Mini Project Report on

Real-time Human Detection & Counting(People Counting) Using Python

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by:

Shristy Chaudhary **2019498**

Under the Mentorship of

Dr.Kireet Joshi
Assistant Professor

Department of Computer Science and Engineering
Graphic Era (Deemed to be University)
Dehradun, Uttarakhand
January 2023

Graphic Era Deemed to be University DEHRADUN

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the project report entitled **"Real-time Human Detection & Counting(People Counting) using Python"** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the under the mentorship of **Dr.Kireet Joshi, Assistant Professor**, Department of Computer Science and Engineering, Graphic Era (Deemed to be University), Dehradun.

Shristy Chaudhary	2019498	signature

Table of Contents

Chapter 1

Introduction

Real-time object detection is the task of doing object detection in real-time with fast inference while maintaining a base level of accuracy.

Human detection and tracking are tasks of computer vision systems for locating and following people in video imagery.

Human detection is the task of locating all instances of human beings present in an image, and it has been most widely accomplished by

searching all locations in the image, at all possible scales, and comparing a small area at each location with known templates or patterns

of people.

Human tracking is the process of temporally associating the human detections within a video sequence to generate persistent paths, or trajectories, of the people. Human detection and tracking are generally considered the first two processes in a video surveillance pipeline, and can feed into higher-level reasoning modules such as action recognition and dynamic scene analysis.

1.1 Computer Vision

Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. It is most widely used field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs[1].Different types of computer vision include image segmentation, object detection, facial recognition, edge detection, pattern detection, image classification, and feature matching.Computer Vision itself is a big domain and is divided into various subdomains like scene reconstruction, object detection, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, visual servoing, 3D scene modeling, and image restoration.[2]

Fig. 1.1

1.2 Human Detection

Human detection[2] is the task of locating all instances of human beings present in an image, and it has been most widely accomplished by searching all locations in the image, at all possible scales, and comparing a small area at each location with known templates or patterns of people.

In this we can use various predefined methods and can detect the human in any image, video and can even get various factors like accuracy, each detections counting, etc.

Some common methods are :

❖ **Using Haar Cascade Classifier[2]:**

Here we make use of .xml file for human detection, and using that we detect the humans in real time videos and images. Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones in their paper Rapid Object Detection using a Boosted Cascade of Simple Features .Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

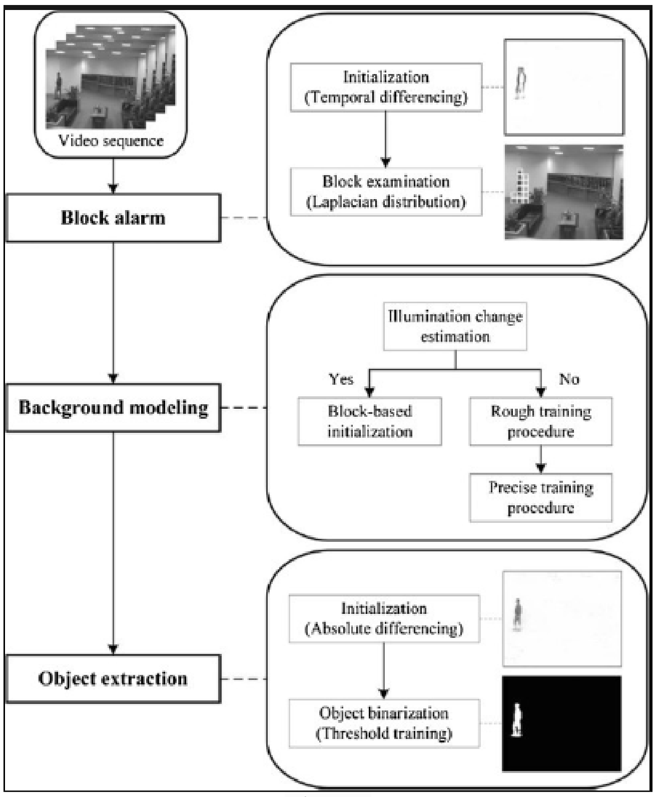

Video sequence
Block alarm
Background modeling
Object extraction
Initialization (Temporal differencing)
Block examination (Laplacian distribution)
Illumination change estimation
Yes
No
Block-based initialization
Rough training procedure
Precise training procedure
Initialization (Absolute differencing)
Object binarization (Threshold training)

Fig. 1.2

Positive images – These images contain the images which we want our classifier to identify.

Negative Images – Images of everything else, which do not contain the object we want to detect.

- **Using HOG(Histogram of Oriented Gradients)[2]** :

  HOG is a feature descriptor used in computer vision and image processing for the purpose of object detection. This is one of the most popular techniques for object detection, to our fortune, OpenCV has already been implemented in an efficient way to combine the HOG Descriptor algorithm with Support Vector Machine or SVM.

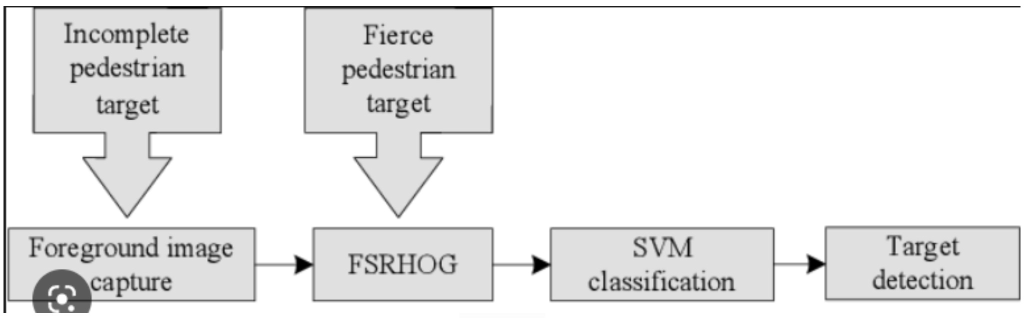


Fig. 1.3

Here we make used of predefined functions and with that we detect, and this case gives some how better accuracy as compared to Harr Cascade Classifier.

❖ Using Tensorflow[3]:

TensorFlow is an open-source API from Google, which is widely used for solving machine learning tasks that involve Deep Neural Networks. And again this method gives even better accuracy than above two methods.

The single biggest benefit TensorFlow provides for machine learning development is *abstraction*. Instead of dealing with the nitty-gritty details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall application logic. TensorFlow takes care of the details behind the scenes.

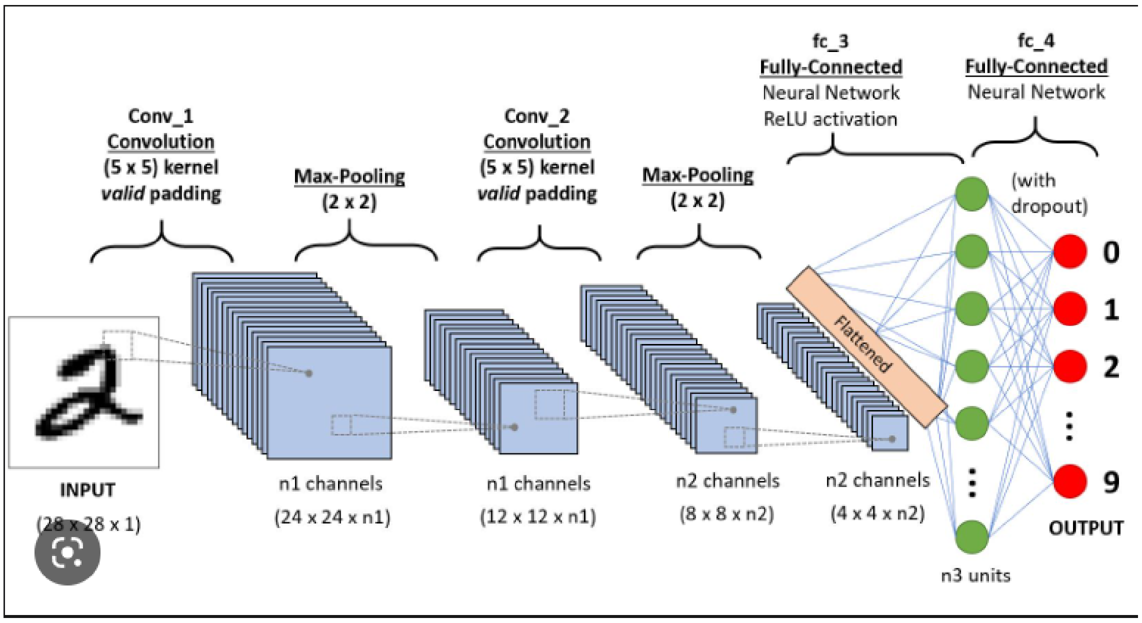


Fig. 1.4

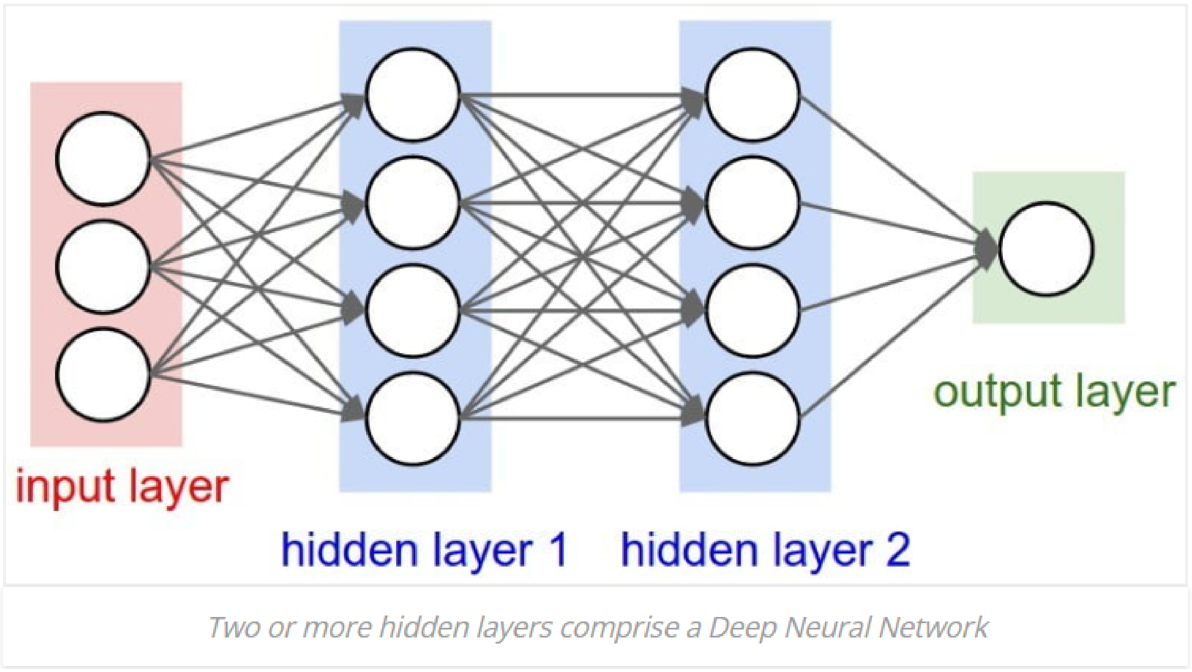


Fig. 1.5

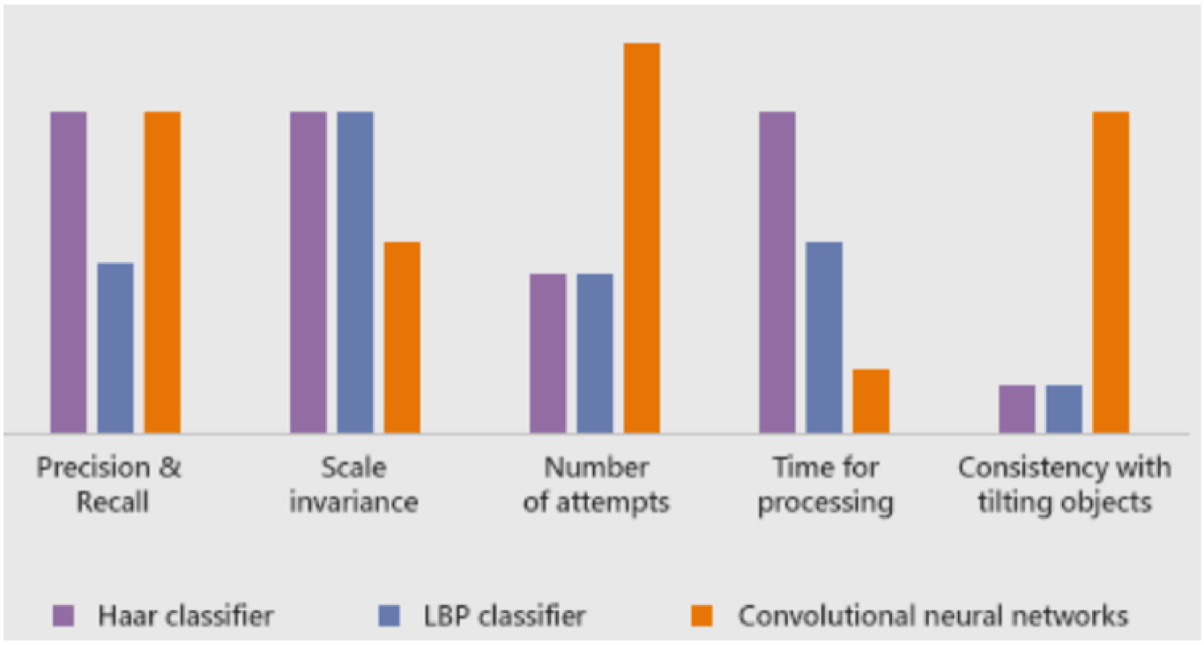


Fig. 1.6

Deep neural networks offer a lot of value to statisticians, particularly in increasing accuracy of a machine learning model.

The deep net component of a ML model is really what got A.I. from generating cat images to creating art. At its simplest, a neural network with some level of complexity, usually at least two layers, qualifies as a deep neural network (DNN), or deep net for short. Deep nets

process data in complex ways by employing sophisticated math modeling.To truly understand deep neural networks, however, it’s best to see it as an evolution. A few items had to be built before
deep nets existed.

Chapter 2

Literature Survey

The script use Tkinter to create a GUI for a real-time human detection and counting application. The script creates a main window with a "START" button that, when clicked, opens a new window with options for detecting humans in either a video, image, or live camera feed. The script also uses the DetectorAPI class from the persondetection library to detect humans in the provided video or image. Additionally, it uses the FPDF library to create a PDF report of the human detection results. The script also provides an exit button to close the window and the program.

2.1 Latest research work

There are several recent research works on real-time human detection and counting, some of them are:

- "Real-time Multi-person Pose Estimation using Part Affinity Fields" by Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh, this work proposes a real-time method to simultaneously detect multiple people in an image and estimate their poses.
- "Real-time Crowd Counting via Adversarial Cross-Scale Consistency Pursuit" by Jiasen Lu, Xiaodan Liang, Jiayuan Gu, Zhiqiang Shen, Jian Dong, and Shu Liu, this work proposes a new real-time crowd counting method that uses an adversarial learning mechanism to improve counting accuracy.
- "Real-Time Human Detection and Tracking for Augmented Reality" by Jie Chen, Xinggang Wang, Wenyu Liu, and Xiaokang Yang, this work proposes a real-time human detection and tracking method for augmented reality applications.
- "Real-time Multi-person 2D Pose Estimation using Part Affinity Fields" by Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh, this work extends previous work on single-person pose estimation to multiple people by proposing a real-time method to simultaneously detect multiple people in an image and estimate their poses.

These are just a few examples, and new research is constantly being published in this field. It's worth noting that these papers may not be directly related to your specific implementation, but rather a general overview of recent research in the field of real-time human detection and counting.

- "Real-time Multi-person Pose Estimation using Part Affinity Fields" by Insafutdinov et al. (2016).
- "Real-time Crowd Counting with Extremely Low Resolution People" by Zhang et al. (2016).
- "Real-time and Accurate Crowd Counting via Adversarial Cross-Scale Consistency Pursuit" by Wang et al. (2019).
- Real-Time Human Detection and Counting System Using Deep Learning Computer Vision Techniques Hamam Mokayed1, Tee Zhen Quan2, Lama Alkhaled1 and V. Sivakumar(2022).

Chapter 3

Methodology

3.1 Libraries Used

3.1.1 Open CV

A strong library used for machine learning.

3.1.2 Imutils

To Image Processing.

3.1.3 Numpy

Used for Scientific Computing. Image is stored in a numpy array.

3.1.4 Argparse

Used to give input in command line.

3.1.5 Tkinter

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Import the Tkinter module.

3.2 Methodology of Main Function

- This code appears to be using the Tkinter library to create a GUI for a human detection and counting application. The GUI has a start button that, when clicked, closes the initial window and opens a new window, window1.
- The code then uses the argparse library to define command line arguments for the application, such as specifying a video or image file to use, or using the camera instead. The code also includes an optional output video file path argument.
- It also uses the library persondetection, FPDF, matplotlib, PIL and cv2.
- The code also uses the exit_win function to close the application when the exit button is pressed, and exit1 is set to False. It appears that the main function of the application is in the window1, but it is not specified in the code.
- It also uses some image and icons and also it uses some font and color for the GUI.

3.3 Methodology of Detection & Counting through Image

- This code appears to be using Tkinter to create a new window, windowi, that is specifically for the detection of humans in an image. It has a button to open an image file, and another button to detect humans in the selected image.
- The code then defines several global variables, such as max_count1, framex1, county1, max1, avg_acc1_list, max_avg_acc1_list, max_acc1 and max_avg_acc1, which will be used to store information about the detection process.
- The detectByPathImage function is the main function for detecting humans in an image. It first calls two other functions, img_enumeration_plot and img_accuracy_plot, which are used to plot the number of humans detected over time, and the average accuracy of the detection, respectively.
- The function then appears to read the image from the specified file path, process the image and detect any humans present in the image. It also uses the library matplotlib for creating the plots.

3.4 Methodology of Detection & Counting through Video

- This code appears to be using Tkinter to create a new window, windowv, that is specifically for the detection of humans in a video. It has a button to open a video file, and another button to detect humans in the selected video.
- The code then defines several global variables, such as max_count2, framex2, county2, max2, avg_acc2_list, max_avg_acc2_list, max_acc2 and max_avg_acc2, which will be used to store information about the detection process.
- The detectByPathVideo function is the main function for detecting humans in a video. It first calls two other functions, vid_enumeration_plot and vid_accuracy_plot, which are used to plot the number of humans detected over time, and the average accuracy of the detection, respectively.
- The function then appears to read the video from the specified file path, process the frames of the video and detect any humans present in the video. It also uses the library cv2 and matplotlib for creating the plots and reading the video file.
- It uses the same arguments as in image detection and also it uses the writer object to write the output video file if specified in the arguments.

3.5 Methodology of Detection & Counting through Camera

- The camera_option function creates a new Tkinter window for the "Human Detection from Camera" section. It sets the window title and icon, and defines the window's size.

- The function also declares several global variables for storing the counts, frames, and accuracy of the detections. The function defines three main functions: open_cam, detectByCamera, and cam_enumeration_plot, cam_accuracy_plot and cam_gen_report.
- The open_cam function is used to open the camera and set the global variables to their initial values.
- The detectByCamera function is used to perform the detection process and call the cam_enumeration_plot, cam_accuracy_plot and cam_gen_report functions.
- The cam_enumeration_plot, cam_accuracy_plot and cam_gen_report functions are used to plot the enumeration, accuracy and generate a report respectively of the human detection from the camera.

3.6 Plots

This section basically deals with the graphical representation of the data[4][5] we got from the detection process. Using this graphical representation, one can do the analysis of the human count and accuracy very well.

In our application, we have basically talked about two basic plots.

- Enumeration Plot
- Avg. Accuracy Plot

3.6.1 Enumeration Plot

This plot basically represent the plot between humans count against each time interval. For this plot, the parameter we took on X-axis is time(in seconds) and on Y-axis, we took, human count at that particular time.[4]

And the highest peak in this enumeration plot, indicates the maximum no. of people detected in whole detection process.

3.6.2 Average Accuracy Plot

This plot basically represent the plot between Average Accuracy against each time interval. For this plot, the parameter we took on X-axis is time(in seconds) and on Y-axis, we took, average accuracy with which humans got detected at that particular time.[4]

And the highest peak in this plot, indicates the maximum avg. accuracy with which people detected in whole detection process.

3.7 Accuracy

Now here we have discussed about the main keypoint of all computer vision project i.e. Accuracy[3]. During the detection process of human, we along with process also kept track of the accuracy with each human is getting detected in image, video and camera.

In our method, we have set the threshold accuracy for the detection process as 70%, so the object detected with accuracy more than the threshold accuracy, we declared it as the well detected human, and display detection indicator around that human during process. We have set this threshold in order to prevent false detection to det displayed while detection process.

Now whenever term accuracy comes, there is always a general question, “What is the maximum accuracy of the detection?” and that we have discussed in the next topic.

3.7.1 Maximum Accuracy

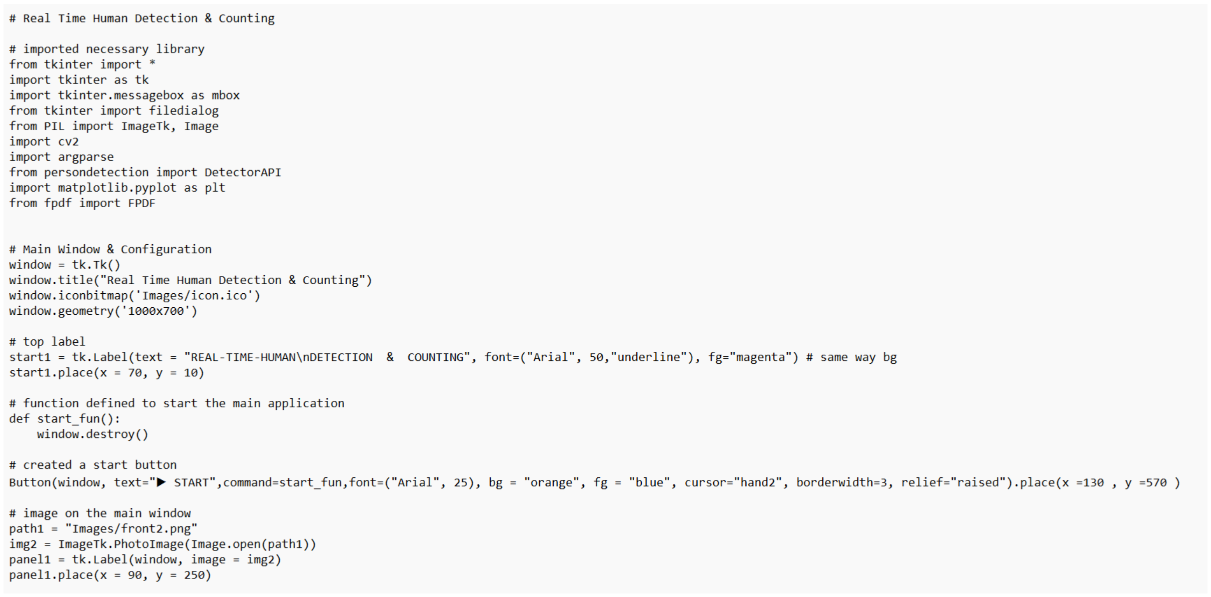
Since we got the factor of accuracy with which each human is getting detected, so we also kept the track of maximum accuracy which we are getting throughout the detection process. This factors basically tells us about the preciseness of our implemented application.

3.7.2 Maximum Average Accuracy

This factor comes in the case of detecting through video and camera. Because in case of image, since the image is static, there is no meaning of maximum avg. accuracy.
In case of video and camera, it basically reflects the maximum of all the average of accuracy that we get for each frames in running
video and webcam.

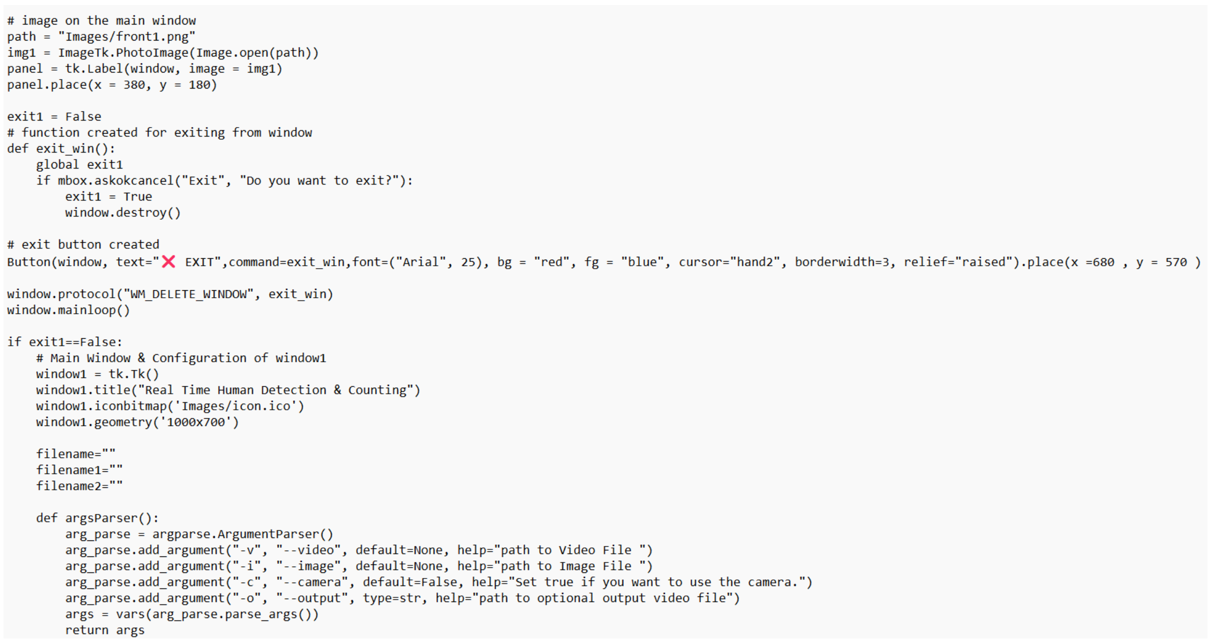
Chapter 4

Result and Discussion



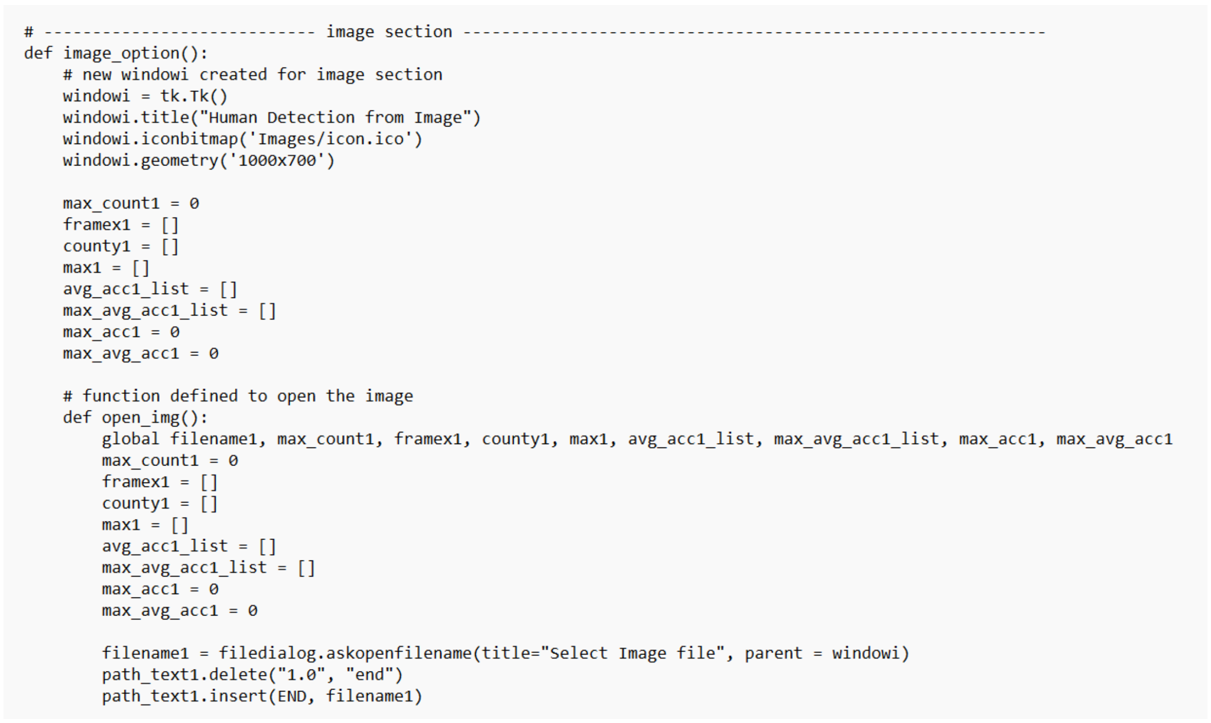
```
# Real Time Human Detection & Counting

# imported necessary library
from tkinter import *
import tkinter as tk
import tkinter.messagebox as mbox
from tkinter import filedialog
from PIL import ImageTk, Image
import cv2
import argparse
from persondetection import DetectorAPI
import matplotlib.pyplot as plt
from fpdf import FPDF


# Main Window & Configuration
window = tk.Tk()
window.title("Real Time Human Detection & Counting")
window.iconbitmap('Images/icon.ico')
window.geometry('1000x700')

# top label
start1 = tk.Label(text = "REAL-TIME-HUMAN\nDETECTION  &  COUNTING", font=("Arial", 50,"underline"), fg="magenta") # same way bg
start1.place(x = 70, y = 10)

# function defined to start the main application
def start_fun():
    window.destroy()

# created a start button
Button(window, text="▶ START",command=start_fun,font=("Arial", 25), bg = "orange", fg = "blue", cursor="hand2", borderwidth=3, relief="raised").place(x =130 , y =570 )

# image on the main window
path1 = "Images/front2.png"
img2 = ImageTk.PhotoImage(Image.open(path1))
panel1 = tk.Label(window, image = img2)
panel1.place(x = 90, y = 250)
```

Fig. 4.1



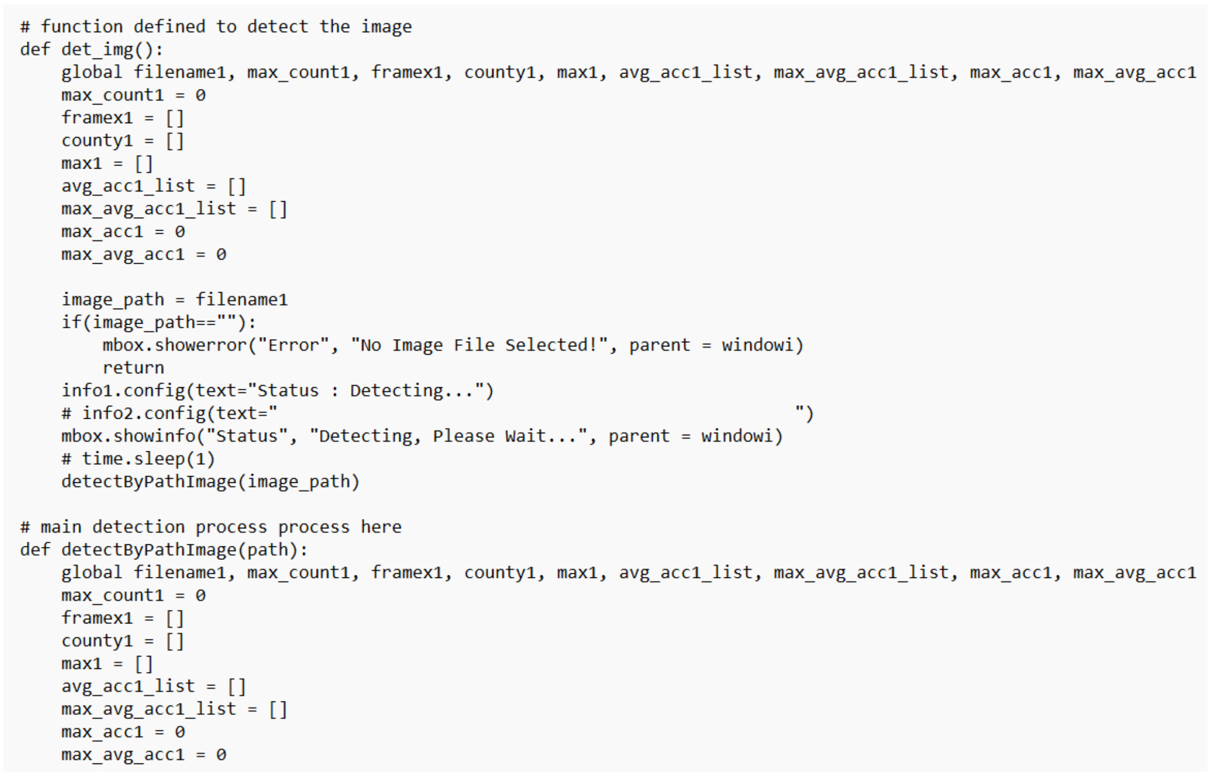
```
# image on the main window
path = "Images/front1.png"
img1 = ImageTk.PhotoImage(Image.open(path))
panel = tk.Label(window, image = img1)
panel.place(x = 380, y = 180)

exit1 = False
# function created for exiting from window
def exit_win():
    global exit1
    if mbox.askokcancel("Exit", "Do you want to exit?"):
        exit1 = True
        window.destroy()

# exit button created
Button(window, text="❌ EXIT",command=exit_win,font=("Arial", 25), bg = "red", fg = "blue", cursor="hand2", borderwidth=3, relief="raised").place(x =680 , y = 570 )

window.protocol("WM_DELETE_WINDOW", exit_win)
window.mainloop()

if exit1==False:
    # Main Window & Configuration of window1
    window1 = tk.Tk()
    window1.title("Real Time Human Detection & Counting")
    window1.iconbitmap('Images/icon.ico')
    window1.geometry('1000x700')

    filename=""
    filename1=""
    filename2=""

    def argsParser():
        arg_parse = argparse.ArgumentParser()
        arg_parse.add_argument("-v", "--video", default=None, help="path to Video File ")
        arg_parse.add_argument("-i", "--image", default=None, help="path to Image File ")
        arg_parse.add_argument("-c", "--camera", default=False, help="Set true if you want to use the camera.")
        arg_parse.add_argument("-o", "--output", type=str, help="path to optional output video file")
        args = vars(arg_parse.parse_args())
        return args
```

Fig. 4.2



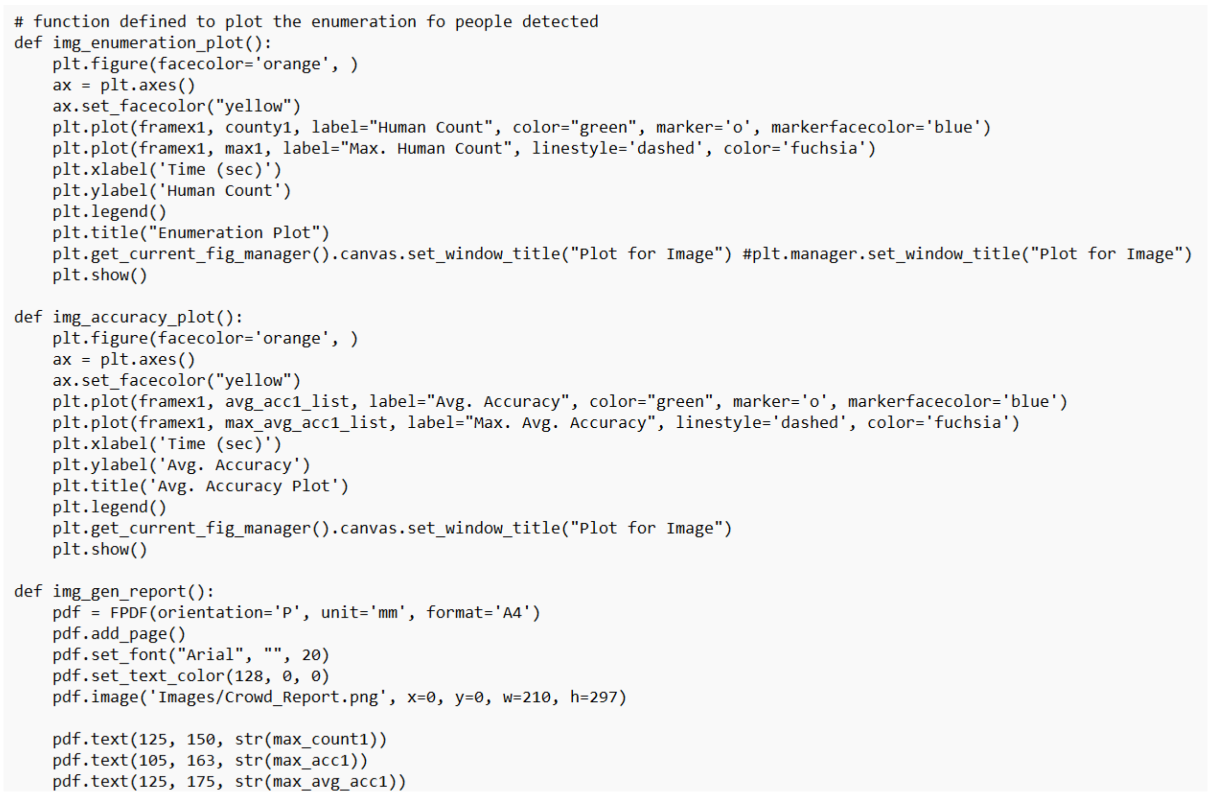
```
# ---------------------------- image section ------------------------------------------------------------
def image_option():
    # new windowi created for image section
    windowi = tk.Tk()
    windowi.title("Human Detection from Image")
    windowi.iconbitmap('Images/icon.ico')
    windowi.geometry('1000x700')

    max_count1 = 0
    framex1 = []
    county1 = []
    max1 = []
    avg_acc1_list = []
    max_avg_acc1_list = []
    max_acc1 = 0
    max_avg_acc1 = 0

    # function defined to open the image
    def open_img():
        global filename1, max_count1, framex1, county1, max1, avg_acc1_list, max_avg_acc1_list, max_acc1, max_avg_acc1
        max_count1 = 0
        framex1 = []
        county1 = []
        max1 = []
        avg_acc1_list = []
        max_avg_acc1_list = []
        max_acc1 = 0
        max_avg_acc1 = 0

        filename1 = filedialog.askopenfilename(title="Select Image file", parent = windowi)
        path_text1.delete("1.0", "end")
        path_text1.insert(END, filename1)
```

Fig. 4.3



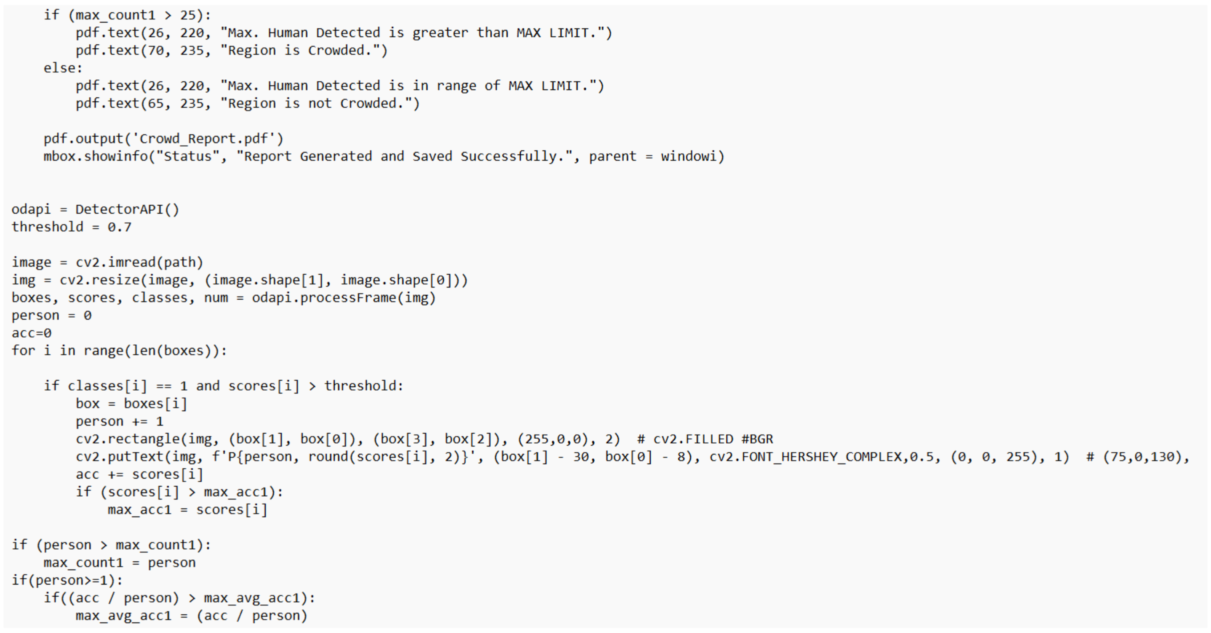
```
# function defined to detect the image
def det_img():
    global filename1, max_count1, framex1, county1, max1, avg_acc1_list, max_avg_acc1_list, max_acc1, max_avg_acc1
    max_count1 = 0
    framex1 = []
    county1 = []
    max1 = []
    avg_acc1_list = []
    max_avg_acc1_list = []
    max_acc1 = 0
    max_avg_acc1 = 0

    image_path = filename1
    if(image_path==""):
        mbox.showerror("Error", "No Image File Selected!", parent = windowi)
        return
    info1.config(text="Status : Detecting...")
    # info2.config(text="                                                  ")
    mbox.showinfo("Status", "Detecting, Please Wait...", parent = windowi)
    # time.sleep(1)
    detectByPathImage(image_path)

# main detection process process here
def detectByPathImage(path):
    global filename1, max_count1, framex1, county1, max1, avg_acc1_list, max_avg_acc1_list, max_acc1, max_avg_acc1
    max_count1 = 0
    framex1 = []
    county1 = []
    max1 = []
    avg_acc1_list = []
    max_avg_acc1_list = []
    max_acc1 = 0
    max_avg_acc1 = 0
```

Fig. 4.4



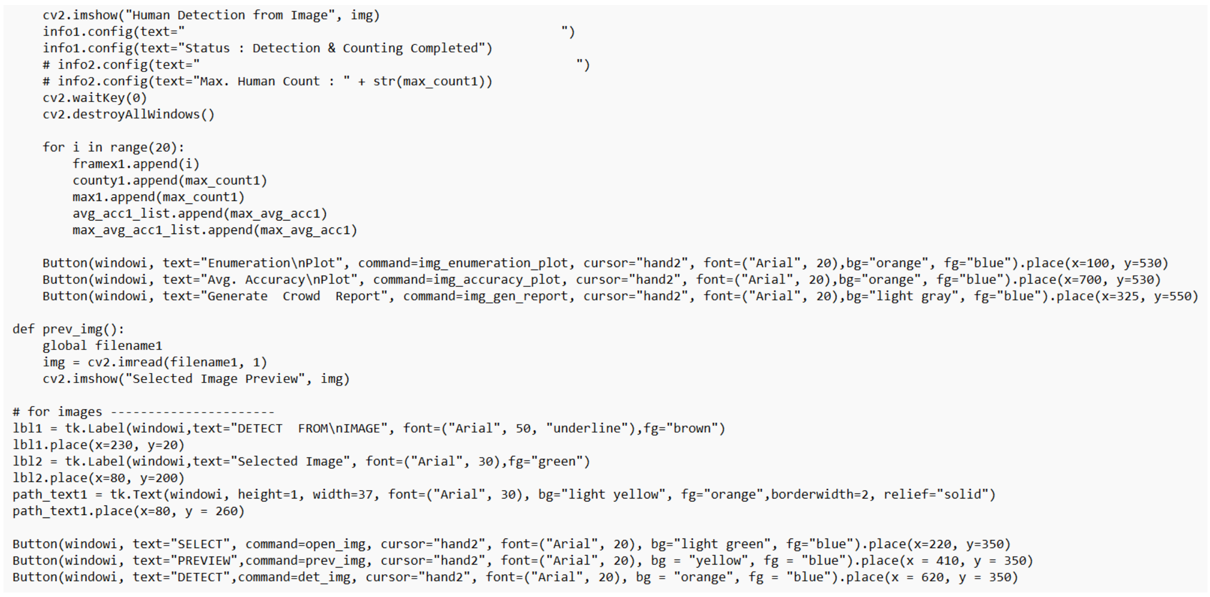
```
# function defined to plot the enumeration fo people detected
def img_enumeration_plot():
    plt.figure(facecolor='orange', )
    ax = plt.axes()
    ax.set_facecolor("yellow")
    plt.plot(framex1, county1, label="Human Count", color="green", marker='o', markerfacecolor='blue')
    plt.plot(framex1, max1, label="Max. Human Count", linestyle='dashed', color='fuchsia')
    plt.xlabel('Time (sec)')
    plt.ylabel('Human Count')
    plt.legend()
    plt.title("Enumeration Plot")
    plt.get_current_fig_manager().canvas.set_window_title("Plot for Image") #plt.manager.set_window_title("Plot for Image")
    plt.show()

def img_accuracy_plot():
    plt.figure(facecolor='orange', )
    ax = plt.axes()
    ax.set_facecolor("yellow")
    plt.plot(framex1, avg_acc1_list, label="Avg. Accuracy", color="green", marker='o', markerfacecolor='blue')
    plt.plot(framex1, max_avg_acc1_list, label="Max. Avg. Accuracy", linestyle='dashed', color='fuchsia')
    plt.xlabel('Time (sec)')
    plt.ylabel('Avg. Accuracy')
    plt.title('Avg. Accuracy Plot')
    plt.legend()
    plt.get_current_fig_manager().canvas.set_window_title("Plot for Image")
    plt.show()

def img_gen_report():
    pdf = FPDF(orientation='P', unit='mm', format='A4')
    pdf.add_page()
    pdf.set_font("Arial", "", 20)
    pdf.set_text_color(128, 0, 0)
    pdf.image('Images/Crowd_Report.png', x=0, y=0, w=210, h=297)

    pdf.text(125, 150, str(max_count1))
    pdf.text(105, 163, str(max_acc1))
    pdf.text(125, 175, str(max_avg_acc1))
```

Fig. 4.5



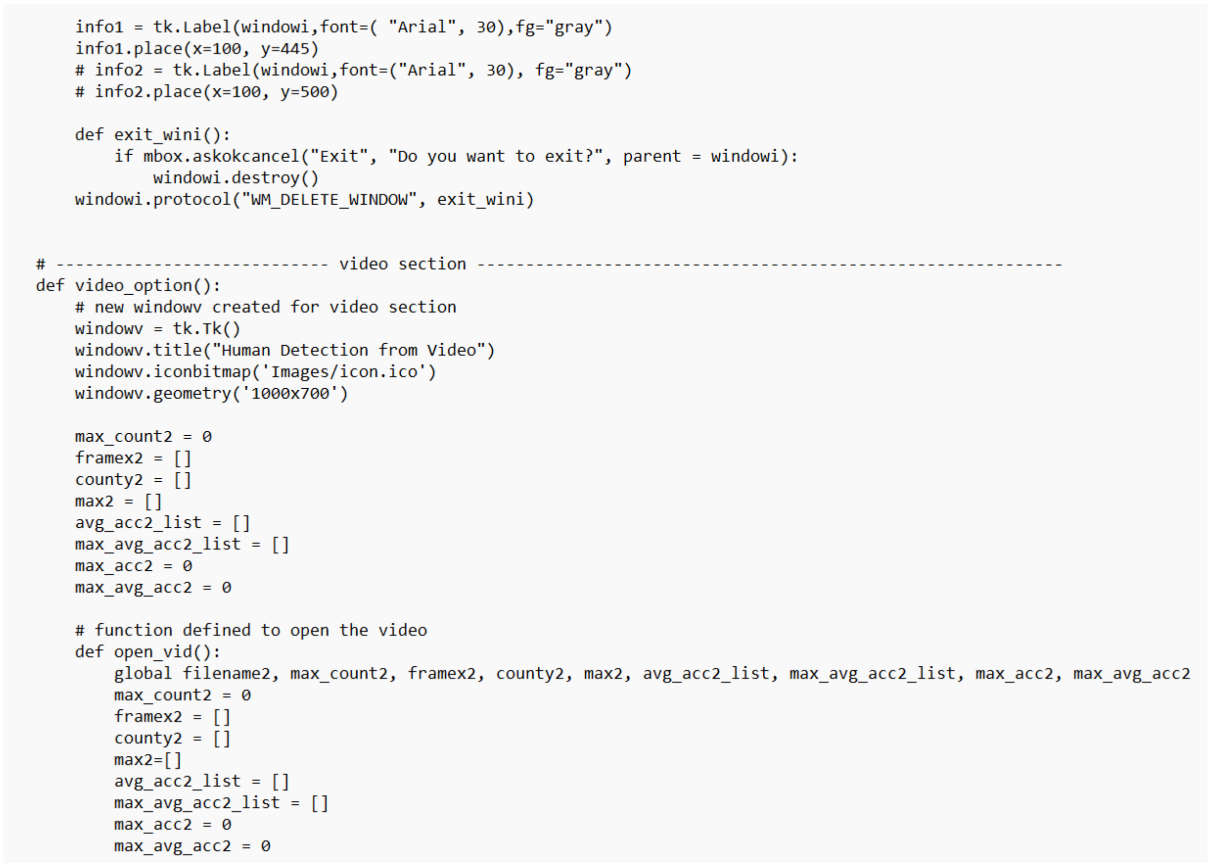
```
    if (max_count1 > 25):
        pdf.text(26, 220, "Max. Human Detected is greater than MAX LIMIT.")
        pdf.text(70, 235, "Region is Crowded.")
    else:
        pdf.text(26, 220, "Max. Human Detected is in range of MAX LIMIT.")
        pdf.text(65, 235, "Region is not Crowded.")

    pdf.output('Crowd_Report.pdf')
    mbox.showinfo("Status", "Report Generated and Saved Successfully.", parent = windowi)


odapi = DetectorAPI()
threshold = 0.7

image = cv2.imread(path)
img = cv2.resize(image, (image.shape[1], image.shape[0]))
boxes, scores, classes, num = odapi.processFrame(img)
person = 0
acc=0
for i in range(len(boxes)):

    if classes[i] == 1 and scores[i] > threshold:
        box = boxes[i]
        person += 1
        cv2.rectangle(img, (box[1], box[0]), (box[3], box[2]), (255,0,0), 2)  # cv2.FILLED #BGR
        cv2.putText(img, f'P{person, round(scores[i], 2)}', (box[1] - 30, box[0] - 8), cv2.FONT_HERSHEY_COMPLEX,0.5, (0, 0, 255), 1)  # (75,0,130),
        acc += scores[i]
        if (scores[i] > max_acc1):
            max_acc1 = scores[i]

if (person > max_count1):
    max_count1 = person
if(person>=1):
    if((acc / person) > max_avg_acc1):
        max_avg_acc1 = (acc / person)
```

Fig. 4.6



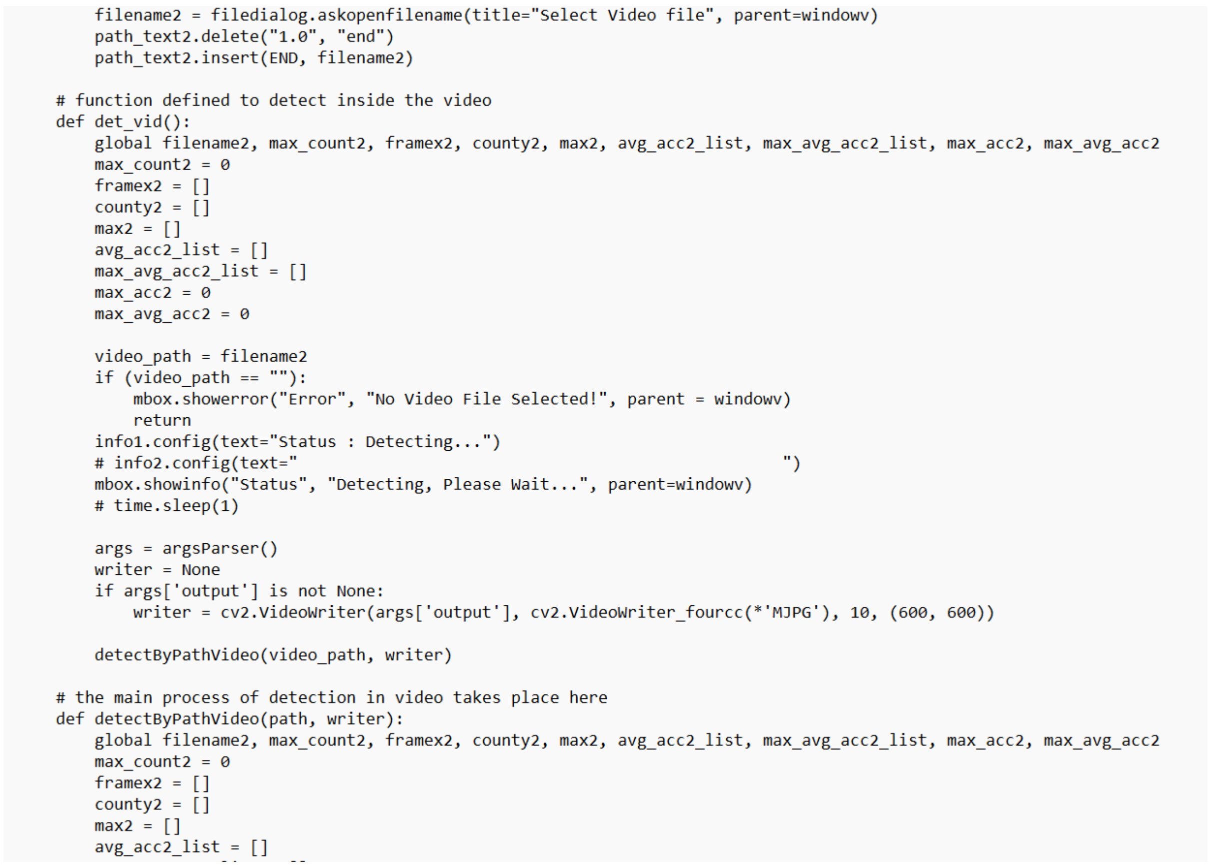
```
    cv2.imshow("Human Detection from Image", img)
    info1.config(text="                                                  ")
    info1.config(text="Status : Detection & Counting Completed")
    # info2.config(text="                                                  ")
    # info2.config(text="Max. Human Count : " + str(max_count1))
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    for i in range(20):
        framex1.append(i)
        county1.append(max_count1)
        max1.append(max_count1)
        avg_acc1_list.append(max_avg_acc1)
        max_avg_acc1_list.append(max_avg_acc1)

    Button(windowi, text="Enumeration\nPlot", command=img_enumeration_plot, cursor="hand2", font=("Arial", 20),bg="orange", fg="blue").place(x=100, y=530)
    Button(windowi, text="Avg. Accuracy\nPlot", command=img_accuracy_plot, cursor="hand2", font=("Arial", 20),bg="orange", fg="blue").place(x=700, y=530)
    Button(windowi, text="Generate  Crowd  Report", command=img_gen_report, cursor="hand2", font=("Arial", 20),bg="light gray", fg="blue").place(x=325, y=550)

def prev_img():
    global filename1
    img = cv2.imread(filename1, 1)
    cv2.imshow("Selected Image Preview", img)

# for images ----------------------
lbl1 = tk.Label(windowi,text="DETECT  FROM\nIMAGE", font=("Arial", 50, "underline"),fg="brown")
lbl1.place(x=230, y=20)
lbl2 = tk.Label(windowi,text="Selected Image", font=("Arial", 30),fg="green")
lbl2.place(x=80, y=200)
path_text1 = tk.Text(windowi, height=1, width=37, font=("Arial", 30), bg="light yellow", fg="orange",borderwidth=2, relief="solid")
path_text1.place(x=80, y = 260)

Button(windowi, text="SELECT", command=open_img, cursor="hand2", font=("Arial", 20), bg="light green", fg="blue").place(x=220, y=350)
Button(windowi, text="PREVIEW",command=prev_img, cursor="hand2", font=("Arial", 20), bg = "yellow", fg = "blue").place(x = 410, y = 350)
Button(windowi, text="DETECT",command=det_img, cursor="hand2", font=("Arial", 20), bg = "orange", fg = "blue").place(x = 620, y = 350)
```

Fig.4.7



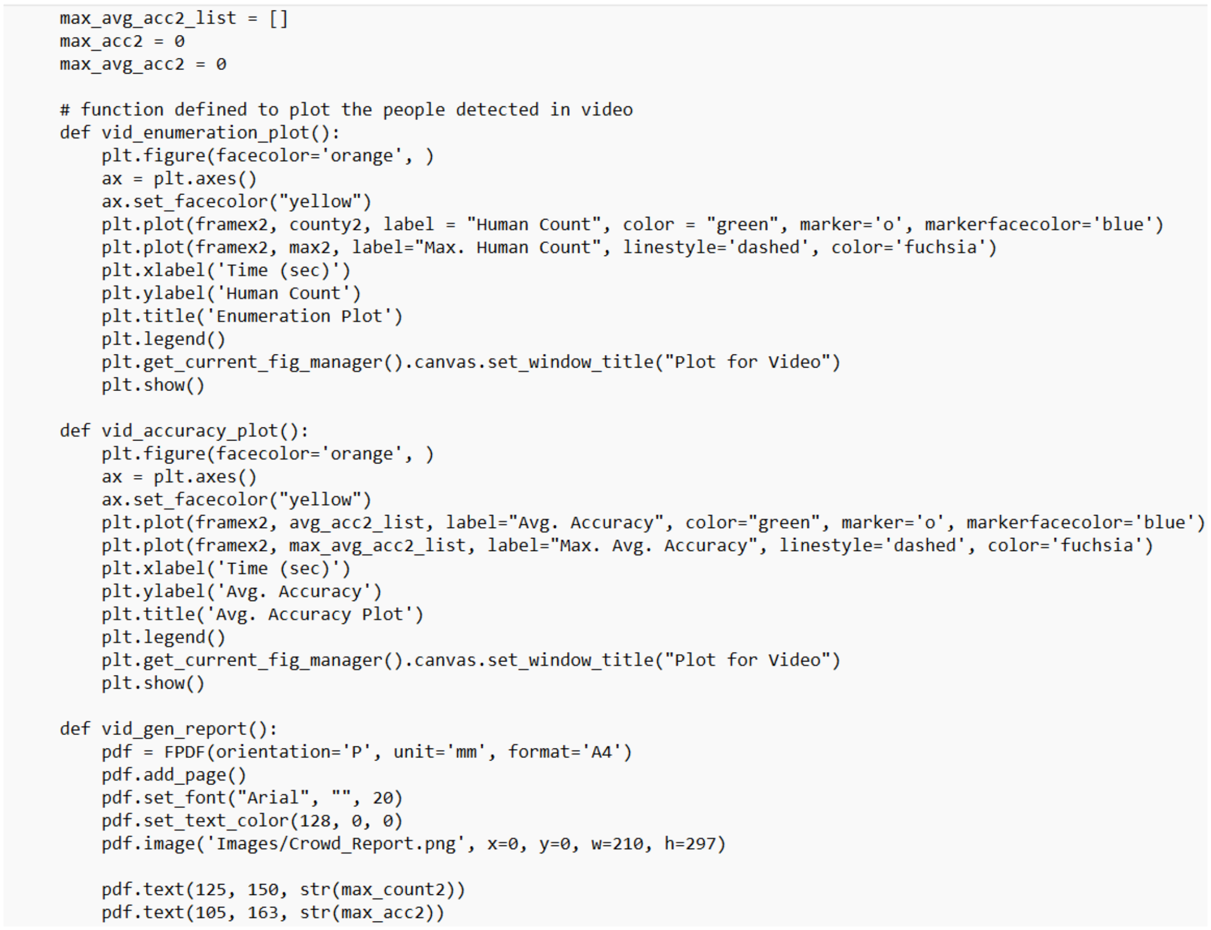
```
        info1 = tk.Label(windowi,font=( "Arial", 30),fg="gray")
        info1.place(x=100, y=445)
        # info2 = tk.Label(windowi,font=("Arial", 30), fg="gray")
        # info2.place(x=100, y=500)

        def exit_wini():
            if mbox.askokcancel("Exit", "Do you want to exit?", parent = windowi):
                windowi.destroy()
        windowi.protocol("WM_DELETE_WINDOW", exit_wini)


    # ---------------------------- video section ------------------------------------------------------------
    def video_option():
        # new windowv created for video section
        windowv = tk.Tk()
        windowv.title("Human Detection from Video")
        windowv.iconbitmap('Images/icon.ico')
        windowv.geometry('1000x700')

        max_count2 = 0
        framex2 = []
        county2 = []
        max2 = []
        avg_acc2_list = []
        max_avg_acc2_list = []
        max_acc2 = 0
        max_avg_acc2 = 0

        # function defined to open the video
        def open_vid():
            global filename2, max_count2, framex2, county2, max2, avg_acc2_list, max_avg_acc2_list, max_acc2, max_avg_acc2
            max_count2 = 0
            framex2 = []
            county2 = []
            max2=[]
            avg_acc2_list = []
            max_avg_acc2_list = []
            max_acc2 = 0
            max_avg_acc2 = 0
```

Fig. 4.8



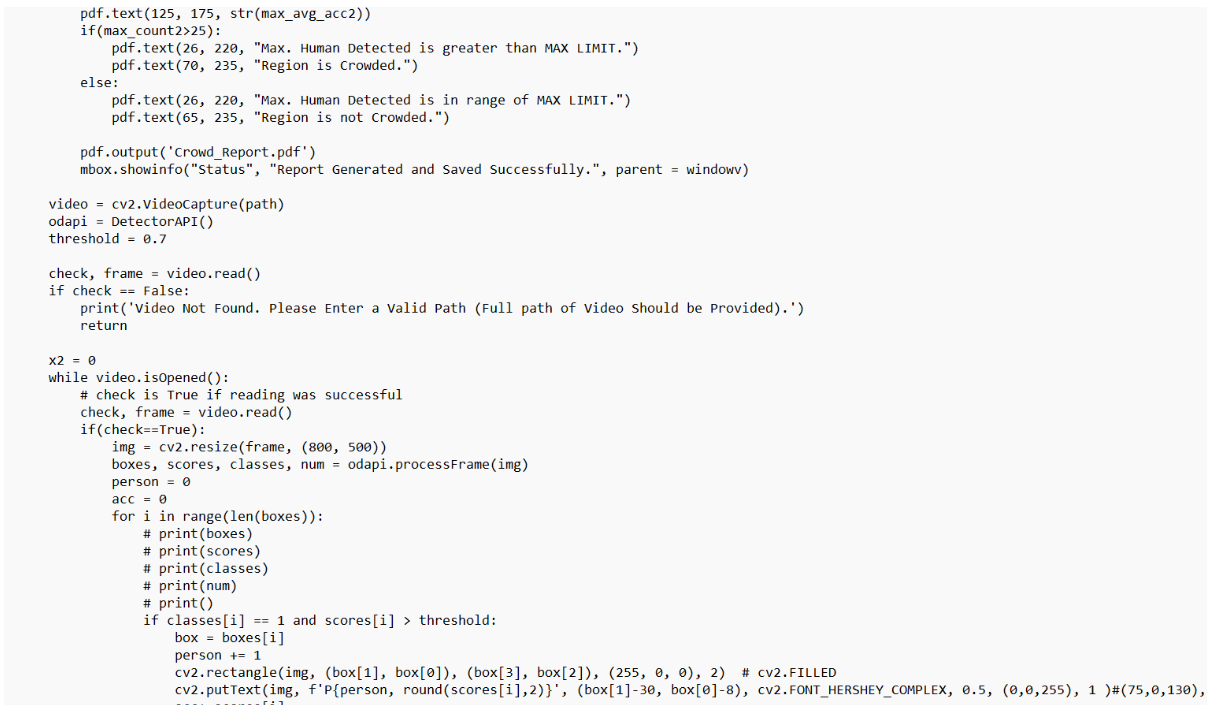
```
    filename2 = filedialog.askopenfilename(title="Select Video file", parent=windowv)
    path_text2.delete("1.0", "end")
    path_text2.insert(END, filename2)

# function defined to detect inside the video
def det_vid():
    global filename2, max_count2, framex2, county2, max2, avg_acc2_list, max_avg_acc2_list, max_acc2, max_avg_acc2
    max_count2 = 0
    framex2 = []
    county2 = []
    max2 = []
    avg_acc2_list = []
    max_avg_acc2_list = []
    max_acc2 = 0
    max_avg_acc2 = 0

    video_path = filename2
    if (video_path == ""):
        mbox.showerror("Error", "No Video File Selected!", parent = windowv)
        return
    info1.config(text="Status : Detecting...")
    # info2.config(text="                                             ")
    mbox.showinfo("Status", "Detecting, Please Wait...", parent=windowv)
    # time.sleep(1)

    args = argsParser()
    writer = None
    if args['output'] is not None:
        writer = cv2.VideoWriter(args['output'], cv2.VideoWriter_fourcc(*'MJPG'), 10, (600, 600))

    detectByPathVideo(video_path, writer)

# the main process of detection in video takes place here
def detectByPathVideo(path, writer):
    global filename2, max_count2, framex2, county2, max2, avg_acc2_list, max_avg_acc2_list, max_acc2, max_avg_acc2
    max_count2 = 0
    framex2 = []
    county2 = []
    max2 = []
    avg_acc2_list = []
```

Fig. 4.9



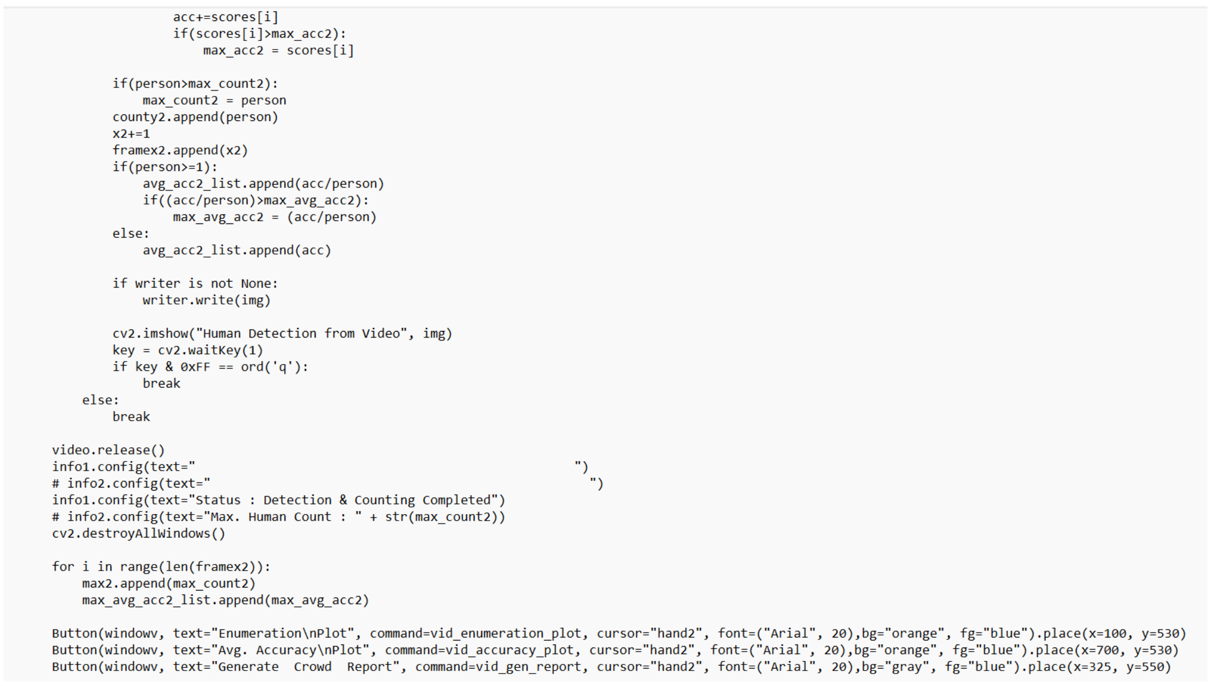
```
    max_avg_acc2_list = []
    max_acc2 = 0
    max_avg_acc2 = 0

# function defined to plot the people detected in video
def vid_enumeration_plot():
    plt.figure(facecolor='orange', )
    ax = plt.axes()
    ax.set_facecolor("yellow")
    plt.plot(framex2, county2, label = "Human Count", color = "green", marker='o', markerfacecolor='blue')
    plt.plot(framex2, max2, label="Max. Human Count", linestyle='dashed', color='fuchsia')
    plt.xlabel('Time (sec)')
    plt.ylabel('Human Count')
    plt.title('Enumeration Plot')
    plt.legend()
    plt.get_current_fig_manager().canvas.set_window_title("Plot for Video")
    plt.show()

def vid_accuracy_plot():
    plt.figure(facecolor='orange', )
    ax = plt.axes()
    ax.set_facecolor("yellow")
    plt.plot(framex2, avg_acc2_list, label="Avg. Accuracy", color="green", marker='o', markerfacecolor='blue')
    plt.plot(framex2, max_avg_acc2_list, label="Max. Avg. Accuracy", linestyle='dashed', color='fuchsia')
    plt.xlabel('Time (sec)')
    plt.ylabel('Avg. Accuracy')
    plt.title('Avg. Accuracy Plot')
    plt.legend()
    plt.get_current_fig_manager().canvas.set_window_title("Plot for Video")
    plt.show()

def vid_gen_report():
    pdf = FPDF(orientation='P', unit='mm', format='A4')
    pdf.add_page()
    pdf.set_font("Arial", "", 20)
    pdf.set_text_color(128, 0, 0)
    pdf.image('Images/Crowd_Report.png', x=0, y=0, w=210, h=297)

    pdf.text(125, 150, str(max_count2))
    pdf.text(105, 163, str(max_acc2))
```

Fig. 4.10



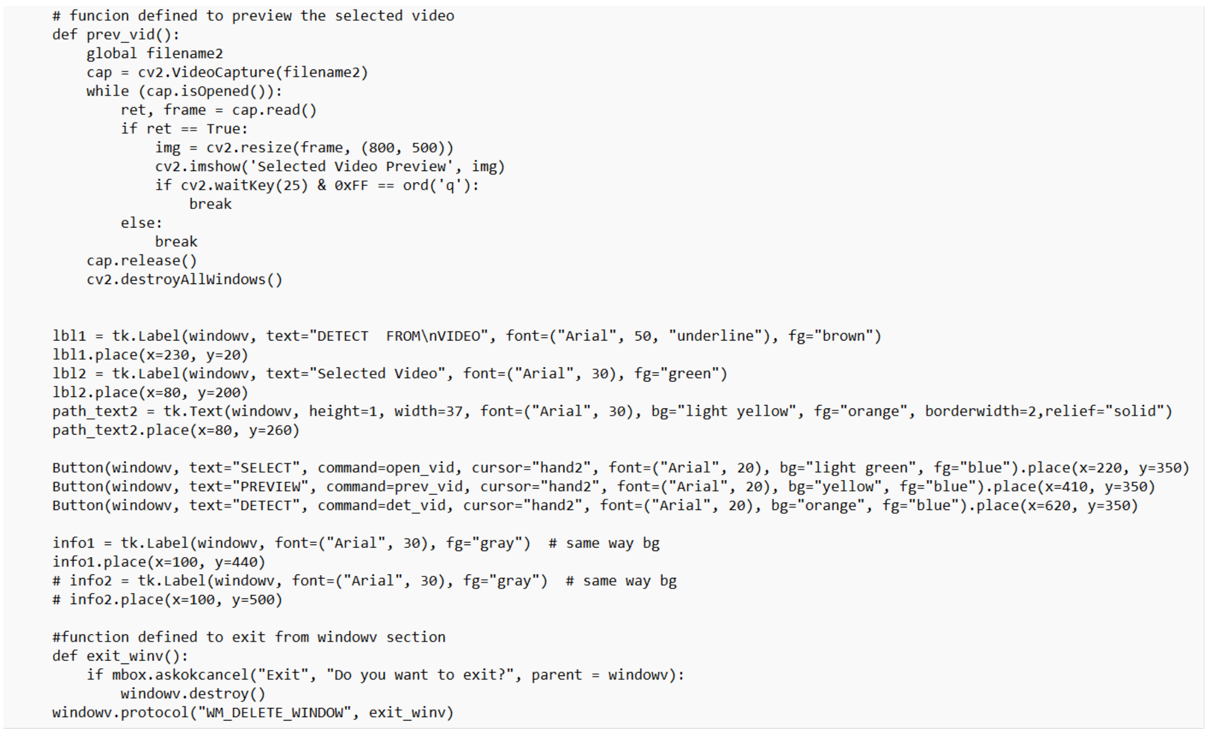
```
    pdf.text(125, 175, str(max_avg_acc2))
    if(max_count2>25):
        pdf.text(26, 220, "Max. Human Detected is greater than MAX LIMIT.")
        pdf.text(70, 235, "Region is Crowded.")
    else:
        pdf.text(26, 220, "Max. Human Detected is in range of MAX LIMIT.")
        pdf.text(65, 235, "Region is not Crowded.")

    pdf.output('Crowd_Report.pdf')
    mbox.showinfo("Status", "Report Generated and Saved Successfully.", parent = windowv)

video = cv2.VideoCapture(path)
odapi = DetectorAPI()
threshold = 0.7

check, frame = video.read()
if check == False:
    print('Video Not Found. Please Enter a Valid Path (Full path of Video Should be Provided).')
    return

x2 = 0
while video.isOpened():
    # check is True if reading was successful
    check, frame = video.read()
    if(check==True):
        img = cv2.resize(frame, (800, 500))
        boxes, scores, classes, num = odapi.processFrame(img)
        person = 0
        acc = 0
        for i in range(len(boxes)):
            # print(boxes)
            # print(scores)
            # print(classes)
            # print(num)
            # print()
            if classes[i] == 1 and scores[i] > threshold:
                box = boxes[i]
                person += 1
                cv2.rectangle(img, (box[1], box[0]), (box[3], box[2]), (255, 0, 0), 2)  # cv2.FILLED
                cv2.putText(img, f'P{person, round(scores[i],2)}', (box[1]-30, box[0]-8), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0,0,255), 1 )#(75,0,130),
```

Fig. 4.11



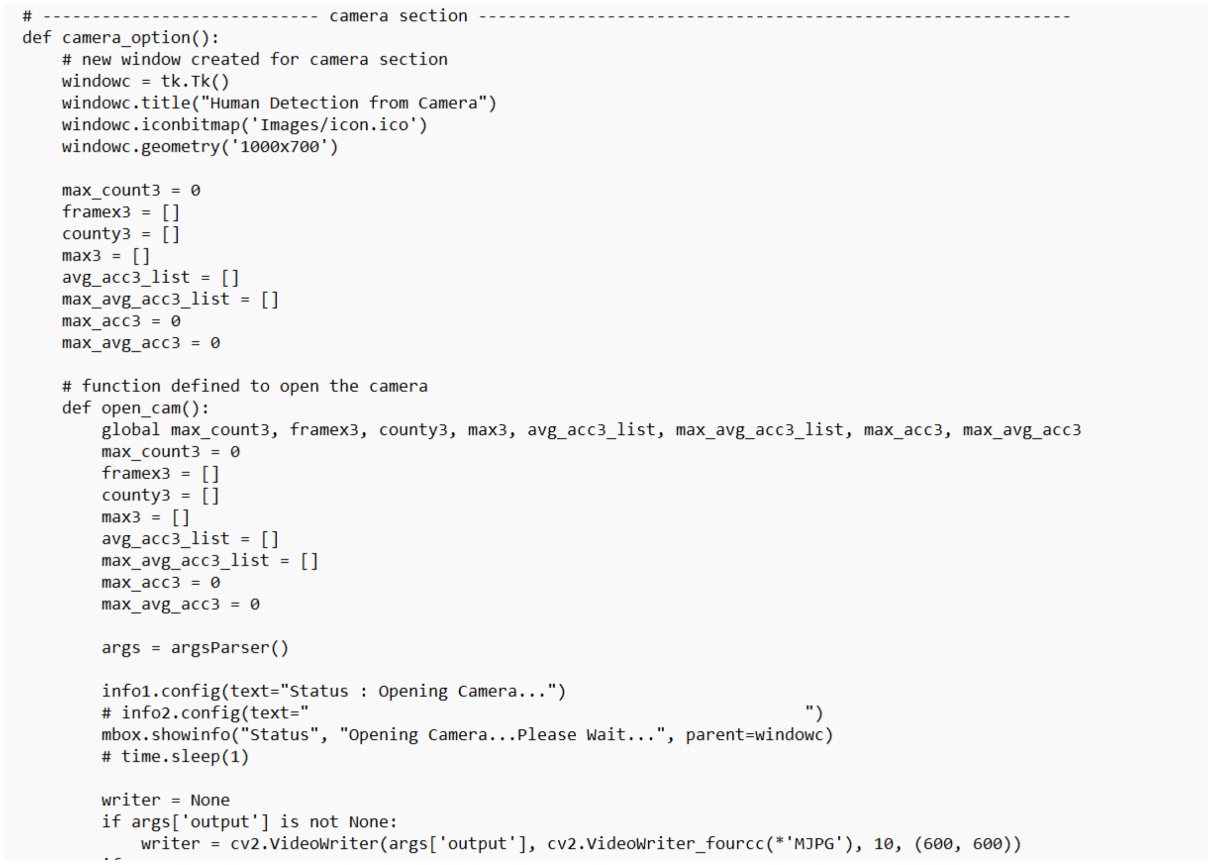
```
                acc+=scores[i]
                if(scores[i]>max_acc2):
                    max_acc2 = scores[i]

        if(person>max_count2):
            max_count2 = person
        county2.append(person)
        x2+=1
        framex2.append(x2)
        if(person>=1):
            avg_acc2_list.append(acc/person)
            if((acc/person)>max_avg_acc2):
                max_avg_acc2 = (acc/person)
        else:
            avg_acc2_list.append(acc)

        if writer is not None:
            writer.write(img)

        cv2.imshow("Human Detection from Video", img)
        key = cv2.waitKey(1)
        if key & 0xFF == ord('q'):
            break
    else:
        break

video.release()
info1.config(text="                                                 ")
# info2.config(text="                                                 ")
info1.config(text="Status : Detection & Counting Completed")
# info2.config(text="Max. Human Count : " + str(max_count2))
cv2.destroyAllWindows()

for i in range(len(framex2)):
    max2.append(max_count2)
    max_avg_acc2_list.append(max_avg_acc2)

Button(windowv, text="Enumeration\nPlot", command=vid_enumeration_plot, cursor="hand2", font=("Arial", 20),bg="orange", fg="blue").place(x=100, y=530)
Button(windowv, text="Avg. Accuracy\nPlot", command=vid_accuracy_plot, cursor="hand2", font=("Arial", 20),bg="orange", fg="blue").place(x=700, y=530)
Button(windowv, text="Generate  Crowd  Report", command=vid_gen_report, cursor="hand2", font=("Arial", 20),bg="gray", fg="blue").place(x=325, y=550)
```

Fig. 4.12



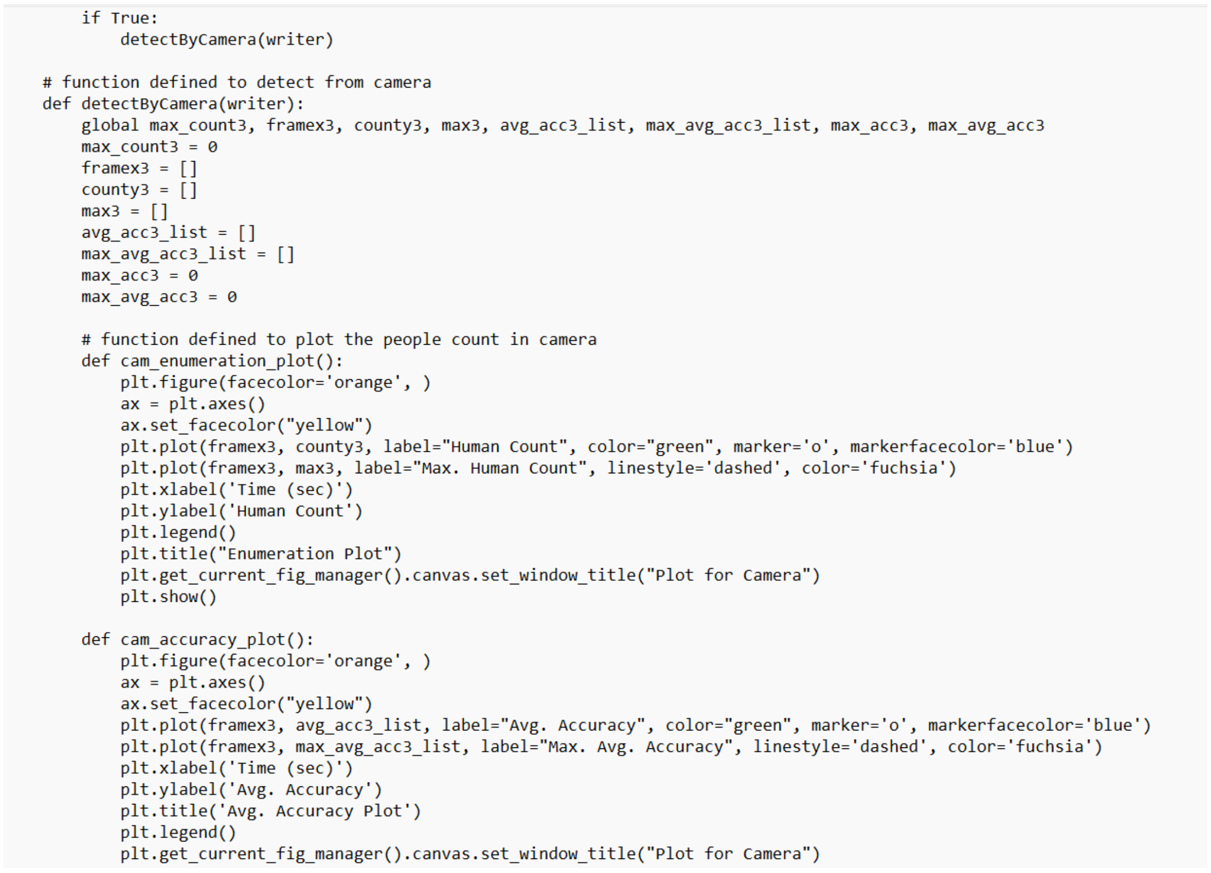
```
# funcion defined to preview the selected video
def prev_vid():
    global filename2
    cap = cv2.VideoCapture(filename2)
    while (cap.isOpened()):
        ret, frame = cap.read()
        if ret == True:
            img = cv2.resize(frame, (800, 500))
            cv2.imshow('Selected Video Preview', img)
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break
        else:
            break
    cap.release()
    cv2.destroyAllWindows()


lbl1 = tk.Label(windowv, text="DETECT  FROM\nVIDEO", font=("Arial", 50, "underline"), fg="brown")
lbl1.place(x=230, y=20)
lbl2 = tk.Label(windowv, text="Selected Video", font=("Arial", 30), fg="green")
lbl2.place(x=80, y=200)
path_text2 = tk.Text(windowv, height=1, width=37, font=("Arial", 30), bg="light yellow", fg="orange", borderwidth=2,relief="solid")
path_text2.place(x=80, y=260)

Button(windowv, text="SELECT", command=open_vid, cursor="hand2", font=("Arial", 20), bg="light green", fg="blue").place(x=220, y=350)
Button(windowv, text="PREVIEW", command=prev_vid, cursor="hand2", font=("Arial", 20), bg="yellow", fg="blue").place(x=410, y=350)
Button(windowv, text="DETECT", command=det_vid, cursor="hand2", font=("Arial", 20), bg="orange", fg="blue").place(x=620, y=350)

info1 = tk.Label(windowv, font=("Arial", 30), fg="gray")  # same way bg
info1.place(x=100, y=440)
# info2 = tk.Label(windowv, font=("Arial", 30), fg="gray")  # same way bg
# info2.place(x=100, y=500)

#function defined to exit from windowv section
def exit_winv():
    if mbox.askokcancel("Exit", "Do you want to exit?", parent = windowv):
        windowv.destroy()
windowv.protocol("WM_DELETE_WINDOW", exit_winv)
```

Fig. 4.13



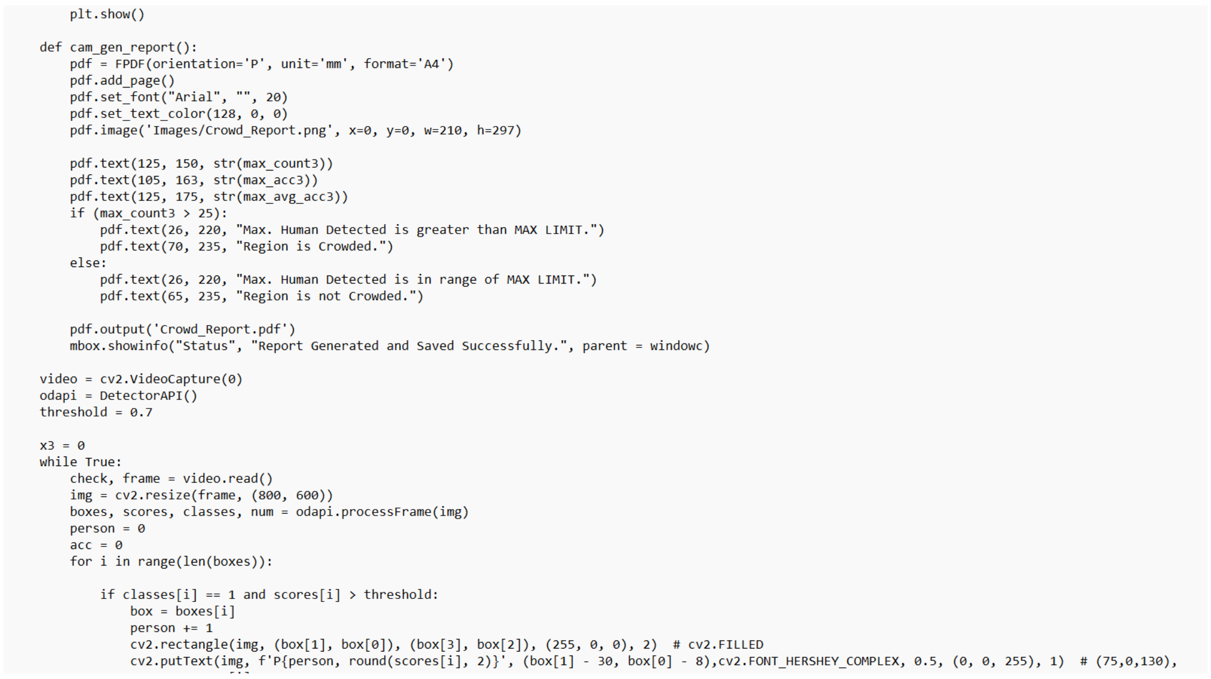
```
# ---------------------------- camera section ------------------------------------------------------------
def camera_option():
    # new window created for camera section
    windowc = tk.Tk()
    windowc.title("Human Detection from Camera")
    windowc.iconbitmap('Images/icon.ico')
    windowc.geometry('1000x700')

    max_count3 = 0
    framex3 = []
    county3 = []
    max3 = []
    avg_acc3_list = []
    max_avg_acc3_list = []
    max_acc3 = 0
    max_avg_acc3 = 0

    # function defined to open the camera
    def open_cam():
        global max_count3, framex3, county3, max3, avg_acc3_list, max_avg_acc3_list, max_acc3, max_avg_acc3
        max_count3 = 0
        framex3 = []
        county3 = []
        max3 = []
        avg_acc3_list = []
        max_avg_acc3_list = []
        max_acc3 = 0
        max_avg_acc3 = 0

        args = argsParser()

        info1.config(text="Status : Opening Camera...")
        # info2.config(text="                                                    ")
        mbox.showinfo("Status", "Opening Camera...Please Wait...", parent=windowc)
        # time.sleep(1)

        writer = None
        if args['output'] is not None:
            writer = cv2.VideoWriter(args['output'], cv2.VideoWriter_fourcc(*'MJPG'), 10, (600, 600))
```

Fig. 4.14



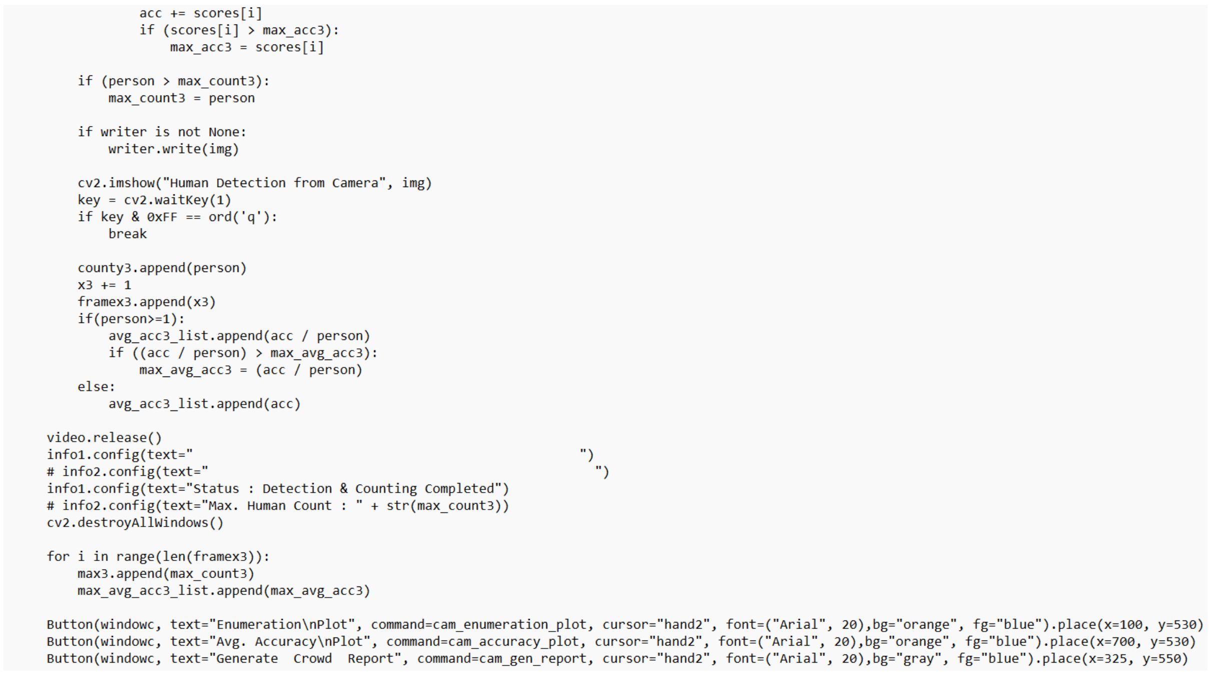
```
    if True:
        detectByCamera(writer)

# function defined to detect from camera
def detectByCamera(writer):
    global max_count3, framex3, county3, max3, avg_acc3_list, max_avg_acc3_list, max_acc3, max_avg_acc3
    max_count3 = 0
    framex3 = []
    county3 = []
    max3 = []
    avg_acc3_list = []
    max_avg_acc3_list = []
    max_acc3 = 0
    max_avg_acc3 = 0

    # function defined to plot the people count in camera
    def cam_enumeration_plot():
        plt.figure(facecolor='orange', )
        ax = plt.axes()
        ax.set_facecolor("yellow")
        plt.plot(framex3, county3, label="Human Count", color="green", marker='o', markerfacecolor='blue')
        plt.plot(framex3, max3, label="Max. Human Count", linestyle='dashed', color='fuchsia')
        plt.xlabel('Time (sec)')
        plt.ylabel('Human Count')
        plt.legend()
        plt.title("Enumeration Plot")
        plt.get_current_fig_manager().canvas.set_window_title("Plot for Camera")
        plt.show()

    def cam_accuracy_plot():
        plt.figure(facecolor='orange', )
        ax = plt.axes()
        ax.set_facecolor("yellow")
        plt.plot(framex3, avg_acc3_list, label="Avg. Accuracy", color="green", marker='o', markerfacecolor='blue')
        plt.plot(framex3, max_avg_acc3_list, label="Max. Avg. Accuracy", linestyle='dashed', color='fuchsia')
        plt.xlabel('Time (sec)')
        plt.ylabel('Avg. Accuracy')
        plt.title('Avg. Accuracy Plot')
        plt.legend()
        plt.get_current_fig_manager().canvas.set_window_title("Plot for Camera")
```

Fig. 4.15



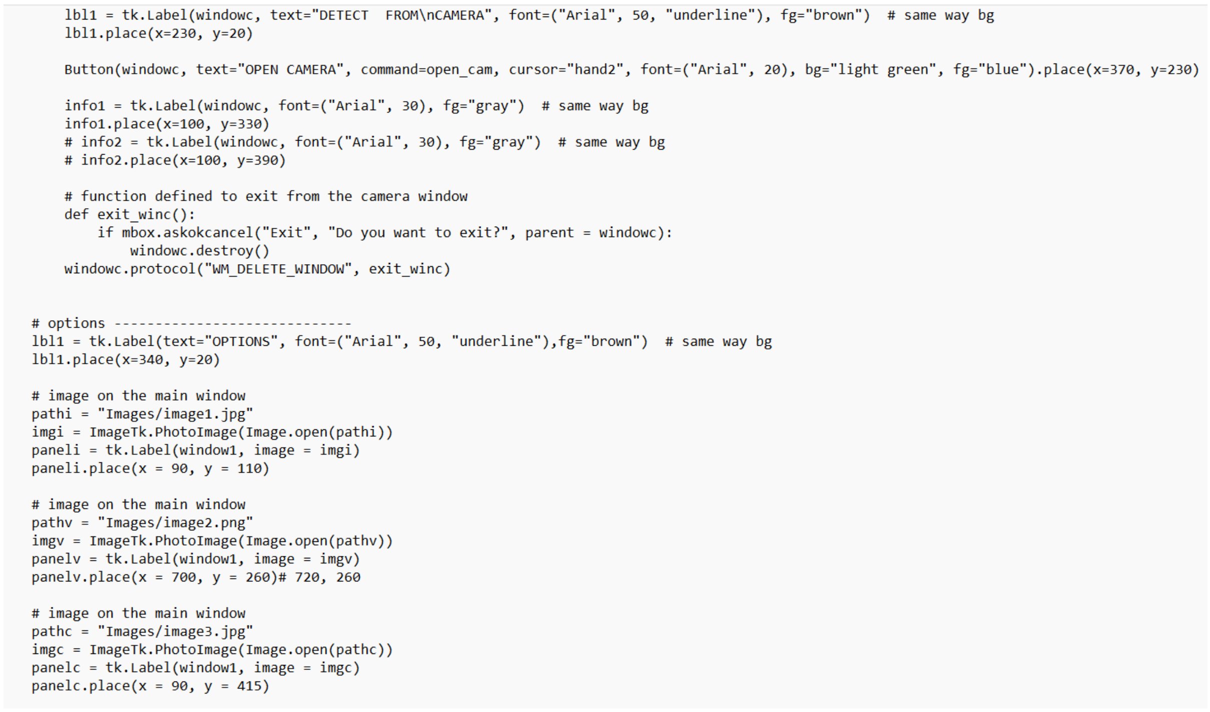
```
        plt.show()

    def cam_gen_report():
        pdf = FPDF(orientation='P', unit='mm', format='A4')
        pdf.add_page()
        pdf.set_font("Arial", "", 20)
        pdf.set_text_color(128, 0, 0)
        pdf.image('Images/Crowd_Report.png', x=0, y=0, w=210, h=297)

        pdf.text(125, 150, str(max_count3))
        pdf.text(105, 163, str(max_acc3))
        pdf.text(125, 175, str(max_avg_acc3))
        if (max_count3 > 25):
            pdf.text(26, 220, "Max. Human Detected is greater than MAX LIMIT.")
            pdf.text(70, 235, "Region is Crowded.")
        else:
            pdf.text(26, 220, "Max. Human Detected is in range of MAX LIMIT.")
            pdf.text(65, 235, "Region is not Crowded.")

        pdf.output('Crowd_Report.pdf')
        mbox.showinfo("Status", "Report Generated and Saved Successfully.", parent = windowc)

    video = cv2.VideoCapture(0)
    odapi = DetectorAPI()
    threshold = 0.7

    x3 = 0
    while True:
        check, frame = video.read()
        img = cv2.resize(frame, (800, 600))
        boxes, scores, classes, num = odapi.processFrame(img)
        person = 0
        acc = 0
        for i in range(len(boxes)):

            if classes[i] == 1 and scores[i] > threshold:
                box = boxes[i]
                person += 1
                cv2.rectangle(img, (box[1], box[0]), (box[3], box[2]), (255, 0, 0), 2)  # cv2.FILLED
                cv2.putText(img, f'P{person, round(scores[i], 2)}', (box[1] - 30, box[0] - 8),cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 255), 1)  # (75,0,130),
```

Fig. 4.16



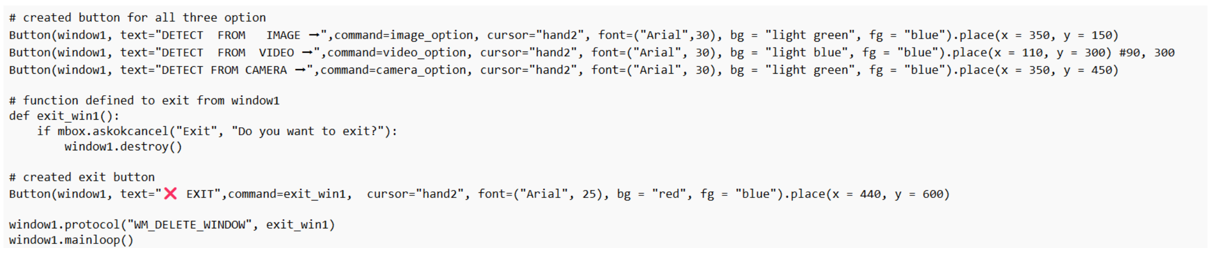
```
                acc += scores[i]
                if (scores[i] > max_acc3):
                    max_acc3 = scores[i]

        if (person > max_count3):
            max_count3 = person

        if writer is not None:
            writer.write(img)

        cv2.imshow("Human Detection from Camera", img)
        key = cv2.waitKey(1)
        if key & 0xFF == ord('q'):
            break

        county3.append(person)
        x3 += 1
        framex3.append(x3)
        if(person>=1):
            avg_acc3_list.append(acc / person)
            if ((acc / person) > max_avg_acc3):
                max_avg_acc3 = (acc / person)
        else:
            avg_acc3_list.append(acc)

    video.release()
    info1.config(text="                                                  ")
    # info2.config(text="                                                  ")
    info1.config(text="Status : Detection & Counting Completed")
    # info2.config(text="Max. Human Count : " + str(max_count3))
    cv2.destroyAllWindows()

    for i in range(len(framex3)):
        max3.append(max_count3)
        max_avg_acc3_list.append(max_avg_acc3)

    Button(windowc, text="Enumeration\nPlot", command=cam_enumeration_plot, cursor="hand2", font=("Arial", 20),bg="orange", fg="blue").place(x=100, y=530)
    Button(windowc, text="Avg. Accuracy\nPlot", command=cam_accuracy_plot, cursor="hand2", font=("Arial", 20),bg="orange", fg="blue").place(x=700, y=530)
    Button(windowc, text="Generate  Crowd  Report", command=cam_gen_report, cursor="hand2", font=("Arial", 20),bg="gray", fg="blue").place(x=325, y=550)
```

Fig. 4.17



```
    lbl1 = tk.Label(windowc, text="DETECT  FROM\nCAMERA", font=("Arial", 50, "underline"), fg="brown")  # same way bg
    lbl1.place(x=230, y=20)

    Button(windowc, text="OPEN CAMERA", command=open_cam, cursor="hand2", font=("Arial", 20), bg="light green", fg="blue").place(x=370, y=230)

    info1 = tk.Label(windowc, font=("Arial", 30), fg="gray")  # same way bg
    info1.place(x=100, y=330)
    # info2 = tk.Label(windowc, font=("Arial", 30), fg="gray")  # same way bg
    # info2.place(x=100, y=390)

    # function defined to exit from the camera window
    def exit_winc():
        if mbox.askokcancel("Exit", "Do you want to exit?", parent = windowc):
            windowc.destroy()
    windowc.protocol("WM_DELETE_WINDOW", exit_winc)


# options ----------------------------
lbl1 = tk.Label(text="OPTIONS", font=("Arial", 50, "underline"),fg="brown")  # same way bg
lbl1.place(x=340, y=20)

# image on the main window
pathi = "Images/image1.jpg"
imgi = ImageTk.PhotoImage(Image.open(pathi))
paneli = tk.Label(window1, image = imgi)
paneli.place(x = 90, y = 110)

# image on the main window
pathv = "Images/image2.png"
imgv = ImageTk.PhotoImage(Image.open(pathv))
panelv = tk.Label(window1, image = imgv)
panelv.place(x = 700, y = 260)# 720, 260

# image on the main window
pathc = "Images/image3.jpg"
imgc = ImageTk.PhotoImage(Image.open(pathc))
panelc = tk.Label(window1, image = imgc)
panelc.place(x = 90, y = 415)
```

Fig. 4.18



```
# created button for all three option
Button(window1, text="DETECT  FROM   IMAGE ➞",command=image_option, cursor="hand2", font=("Arial",30), bg = "light green", fg = "blue").place(x = 350, y = 150)
Button(window1, text="DETECT  FROM  VIDEO ➞",command=video_option, cursor="hand2", font=("Arial", 30), bg = "light blue", fg = "blue").place(x = 110, y = 300) #90, 300
Button(window1, text="DETECT FROM CAMERA ➞",command=camera_option, cursor="hand2", font=("Arial", 30), bg = "light green", fg = "blue").place(x = 350, y = 450)

# function defined to exit from window1
def exit_win1():
    if mbox.askokcancel("Exit", "Do you want to exit?"):
        window1.destroy()

# created exit button
Button(window1, text="❌ EXIT",command=exit_win1,  cursor="hand2", font=("Arial", 25), bg = "red", fg = "blue").place(x = 440, y = 600)

window1.protocol("WM_DELETE_WINDOW", exit_win1)
window1.mainloop()
```

Fig. 4.19

In this section we are going to discuss our code and their result in parts:

The code creates a graphical user interface (GUI) using the Tkinter library in Python. The GUI has a start button that, when clicked, opens up a new window and presents the user with options for real-time human detection and counting from videos, images, or a camera. The code also includes functions for opening and detecting video and image files, as well as for plotting the results. Additionally, the code includes an exit button that allows the user to close the program.

It looks like the code is defining several functions that are related to detecting humans in various types of media (images, videos, and camera feeds). The functions in the "image section" include:

- **open_img()**: opens a file dialog to select an image file, and sets the global variable **filename1** to the path of the selected file.
- **det_img()**: detects humans in the image specified by **filename1**, and calls several functions to display the results.
- **detectByPathImage()**: the main process of detecting humans in an image. It calls several other functions to display the results.

The functions in the "video section" and "camera section" are similar, but are used for videos and camera feeds respectively.

The code is defining a function "video_option" that creates a new Tkinter window with a specific title and dimensions. It also defines several global variables related to human detection in videos, such as "max_count2" and "framex2". The function also defines several other functions, such as "open_vid" which allows a user to select a video file, "det_vid" which detects humans in the selected video, and "detectByPathVideo" which is the main function for human detection in videos. Additionally, the function also includes functions for generating plots and a report, such as "vid_enumeration_plot" and "vid_accuracy_plot".

The code is for a camera section of the larger program, likely a GUI application built using the tkinter library in Python. The camera_option() function creates a new window for the camera section of the program and defines several functions that can be called when buttons are pressed in the GUI.

The open_cam() function starts the camera and begins the process of detecting humans in the camera feed. The detectByCamera() function processes the camera feed and calls several other functions to plot the data and generate a report. The cam_enumeration_plot() function plots the human count over time, cam_accuracy_plot() plots the average accuracy over time, and cam_gen_report() generates a report in PDF format.

Chapter 5

Conclusion and Future Work

In the last section of the project, we generate Crowd Report[5], which will give some message on the basis of the results we got from the detection process. For this we took some threshold human count and we gave different message for different results of human count we got form detection process.

Now coming to the future scope of this project or application, since in this we are taking any image, video or with camera we are detecting humans and getting count of it, along with accuracy. So some of the future scope can be :

- This can be used in various malls and other areas, to analyse the maximum people count, and then providing some restrictions on number of people to have at a time at that place.
- This can replace various mental jobs, and this can be done more efficiently with machines.
- This will ultimately leads to some kind of crowd-ness control in some places or areas when implemented in that area.

References

[1] Programming Computer Vision with Python, 1st Edition, Jan Eric Solem, 2012, O' Reily

[2] Learning OpenCv, Adrian Kaehler and Gary Rost Bradski, 2008, O' Reily

[3] Deep Learning with Tensorflow, Giancarlo Zaccone, Md. Rezaul Karim, Ahmed Menshawy, 2017

[4] Python GUI Programming with Tkinter, Alan D. Moore, 2018

[5] Python Standard Library, Fredrik Lundh, 2001, O' Reily