“Social Distancing and Gait Detection”

INDEX

1. INTRODUCTION

When the novel coronavirus (Covid-19) pandemic emerges, the spread of the virus has left public keep anxiety if they do not have any effective cure. The World Health Organization (WHO) has declared Covid-19 as a pandemic due to the increase in the number of cases reported around the world. To contain the pandemic, many countries have implemented a lockdown where the government enforced that the citizens to stay at home during this critical period. The public health bodies such as the Centers for Disease Control and Prevention (CDC) had to make it clear that the most effective way to slow down the spread of Covid-19 is by avoiding close contact with other people. To flatten the curve on the Covid-19 pandemic, the citizens around the world are practicing physical distancing.

To implement social distancing, group activities and congregations such as travel, meetings, gatherings, workshops, praying had been banned during the quarantine period. The people are encouraged to use phone and email to manage and conduct events as much as possible to minimize the person-to-person contact. To further contain the spread of the virus, people are also informed to perform hygiene measures such as frequently washing hands, wearing mask and avoiding close contact with people who are ill. However, there is a difference between knowing what to do to reduce the transmission of the virus and putting them into practice.

Sometimes, people tend to forget or neglect the implementation of social distancing. Hence, this work aims to facilitate the enforcement of social distancing and gait by providing automated detection of social distance violation and human pose in workplaces and public areas using a deep learning model where bend pose will be considered as the inactivity of the person. In this section, we are going to use OpenCV to do real-time social distance detection from a live stream via our webcam. As you know videos are basically made up of frames, which are still images. We perform the detection for each frame in a video. So, when it comes to detecting the person in still image and detecting a person in a real-time video stream, there is not much difference between them. In this project, we are going to make a classifier that can differentiate between faces violated and not violated. So, for creating this classifier, we need data in the form of images.

1.1 Scope

Social Distance and Gait Detection Platform uses Artificial Network to recognize if a user is not maintaining social distance and inactive. In order to detect the social distance and bend pose, we need to break our project into three distinct phases, each with its own respective sub-steps:

1. **Social Distance Detection:** Here we'll focus on people maintaining social distance will be allowed within green bounding box and the distance between people can be calculated by Euclidian formula and the person who is not following will be represented as red bounding box and count of violations will be printed on screen by using YOLO V3, Deep Learning.

2. **Bend Pose Detection:** The detection of bend pose can be calculated by the distance calculation between the joints of neck and nose. Here the minimum distance will be given and that minimum value can be observed by giving many images and many videos as inputs. Thus, they will be detected as inactive persons since they are walking with bend position and this can be done by OpenCV, Deep Learning.

3. **Deployment:** Once the input is given, we can then move on to loading the distance and bend detector, performing social distance and bend pose detection, and then classifying each person as active or inactive and maintaining social distancing or not.

1.2 Existing System:

In present days, in locations where the social distancing and health monitoring is a mandatory action, checking whether a person is maintaining social distancing or not is done manually. It is difficult to check each and every person entering the public places and it is waste of man power. So, we have come up with the new system which reduces man power and can easily detect the healthy and unhealthy persons.

1.3 Proposed System

People are forced by laws to maintain WHO measures like maintain social distancing in public in many countries. These rules and laws were developed as an action to the exponential growth in cases and deaths in many areas. However, the process of monitoring large groups of people is becoming more difficult. The monitoring process involves the detection of anyone who is not following social distancing and unhealthy. Here we introduce a detection model that is based on computer vision and deep learning. The proposed model can be integrated with surveillance cameras to impede the COVID-19 transmission by allowing the detection of people following social distance or not and healthy or unhealthy. The model is integration between deep learning and classical machine learning techniques with OpenCV, tensor flow and YOLO. We have used yolo coco dataset for detecting the person class from 13,000classes present in the dataset. After detecting the persons were labelled with bounding boxes differentiated with green and red colours as violated and non-violated respectively. In the case of human bend pose, we took Open Pose as reference for pose estimation and we calculated bend pose by distance between neck and nose joint in a human body. We introduced a comparison between them to find the most suitable algorithm that achieved the highest accuracy and consumed the least time in the process of training and detection.

2. SYSTEM ANALYSIS

2.1 Software Requirements:

Python IDE:

An IDE (or Integrated Development Environment) is a program dedicated to software development. As the name implies, IDEs integrate several tools specifically designed for software development. These tools usually include: An editor designed to handle code and understand your code much better than a text editor. It usually provides features such as build automation, code linting, testing and debugging. This can significantly speed up your work. The downside is that IDEs can be complicated to use.

Google Colaboratory:

Google Colaboratory (also known as Colab) is a free Jupyter notebook environment that runs in the cloud and stores its notebooks on Google Drive. Colab was originally an internal Google project; an attempt was made to open source

all the code and work more directly upstream, leading to the development of the "Open in Colab" Google Chrome extension, but this eventually ended, and Colab development continued internally. The Colaboratory UI only allows for the creation of notebooks with Python 2 and Python 3 kernels; however, an existing notebook whose kernelspec is IR or Swift will also work, since both R and Swift are installed in the container. Julia language can also work on Colab (with e.g. Python and GPUs; Google's tensor processing units also work with Julia on Colab).

Jupyter Notebook:

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension. The Jupyter Notebook that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.

2.2 Hardware Requirements:

GPU (Graphics Processing Unit):

A programmable processor specialized for rendering all images on the computer's screen. A GPU provides the fastest graphics processing, and for gamers, the GPU is a stand-alone card plugged into the PCI Express (PCIe) bus. GPU circuitry can also be part of the motherboard chipset or on the CPU chip itself.
A GPU performs parallel operations. Although it is used for 2D data as well as for zooming and panning the screen, a GPU is essential for smooth decoding and rendering of 3D animations and video. The more sophisticated the GPU, the higher the resolution and the faster and smoother the motion. GPUs on stand-alone cards include their own memory, while GPUs built into the chipset or CPU chip share main memory with the CPU. Since GPUs perform parallel operations on multiple sets of data, they are increasingly used for scientific and AI applications that require repetitive computations. For example, in 2010, a Chinese supercomputer achieved the record for top speed using more than seven thousand GPUs in addition to its CPUs.

Webcam:

A webcam is a video camera that feeds or streams an image or video in real time to or through a computer to a computer network, such as the Internet. Webcams are typically small cameras that sit on a desk, attach to a user's monitor, or are built into the hardware. Webcams can be used during a video chat session involving two or more people, with conversations that include live audio and video. Webcam software enables users to record a video or stream the video on the Internet. As video streaming over the Internet requires much bandwidth, such streams usually use compressed formats. The maximum resolution of a webcam is also lower than most handheld video cameras, as

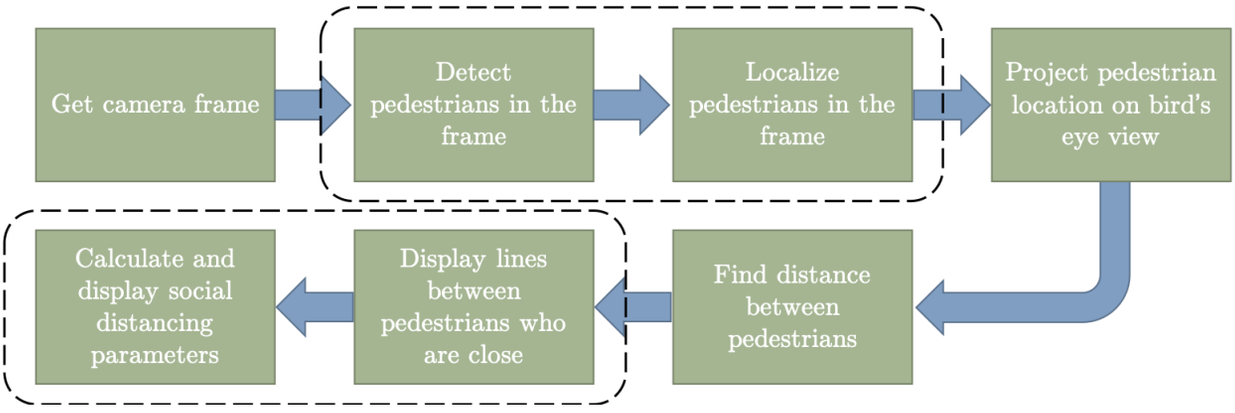
higher resolutions would be reduced during transmission. The lower resolution enables webcams to be relatively inexpensive compared to most video cameras, but the effect is adequate for video chat sessions.

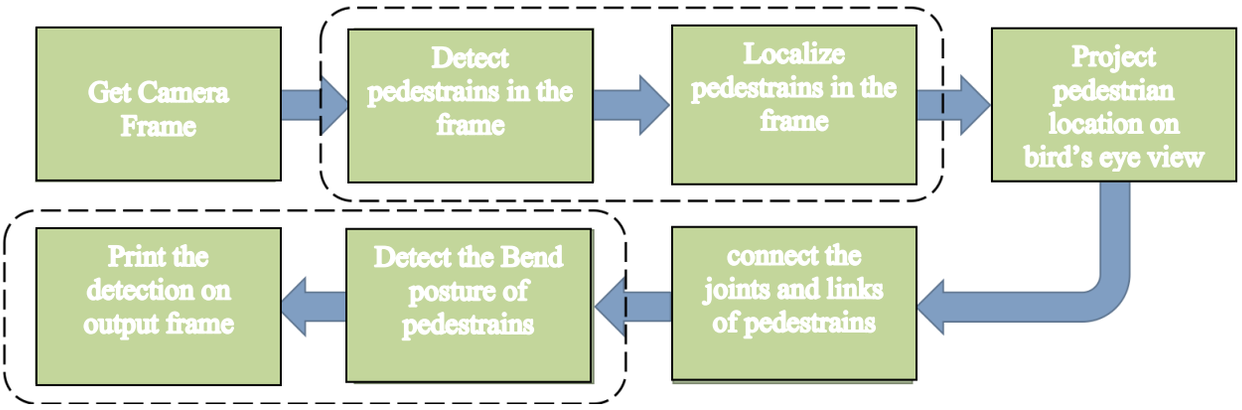
The term "webcam" (a clipped compound) may also be used in its original sense of a video camera connected to the Web continuously for an indefinite time, rather than for a particular session, generally supplying a view for anyone who visits its web page over the Internet. Some of them, for example, those used as online traffic cameras, are expensive, rugged professional video cameras.

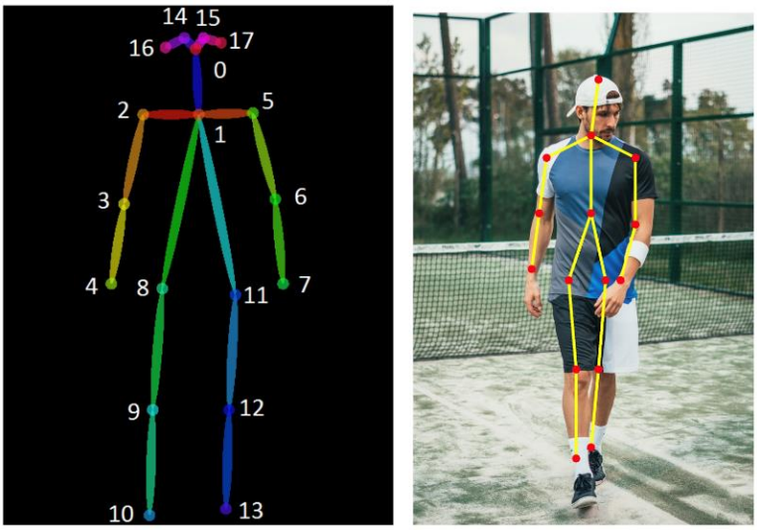
3. SYSTEM DESIGN

3.1 Architecture Design

The input of camera frame will be given to the algorithm and the algorithm detect the pedestrains in the frame and it will localize pedestrians in the frame. After locating the pedestrians, we will project the pedestrains location on bird's eye view. The next step is to find the distance between the people and display them by bounding boxes as violated and non-violated by red and green respectively. Atlast, the distance will be calculated and count of violations will be displayed on output frame.







3.2 Modules:

OpenCV (Open-Source Computer Vision Library)

It is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies. Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many start-ups such as Applied Minds, Video Surf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick-up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers.

To install OpenCV, the command is: **pip install Opencv-contrib-python print**

Yolo:

YOLO (You Only Look Once) real-time object detection algorithm, which is one of the most effective object detection algorithms that also encompasses many of the most innovative ideas coming out of the computer vision research community. Object detection is a critical capability of autonomous vehicle technology. It's an area of computer vision that's exploding and working so much better than just a few years ago.

Object detection is one of the classical problems in computer vision where you work to recognize what and where — specifically what objects are inside a given image and also where they are in the image. The problem of object detection is more complex than classification, which also can recognize objects but doesn't indicate where the object is located in the image. In addition, classification doesn't work on images containing more than one object.

YOLO uses a totally different approach. YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This model has a number of benefits over other object detection methods:

- YOLO is extremely fast.
- YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.
- YOLO learns generalizable representations of objects so that when trained on natural images and tested on artwork, the algorithm outperforms other top detection methods.

TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Itis used for both research and production at Google, TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). Tensor Flow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms

including Android and iOS. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google. Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems, not least RankBrain in Google search and the fun DeepDream project. It can run on single CPU systems, GPUs as well as mobile devices and large-scale distributed systems of hundreds of machines.

To install tensorflow, the command is: **pip install tensorflow**

Numpy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.
Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

To install numpy, the command is: **pip install numpy**

Argparse:

The argparse module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

To import Argpasre, the command is: **pip install argparse**

Math:

The math module is used to perform mathematical operations when some of the mathematical formulas don't work with basic math of python. Math is predefined module in python so, we need not to install it but we will import it.

To import math, the command is: **import math**

Imutils

Imutils are a series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and both Python 2.7 and Python 3.

To install Imutils, the command is: **pip install imutils**

4. SYSTEM IMPLEMENTATION

The algorithm is developed in to detect the social distancing and unhealthy persons by pend posture from the input video. The persons who are not maintaining social distance will be represented with red bounding box and whereas the persons who are maintaining social distance will be represented with green bounding box. In the case of Bend pose, the people will be detected with the body joints and then the bend posture will be calculated by the distance between nose and neck by putting minimum distance. If the distance is less than minimum distance then they will be detected as bent.

Code:

Social Distance Detection:

```
# 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

# **Creating the People Detection Function**

# 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
```

 # **Grab frames from video and make prediction measuring distances of detected people**

```
# 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/yolo-coco/christmas.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/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/yolo-coco/yolov3.weights"])

configPath = os.path.sep.join(["/content/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[0] - 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)
```

Bend Pose Detection:

#importing modules

```
import cv2 as cv
import math
import matplotlib.pyplot as plt
```

#importing weights from graph_pot.pb file

```
net = cv.dnn.readNetFromTensorflow("graph_opt.pb")
inWidth = 368
inHeight = 368
thr = 0.2
```

```
BODY_PARTS = { "Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3, "RWrist": 4, "LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8, "RKnee": 9, "RAnkle": 10, "LHip": 11, "LKnee": 12, "LAnkle": 13, "REye": 14, "LEye": 15, "REar": 16, "LEar": 17, "Background": 18 }

BODY_PART1 = { "Nose": 0 }

POSE_PAIRS = [ ["Neck", "RShoulder"], ["Neck", "LShoulder"], ["RShoulder", "RElbow"], ["RElbow", "RWrist"], ["LShoulder", "LElbow"], ["LElbow", "LWrist"], ["Neck", "RHip"], ["RHip", "RKnee"], ["RKnee", "RAnkle"], ["Neck", "LHip"], ["LHip", "LKnee"], ["LKnee", "LAnkle"], ["Neck", "Nose"], ["Nose", "REye"], ["REye", "REar"], ["Nose", "LEye"], ["LEye", "LEar"] ]

print(len(BODY_PARTS))

print(len(BODY_PART1))

BODY_PART2 = { "NE": 1 }

element = next(iter(BODY_PART2))

print(element)

print(len(element))

#imput image is given

img = cv.imread("lean_pose1.jpg")

plt.imshow(img)

plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))

#For input image

def pose_estimation(frame):

    frameWidth = frame.shape[1]

    frameHeight = frame.shape[0]

    inp = cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight),(127.5, 127.5, 127.5), swapRB=True, crop=False)

    net.setInput(inp)

    out = net.forward()
```

```
out = out[:, :19, :, :]

assert(len(BODY_PARTS) == out.shape[1])

points = []

for i in range(len(BODY_PARTS)):

    # Slice heatmap of corresponding body's part.

    heatMap = out[0, i, :, :]

    # Originally, we try to find all the local maximums. To simplify a sample

    # we just find a global one. However only a single pose at the same time

    # could be detected this way.

    _, conf, _, point = cv.minMaxLoc(heatMap)

    x = (frameWidth * point[0]) / out.shape[3]

    y = (frameHeight * point[1]) / out.shape[2]

    print("x=",x)

    print("y=",y)

    # Add a point if it's confidence is higher than threshold.

    points.append((int(x), int(y)) if conf > thr else None)

for i in range(len(BODY_PART1)):

    heatMap = out[0, i, :, :]

    _, conf, _, point = cv.minMaxLoc(heatMap)

    x1 = (frameWidth * point[0]) / out.shape[3]

    y1 = (frameHeight * point[1]) / out.shape[2]

    print("       ")

    print("x1=",x1)
```

```
    print("y1=",y1)
for i in range(len(element)):
    heatMap = out[0, i, :, :]
    _, conf, _, point = cv.minMaxLoc(heatMap)
    x2 = (frameWidth * point[0]) / out.shape[3]
    y2 = (frameHeight * point[1]) / out.shape[2]
    print("       ")
    print("x2=",x2)
    print("y2=",y2)
    print(" ")
distance = ((((x2-x1)**2)+((y2-y1)**2))**0.5)
print("the distance between them is ", distance)
for pair in POSE_PAIRS:
    partFrom = pair[0]
    partTo = pair[1]
    assert(partFrom in BODY_PARTS)
    assert(partTo in BODY_PARTS)
    idFrom = BODY_PARTS[partFrom]
    idTo = BODY_PARTS[partTo]
    if points[idFrom] and points[idTo]:
        cv.line(frame, points[idFrom], points[idTo], (0, 0, 0), 3)
        cv.ellipse(frame, points[idFrom], (3, 3), 0, 0, 360, (255, 255, 255), cv.FILLED)
        cv.ellipse(frame, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
```

```
    t, _ = net.getPerfProfile()
    freq = cv.getTickFrequency() / 1000
    cv.putText(frame, '%.2fms' % (t / freq), (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
    return frame
    return distance
expected_image = pose_estimation(img)
plt.imshow(cv.cvtColor(expected_image, cv.COLOR_BGR2RGB))=
#For input video
lean = cv.VideoCapture('video1.gif')
lean.set(3,800)
lean.set(4,800)
if not lean.isOpened():
    lean = cv.VideoCapture(0)
if not lean.isOpened():
    raise IOError("can't open the video file")
while cv.waitKey(1) < 0:
    hasFrame,frame = lean.read()
    if not hasFrame:
        cv.WaitKey()
        break
    frameWidth = frame.shape[1]
    frameHeight = frame.shape[0]
    inp = cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight),(127.5, 127.5, 127.5), swapRB=True, crop=False)
```

```
net.setInput(inp)

out = net.forward()

out = out[:, :19, :, :]

assert(len(BODY_PARTS) == out.shape[1])

points = []

for i in range(len(BODY_PARTS)):

    # Slice heatmap of corresponding body's part.

    heatMap = out[0, i, :, :]

    # Originally, we try to find all the local maximums. To simplify a sample

    # we just find a global one. However only a single pose at the same time

    # could be detected this way.

    _, conf, _, point = cv.minMaxLoc(heatMap)

    x = (frameWidth * point[0]) / out.shape[3]

    y = (frameHeight * point[1]) / out.shape[2]

    # Add a point if it's confidence is higher than threshold.

    points.append((int(x), int(y)) if conf > thr else None)

for i in range(len(BODY_PART1)):

    heatMap = out[0, i, :, :]

    _, conf, _, point = cv.minMaxLoc(heatMap)

    x1 = (frameWidth * point[0]) / out.shape[3]

    y1 = (frameHeight * point[1]) / out.shape[2]

    print("       ")

    print("x1=",x1)
```

```
    print("y1=",y1)
for i in range(len(element)):
    heatMap = out[0, i, :, :]
    _, conf, _, point = cv.minMaxLoc(heatMap)
    x2 = (frameWidth * point[0]) / out.shape[3]
    y2 = (frameHeight * point[1]) / out.shape[2]
    print("       ")
    print("x2=",x2)
    print("y2=",y2)
distance = ((((x2-x1)**2)+((y2-y1)**2))**0.5)
print("the distance between them is ", distance)
for i in range(len('lean.gif')):
    if distance <71:
        text = "bend pose detected: {}".format(distance)
        cv.putText(frame, text, (10, frame.shape[0] - 25),cv.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0),2)
        print("detected")
    else:
        text1 = "bend pose not detected: {}".format(distance)
        cv.putText(frame, text1, (10, frame.shape[0] - 25),cv.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0),2)
        print("not detected")
for pair in POSE_PAIRS:
    partFrom = pair[0]
    partTo = pair[1]
```

```
        assert(partFrom in BODY_PARTS)
        assert(partTo in BODY_PARTS)
        idFrom = BODY_PARTS[partFrom]
        idTo = BODY_PARTS[partTo]
        if points[idFrom] and points[idTo]:
            cv.line(frame, points[idFrom], points[idTo], (0, 255, 0), 3)
            cv.ellipse(frame, points[idFrom], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
            cv.ellipse(frame, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
    t, _ = net.getPerfProfile()
    freq = cv.getTickFrequency() / 1000
    cv.putText(frame, '%.2fms' % (t / freq), (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
    cv.imshow('video executed successfully ', frame)
```

#For Live Video Input

```
lean = cv.VideoCapture(1)
lean1 = cv.VideoCapture('video2.mp4')
lean.set(cv.CAP_PROP_FPS,10)
lean.set(3,800)
lean.set(4,800)
if not lean.isOpened():
    lean = cv.VideoCapture(0)
if not lean.isOpened():
    raise IOError("can't access and open webcam")
while cv.waitKey(1) < 0:
```

```
hasFrame,frame = lean.read()
if not hasFrame:
    cv.WaitKey()
    break
frameWidth = frame.shape[1]
frameHeight = frame.shape[0]
inp = cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight),(127.5, 127.5, 127.5), swapRB=True, crop=False)
net.setInput(inp)
out = net.forward()
out = out[:, :19, :, :]
assert(len(BODY_PARTS) == out.shape[1])
points = []
for i in range(len(BODY_PARTS)):
    # Slice heatmap of corresponding body's part.
    heatMap = out[0, i, :, :]
    # Originally, we try to find all the local maximums. To simplify a sample
    # we just find a global one. However only a single pose at the same time
    # could be detected this way.
    _, conf, _, point = cv.minMaxLoc(heatMap)
    x = (frameWidth * point[0]) / out.shape[3]
    y = (frameHeight * point[1]) / out.shape[2]
    # Add a point if it's confidence is higher than threshold.
    points.append((int(x), int(y)) if conf > thr else None)
```

```
for i in range(len(BODY_PART1)):
    heatMap = out[0, i, :, :]
    _, conf, _, point = cv.minMaxLoc(heatMap)
    x1 = (frameWidth * point[0]) / out.shape[3]
    y1 = (frameHeight * point[1]) / out.shape[2]
    print("       ")
    print("x1=",x1)
    print("y1=",y1)
for i in range(len(element)):
    heatMap = out[0, i, :, :]
    _, conf, _, point = cv.minMaxLoc(heatMap)
    x2 = (frameWidth * point[0]) / out.shape[3]
    y2 = (frameHeight * point[1]) / out.shape[2]
    print("       ")
    print("x2=",x2)
    print("y2=",y2)
distance = ((((x2-x1)**2)+((y2-y1)**2))**0.5)
print("the distance between them is ", distance)
for i in range(len('video2.mp4')):
    if distance <100:
        text = "bend pose detected: {}".format(distance)
        cv.putText(frame, text, (10, frame.shape[0] - 25),cv.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0),2)
        print("detected")
```

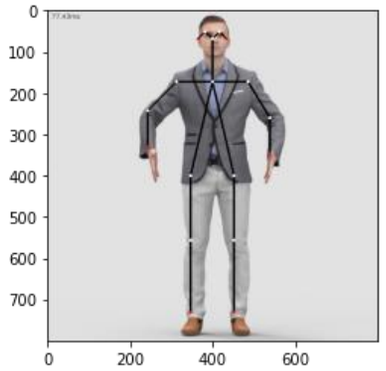
```
    else:
        text1 = "bend pose not detected: {}".format(distance)
        cv.putText(frame, text1, (10, frame.shape[0] - 25),cv.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0),2)
        print("not detected")
for pair in POSE_PAIRS:
    partFrom = pair[0]
    partTo = pair[1]
    assert(partFrom in BODY_PARTS)
    assert(partTo in BODY_PARTS)
    idFrom = BODY_PARTS[partFrom]
    idTo = BODY_PARTS[partTo]
    if points[idFrom] and points[idTo]:
        cv.line(frame, points[idFrom], points[idTo], (0, 255, 0), 3)
        cv.ellipse(frame, points[idFrom], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
        cv.ellipse(frame, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
t, _ = net.getPerfProfile()
freq = cv.getTickFrequency() / 1000
cv.putText(frame, '%.2fms' % (t / freq), (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
cv.imshow('LIVE WEBCAM ', frame)
```

5. Output:

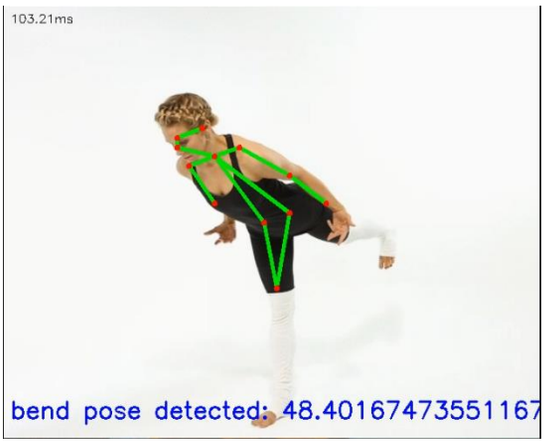
Social Distancing Detection:

Gait and Bend Posture Detection:

Output for input image:



Output for input video:



Output for live video:

6. APPLICATIONS

Hospitals: Implement effective supervision of staff and patients, entering your premises. Encourage all personnel to be personal protective gear at all times to minimize healthcare risks. Manage patients' behaviour to verify that all quarantined patients abide by the rules and maintain social distancing. Perform detection and show the count of violations automatically to avoid stretching your human resources on disciplinary measures, instead of keeping them focused on their core duties.

Airports and railway stations: Ensure that all transiting passengers and your staff remain safe at all times with a real-time social distancing and gait detection system, capable of effectively recognizing the distance even in crowded areas. Control social distancing and unhealthy passengers on during the transit without placing any extra burden on your personnel or undermining passenger experience with personal admonishments.

Public transportation: Encourage passengers to follow the health safety measures and maintain the social distance on at the stations and during their entire journey. Relieve your drivers and supporting staff from the daunting chore of enforcing proper behaviour. Instill more confidence in passengers who are choosing to use public transportation instead of personal vehicles, by demonstrating that they'll ride along with others who respect the necessary safety measures.

Retail locations: Verify that all shoppers, entering your premises, are maintaining social distance and helathy without placing a security team on the door of your venue. Maintain a delightful customer experience and promote proper behaviour among your staff using a combo of social distancing and gait detection and personalized alerts, issued to employees who are not following the rules. Re-attract the reluctant, health-conscious consumers to your brick-and-mortar locations by showcasing how your business stands for safety above all.

7. ADVANTAGES

- **Easy implementation:** software or hardware implementation encompasses all the post-sale processes involved in something operating properly in its environment, including analyzing requirements, installation, configuration, customization, running, testing, systems integrations, user training, delivery and making necessary changes. The word "deployment" is sometimes used to mean the same thing.

- **Staff Friendly:** An employee-friendly workplace is one that places a priority on making employees happy at work. A workforce with a high level of job satisfaction can affect productivity, customer satisfaction, and overall profitability.

- **Camera Agnostic:** Device agnosticism is the capacity of a computing component to work with various systems without requiring any special adaptations. The term can apply to either hardware or software. In an IT context, agnosticism refers to anything that is designed to be compatible across most common systems.

- **No new hardware needed:** System has inbuilt GPU and WEBCAM that are required for the project to get executed.

DISADVANTAGES

- **Varying body angles:** Multiview gait recognition is an extension of appearance-based frontal image recognition. Here, gallery images of every subject at many different poses are needed. Most algorithms in this category require several images of each subject in the data base and consequently require much more computation for searching and memory for storage.

- **Lack of clarity:** Moving images does not have much clarity and thus becomes a challenging task.

- **Limited** number of gait detections.

8. CONCLUSION

As the technology is blooming with emerging trends the availability so we have novel social distancing and bend pose detection which can possibly contribute to public healthcare. The architecture consists of Yolo and OpenCV as the backbone and it can be used for high and low computation scenarios. In order to extract more robust features, we utilize transfer learning to adopt weights from a similar task of detection, which is trained on a very large dataset. We used OpenCV, tensor flow, Yolo and CNN to detect whether people were healthy or not and maintaining social distancing or not. The models were tested with images and real-time video streams. The accuracy of the model is achieved and the optimization of the model is a continuous process and we are building a highly accurate solution by tuning the hyper parameters. This specific model could be used as a use case for edge analytics. Furthermore, the proposed method achieves state-of-the-art results on a public dataset. By the development of social distancing and gait detection, we can detect if the person is healthy or not and maintaining social distancing or not and allow their entry would be of great help to society.

FUTURE SCOPE

More than fifty countries around the world have recently initiated social distance compulsory. People have to cover their faces in public, supermarkets, public transports, offices, and stores. Retail companies often use software to count the number of people entering their stores. They may also like to measure impressions on digital displays and promotional screens. We are planning to improve our Detection tool and release it as an open-source project. Our software can be equated to any existing USB, IP cameras, and CCTV cameras to detect people who are violating. This detection live video feed can be implemented in web and desktop applications so that the operator can see notice the count of violations. Software operators can also get an image in case someone is not healthy and social distance violators. We would like to give the person an announcement that please maintain social distance and go for a check-up if he is unhealthy.

9. REFERENCES

[1] Özbek, M.M., Syed, M. and Öksüz, I., 2021. Subjective analysis of social distance monitoring using YOLO v3 architecture and crowd tracking system. Turkish Journal of Electrical Engineering & Computer Sciences, 29(2).

[2] Ahmed, Imran, Misbah Ahmad, Joel JPC Rodrigues, Gwanggil Jeon, and Sadia Din. "A deep learning-based social distance monitoring framework for COVID-19." Sustainable Cities and Society 65 (2021): 102571.

[3] Ahmed, I., Ahmad, M., Rodrigues, J.J., Jeon, G. and Din, S., 2021. A deep learning-based social distance monitoring framework for COVID-19. Sustainable Cities and Society, 65, p.102571.

[4] Bhambani K, Jain T, Sultanpure KA. Real-time Face Mask and Social Distancing Violation Detection System using YOLO. In2020 IEEE Bangalore Humanitarian Technology Conference (B-HTC) 2020 Oct 8 (pp. 1-6). IEEE.

[5] Wang, Liang, Tieniu Tan, Huazhong Ning, and Weiming Hu. "Silhouette analysis-based gait recognition for human identification." IEEE transactions on pattern analysis and machine intelligence 25, no. 12 (2003): 1505-1518.

[6] Nixon, M.S., Carter, J.N., Cunado, D., Huang, P.S. and Stevenage, S.V., 1996. Automatic gait recognition. In Biometrics (pp. 231-249). Springer, Boston, MA.

[7] Lee, L. and Grimson, W.E.L., 2002, May. Gait analysis for recognition and classification. In Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition (pp. 155-162). IEEE.

[8] Desai, Miral M., and Hiren K. Mewada. "Review on Human Pose Estimation And Human Body Joints Localization." International Journal of Computing and Digital Systems 10 (2021).

[9] Nour M, Gardoni M, Renaud J, Gauthier S. Real-time detection and motivation of eating activity in elderly people with dementia using Pose Estimation with TensorFlow and OpenCV.