CSC420 Project: Social Distancing Detection

Group: lMengineer

Member: Yiyang Hua(1003201475),

Wenjie Hao(1002183059),

Chunjing Zhang(1003501472)

Table of Contents

1. Abstract

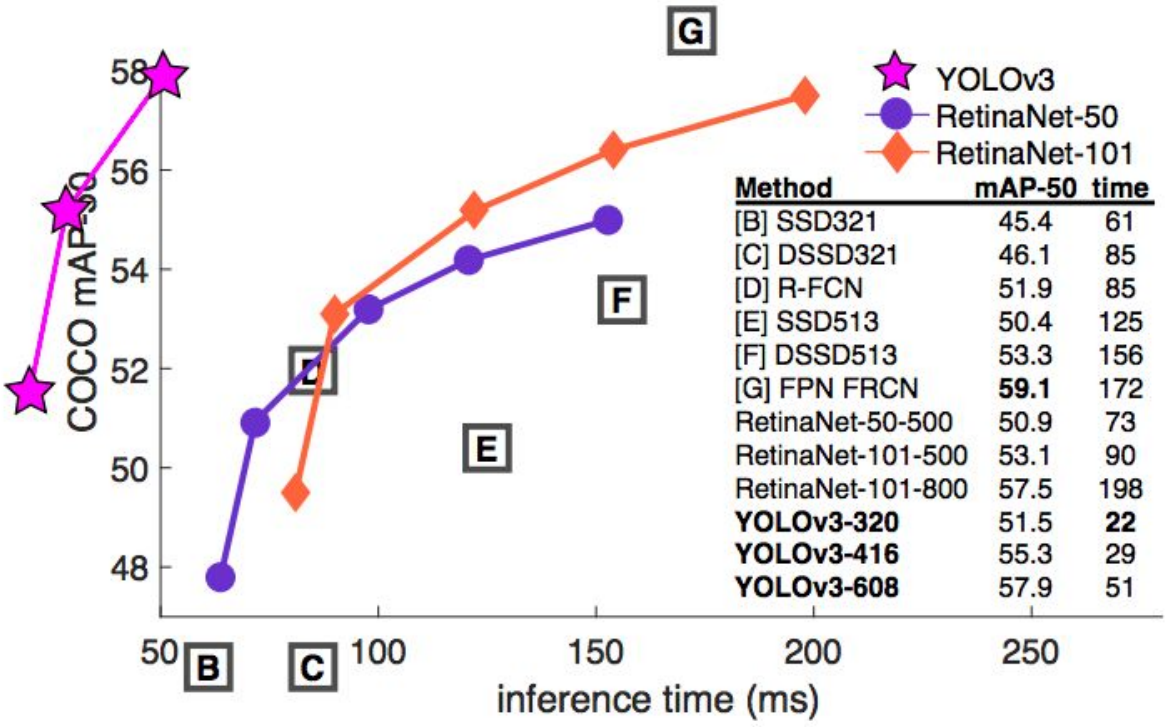
Because of COVID-19, keeping social distance has become essential for the whole world. In order to ensure people's safety, we should keep physical distance and reduce close contact, which can effectively reduce the spread of infectious diseases. The social distance detector our team developed is one tool that can identify those people who have violated the social distancing norm of 2 meters. Our detector uses YOLOv3 for human detection as YOLOv3 is a more balanced model that achieves both efficient performance and high accuracy compared to other state-of-art algorithms such as faster region-based CNN (faster RCNN) and single shot detector(SSD). Additionally, our detector involves a novel implementation in finding the pixel distance of the field environment that does not require users to manually determine the pixel distance. The detector could automatically estimate the unit pixel distance based on height and width of a selected human detected by YOLOv3. This implementation provides more flexibility for the users to use. After comparing the video output produced by our detector with the video output produced by the social distance detector developed by Landing AI company which is established by the reputable scientist Andrew Ng, we found that our video output gives similar results as theirs. This means that our detector has a considerably high level of accuracy in detecting the violation social distance. Therefore, our social distance detector is tested to be flexible to use, meanwhile, is accurate in detecting social distance violations.

2. Introduction and literature review

Our solution for detecting violations of social distance is developing a social distance detector that will box the people who have violated the social distance norm of 2 meters in red and box the people who have not violated the social distance in green.

In developing the detector, our team has made two major design decisions. The first major design decision is using VOLOv3 with OpenCV2 for human detection

over other object detection algorithms such as faster RCNN and SSD. This is because VOLOv3 is the most balanced model that achieves both efficient performance and accuracy compared to others. According to the paper presented by Joseph Redmon and Ali Farhadi in "YOLOv3: An Incremental Improvement", it states that YOLOv3 could achieve comparable accuracy (in terms of mAP), but considerably faster speed than other algorithms as illustrated in the graph below, at COCO mAP 50 benchmark. **(Redmon & Farhadi, 2018) [1]**



(Figure 1: mAP and time of running different object detection algorithms, sourced from https://pjreddie.com/media/files/papers/YOLOv3.pdf) **(Redmon & Farhadi, 2018) [1]**

Additionally, in the experiments done in the paper, "Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLOv3 and Deepsort techniques", issued by Narinder Singh Punn, Sanjay Kumar Sonbhadra and Sonali Agarwa, it also verified that although some algorithms such as faster RCNN could achieve "minimal loss with maximum mAP, however, has the lowest FPS, which makes it not suitable for real-time applications. Furthermore, as compared to SSD, YOLOv3 achieved better results with balanced mAP, training time, and FPS score." **(Singh Punn, Kumar Sonbhadra & Agarwal, 2020) [2]**

It’s essential for a social distance detector to be able to detect social distance violations in real-time, and thus the speed of detection is even more important than the accuracy of detection. Therefore, YOLOv3 is certainly the best candidate amongst all of our choices.

The second major design decision is automatically approximating the unit pixel distance of the bird eye view for the video instead of manually setting the pixel distance by the detector users. This implementation could make the detector more flexible to be used in different locations without asking users to go to real fields to obtain the real scale and measurement of the field. Our implementation is based on two assumptions. This first assumption is that a person’s real height is 170 cm and it is the height of the box produced by the VOLOv3. This second assumption is that a person’s real width of shoulder is represented by the width of the box produced by the VOLOv3. Based on these assumptions, the unit pixel distance can be easily deducted given a reference target produced by YOLOv3 which represents the detected person.

There is one major challenge for implementing the idea. The challenge is that the selected person for reference cannot be an outlier who cannot represent a typical person’s size facing front. For example, the selected person could be completely standing sideways, or a person with outlier size is selected. Then, the improper reference target could lead to an inaccurate unit pixel deduction which is not expected. In order to address this challenge, our detector loops over all the frames of the inputted video to collect all the boxes that represent detected people. Then, it seletect the box with median width-to-height ratio as the reference target. This design could eliminate the possibility of selecting an outlier or improper reference.

At the end, in order to test for our detector’s accuracy, we compare our output video with the output video produced by the social distance detector developed by the Landing AI company which is established by Andrew Ng. The result is that our output video delivers similar results as theirs. This means that our social distance detector could achieve high accuracy in detecting social distance violations.

The implementation of the detector is mainly composed of five steps:

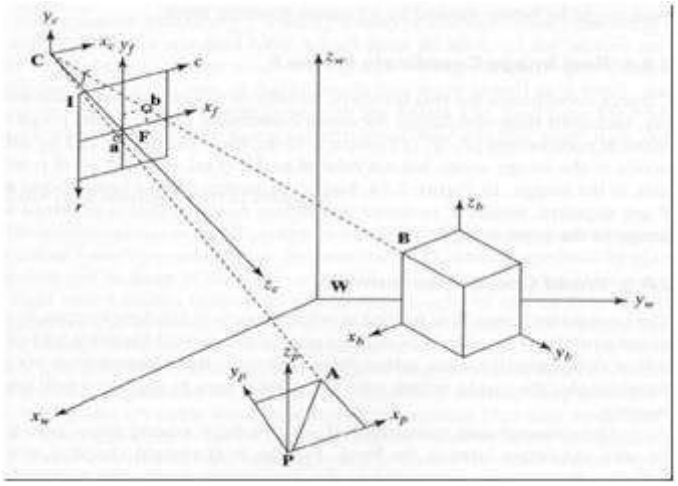
- The first step is setting up the YOLOv3 with OpenCV and filling it with the pre-trained model that would be further used for human detection. The input image of the neural network needs to be in some format called blob. After reading the frame from the input image or video stream, it will convert it into the input blob of the neural network through the blobFromImage function. Then the output blob is passed to the network as its input, and forward pass is run to get the predicted bounding box list as the output of the network.
- The second step is finding the perspective transformation matrix which can transform the camera view into bird eye view. Under bird eye view, the position of people can be illustrated clearly and subsequently the distance between people can be retrieved easily. The built-in function cv2.getPerspectiveTransform() needs 4 source points and 4 destination points. We chose to let the user pick the 4 source points of the region that need to be transformed to bird eye view. Among these 4 points, 3 of them should not be collinear. At the same time, the order of input should be bottom-right, up-right, up-left and bottom-left.
- The third step is finding estimated unit pixel distance. In this part, it will loop over all frames in the video to extract boxes outputted by YOLOv3 which represent detected humans to find the most suitable shoulder width to height ratio. We assume that one person's height is 170cm. When we find the appropriate ratio of shoulder width to height, we can calculate the pixel distance of 2 meters.
- The fourth step is looping over each frame and transforming the position of each detected person into the position under a bird eye view. For each frame, it needs to check for pairwise distance of people by using the Pythagorean theorem and compare with the previous pixel distance of 2 meters, which is the safety distance. After that, categorize people as safe and unsafe.
- The final step is computing the corresponding bird eye view with coloring each plot correctly and drawing a bounding box with the right color for every person. For each pair with unsafe distances, a red line will appear between them. Our codes will write all frames to corresponding folders.

3. Methodology, Results, and Experiments

● Theoretical Explanation

1. Perspective Transform

Perspective projection or perspective transformation is a projection on a 3D plane. This can cause distant objects to appear smaller than nearby objects.



(Figure 2: Illustration of Perspective Transformation) **(Nayak, 2018)[3]**

From the dimension level:

In order to analyze a 3d image, we must have 5 different frames of references.

3D coordinate system:

I. “Object coordinate frame is used for modeling objects. “[5]

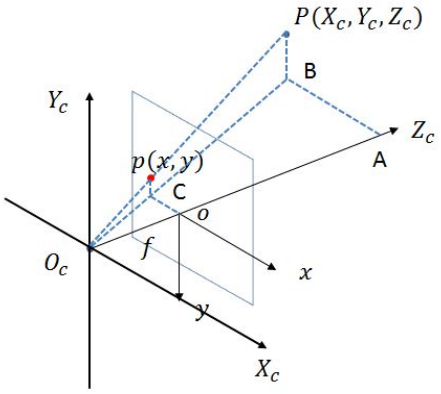
II. The world coordinate frame is used for interrelated objects in the 3D world. It selects a reference coordinate system in the environment to describe the position of camera and object. The relationship between camera coordinate system and world coordinate system can be described by rotation matrix R and translation vector t, assuming that the coordinates of P in the world coordinate system are ***(X, Y, Z)_w***.

Then there is a transformation relationship system between the camera coordinate system and the world coordinate system as follows: **(MADALI, 2020)[5]**

$$\begin{Bmatrix} X \\ Y \\ Z \\ 1 \end{Bmatrix}_c = \begin{Bmatrix} R & t \\ 0 & 1 \end{Bmatrix} \begin{Bmatrix} X \\ Y \\ Z \\ 1 \end{Bmatrix}_w$$

(Figure 3: conversion relationship between the camera coordinate system and the world coordinate system**(MADALI, 2020)[5]**)

III. The camera coordinate system is used to relate objects to the camera. The camera coordinate system is shown in the figure below:[5]



(Figure 4: camera coordinate system)

"The origin of the camera coordinate system is the optical center, the Xc and Yc axes are parallel to the u axis and the v axis of the pixel coordinate system, and the Zc axis is the optical axis of the camera. The distance from the optical center to the pixel plane is the focal length f. It can be seen from the figure that there is a perspective projection relationship between the points on the camera coordinate system and the points on the imaging plane coordinate system. Assuming that the coordinate of the point P in the camera coordinate system corresponding to p is (Xc,Yc,Zc), there is the following conversion relationship between the imaging plane coordinate system and the camera coordinate system": **(MADALI, 2020)[5]**

$$\begin{cases} x = f\dfrac{X_c}{Z_c} \\ y = f\dfrac{Y_c}{Z_c} \end{cases} \Rightarrow Z_c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

(Figure 5: conversion relationship between the imaging plane coordinate system and the camera coordinate system **(MADALI, 2020)[5]**)

2D coordinate system:

IV. Image coordinate system is used to describe the mapping of 3D points in a 2D image plane. Suppose that the coordinates of the imaging plane corresponding to P are (x, y), dx and dy represent the physical size of each pixel in the image plane. The coordinates of the origin of the imaging plane in the pixel coordinate system are (u_0, v_0). Then the conversion formula between the pixel coordinate system and imaging plane coordinate system is as follows:**(MADALI, 2020)[5]**

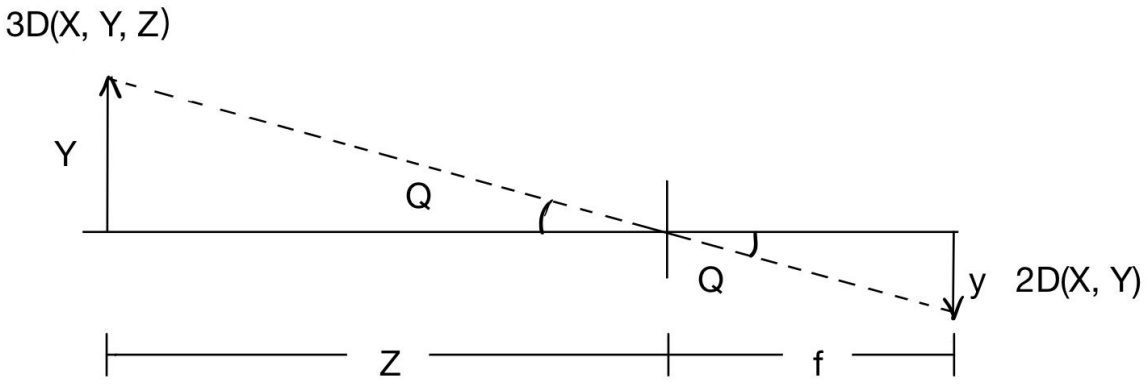
$$\begin{cases} u = \dfrac{x}{dx} + u_0 \\ v = \dfrac{y}{dy} + v_0 \end{cases} \Rightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

(Figure 6: conversion relationship between the pixel coordinate system and the imaging plane coordinate system**(MADALI, 2020)[5]**)

V. On the plane of the pixel coordinate system, every pixel has a value of pixel coordinates.

From a mathematical point of view:

In mathematics, this problem can be described as the following image.



(Figure 7: mathematical point of view of Perspective Transform)

Where Y is a 3D object, y is a 2D image, f is the focal length of the camera, and Z is the distance between the object and the camera. There are two different angles formed in this transform which are represented by Q.

The two angles are $tan\,\theta = \frac{Y}{Z}$ and $tan\,\theta = -\frac{y}{f}$, where the negative sign represents that image is inverted.

After combining these two equation, we can get $y = -f(\frac{Y}{Z})$.

This equation means that when light is reflected off an object, it comes from the camera and forms an inverted image.

From the view of CS students:

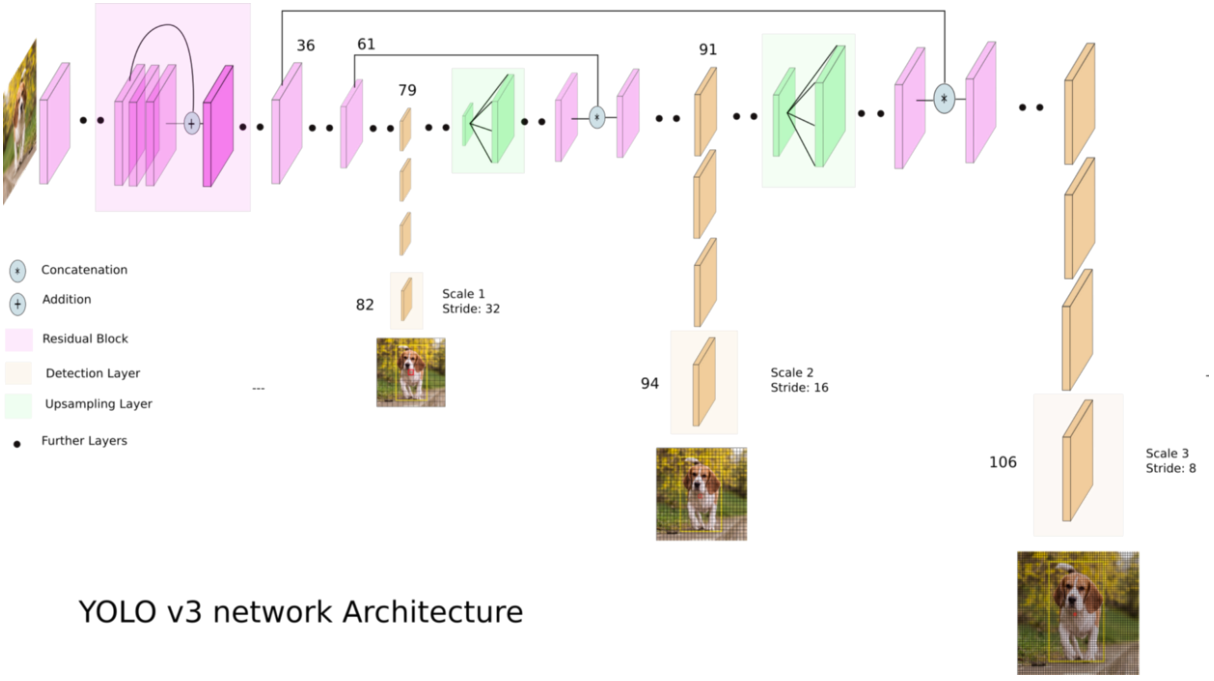
To write code for perspective transformation, it needs a 3x3 transformation matrix. We want even after the modification, the straight line will remain straight. In OpenCV, it has a built-in function cv2.getPerspectiveTransform() to help us compute the transformation matrix and do the perspective transform easily. To call cv2.getPerspectiveTransform(), we need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then the transformation matrix can be found. By checking the documentation of cv2.perspectiveTransform(), we found cv2.warpPerspective() is used to transform the image, and cv2.perspectiveTransform() is used to transform points in the image. Since we only want to transform the coordinates of all detected people, we only need to draw a white image as the base of the bird eye view and transform every

object in the image to the plot in the bird eye view by using cv2.perspectiveTransform(). **("cv2.perspectiveTransformation()-OpenCV") [4]**

2. YOLOv3

YOLOv3 Network Architecture:

"Darknet originally had a 53 layer network trained on Imagenet. For the detection task, more than 53 layers are superimposed on it, giving us a 106 layer full convolution infrastructure for YOLOv3."**(Kathuria, 2018)[7]** The following figure is what the architecture of YOLOv3 looks like.



(Figure 8: YOLOv3 Network Architecture) **(Kathuria, 2018) [7]**

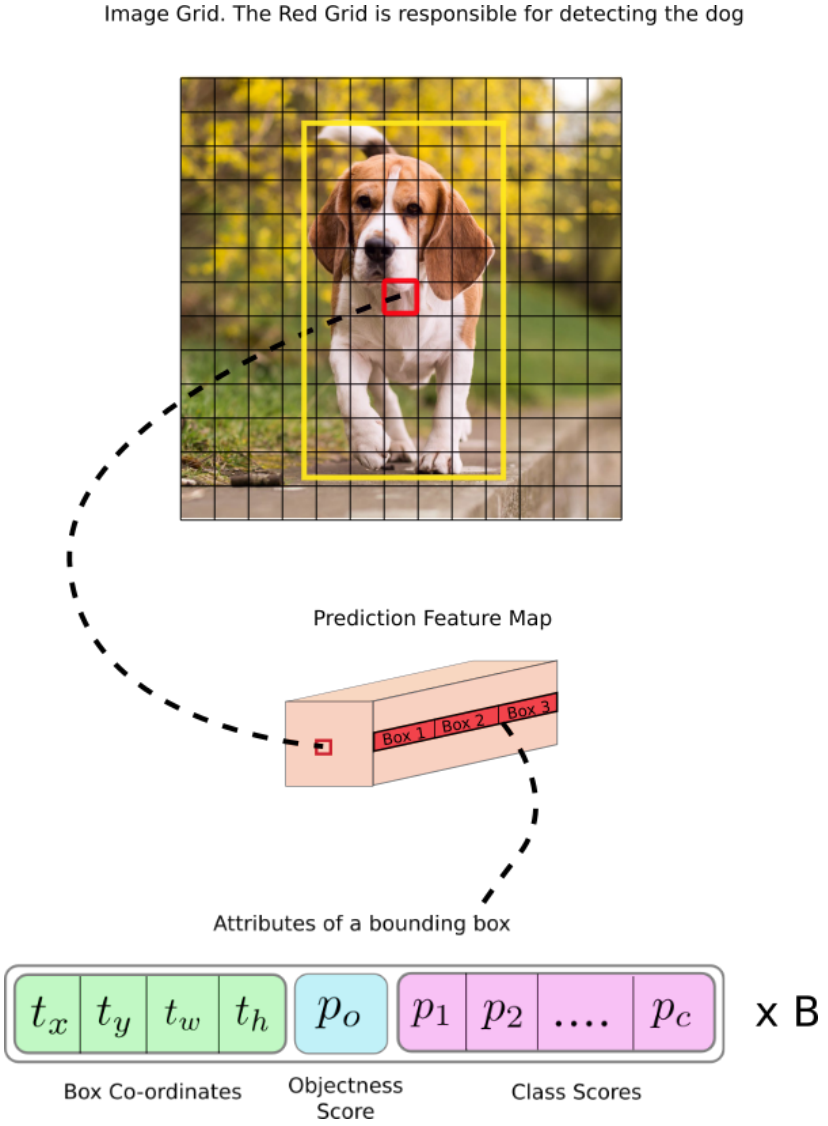
Detection Kernel:

"In YOLOv3, the detection is done by applying *1 x 1* detection kernels on feature maps of three different sizes at three different places in the network.

The shape of the detection kernel is *1 x 1 x (B x (5 + C))*. " **(Kathuria, 2018) [7]**

YOLOv3 is pre-trained on COCO, the number of bounding boxes a cell on the feature map can predict, which is B, is 3 and the number of classes, C, is 80, so the kernel size is 1 x 1 x 255. The feature map generated by the kernel has

the same height and width as the previous feature mapping, and has the detection attribute along the depth as described above. **(Kathuria, 2018) [7]**



(Figure 9: YOLOv3 detection kernel) **(Kathuria, 2018)[7]**

Loss Function:

"The last three terms in the loss function of YOLOv2 are the squared errors, whereas in YOLOv3, they've been replaced by cross-entropy error terms." **(Kathuria, 2018) [7]** So the loss function of YOLOv3 will look like as the follow image:

$$Loss = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} [(x_i - \hat{x}_i^j)^2 + (y_i - \hat{y}_i^j)^2] +$$
$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} [(\sqrt{w_i^j} - \sqrt{\hat{w}_i^j})^2 + (\sqrt{h_i^j} - \sqrt{\hat{h}_i^j})^2] -$$
$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} [\hat{C}_i^j \log(C_i^j) + (1 - \hat{C}_i^j) \log(1 - C_i^j)] -$$
$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} [\hat{C}_i^j \log(C_i^j) + (1 - \hat{C}_i^j) \log(1 - C_i^j)] -$$
$$\sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in classes} ([\hat{P}_i^j \log(P_i^j) + (1 - \hat{P}_i^j) \log(1 - P_i^j)]$$

(Figure 10: YOLOv3 Loss Function) **(dlut_yan, 2020) [6]**

- Empirical Work

```
import cv2
import numpy as np
import glob
```

Function: get_perspective_transform(points, height, width):

Input description:

- ***points:*** the 4 points of the region that the user picks, the main function will ask the user to give those points as inputs
- ***height*****:** the height of the video input
- ***width***: the width of the video input

Output description:

- ***return***: the transformation matrix between the region that the user picks in the video and the corresponding bird eye view

This function is written for finding the transformation matrix between the region that the user picks in the video and the corresponding bird eye view.

```
# points: the 4 points of the region that we picked by hand
def get_perspective_transform(points, height, width):
        source = np.float32(np.array(points))
        destination = np.float32([[0, height], [width, height], [width, 0], [0, 0]])
        perspective_transform = cv2.getPerspectiveTransform(source, destination)
        return perspective_transform
```

Function: get_perspective_points(human_boxes,prespective_transform)

Input description:

- ***human_boxes*:** list of lists, every element is the data of one detected human bounding box, [x of left border, y of top border, width, height]
- ***perspective_transform*:** the transformation matrix that compute by *get_perspective_transform()*

Output description:

- ***return*:** corresponding plots of all detected humans in bird eye view

This function is written for computing all detected humans to corresponding plots in the bird eye view. For every detected human, we will find the bottom center point and use cv2.perspectiveTransform() to transform it to the corresponding plot in the bird eye view.

```
# human_boxes: the points of bounding box of all detected human, output of yolov3
def get_perspective_points(human_boxes, prespective_transform):
        perspective_points = []
        for human in human_boxes:
                points = np.array([[[int(human[0]+(human[2]*0.5)),int(human[1]+human[3])]]], dtype="float32")
                perspective_point = cv2.perspectiveTransform(points, prespective_transform)[0][0]
                perspective_points += [(int(perspective_point[0]), int(perspective_point[1]))]
        return perspective_points
```

Function: bird_eye_view(height, width, measured_distance, perspective_points)

Input description:

- ***height*:** the height of the video input
- ***width*:** the width of the video input
- ***measured_distance***: the output of *measure_distance()*, list of lists, every element is [distance, the perspective point of human1, the perspective of human2]
- ***perspective_points*:** the output of *get_perspective_points()*

Output description:

- ***return***: the corresponding bird eye view frame and the number of people that have unsafe distance

This function is the main function to compute bird eye view. First, it draws a white picture with the given height and width. And then it checks distances between all pairs of 2 people, and divides them to safe and unsafe. If one pair of 2 people has unsafe distance, the codes will draw a red line between them. Finally, it draws green plots for all people that have safe distance, and red plots for all people that have unsafe distance.

```
def bird_eye_view(height, width, measured_distance, perspective_points):
        red = (0, 0, 255)
        green = (0, 255, 0)
        white = (200, 200, 200)
        # white picture, 8-bit unsigned integer (0 to 255)
        result_frame = np.zeros((int(height), int(width), 3), np.uint8)
        result_frame[:] = white
        unsafe = []
        safe = []

        if len(measured_distance) != 0:
                for pair in measured_distance:
                        if pair[0] <= 2: # the unsafe distance is 2m
                                if (pair[1] not in unsafe) and (pair[1] not in safe):
                                        unsafe += [pair[1]]
                                if (pair[2] not in unsafe) and (pair[2] not in safe):
                                        unsafe += [pair[2]]
                                result_frame = cv2.line(result_frame, (int(pair[1][0]), int(pair[1][1])), (int(pair[2][0]), int(pair[2][1])), red, 2)

        if len(perspective_points) != 0:
                for human in perspective_points:
                        if human not in unsafe:
                                safe += [human]

        for plot in safe:
                result_frame = cv2.circle(result_frame, (int(plot[0]), int(plot[1])), 5, green, 10)
        for plot in unsafe:
                result_frame = cv2.circle(result_frame, (int(plot[0]), int(plot[1])), 5, red, 10)

        return result_frame, len(unsafe)
```

Function: get_pixel_scale_of_2m(target_reference, perspective_matrix)
Input description:

- ***target_reference*:** the selected reference used for estimating the pixel distance of 2 meters in bird eye view.
  target_reference[0]: shoulder_to_height_ratio of the target
  target_reference[1]: x-coordinate of the target on its bottom left
  target_reference[2]: y-coordinate of the target on its bottom left
  target_reference[3]: width of the target
  target_reference[4]: height of the target
- ***perspective_matrix***: the output of get_perspective_transform(). This could transform the video in camera view into video in bird eye view.

Output description:

- ***return***: the number of pixels in bird eye view that represents the corresponding 2 meters of real scale.

This function is used to compute the pixel distance of 2 meter in the bird eye view based on the measurements of a selected person as reference and a computed perspective matrix used for bird eye view transformation.

```
def get_pixel_scale_of_2m(target_reference, perspective_matrix):
    ''' target_reference[0]: shoulder_to_height_ratio of the target
        target_reference[1]: x-coordinate of the target on its bottom left
        target_reference[2]: y-coordinate of the target on its bottom left
        target_reference[3]: width of the target
        target_reference[4]: height of the target
        output: safe distance of 2m's representation in bird view perspective.
    '''
    shoulder_to_height_ratio = target_reference[0]
    x = target_reference[1]
    y = target_reference[2]
    width = target_reference[3]
    height = target_reference[4]
    assumed_real_height = 1.70
    absolute_scale_shoulder = (assumed_real_height / height) * width
    points = np.float32(np.array([[[x, y],[x+width, y]]]))
    print(points)
    perspective_point1 = cv2.perspectiveTransform(points, perspective_matrix)[0][0]
    perspective_point2 = cv2.perspectiveTransform(points, perspective_matrix)[0][1]
    dis_x = abs(perspective_point2[0]-perspective_point1[0])
    dis_y = abs(perspective_point2[1]-perspective_point1[1])
    distance = int(np.sqrt(((dis_x)**2) + ((dis_y)**2)))
    pixel_distance_of_2m = int(round(distance / absolute_scale_shoulder * 2))
    return pixel_distance_of_2m
```

Function: measure_distance(safety_distance, perspective_points)

Input description:

- ***safety_distance***: the output of get_pixel_distance_of_2m
- ***perspective_points***: the output of *get_perspective_points()*

Output description:

- ***return***: list of lists, every element is [distance, the perspective point of human1, the perspective of human2]

This function is the helper function of bird_eye_view, it checks for pairwise distance of people and computes a list of lists that every element in it has 3 elements. The 1st element is distance, the 2nd and 3rd elements are the perspective points of two people.

```
def measure_distance(safety_distance, perspective_points):
        measured_distance = []
        for human1 in perspective_points:
                for human2 in perspective_points:
                        if human1 != human2:
                                dis_x = abs(human2[0]-human1[0])
                                dis_y = abs(human2[1]-human1[1])
                                distance = float((np.sqrt(((dis_x)**2) + ((dis_y)**2))/safety_distance) * 2)
                                measured_distance += [(distance, human1, human2)]
        return measured_distance
```

Function: social_distancing_view(frame, perspective_transform, boxes, safety_distance)

Input description:

- ***frame****:* the current image frame read by video.read()
- ***perspective_transform:*** output of get_perspective_transform(points, height, width), that gives a transformation to map points to bird-eye view, in order to measure actual distance between each pair of detected people.
- ***boxes:*** list of [x of left border, y of top border, width, height] in original video coordinate for each detected human, contains all information needed to draw a box around each person.
- ***safety_distance****:* the output of get_pixel_distance_of_2m

Output description:

- ***return***: the modified frame with colored boxes around each detected person drawn based on input information.

This function is a helper to visualize whether detected people in the original video are having safe distances or not. People who are in safe distance from others will be bound by a green box, while those who are too close to others will be bound by a red box.

```
def social_distancing_view(frame, perspective_transform, boxes, safety_distance):
        """
        boxes: list of [x of left border, y of top border, width, height] in original video
        """
        red = (0, 0, 255)
        green = (0, 255, 0)

        # draw green box for everyone
        for i in range(len(boxes)):
                x, y, w, h = boxes[i]
                frame = cv2.rectangle(frame, (x, y), (x + w, y + h), green, 2)

        for i in range(len(boxes)):
                for j in range(len(boxes)):
                        if j != i:
                                x1, y1, w1, h1 = boxes[i]
                                x2, y2, w2, h2 = boxes[j]
                                points1 = np.array([[[int(x1 + (w1 * 0.5)), int(y1 + h1)]]], dtype="float32")
                                points2 = np.array([[[int(x2 + (w2 * 0.5)), int(y2 + h2)]]], dtype="float32")
                                perspective_point1 = cv2.perspectiveTransform(points1, perspective_transform)[0][0]
                                perspective_point2 = cv2.perspectiveTransform(points2, perspective_transform)[0][0]
                                perspective_points1 = (int(perspective_point1[0]), int(perspective_point1[1]))
                                perspective_points2 = (int(perspective_point2[0]), int(perspective_point2[1]))
                                if len(measure_distance(safety_distance, [perspective_points1, perspective_points2])) != 0:
                                        distance = measure_distance(safety_distance, [perspective_points1, perspective_points2])[0][0]

                                if distance <= 2:     # unsafe
                                        # box for human 1
                                        frame = cv2.rectangle(frame, (x1, y1), (x1 + w1, y1 + h1), red, 2)
                                        # box for human 2
                                        frame = cv2.rectangle(frame, (x2, y2), (x2 + w2, y2 + h2), red, 2)
                                        # line between human 1 and human 2
                                        frame = cv2.line(frame, (int(x1 + w1 / 2), int(y1 + h1 / 2)), (int(x2 + w2 / 2), int(y2 + h2 / 2)), red, 2)


        return frame
```

Function: get_social_distance_detected_video(video_path)

Input description:

- ***video_path*:** a string that gives the path for the input video
- ***picked_pts*:** a list of lists that represents the 4 corner points of the region of interest

Output description:

- ***return***: None

This function is like the main function of the project that takes a video (path) as input. It integrates and calls all other helper functions to produce a bunch of desired frames (for creating videos to visualize detection results for both bird-eye view and camera view).

For bird-eye view: colored points represent each person; if social distance violation is detected, colored the points in red, otherwise colored the points in green.

For camera view: colored boxes around each detected person; if social distance violation is detected, colored the points in red, otherwise colored the points in green.

```
# Acknowledge to Nayak, S. (2018). Deep Learning based Object Detection using YOLOv3 with OpenCV ( Python / C++).
# Retrieved 18 December 2020, from https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/
# Learned the usage and the implementation YOLOv3 on object detection from this website.
def get_social_distance_detected_video(picked_pts, video_path):
        model = '/content/drive/MyDrive/CSC420/project/yolov3.weights'
        config = '/content/drive/MyDrive/CSC420/project/yolov3.cfg'
        video = cv2.VideoCapture(video_path)
        net_layer = cv2.dnn.readNetFromDarknet(config, model)
        layersNames = net_layer.getLayerNames()
        outputlayers_Names = [layersNames[i[0] - 1] for i in net_layer.getUnconnectedOutLayers()]
        net = cv2.dnn.readNetFromDarknet(config, model)
        hasFrame = True
        frame_num = 0
        shoulder_to_height_ratio = []
        while (hasFrame):
                (hasFrame, frame) = video.read()
                if not hasFrame:
                        break
                height = []
                frames = []
                blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True, crop=False)
                net.setInput(blob)
                predictions_array = net.forward(outputlayers_Names)
                boxes = []
                framesID = []
                frame_height, frame_width = frame.shape[0],frame.shape[1]
                for output in predictions_array:
                        for detection in output:
                                scores = detection[5:]
                                classID = np.argmax(scores)
                                confidence = scores[classID]
                                if classID == 0 and confidence > 0.8:
                                        box = detection[0:4] * np.array([frame_width, frame_height, frame_width, frame_height])
                                        (centerX, centerY, width, height) = box.astype("int")
                                        x = int(round((centerX - (width / 2))))
                                        y = int(round((centerY + (height / 2))))
                                        shoulder_to_height_ratio.append((width/height, x, y, width, height))
                frame_num += 1
```

```
        human_count = len(shoulder_to_height_ratio)
        shoulder_to_height_ratio.sort()
        target_reference = shoulder_to_height_ratio[human_count//2]
        # target_reference = (0.45652173913043476, 1106, 142, 63, 138)
        # picked_pts = [[1800, 100], [1800, 900], [100, 900], [100, 100]]
        # transform_matrix = get_perspective_transform(picked_pts, 1080, 1920)
        transform_matrix = get_perspective_transform(picked_pts, frame_height, frame_width)
        safety_distance = get_pixel_scale_of_2m(target_reference, transform_matrix)
        print(safety_distance)

        video = cv2.VideoCapture(video_path)
        net_layer = cv2.dnn.readNetFromDarknet(config, model)
        layersNames = net_layer.getLayerNames()
        outputlayers_Names = [layersNames[i[0] - 1] for i in net_layer.getUnconnectedOutLayers()]
        net = cv2.dnn.readNetFromDarknet(config, model)
        hasFrame = True
        frame_num = 0
        while (hasFrame):
                (hasFrame, frame) = video.read()
                if not hasFrame:
                        break
                height = []
                frames = []
                blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True, crop=False)
                net.setInput(blob)
                predictions_array = net.forward(outputlayers_Names)
                boxes = []
                framesID = []
                confidences = []
                frame_height, frame_width = frame.shape[0],frame.shape[1]
                for output in predictions_array:
                        for detection in output:
                                scores = detection[5:]
                                classID = np.argmax(scores)
                                confidence = scores[classID]
```

```
                        if classID == 0 and confidence > 0.8:
                                box = detection[0:4] * np.array([frame_width, frame_height, frame_width, frame_height])
                                (centerX, centerY, width, height) = box.astype("int")
                                x = int(round((centerX - (width / 2))))
                                y = int(round((centerY - (height / 2))))
                                boxes += [[x, y, int(width), int(height)]]
                                confidences += [float(confidence)]
                frame_num += 1
                indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.5)
                boxes1 = []
                for i in indices:
                        i = i[0]
                        box = boxes[i]
                        boxes1 += [box]


                perspective_points = get_perspective_points(boxes, transform_matrix)
                measured_distance = measure_distance(safety_distance, perspective_points)
                bird_eye_view_image, num_unsafe = bird_eye_view(frame_height, frame_width, measured_distance, perspective_points)
                for point in picked_pts:
                        frame = cv2.circle(frame, (int(point[0]), int(point[1])), 5, (200, 200, 200), 10)
                frame = cv2.line(frame, (int(picked_pts[0][0]), int(picked_pts[0][1])), (int(picked_pts[1][0]), int(picked_pts[1][1])), (200, 200, 200), 2)
                frame = cv2.line(frame, (int(picked_pts[1][0]), int(picked_pts[1][1])), (int(picked_pts[2][0]), int(picked_pts[2][1])), (200, 200, 200), 2)
                frame = cv2.line(frame, (int(picked_pts[2][0]), int(picked_pts[2][1])), (int(picked_pts[3][0]), int(picked_pts[3][1])), (200, 200, 200), 2)
                frame = cv2.line(frame, (int(picked_pts[3][0]), int(picked_pts[3][1])), (int(picked_pts[0][0]), int(picked_pts[0][1])), (200, 200, 200), 2)
                social_distance_view_image = social_distancing_view(frame, transform_matrix, boxes1, safety_distance)
                if not cv2.imwrite('/content/drive/MyDrive/CSC420/project/bird_eye_view/frame%d.jpg' % frame_num, bird_eye_view_image):
                        print("Could not write image")
                if not cv2.imwrite('/content/drive/MyDrive/CSC420/project/social_distance_view/frame%d.jpg' % frame_num, social_distance_view_image):
                        print("Could not write image")
                print('%d frame done' % frame_num)
```

Function: create_video(path)

Input description:

- ***path***: a string that gives the path for the input frames and the output video

Output description:

- ***return***: None

This function is used for creating a video that 25 frames per second for the frames of bird eye view and the frames of social distancing view. It will save the output video as out.avi in the folder of the input path.

```
def create_video(path):
        img_list = []
        count = 0
        for filename in glob.glob(path + '/*.jpg'):
                count += 1
        for i in range(count):
                index = i+1
                img = cv2.imread(path + '/frame' + str(index) +'.jpg')
                img_list += [img]

        height,width,layers = img_list[0].shape
        out = cv2.VideoWriter(path + '/out.avi',cv2.VideoWriter_fourcc(*'DIVX'), 25, (width, height))
        for img in img_list:
                out.write(img)
        out.release()
```

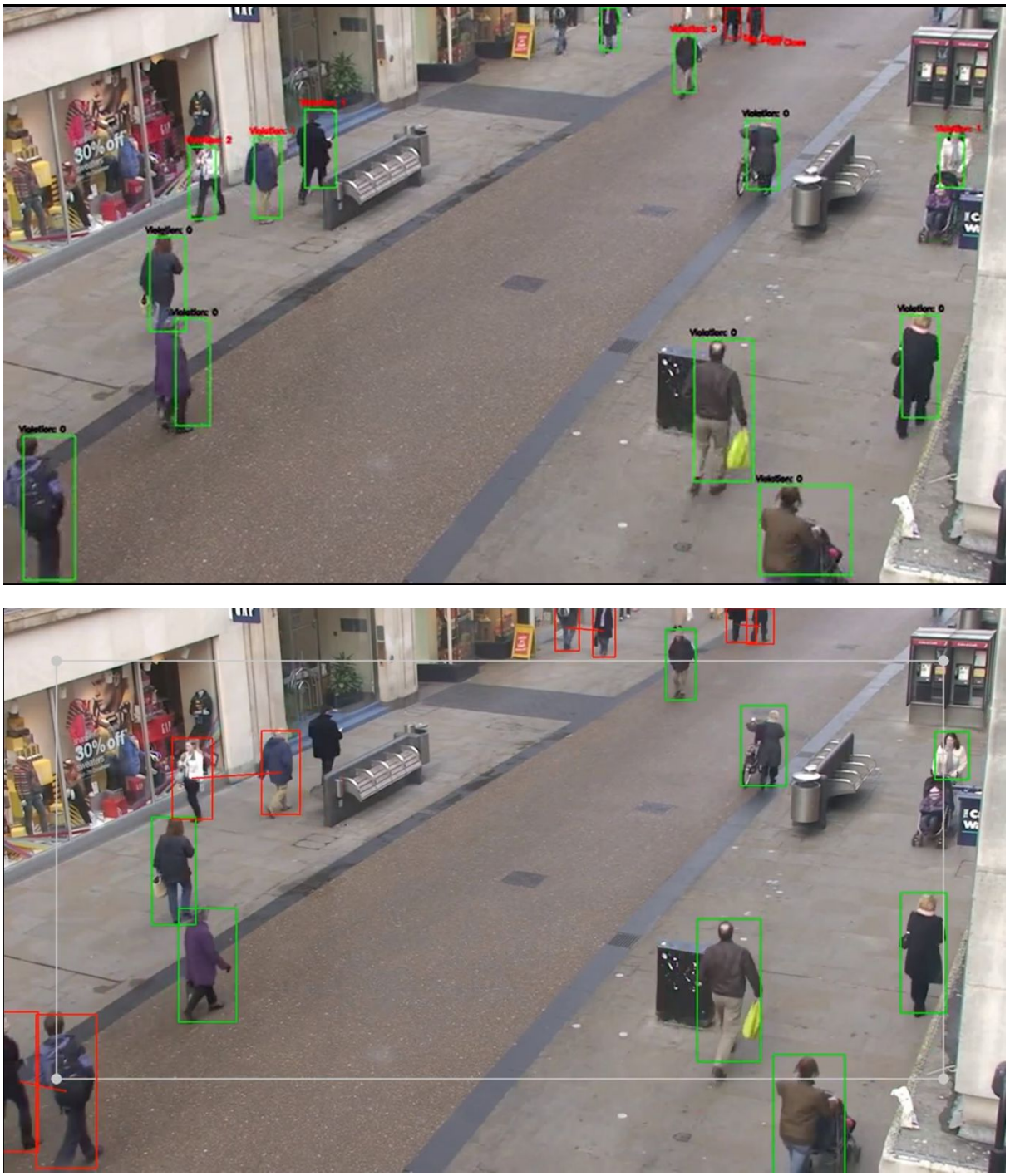
```
create_video('/content/drive/MyDrive/CSC420/project/social_distance_view')
```

```
create_video('/content/drive/MyDrive/CSC420/project/bird_eye_view')
```

- Presentation of Results

In order to test for the accuracy of our detector, our team chooses to use “Test video for Object Detection || TRIDE” on Youtube as an input. **(source: TRIDE) [8]** Our team compares the output video produced by our detector with the output video produced by the social distance detector developed by Landing AI company which is established by the reputable scientist Andrew Ng. We found that our video output gives similar results as theirs.

Three sets of frames of the two output videos will exhibit below for comparison.



(1st set of output videos:

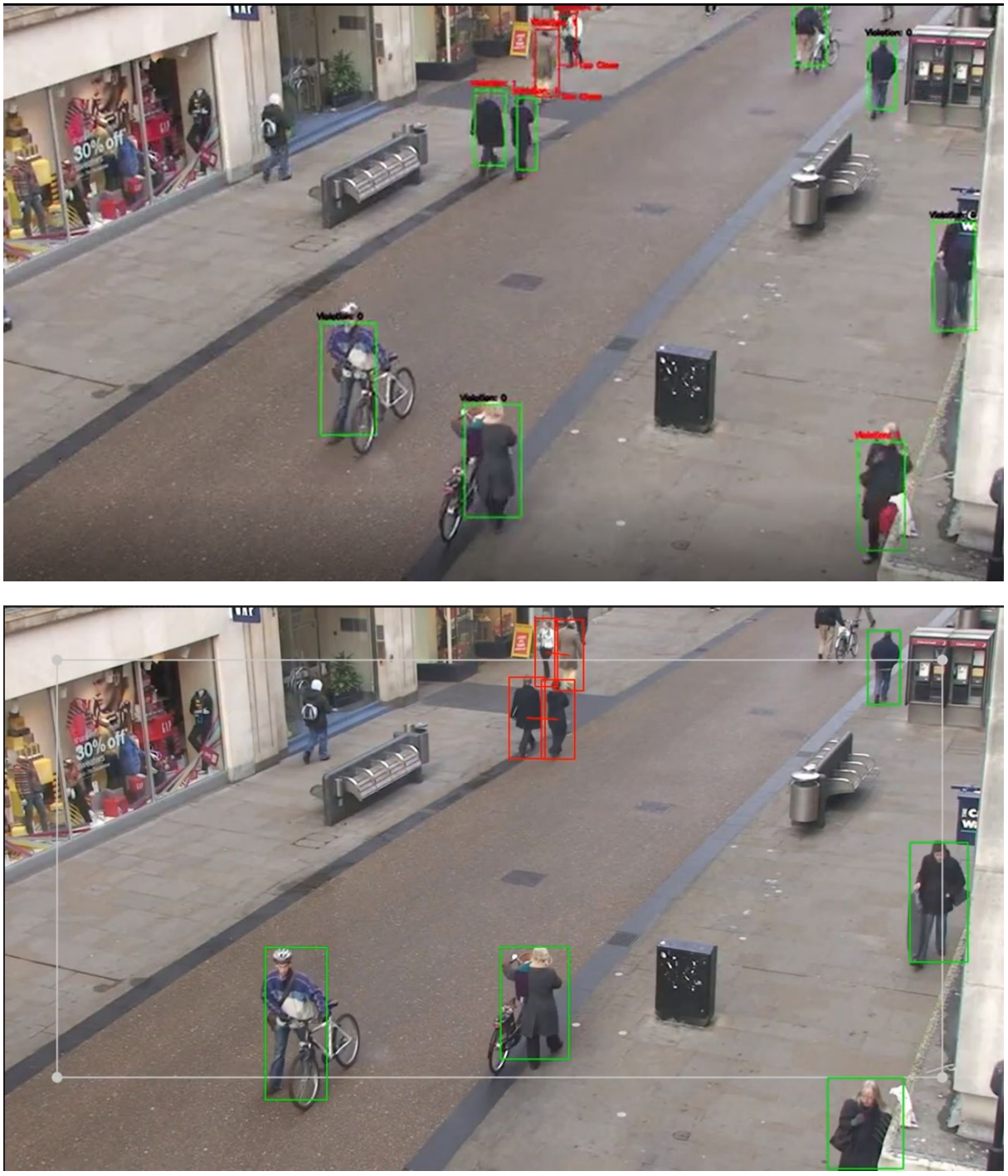
Landing AI company's detector's output on the top

Our detector's output on the bottom)

(2nd set of output videos:

Landing AI company’s detector’s output on the top

Our detector’s output on the bottom)



(3rd set of output videos:
Landing AI company's detector's output on the top
Our detector's output on the bottom)

Since, the results are considerably similar which could mean that our detector could achieve a high level of accuracy. Observing by eyes, our result seems even

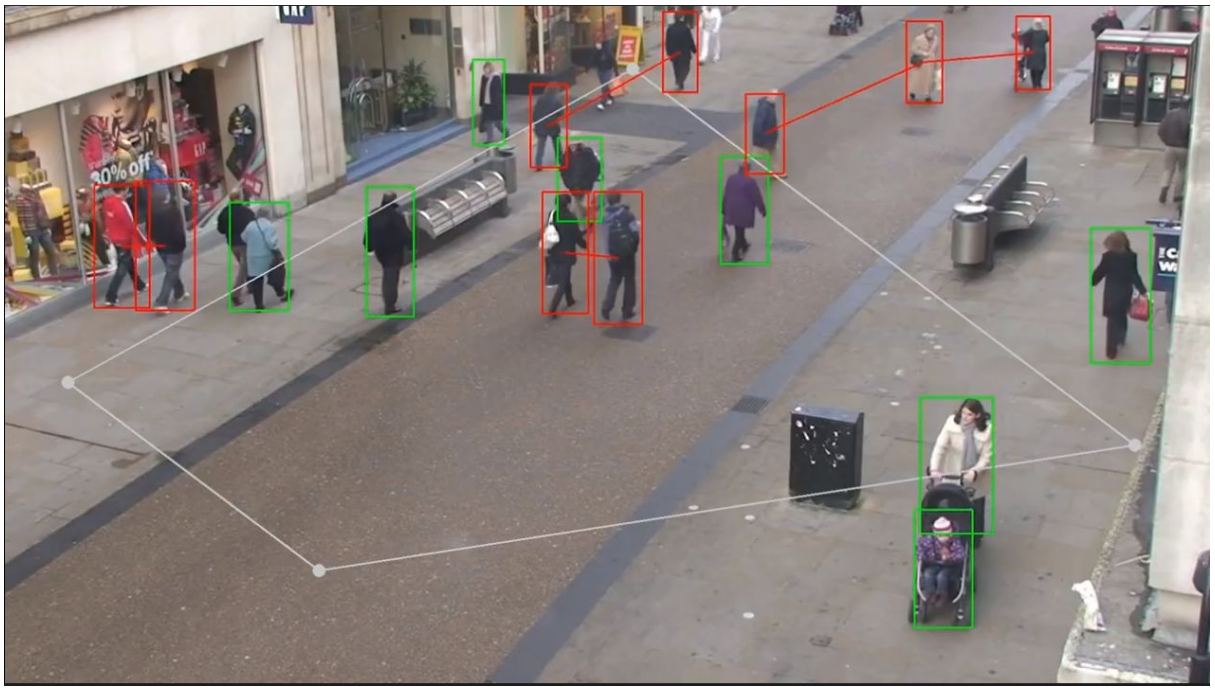
more accurate than the output video generated by the detector developed by Landing AI company.

Experiment on reducing the number of overlapping bounding box

The highest score in a box is called confidence. If the confidence level of a frame is less than a given threshold, the bounding box is discarded without considering further processing. Boxes with confidence levels equal to or greater than the confidence threshold are subject to non maximum suppression. This will reduce the number of overlapping boxes. The nmsThreshold parameter of the built-in function cv2.dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThreshold) controls the non maximum suppression. If nmsThreshold is set too low, such as 0.1, we may not be able to detect overlapping objects of the same or different classes. But if it's set too high, for example, 1, we'll get multiple boxes for the same object. So after trials and errors, we eventually used an intermediate value of 0.5 in our code.

Limitation of the detector

The area surrounded by white dots and white lines is the area that will be projected onto the bird eye view. From the following two figures, we can see that the accuracy of detection outside this area is much lower than that of detection within this area. This is because when we measure and compare distances, we first project each person's bottom center point into bird eye view. Points not in this region may become negative coordinates or still be positive coordinates after perspective transformation. When the negative coordinates and positive coordinates are calculated, it may occur that two people who originally have a safe distance are considered to have unsafe distance after calculation. To avoid this problem, we strongly recommend that users partition the white area as large as possible and include all characters as much as possible.



(Figure 11: Output frames of our social distancing detector)

Discussion of the method of estimation of unit pixel distance

According to the experiments and tests we have done that mentioned above, our detector could detect social distance violations accurately. This implies that our method of estimating the unit pixel distance under the bird eye view could deliver a relatively accurate estimation of unit pixel distance that is closed to the true unit pixel distance. Although our algorithm for the estimation of the pixel distance still involves a certain degree of uncertainty in reference selection, it's an alternative implementation that makes our detector more flexible to be used in different locations asking users to go to real fields to obtain the real scale and measurement of the field. In conclusion, our social distance detector has compromised some accuracy with much more practicality, flexibility and ease of use. If we can further find ways in improving the quality of reference selection, this could make our detector more practical and give it the potential to be wide-adapted in the world.

4. Conclusion

For now, social distance and other basic health measures are very important to keep the spread of covid-19 as slow as possible. Our work proposes a simple framework based on deep learning, which can automatically monitor social distance through object detection and tracking methods, and identify each individual in real time with the help of bounding boxes. The generated bounding box is helpful to identify the groups that satisfy the closeness attribute calculated by the pairwise vectorization method.

When doing literature research, we compared the popular centralized models: RCNN, SSD and YOLOV3, in which YOLOV3 shows the effective performance of balancing FPS and map scores. Since this method is highly sensitive to the spatial position of the camera, we can fine tune the same method to better adjust according to the corresponding field of view, and improve the accuracy of our work.

After confirming the model used, our work uses YOLOV3 to detect humans in the input video, finds the most appropriate shoulder width-to-height ratio, and then finds the most appropriate pixel distance of 2m. Meanwhile, in order to make our detector more accurate, our algorithm reduces the overlapping bounding box through trials and errors on the selection of nmsThreshold. We eventually choose to use a value of 0.5 for the nmsThreshold in our code.

To illustrate the absolute distance and relative position between people more clearly, we choose to convert the specified area into a bird eye view. Our algorithm finds the transformation matrix by using OpenCV's built-in function, cv2. getperspectivetransform(), and converts each person to the corresponding point on the bird eye view.

Our team aims at implementing a social distance detector that could accurately identify social distance violations. In order to test for the accuracy of our detector, our team chooses to use “Test video for Object Detection || TRIDE” on Youtube as an input. Our team compares the output video produced by our detector with the output video produced by the social distance detector developed by Landing AI company which is established by the reputable scientist Andrew Ng. We found that our video output gives similar results as theirs.

According to the test result, our detector could detect social distance violations accurately. This implies that our method of estimating the unit pixel

distance under the bird eye view could deliver a relatively accurate estimation of unit pixel distance that is closed to the true unit pixel distance. Although our algorithm for the estimation of the pixel distance still involves a certain degree of uncertainty in reference selection, it's an alternative implementation that makes our detector more flexible to be used in different locations without asking users to go to real fields to obtain the real scale and measurement of the field. In conclusion, our social distance detector has compromised some accuracy with much more practicality, flexibility and ease of use.

However, there is a significant and obvious drawback in implementing the detector. It is that our object detection algorithm runs twice over each frame in the video. The first round of looping over the frames is due to the approximation of unit pixel distance. As we want to extract the most appropriate human as reference, our algorithm firstly goes over all the frames to collect the measurements of the respective boxes. After the first round of loop, we could find the most appropriate reference (the one with median width-to-height ratio) for unit pixel distance deduction. Then, we run the second round of looping over the frames in order to check for social distance violations. However, these two loops can be combined in a way that we don't need to detect each person twice which is extremely time-consuming. This is one major improvement we could make in the future.

Additionally, If we can further find ways in improving the quality of reference selection, this could make our detector more practical and give it the potential to be wide-adapted in the world.

5. Authors' contributions

Yiyang Hua: I write the codes that relate to the computation of bird eye view and creating video and reducing overlapping bounding boxes. And I write the theoretical part, limitation of our codes and empirical work of my codes in the report.
get_perspective_transform(points, height, width),
get_perspective_points(human_boxes, prespective_transform),
bird_eye_view(height, width, measured_distance, perspective_points),

measure_distance(safety_distance, perspective_points),
create_video(path).

Wenjie Hao: I wrote the codes that relate to the finding the proper human target as reference, estimation of pixel distance and human detection. And I write abstract, introduction, literature review, empirical work of my codes in the report and bibliography. I came up with the idea of approximating pixel distance from human height and I successfully implemented and tested the idea. Additionally, I helped organize the report.
get_pixel_scale_of_2m(target_reference, perspective_matrix)
get_social_distance_detected_video(video_path)

Chunjing Zhang: wrote code for visualizing boxes on original video based on distances calculated for each detected person. Also write empirical work for this function. Helped with empirical work for get_social_distance_detected_video (video_path). Wrote conclusion part and created table of contents for this report.
social_distancing_view(frame, perspective_transform, boxes, safety_distance)

6. Bibliography

1. Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement* [Ebook] (p. 4). University of Washington. Retrieved from https://pjreddie.com/media/files/papers/YOLOv3.pdf

2. Singh Punn, N., Kumar Sonbhadra, S., & Agarwal, S. (2020). *Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deepsort techniques* [Ebook] (2nd ed., pp. 8-9). arXivLabs. Retrieved from https://arxiv.org/pdf/2005.01385.pdf

3. Nayak, S. (2018). Deep Learning based Object Detection using YOLOv3 with OpenCV (Python / C++). Retrieved 18 December 2020, from

https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/

4. OpenCV 2.4.13.7 documentation, "*Documentation of cv.PerspectiveTransform()".* Retrieved 18 December 2020, from https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html?highlight=perspectivetransform#cv.PerspectiveTransform

5. MADALI, N. (2020). Inverse Perspective Transformation. Retrieved 18 December 2020, from https://medium.com/ai-in-plain-english/inverse-perspective-transformation-b62b5eedb44a

6. 【论文理解】yolov3损失函数_DLUT_yan的博客-CSDN博客_yolov3损失函数. (2020). Retrieved 18 December 2020, from https://blog.csdn.net/weixin_43384257/article/details/100986249

7. Kathuria, A. (2018). What's new in YOLO v3?. Retrieved 18 December 2020, from https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

8. Test video for Object Detection || TRIDE. (2020). Retrieved 18 December 2020, from https://www.youtube.com/watch?v=pk96gqasGBQ