PANDEMIC CONTROL MEASURES AND ANALYSIS USING COMPUTER VISION

A PROJECT REPORT
Submitted by

Kavin Chandar Arthanari Eswaran [RA1811003010205]
Akshat Prakash Srivastava [RA1811003010157]

Under the guidance of
Dr. Gayathri M

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that 18CSP109L major project report titled "**Pandemic control measures and analysis using computer vision**" is the bonafide work of "**Kavin Chandar Arthanari Eswaran [RA1811003010205], Akshat Prakash Srivastava [RA1811003010157]**" who carried out the minor project work[internship] under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

GUIDE
Dr. Gayathri M
Assistant Professor
Dept. of Computing Technologies

PANEL HEAD
Dr. G. Niranjana
Associate Professor
Dept. of Computing Technologies

CO-GUIDE(if any)
Name of the Co-guide
designation
Dept. of _____

HEAD OF THE DEPARTMENT
Dr. M. Pushpalatha
Professor
Dept. of Computing Technologies

INTERNAL EXAMINER

EXTERNAL EXAMINER

SRM Institute of Science & Technology
School of Computing
Department of Computing Technologies

Own Work Declaration Form

This sheet must be filled in and signed with date along with the student registration number, work will not be marked unless this is done.

To be completed by the student:

Degree/ Course :

Student Name :

Registration Number :

Title of Work :

I / We hereby certify that this work compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this project is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

Signature of the Student(s)

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, and **Dr. T. V. Gopal**, Dean CET, SRM Institute of Science and Technology, for the facilities extended for the project work and their continued support. We extend our sincere thanks to **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her invaluable support. We wish to thank **Dr. M. Pushpalatha**, Professor & Head, Department of Computing Technologies, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the project work.

We are incredibly grateful to our Panel Head, **Dr. G. Niranjana**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her input during the project reviews. We register our immeasurable thanks to our Faculty Advisor, **Dr. C. Jothi Kumar**, Assistant Professor (Sr.G), Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Dr. Gayathri M**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing me an opportunity to pursue my project under her mentorship. She offered me the freedom and support to explore the research topics of my interest.

We sincerely thank the staff and students of the Computing Technologies Department, SRM Institute of Science and Technology, for their help during my project work. Finally, we would like to thank my parents, family members, and friends for their unconditional love, constant support, and encouragement.

Kavin Chandar Arthanari Eswaram

Akshat Prakash Srivastava

ABSTRACT

Abstract— Objects are commonplace, and we intuitively understand their purpose and objectives. We know a coffee cup is a cup by how it looks, and we can differentiate an empty one from a filled one just from a glance. Championing this idea forward with new-age technologies, we create a system to detect and sort the status of objects. We custom tailor our model to quantify pandemic control measures such as face mask usage and social-distancing protocols using computer vision. For face mask usage, we detect various facial landmarks and correlate them with face mask position to accurately determine the different states of wearing a mask in real-time and provide analytics on overall face mask usage. For the social-distancing protocols, we detect and count pedestrians in a controlled space from a forty-five-degree angled top-down camera to determine if they are in line with official norms in real-time. And provide statistics on the status of social distancing in a specific area to inspire reformative changes. We aim to contribute to control measures and analysis using computer vision regarding the pandemic.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **iv**

ABSTRACT **v**

TABLE OF CONTENTS **vi**

LIST OF FIGURES **vii**

ABBREVIATIONS **viii**

1 INTRODUCTION **1**

1.1 Object classification model 1

1.2 Face-mask recognition system 2

1.3 Social-distancing measure system 3

1.4 Software requirements specification 4

2 LITERATURE SURVEY **5**

3 SYSTEM ARCHITECTURE AND DESIGN **8**

3.1 Overall block diagram 8

3.2 Face-mask recognition system 8

3.3 Social-distancing measure system 9

3.4 Facial landmarks 9

3.5 Design of modules 10

4 METHODOLOGY **11**

4.1 Object classification model 11

4.2 Face-mask recognition system 14

4.3 Social-distancing measure system 19

5 CODING AND TESTING **24**

5.1 Object classification model 24

5.2 Face-mask recognition system 26

5.3 Social-distancing measure system 32

5.4 Output 36

6 RESULTS AND DISCUSSIONS **44**
6.1 Findings 44
6.2 Challenges 48
7 CONCLUSION **49**
7.1 REFERENCES **50**
7.2 APPENDIX **52**

A PLAGIARISM REPORT
B CONFERENCE PUBLICATION
C JOURNAL PUBLICATION

LIST OF FIGURES

Fig 3.1 Overall Base diagram

Fig 3.2 Face mask recognition system

Fig 3.3 Social distancing measure system

Fig 3.4 Facial Landmarks

Fig 4 Face-mask Output

Fig 5 No face detected output

Fig. 6 Social distancing of ideal output

Fig. 7 Algorithm visualization output

Fig. 8 Eye-level footage output

Fig. 9 Train station footage output

Fig. 10 Eye-level blurred footage output

Fig. 11 Uncontrolled environment footage output

Fig. 12 Object classification real-time output

Fig. 12.1 Output Analytics

Fig. 13 OCS Performance Analytics

Fig. 14 FMRS Performance Analytics

Fig. 15 SDMS Performance Analytics

Fig. 16 Synopsis of proposed models

ABBREVIATIONS

CCTV - Closed-circuit television

MAFA - Masked face analysis

COCO - Common Objects In Context

YOLO - You only look once

SSDMNV2 - Single shot multibox detector and MobileNetV2

OCS - Object classification system

FMRS - Face mask recognition system

SDMS - Social distancing measure system

FPS - Frames per second

COVID-19 - Serious Acute Respiratory Syndrome Corona virus-2

GLCM - gray level co-occurrence matrix

BPNN - Back-propagation neural network

RMI - Remote motion image

HEX - Hierarchy, and Exclusion

CHAPTER 1
INTRODUCTION

The world has turned upside down (Covid -19). We have encountered multiple variants that come and go in waves. To aid this ongoing battle of attrition against covid and similar future events, we wish to contribute as best as we can by developing widely accessible systems at a coffee cost. We feel our vision-based face-mask recognition system and social-distancing measure system can help drastically reduce transmission rates to achieve herd immunity at the earliest. Using computer vision, a subfield of artificial intelligence, aids us in our pursuit of deriving meaningful results from images. We first start with an object classification system which is the catalyst for developing more pandemic control measure and analysis systems. These computer vision-based solutions come at a coffee cost and are versatile in application. They fit well into already existing systems like CCTV cameras and smartphones etc. Software marvels like these inspire us to develop solutions to aid current problems and contribute to leaps in the future.

1. 1 Object classification model

We take the live camera feed and pass it to the object classification model. The object classification model recognizes everyday objects. This model is custom-tailored to recognize face masks and pedestrians for the project. Each of these models then provides valuable results. The base model recognizes objects and sorts them alphabetically along with their detection confidence.

1.2 Face-mask recognition system

The face-mask recognition system has various use-cases in our current and future society, From entry checks into convenience stores to super malls and monitoring face mask usage in controlled environments such as offices, schools, and university campuses. We use a pre-trained mask detection model and layer it over a face detection algorithm. We consider various cases of interaction between the position of the mask and the prominent facial landmarks such as the nose, mouth, and eyes. We can then classify a worn face mask's different states such as mask on, mask worn incorrectly and mask off in real-time by using face-mask position and facial landmarks in unison. Hence, our objectives are to determine and differentiate the different states of wearing a mask and accurately classify and Analyze face-mask usage for an individual. We also prioritize the ease of implementation on any device without compromising quality and improve detection confidence to recognize any face masks and make sure only masks are recognized. We pictorially represent the acquired data for clarity and send non-intrusive warning cues.

1.3 Social-distancing measure system

The social-distancing measure system represented is specifically made to quantify the status of social distancing in controlled environments. We believe it helps us make informed decisions to regulate crowds. We count and organize crowds into groups of followers and deviants. We achieve this by using a pedestrian detection algorithm from a top-down CCTV camera and accurately gauging the distance between the calculated centroids of pedestrians. We then accurately detect individuals in high-density crowds and classify them in real-time.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space

q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

We use this euclidean distance equation to calculate the distance between centroids of pedestrians correctly.

1.4 Software requirements specification

- OpenCV - we use openCV for live video capture and display purposes
- Dlib - primarily used to detect the 68 facial landmarks
- Numpy - convenient mathematical utility library
- Matplotlib.pyplot - helps us plot graphs and pictorially represent statistics and analytics
- Tensorflow – Keras - used to load model, preprocess input, and change image to an array
- Mask_detector.model - pre-trained model helps us recognize a plethora of masks
- OpenCV - we use openCV for live video capture and display purposes
- Numpy - convenient mathematical utility library
- Imutils – resize display window
- Scipy. spatial - used to calculate the distance between centroids
- Matplotlib.pyplot - helps us plot graphs and pictorially represent statistics and analytics

CHAPTER 2

LITERATURE SURVEY

A. The vision-based face-mask recognition system

[1-5] - These papers use methods such as the Deep Learning model (Inception V3) and achieve a 99.9 % rate of classification, Resnet 50 for feature extraction, Support vector classification, and ensemble algorithm for classification, SSDMNV2 for face detection, and MobilenetV2 for mask detection, One particular paper performs random sampling on MAFA dataset and achieves a p-value of 1.07 from 11.82, Retina Face Mask method uses new dataset for incorrectly worn masks. From these papers, we borrow a few methods and use the best of them for our applications. The coco dataset, frozen_inference_graph, and ssd_mobilenet_v3_large_coco all have been tested and optimized for our particular model.

[6-10] - We observe many advancements made here in face-mask recognition. New research has proposed a data set of realistic fabric face masks evaluated using GLCM. The following paper identifies the increased vocal effort and muscle constrictions made when speaking under a face mask. Next, we observe a deep learning method that uses a massive dataset of 25,000 images using a 224x224 resolution to distinguish face mask status. This particular method has achieved a 96% success rate. We also perceive a new automated face mask detection system that uses BPNN and backpropagation and achieves a 91% success rate in detecting face masks. Finally, we look at a paper that improves upon the YOLOv3 with a YOLOv3_slim which improves the selection speed from 15.67 fps to 16.89 fps. These research papers give us a wide array of what makes a good face-mask recognition algorithm.

B. Social-Distancing measure system

[11-15] - These models use methods such as distance classification using YOLO's dataset paired with a thermal camera for temperature monitoring to identify SD Violators. Mobilenet, resnet, and VGG are used by another model to classify SD and face-mask violators. Next, some models send auditory and visual warning cues to alert individuals and point out high-density areas. Finally, one impressive model uses Twitter's social mobility index, public geolocated Twitter data, to find SD violators. One particular paper finds the distance between pre-distributed mobile nodes in places like university campuses as a method.

[16-20] - These papers provide info on the impact of social distancing and its effect on reducing transmission rates. We see particular pacers researched, particularly in Brazil and Italy. One specific paper delves into the changing growth rates in Covid-19 cases based on enforcing social distancing. It has shown results of slowing the growth rate of covid-19 by 5.4% in less than a week, 6.8% in ten days, 8.2% after eleven to fifteen days, and 9.1% after twenty days. Some papers have observed its effect with other mitigation measures such as isolating contaminations, contact tracing, school closures, workplace closures, avoiding high-density crowds, etc. All these reviews conjugate the available evidence on the impact social distancing has on epidemics.

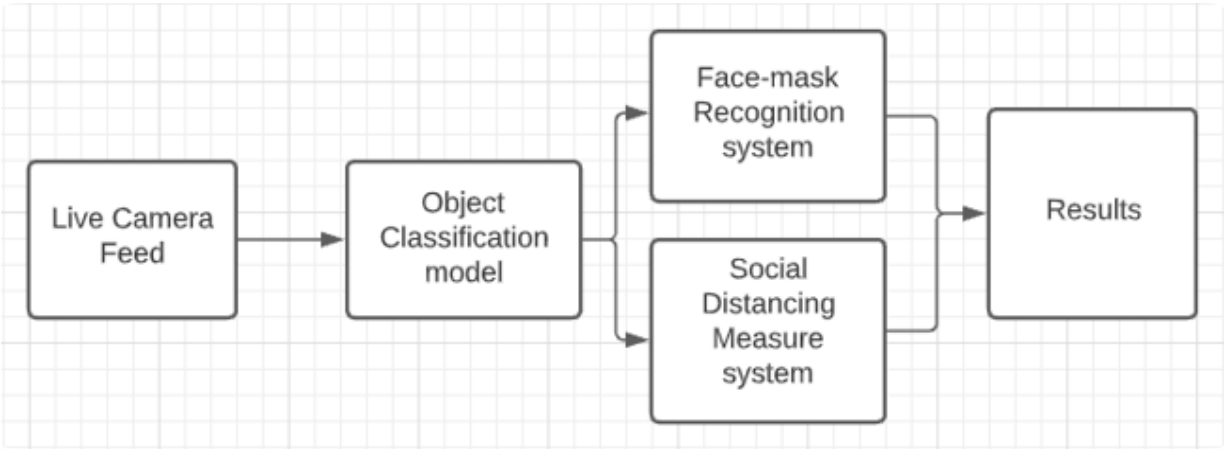
C. Vision-based Object Sorting System

[21-25]- The following sorting systems specialize in particular fields, such as mango feature extraction using 2D and 3D visual properties. Next, we observe a vision-based book sorting algorithm that extracts book contours and normalizes the image to sort books. We also see a unique prototypical computer vision-based date grading and sorting system. More widely used object sorting systems are robotic garbage sorting systems using RPN and VGG-16. Finally, we see a vision-based robot sorting system that uses STM32F4 as the central controller and CDD imaging sensor for the general sorting of targets using a robot.

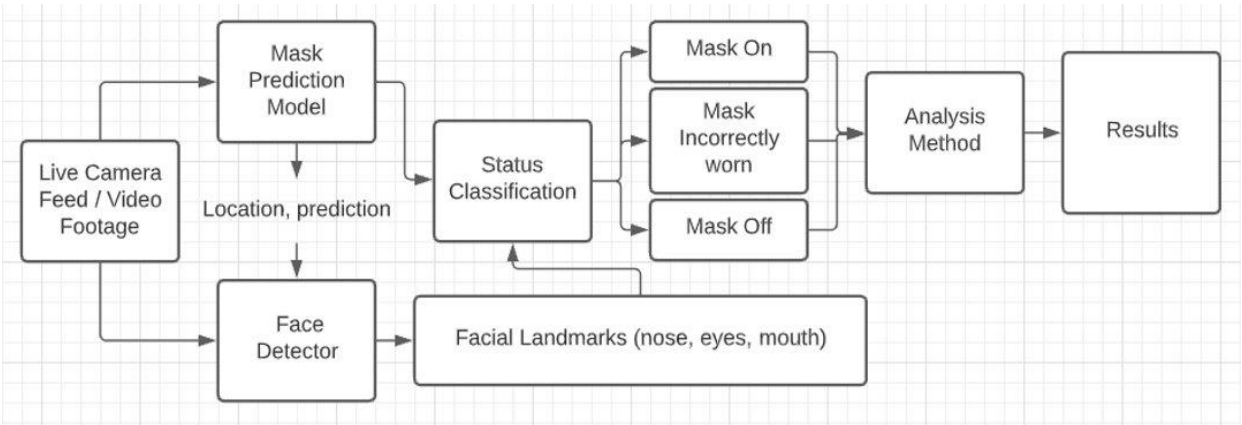
[26-30] - We look at papers with various methods of vision-based object classification. There are methods such as detecting recurrent motion for each object and developing a vector called RMI to calculate the repeated motion of objects. We also observe a new formalism that holds semantic relations between multiple labels used for the same object. These account for mutual exclusion and overlap using HEX graphs. Finally, we look at models that push the benchmark on object classification using deep hierarchical architectures such as NORB, CIFAR10, and MNIST for handwritten digit recognition.

To conclude, All these papers across different research fields such as object sorting systems, face mask recognition, and social distancing measure empower us to develop our best possible vision-based pandemic control measures and analysis systems in our battle against the pandemic and contribute to the current and future post pandemic endeavors.

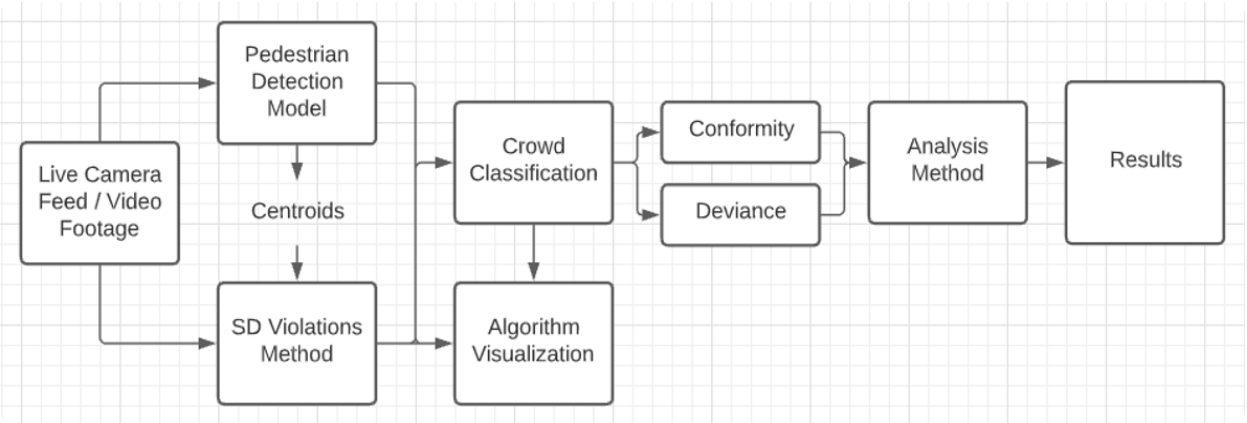
CHAPTER 3
SYSTEM ARCHITECTURE AND DESIGN



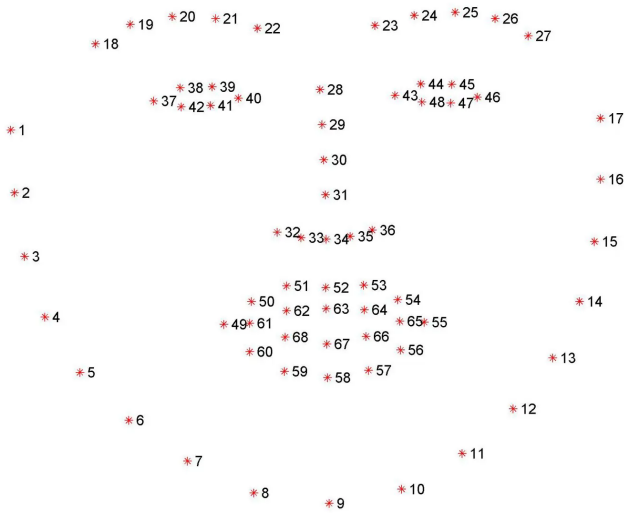
3.1 Overall base diagram



3.2 Face-mask recognition system



3.3 Social-distancing measure system



3.4 Facial Landmarks

3.5 Design of modules

We start by designing an object classification model to recognize and sort everyday household objects. We use the Coco dataset and openCv to accomplish the above task. Next up, we custom tailor this model for a face-mask recognition system and a social-distancing measure system. We begin by using a face-mask detection model and a facial feature detection model to accurately determine the positions of the face-mask relative to the face at all times. We achieve this by using the modules openCV, coco dataset, dlib, NumPy, TensorFlow, etc.

OpenCV - we use openCV for live video capture and display purposes. Dlib - primarily used to detect the 68 facial landmarks.

Numpy - convenient mathematical utility library.

Tensorflow – Keras - used to load model, preprocess input, and change image to array.

Mask_detector.model - pre-trained model helps us recognize a plethora of masks.

Imutils – resize display window.

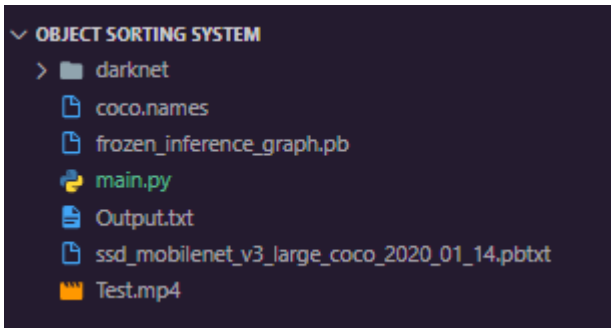
Scipy. Spatial - used to calculate the distance between centroids.

Matplotlib.pyplot - helps us plot graphs and pictorially represent statistics and analytics.

CHAPTER 4
METHODOLOGY

4.1 Object Classification model

The object classification model recognizes and sorts everyday household objects. We then use this model to sort objects with high confidence rates alphabetically. We can then custom tailor this model to build the fundamentals for our face-mask recognition system and social distancing measure applications.



The object classification model comprises a few building blocks: the coco dataset, frozen_inference_graph, ssd_mobilenet_v3_large_coco, and test.mp4. These have been trained to recognize about 80-90 household items.

```
import cv2
import imutils
import matplotlib.pyplot as plt
import time
```

We import OpenCV to help us capture video footage and display our findings in real-time. It is also a handy computer vision library that allows us to analyze video footage frame by frame, and we also use its new deep neural network detection model. Imutils is used to resize the video footage to improve recognition, confidence, and feasibility. Next up, Matplotlib. pyplot is used to represent all our final results and findings pictorially. Finally, The time module helps with calculating the total frames per second for any program in real-time and for other niche uses.

```
MIN_CONFIDENCE = 0.2
NMS_THRESHOLD = 0.3
```

We set the minimum confidence and non-maxima suppression threshold, actual values for setting standards. First, we import our coco dataset into an objects list. Next, we import the config path and weights path. These are used for OpenCV's new deep neural network detection model. We then set the input size, scale, mean, and swap.

```
def fps(picture, prev_Time):
```

We have our fps utility function. This method takes the current time and prev time of a previous frame to calculate the frames per second in a given moment. We also store these values in a list to calculate averages and pictorially represent these changes using graphs. However, we display these fps values in real-time on the screen using OpenCV's put text method. Finally, we update the prev_time with current_time and return the updated value fr future calculations.

```
def analysis(d):
```

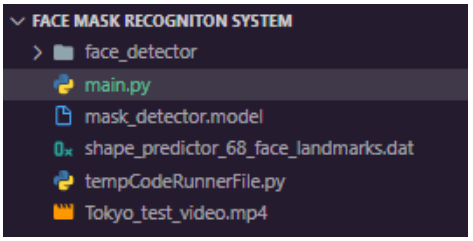
The analysis method is a simple method that allows us to sort and display the acquired eligible values in the dictionary. We write into the output.txt file a table showing an object and its confidence. We also make sure it is displayed in alphabetical order considering cases of substantial inputs. We can then use this final output.txt file as a dataset for other applications and has a myriad of use-cases.

```
def main():
    capture = cv2.VideoCapture(0)
    #capture = cv2.VideoCapture('Test.mp4')
```

The primary method is our main runner function. Here, we capture video footage and analyze it frame by frame. We flip the frame to simulate a mirror-like quality of the user. We also resize the bracket for more efficiency. Next, we pass this resized frame through a net. detect() method with a confidence threshold of 50%. Objects that meet this criterion return their classIDs, confidence rate, and bounding box dimensions. The classID allows us to identify the detected object from the dataset. We can proceed as long as we have witnessed something and classIDs aren't empty. We traverse through this list of identified things and document them in our dictionary if they haven't been written before and have a confidence percentage of over 70%. This dictionary is mutable, and we use this data in our analysis method to sort and display as needed. We draw clear green bounding boxes and display confidence rates in tandem to visually represent the detection in real-time. Finally, we map 'X' as the key to terminate all real-time actions and begin the analysis method to find and display findings appropriately.

4.2 Face-mask recognition system

The face-mask recognition system is built to recognize and identify all the different stages of wearing a face mask. It provides statistics and pictorially represents the acquired data. We display the overall duration and the duration of each set of worn face masks. This info proves to be valuable in monitoring face mask usage.



We import a mask_detector.model trained to identify masks and provide a binary result. Then we use this data along with the shap_predictor_68 face landmarks, which detects facial landmarks such as eyes, nose, and mouth. These together form the core of this application. We also run this program over a test video and live webcam footage to run tests and optimize accordingly.

```
# imports
import time
import cv2
import dlib
import numpy as np
import matplotlib.pyplot as plt
```

The imported modules serve us in implementing the methods for analyzing and displaying results. The time module helps us calculate the duration of the different stages and the overall duration of the module. The OpenCV computer vision module is essential for displaying the real-time intricacies of face mask recognition. Dlib is important for face recognition and is crucial for the module as it is what helps us determine the facial landmarks necessary for us to determine the desired results. Finally, Matplotlib. pyplot helps us pictorially represent the final results.

```
# keras model imports
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
```

We import TensorFlow Keras models to pre-process face input and convert an image to the array.

```
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
facemodel = cv2.dnn.readNet(prototxtPath, weightsPath)
```

Next, we use OpenCV's deep neural network library to detect face models.

```
face_detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

We also use dlib to acquire frontal face data. We also use dlib's shape predictor method to get the 68 facial landmarks. We then load the mask detector model, which helps us with the mask detection portion of the application.

```
def test():
```

The test method for the face mask recognition system is where we plot bar and line graphs to observe the smooth workings of the application. The bar graph denotes the ballpark of how many frames per second we can muster. The line graph helps us predict the consistency of the frames per second and observe frame drops, if any. We take note of the average frames per second which is a decent indicator of how well the model runs in an average computer. We also actively work on optimizing processes as much as we can and observe the fruits it bears in the overall runtime and frames per second we acquire.

```
def fps(picture, prev_Time):
```

The fps method for the face mask recognition system is used to calculate the frames per second for every moment, and this data is then displayed in real-time. We use this data also for testing and optimization purposes.

```
def duration():
```
```
def formatter(seconds):
```

The duration and formatter methods are quality of life methods used to analyze the module and its results. The duration method gets the total time, and the formatter formats it into hours, minutes, and seconds for the convenience of comparison.

```
def analysis():
```

The analysis method displays a pie chart to compare the durations of each stage of a worn face mask. It also showcases the exact results in text for full transparency. The pie chart helps convey the message that isn't bluntly aware through numbers and data. It also provides a blank page if no subject is detected in the frame for the entire duration of the live webcam or video footage.

```
def mask_prediction(frame, facemodel, maskmodel):
```

The mask prediction method is the heart of our application. This method uses the frame, face, and mask models as arguments. It then passes the frame through a deep neural network to return a blob, passing through a face model to gain faces, location, and prediction data. Traversing through these detections, we display the detected face and make a list of all faces and their respective areas. We return these lists for other purposes we'll explore down the line.

```
def nose_detection(frame, landmarks, color, draw):
```

As the name indicates, the nose detection method displays four important facial landmarks present on the nose. nose_top, nose_left, node_right, nose_bottom. These points are enough to detect the position of the nose adequately. This method is important for us as we correlate this position of the nose and the position of the face mask to detect if a face mask is worn incorrectly. We return true to detect if our nose is not covered. We also make sure that whatever is covering the nose and undetectable is a mask. This method helps the system efficiently draw results from the frames in real-time.

```
def mouth_detection(frame, landmarks, color, draw):
```

The mouth detection method is similar to the nose detection method. However, we use this information to differentiate between the incorrectly worn mask and a mask that is completely off. We use the nose, mouth, and mask positional data to judge if a mask is correctly adorned. Therefore, this method is crucial in the face-mask recognition module.

```
def eye_detection(frame, landmarks, color, draw):
```

The eye-detection method is similar to the above two methods but is necessary so we can efficiently predict the locations of the nose, mouth, and mask with the eyes. The positional data of the eyes helps us note if the face mask is positioned to cover them. With all these landmarks and the mask position, we can pinpoint the role of the face mask in real-time.

```
def status(mask, withoutMask, nose, mouth, eye):
```

The mask status function serves to return the final status of the input frame. We create global variables for counting and dividing frames based on their position. First, we check to see if the mask on confidence is higher than the mask off confidence to ensure a mask is present in the frame. Next, we compare this data and check for covered noses and mouths. If all conditions are accurate, we can confirm the subjects in the frame are wearing the face mask correctly. Secondly, If the nose or mouth is exposed and without a mask, confidence is below the specific value of 0.99, we can determine that a mask is worn incorrectly. Finally, If neither of the above two conditions pass, we can safely confirm that the subjects given the camera are not wearing a mask in that exclusive frame. Once we confirm the status of the particular frame, we can assign a label and a color to denote the result of a frame in real-time. If the 'Mask on' position has been identified, we set the color to green and 'Mask on.' This label and color variables are then returned to the runner function, where they are used to visualize the result in the display output. Other alternative labels and colors are 'Mask worn incorrectly' in blue and 'Mask off' in red. These serve to convey the message clearly and visually. Next, the global variables m_on, m_half, and m_off are used to segregate frames. These variables are used in the analysis method to represent the final statistics pictorially for viewing purposes.

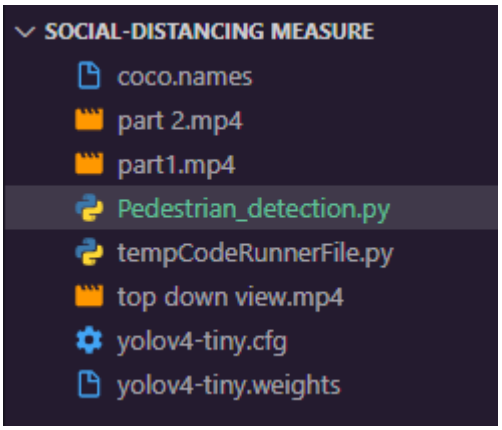
```
def main():
    global start
    print("Starting Face-mask Detection protocol...")
    capture = cv2.VideoCapture(0)
```

We use OpenCV to capture live video footage. We then go frame by frame through the video. Each frame is passed to a mask_prediction method() from which we gain the location and prediction results of a face mask if detected.

We also pass structures to a face detector method from which we earn a list of all detected faces. We analyze each for facial landmarks such as nose, mouth, and eyes through this face list. We then correlate the prediction result of the mask detection model with the detected facial landmarks in real-time through a particular status method. This method allows us to properly distinguish and identify all possible states of a worn face mask. This method then returns the appropriate label and color to determine forms in real-time. Next, we allow the user to press 'X' to pause the video and produce the proper face-mask usage statistics until that point. Using matplotlib to show the relational duration with a pie chart, we pictorially represent this. Finally, we also run tests to assess the program's overall performance to address and improve upon challenges and limitations objectively.

4.3 Social distancing measure system

The Social distancing measure system is built to observe crowds in a controlled environment. The model recognizes all pedestrians in the frame and classifies them into two categories. People who conform to social distancing protocols are known as conformers, and people who don't, aka deviants. This classification occurs in real-time, and the algorithm's workings are visualized.



We bring in the coco and Yolo datasets and modules to serve us in detecting pedestrians. We use a few sample videos to test our algorithm in different scenarios. Ideally, we observe it's working from a top-down camera at a forty-five-degree angle.

```
from cProfile import label
import numpy as np
import math
import cv2
import imutils
from scipy.spatial import distance as dist
import matplotlib.pyplot as plt
import time
```

The above imports are necessary for implementing the module as intended. The cProfile python library is most commonly used for profiling, and we use it to test the overall runtime of our program. The NumPy library is widely used for various mathematical operations on arrays necessary for our model's work. The math module has plenty of presets, making testing different mathematical approaches time and space-efficient. OpenCV module, as discussed before, is our standard computer vision library, and imutils help in resizing the final display frame to cater to the video footage. The scipy.spatial important helps calculate the distance between centroids in real-time. The matplotlib.pyplot is used here for pictorially representing the female findings. Finally, the time module is used to calculate frames per second.

```
NMS_THRESHOLD = 0.3
MIN_CONFIDENCE = 0.2
MIN_DISTANCE = 40
```

The above three constants are crucial in setting the boundaries for the application and help us alter them as we please. The NMS threshold is essential for non-maximal suppression, which helps us from overlapping bounding boxes. The MIN confidence is critical for setting the minimum standard required for recognizing an object as a pedestrian. Finally, The MIN distance is the heart of the application as it is the very determinator between conformers and deviants.

```
def distance_function(x1, y1, x2, y2):
    return ((x2 - x1)**2 - (y2-y1)**2)**(0.5)
```

The distance method is a utility function used here to calculate the euclidean distance between pedestrians. We pass two centroids, and the process returns the euclidean distance between them.

```
def test():
```

The test method here is used to display the FPS consistency bar graph, and the FPS drops line graph. Both of these graphs help us observe the system's overall performance in any machine. It also allows us to record the performance changes based on the input footage. All of these changes are colorfully represented in a bar graph. The drops in frames per second denoting stutters are visually illustrated in an easily understood line graph. These visual indicators help us make and observe new optimizations made to the program.

```
def fps(picture, prev_Time):
```

The fps method for the social distancing measure is used to calculate the frames per second for every moment, and this data is then displayed in real-time. We use this data also for testing and optimization purposes. It functions as a handy utility function.

```
def analysis(right, wrong):
```

The analysis method visually represents the final findings of the program. We use a clear and colorful pie chart to show the difference between conformers and deviants. We also count crowds and pictorially depict the split among them for ease of understanding. The total average performance of the system, the real people in view of the camera, and the conformers and deviants are all counted and documented alongside the pie chart.

These help objectively access the situation and make essential decisions in regulating controlled crowds moving forward. Finally, we can automate classifying masses in real-time and send statistical reports to improve the social distancing index of controlled environments such as airport lobbies, train station pathways, universities, school classrooms, etc.

```
def pedestrian_detection(image, model, layer_name, personidz=0):
```

The pedestrian detection method recognizes and highlights detected pedestrians from the input video footage. We pass a particular frame to acquire a blob using OpenCV's deep neural network method blob from Image. We give the detected blob through a few trained models, which return our desired results. Next, we iterate and traverse through these results, looking for detections. If found, we check if the detected object is a pedestrian based on classID and make sure the confidence is above the minimum threshold. Once these conditions pass for true, The bounding box dimensions, centroids of each pedestrian, and confidence rates are all stored in lists. We also perform non-maxima suppression on our detected objects to suppress weak and overlapping bounding boxes. Finally, we make sure idzs list isn't empty to ensure results are returned only when intended objects are detected. The final result list consists of the confidence rate, the bounding box dimensions, and centroid coordinates returned for any input frame passed through the method.

```
def centroid_visual(image, centroids, draw=False):
```

The visual centroid method serves us to visualize the distance calculations made between the pedestrian centroids in real-time. The distance between one centroid and every other centroid is calculated, and only the centroids within the proximity of minimum distance are considered for visualization. The distance function used here calculates the euclidean distance between centroids.

If all conditions are met, we use OpenCV to draw a line between social-distancing violators in real-time. If the user sets the draw variable to false, we disable the entire visualization process, and all the visualized calculations are made in the background purely for functionality.

```
def violations(image, right, wrong):
```

This core method elevates this pedestrian detection program to a social distancing measure system is the violations method. This method analyzes the input frame and records all the distance violations. These violations are intersected with results from the pedestrian detection method. The pedestrians that are in violation are painted red and counted as deviants. The pedestrians in line are counted as conformers and remain in the green. Finally, We also display the crowd count, conformers, and wrongdoers in the same frame in real-time. The updated image, proper count, and wrong count of that particular frame are then returned for documentation and analysis purposes.

```
labelsPath = "coco.names"
LABELS = open(labelsPath).read().strip().split("\n")
```

```
weights_path = "yolov4-tiny.weights"
config_path = "yolov4-tiny.cfg"
model = cv2.dnn.readNetFromDarknet(config_path, weights_path)
```

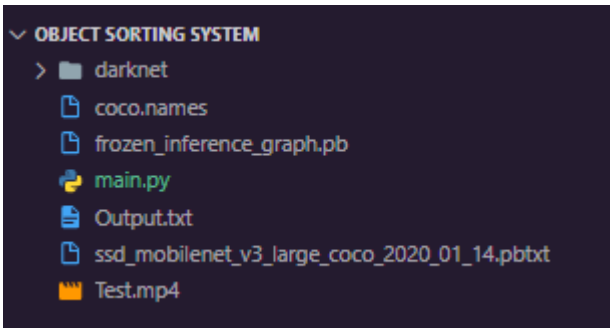
The coco dataset and the Yolo models are incorporated into the program here. They are integral in helping us identify pedestrians from video footage in real-time. The program is tested on different videos such as ideal top-down footage, average eye-level footage, and worse case footage scenarios in low lighting or blurred footage to test the program's limits in different circumstances.

```
cap = cv2.VideoCapture("top down view.mp4")
#cap = cv2.VideoCapture("part1.mp4")
#cap = cv2.VideoCapture("part 2.mp4")
```

We take live video footage from CCTV cameras present at a forty-five-degree incline from the top. We then analyze this footage in real-time to detect social-distancing violators. We first begin by detecting pedestrians by passing through a pedestrian_detection model. This method returns the confidence percentage, bounding box dimensions, and centroids of the detected pedestrians. We then give the list of centroids and their respective frames to the violation method. This method calculates the distance between the centroids to check for social-distancing violations and then returns a color to match the status. We also use this method to count crowds in a frame and classify them into followers and deviants in real-time. We pictorially represent this data using matplotlib—pyplot as a pie chart to indicate the social-distancing index. Finally, we run tests to test the system's limitations to optimize as best as possible.

CHAPTER 5
CODING AND TESTING

5.1 Object Classification model

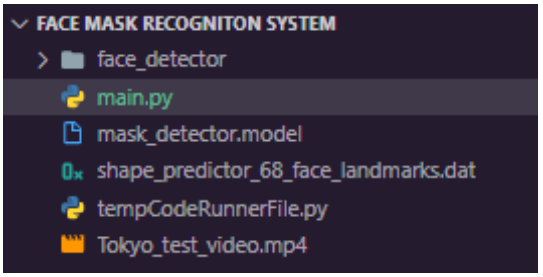



```python
import cv2
import imutils
import matplotlib.pyplot as plt
import time

MIN_CONFIDENCE = 0.2
NMS_THRESHOLD = 0.3

# object data set
Objects = []
with open('coco.names', 'rt') as f:
    Objects = f.read().rstrip('\n').split('\n')

# object detection model
configPath = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
weightsPath = 'frozen_inference_graph.pb'
net = cv2.dnn_DetectionModel(weightsPath, configPath)
net.setInputSize(320, 320)
net.setInputScale(1.0 / 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)

# fps
avg_fps = []
test_time = []
prev_Time = 0
```

```python
def fps(picture, prev_Time):
    current_Time = time.time()
    fps = 1 / (current_Time - prev_Time)
    avg_fps.append(fps)
    test_time.append(current_Time)
    prev_Time = current_Time
    cv2.putText(picture, f'FPS: {int(fps)}', (300, 300),
                cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 2)
    return prev_Time


d = {}


def analysis(d):
    with open('Output.txt', 'w') as f:
        f.write('Object ')
        f.write('Confidence\n')
        f.write(' ')
        f.write(' \n')
        for i in sorted(d):
            f.write(i+' ')
            f.write(str(d[i])+'\n')
```

```python
def main():
    capture = cv2.VideoCapture(0)
    #capture = cv2.VideoCapture('Test.mp4')
    prev_Time = 0
    while capture.isOpened():
        _, frame = capture.read()
        frame = cv2.flip(frame, 1)
        frame = imutils.resize(frame, 700)
        classIds, confs, box = net.detect(frame, confThreshold=0.55)
        if len(classIds) != 0:
            for classId, confidence, box in zip(classIds.flatten(), confs.flatten(), box):
                if Objects[classId-1] not in d and confidence >= 0.7:
                    d[Objects[classId-1]] = str(confidence*100)[:2]+'%'
                cv2.rectangle(frame, box, color=(0, 255, 0), thickness=2)
                cv2.putText(frame, f'{Objects[classId-1].upper()} {int(confidence*100)} %',
                    (box[0], box[1]-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        prev_Time = fps(frame, prev_Time)
        cv2.imshow("Object Detection Model", frame)
        if cv2.waitKey(10) & 0xFF == ord('x'):
            analysis(d)
            break

    cv2.destroyAllWindows()
    capture.release()


if __name__ == '__main__':
    main()
```

5.2 Face-mask recognition system




```python
# imports
import time
import cv2
import dlib
import numpy as np
import matplotlib.pyplot as plt

# keras model imports
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array

# facemodel
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
facemodel = cv2.dnn.readNet(prototxtPath, weightsPath)
face_detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# mask model
maskmodel = load_model("mask_detector.model")

# test


def test():
    New_Colors = ['green', 'blue', 'purple', 'brown', 'teal']
    # bar graph
    plt.bar(avg_fps, test_time, color=New_Colors)
    plt.title('Face-mask recognition system')
    plt.xlabel('FPS')
    plt.ylabel('Time (s)')
    plt.show()
    # line graph
    plt.plot(test_time, avg_fps)
    plt.xlabel('Time (s)')
    plt.ylabel('FPS')
    plt.show()
```

```python
def fps(picture, prev_Time):
    current_Time = time.time()
    fps = 1 / (current_Time - prev_Time)

    avg_fps.append(fps)
    test_time.append(current_Time)
    prev_Time = current_Time
    cv2.putText(picture, f'FPS: {int(fps)}', (300, 400),
                cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 2)
    return prev_Time


# analysis
start = 0
end = 0


def duration():
    global end
    end = time.time()
    dur = end - start
    return dur


def formatter(seconds):
    return time.strftime('%H:%M:%S', time.gmtime(seconds))
```

```python
def analysis():
    label = ['Mask on', 'Mask worn incorrectly', 'Mask off']
    total = m_on+m_half+m_off
    if total != 0:
        percent = [f'{int((m_on/total)*100)}%',
                   f'{int((m_half/total)*100)}%', f'{int((m_off/total)*100)}%']
        colours = ['green', 'cyan', 'red']
        myexplode = [0.02, 0.01, 0.01]
        dur = duration()
        plt.pie(np.array([m_on, m_half, m_off]), labels=percent,
                explode=myexplode, shadow=False, colors=colours, radius=0.6)
        plt.legend(title='Status Frequency', loc='lower right', labels=label,
                   bbox_to_anchor=(1.3, 0.8))
        plt.text(-1.5, 1, f'Performance: {int(sum(avg_fps)/len(avg_fps))} fps',
                 family='serif', ha='left')
        plt.text(-1.5, -0.9, f'Total Duration: {formatter(dur)}',
                 family='serif', ha='left')
        plt.text(-1.5, -1.0, f'Mask on: {formatter(dur*(int(percent[0][:-1])/100))}',
                 family='serif', ha='left')
        plt.text(-1.5, -1.2, f'Incorrectly worn: {formatter(dur*(int(percent[1][:-1])/100))}',
                 family='serif', ha='left')
        plt.text(-1.5, -1.1, f'Mask off: {formatter(dur*(int(percent[2][:-1])/100))}',
                 family='serif', ha='left')
        plt.title('Results')
        plt.show()
    else:
        ax = plt.gca()
        plt.text(0.5, 0.5, 'No Face Detected', family='serif', ha='center')
        ax.axes.xaxis.set_visible(False)
        ax.axes.yaxis.set_visible(False)
        plt.title('Results')
        plt.show()
```

```python
# mask prediction


def mask_prediction(frame, facemodel, maskmodel):
    (height, width) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
                                 (104.0, 177.0, 123.0))
    facemodel.setInput(blob)
    detections = facemodel.forward()
    faces, locs, preds = [], [], []
    for i in range(0, detections.shape[2]):
        confidence = detections[0][0][i][2]
        if confidence > 0.5:
            box = detections[0, 0, i, 3:7] * \
                np.array([width, height, width, height])
            (x1, y1, x2, y2) = box.astype("int")
            (x1, y1) = (max(0, x1), max(0, y1))
            (x2, y2) = (min(width - 1, x2), min(height - 1, y2))
            face = frame[y1:y2, x1:x2]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)
            faces.append(face)
            locs.append((x1, y1, x2, y2))
    if faces:
        faces = np.array(faces, dtype="float32")
        preds = maskmodel.predict(faces, batch_size=32)
    return (locs, preds)
```

```python
# nose detection


def nose_detection(frame, landmarks, color, draw):
    if landmarks.part(30) and landmarks.part(31) and landmarks.part(35) and landmarks.part(33):
        nose_top_x, nose_top_y = landmarks.part(30).x, landmarks.part(30).y
        nose_left_x, nose_left_y = landmarks.part(31).x, landmarks.part(31).y
        nose_right_x, nose_right_y = landmarks.part(35).x, landmarks.part(35).y
        nose_bottom_x, nose_bottom_y = landmarks.part(
            33).x, landmarks.part(33).y
        if draw:
            cv2.circle(frame, (nose_top_x, nose_top_y), 3, color, -1)
            cv2.circle(frame, (nose_left_x, nose_left_y), 3, color, -1)
            cv2.circle(frame, (nose_right_x, nose_right_y), 3, color, -1)
            cv2.circle(frame, (nose_bottom_x, nose_bottom_y), 3, color, -1)
        return True
    return False
```

```python
# mouth detection


def mouth_detection(frame, landmarks, color, draw):
    if landmarks.part(51) and landmarks.part(48) and landmarks.part(54) and landmarks.part(57):
        mouth_top_x, mouth_top_y = landmarks.part(51).x, landmarks.part(51).y
        mouth_left_x, mouth_left_y = landmarks.part(48).x, landmarks.part(48).y
        mouth_right_x, mouth_right_y = landmarks.part(
            54).x, landmarks.part(54).y
        mouth_bottom_x, mouth_bottom_y = landmarks.part(
            57).x, landmarks.part(57).y
        if draw:
            cv2.circle(frame, (mouth_top_x, mouth_top_y), 3, color, -1)
            cv2.circle(frame, (mouth_left_x, mouth_left_y), 3, color, -1)
            cv2.circle(frame, (mouth_right_x, mouth_right_y), 3, color, -1)
            cv2.circle(frame, (mouth_bottom_x, mouth_bottom_y), 3, color, -1)
        return True
    return False
```

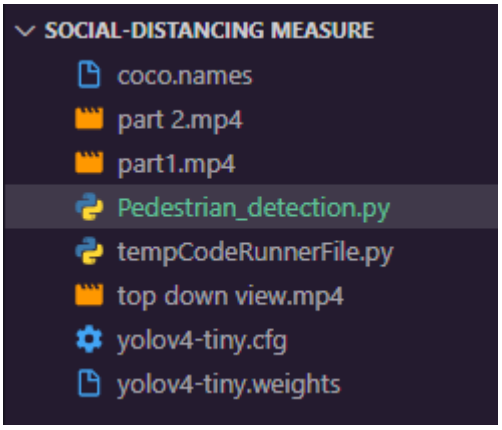
```python
# eye detection


def eye_detection(frame, landmarks, color, draw):
    if landmarks.part(36) and landmarks.part(39) and landmarks.part(42) and landmarks.part(45):
        lefteye_left_x, lefteye_left_y = landmarks.part(
            36).x, landmarks.part(36).y
        lefteye_right_x, lefteye_right_y = landmarks.part(
            39).x, landmarks.part(39).y
        righteye_left_x, righteye_left_y = landmarks.part(
            42).x, landmarks.part(42).y
        righteye_right_x, righteye_right_y = landmarks.part(
            45).x, landmarks.part(45).y
        if draw:
            cv2.circle(frame, (lefteye_left_x, lefteye_left_y), 3, color, -1)
            cv2.circle(frame, (lefteye_right_x, lefteye_right_y), 3, color, -1)
            cv2.circle(frame, (righteye_left_x, righteye_left_y), 3, color, -1)
            cv2.circle(frame, (righteye_right_x,
                               righteye_right_y), 3, color, -1)
        return True
    return False
```

```python
# mask status function


m_on, m_off, m_half = 0, 0, 0
avg_fps = []
test_time = []


def status(mask, withoutMask, nose, mouth, eye):
    global m_on
    global m_half
    global m_off
    if mask > withoutMask and not nose and not mouth:
        label = "Mask on"
        color = (0, 255, 0)
        m_on += 1
    elif (nose or mouth) and withoutMask < 0.9999:
        label = "Mask worn incorrectly"
        color = (255, 255, 0)
        m_half += 1
    else:
        label = "Mask off"
        color = (0, 0, 255)
        m_off += 1
    return (label, color)
```

```python
# main function
def main():
    global start
    print("Starting Face-mask Detection protocol...")
    capture = cv2.VideoCapture(0)
    prev_Time = 0
    start = time.time()
    while True:
        nose, mouth, eye = False, False, False

        _, frame = capture.read()
        frame = cv2.flip(frame, 1)
        locs, preds = mask_prediction(
            frame, facemodel, maskmodel)
        faces = face_detector(frame)

        for face in faces:
            gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
            landmarks = predictor(gray, face)
            nose = nose_detection(frame, landmarks, (255, 0, 255), draw=True)
            mouth = mouth_detection(frame, landmarks, (255, 255, 0), draw=True)
            eye = eye_detection(frame, landmarks, (255, 0, 0), draw=True)
```

```
        for (box, pred) in zip(locs, preds):
            (startX, startY, endX, endY) = box
            (mask, withoutMask) = pred
            #print(mask*100, withoutMask*100)
            label, color = status(mask, withoutMask, nose, mouth, eye)
            cv2.putText(frame, label, (startX, startY - 20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
            confidence = str(max(mask, withoutMask) * 100)[:4]+'%'
            cv2.putText(frame, confidence, (startX, endY + 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)

        prev_Time = fps(frame, prev_Time)
        cv2.imshow("Face-mask recognition system", frame)
        if cv2.waitKey(10) & 0xFF == ord('x'):
            analysis()
            test()
            break

    cv2.destroyAllWindows()
    capture.release()


if __name__ == '__main__':
    main()
```

5.3 Social distancing measure system




```python
from cProfile import label
import numpy as np
import math
import cv2
import imutils
from scipy.spatial import distance as dist
import matplotlib.pyplot as plt
import time

NMS_THRESHOLD = 0.3
MIN_CONFIDENCE = 0.2
MIN_DISTANCE = 40


def distance_function(x1, y1, x2, y2):
    return ((x2 - x1)**2 - (y2-y1)**2)**(0.5)


# test
def test():
    New_Colors = ['green', 'blue', 'purple', 'brown', 'teal']
    # bar graph
    plt.bar(avg_fps, test_time, color=New_Colors)
    plt.title('Social-distancing measure system')
    plt.xlabel('FPS')
    plt.ylabel('Time (s)')
    plt.show()
    # line graph
    plt.plot(test_time, avg_fps)
    plt.xlabel('Time (s)')
    plt.ylabel('FPS')
    plt.show()
```

```python
# pedestrian detection


def pedestrian_detection(image, model, layer_name, personidz=0):
    (H, W) = image.shape[:2]
    results = []
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
                                 swapRB=True, crop=False)
    model.setInput(blob)
    layerOutputs = model.forward(layer_name)
    boxes = []
    centroids = []
    confidences = []
```

```python
    for output in layerOutputs:
        for detection in output:

            scores = detection[5:]
            classID = np.argmax(scores)
            confidence = scores[classID]

            if classID == personidz and confidence > MIN_CONFIDENCE:

                box = detection[0:4] * np.array([W, H, W, H])
                (centerX, centerY, width, height) = box.astype("int")

                x = int(centerX - (width / 2))
                y = int(centerY - (height / 2))

                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
    idzs = cv2.dnn.NMSBoxes(boxes, confidences, MIN_CONFIDENCE, NMS_THRESHOLD)

    if len(idzs) > 0:
        for i in idzs.flatten():
            (x, y) = (boxes[i][0], boxes[i][1])
            (w, h) = (boxes[i][2], boxes[i][3])
            res = [confidences[i], (x, y, x + w, y + h), centroids[i]]
            results.append(res)

    return results
```

```python
# centroid dist visualization


def centroid_visual(image, centroids, draw=False):
    if draw:
        n = len(centroids)
        for i in range(0, n):
            for j in range(i+1, n):
                if distance_function(centroids[i][0], centroids[i][1], centroids[j][0], centroids[j][1]) < MIN_DISTANCE:
                    cv2.line(image, centroids[i],
                             centroids[j], (255, 255, 0), 2)
    else:
        pass
```

```python
def violations(image, right, wrong):
    violation = set()
    total = len(results)

    if total >= 2:
        centroids = np.array([res[2] for res in results])
        n = len(centroids)
        #centroid_visual(image, centroids, True)
        distance = dist.cdist(centroids, centroids, metric='euclidean')
        for i in range(0, distance.shape[0]):
            for j in range(i+1, distance.shape[1]):
                if distance[i][j] < MIN_DISTANCE:
                    violation.add(i)
                    violation.add(j)
```

```python
    for (i, (prob, bbox, centroid)) in enumerate(results):
        (x1, y1, x2, y2) = bbox
        (cx, cy) = centroid
        color = (0, 255, 0)
        if i in violation:
            color = (0, 0, 255)
            wrong += 1
        else:
            right += 1
        cv2.rectangle(image, (x1, y1), (x2, y2),
                      color, 2)
        #cv2.circle(image, (cx, cy), 3, color, 2)

    cv2.putText(image, f'Total: {total}', (20, 130),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
    cv2.putText(image, f'Followers: {right}', (20, 150),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
    cv2.putText(image, f'Deviants: {wrong} ', (20, 170),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

    return image, right, wrong
```

```
labelsPath = "coco.names"
LABELS = open(labelsPath).read().strip().split("\n")
weights_path = "yolov4-tiny.weights"
config_path = "yolov4-tiny.cfg"
model = cv2.dnn.readNetFromDarknet(config_path, weights_path)
'''
model.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
model.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
'''

layer_name = model.getLayerNames()
layer_name = [layer_name[i[0] - 1] for i in model.getUnconnectedOutLayers()]

cap = cv2.VideoCapture("top down view.mp4")
#cap = cv2.VideoCapture("part1.mp4")
#cap = cv2.VideoCapture("part 2.mp4")
```

```
while cap.isOpened():
    _, frame = cap.read()
    frame = imutils.resize(frame, width=700)
    results = pedestrian_detection(frame, model, layer_name,
                                   personidz=LABELS.index("person"))
    frame, right, wrong = violations(frame, 0, 0)
    prev_time = fps(frame, prev_time)
    cv2.imshow("Social Distance Measure", frame)
    if cv2.waitKey(1) & 0xFF == ord('x'):
        analysis(right, wrong)
        test()
        break


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

5.4 Output

Face-mask recognition system

Output-face mask recognition system

Fig. 1.0 Mask On

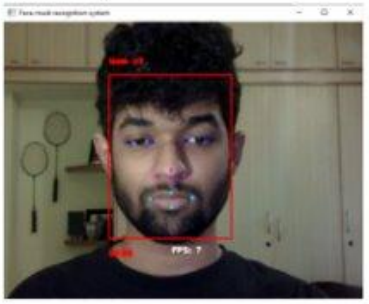


Fig. 2.0 Mask Off

Fig. 3.0 Nose exposed

Fig. 3.1 Nose and Mouth exposed

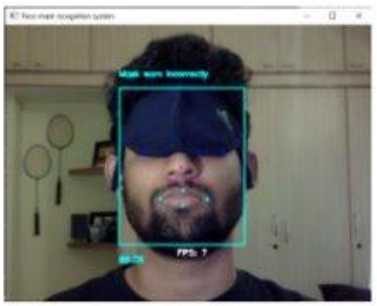
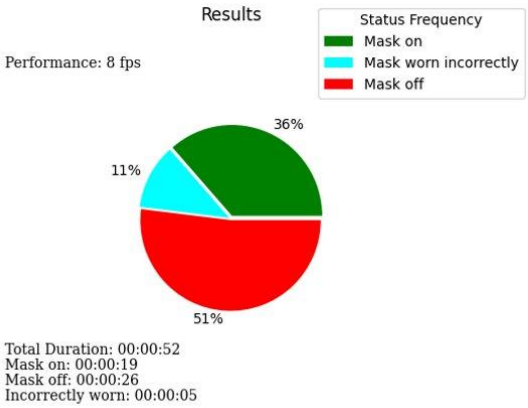


Fig. 3.2 Eyes covered



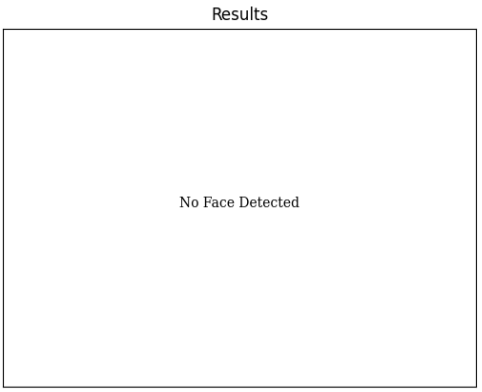
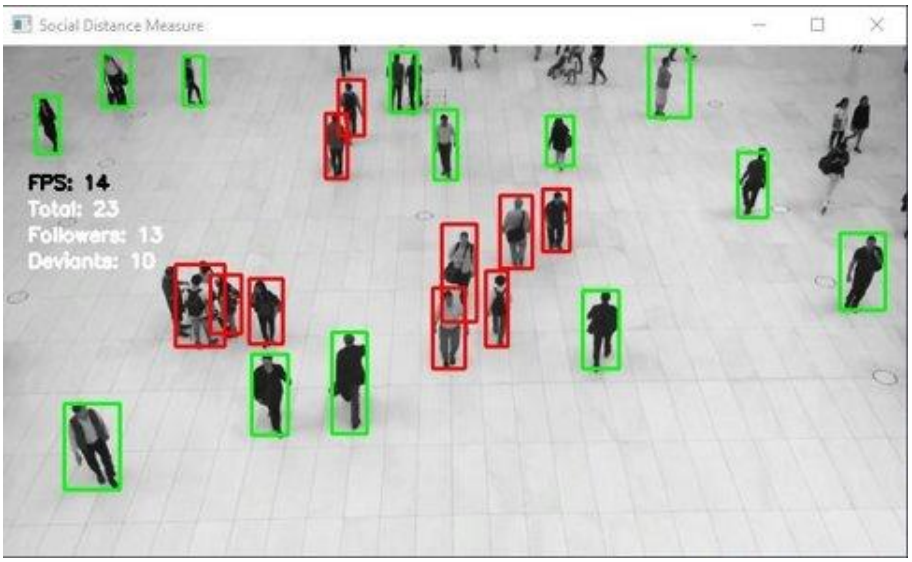


Fig 4 Face-mask Output **Fig 5 No face detected output**

Social-distance measure system



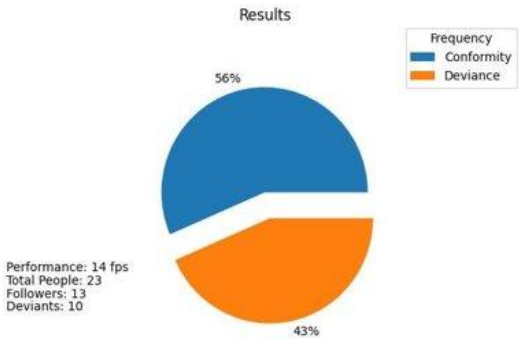


Fig. 6 Social distancing ideal output

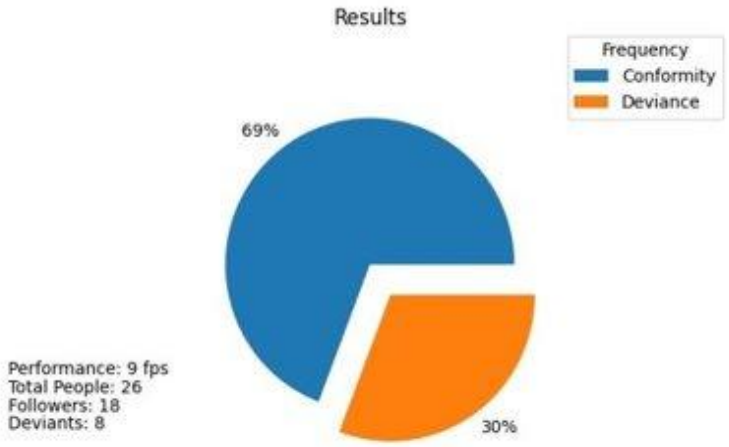
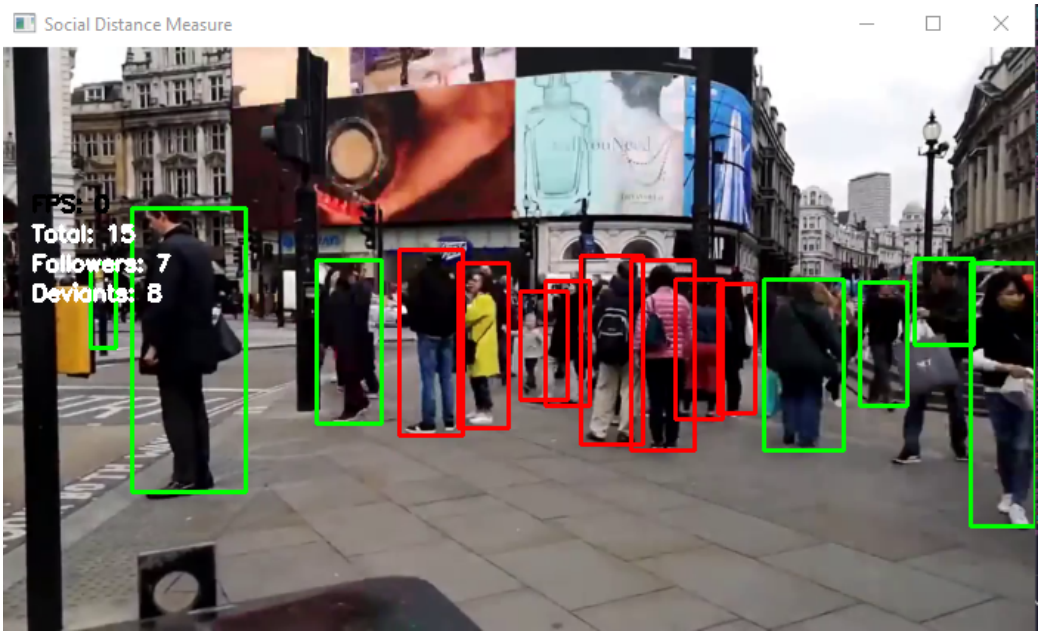


Fig. 7 Algorithm visualization output



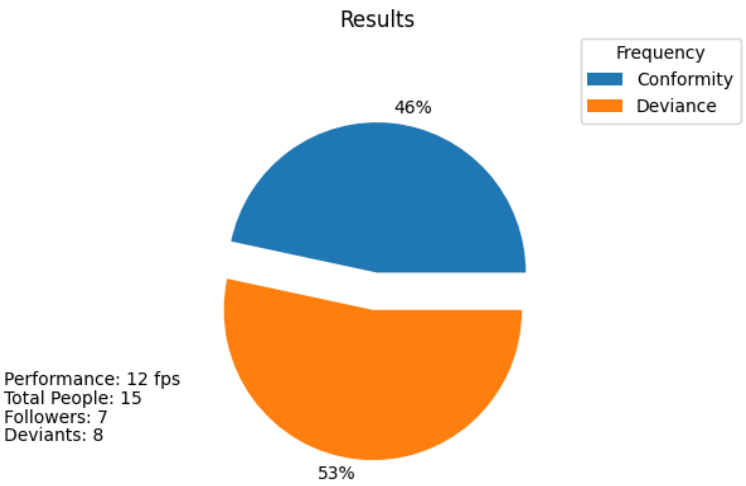


Fig. 8 Eye-level footage output

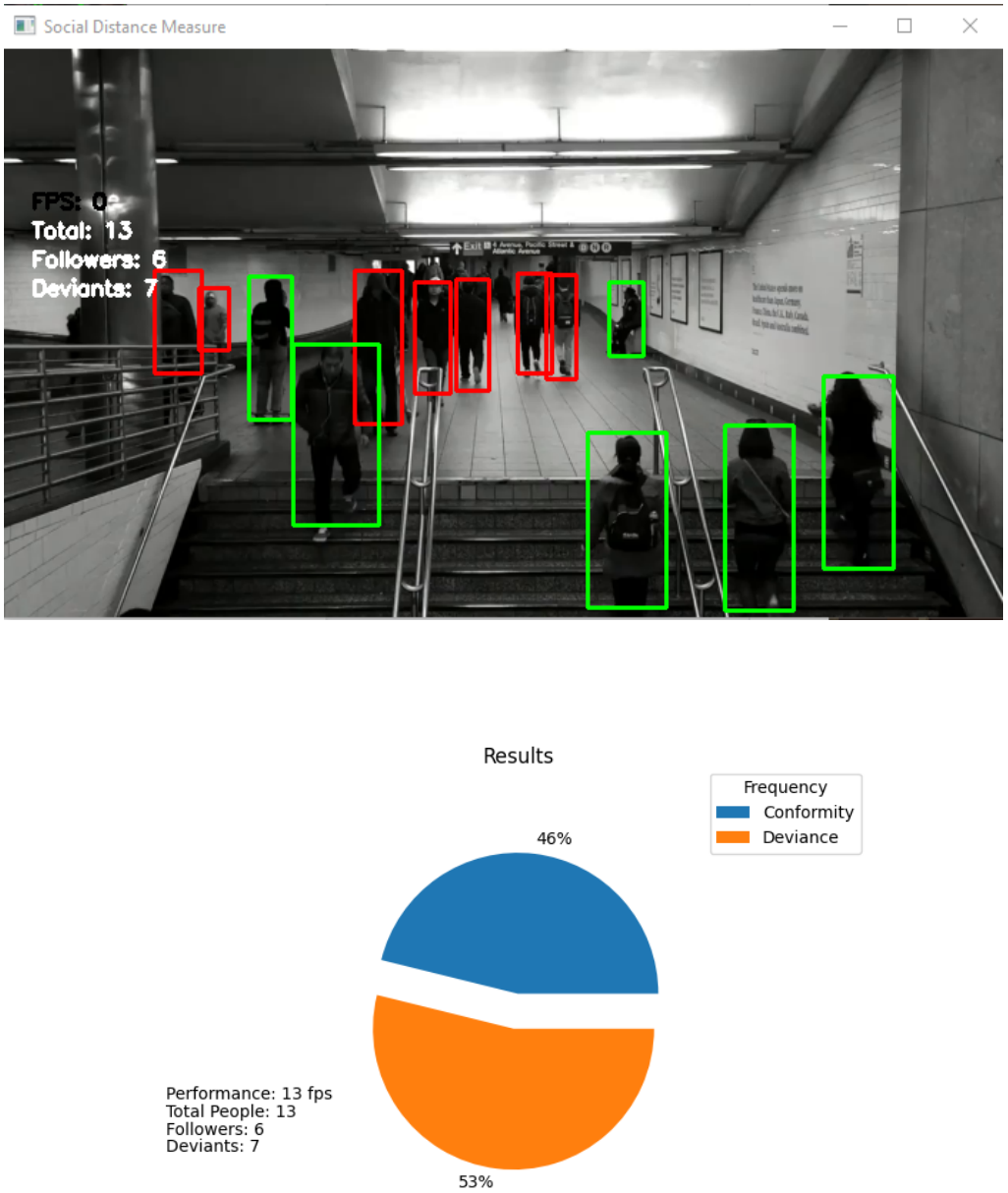


Fig. 9 Train station footage output

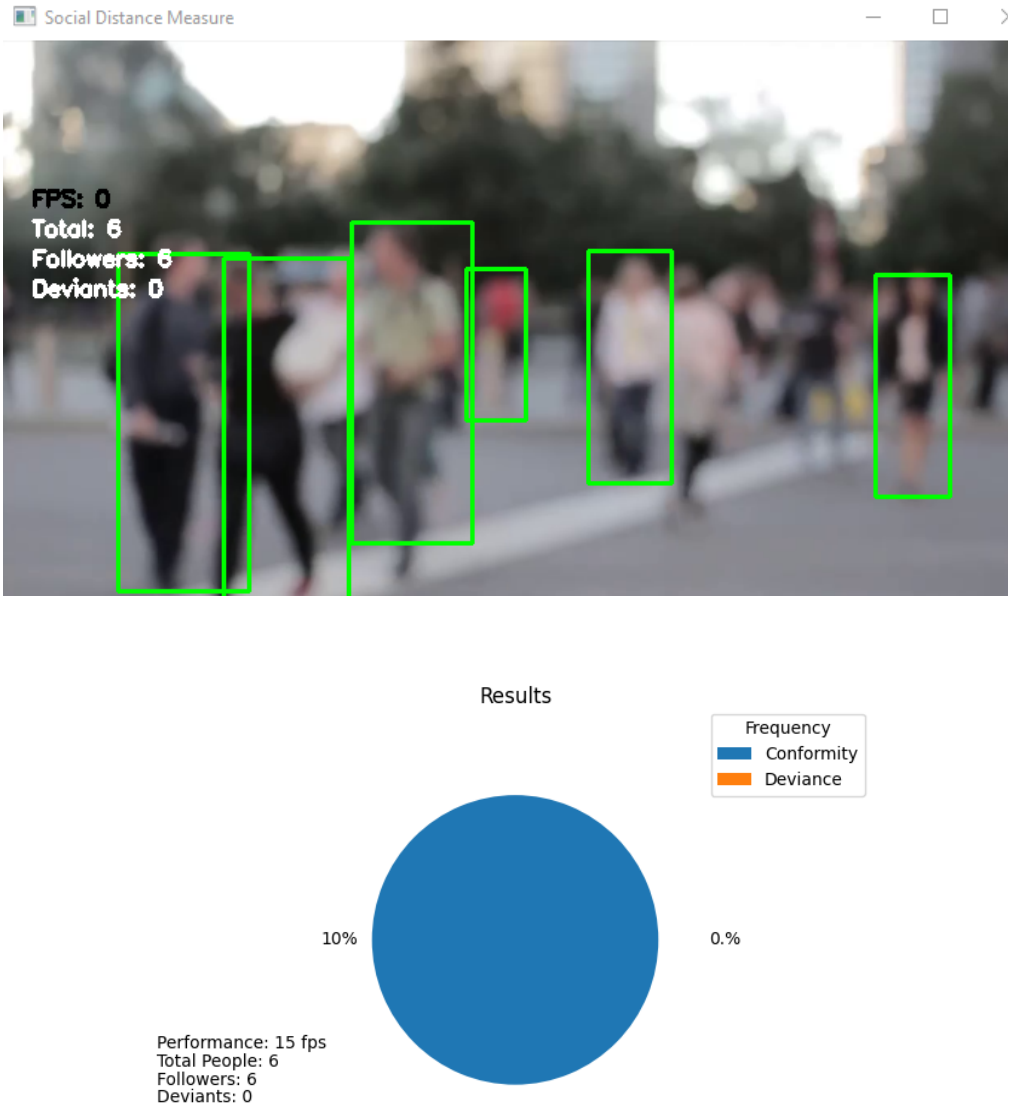


Fig. 10 Eye-level blurred footage output

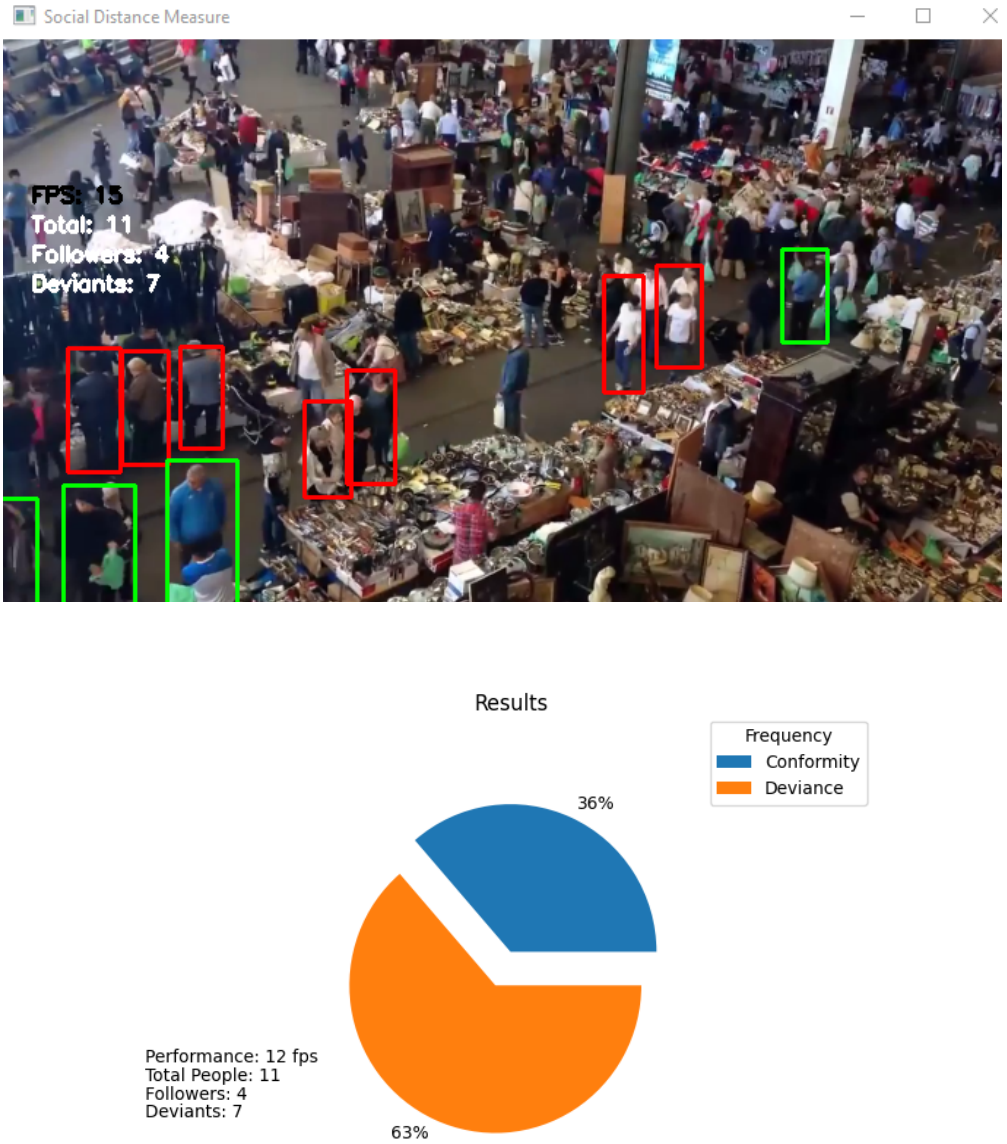


Fig. 11 Uncontrolled environment footage output

Object classification model

Fig. 12 Object classification real-time output

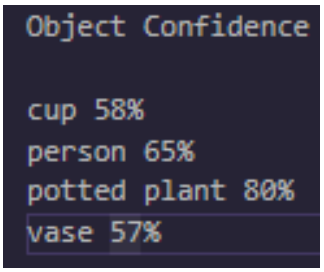


Fig. 12.1 Output Analytics

CHAPTER 6
RESULTS AND DISCUSSIONS

6.1 Findings

Object Classification model

The object classification model serves well as the foundation of all our applications. We are able to consistently detect and sort objects based on the rate of confidence. The current model is able to average 16.0 frames per second on an average system.

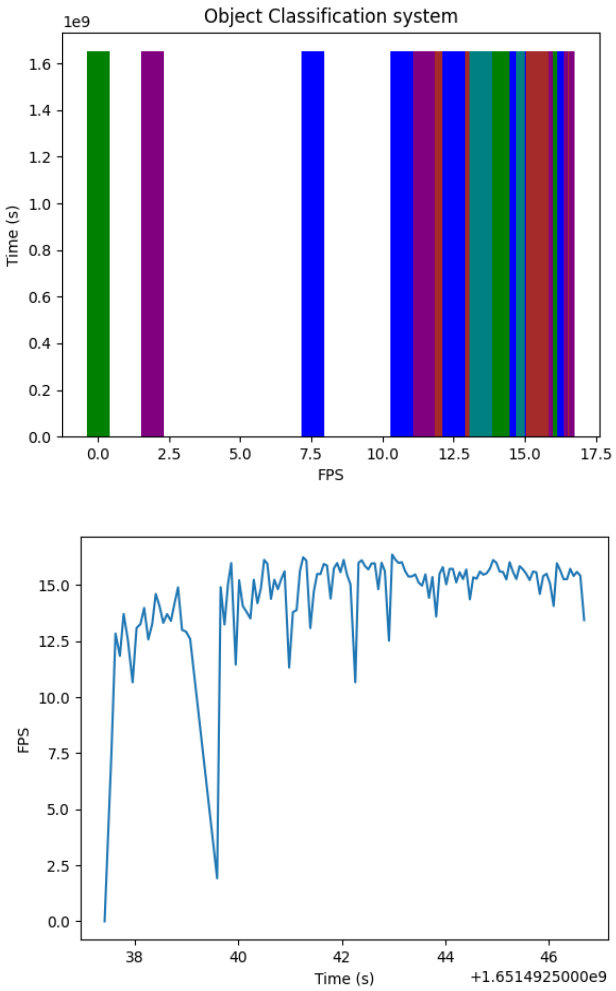
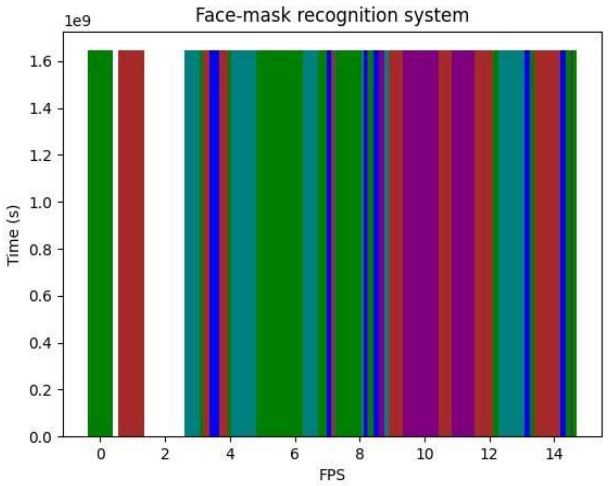


Fig. 13 OCS Performance Analytics

Face-mask recognition system

We can confidently distinguish the different states of a worn face mask, Provide the statistical relationship between them, and pictorially represent them. However, the performance of our system averages at 8.8 fps and has immense fluctuations in the frame rate, which requires further optimization age.



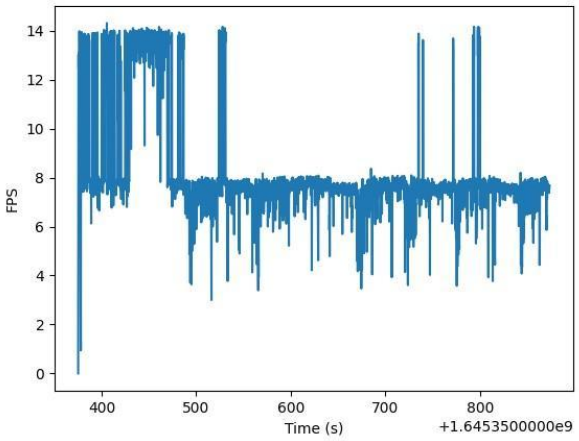
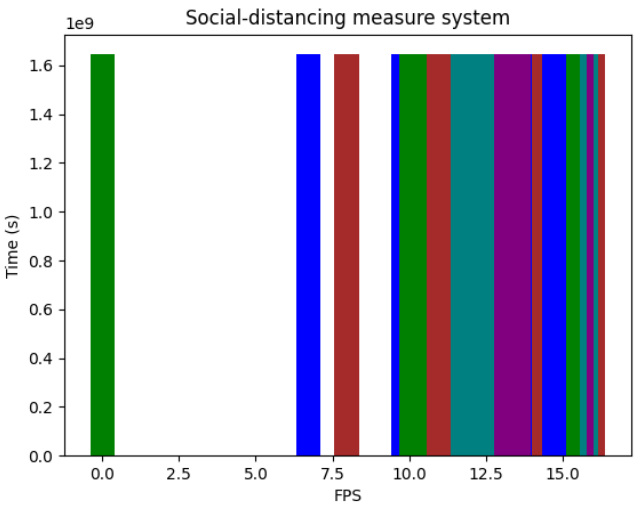


Fig. 14 FMRS Performance Analytics

Social distancing measure system

We can distinguish the different states of pedestrian movement into conformists and deviants accordingly. Provide the statistical relationship between them all and pictorially represent them. However, the performance of our system averages at nine fps and has immense fluctuations in the frame rate, which requires further optimization.



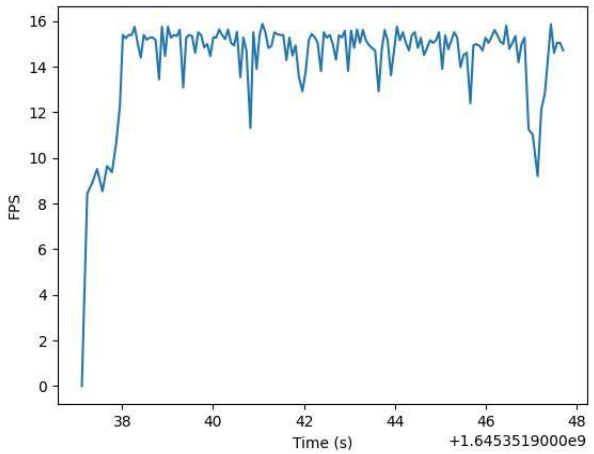


Fig. 15 SDMS Performance Analytics

Combining the results of all our performance tests from our modules, we have produced a tabular comparison of the developed application's performance in similar conditions. Naturally, the object classification model works best as the module's foundation. The face-mask recognition system and social-distancing measure system take performance hits due to adding extra functionalities and trained models.

Fig. 16

S.No	Modules	Average Fps
1.	Object Classification model	16.0 Frames per second
2.	Face-mask Recognition system	8.8 Frames per second
3.	Social-distancing measure system	9.0 Frames per second

Synopsis of proposed models

Hence, we have successfully developed and tested pandemic control measures and analysis methods using computer vision.

6.2 Challenges

Object classification system

- low rates of confidence in detecting objects live
- no trained models for obscure objects
- various case by case methods of sorting objects
- overlap of many things in the frame
- limited lighting and poor resolution confuse object detection

Face-mask recognition system

- Low confidence in mask detection in low-light settings
- The manual method of mask recognition is risky, challenging, and the most widely used of all approaches
- Current similar vision-based methods don't recognize or differentiate the correctness of a worn mask
- Current similar vision-based methods don’t detect multiple faces with stability in real-time
- Current systems don't accurately classify and analyze the acquired data

Social-distancing measure system

- Current systems are hardware intensive and require expensive equipment
- Accurately gauging the distance between people in a 3D plane
- Accurately detecting individuals in high-density crowds
- Counting crowds and classifying them in real-time
- Compatibility with CCTV and other cameras

These are a few of the many obstacles we faced while working on this project. We are committed to discovering more challenges to face and improving upon our existing systems.

CHAPTER 7
CONCLUSION

Due to its software-only nature, It is incredibly cost-effective to layer onto other systems to check for face-mask usage and social-distancing awareness. Our particular system alone can be deployed onto every webcam to every high-rise CCTV camera system to monitor pandemic protocols. We genuinely believe it helps with making informed decisions in a post-pandemic society. From healthcare industries to commercial monopolies, Vision-based object status recognition can always be used to analyze and automate processes for various use-cases. There is immense potential for further improvement with Pandemic control measures and computer vision analysis.

REFERENCES

[1] Chowdary, G. Jignesh, et al. "Face mask detection using transfer learning of inceptionv3." *International Conference on Big Data Analytics*. Springer, Cham, 2020.

[2] Loey, Mohamed, et al. "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic." *Measurement* 167 (2021): 108288.

[3] Nagrath, Preeti, et al. "SSDMNV2: A real-time DNN-based face mask detection system using single shot multibox detector and MobileNetV2." *Sustainable cities and society* 66 (2021): 102692.

[4] Sethi, Shilpa, Mamta Kathuria, and Trilok Kaushik. "Face mask detection using deep learning: An approach to reduce risk of coronavirus spread." *Journal of biomedical informatics* 120 (2021): 103848.

[5] Jiang, Mingjie, Xinqi Fan, and Hong Yan. "Retinamask: A face mask detector." *arXiv preprint arXiv:2005.03950* (2020).

[6] Militante, Sammy V., and Nanette V. Dionisio. "Real-time facemask recognition with alarm system using deep learning." *2020 11th IEEE Control and System Graduate Research Colloquium (ICSGRC)*. IEEE, 2020.

[7] Ayyappa, Yalanati, et al. "Automatic face mask recognition system With FCM AND BPNN." *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2021.

[8] Jiang, Xiaoming, et al. "YOLOv3_slim for face mask recognition." *Journal of Physics: Conference Series*. Vol. 1771. No. 1. IOP Publishing, 2021.

[9] Lionnie, Regina, Catur Apriono, and Dadang Gunawan. "Face mask recognition with realistic fabric face mask data set: A combination using surface curvature and glcm." *2021 IEEE International IoT, Electronics and Mechatronics Conference (IEMTRONICS)*. IEEE, 2021.

[10] Saeidi, Rahim, Ilkka Huhtakallio, and Paavo Alku. "Analysis of Face Mask Effect on Speaker Recognition." *Interspeech*. 2016.

[11] Saponara, Sergio, Abdussalam Elhanashi, and Alessio Gagliardi. "Implementing a real-time, AI-based, people detection and social distancing measuring system for Covid-19." *Journal of Real-Time Image Processing* (2021): 1-11.

[12] Teboulbi, Safa, et al. "Real-Time Implementation of AI-Based Face Mask Detection and Social Distancing Measuring System for COVID-19 Prevention." *Scientific Programming* 2021 (2021).

[13] Yang, Dongfang, et al. "A vision-based social distancing and critical density detection system for COVID-19." *Sensors* 21.13 (2021): 4608.

[14] Xu, Paiheng, Mark Dredze, and David A. Broniatowski. "The Twitter Social Mobility Index: Measuring Social Distancing Practices With Geolocated Tweets." *Journal of medical Internet research* 22.12 (2020): e21499.

[15] Kobayashi, Yutaro, et al. "A system for monitoring social distancing using microcomputer modules on university campuses." *2020 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, 2020.

[16] Courtemanche, Charles, et al. "Strong Social Distancing Measures In The United States Reduced The COVID-19 Growth Rate: Study evaluates the impact of social distancing measures on the growth rate of confirmed COVID-19 cases across the United States." *Health Affairs* 39.7 (2020): 1237-1246.

[17] Fong, Min W., et al. "Nonpharmaceutical measures for pandemic influenza in nonhealthcare settings—social distancing measures." *Emerging infectious diseases* 26.5 (2020): 976.

[18] Ahmed, Faruque, Nicole Zviedrite, and Amra Uzicanin. "Effectiveness of workplace social distancing measures in reducing influenza transmission: a systematic review." *BMC public health* 18.1 (2018): 1-13.

[19] Aquino, Estela ML, et al. "Social distancing measures to control the COVID-19 pandemic: potential impacts and challenges in Brazil." *Ciencia & saude coletiva* 25 (2020): 2423-2446.

[20] Briscese, Guglielmo, et al. *Compliance with covid-19 social-distancing measures in Italy: the role of expectations and duration*. Vol. 27. Cambridge, MA, USA:: National Bureau of Economic Research, 2020.

[21] Chalidabhongse, Thanarat, Panitnat Yimyam, and Panmanas Sirisomboon. "2D/3D vision-based mango's feature extraction and sorting." *2006 9th International Conference on Control, Automation, Robotics, and Vision*. IEEE, 2006.

[22] Gotoh, Toshiyuki, et al. "A flexible vision-based algorithm for a book sorting system." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.3 (1988): 393-399.

[23] Al Ohali, Yousef. "Computer vision-based date fruit grading system: Design and implementation." *Journal of King Saud University-Computer and Information Sciences* 23.1 (2011): 29-36.

[24] Zhihong, Chen, et al. "A vision-based robotic grasping system using deep learning for garbage sorting." *2017 36th Chinese Control Conference (CCC)*. IEEE, 2017.

[25] Zhang, Yawan, and Wei Cheng. "Vision-based robot sorting system." *IOP Conference Series: Materials Science and Engineering*. Vol. 592. No. 1. IOP Publishing, 2019

[26] Cireşan, Dan C., et al. "High-performance neural networks for visual object classification." *arXiv preprint arXiv:1102.0183* (2011).

[27] Javed, Omar, and Mubarak Shah. "Tracking and object classification for automated surveillance." *European Conference on Computer Vision*. Springer, Berlin, Heidelberg, 2002.

[28] Jarvis, J. F., and J. A. Tyson. "FOCAS-Faint object classification and analysis system." *The Astronomical Journal* 86 (1981): 476-495.

[29] Gehler, Peter, and Sebastian Nowozin. "On feature combination for multiclass object classification." *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009.

[30] Deng, Jia, et al. "Large-scale object classification using label relation graphs." *European conference on computer vision*. Springer, Cham, 2014.

APPENDIX A

PLAGIARISM REPORT

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

Office of Controller of Examinations

REPORT FOR PLAGIARISM CHECK ON THE SYNOPSIS/THESIS/DISSERTATION/PROJECT REPORTS

1	Name of the Candidate **(IN BLOCK LETTERS)**	Kavin Chandar Arthanari Eswaran Akshat Prakash Srivastava
2	Address of the Candidate	D3, Ashwin Flats 24, Vembuliamman Koil Street KK Nagar, west Chennai - 78 **Mobile Number :** +91 9789826431
3	Registration Number	RA181003010205, RA1811003010157
4	Date of Birth	20/ 11/ 2000
5	Department	Department of Computer Science and Engineering
6	Faculty	Dr. C. Jothi Kumar
7	Title of the Synopsis/ Thesis/ Dissertation/Project	Pandemic control measures and analysis using computer vision
8	Name and address of the Supervisor / Guide	Dr. Gayathri M **Mail ID :** gayathrm2@srmist.edu.in **Mobile Number :** +91 95000 91731
9	Name and address of the Co-Supervisor / Co- Guide (if any)	**Mail ID :** **Mobile Number :**
10	Software Used	Turnitin
11	Date of Verfication	04/ 05/ 2022

Date : 04/05/2022

12	Plagiarism Details: (to attach the final report)			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self citation)**	**% of plagarism after excluding Quotes, Bibliography, etc.,**
1	Introduction	0%	0%	0%
2	Literature Survey	<1%	<1%	<1%
3	System Architecture and Design	0%	0%	0%
4	Methodology	<1%	<1%	<1%
5	Coding and Testing	0%	0%	0%
6	Results and Discussions	0%	0%	0%
7	Conclusion	0%	0%	0%
8				
9				
10				
Thesis abstract		0%	0%	0%
Appendices		0%	0%	0%
I / We declare that the above information have been verified and found true to the best of my / our knowledge.				
Signature of the Candidate		**SSignature of the Supervisor / Guide**		
Signature of the Co-Supervisor/Co-Guide		**Signature of the HOD**		

Date : 04/ 05/ 2022

final project report

ORIGINALITY REPORT

1%	0%	0%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to California Southern University Student Paper	<1%
2	Yousef Al Ohali. "Computer vision based date fruit grading system: Design and implementation", Journal of King Saud University - Computer and Information Sciences, 2011 Publication	<1%
3	Submitted to City University of Hong Kong Student Paper	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

APPENDIX B
CONFERENCE PRESENTATION

APPENDIX C
PUBLICATION DETAILS