1. INTRODUCTION

1.1 OBJECTIVE

The COVID-19 pandemic has reshaped life as we know it. Many of us are staying home, avoiding people on the street and changing daily habits, like going to school or work, in ways we never imagined.

Some viruses – like the virus that causes COVID-19 – spread easily, according to the Centers for Disease Control and Prevention. Social distancing puts space between individuals. If someone is sick and there are no people around, a virus cannot spread.

“Social distancing means we are doing our best to stay away from people so as to limit the spread of coronavirus,” says UNC family medicine physician Sarah Ruff, MD. “It’s really important because if everyone gets coronavirus at once and ends up in the hospital at the same time, our health care system would be overwhelmed. We need to ‘flatten the curve.”

Social distancing, more appropriately called physical distancing, is not always straight forward and if you are sometimes unsure, you are not alone. Even though I have experience working in infectious disease prevention and control, I too face moments of indecision translating what I know about social distancing into daily life.

In order to put the theory into practice, we need to adhere to two principles. Firstly, assume everyone we meet has coronavirus, regardless of how they look or who they are. And secondly, also assume that we have coronavirus, and could give it to other people. It’s important we all act as though we are potentially carrying the virus.

I could go on to list all the dos and don’ts of social distancing, but there’s an easier way to understand what you need to do: The smelly-person rule.

I pretend that everyone in the world has not washed for six weeks, including me, and I behave accordingly. In this imaginary world where no one has washed, and everybody smells a bit, I definitely don’t want to be too physically close to anyone - certainly I want to be at least one to two metres away.

I also don’t want to greet anyone with a handshake, a hug, and definitely not a kiss. I want to avoid any meetings I can - there’s nothing worse than being in a room full of people who don’t smell good.

Naturally, in my imagined malodourous world, all activities that bring us into close contact with a lot of smelly unwashed people must be avoided including travelling, cafes, social functions, family gatherings and work meetings.

Remember, distancing ourselves from others protects everybody - particularly the more vulnerable in society.

It is important to be clear that it is close and extended personal contact that increases our risk of transmission. Coming into close proximity to someone incidentally is unlikely to lead to transmission. However, the longer we are in close contact, the greater our risk.

Our project is based on the monitoring of public places to make sure social distancing is being followed. We have used various technology and techniques to complete the task. This can be applied in many public places like markets, malls, or even streets and can be used to monitor people.

Social distancing is crucial to preventing the spread of disease. Using computer vision technology based on OpenCV and YOLO-based deep learning, we are able to estimate the social distance of people in video streams.

1.2 PROJECT DESCRIPTION

We have used OpenCV, computer vision, and deep learning to implement social distancing detectors.

The steps to build a social distancing detector include:

1. Apply object detection to detect all people (and *only* people) in a video stream.
2. Compute the pairwise distances between all detected people
3. Based on these distances, check to see if any two people are less than *N* pixels apart

The simplest approach to build an Object Detection model is through a Sliding Window approach. As the name suggests, an image is divided into regions of a particular size and then every region is classified into the respective classes.

In this project we aim at using YOLO. YOLO stands for You Only Look Once. It’s a fast-operating object detection system that can recognize various object types in a single frame more precisely than other detection systems.

After detecting people in the frame, we follow the steps mentioned below:

1. Calculate Euclidean distance between two points
2. Convert centre coordinates into rectangle coordinates
3. Filter the person class from the detections and get a bounding box centroid for each person detected
4. Check which person bounding boxes are close to each other
5. Display risk analytics and risk indicators

For the most accurate results, we can *calibrate our camera* through intrinsic/extrinsic parameters so that we can map *pixels* to *measurable units.*

An easier alternative (but less accurate) method would be to apply triangle similarity calibration but, in this project, we have calculated Euclidean distance between each combination of every two people detected.

Both of these methods can be used to map pixels to measurable units.

For the sake of simplicity, our OpenCV social distancing detector implementation will rely on pixel distances.

The project is integrated with another project and Graphical use interface has been implemented to make it easier to use for common people with no coding knowledge or no understanding of the technology used to develop the entire project.

For creating GUI we have used Tkinter library that is used with python to create user friendly graphical interfaces with ease.

2. SYSTEM STUDY

2.1 TOOLS AND TECHNOLOGIES USED

Python - Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming.

Anaconda Navigator - Anaconda is a conditional free and open-source[6] distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator,[13] as a graphical alternative to the command line interface (CLI).

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.

Sublime Text - Sublime Text is a shareware cross-platform source code editor with a Python application programming interface (API). It natively supports many programming languages and markup languages, and functions can be added by users with plugins, typically community-built and maintained under free-software licenses. It is needed in this project as jupyter notebook does not work well with graphic contents.

Numpy – Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

Open-CV - OpenCV (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real-time computer vision.[1] Originally developed by Intel, it was later supported by Willow Garage then Itseez. Advance vision research by providing not only open but also optimized code for basic vision infrastructure.

Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable. Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Ego motion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object detection

Imultis - A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much easier with OpenCV and both Python 2.7 and Python 3.

Tkinter - The Tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and Tkinter are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

Most of the time, Tkinter is all you really need, but a number of additional modules are available as well. The Tk interface is located in a binary module named _Tkinter. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

2.2 HARDWARE & SOFTWARE REQUIREMENTS

Webcam – To take video input

- Inbuilt sensitive microphone and image sensor high quality CMOS sensor
- Image resolution interpolated to 25 mega pixels with 6 light sensors
- Image control color saturation, brightness, sharpness and brightness is adjustable; Snap shot switch for taking still pictures
- Anti-flicker 50Hz, 60Hz or outdoor; Resolution hardware: 500K pixels; Image quality: RGB24 or I420
- Exposure: Auto or manual and angle of view: 58 Degree; Interface: USB2.0; Frame rate: 30 fps

Laptop / Desktop – Any Standard laptop / Desktop

3.YOLO OBJECT DETECTOR AND COCO DATASET

When it comes to deep learning-based object detection, there are three primary object detectors you'll encounter:

- R-CNN and their variants, including the original R-CNN, Fast R- CNN, and Faster R-CNN
- Single Shot Detector (SSDs)
- YOLO

R-CNNs are one of the first deep learning-based object detectors and are an example of a two-stage detector.

1. In the first R-CNN publication, Rich feature hierarchies for accurate object detection and semantic segmentation, (2013) Girshick et al. proposed an object detector that required an algorithm such as Selective Search (or equivalent) to propose candidate bounding boxes that could contain objects.

2. These regions were then passed into a CNN for classification, ultimately leading to one of the first deep learning-based object detectors.

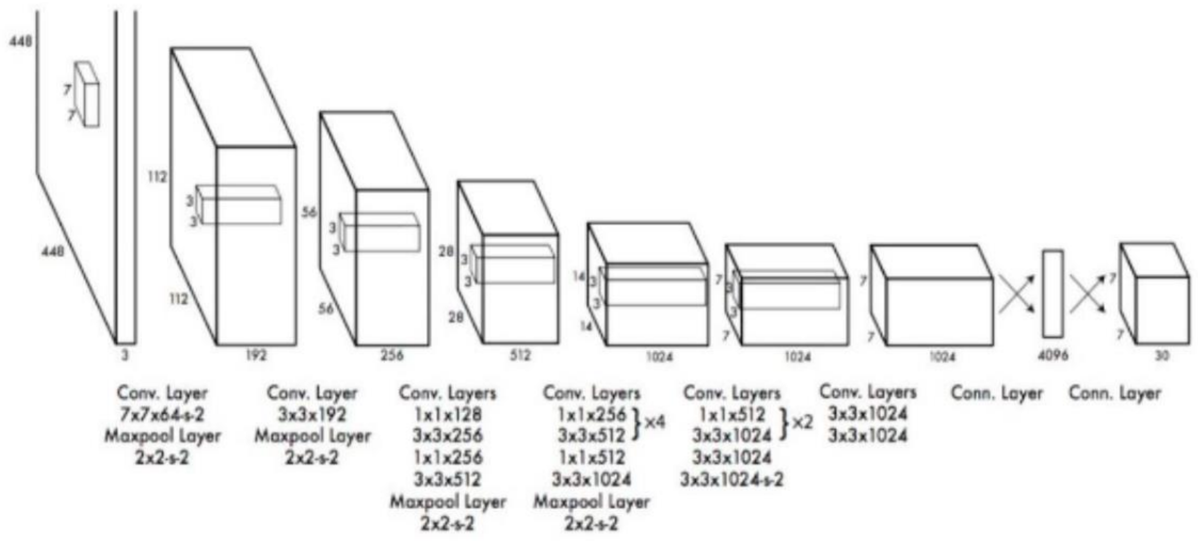
The problem with the standard R-CNN method was that it was painfully slow and not a complete end-to-end object detector.

Girshick et al. published a second paper in 2015, entitled Fast R- CNN. The Fast R-CNN algorithm made considerable improvements to the original R-CNN, namely increasing accuracy and reducing the time it took to perform a forward pass; however, the model still relied on an external region proposal algorithm.

It wasn't until Girshick et al.'s follow-up 2015 paper, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, that R-CNNs became a true end-to-end deep learning object detector by removing the Selective Search requirement and instead relying on a Region Proposal Network (RPN) that is (1) fully convolutional and (2) can predict the object bounding boxes and "objectness" scores (i.e., a score quantifying how likely it is a region of an image may contain an image). The outputs of the RPNs are then passed into the R-CNN component for final classification and labelling.

While R-CNNs tend to very accurate, the biggest problem with the R-CNN family of networks is their speed — they were incredibly slow, obtaining only 5 FPS on a GPU.

To help increase the speed of deep learning-based object detectors, both Single Shot Detectors (SSDs) and YOLO use a one-stage detector strategy.



448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

The Architecture of YOLO

These algorithms treat object detection as a regression problem, taking a given input image and simultaneously learning bounding box coordinates and corresponding class label probabilities.

In general, single-stage detectors tend to be less accurate than two-stage detectors but are significantly faster.

YOLO is a great example of a single stage detector.

First introduced in 2015 by Redmon et al., their paper, You Only Look Once: Unified, Real-Time Object Detection, details an object detector capable of super real-time object detection, obtaining 45 FPS on a GPU.

Note: A smaller variant of their model called “Fast YOLO” claims to achieve 155 FPS on a GPU.

YOLO has gone through a number of different iterations, including YOLO9000: Better, Faster, Stronger (i.e., YOLOv2), capable of detecting over 9,000 object detectors.

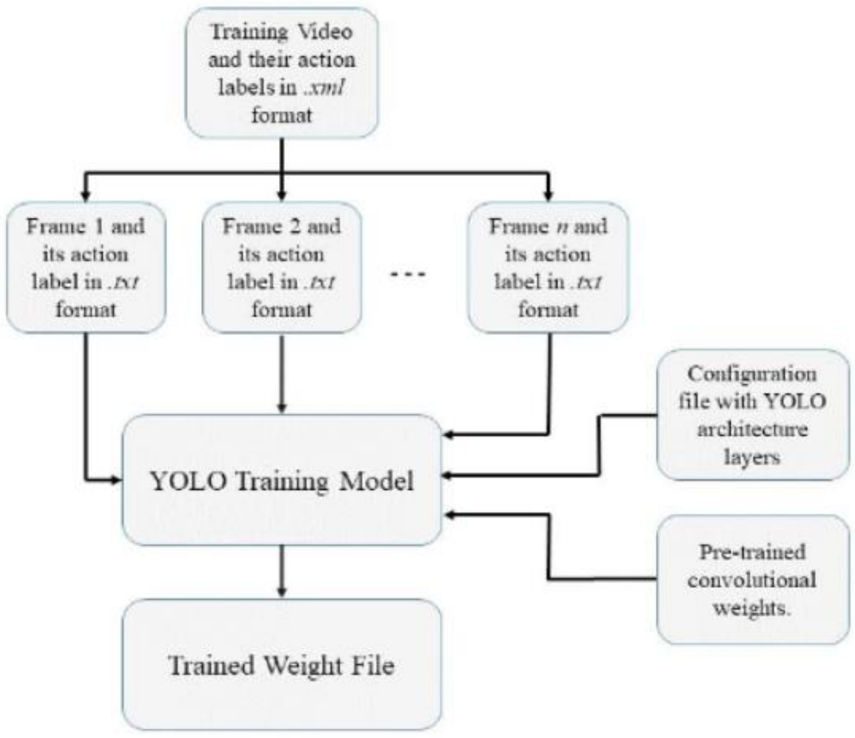
Redmon and Farhadi are able to achieve such a large number of object detections by performing joint training for both object detection and classification. Using joint training the authors trained YOLO9000 simultaneously on both the ImageNet classification dataset and COCO detection dataset. The result is a YOLO model, called YOLO9000, that can predict detections for object classes that don't have labelled detection data.

While interesting and novel, YOLOv2's performance was a bit underwhelming given the title and abstract of the paper.

On the 156 class version of COCO, YOLO9000 achieved 16% mean Average Precision (mAP), and yes, while YOLO can detect 9,000 separate classes, the accuracy is not quite what we would desire.

Redmon and Farhadi recently published a new YOLO paper, YOLOv3: An Incremental Improvement (2018). YOLOv3 is significantly larger than previous models but is, in my opinion, the best one yet out of the YOLO family of object detectors.

We'll be using YOLOv3 in this blog post, in particular, YOLO trained on the COCO dataset.


Training Video and their action labels in .xml format
Frame 1 and its action label in .txt format
Frame 2 and its action label in .txt format
- - -
Frame n and its action label in .txt format
Configuration file with YOLO architecture layers
YOLO Training Model
Pre-trained convolutional weights.
Trained Weight File

The flowchart of training of YOLO

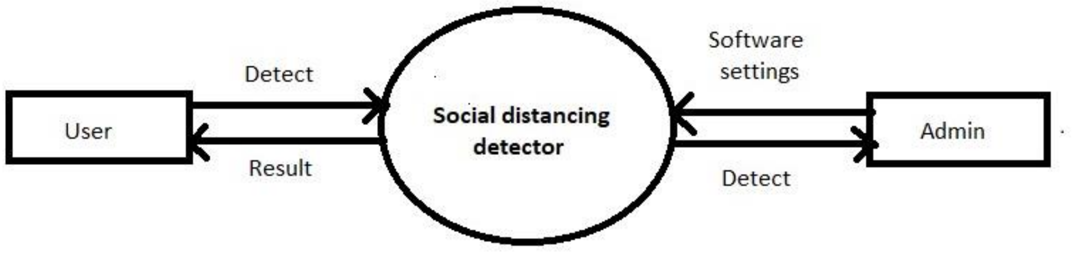
The COCO dataset consists of 80 labels, including, but not limited to:

- People
- Bicycles
- Cars and trucks
- Airplanes
- Stop signs and fire hydrants
- Animals, including cats, dogs, birds, horses, cows, and sheep, to name a few
- Kitchen and dining objects, such as wine glasses, cups, forks, knives, spoons, etc.
- …and much more!

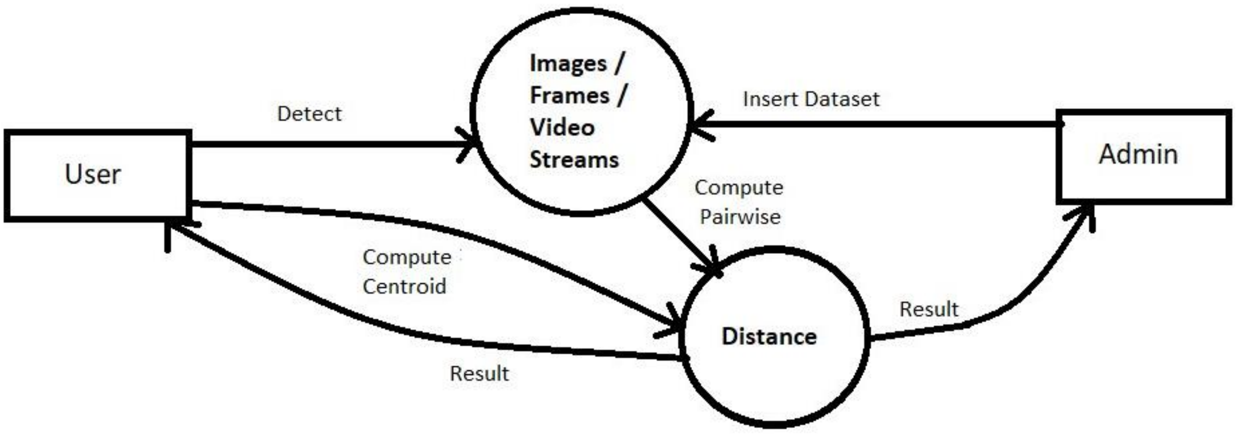
4. SYSTEM ANALYSIS AND DESIGN

4.1 CONTEXT DIAGRAM (DFD)

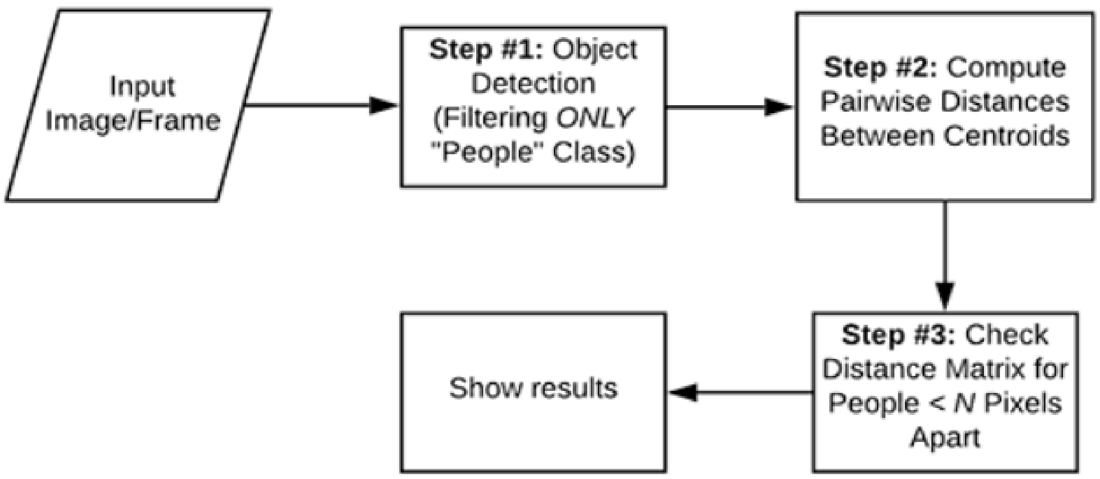
Level 0:



Level 1:



4.2 FLOW Diagram


Input Image/Frame
Step #1: Object Detection (Filtering ONLY "People" Class)
Step #2: Compute Pairwise Distances Between Centroids
Step #3: Check Distance Matrix for People < N Pixels Apart
Show results

5. IMPLEMENTATION

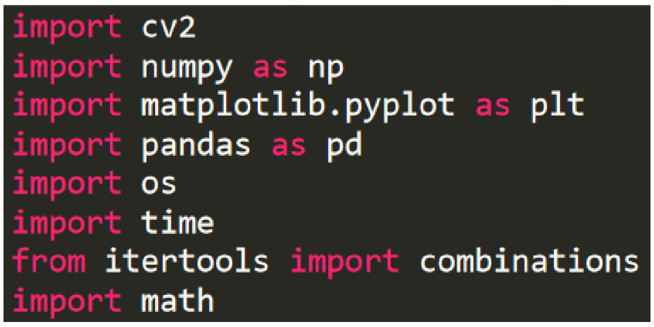
The implementation of this project is done in python language using OpenCV and concepts deep learning and Tkinter for the GUI part. The code is in 2 parts:

1. The Social Distancing Detector
2. GUI for integrating the project

The social distancing detector consists of the main project model in which we use a detector algorithm to detect people and perform necessary social distancing task.

The implementation code for detecting people is given below.

First, we import the required packages and modules.



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import time
from itertools import combinations
import math
```

Cv2 – OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

Numpy – NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Matplotlib.pyplot – matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Pandas – pandas is a software library written for the Python programming language for data manipulation and analysis.

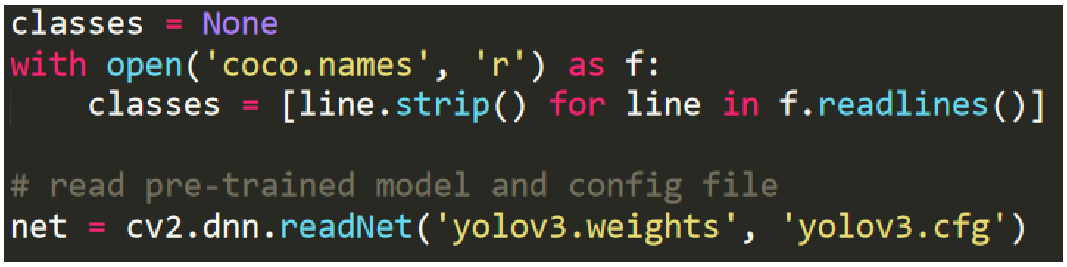
Os – The OS module in Python provides functions for interacting with the operating system.

Time – time module provides functions for working with times, and for converting between representations.

Combinations – The itertools. combinations() function takes two arguments—an iterable inputs and a positive integer n —and produces an iterator over tuples of all combinations of n elements in inputs

Math – This module provides access to the mathematical functions defined by the C standard.

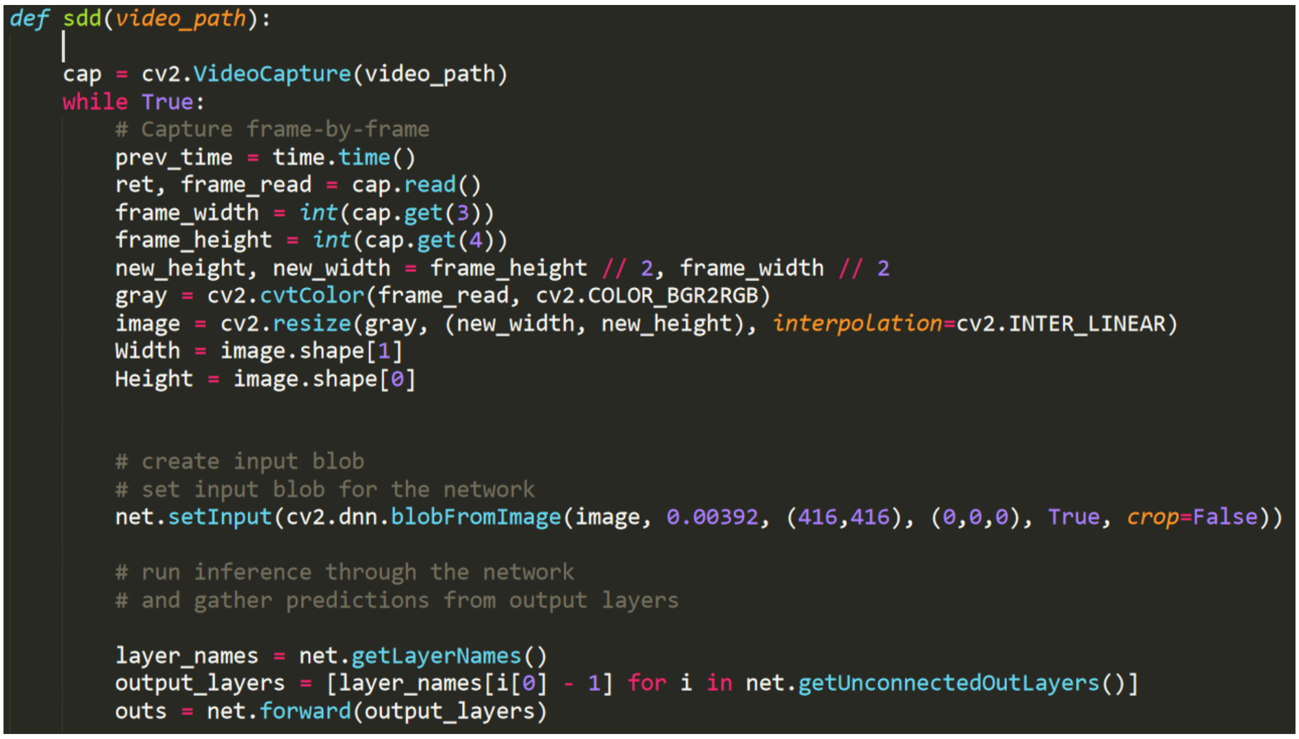
Now, we will create a list with pretrained object names. We'll read pretrained model and config files. The YOLO object detector is pre-trained on coco dataset. It uses two additional files 'yolov3.weights' that contains the weights used in training the model and configuration file 'yolov3.cfg'.



```
classes = None
with open('coco.names', 'r') as f:
    classes = [line.strip() for line in f.readlines()]

# read pre-trained model and config file
net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')
```

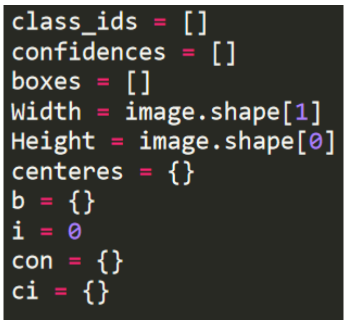
We use videoCapture() function to start the video and read each frame of the video in the form of image using a loop. After reading each frame, we will apply the yolo detector on frames to generate output frame

For this, we'll create and set input blob for the network, run inference through the network and gather predictions from output layers.



```python
def sdd(video_path):

    cap = cv2.VideoCapture(video_path)
    while True:
        # Capture frame-by-frame
        prev_time = time.time()
        ret, frame_read = cap.read()
        frame_width = int(cap.get(3))
        frame_height = int(cap.get(4))
        new_height, new_width = frame_height // 2, frame_width // 2
        gray = cv2.cvtColor(frame_read, cv2.COLOR_BGR2RGB)
        image = cv2.resize(gray, (new_width, new_height), interpolation=cv2.INTER_LINEAR)
        Width = image.shape[1]
        Height = image.shape[0]


        # create input blob
        # set input blob for the network
        net.setInput(cv2.dnn.blobFromImage(image, 0.00392, (416,416), (0,0,0), True, crop=False))

        # run inference through the network
        # and gather predictions from output layers

        layer_names = net.getLayerNames()
        output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
        outs = net.forward(output_layers)
```

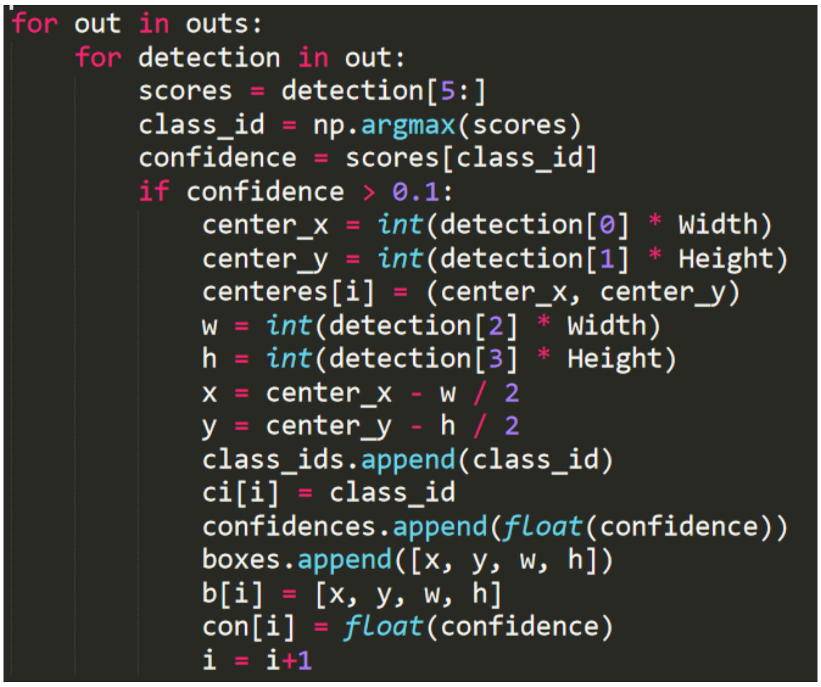
In the following code we have defined the necessary variables and data structures that will be used further.



```python
class_ids = []
confidences = []
boxes = []
Width = image.shape[1]
Height = image.shape[0]
centeres = {}
b = {}
i = 0
con = {}
ci = {}
```

Next, we use the output layer detections and obtain highest score, class_id (obtained from yolo pre-trained model) and confidence. Confidence is the minimum probability to filter weak detections.

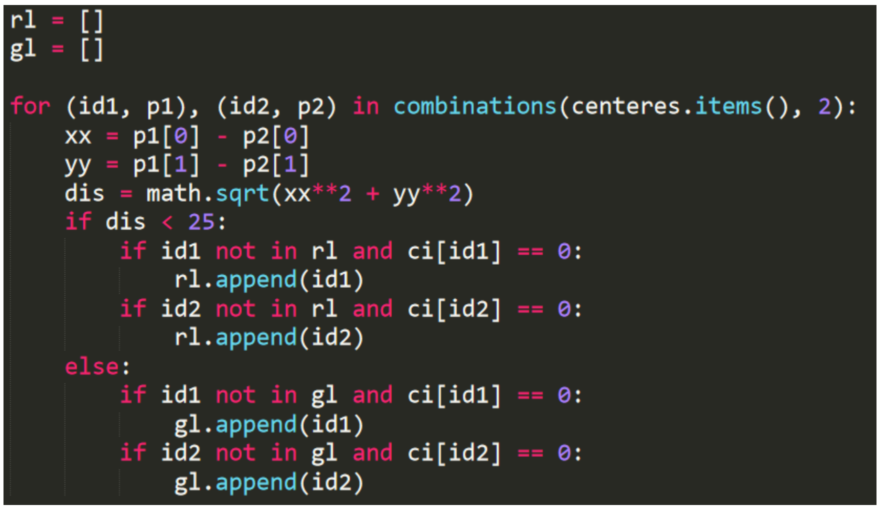
Also, we calculate the co-ordinates required to draw the rectangle around detected part of the frame. (Note: the following code is inside the sdd() function.)



```python
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.1:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            centeres[i] = (center_x, center_y)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            ci[i] = class_id
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])
            b[i] = [x, y, w, h]
            con[i] = float(confidence)
            i = i+1
```

In the next step, we will use combinations() function to obtain all possible pairs of co-ordinates and calculate distance between them. We have used the formula for Euclidean distance between two points. Here, we have used the 25 as the pixel value (N).

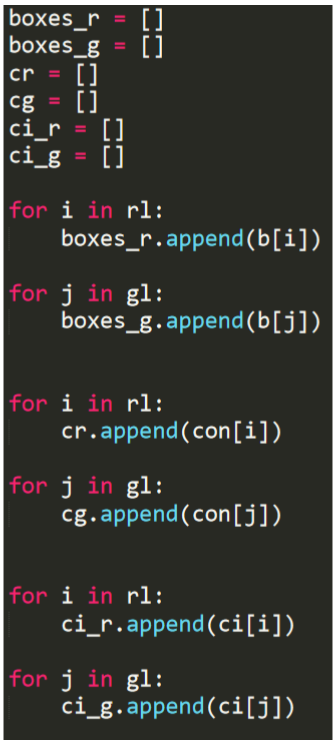
Formula used: $d=\sqrt{((x_2-x_1)^2+(y_2-y_1)^2)}$



```python
rl = []
gl = []

for (id1, p1), (id2, p2) in combinations(centeres.items(), 2):
    xx = p1[0] - p2[0]
    yy = p1[1] - p2[1]
    dis = math.sqrt(xx**2 + yy**2)
    if dis < 25:
        if id1 not in rl and ci[id1] == 0:
            rl.append(id1)
        if id2 not in rl and ci[id2] == 0:
            rl.append(id2)
    else:
        if id1 not in gl and ci[id1] == 0:
            gl.append(id1)
        if id2 not in gl and ci[id2] == 0:
            gl.append(id2)
```

We will use two different list to store the values of pair of detections that are less than the given pixel apart and vice versa.

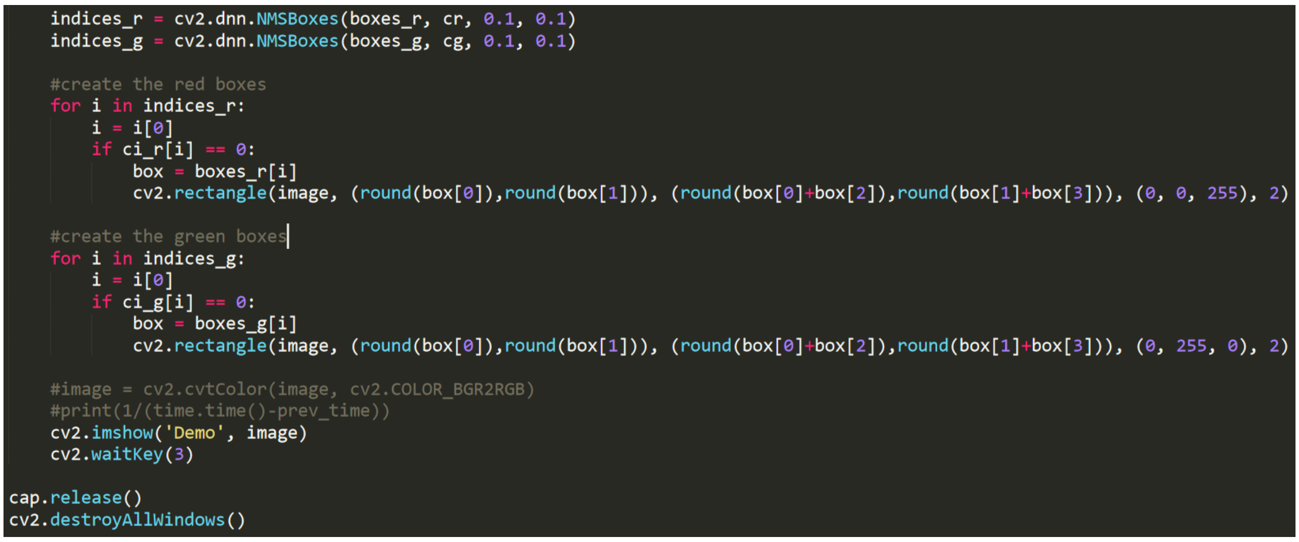
Also, we will separate the confidences and measurements for each detection into two.



```
boxes_r = []
boxes_g = []
cr = []
cg = []
ci_r = []
ci_g = []

for i in rl:
    boxes_r.append(b[i])

for j in gl:
    boxes_g.append(b[j])


for i in rl:
    cr.append(con[i])

for j in gl:
    cg.append(con[j])


for i in rl:
    ci_r.append(ci[i])

for j in gl:
    ci_g.append(ci[j])
```

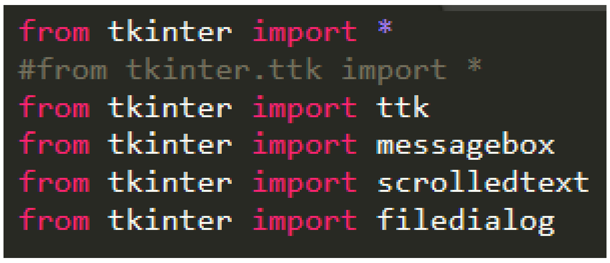
We will use cv2.dnn to classify the object detection boxes into people or not based on confidence and detections.

At last, we will create red boxes around the detections that are less than the given pixels apart and green boxes for the detections that are more than the given pixels apart and release the output video.



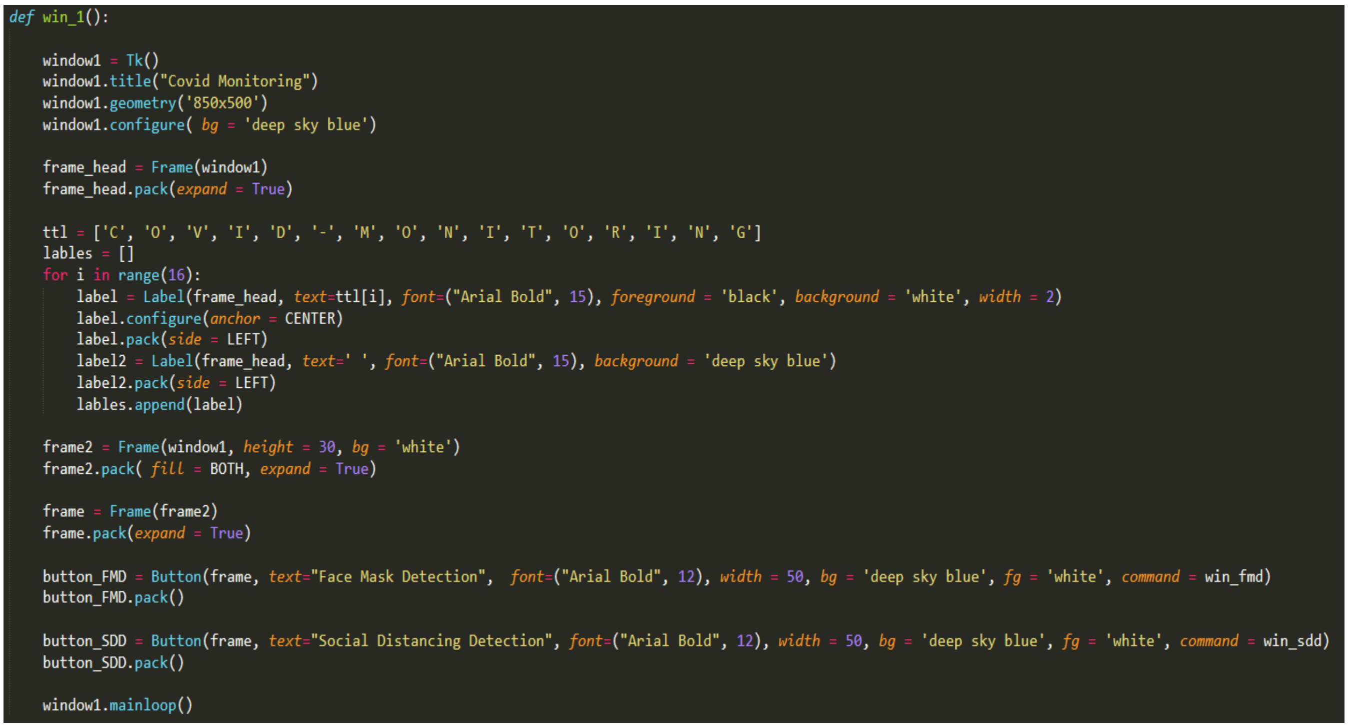
```
    indices_r = cv2.dnn.NMSBoxes(boxes_r, cr, 0.1, 0.1)
    indices_g = cv2.dnn.NMSBoxes(boxes_g, cg, 0.1, 0.1)

    #create the red boxes
    for i in indices_r:
        i = i[0]
        if ci_r[i] == 0:
            box = boxes_r[i]
            cv2.rectangle(image, (round(box[0]),round(box[1])), (round(box[0]+box[2]),round(box[1]+box[3])), (0, 0, 255), 2)

    #create the green boxes
    for i in indices_g:
        i = i[0]
        if ci_g[i] == 0:
            box = boxes_g[i]
            cv2.rectangle(image, (round(box[0]),round(box[1])), (round(box[0]+box[2]),round(box[1]+box[3])), (0, 255, 0), 2)

    #image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    #print(1/(time.time()-prev_time))
    cv2.imshow('Demo', image)
    cv2.waitKey(3)

cap.release()
cv2.destroyAllWindows()
```

The GUI part is created using Tkinter library. First, we import the required modules and libraries.



```python
from tkinter import *
#from tkinter.ttk import *
from tkinter import ttk
from tkinter import messagebox
from tkinter import scrolledtext
from tkinter import filedialog
```

As mentioned above this project is integrated with another project ‘Covid-19 Face Mask Detection’. The UI consist of 3 windows, first is the main window with two buttons as option for selecting the project to use.

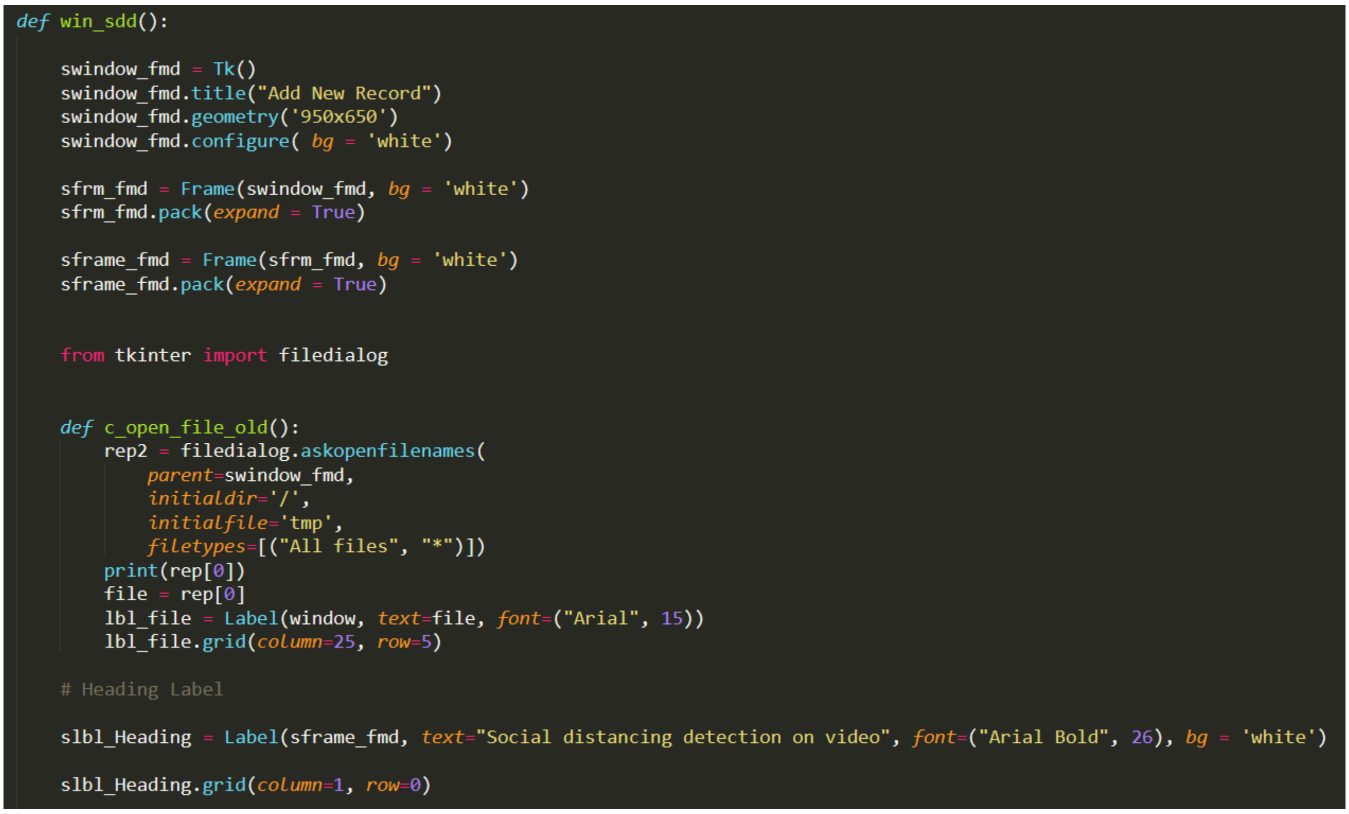


```python
def win_1():

    window1 = Tk()
    window1.title("Covid Monitoring")
    window1.geometry('850x500')
    window1.configure( bg = 'deep sky blue')

    frame_head = Frame(window1)
    frame_head.pack(expand = True)

    ttl = ['C', 'O', 'V', 'I', 'D', '-', 'M', 'O', 'N', 'I', 'T', 'O', 'R', 'I', 'N', 'G']
    lables = []
    for i in range(16):
        label = Label(frame_head, text=ttl[i], font=("Arial Bold", 15), foreground = 'black', background = 'white', width = 2)
        label.configure(anchor = CENTER)
        label.pack(side = LEFT)
        label2 = Label(frame_head, text=' ', font=("Arial Bold", 15), background = 'deep sky blue')
        label2.pack(side = LEFT)
        lables.append(label)

    frame2 = Frame(window1, height = 30, bg = 'white')
    frame2.pack( fill = BOTH, expand = True)

    frame = Frame(frame2)
    frame.pack(expand = True)

    button_FMD = Button(frame, text="Face Mask Detection",  font=("Arial Bold", 12), width = 50, bg = 'deep sky blue', fg = 'white', command = win_fmd)
    button_FMD.pack()

    button_SDD = Button(frame, text="Social Distancing Detection", font=("Arial Bold", 12), width = 50, bg = 'deep sky blue', fg = 'white', command = win_sdd)
    button_SDD.pack()

    window1.mainloop()
```

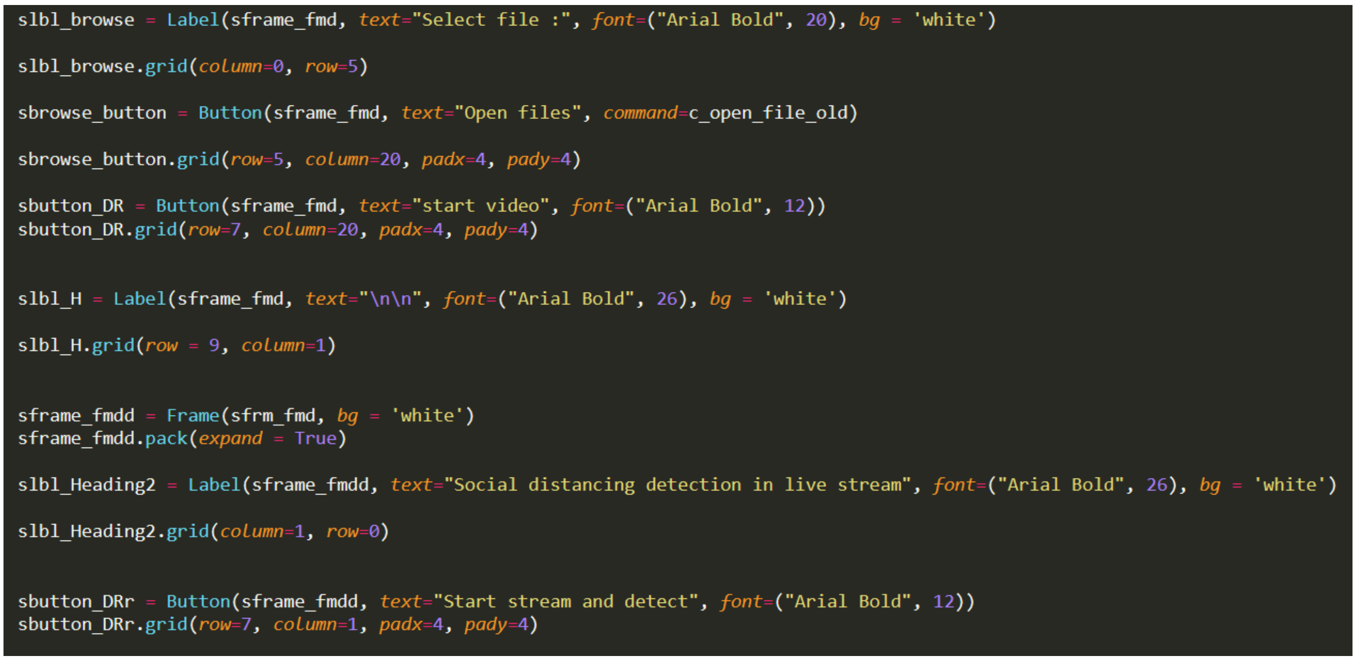
The second window is for the Covid-19 face mask detection.

The third window provides interaction for our Social Distancing Detector. It provides the user with two options, to browse and select a video or go for live video stream using camera.

The code for third window is given below.



```python
def win_sdd():

    swindow_fmd = Tk()
    swindow_fmd.title("Add New Record")
    swindow_fmd.geometry('950x650')
    swindow_fmd.configure( bg = 'white')

    sfrm_fmd = Frame(swindow_fmd, bg = 'white')
    sfrm_fmd.pack(expand = True)

    sframe_fmd = Frame(sfrm_fmd, bg = 'white')
    sframe_fmd.pack(expand = True)


    from tkinter import filedialog


    def c_open_file_old():
        rep2 = filedialog.askopenfilenames(
            parent=swindow_fmd,
            initialdir='/',
            initialfile='tmp',
            filetypes=[("All files", "*")])
        print(rep[0])
        file = rep[0]
        lbl_file = Label(window, text=file, font=("Arial", 15))
        lbl_file.grid(column=25, row=5)

    # Heading Label

    slbl_Heading = Label(sframe_fmd, text="Social distancing detection on video", font=("Arial Bold", 26), bg = 'white')

    slbl_Heading.grid(column=1, row=0)
```



```python
slbl_browse = Label(sframe_fmd, text="Select file :", font=("Arial Bold", 20), bg = 'white')

slbl_browse.grid(column=0, row=5)

sbrowse_button = Button(sframe_fmd, text="Open files", command=c_open_file_old)

sbrowse_button.grid(row=5, column=20, padx=4, pady=4)

sbutton_DR = Button(sframe_fmd, text="start video", font=("Arial Bold", 12))
sbutton_DR.grid(row=7, column=20, padx=4, pady=4)


slbl_H = Label(sframe_fmd, text="\n\n", font=("Arial Bold", 26), bg = 'white')

slbl_H.grid(row = 9, column=1)


sframe_fmdd = Frame(sfrm_fmd, bg = 'white')
sframe_fmdd.pack(expand = True)

slbl_Heading2 = Label(sframe_fmdd, text="Social distancing detection in live stream", font=("Arial Bold", 26), bg = 'white')

slbl_Heading2.grid(column=1, row=0)


sbutton_DRr = Button(sframe_fmdd, text="Start stream and detect", font=("Arial Bold", 12))
sbutton_DRr.grid(row=7, column=1, padx=4, pady=4)
```

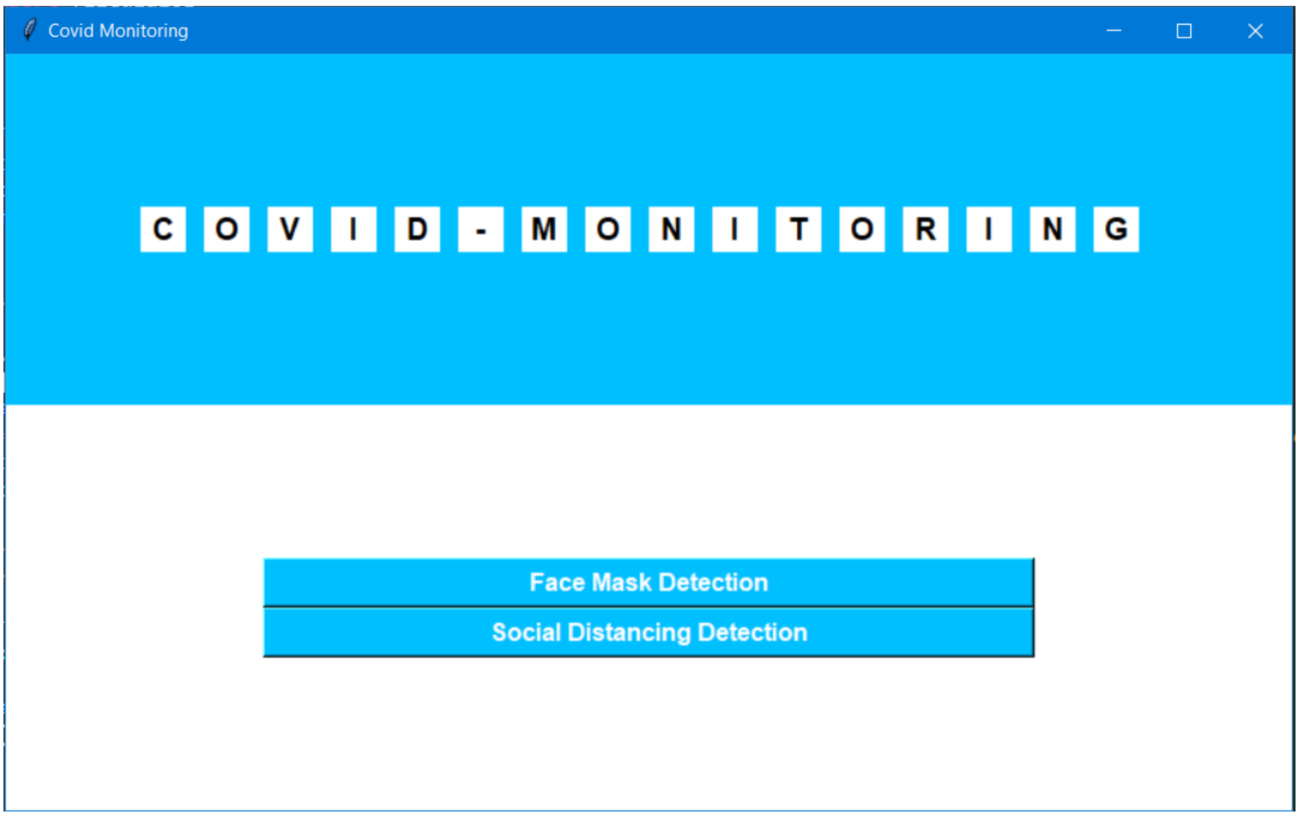
6. RESULT AND DISCUSSIONS

The GUI shows a window giving the option to open another window for performing Social distancing detection in videos. The second window provides the user with two options, browse and select a video and obtain an output video showing the detections and violations of social distancing and it also provides the option of social distancing detection in live video stream using a camera.

The output video shows the people walking on public street. In this work, the video frame is fixed at a specific angle to the ground. The perspective view of the frame is still. We have shown red boxes for the violations of social distancing and green for the ones following social distancing, implemented using pixel value distancing in all frames shown one by one continuously as video output.

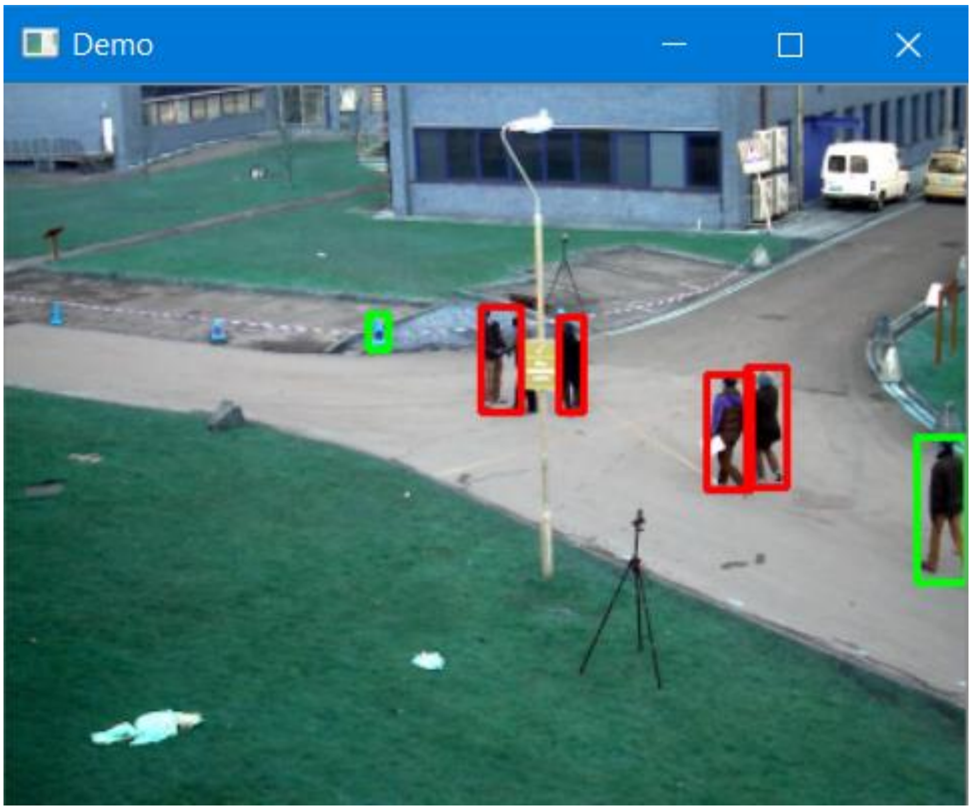
The precision of the distance measurement between pedestrians is also affected by the pedestrian detection algorithm. The YOLO algorithm is also able to detect the half body of the pedestrian as an object by showing the bounding box, the position of the pedestrian corresponds the middle-point of bottom line is estimated based on the bounding box will less precise. Hence, only the people walking in specified spaces and completely in the frame will provide précised results.

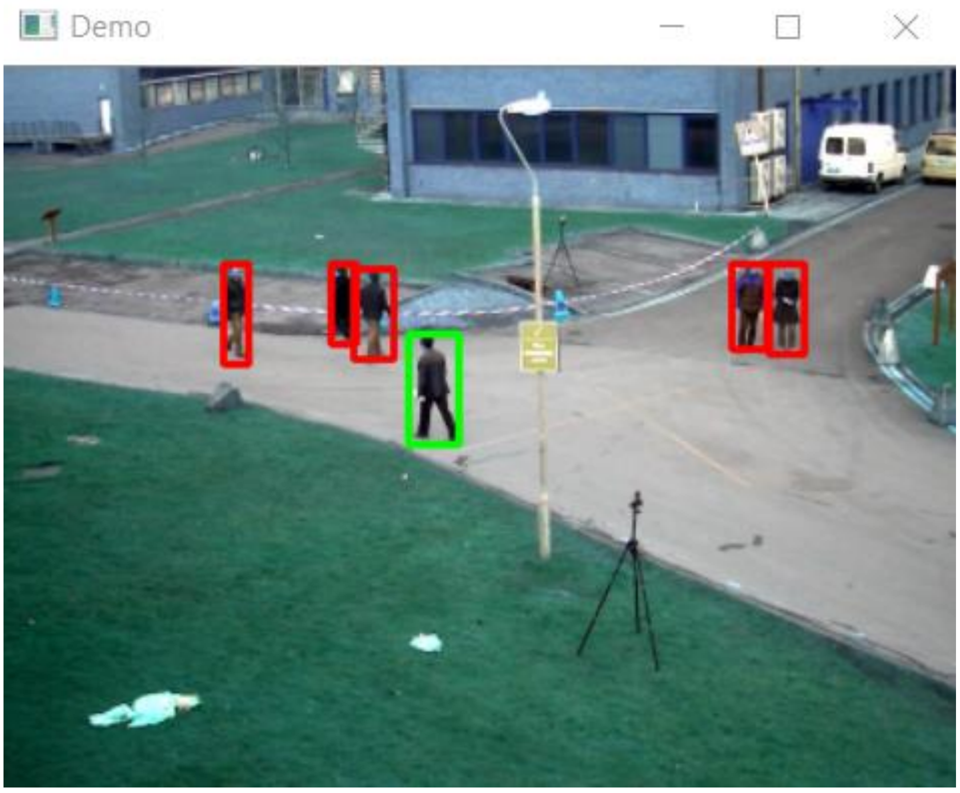
Window -1



Window -3

Social Distancing Detector output for sample video.





7. CONCLUSION

A methodology of social distancing detection tool using a deep learning model is proposed. By using computer vision, the distance between people can be estimated and any noncompliant pair of people will be indicated with a red frame and a red line. The proposed method was validated using a video showing pedestrians walking on a street. The visualization results showed that the proposed method is capable to determine the social distancing measures between people which can be further developed for use in other environment such as office, restaurant, and school.

To create our social distancing detector, earlier we used HOG descriptor for detecting people. OpenCV has a built-in method to detect pedestrians. It has a pre-trained HOG (Histogram of Oriented Gradients) + Linear SVM model to detect people in images and video streams. But the detections were not accurate and the detector was not able to identify more than one or two people. Also, the measurements of box boundary were not up to the mark.

To overcome the problem, we switched to YOLO detector version 3, YOLOV3. YOLO predicts several bounding boxes for each grid cell. In the training stage, it only requires one predictor of the bounding box to be responsible for each class. The predictor is assigned to predict an object which has the highest Intersection over Union value (IoU) for the ground truth. This process leads to specialization within the bounding boxes prediction. Here, we were able to process the entire video on our CPU, and as the results show, our social distancing detector is almost correctly marking people who violate social distancing rules**.**

The problem with this current implementation is speed. Our CPU-based social distancing detector is obtaining less than 2 FPS**,** which is *far too slow* for real-time processing. We can obtain a higher frame processing rate by (1) utilizing an NVIDIA CUDA-capable GPU and (2) compiling/installing OpenCV's "dnn" module with NVIDIA GPU support.

The proposed technique achieved promising results for people detection in terms of evaluation the accuracy and precision of the detector comparable to the other deep learning models. A specific algorithm was implemented on bounding boxes to distinguish between safe and unsafe conditions, respectively, marking as green and red the bounding box for detected persons. The proposed technique showed better results for real-time performance vs other object detectors. The proposed approach can be implemented in a distributed video surveillance system; indeed, it is a suitable solution for the authorities to visualize the compliance of people with social distancing.

7. FUTURE ENHANCEMENTS

As already mentioned earlier, our social distancing detector *did not* leverage a proper camera calibration, meaning that we could not (easily) map distances in pixels to actual measurable units (i.e., meters, feet, etc.). Therefore, the first step to improving our social distancing detector is to utilize a proper camera calibration. Doing so will yield better results and enable you to compute actual measurable units (rather than pixels).

Secondly, you should consider applying a top-down transformation of your viewing angle, as this implementation has done. Applying a perspective transform or using stereo computer vision would allow you to get a more accurate representation of social distancing with OpenCV. While more accurate, the engineering involved in such a system is more complex and isn't always necessary. From there, you can apply the distance calculations to the top-down view of the pedestrians, leading to a better distance approximation.

Our third recommendation is to improve the people detection process. OpenCV's YOLO implementation is quite slow *not* because of the model itself but because of the additional post-processing required by the model. To further speedup the pipeline, consider utilizing a Single Shot Detector (SSD) running on your GPU — that will improve frame throughput rate *considerably.*

Apart from this, we have planned to integrate the Social Distancing Detector project with Face Mask Detector project and combinedly make a 'COVID Monitoring' project for overall evaluation of COVID-19 protocols. The new project aims at monitoring of these protocol in all public places possible.

Nowadays, social distancing along with other basic sanitary measures are very important to keep the spread of the Covid-19 as slow as possible. But this project is only a proof of concept and was not made to be used to monitor social distancing in public or private areas because of ethical and privacy issues.

BIBLIOGRAPHY

https://towardsdatascience.com/a-social-distancing-detector-using-a-tensorflow-object-detection-model-python-and-opencv-4450a431238

https://reader.elsevier.com/reader/sd/pii/S1877050918310652?token=B4889E3CE0FB91DC317F77777FE3B473DA9CC0C558FC2C2B8C687487E3704921BCC429F2FFEE4C298008D87092FAD039&originRegion=eu-west-1&originCreation=20210521120630

https://medium.com/@luanaebio/detecting-people-with-yolo-and-opencv-5c1f9bc6a810

https://www.pyimagesearch.com/2020/06/01/opencv-social-distancing-detector/

https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

https://github.com/AlexeyAB/darknet

https://pjreddie.com/darknet/yolo/

https://link.springer.com/article/10.1007/s11554-021-01070-6#:~:text=Social%20distancing%20detector%20steps&text=Prepare%20the%20thermal%20images%20or,the%20images%20or%20video%20stream.