National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Isha Shete
Student ID: 18181678

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljvec

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Isha Shete
Student ID:	18181678
Programme:	Msc in Data Analytics **Year:** 2019-2020
Module:	Research Project
Lecturer:	Vladimir Milosavljvec
Submission Due Date:	17/08/2020
Project Title:	Social Distancing and Face Mask Detection using Deep Learning and Computer Vision
Word Count:	1260 **Page Count:**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Isha Shete
Date:	16/08/2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□

You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Isha Shete
Student ID: 18181678

1. Introduction

The following configuration manual specifies the hardware, the software requirements and the programming codes of the below mentioned research project:

"**Social Distancing and Face Mask Detection using Deep Learning and Computer Vision**"

2. System Configurations

2.1 Hardware

Processor	Intel® Core™ i5-8250U CPU @ 1.60GHz 1.80 GHz
RAM	8GB
Graphics	2GB Nvidia 850M GEFORCE GTX
System Type	Windows OS, 64-bit

Table.1. Hardware Configuration

2.2 Software Requirements

The research project is implemented using deep learning techniques in the Python programming language. Software used for this research are:

- **Google Drive[1]:** It is an online storage and synchronization service that is developed by Google. The files related to this research project is saved on the Drive. The results that are gained are also saved on the Drive.

- **Google Collaboratory[2]:** Also known as Colab, it is a freely available cloud service that helps users with utilizing free GPU services to run machine learning models. Colab is used for implementing the models as well as gaining the results for this research. To enable the GPU settings, from the notebook screen open the Runtime menu. Select the option Change Runtime Type to GPU as shown in Figure 1.

[1] https://drive.google.com/
[2] https://colab.research.google.com/

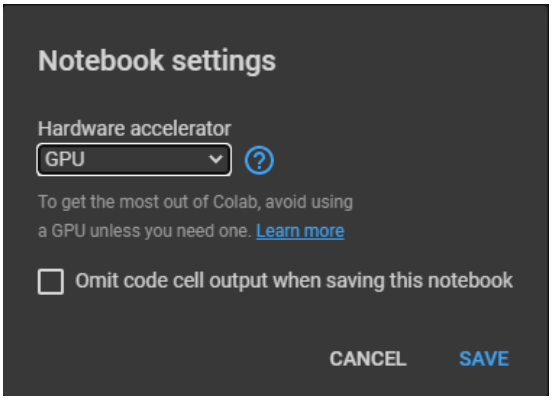

Figure 1: Google Colab: GPU settings

3. Project Development

The project implementation is carried out using the programming language Python in Google Colab. The first stage includes the mounting of Google Drive as shown in Figure 2. This is where all the files are stored and through the drive, this code gets its input and saves the output.

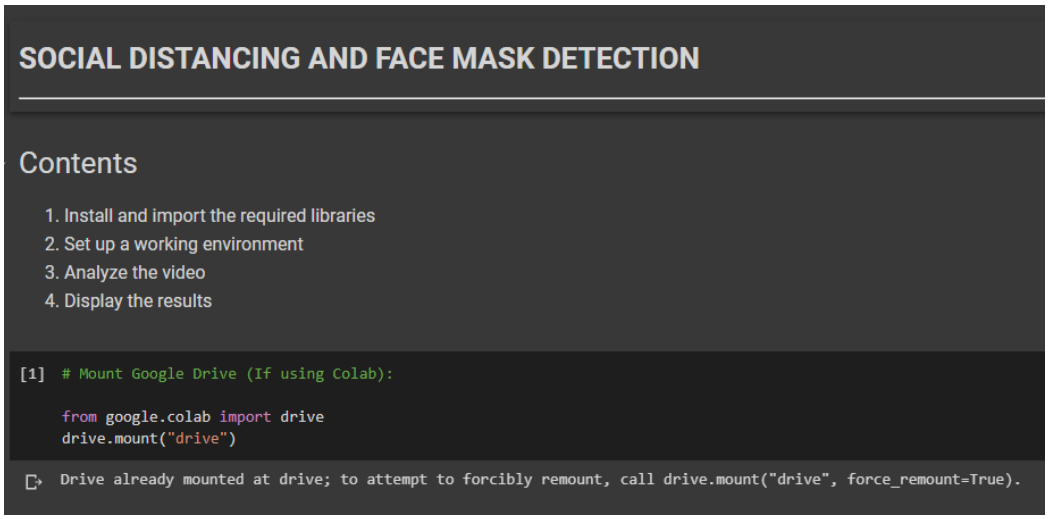

Figure 2: Mounting Google Drive

The next step is to install all the required packages and libraries. Figure 3 shows the packages that need to be installed and imported.

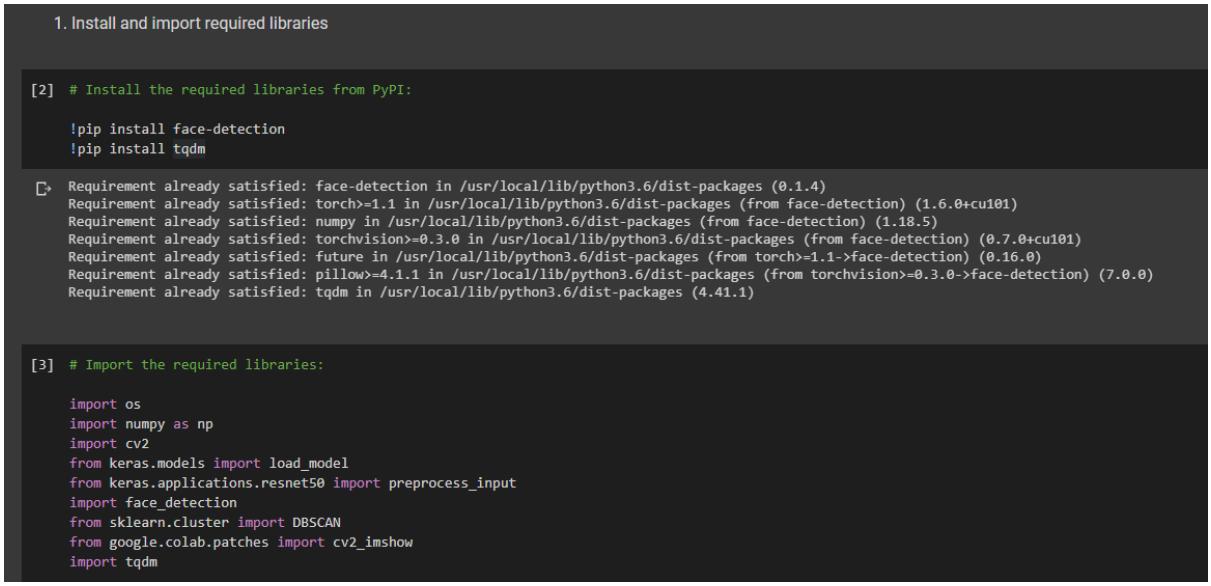

Figure 3: Install and import libraries

Following are the uses of the tools and libraries that are imported as seen in Figure 3:

- Google Colab[2]: Used as the development environment for executing high-end computations on its backend GPUs/TPUs.
- NumPy[3]: Used for storing and manipulating high dimensional arrays.
- OpenCV[4]: Used for manipulating images and video streams.
- Keras[5]: Used for designing and training the Face Mask Classifier model.
- Face_detection[6]: Used for detecting faces with Dual Shot Face Detector.
- Scikit-Learn[7]: Used for DBSCAN clustering.
- Tqdm[8]: Used for showing progress bars.

Setting up and environment is necessary, therefore an environment is created in Google Drive where all the required files are stored. Figure 4 shows that the **BASE_PATH** and the **VIDEO_PATH**s are set. This research project is using a pre-trained face detector, that is, the DSFD Detector[9] and therefore as seen in Figure 4, the detector is downloaded and initialized. A **confidence_threshold** value and an **nms_iou_threshold** value is also initialized. The greater the threshold value, the better the faces are detected. Confidence threshold and NMS IOU thresholds are explained in the Research Report in the Evaluation section.

[3] https://numpy.org/
[4] https://opencv.org/
[5] https://keras.io/
[6] https://github.com/hukkelas/DSFD-Pytorch-Inference
[7] https://scikit-learn.org/stable/
[8] https://github.com/tqdm/tqdm
[9] http://folk.ntnu.no//haakohu/WIDERFace_DSFD_RES152.pth

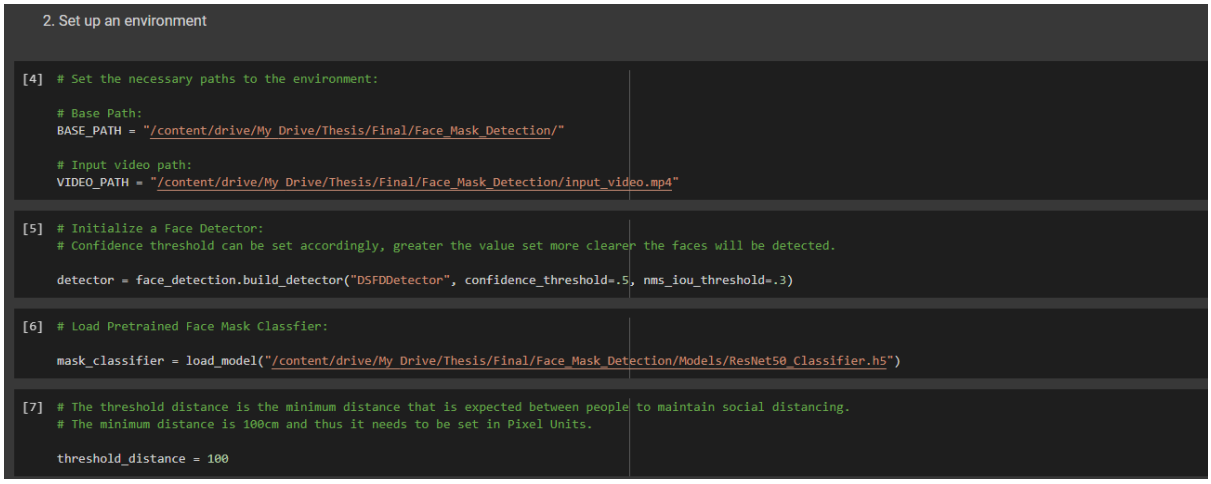

Figure 4: Setting up an environment

Along with the face detector, this project also uses a pre-trained face mask classifier, that is, the ResNet50 Classifier. Thus, the model is downloaded and initialized in the next step. A **threshold_distance** is also seen to be initialized. this distance is nothing but the minimum distance expected between two individuals. this has been set to **100 cm**, that is, 1m, according to the new rules[10].

Analyzing the video is the next step. Here the first step is to initialize the YOLOv3 models, that is, the **yolo.weights**[11] and **yolo.cfg** files. YOLOv3 has 3 output layers (82, 94 and 106). Later, the COCO dataset is loaded and read line by line to get the classes of the objects. **getLayerNames()** gets all the layer names of the network. **getUnconnectedOutLayers()** gets the index of the output layers. With the help of cv2, the video properties like capturing the video, its frames per second rate, its height, width are achieved. Here the number of frames is specified beforehand.

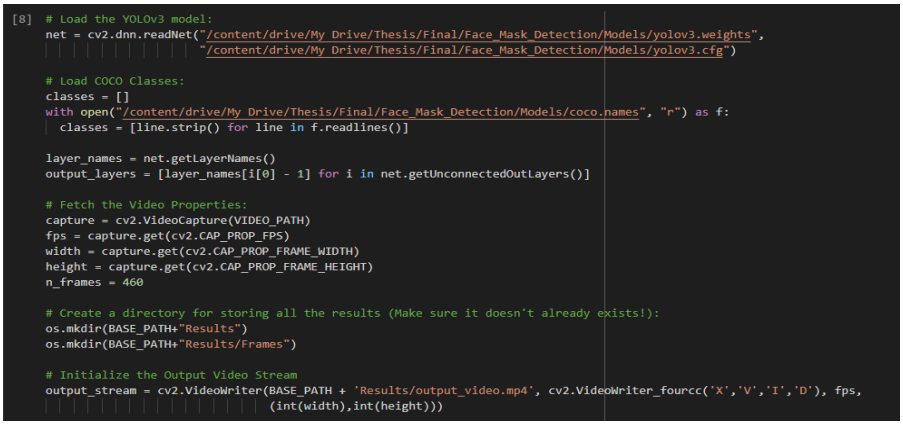

Figure 5: Analyzing the video

Later, an empty directory is created in the root working directory where the results are stored. The output stream is initialized with the help of the function **cv2.VideoWriter**. This

[10] https://www.who.int/westernpacific/emergencies/covid-19/information/physical-distancing
[11] https://pjreddie.com/media/files/yolov3.weights

function contains the filename, the specifics of the output video which is mentioned with the help of fourcc which is a 4-byte code that is utilized to define the video codec, the fps, and the size of the frame. In this project the Windows codec is used, that is, **DIVX** and is passed as **XVID**. Figure 5 depicts these functions.

```
# Process the Frames:

print("Processing Frames : ")

for frame in tqdm.notebook.tqdm(range(int(n_frames))):
  # Capture frame-by-frame:
  ret, image = capture.read()

  # Check EOF:
  if ret == False:
    break;

  # Get frame dimentions:
  height, width, channels = image.shape

  # Detect objects in the frame with YOLOv3:
  blob = cv2.dnn.blobFromImage(image, scalefactor = 0.00392, size = (416, 416), mean = (0,0,0), swapRB = True, crop = False)
  net.setInput(blob)
  outs = net.forward(output_layers)
  class_ids = []
  confidences = []
  boxes = []
```

Figure 6: Processing the frames

- Figure 6 and Figure 6(a) shows how the frames are captured and processed. Here the capturing process takes place from frame-to-frame. The height, width is retrieved. Further, the objects are detected using YOLOv3.

```
# Store the detected objects with Labels, Bounding_Boxes and their Confidences:
for out in outs:
  for detection in out:
    scores = detection[5:]
    class_id = np.argmax(scores)
    confidence = scores[class_id]
    if confidence > 0.5:

      # Get the Center, Height and Width of the Box:
      x_center = int(detection[0] * width)
      y_center = int(detection[1] * height)
      w = int(detection[2] * width)
      h = int(detection[3] * height)

      # Topleft Co-ordinates
      x = int(x_center - w / 2)
      y = int(y_center - h / 2)

      boxes.append([x, y, w, h])
      confidences.append(float(confidence))
      class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
```

Figure 6(a): Processing the frames

```
# Initialize empty lists for storing Bounding Boxes of People and their Faces:
persons = []
masked_faces = []
unmasked_faces = []

# Work on Detected Persons in the Frame:
for i in range(len(boxes)):
  if i in indexes:
    box = np.array(boxes[i])
    box = np.where(box<0,0,box)
    (x, y, w, h) = box

    label = str(classes[class_ids[i]])
    if label == "person":
      persons.append([x,y,w,h])
      # Detect Face in the Person
      person_rgb = image[y:y+h,x:x+w,::-1]   # Crop & BGR to RGB
      detections = detector.detect(person_rgb)

      # If a Face is Detected
      if detections.shape[0] > 0:
        detection = np.array(detections[0])
        detection = np.where(detection < 0, 0, detection)

        # Calculating Co-ordinates of the Detected Face
        x1 = x + int(detection[0])
        x2 = x + int(detection[2])
        y1 = y + int(detection[1])
        y2 = y + int(detection[3])
```

Figure 7: Person Detection

- In Figure 7, empty lists are created so the calculated values can be stored in it. Here the person detection takes place. A bounding box is created that surrounds the person that has been detected. After that, face detection takes place.
- Figure 8 shows how the faces are detected and then classified into two lists, masked and unmasked faces.

```
        try :

          # Crop & BGR to RGB
          face_rgb = image[y1:y2, x1:x2, ::-1]

          # Preprocess the Image
          face_arr = cv2.resize(face_rgb, (224, 224), interpolation = cv2.INTER_NEAREST)
          face_arr = np.expand_dims(face_arr, axis = 0)
          face_arr = preprocess_input(face_arr)

          # Predict if the Face is Masked or Not
          score = mask_classifier.predict(face_arr)

          # Determine and store Results
          if score[0][0] < 0.5:
            masked_faces.append([x1, y1, x2, y2])
          else:
            unmasked_faces.append([x1, y1, x2, y2])
        except:
          continue

# Calculate Coordinates of People Detected and find Clusters using DBSCAN
person_coordinates = []
```

Figure 8: Face Mask Detection

- The social distancing detection is carried out in Figure 9. Here, with the help of **DBSCAN**, clusters are detected. The midpoints of the bounding boxes of the people are considered. The distance between one midpoint and the other is calculated. If the distance calculated is less than 100cm, then a red bounding box is created, else a green bounding box is created.

```
for p in range(len(persons)):
    person_coordinates.append((persons[p][0] + int(persons[p][2] / 2), persons[p][1] + int(persons[p][3] / 2)))
clustering = DBSCAN(eps = threshold_distance, min_samples=2).fit(person_coordinates)
isSafe = clustering.labels_

# Count
person_count = len(persons)
masked_face_count = len(masked_faces)
unmasked_face_count = len(unmasked_faces)
safe_count = np.sum((isSafe == -1) * 1)
unsafe_count = person_count - safe_count

# Show Clusters using Red Lines
arg_sorted = np.argsort(isSafe)

for i in range(1,person_count):
    if isSafe[arg_sorted[i]] != -1 and isSafe[arg_sorted[i]] == isSafe[arg_sorted[i-1]]:
        cv2.line(image, person_coordinates[arg_sorted[i]], person_coordinates[arg_sorted[i - 1]], (0, 0, 255), 2)

# Put Bounding Boxes on People in the Frame
for p in range(person_count):
    a, b, c, d = persons[p]

    # Green if Safe, Red if UnSafe
    if isSafe[p] == -1:
        cv2.rectangle(image, (a, b), (a + c, b + d), (0,255,0), 2)
    else:
        cv2.rectangle(image, (a, b), (a + c, b + d), (0,0,255), 2)
```

Figure 9: Social Distancing

- Figure 10 is where the total number of people, safe or unsafe, wearing a mask or not, is displayed in a box.

```
# Green if Safe, Red if UnSafe
for f in range(masked_face_count):
    a, b, c, d = masked_faces[f]
    cv2.rectangle(image, (a, b), (c, d), (0, 255, 0), 2)

for f in range(unmasked_face_count):
    a, b, c, d = unmasked_faces[f]
    cv2.rectangle(image, (a, b), (c, d), (0, 0, 255), 2)

# Show Monitoring Status in a Black Box at the Top
cv2.rectangle(image, (0, 0), (width, 50), (0, 0, 0), -1)
cv2.rectangle(image, (1, 1), (width - 1, 50), (255, 255, 255), 2)

xpos = 15

string = "Total People = "+str(person_count)
cv2.putText(image,string,(xpos,35),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2)
xpos += cv2.getTextSize(string,cv2.FONT_HERSHEY_SIMPLEX,1,2)[0][0]

string = " ( "+str(safe_count) + " Safe "
cv2.putText(image,string,(xpos,35),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2)
xpos += cv2.getTextSize(string,cv2.FONT_HERSHEY_SIMPLEX,1,2)[0][0]

string = str(unsafe_count)+ " Unsafe ) "
cv2.putText(image,string,(xpos,35),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
xpos += cv2.getTextSize(string,cv2.FONT_HERSHEY_SIMPLEX,1,2)[0][0]

string = "( " +str(masked_face_count)+" Masked "+str(unmasked_face_count)+" Unmasked "+\
        str(person_count-masked_face_count-unmasked_face_count)+" Unknown )"
cv2.putText(image,string,(xpos,35),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,255),2)
```

Figure 10: Creating an Output Box

- And lastly, Figure 11 shows the successful implementation of the code and where the output video is saved.

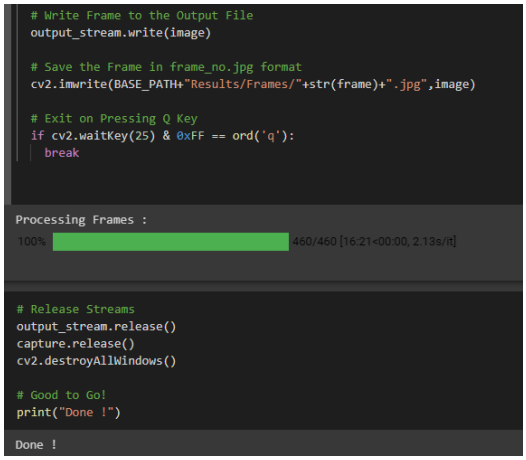

Figure 11: Save the results