National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Olaomopo Bandele
Student ID: x21118388

School of Computing
National College of Ireland

Supervisor: Bharat Agarwal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: OLAOMOPO BANDELE

Student ID: X21118388

Programme: MSc Data Analytics **Year:** 2022

Module: MSc Research Projects

Lecturer: Bharat Agarwal

Submission Due Date: 15th December 2022

Project Title: Real-Time Drowsiness Detection

Word Count: **1230** **Page Count: 13**

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: Olaomopo Bandele

Date: 14th December 2022

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

Olaomopo Bandele
Student ID: x21118388

1 Introduction

This configuration manual explains and illustrates the steps needed to build real-time sleepiness detection using deep learning and computer vision models. Additionally, this manual's several sections discuss the necessary libraries, and machine specifications needed to accurately reproduce the research.

2 System Configuration

The hardware and software configurations that were used to carry out this project are listed below.

2.1 Hardware Specifications

The following hardware setups were used for this project implementation:

Hardware	**Configurations**
RAM	8GB
Hard disk	512GB (SSD)
Graphics card	Intel (R) Iris(R) Xe Graphics
Processor	INTEL 11TH generation CORE i7

Table 1: Hardware specifications

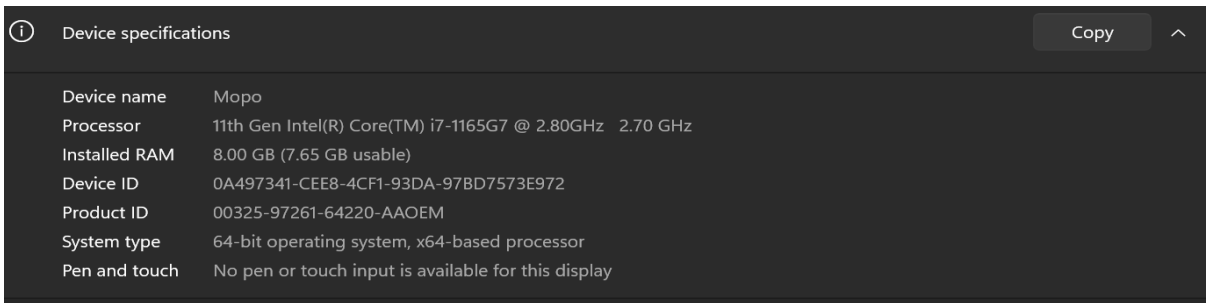

Figure 1: Operating system Configurations

The operating system used for this research is Windows 11 with an 8GB RAM

2.2 Software Specifications

The software utilized in this project, is listed in Table 2.

Software	Configurations
Operating system	Windows 11 (64bit)
IDE	Jupiter Notebook (Anaconda navigator)
Programming language	Python
Programing language version	Python 3.7

Table 2: Software Specifications

2.2.1 Integrated Development Environment

To train the models and use the Jupiter Notebook environment, a recent version of Anaconda was installed. The steps taken to install Anaconda is stated below.

- Step 1

  Visit the anaconda website and click on the download icon.

Figure 2: Anaconda download page

- Install it into your system and click next to go further in the installation process

- Agree to the licence of agreement and click next

- Select "just me" for the installation type as seen in figure 3

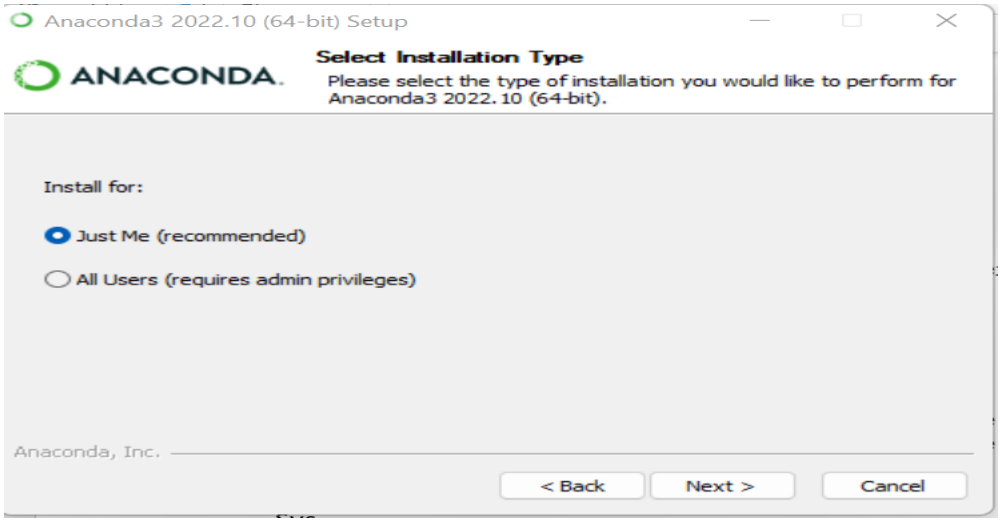

Figure 3: Anaconda installation Process

- Select the installation location

- The installation will begin and will take roughly 30 seconds to 5 minutes

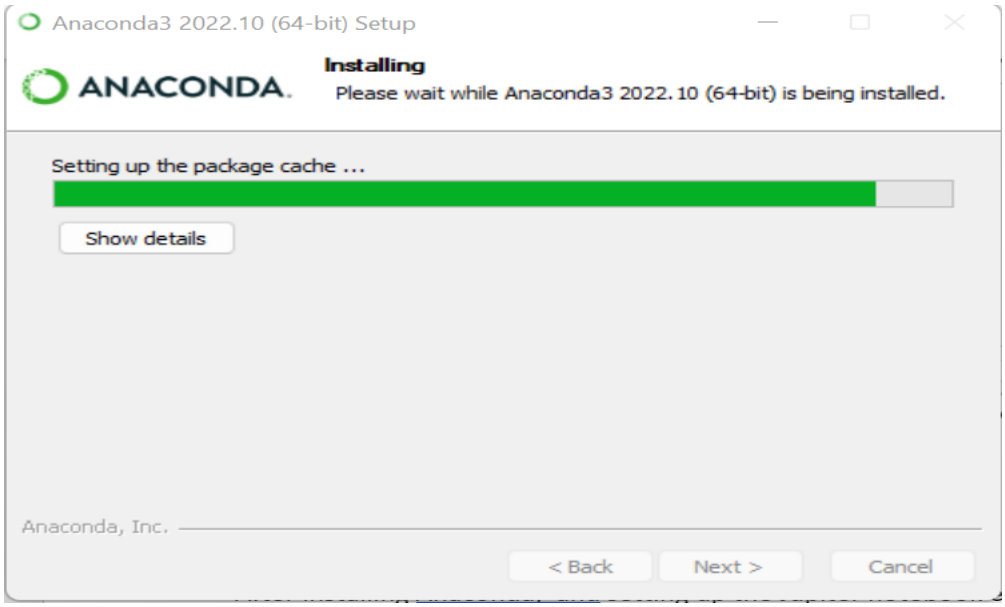

Figure 4: Anaconda installing

- Once the installation has been completed a prompt like that in figure 5 will be displayed.

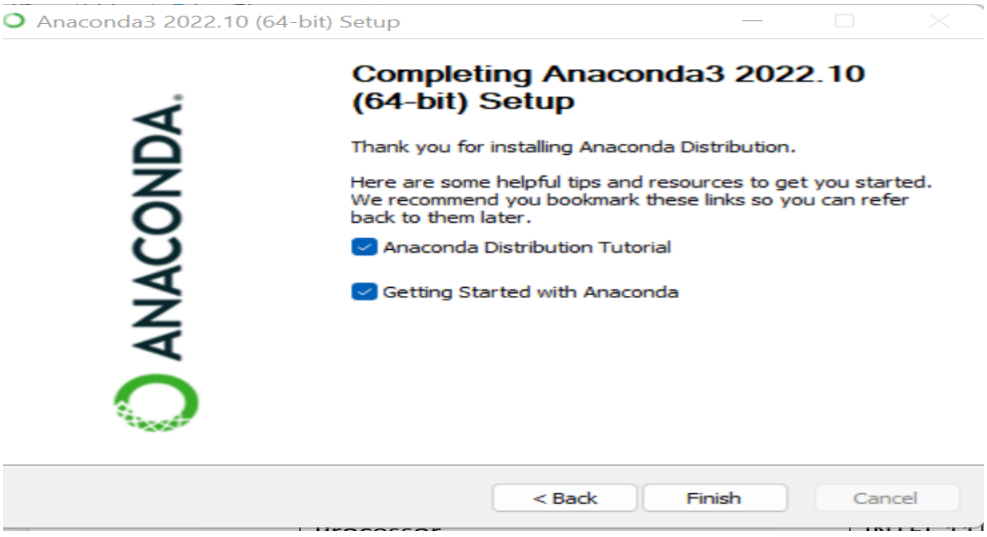

Figure 5: Anaconda installation complete

- Click on finish and it will open a website
- Close the website opened and click on your windows button (located on the left-hand side for windows 11) and search for Anaconda
- Click on the anaconda icon, and an interface like that in figure 6 will be opened.

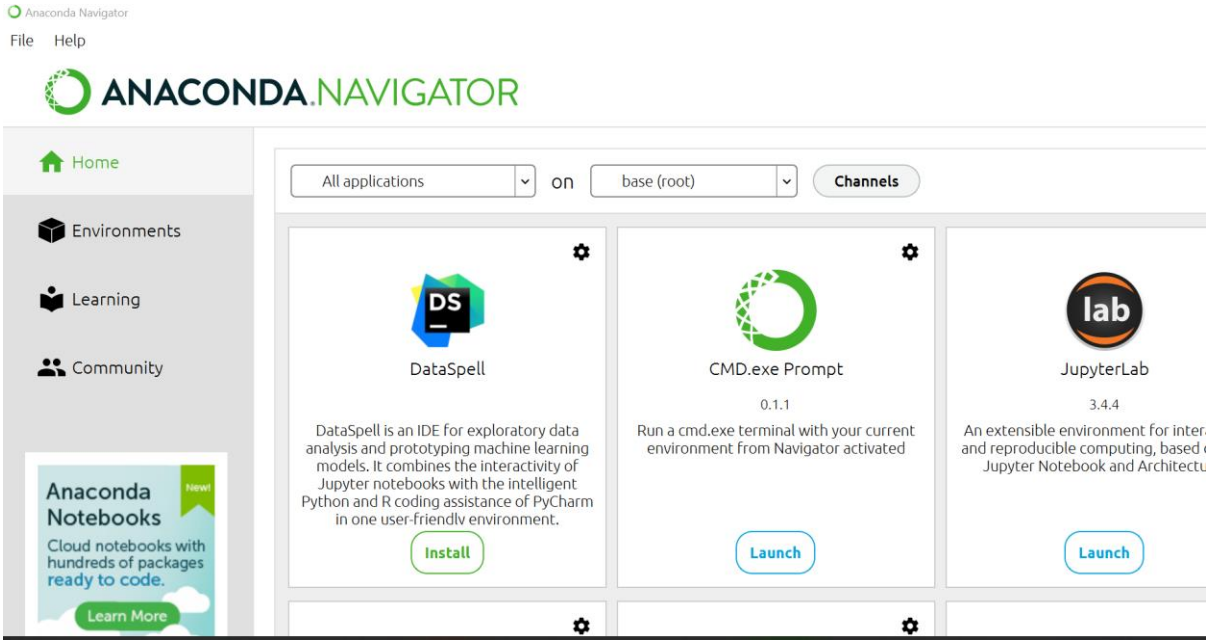

Figure 6: Anaconda Interface

- Search for Jupiter notebook and click on launch

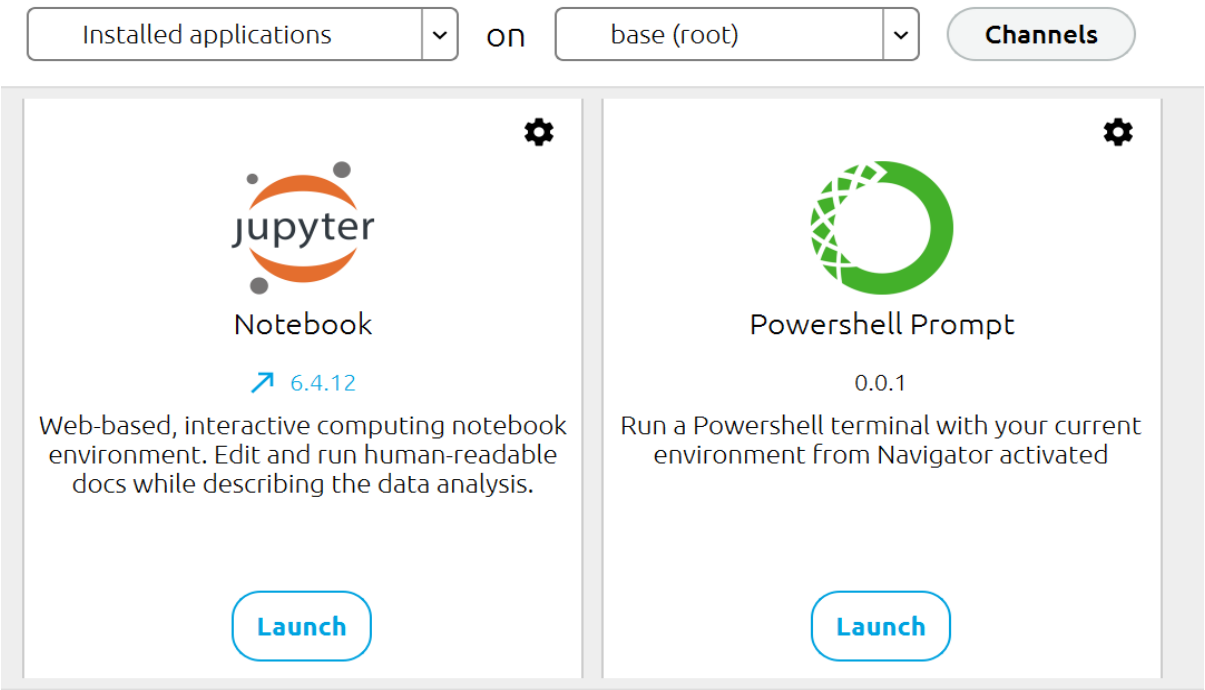

Figure 7: Jupiter Notebook in Anaconda environment

- Launching Jupiter notebook will open a website interface

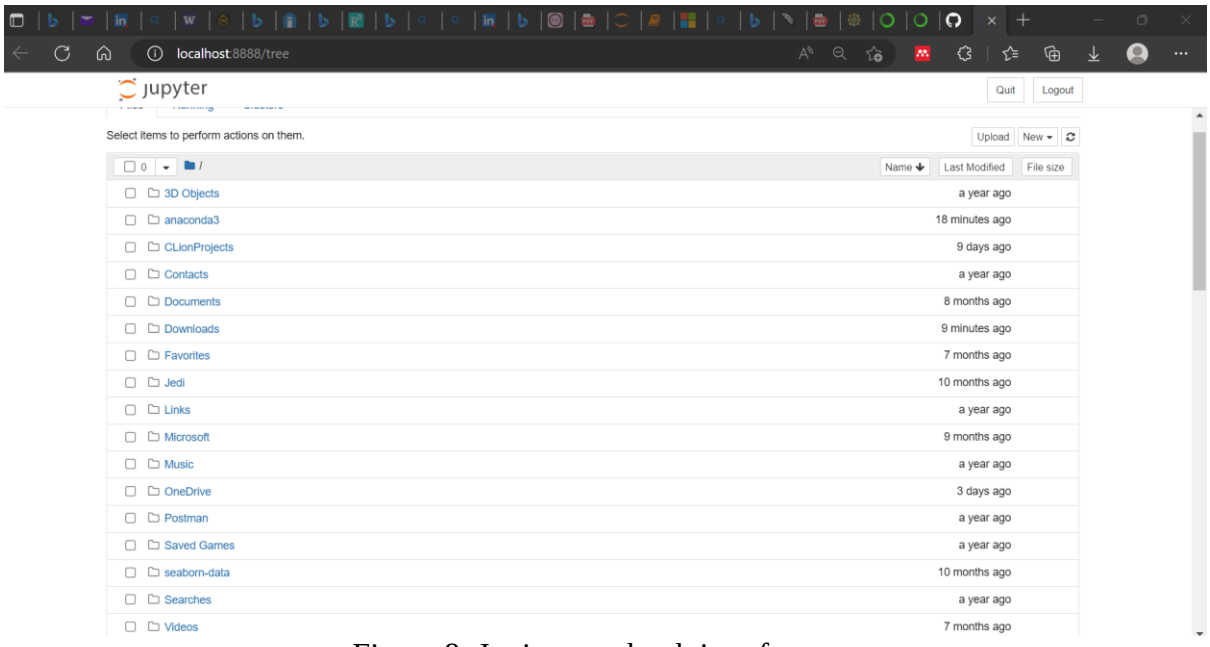

Figure 8: Jupiter notebook interface

- Click on new at the top right-hand corner in the Jupiter notebook environment.

- A new notebook will open and codes can be written there.

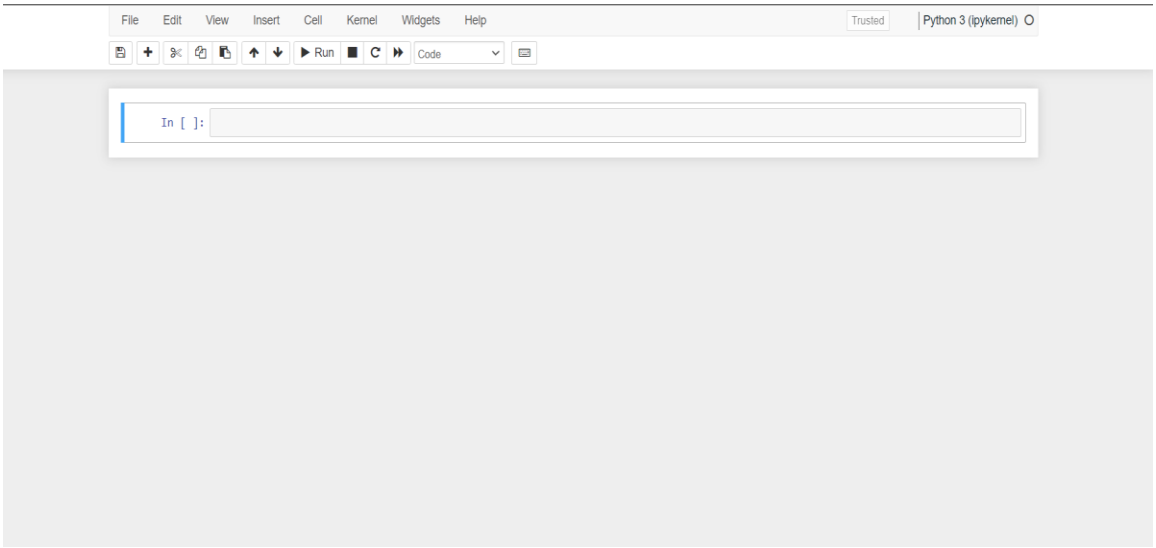

Figure 9: Jupiter Notebook

2.3 Libraries

After installing Anaconda, and setting up the Jupiter notebook environment, essential libraries needed to carry out the research project were installed, and these libraries are stated below:

Os	Cv2
Sys	Shutil
Random	NumPy
Pandas	Sklearn
Matplotlib.pyplot	Datetime
Tensorflow	Keras
Face recognition	Cmake
Dlib	PIL
Split- folders	Tensorflow.Keras.layers
sklearn.metrics	tensorflow.keras.preprocessing

3 Data Handling

Two datasets were used for the successful completion of this project

3.1 Data Collection

Drowsiness Detection 1: This dataset includes low- and high-resolution infrared photos that were all taken under varied lighting circumstances and with various cameras. It contains photos of 37 distinct people, 33 men and 4 women, with resolutions ranging from 640 x 480 to 1280 x 1024 and 752 x 480. A total of 84,898 photos from two distinct classes (closed and open eyes) are included in the dataset, with each class containing half of the overall number of photos.

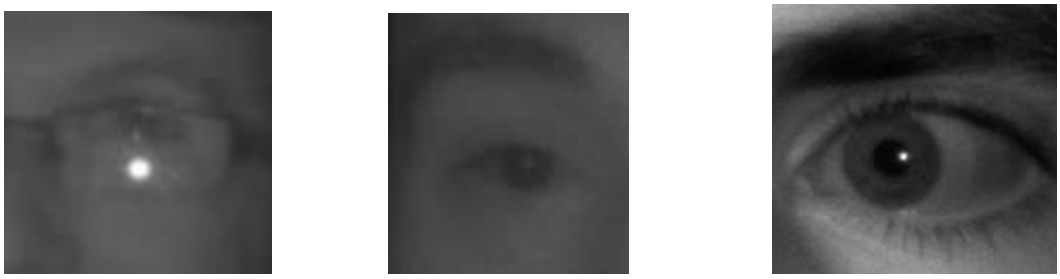

Figure 10: Drowsiness detection Images

Drowsiness Detection 2: This dataset contains coloured images of people's eyes. 1452 photos altogether, 726 of which featured eyes that were closed and 726 open-eye images. Each image had a different size, ranging from 86 x 86 to 244 x 244 for the largest.

Figure 11: Drowsiness detection Images (Dataset2)

3.2 Data Merging

The dataset utilized for the study was gotten from two different sources, therefore utilizing a Python OS module, the data from Dataset 1 was retrieved and then transferred to Dataset 2 folder.

```
In [20]: # Providing the folder path of opened eyes
trg = r"C:\Users\bande\Downloads\Master theisis\train\Open"   #images will be retrived from here (dataset1)
src = r"C:\Users\bande\Downloads\Master theisis\open_eye"     #new location the file will be copied to (dataset 2)

# Fetching the list of all the files
files = os.listdir(src)

for fname in files:

    # copying the files to the
    # destination directory
    shutil.copy2(os.path.join(src,fname), trg)  #coying the file thr folder path stated above



# Providing the folder path of closed eyes
trgg = r"C:\Users\bande\Downloads\Master theisis\train\Closed"        #images will be retrived from here (dataset1)
srcc = r"C:\Users\bande\Downloads\Master theisis\closed_eye"          #new location the file will be copied to (dataset 2)

# Fetching the list of all the files
file = os.listdir(srcc)

for fnam in file:

    # copying the files to the
    # destination directory
    shutil.copy2(os.path.join(srcc,fnam), trgg)             #coying the file thr folder path stated above
```

Figure 12: Code snippet for data merging

3.3 Splitting of Data into Folders

The "split folders" library in figure 13 was used to divide the data into train validation and test and store them in memory at a ratio of 70 for training, 20 for validation, and 10 for test,

ensuring that the images used to test and validate the machine learning model had never been seen before.

```
In [21]: #since the data is located in just 2 folders totled drowsy and non drowsy, they will be splitted into three sets
         #train test and validation
         import splitfolders # a python package to splitdata

In [22]: input_folder = r"C:\Users\bande\Downloads\Master theisis\train"        #location of the data
         output = r"C:\Users\bande\Downloads\Master theisis\train"            #location of the split data

In [23]: splitfolders.ratio(input_folder, output=output, seed=42, ratio=(.7, .2, .1)) #to split the data into train test and validation se

         Copying files: 49452 files [05:47, 142.37 files/s]
```

Figure 13: Code snippet for data splitting

4 Image Pre-processing

4.1 Image Processing

The dataset has undergone pre-processing to identify drowsiness while accounting for various viewing angles and lighting conditions. Greyscale conversion, rescaling, and rotation are all parts of the pre-processing techniques and are very important when training models to avoid issues with overfitting (Wang, 2017). The photos used to train the SVM model merely underwent greyscale conversion and rescaling, however, the images used to train the CNN model underwent each of the aforementioned pre-processing methods. The image processing techniques shown in figure 6 will provide the model with more different images during training.

```
In [348]: X = []
          Y = []
          for cls in classes:
              pth = "C:/Users/bande/Downloads/Master theisis/train/train/"+ cls
              for j in os.listdir(pth):
                  img = cv2.imread(pth+'/'+j, 0)
                  eye_img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)          #convert all images to colored
                  eye_img = cv2.cvtColor(eye_img, cv2.COLOR_BGR2GRAY)      #convert all colored images back to greyscale
                  eye_img = cv2.resize(eye_img, (75,75))                   # resize images to 75 by 75
                  X.append(eye_img)
                  Y.append(classes[cls])
```

IMAGE PREPROCESSING FOR CNN

```
In [219]: datagen = ImageDataGenerator(rescale=1./255,              #Reduce image pixel to be within a uniform size of 0 and 1
                                       rotation_range=20,           #Rotate the images by 20 degrees
                                      zoom_range=0.2,               # Zoom image by 20 degrees
                                      horizontal_flip=True)         #flip the images
          Augmentation = datagen.flow_from_directory(r"C:\Users\bande\Downloads\Master theisis\train\train", target_size=(75,75),
                                                     shuffle=True,  color_mode="grayscale", class_mode = "binary")

          Found 34616 images belonging to 2 classes.
```

Figure 14: Image preprocessing

5 Model Building

The project is implemented using a deep algorithm (CNN) and a machine learning algorithm, (SVM), Scikit-Python library is used to design the machine learning algorithm (SVM) and the CNN model was developed using Keras and Tensorflow.

5.1 Convolutional Neural Network Model (CNN)

```
In [336]: model = Sequential()
          #Layers of the CNN model
          model.add(Conv2D(128, (3,3), 1, activation='relu', input_shape=(75,75,1), padding='same'))
          model.add(MaxPooling2D(2,2))
          model.add(Dropout(0.3))
          model.add(Conv2D(128, (3,3), 1, activation='relu',  padding='same'))
          model.add(MaxPooling2D(2,2))

          model.add(Conv2D(128, (3,3), 1, activation='relu', padding='same'))
          model.add(MaxPooling2D(2,2))


          model.add(Conv2D(32, (3,3), 1, activation='relu',))



          model.add(Flatten())
          model.add(Dense(124, activation='relu'))

          model.add(Dropout(0.3))

          model.add(Dense(1, activation='sigmoid'))
```

Figure 15: CNN model building

5.2 Support Vector Machine Model

```
In [357]: from sklearn.svm import SVC
```

```
In [358]: sv = SVC(C= 1,gamma = 0.001,kernel = 'rbf')
          start = dt.now()     #begin to recod time of training


          sv.fit(xtrain, ytrain)
          running = (dt.now() - start).seconds          #end time of training

          print("Total time of training SVM model:" ,running)

          Total time of training SVM model: 1062
```

Figure 16: SVM model building

6 Realtime Testing

After training the CNN model, it was tested in real-time using the webcam of a computer The code snippet to achieve this can be seen in Figure 17 - 20.

```
# webcam frame is inputted into function
def eye_cropper(frame):

    # create a variable for the facial feature coordinates
    facial_features_list = face_recognition.face_landmarks(frame)

    # create a placeholder list for the eye coordinates
    # and append coordinates for eyes to list unless eyes
    # weren't found by facial recognition
    try:
        eye = facial_features_list[0]['left_eye']
    except:
        try:
            eye = facial_features_list[0]['right_eye']
        except:
            return

    # establish the max x and y coordinates of the eye
    x_max = max([coordinate[0] for coordinate in eye])
    x_min = min([coordinate[0] for coordinate in eye])
    y_max = max([coordinate[1] for coordinate in eye])
    y_min = min([coordinate[1] for coordinate in eye])

    # establish the range of x and y coordinates
    x_range = x_max - x_min
    y_range = y_max - y_min

    # in order to make sure the full eye is captured,
    # calculate the coordinates of a square that has a
    # 50% cushion added to the axis with a larger range and
    # then match the smaller range to the cushioned larger range
    if x_range > y_range:
        right = round(.5*x_range) + x_max
```

Figure 17: Code snippet for real-time testing

```
    # then match the smaller range to the lengthened larger range
    if x_range > y_range:
        right = round(.5*x_range) + x_max
        left = x_min - round(.5*x_range)
        bottom = round((((right-left) - y_range))/2) + y_max
        top = y_min - round((((right-left) - y_range))/2)
    else:
        bottom = round(.5*y_range) + y_max
        top = y_min - round(.5*y_range)
        right = round((((bottom-top) - x_range))/2) + x_max
        left = x_min - round((((bottom-top) - x_range))/2)

    # crop the image according to the coordinates determined above
    cropped = frame[top:(bottom + 1), left:(right + 1)]

    # resize the image
    cropped = cv2.resize(cropped, (75,75))
    image_for_prediction = cropped.reshape(-1, 75, 75, 1)

    return image_for_prediction

# initiate webcam
cap = cv2.VideoCapture(0)
w = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
h = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
if not cap.isOpened():
    raise IOError('Cannot open webcam')

# set a counter
counter = 0
```

Figure 18: Code snippet for real-time testing

```
while True:

    # capture frames being outputted by webcam
    ret, frame = cap.read()

    # use only every other frame to manage speed and memory usage
    frame_count = 0
    if frame_count == 0:
        frame_count += 1
        pass
    else:
        count = 0
        continue

    # function called on the frame
    image_for_prediction = eye_cropper(frame)
    try:
        image_for_prediction = image_for_prediction/255.0
    except:
        continue

    # get prediction from model
    prediction = eye_model.predict(image_for_prediction)

    # Based on prediction, display either "Open Eyes" or "Closed Eyes"
    if all (prediction < 0.5):
        counter = 0
        status = 'Open'

        cv2.rectangle(frame, (round(w/2) - 110,20), (round(w/2) + 110, 80), (38,38,38), -1)

        cv2.putText(frame, status, (round(w/2)-80,70), cv2.FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 2, cv2.LINE_4)
        x1, y1,w1,h1 = 0,0,175,75
        ## Draw black backgroun rectangle
        cv2.rectangle(frame, (x1,x1), (x1+w1-20, y1+h1-20), (0,0,0), -1)
```

Figure 19: Code snippet for real-time testing

```
        cv2.rectangle(frame, (round(w/2) - 110,20), (round(w/2) + 110, 80), (38,38,38), -1)

        cv2.putText(frame, status, (round(w/2)-80,70), cv2.FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 2, cv2.LINE_4)
        x1, y1,w1,h1 = 0,0,175,75
        ## Draw black backgroun rectangle
        cv2.rectangle(frame, (x1,x1), (x1+w1-20, y1+h1-20), (0,0,0), -1)
        ## Add text
        cv2.putText(frame, 'Active', (x1 +int(w1/10), y1+int(h1/2)), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255,0),2)
    else:
        counter = counter + 1
        status = 'Closed'

        cv2.rectangle(frame, (round(w/2) - 110,20), (round(w/2) + 110, 80), (38,38,38), -1)

        cv2.putText(frame, status, (round(w/2)-104,70), cv2.FONT_HERSHEY_SIMPLEX, 2, (0,0,255), 2, cv2.LINE_4)
        x1, y1,w1,h1 = 0,0,175,75
        ## Draw black backgroun rectangle
        cv2.rectangle(frame, (x1,x1), (x1+w1-20, y1+h1-20), (0,0,0), -1)
        ## Add text
        cv2.putText(frame, 'Active', (x1 +int(w1/10), y1+int(h1/2)), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255,0),2)

        # if the counter is greater than 3, play and show alert that user is asleep
        if counter > 2:

            ## Draw black background rectangle
            cv2.rectangle(frame, (round(w/2) - 160, round(h) - 200), (round(w/2) + 160, round(h) - 120), (0,0,255), -1)
            cv2.putText(frame, 'DRIVER SLEEPING', (round(w/2)-136,round(h) - 146), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,0), 2, cv2.l
            cv2.imshow('Drowsiness Detection', frame)
            k = cv2.waitKey(1)
            continue

    cv2.imshow('Drowsiness Detection', frame)
    k = cv2.waitKey(1)
    if k == 27:
```

Figure 20: Code snippet for real-time testing

References

Wang, J. P. L., 2017. *The Effectiveness of Data Augmentation in Image Classification using Deep Learning.* s.l.:arXiv:1712.04621v1.