

Configuration Manual

MSc Research Project
Data Analytics

Nevin Saini
Student ID: x18132260

School of Computing
National College of Ireland

Supervisor: Dr. Cristina Muntean

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Nevin Saini
Student ID: x18132260
Programme: Data Analytics **Year:** 2018-19
Module: MSc. Research Project
Lecturer: Dr. Cristina Muntean
Submission Due Date: 12/12/2019
Project Title: Configuration Manual
Word Count: 1116 **Page Count:** 16

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: Nevin Saini
Date: 12/12/2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input checked="" type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input checked="" type="checkbox"/>
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.	<input checked="" type="checkbox"/>

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

Nevin Saini
x18132260

1 Introduction

The configuration manual contains all the pertinent information related to the software and hardware used in the research project. Also, it specifies the important libraries that are used and the data description is given in section 3. Moreover, it elucidates several steps that needs to be taken to reproduce the work in any machine satisfying the requirements which is covered in the following sections

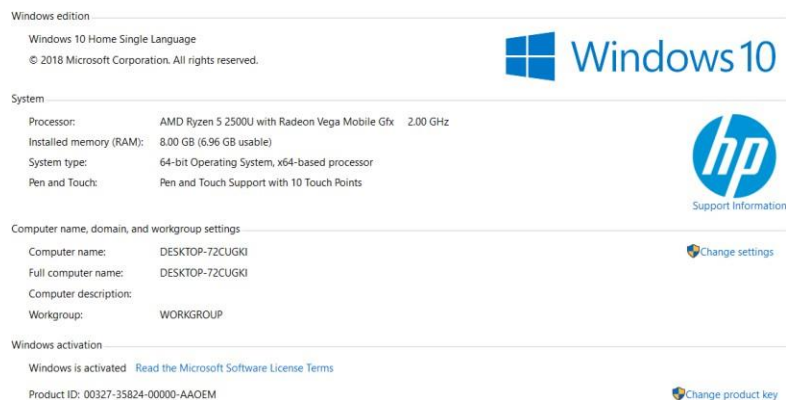
2 Environment Specifications

The MSc. Research project runs on a system which has certain specifications for both software and hardware that are described in the following subsections.

2.1 Hardware Specifications

This project is implemented on the hardware with the following configurations:

Hardware	Configurations
System	HP Envy
Operation System	Windows 10 (64-bit Operating system)
RAM	8 GB
Hard Disk	512 GB (SSD)
Graphic Card	2 GB Ryzen



The screenshot shows the Windows 10 system information page. It includes the following details:

- Windows edition:** Windows 10 Home Single Language, © 2018 Microsoft Corporation. All rights reserved.
- System:** Processor: AMD Ryzen 5 2500U with Radeon Vega Mobile Gfx 2.00 GHz; Installed memory (RAM): 8.00 GB (6.96 GB usable); System type: 64-bit Operating System, x64-based processor; Pen and Touch: Pen and Touch Support with 10 Touch Points.
- Computer name, domain, and workgroup settings:** Computer name: DESKTOP-72CUGKI; Full computer name: DESKTOP-72CUGKI; Computer description: ; Workgroup: WORKGROUP.
- Windows activation:** Windows is activated. Read the Microsoft Software License Terms. Product ID: 00327-35824-00000-AAOEM.

Logos for Windows 10 and HP are also visible on the right side of the page.

2.2 Software Specifications

In this project, plethora of softwares are used which are represented in Table 2.

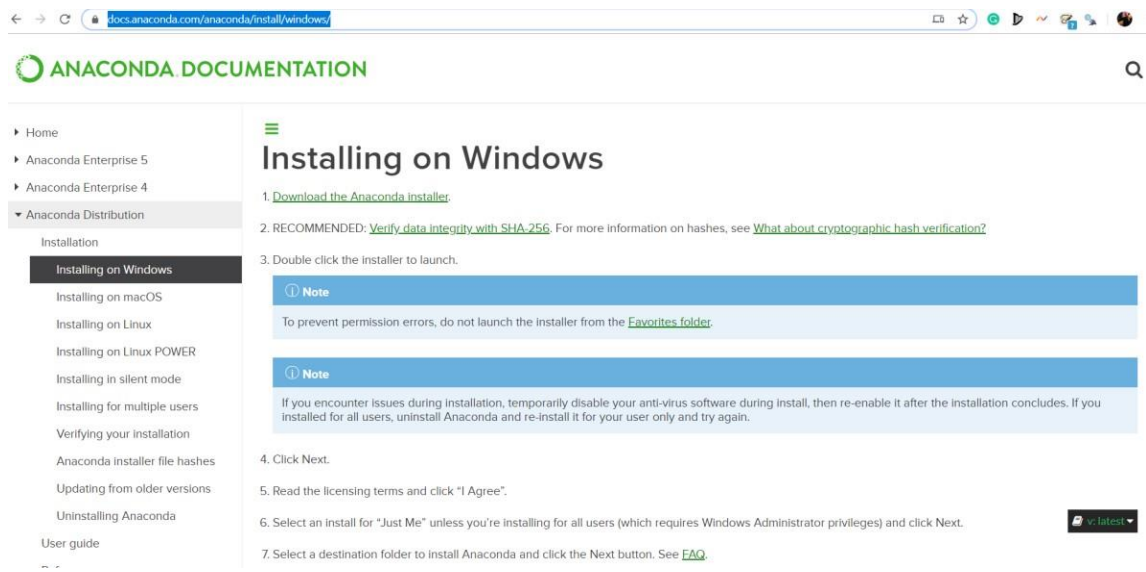
Software	Configurations
Operating System	Windows 10 (64-bit Operating system)
IDE	Spyder (Anaconda Navigator)
Scripting Language	Python
Scripting Language version	Python 3.7

2.2.1 Integrated Development Environment

A latest version of Anaconda navigator is installed to implement the highly demanding models. Its installation includes series of several steps which is described below:

Step 1:

- Visit the website and click on the link of first step.

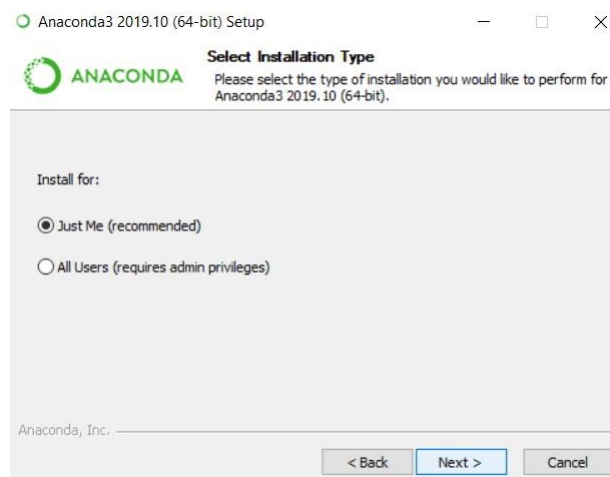
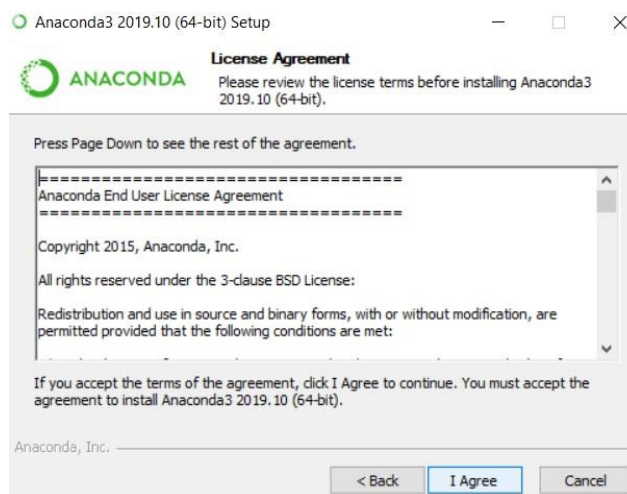


- Click on the file and install it in the system.

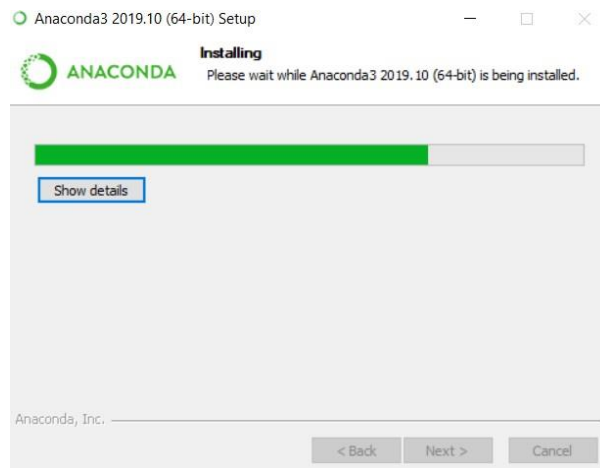
- Click Next to step further..



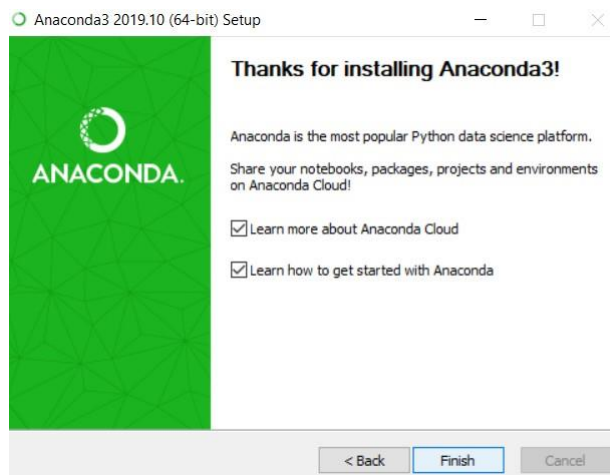
- Click on “I Agree” and then “Next” to step further in installation process



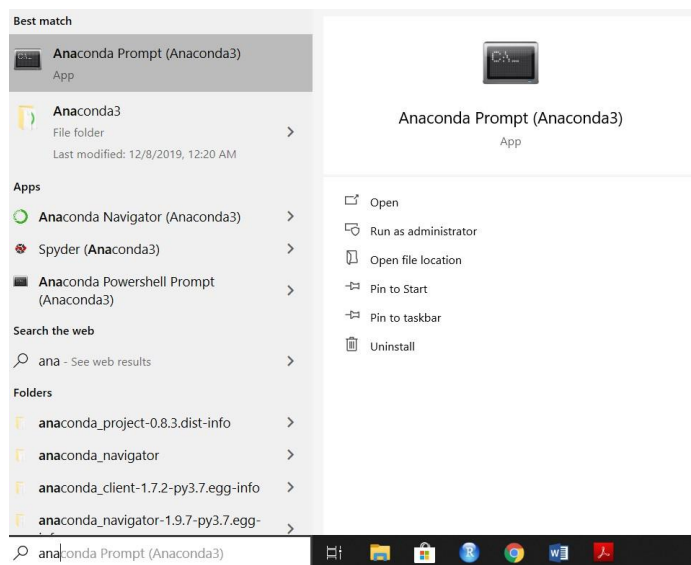
- It will start installation which will take some time considering the size of the software.



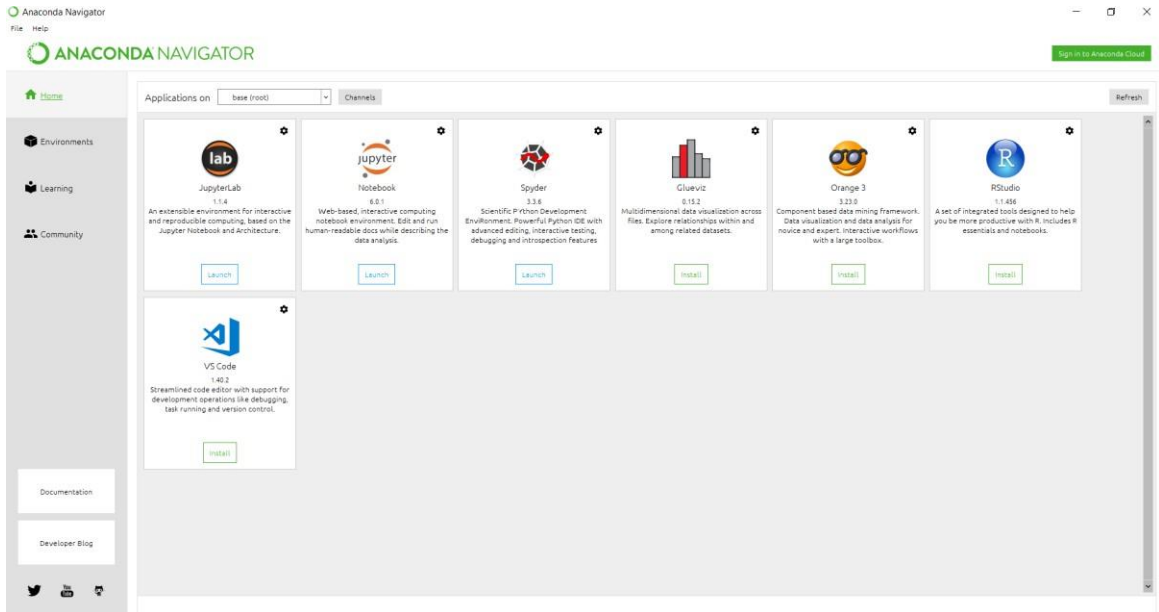
- Click on the “Next” button.



- Go to the search icon on the bottom left in windows 10 machine and type anaconda.

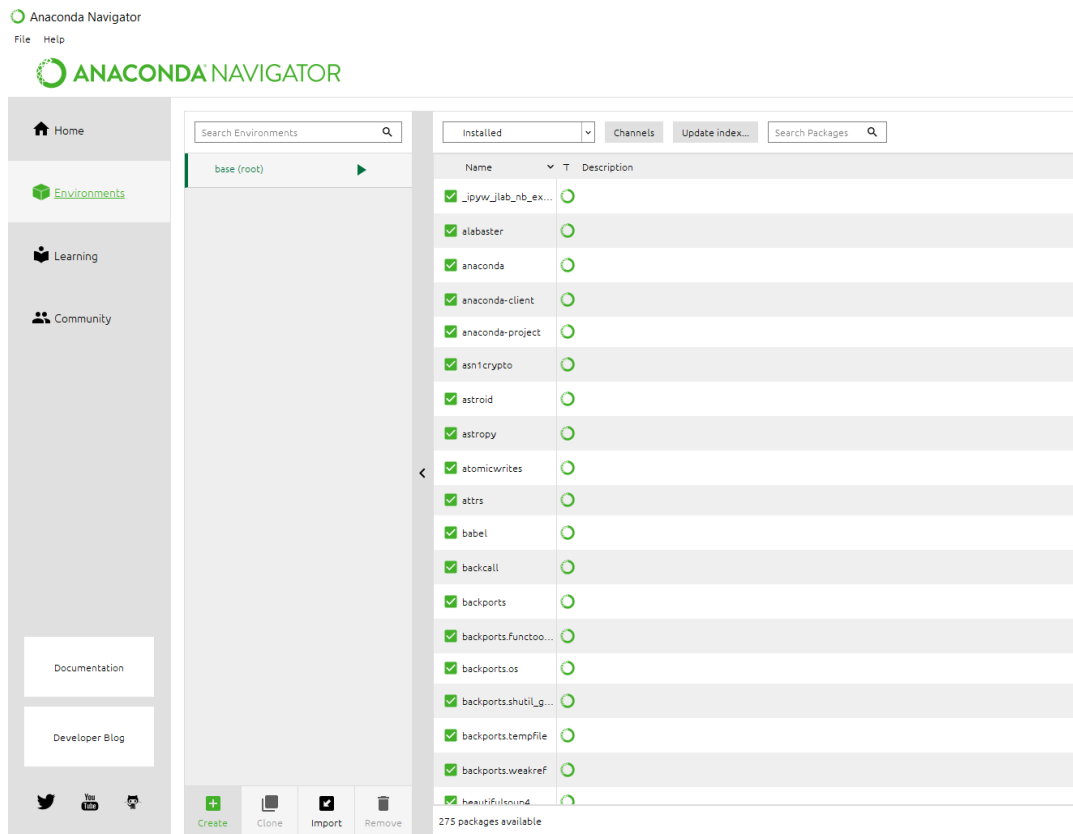


- Click on the specified Icon of Anaconda Navigator to launch the application.

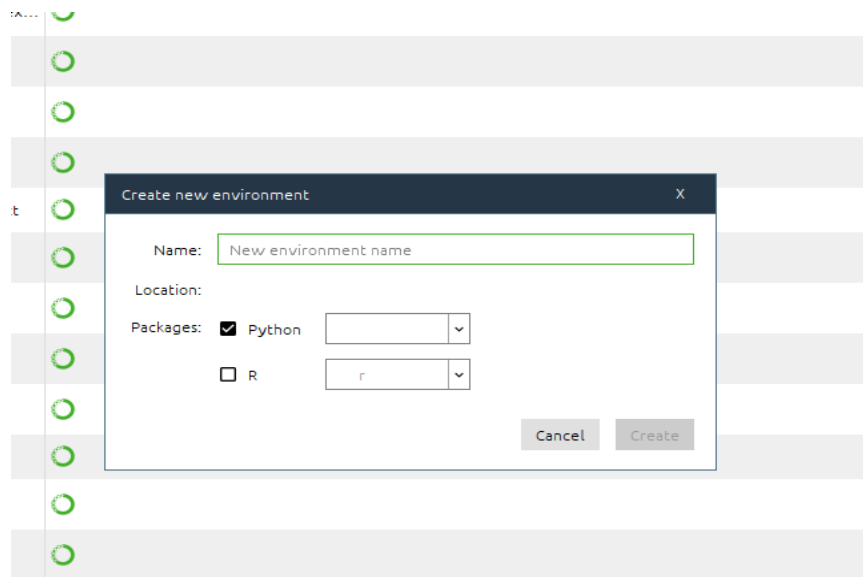


- It is important to create a new environment for the project where all the useful libraries and packages can be installed.

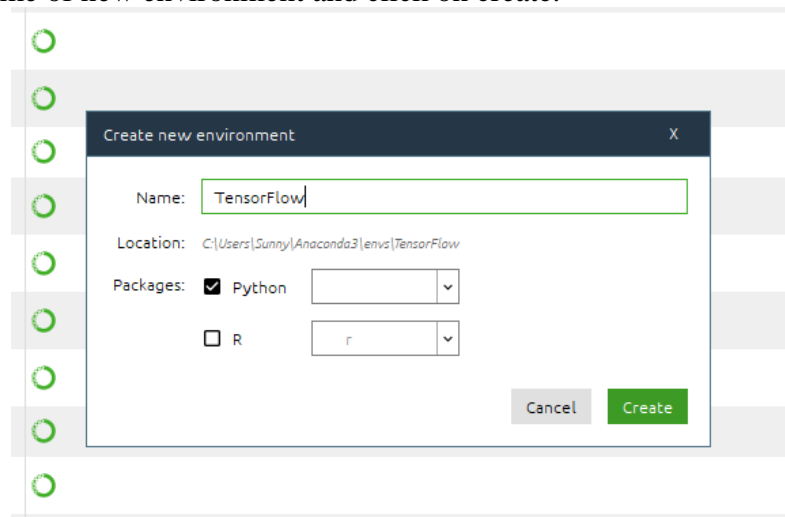
To do this, Click on Environments.



- Click on create button on the bottom screen to create a new environment.



- Add the name of new environment and click on create.



2.3 Libraries

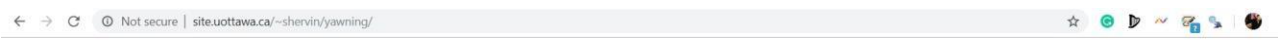
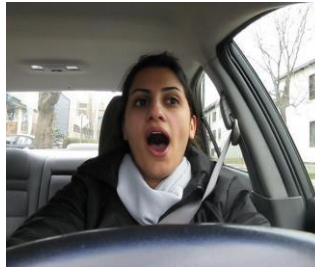
After the creation of an environment in Anaconda, we have installed all the vital libraries required to execute the research project which are discussed below.

matplotlib	tensorflow.keras.layers
matplotlib.pyplot	sklearn.metrics
numpy	sklearn.preprocessing
sklearn	sklearn.model_
pandas	sklearn.grid_search
sklearn.ensemble	sklearn.neighbors
sklearn	tensorflow
opencv	skimage.io
skimage	tensorflow.keras.preprocessing
tensorflow.keras.models	

3. Dataset

There are two datasets used for the research project which are as follows:

3.1 Yawning Dataset: It contains two datasets in video format of drivers with different facial features to detect drowsiness of drivers. 322 videos are taken in real time of different individuals. In addition, 29 more videos of both male and female drivers are taken from driver’s dash.



YawDD: Yawning Detection Dataset

Purpose

YawDD contains two video datasets of drivers with various facial characteristics, to be used for testing algorithms and models for mainly yawning detection, but also recognition and tracking of face and mouth. The videos are taken in real and varying illumination conditions.

- In the first dataset a camera is installed under the front mirror of the car. Each participant has three/four videos and each video contains different mouth conditions such as normal, talking/singing, and yawning. This dataset provides 322 videos consisting of both male and female drivers, with and without glasses/sunglasses, from different ethnicities, and in 3 different situations: 1- normal driving (no talking), 2- talking or singing while driving, and 3- yawning while driving.
- In the second dataset the camera is installed on the driver’s dash. Each participant has a single video containing scenes with driving, driving while talking, and driving while yawning. This dataset provides 29 videos consisting of both male and female drivers, with and without glasses/sunglasses, from different ethnicities.

Format and Available Data

The videos are in 640x480 24-bit true color (RGB) 30 frames per second AVI format without audio. The total data size is about **5 Gigabytes**. The available data and their features are listed in [table 1](#) for dataset 1 and [table 2](#) for dataset 2, and can be downloaded from ACM MMsys Dataset webpage (see link below).

License and Usage

The videos are for non-commercial and research purposes only! For all other usage, please contact shervin@ieee.org. The videos are free for use in non-commercial and/or academic papers/reports, which study, design, and test algorithms and methods to detect face, facial features, yawning, etc. In addition, screenshots of some (not all) videos can be used in such papers. Please check the *Allow Researchers to use picture in their paper* column in the above two tables to see if you can use a screenshot of a particular video or not. **If for a particular video that column is “no”, you are NOT allowed to use pictures from that video in your papers and publications.**

To refer to this dataset in your paper, please use the following citation:

S. Abtahi, M. Omidyeganeh, S. Shirmohammadi, and B. Hariri, “YawDD: A Yawning Detection Dataset”, *Proc. ACM Multimedia Systems*, Singapore, March 19-21 2014, pp. 24-28.
DOI: [10.1145/2557642.2563678](https://doi.org/10.1145/2557642.2563678)

You might also be interested to read the following paper, in which we used the dataset to test our yawning detection algorithm designed for embedded smart cameras:

M. Omidyeganeh, S. Shirmohammadi, S. Abtahi, A. Khurshid, M. Farhan, J. Scharcanski, B. Hariri, D. Laroche, and L. Martel, “Yawning Detection Using Embedded Smart Cameras”, *IEEE Trans. on Instrumentation and Measurement*, Vol. 65, Issue 3, March 2016, pp. 570-582.
DOI: [10.1109/TIM.2015.2507378](https://doi.org/10.1109/TIM.2015.2507378)

These images are converted into frames for both yawning and open eyes. Almost 1479 images are used for yawning dataset and 1222 frames contain images of individuals with open eyes.

3.2 Closed eyes: This contains around 1189 images of different people from diverse angles with closed eyes and different environmental changes like blur and lighting.



4. Image Pre-processing

Image Pre-processing has been done on the dataset to detect drowsiness of an ATC and considering different situation that a controller can phase in terms of lighting. It involves several independent steps like Rescaling, Rotation, Gray scaling, edge detection, flipping and is done before segmentation. According to Bazeille et al. (2010), Models shows great accuracy while filtering the images using edge detection.

4.1 CNN

```
# data augmentation to prevent overfitting
train_datagen = ImageDataGenerator(
    featurewise_center=False, # set input mean to 0 over the dataset
    samplewise_center=False, # set each sample mean to 0
    featurewise_std_normalization=False, # divide inputs by std of the dataset
    samplewise_std_normalization=False, # divide each input by its std
    zca_whitening=False, # apply ZCA whitening
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.1, # Randomly zoom image
    width_shift_range=0.2, # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.2, # randomly shift images vertically (fraction of total height)
    horizontal_flip=True, # randomly flip images
    vertical_flip=False) # randomly flip images

test_datagen = ImageDataGenerator(rescale=1./255)
```

4.2 Machine learning models

```
plt.imshow(image)

RGB_img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

scale_percent = 60 # percent of original size
width = int(RGB_img.shape[1] * scale_percent / 100)
height = int(RGB_img.shape[0] * scale_percent / 100)
dim = (width, height)

# resize image
image_resized = cv2.resize(RGB_img, (70,70), interpolation = cv2.INTER_AREA)

image_gray = cv2.cvtColor(image_resized, cv2.COLOR_BGR2GRAY)

image_hsv = cv2.cvtColor(image_resized, cv2.COLOR_BGR2HSV)

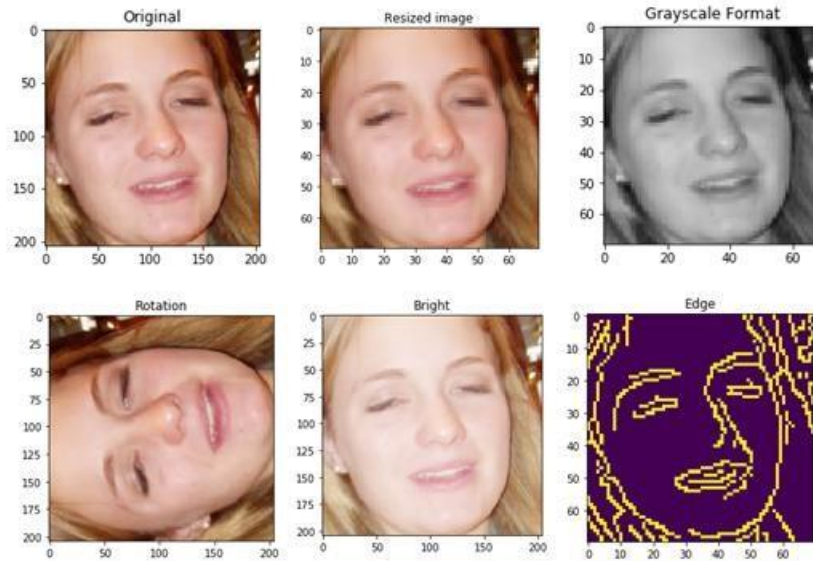
image_rotated = rotate(RGB_img, angle=90)

image_flip = fliplR(RGB_img)

image_bright = exposure.adjust_gamma(RGB_img, gamma=0.5,gain=1)

image_dark = exposure.adjust_gamma(RGB_img, gamma=1.5,gain=1)
```

Both image-pre-processing techniques were performed on the datasets to increase the number of images and to improve the performance of the system due to the presence of different versions of same image.



5. Model Creation

Different models are designed to implement the project in both deep and machine learning like CNN, SVM, KNN, XGBoost and Random Forest.

Machine learning techniques are designed using Scikit python library and CNN model is created by utilizing keras and Tensorflow . Through the results, It is observed that all the algorithms have performed well, however, Deep learning robust model CNN has outperformed other techniques based on some performance measures like Accuracy, Precision, Recall and F1 Score.

The execution of these techniques is done on two models i.e. Baseline Models and Tuned Models (after hyperparameter tuning) that are shown below.

5.1 Baseline Models

5.1.1 Convolutional Neural Network

```

# =====
# Preparing layers of an improved CNN model
# =====
improvedmodel = Sequential()
improvedmodel.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
improvedmodel.add(MaxPool2D(pool_size=(2, 2)))
improvedmodel.add(Dropout(rate=0.25))
improvedmodel.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
improvedmodel.add(MaxPool2D(pool_size=(2, 2)))
improvedmodel.add(Dropout(rate=0.25))
improvedmodel.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
improvedmodel.add(MaxPool2D(pool_size=(2, 2)))
improvedmodel.add(Dropout(rate=0.25))
improvedmodel.add(Flatten())
improvedmodel.add(Dense(256, activation='relu'))
improvedmodel.add(Dropout(rate=0.5))
improvedmodel.add(Dense(3, activation='softmax'))

```

5.1.2 SVM

```
# =====  
# Dividing the datasets into 75%:25% Training and Testing set ratio  
# =====  
X_train, X_val, y_train, y_val = train_test_split(data_scaled, labels, test_size=0.25, random_state=42)  
  
# =====  
# Designing a baseline SVM Model  
# =====  
model = svm.SVC()  
  
# =====  
# Training the Model on the training image dataset  
# =====  
model.fit(X_train, y_train)  
  
# =====  
# Predicting the Labels on testing image dataset  
# =====  
y_pred = model.predict(X_val)  
  
# =====  
# Calculating the Accuracy of the baseline model  
# =====  
accuracy = metrics.accuracy_score(y_pred, y_val) * 100  
print("Accuracy with Baseline SVM Model: {0:.2f}%".format(accuracy))  
  
# =====  
# Generating classification table for all the individual classes  
# =====  
print(classification_report(y_val, y_pred))
```

5.1.3 KNN

```
# =====  
# Dividing the datasets into 75%:25% Training and Testing set ratio  
# =====  
X_train, X_val, y_train, y_val = train_test_split(data, labels, test_size=0.25, random_state=42)  
  
# =====  
# Designing a baseline KNN Model  
# =====  
model = KNeighborsClassifier()  
  
# =====  
# Training the Model on the training image dataset  
# =====  
model.fit(X_train, y_train)  
  
# =====  
# Predicting the Labels on testing image dataset  
# =====  
y_pred = model.predict(X_val)  
  
# =====  
# Calculating the Accuracy of the baseline model  
# =====  
accuracy = metrics.accuracy_score(y_pred, y_val) * 100  
print("Accuracy with K-NN: {0:.2f}%".format(accuracy))
```

5.1.4 XGBoost

```

# =====
# Dividing the datasets into 75%:25% Training and Testing set ratio
# =====
X_train, X_val, y_train, y_val = train_test_split(data_scaled, labels, test_size=0.25, random_state=42)

# =====
# Designing a baseline XGBoost Model
# =====
model = xgb.XGBClassifier()

# =====
# Training the Model on the training image dataset
# =====
model.fit(X_train, y_train)

# =====
# Predicting the Labels on testing image dataset
# =====
y_pred = model.predict(X_val)

# =====
# Calculating the Accuracy of the baseline model
# =====
accuracy = metrics.accuracy_score(y_pred, y_val) * 100
print("Accuracy with Baseline XGBoost Model: {0:.2f}%".format(accuracy))

```

5.1.5 Random Forest

```

# =====
# Dividing the datasets into 75%:25% Training and Testing set ratio
# =====
X_train, X_val, y_train, y_val = train_test_split(data_scaled, labels, test_size=0.25, random_state=42)

# =====
# Designing a baseline Random Forest Model
# =====
model = RandomForestClassifier()

# =====
# Training the Model on the training image dataset
# =====
model.fit(X_train, y_train)

# =====
# Predicting the Labels on testing image dataset
# =====
y_pred = model.predict(X_val)

# =====
# Calculating the Accuracy of the baseline model
# =====
accuracy = metrics.accuracy_score(y_pred, y_val) * 100
print("Accuracy with Baseline Random Forest Model: {0:.2f}%".format(accuracy))

```

5.2 Hyper parameter Tuning

Several machine learning models requires setting many hyperparameters before testing them on the validation set (Boulesteix and Bischl, 2018) and It is extremely important to search the right parameters to be used in a particular machine learning algorithm which improves the performance of a baseline model by increasing the ability to find the correct results out of the total predictions done by the model.

5.2.1 SVM

```
# =====  
# Hyperparameter tuning for SVM  
# =====  
  
# =====  
# Creating all the important parameters and the possible values to be compared for the tuning  
# =====  
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 10]}  
  
# =====  
# Using GridSearch, searchinh through several combinations of parameter to find the best one  
# =====  
tunedModel = GridSearchCV(svm.SVC(), param_grid, verbose=1)  
tunedModel.fit(X_train, y_train)  
tunedModel.best_params_  
  
# =====  
# Predicting the labels on testing image dataset  
# =====  
y_predTuned = tunedModel.predict(X_val)  
  
# =====  
# Calculating the Accuracy of the tuned model  
# =====  
tuned_accuracy = metrics.accuracy_score(y_pred, y_val) * 100  
print("Accuracy with SVM: {0:.2f}%".format(tuned_accuracy))  
  
# =====  
# Generating classification report of the tuned Model  
# =====  
print(classification_report(y_val, y_pred))
```

5.2.2 KNN

```
# =====  
#  
# Hyper Parameter tuning for KNN  
# # =====  
  
# =====  
# Creating all the important parameters and the possible values to be compared for the tuning  
# =====  
params_knn = {'n_neighbors':[5,7,8,9], 'weights':['uniform', 'distance']}  
  
# =====  
# Using GridSearch, searchinh through several combinations of parameter to find the best one  
# =====  
tunedModel = GridSearchCV(model, param_grid=params_knn)  
tunedModel.fit(X_train,y_train)  
tunedModel.best_params_  
  
# =====  
# Predicting the labels on testing image dataset  
# =====  
y_predTuned = tunedModel.predict(X_val)  
  
# =====  
# Calculating the Accuracy of the tuned model  
# =====  
accuracy_tuned = metrics.accuracy_score(y_predTuned, y_val) * 100  
print("Accuracy with K-NN: {0:.2f}%".format(accuracy_tuned))  
  
# =====  
# Generating classification report of the tuned Model  
# =====  
print(classification_report(y_val, y_pred))
```

5.2.3 XGBoost

```
# =====  
# Hyperparameter Tuning for XGBoost  
# =====  
  
# =====  
# Creating all the important parameters and the possible values to be compared for the tuning  
# =====  
param_grid = {'learning_rate': [0.01, 0.1], 'gamma': [0.5, 0, 1], 'verbosity' : [0]}  
  
# =====  
# Using GridSearch, searchinh through several combinations of parameter to find the best one  
# =====  
tunedModel = GridSearchCV(model, param_grid)  
  
# =====  
# Using the best fitted parameters chosen, designing the tuned model  
# =====  
tunedModel.fit(X_train, y_train)  
tunedModel.best_params_  
print("Best Parameters:\t", tunedModel.best_params_)  
  
# =====  
# Predicting the Labels on testing image dataset  
# =====  
y_predTuned = tunedModel.predict(X_val)  
  
# =====  
# Calculating the Accuracy of the tuned model  
# =====  
accuracy_tuned = metrics.accuracy_score(y_predTuned, y_val) * 100  
print("Accuracy with Tuned XGBoost Model: {:.2f}%".format(accuracy_tuned))  
  
# =====  
# Generating classification report of the tuned Model  
# =====  
print(classification_report(y_val, y_pred))
```

5.2.4 Random Forest

```
# =====  
# Hyperparameter Tuning for XGBoost  
# =====  
  
# =====  
# Creating all the important parameters and the possible values to be compared for the tuning  
# =====  
params_RF = {"max_depth": [3,5,6,7,8],  
            "min_samples_split": [2, 3, 10],  
            "min_samples_leaf": [1, 3, 10],  
            "criterion": ["gini", "entropy"]}  
  
# =====  
# Using GridSearch, searchinh through several combinations of parameter to find the best one  
# =====  
tunedModel = GridSearchCV(model, param_grid=params_RF)  
tunedModel.fit(X_train,y_train)  
tunedModel.best_params_  
  
# =====  
# Predicting the Labels on testing image dataset  
# =====  
y_predTuned = tunedModel.predict(X_val)  
  
# =====  
# Calculating the Accuracy of the tuned model  
# =====  
accuracy_tuned = metrics.accuracy_score(y_predTuned, y_val) * 100  
print("Accuracy with K-NN: {:.2f}%".format(accuracy_tuned))  
  
# =====  
# Generating classification report of the tuned Model  
# =====  
print(classification_report(y_val, y_pred))
```

References

Bazeille, S. et al. (2010) ‘Automatic underwater image pre-processing’, France, 16-19 October.

Boulesteix, A. and Bischl, B. (2018) ‘Tunability: Importance of Hyperparameters of Machine Learning Algorithms’, *Journal of Machine Learning Research* , pp. 1–22.