Social Distancing project using Computer Vision and Deep Learning

Team Name: ExcelInnovator

Author(s): Anirudh V

Date of Submission: 14.07.2023

ABSTRACT

The outbreak of the COVID-19 pandemic has highlighted the critical importance of social distancing in preventing the spread of infectious diseases. To ensure compliance with social distancing guidelines, this project proposes a novel approach that leverages computer vision and deep learning techniques to monitor and enforce social distancing in public spaces.

The project aims to develop an intelligent system capable of automatically detecting and analyzing human interactions in real-time using surveillance cameras. By employing state-of-the-art deep learning algorithms, the system will be able to accurately estimate the distance between individuals and identify potential violations of social distancing rules.

The proposed system will consist of several key components. First, a robust object detection model will be trained to identify and track individuals in the camera feed. Next, a distance estimation module will be developed to measure the distances between detected individuals, taking into account factors such as camera calibration and perspective distortion. The system will then analyze the detected distances and compare them against predefined social distancing guidelines to determine if any violations have occurred.

Upon detecting a violation, the system will generate alerts in real-time, notifying relevant personnel or authorities. These alerts can be delivered through various channels, such as a user interface, email, or mobile notifications. Additionally, the system will generate statistical reports and visualizations to provide insights into compliance levels and identify hotspots where social distancing violations are frequently observed.

To evaluate the effectiveness of the proposed system, extensive experiments will be conducted in diverse real-world scenarios. The system's accuracy in detecting violations and its ability to handle different camera angles, lighting conditions, and crowd densities will be assessed. Fine-tuning and optimization techniques will be employed to enhance the system's performance and make it adaptable to various environments.

In conclusion, this project seeks to leverage computer vision and deep learning techniques to develop an intelligent system for monitoring and enforcing social distancing guidelines. The proposed system has the potential to assist public health officials, businesses, and organizations in ensuring compliance with social distancing measures and ultimately contribute to mitigating the spread of infectious diseases in public spaces.

INTRODUCTION

The COVID-19 pandemic has significantly impacted societies worldwide, emphasizing the critical role of social distancing in curbing the spread of infectious diseases. Adhering to social distancing guidelines has become a priority for public health officials, businesses, and organizations. However, monitoring and enforcing social distancing in public spaces can be a challenging task. Traditional methods rely on manual observation or physical barriers, which are often insufficient and impractical. To address this challenge, emerging technologies such as computer vision and deep learning offer promising solutions. By leveraging the power of artificial intelligence, these technologies can enable automated and accurate monitoring of social distancing compliance. This project aims to explore the application of computer vision and deep learning techniques to develop an intelligent system that can detect, analyze, and enforce social distancing measures in real-time, thereby assisting in preventing the spread of infectious diseases.

MOTIVATION BEHIND THE PROBLEM

The motivation behind tackling the problem of monitoring social distancing using computer vision and deep learning stems from several key factors.

Firstly, social distancing has proven to be a crucial measure in preventing the spread of infectious diseases, particularly in the context of the COVID-19 pandemic. By maintaining a safe physical distance from others, individuals can significantly reduce the risk of transmission. However, ensuring compliance with social distancing guidelines in public spaces can be challenging, as it requires constant monitoring of large crowds and the ability to identify violations promptly.

Secondly, manual monitoring of social distancing is labor-intensive, time-consuming, and prone to human error. Assigning personnel to continuously observe and enforce social distancing guidelines is neither efficient nor scalable, particularly in crowded environments. Therefore, there is a need for automated systems that can augment human efforts, providing accurate and real-time monitoring of social distancing compliance.

Thirdly, the advances in computer vision and deep learning have opened up new possibilities for analyzing visual data. These technologies have demonstrated remarkable capabilities in object detection, tracking, and distance estimation. By leveraging these techniques, it becomes feasible to develop intelligent systems that can analyze surveillance camera feeds and accurately detect violations of social distancing guidelines.

Moreover, by employing computer vision and deep learning algorithms, it becomes possible to handle complex scenarios, such as crowded spaces, varying camera angles, and challenging lighting conditions. These technologies can adapt to different environments and provide insights into compliance levels, allowing authorities and organizations to identify hotspots where violations frequently occur and take appropriate actions.

The motivation behind this project lies in harnessing the power of computer vision and deep learning to automate the monitoring and enforcement of social distancing guidelines. By doing

so, we can enhance public safety, assist in the prevention of infectious diseases, and contribute to the overall well-being of communities.

BACKGROUND

Several research studies and projects have explored the use of computer vision and deep learning for social distancing:

Social Distancing Monitoring: Computer vision techniques have been employed to monitor public spaces and assess social distancing compliance. This typically involves analyzing video feeds from surveillance cameras to detect individuals and measure the distances between them. By comparing the detected distances with predefined thresholds, violations can be identified and appropriate actions can be taken.

Crowd Density Estimation: Deep learning models have been trained to estimate crowd density in real-time. By analyzing video streams, these models can provide insights into crowd sizes and densities, allowing authorities to monitor high-risk areas and take necessary precautions.

People Tracking and Behavior Analysis: Computer vision algorithms can track individuals within a scene and analyze their behavior. By understanding how people move and interact, it becomes possible to identify situations where social distancing guidelines are being violated. Such information can be used for real-time interventions or to optimize space layouts to encourage social distancing.

Alert Systems and Warnings: Deep learning models can be used to develop alert systems that notify individuals or authorities when social distancing violations occur. These systems can leverage computer vision techniques to detect close interactions between people and issue warnings or reminders to maintain distance.

APPROACH

In order to implement the project YOLO algorithm has been used.

The YOLO (You Only Look Once) algorithm is a popular object detection algorithm that can be used in a social distancing project involving computer vision and deep learning. YOLO has gained significant attention due to its ability to provide real-time object detection in images and videos. Here's how the YOLO algorithm is be applied in the social distancing project:

Object Detection:

The first step is to use the YOLO algorithm to detect individuals in a given scene. YOLO divides the input image or video into a grid and predicts bounding boxes and class probabilities for objects within each grid cell. By training the YOLO model on a dataset that includes labeled individuals, it can learn to accurately detect people in the scene.

Distance Estimation:

Once the individuals are detected, the YOLO algorithm can be used to estimate the distance between them. This can be achieved by measuring the size of the bounding boxes corresponding to each person in the image. By calibrating the system with known distances, such as using reference objects of known size, it is possible to estimate the real-world distances between individuals.

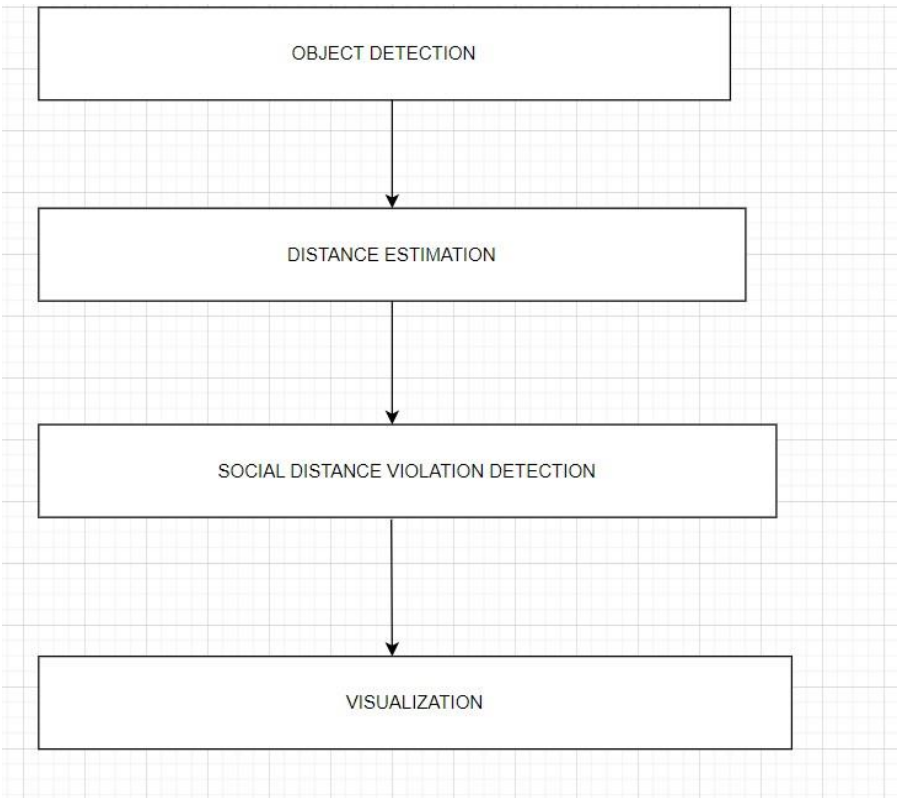
Social Distancing Violation Detection:

Using the distance estimates obtained from the YOLO algorithm, the system can determine whether social distancing guidelines are being followed or violated. By comparing the measured distances with predefined thresholds (e.g., 1 or 2 meters), violations can be identified. If the distance between two detected individuals falls below the threshold, the system can flag it as a potential violation.

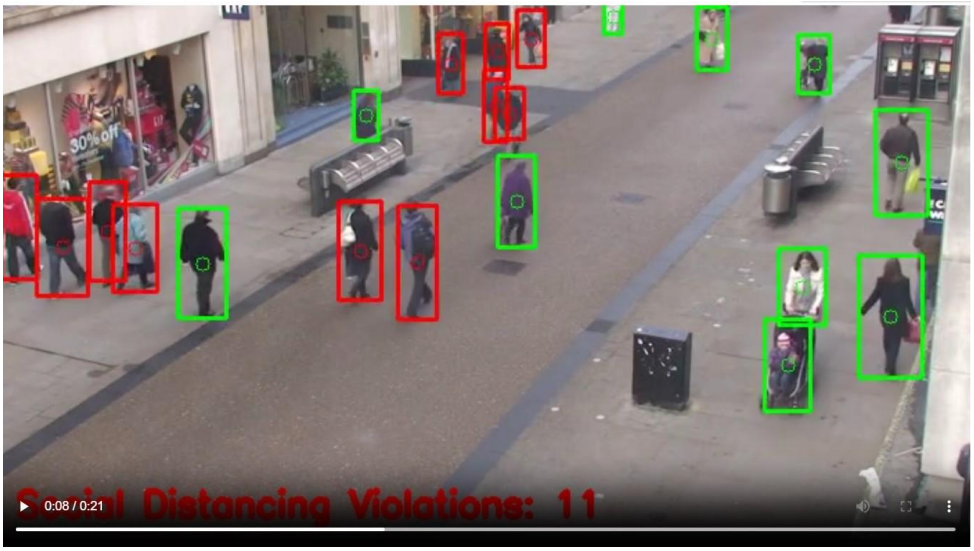
Visualizations:

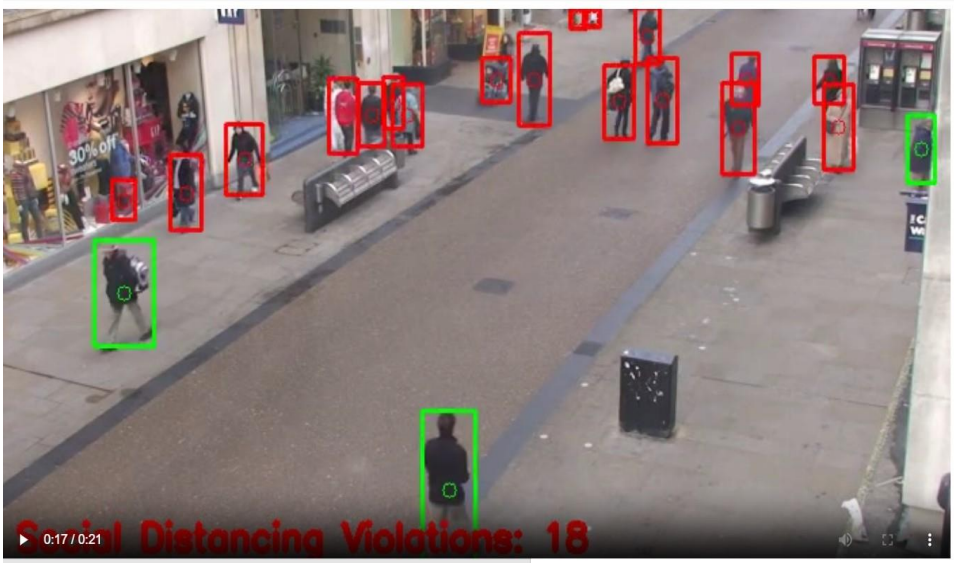
The YOLO-based social distancing project can include visualizations to indicate instances of social distancing violations. This can involve highlighting the individuals or bounding boxes that violate the social distancing rules, generating warnings or alarms, or displaying real-time visual feedback to users or authorities

Pipeline of project



RESULTS





In the above given screenshots we can see that red rectangular frame represents social distance violation and green rectangular frame represents social distance being followed.

A person violating minimum distance will be marked by red frame and those who are following will marked by green frame.

REFERENCES

Y. C. Hou, M. Z. Baharuddin, S. Yussof and S. Dzulkifly, "Social Distancing Detection with Deep Learning Model," 2020

Ahmed I, Ahmad M, Rodrigues JJPC, Jeon G, Din S.'' A deep learning-based social distance monitoring framework for COVID-19", 2020

S. Madane and D. Chitre, "Social Distancing Detection and Analysis through Computer Vision," 2021

Link to Solution:
https://colab.research.google.com/drive/13R7YlB4LOKcBRA5ff4BAi2IeJ6doVciv

Github link:

https://github.com/ANIRUDH-V-SIT2020/Social_distancing

Model link:

https://github.com/ultralytics/yolov3

CODE

```
from google.colab import drive
drive.mount('/content/drive')

"""# New Section"""

# initialize minimum probability to filter weak detections along with
# the threshold when applying non-maxima suppression
MIN_CONF = 0.3
NMS_THRESH = 0.3

# define the minimum safe distance (in pixels) that two people can be
# from each other
MIN_DISTANCE = 50

# import the necessary packages

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
                # 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)



# 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

# USAGE
# python social_distance_detector.py --input pedestrians.mp4
# python social_distance_detector.py --input pedestrians.mp4 --output output.avi

# import the necessary packages
from google.colab.patches import cv2_imshow
from scipy.spatial import distance as dist
import numpy as np
import argparse
import imutils
import cv2
import os

# construct the argument parse and parse the 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(["--input","/content/drive/MyDrive/distance/pedestrians.mp4","--output","my_output.avi","--display","1"]))

# load the COCO class labels our YOLO model was trained on
```

```
labelsPath = os.path.sep.join(["/content/drive/MyDrive/distance/yolo-coco/coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join(["/content/drive/MyDrive/distance/yolov3.weights"])
configPath = os.path.sep.join(["/content/drive/MyDrive/distance/yolo-coco/yolov3.cfg"])

# load our YOLO object detector trained on COCO dataset (80 classes)
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

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

# initialize the video stream and pointer to output video file
print("[INFO] accessing video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)
writer = None

# loop over the frames from the video stream
while True:
	# read the next frame from the file
	(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 minimum social
# distance
violate = 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] < MIN_DISTANCE:
				# update our violation set with the indexes of
				# the centroid pairs
				violate.add(i)
				violate.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 set, then
        # update the color
        if i in violate:
                color = (0, 0, 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, 1)

# draw the total number of social distancing violations on the
# output frame
text = "Social Distancing Violations: {}".format(len(violate))
cv2.putText(frame, text, (10, frame.shape[0] - 25),
        cv2.FONT_HERSHEY_SIMPLEX, 0.85, (0, 0, 255), 3)

# check to see if the output frame should be displayed to our
# screen
# if args["display"] > 0:
#         # show the output frame
#         #cv2_imshow(frame)
#         key = cv2.waitKey(1) & 0xFF
```

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

        # 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)
```

```
!pip install moviepy
```

```
!pip install ffmpeg-python
```

```
import ffmpeg

def convert_avi_to_mp4(input_path, output_path):
    try:
        ffmpeg.input(input_path).output(output_path).run()
        print("File converted successfully!")
    except ffmpeg.Error as e:
        print(f"Error converting file: {e.stderr}")
```

```
# Specify the paths for the input AVI file and the output MP4 file
avi_file_path = '/content/my_output.avi'
mp4_file_path = '/content/output_file.mp4'

# Convert the AVI file to MP4
convert_avi_to_mp4(avi_file_path, mp4_file_path)

from IPython.display import Video

def play_mp4_file(file_path):
    # Display the video
    return Video(file_path, embed=True,width = 1080)

# Specify the path to your MP4 file
mp4_file_path = '/content/output_file.mp4'

# Call the function to play the MP4 file
play_mp4_file(mp4_file_path)

"""UPGRADED DISPLAYING THE RESULTS IN REAL TIME WEBSITE  USING FLASK AND NGROK"""

!pip install flask_ngrok

!curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc | sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null && echo "deb https://ngrok-agent.s3.amazonaws.com buster main" | sudo tee /etc/apt/sources.list.d/ngrok.list && sudo apt update && sudo apt install ngrok

from flask_ngrok import run_with_ngrok
from flask import Flask
```

```
!pip install flask

port_no = 5000

from flask import Flask
from pyngrok import ngrok

!pip install pyngrok

import cv2
from flask import Flask, render_template, Response,render_template_string
import threading
from base64 import b64encode
from google.colab.patches import cv2_imshow
from scipy.spatial import distance as dist
import numpy as np
import argparse
import imutils
import os

# Initialize Flask application
app = Flask(__name__)
ngrok.set_auth_token("2RulHmqco9uyS22yVfHzlgao4cc_3nLW36fxJJjj3tbrGt6Vf")
public_url =  ngrok.connect(port_no).public_url

# Construct the argument parse and parse the 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(["--input", "/content/drive/MyDrive/distance/pedestrians.mp4",
                           "--output", "my_output.avi", "--display", "1"]))

# Load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join(["/content/drive/MyDrive/distance/yolo-coco/coco.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# Derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join(["/content/drive/MyDrive/distance/yolov3.weights"])
configPath = os.path.sep.join(["/content/drive/MyDrive/distance/yolo-coco/yolov3.cfg"])

# Load our YOLO object detector trained on the COCO dataset (80 classes)
print("[INFO] Loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

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

# Initialize the video stream and pointer to the output video file
print("[INFO] Accessing video stream...")
vs = cv2.VideoCapture(args["input"] if args["input"] else 0)
writer = None

# Initialize an empty list to store the processed frames
processed_frames = []
```

```
# Function to detect people and calculate social distancing violations
def detect_people(frame, net, ln, personIdx=0):
    (H, W) = frame.shape[:2]
    results = []

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

    # Initialize the lists of detected bounding boxes, centroids, and confidence
    boxes = []
    centroids = []
    confidences = []

    # Loop over the layer outputs
    for output in layerOutputs:
        # Loop over each detection
        for detection in output:
            # Extract the class ID and confidence
            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            # Filter detections for people class and minimum confidence
            if classID == personIdx and confidence > 0.5:
                # Scale the bounding box coordinates
                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")
```

```
            # Calculate the top-left corner of the bounding box
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))

            # Update the lists
            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, 0.5, 0.3)

    # Ensure at least one detection exists
    if len(idxs) > 0:
        # Loop over the indexes
        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])

            # Add the detection result to the list of results
            results.append((confidences[i], (x, y, x + w, y + h), centroids[i]))

    # Return the list of results
    return results

# Function to process the frames
def process_frames():
    global processed_frames, vs, writer
```

```
# Loop over the frames from the video stream
while True:
    # Read the next frame from the file
    (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 minimum social distance
    violate = set()

    # Ensure there are at least two people detections (required to compute 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 if the distance between any two centroid pairs is less than the configured number of pixels
                if D[i, j] < 50:
                    # Update the violation set with the indexes of the centroid pairs
```

```
                violate.add(i)
                violate.add(j)

# Loop over the results
for (i, (prob, bbox, centroid)) in enumerate(results):
    # Extract the bounding box and centroid coordinates
    (startX, startY, endX, endY) = bbox
    (cX, cY) = centroid
    color = (0, 255, 0)

    # If the index pair exists within the violation set, then update the color
    if i in violate:
        color = (0, 0, 255)

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

# Draw the total number of social distancing violations on the output frame
text = "Social Distancing Violations: {}".format(len(violate))
cv2.putText(frame, text, (10, frame.shape[0] - 25),
            cv2.FONT_HERSHEY_SIMPLEX, 0.85, (0, 0, 255), 3)

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

# Append the processed frame to the list
processed_frames.append(frame)
```

```
    # Release the video writer and video stream
    if writer is not None:
        writer.release()
    vs.release()


# Route for the home page
@app.route('/')
def index():
    html = '''
    <!DOCTYPE html>
    <html>
      <head>
        <title>Social Distancing Detection</title>
        <style>
          body {
            background-image: url('https://example.com/social_distance.gif');
            background-repeat: no-repeat;
            background-size: cover;
          }
          .container {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
          }
          .frame {
            border: 1px solid #ccc;
            box-shadow: 0px 0px 5px 0px rgba(0,0,0,0.5);
            padding: 10px;
```

```
        }
      </style>
    </head>
    <body>
      <h1>Social Distancing Detection</h1>
      <div class="container">
        <div class="frame">
          <img src="{{ url_for('video_feed') }}" width="700" height="500" />
        </div>
      </div>
    </body>
  </html>
  '''
  return render_template_string(html)

# Function to generate video frames
def generate_frames():
  global processed_frames

  # Loop over the processed frames
  for frame in processed_frames:
    # Encode the frame in JPEG format
    (flag, encoded_frame) = cv2.imencode(".jpg", frame)

    # Ensure the frame was successfully encoded
    if not flag:
      continue

    # Yield the output frame in the byte format
    yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' +
```

```
            bytearray(encoded_frame) + b'\r\n')


# Route for the video feed
@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')


if __name__ == '__main__':
    # Start a new thread to process frames
    t = threading.Thread(target=process_frames)
    t.daemon = True
    t.start()

    # Run the Flask application
    print(f"To acces the Gloable link please click {public_url}")
    app.run(port=port_no)
```