public health professionals (Line 82). If two people are too close, we add them to the violate set



```python
88      # loop over the results
89      for (i, (prob, bbox, centroid)) in enumerate(results):
90          # extract the bounding box and centroid coordinates, then
91          # initialize the color of the annotation
92          (startX, startY, endX, endY) = bbox
93          (cX, cY) = centroid
94          color = (0, 255, 0)
95
96          # if the index pair exists within the violation set, then
97          # update the color
98          if i in violate:
99              color = (0, 0, 255)
100             text = "Alert"
101             cv2.putText(frame, text, (startX, startY - 5), cv2.FONT_HERSHEY_SIMPLEX,0.5, color, 2)
102         else:
103             color = (0, 255, 0)
104             text = "Normal"
105             cv2.putText(frame, text, (startX, startY - 5), cv2.FONT_HERSHEY_SIMPLEX,0.5, color, 2)
106
107         # draw (1) a bounding box around the person and (2) the
108         # centroid coordinates of the person,
109         cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
110         cv2.circle(frame, (cX, cY), 5, color, 1)
111
112     # draw the total number of social distancing violations on the
113     # output frame
114     text = "Social Distancing Violations: {}".format(len(violate))
115     cv2.putText(frame, text, (10, frame.shape[0] - 25),
116         cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 0, 255), 3)
117     text = "Person: {}".format(len(results))
118     cv2.putText(frame, text, (10, frame.shape[0] - 10),
119         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

→ Looping over the results on Line 89, we proceed to:

- Extract the bounding box and centroid coordinates (Lines 92 and 93)
- Initialize the color of the bounding box to green (Line 94)
- Check to see if the current index exists in our violate set, and if so, update the color to red (Lines 98 and 99) then give a text warning above the box (Lines 100 and 101)
- Check if the index is not currently in our violation set, then update the color to green (Lines 102 and 103) then give a text normal above the box (Lines 104 and 105)
- Draw both the bounding box of the person and their object centroid (Lines 109 and 110). Each is color-coordinated, so we'll see which people are too close.
- Display information on the total number of social distancing violations (the length of our violate set (Lines 114-116)
- Display information on the total number of people on the screen (People (Lines 117-119)



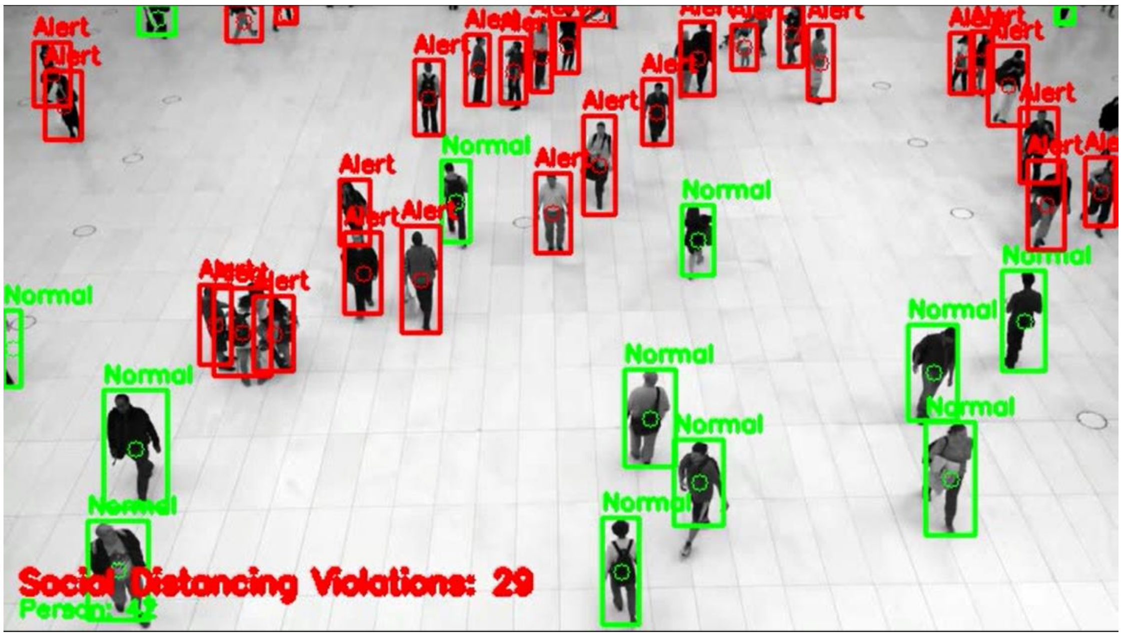
```
122        # check to see if the output frame should be displayed to our
123        # screen
124        if args["display"] > 0:
125            # show the output frame
126            cv2.imshow("Frame", frame)
127            key = cv2.waitKey(1) & 0xFF
128
129            # if the `q` key was pressed, break from the loop
130            if key == ord("q"):
131                break
132
133        # if an output video file path has been supplied and the video
134        # writer has not been initialized, do so now
135        if args["output"] != "" and writer is None:
136            # initialize our video writer
137            fourcc = cv2.VideoWriter_fourcc(*"MJPG")
138            writer = cv2.VideoWriter(args["output"], fourcc, 25,
139                (frame.shape[1], frame.shape[0]), True)
140
141        # if the video writer is not None, write the frame to the output
142        # video file
143        if writer is not None:
144            writer.write(frame)
```

→ To close out, we Display the frame to the screen if required (Lines 124-126) while waiting for the q (quit) key to be pressed (Lines 127-131)

→ Initialize our video writer if necessary (Lines 135-139)

→ Write the processed (annotated) frame to disk (Lines 143 and 144)

4. Results



Here, you can see that I was able to process the entire and as the results show, our social distancing detector is correctly marking people who violate social distancing rules.

5. Conclusion.

- A methodology of social distancing detection tool using a deep learning model is proposed. By using computer vision, the distance between people can be estimated and any non-compliant pair of people will be indicated with a red frame and red alert. The proposed method was validated using a video showing pedestrians walking on a street. The visualization results showed that the proposed method is capable of determining the social distancing measures between people.

References

G V Shalini et a | Journal of Physics: Conference Series. Retrieved March 8, 2022, from https://iopscience.iop.org/article/10.1088/1742-6596/1916/1/012039/pdf

International journal of innovation in engineering research and technology. Retrieved March 8, 2022, from https://repo.ijiert.org/index.php/ijiert/article/view/2569/2354

Deepak Birla | Social Distancing AI using Python, Deep Learning and Computer Vision.Retrieved March 8, 2022, from https://medium.com/@birla.deepak26/social-distancing-ai-using-python-deep-learning-c26b20c9aa4c

PyImageSearch | Adrian Rosebrock, PhD. *OpenCV Social Distancing Detector*. Retrieved March 8, 2022, from https://pyimagesearch.com/2020/06/01/opencv-social-distancing-detector/