

# Real time Emotion Detection

Panagiotis Konstantinou – 201859619

Supervisor: Dr Martin Goodfellow

This dissertation was submitted in part fulfilment of requirements for the degree of MSc Software Development

DEPT. OF COMPUTER AND INFORMATION SCIENCES UNIVERSITY OF STRATHCLYDE

AUGUST 2019

## Declaration

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself.

Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive.

(please tick) Yes ☐ No ☐

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is 18657 words.

I confirm that I wish this to be assessed as a Type 1 2 3 4 5 Dissertation (please circle)

Signature:

Date:

## Abstract

Computer vision is the subfield of artificial intelligence that deals with how computers can gain a high-level understanding from images. Examples of this include face detection and recognition systems, line tracking systems that are currently implemented on self-driving cars and authentication of documents such as passports or drivers' licenses. The area of interest of this project revolves around the field of emotion recognition.

This dissertation thesis outlines the development of a Web-Application that can run natively on a Raspberry pi with the main functionality of detecting emotions in an input video stream. Different solutions to this problem are compared and the most lightweight one is implemented, demonstrating that the modern minicomputers are capable of carrying out tasks as complex as this one. The emotion detection module of the system is comprised of a face detection and extraction algorithm in series with a face recognition algorithm trained to recognise emotions. During the development two face detection algorithms and three face recognition algorithms are compared, and the best performing algorithms are utilised.

Results show that the system that has been developed can recognise emotions in an input video stream in real time, but when ran natively on the Raspberry Pi the limitations of the processing power impact the performance of the system. Emotion recognition works for pre-recorded video clips, but in real time the frame rate at which emotion detection happens is reduced.

## Acknowledgements

I would like to thank my supervisor, Dr Martin Goodfellow for the continuous support throughout the duration of this project. The development of the solution would not have been possible without his guidance.

I would like to thank Takeo Kanade, Jeffrey F. Cohn and Yingli Tian, Lyons, Shigeru Akemastu, Miyuki Kamachi and Jiro Gyoba, Natalie C. Ebner, Michaela Riediger, and Ulman Lindenberger and Mr Paul van Gent for making the datasets used in this project publicly available, allowing for high quality training data to be used for training the emotion detection model.

I would also like to thank the authors at <https://www.pyimagesearch.com/> for providing helpful and resourceful tutorials on OpenCV and computer vision based applications on the Raspberry Pi. Mr Paul van Gent also had some insightful tutorials on his tech blog at <http://www.paulvangent.com/>.

Finally, I would like to thank my family, and fiancé for their encouragement and continuous support throughout the duration of my studies.

# List of Contents

## Contents

Real time Emotion Detection .....	i
Declaration .....	i
Abstract .....	ii
Acknowledgements .....	iii
List of Contents .....	iv
List of figures .....	vii
List of tables.....	viii
Chapter 1 .....	1
Introduction:.....	1
Objective:.....	1
Outcome: .....	1
Report Structure: .....	2
Chapter 2 .....	3
Real World Applications:.....	3
Literature Review.....	4
Related Work .....	4
Emotion Detection:.....	4
Emotion Detection in Real Time: .....	4
Emotion Detection in images:.....	5
Emotion Detection on a Raspberry Pi: .....	6
Face Detection: .....	6
Face Detection in Real Time:.....	6
Face Detection in images: .....	7
Face Detection on a Raspberry Pi: .....	10
Conclusions .....	11
Chapter 3 .....	13
Problem Description and Specification .....	13
Problem Specification: .....	13
Requirements: .....	13
User stories: .....	13
Functional Requirements: .....	14
Requirements Prioritisation: .....	15

Development Methodology:.....	16
Chapter 4 .....	17
System Design:.....	17
High Level Design:.....	17
Low Level Design:.....	17
Programming Languages:.....	18
Libraries & Frameworks: .....	18
Hardware: .....	19
Development Tools: .....	19
System Architecture:.....	19
Chapter 5 .....	21
Detailed Design and Implementation: .....	21
Emotion Detection:.....	21
Methodology: .....	21
Face Detection: .....	21
Emotion Detection: .....	23
Web Application: .....	35
Methodology: .....	35
Back end: .....	35
Front end: .....	36
Integration of back end with the front end:.....	39
Raspberry Pi Development: .....	40
Chapter 6 .....	41
Verification and Validation: .....	41
Test Plan Overview: .....	41
Unit Testing:.....	41
Integration Testing:.....	42
Emotion Detection Tests:.....	42
Details of tests: .....	44
Web application Tests:.....	45
Details of tests: .....	46
Raspberry Pi Tests:.....	48
Chapter 7 .....	50
Results and Evaluation:.....	50
Final Application: .....	50
User Interface/Front end: .....	50

Functionality/Back end: .....	55
Emotion Detection: .....	55
Face Detection: .....	55
Emotion Detection: .....	55
Emotion Detection Performance: .....	56
Comparison with Specification: .....	57
Chapter 8 .....	59
Summary and Conclusions: .....	59
Summary/Reflection: .....	59
Future Work: .....	60
Web Application: .....	60
Emotion Detection: .....	60
Conclusion: .....	61
Appendix A .....	62
References: .....	62
Appendix B .....	65
Sort images script: .....	65
Appendix C .....	66
Model training scripts: .....	66
Model training script – multiple: .....	66
Model training script – individual: .....	68

## List of figures

Figure 1 - Interaction of the systems Sub-Modules .....	20
Figure 2 - Available Haar Cascade Classifiers .....	22
Figure 3 - Available deep learning models .....	23
Figure 4 - Machine Learning training and Predicting .....	24
Figure 5 - Original face .....	27
Figure 6 - Extracted face .....	28
Figure 7 - Original face .....	29
Figure 8 - Wrongfully extracted face.....	29
Figure 9 - Training emotion detection model .....	31
Figure 10 - System output after training .....	31
Figure 11 - System output after training .....	33
Figure 12 - Surprised face .....	34
Figure 13 - Happy face .....	34
Figure 14 - Emotion detection methodology .....	35
Figure 15 - Home page.....	37
Figure 16 - Emotion detection page.....	37
Figure 17 - Library page .....	38
Figure 18 - Video playback page .....	38
Figure 19 - Statistics page .....	39
Figure 20 - MVC architecture .....	40
Figure 21 - Wrongfully identified emotion due to variance in pose.....	56
Figure 22 - Wrongfully identified emotion due to variance in illumination .....	56
Figure 23 - Wrongfully identified emotion due to monitor reflection on glasses .....	57



## List of tables

Table 1 - Emotion detection acceptance tests .....	44
Table 2 - Web-Application acceptance tests .....	46
Table 3 - Evaluation of requirements.....	58

# Chapter 1

## Introduction:

Over the past few decades the fields of computer vision and machine learning have been advancing exponentially. With the advance in these fields, computers can now process information and gain a better understanding of their surroundings than ever before. An example of this is a computerised system that can recognise the emotions expressed by a human face in a video feed. There are multiple solutions to this task, but the aim of this project is to develop one such system that is lightweight enough to be deployable to embedded devices such as the Raspberry Pi. The Raspberry Pi is a low cost, minicomputer that can fit into a pocket. This device was selected as the embedded system utilised by this project because of how easy it is to configure and the processing power offered by it compared to its price.

## Objective:

The primary objective of this project was to develop an emotion recognition web-application that was capable of recognising emotions expressed by people in a video stream but could also be deployed to run natively on embedded devices. Current solutions revolve around deep learning techniques which require a great deal of processing. This means that when embedded devices are used, they function primarily as an input video source for a remote server that carries out the processing.

The web-application developed had to be capable of recording a video stream and detecting the emotions expressed in it. The detected emotions had to then be displayable in a meaningful manner in order to give insights into the emotions expressed by people being recorded. This could then be used by marketing professionals to gain truthful information about how people feel when watching an advert or having a product demonstrated to them.

## Outcome:

Upon completion of this project, a fully functioning web-application was developed that could detect emotions in real time, record the emotion detection procedure and store it to a library. The library could display its contents, and when a video from the library was selected it could be played within the application. Statistics detailing how long each emotion was detected for the duration that emotion detection was running could also be displayed. This was deployed to a Raspberry Pi, and although the performance was impacted due to the limitations of the Raspberry Pi's processing power, the functionality of the application was still reflected on the embedded device.

## Report Structure:

The following chapters of the report detail the procedures followed to implement the proposed solution. Chapter 2 examines existing solutions and related work, from which the methodology used to detect emotions is determined. Chapter 3 gives information regarding the requirements gathering of the system. Chapter 4 lists the technology necessary for the implementation before giving information about the architecture of the proposed solution. Chapter 5 then elaborates on the design and demonstrates how the solution was implemented, with chapter 6 explaining how the developed solution was tested. Chapter 7 re-visits the requirements of the system defined in Chapter 3 and verifies that the required functionality was successfully implemented. Finally, Chapter 8 gives a reflection on the development carried out, before providing information regarding the future of the application and finishes with a conclusion of the project.

## Chapter 2

### Real World Applications:

Emotion detection has multiple real-world applications that are currently being researched in the hopes that this technology can be deployed in the near future. Car manufacturers are working on emotion recognition systems for “smart” cars. By monitoring the emotional state of the driver, safety can be increased as the car can react better to the emotion of the driver. For example, an angry driver would be more likely to drive recklessly, so the car could adapt to this and minimize the control over the car that the driver has.

An article on linked in explained that Automated Teller Machines (ATM) could implement this technology, and when the person withdrawing money is expressing fear, the ATM machine could refuse to dispense money as this could likely be associated with a robbery.

This technology could be implemented in the healthcare sector in a few ways. Aiding autistic people understand emotions through a smart device that gives feedback according to the emotion detected in the person in front of them. Doctors can better understand how patients feel about their treatments and attempt to make treatment a better experience for patients.

Other applications include interview systems, where video interviews can be analysed, and the candidate can be evaluated based on their reactions to interview questions. This can give insights as to whether or not the applicant is a good fit for the company by comparing how current employees react to questions with how the applicant does.

Another example of an application that uses this technology is the system that is being built. An application that can aid in marketing research by replacing interviews and questionnaires. With such a system, truthful information can be extracted and used by companies when deciding on what changes to make to a product or which advert out of a selection they would like to air.

## Literature Review

### Related Work

The following section of the report documents the research into existing solutions to emotion detection. I started by examining work related to emotion detection in real time. I wanted to make sure that emotion detection could be implemented on a Raspberry Pi too, so I made sure to examine existing literature related to this too. Once the literature has been examined, I should have had an idea of the methodology that would be followed to develop my solution.

#### Emotion Detection:

##### Emotion Detection in Real Time:

In the first section of the literature review, the most relatable literature was examined. As the intention was to build a system that could detect emotions in a video stream in real time, this is the type of literature that was reviewed.

Bartlett et al., 2003 built a real time face detection and emotion recognition system. Their system detects a face and then the emotion corresponding to the detected face. The first stage of the system uses the Viola Jones (Haar Feature-based Cascade Classifiers) face detector. This is one of the first real time object detection algorithms, developed by Paul Viola and Michael Jones. Although it can be trained to detect any object, it is primarily used to detect faces. Once a face has been detected it is passed to a Support Vector Machine (SVM) Classifier, an algorithm that creates a line that separates data into classes. The face that is detected is then matched against known emotions (Classes) and the one with the highest match is selected as the emotional state of the face. They then implement a combination of Adaboost, a boosting algorithm that converts a set of weak classifiers into a strong one with SVM for emotion recognition. The addition of Adaboost to the system allows for the system to function in real time. (Bartlett et al., 2003)

Ghandi, Nagarajan and Desa developed a system that detects emotions in faces using a modification of the Particle Swarm Optimisation (PSO) algorithm, an algorithm that optimises a problem by implementing different solutions in an iterative manner. They call their modified algorithm the Guided PSO algorithm. Their algorithm tracks the movements of different points of the face to distinguish emotions and works well on pre-recorded video streams because it relies on pre-processing a video for the algorithm to function. They then apply the Lucas-Kanade optical flow algorithm, an algorithm that basically computes a vector pointing from each pixel in a video frame to the corresponding pixel

in the next frame of the video. This allows them to track the movements of the face in real time and therefore allows them to detect emotions in real time. (Ghandi, Nagarajan and Desa, 2010)

### Emotion Detection in images:

In the literature examined above, at a first stage Bartlett et al., used the Viola Jones face detection algorithm to detect a face prior to attempting to detect an emotion in that face. The Viola Jones algorithm is an algorithm that detects a face in images and not videos, but as a video is essentially a series of images it made sense that literature related to emotion detection in images should be explored.

Singla and Bansal used the FisherFace algorithm ([see Chapter 5](#)) to detect different emotions in images. The FisherFace algorithm is a face recognition algorithm based on Principal Component Analysis (PCA) that is a technique used to extract features. They use an emotion database which contains images of faces with their corresponding Action Units and Labels. This allows them to train their algorithm and determine the classifier that can be used to identify an emotion in an input image. They show that their implementation has a correct detection rate of around 86%, which indicates that this approach can be used to successfully identify emotions in images. They didn't need to detect faces in images, because they only used images of faces. (Singla and Bansal, 2011)

Beltrán Prieto and Komínková-Oplatková compared the performance of two emotion recognition implementations. Both implementations rely on the Viola Jones face-detection algorithm to detect a face, and once a face has been detected it is passed to the two emotion recognition systems. The first system was built by adapting the FisherFace technique to recognise emotions instead of faces through OpenCV, an open source computer vision library. Whilst the second was a C# application that makes requests to a Cognitive Services API, an external library that includes artificial intelligence features. Their research showed that the OpenCV based implementation had the best results regarding correct detection rate. (Beltrán Prieto and Komínková-Oplatková, 2017)

Thuseethan and Kuhanesan used the EigenFace algorithm ([see Chapter 5](#)), another face recognition algorithm also based on PCA to recognise emotions instead of faces in images. They used the Japanese Female Facial Expressions (JAFPE) database, which is a database of images of Japanese females expressing emotions to train and evaluate their implementation. They achieve an emotion recognition rate above 75%. (Thuseethan and Kuhanesan, 2014) However, their tests are carried out on images of Chinese or Japanese faces in controlled environments (perfect lighting and face looking directly at the camera), so it is unknown how this would perform in uncontrolled environments with different ethnicities.

Farahani, Sheikhan and Farrokhi implemented emotion recognition using the Gravitational Search Algorithm (GSA), which is a population search algorithm. Their emotion recognition algorithm recognises emotions from eyes and mouth features of coloured images. They use a Matlab image processing library to detect faces and extract the features that are passed to the GSA algorithm. Their tests show an accuracy of around 72% for their solution. (Farahani, Sheikhan and Farrokhi, 2014)

### Emotion Detection on a Raspberry Pi:

One of the goals of this project was for the solution to be implemented on an embedded device (Raspberry Pi), so in the next section of the literature review, solutions implemented on the Raspberry Pi were examined.

Lee, Chen and Wei developed an IoT based emotion recognition system by using a Raspberry Pi and Microsoft's Emotion API. They use the Raspberry Pi with a camera module as the input to the system, this is then passed through the Viola Jones faces-detection algorithm to detect a face. Once a face has been detected it is sent via WIFI to a PC running Microsoft's Emotion API to detect the emotion in the face. The corresponding emotion is then sent back to the Raspberry Pi for a controlling action to be taken such as ringing a buzzer or lighting an LED. (Lee, Chen and Wei, 2018)

Sajjad et al., developed a facial expression recognition system using a Raspberry Pi. Their approach is to use the Raspberry Pi camera module for capturing video streams, the captured video stream is then passed through the Viola Jones Algorithm in order to detect a face. Once a face is detected feature extraction is carried out using an SVM Classifier Algorithm. The features are then sent to a cloud service that carries out the facial expression recognition in real time. (Sajjad et al., 2019)

Suchitra, Suja P. and Tripathi developed an emotion recognition system using a Raspberry Pi, but unlike others in this literature, their solution carries out the processing on the actual Pi. They use the Viola Jones face detection algorithm for face detection. The detected face is then passed through an Active Shape Model (ASM), a statistical model that shapes objects from the input and tries to match the shape to a known shape. This is used for feature extraction, with the extracted features passed through an Adaboost Classifier for classification of emotions. They claim an emotion detection accuracy of 94% and a computational time of 120ms. (Suchitra, Suja P. and Tripathi, 2016)

### Face Detection:

#### Face Detection in Real Time:

All the literature examined above has a common theme prior to attempting to detect the emotional state of the person detected in a video or image. In order to detect the emotion of the person in the

image, first their face must be detected. If a face is detected, then an emotion corresponding to that face can be determined. So the next section of the literature review is oriented around face detection, starting with face detection in real time. Again, because the solution that was developed was intended to run in real time.

Yang and Waibel developed a real time face tracker and follower capable of tracking a face in a video stream at a rate of 30 frames per second. Their system is comprised of three stages, first a face is detected, then a model that estimates the motion of the face in the image and finally a camera controller used to control the motion of the camera. In their solution, face detection is carried out using a stochastic model to characterise skin colour distribution in human faces. (Yang and Waibel, n.d.)

Shakhnarovich, Viola and Moghaddam attempted to adapt the Viola Jones face-detection algorithm in order to detect the gender and ethnicity of the faces in images. They use the Viola Jones algorithm to detect faces, and then put the detected faces through a demographic classifier, which is their adapted Viola Jones algorithm based on the same architecture as the face detector. They show that the process is successful and can achieve face and demographic detection at a rate of 10 frames per second. (Shakhnarovich, Viola and Moghaddam, n.d.)

Wati and Abadianto developed a face detection and recognition system implemented on a MyRIO 1900 that is a combination of Field Programmable Gate Array (FPGA) and a Microcontroller. An FPGA is essentially a matrix of configurable logic gates that can be programmed to function as required by the system it is used in. They detect faces using Template Matching, which is a technique that matches sections of an input image with a known template. They then recognise the person in the face using the PCA algorithm. Their solution can detect and recognise faces in real time at distances up to 240cm away from the camera. (Wati and Abadianto, 2017)

### Face Detection in images:

For the same reason that emotion detection in images was examined, face detection in images had to also be examined. The idea being that if a face can be detected in an image, then it can also be detected in a video prior to attempting to detect the emotion of the person in the video.

### *Face Detection using Viola Jones:*

The majority of solutions to emotion detection implemented on the Raspberry Pi use the Viola Jones face-detection algorithm, to detect faces prior to detecting emotions, so I had to examine this in more detail.



Viola and Jones essentially developed a face detection framework that achieves a high detection rate in a rapid timeframe. Their framework works using a representation of an image called the “Integral Image”, which is based on the sum of the pixel intensity values. They generate the integral image for different sections of the input image and return possible features very quickly. Once a feature is detected, it gets passed through a classifier built on the AdaBoost learning algorithm that selects critical features from a set of potential features. They then combine classifiers in cascade allowing for background regions of the image to be discarded and only the face-like region to remain. (Viola and Jones, 2004)

Next I decided to examine face detection implementations using the Viola Jones algorithm, and it turned out that although the Viola Jones algorithm is widely used, there are some shortcomings when building a system that utilises it.

Laytner, Ling and Xiao explain that dark skin coloured faces or badly illuminated faces are the biggest cause of faces not being detected when using the Viola Jones face-detection algorithm. They developed their own face detection algorithm consisting of Histogram analysis, Haar Wavelet transformation and Adaboost learning techniques. They then compared their algorithm to the “Classic” (Viola Jones) algorithm and showed that the lighting and illumination issue can be partially overcome with some image pre-processing prior to face detection. (Laytner, Ling and Xiao, 2014)

Magalhaes, Ren and Cavalcanti carried out research regarding variations in illumination of facial images. Instead of pre-processing images and then passing them to a face detection model, they applied the GradientFaces technique before training their face detection model. The GradientFaces algorithm is a facial recognition algorithm that transforms the image into the gradient domain, making the image illumination insensitive. They re-trained the Viola Jones face-detection algorithm using pre-processed (GradientFaces) images and then compared it to the “Classic” (Viola Jones) algorithm. Their results show that this approach is more robust than other methods of dealing with the issue of varied illuminations. (Magalhaes, Ren and Cavalcanti, 2012)

#### *Face Detection not using Viola Jones:*

The authors above, show that the biggest shortcoming of the Viola Jones algorithm is when attempting to detect faces in environments without ideal lighting conditions which led to the question of what other methods of detecting a face are possible?

Li et al. recognise that when dealing with face detection algorithms, one of the main issues that must be overcome is the variation in pose of faces in images or partial occlusion (Someone wearing glasses or a hat). They explain that “Classic” approaches to facial recognition fail to overcome this and attempt

to deal with this by using deep learning, a form of machine learning inspired by the structure and function of the human brain. With deep learning, different layers can be used to extract different features. They conclude that illumination variance can be overcome using deep learning but occlusion is a more difficult task. (Li et al., 2015)

Laboreiro, Maia and de Araujo use a decision tree-based approach to detect faces in images. A decision tree is basically a decision support tool modelled on a tree like graph, where all possible decisions and outcomes are mapped to different branches. They explain that for real time applications face detection is typically carried out by partitioning the image into two classes: face and non-face but they attempt a multistage segmentation instead. They use a decision tree by segmenting a facial image into 7 different classes: eyes, nose, mouth, eyebrows, hair, facial skin and background. Although this approach is more computationally intense, it doesn't require any previous knowledge, so it is a method of unsupervised learning that can be used to detect faces in images. Unsupervised learning is a section of machine learning that refers to algorithms that learn from input data without prior training. (Laboreiro, Maia and de Araujo, 2012)

#### *Comparison:*

Before I made any conclusions about what face detection technique should be used in my system, I examined literature that compared different face detection algorithms.

Dang and Sharma carried out experiments to compare different face detection algorithms regarding their accuracy. They compared the Viola Jones face-detection algorithm, SMQT Features and Snow Classifier, Neural Network Based Face Detection and Support Vector Machines-Based face detection. They concluded that the Viola James algorithm was the most accurate one from the algorithms compared in their study. (Dang and Sharma, 2017)

Sharifara, Mohd Rahim and Anisi gave a general review of face detection algorithms, explaining that face detection is a complex task because images of faces vary with regards to facial characteristics but also environmental conditions such as illumination. They explain that computational speed is a major factor to consider when developing a face detection algorithm and demonstrate that the Viola Jones face-detection algorithm is the most computationally efficient. However, there is a trade-off between computational speed and false detection rates. Because of this the authors recommend using the Viola Jones algorithm in parallel with a deep neural network, giving high speed and low false detection rates. (Sharifara, Mohd Rahim and Anisi, 2014)

The authors above show that the fastest and most computationally efficient algorithm is the Viola Jones face detection algorithm. They explain that the issues regarding false detection rates can be

overcome by combining the Viola Jones algorithm with a deep neural network. So, depending on how much face detection accuracy is an issue for any project that utilises the Viola Jones algorithm, there are methods that can overcome the shortcomings of it. For example, a face recognition system in a university building doesn't require as high accuracy as a face recognition system in a government building. The university building may not necessarily need an implementation using a deep neural network, whilst the government build probably would.

### Face Detection on a Raspberry Pi:

The final stage of the literature review focuses on implementations of a face detection algorithm on the Raspberry Pi. The findings show that most solutions are implemented using OpenCV, because OpenCV offers a pre-trained Viola Jones face detection algorithm on installation meaning that face detection can be implemented with minimal effort.

Bhanse and Jaybhaye used OpenCV on a Raspberry pi to implement the Viola Jones face-detection algorithm to detect and track a face in a video stream. They explored how running the Viola Jones algorithm at different resolutions impacts the performance of video capture and found that the Raspberry Pi could detect faces in a video with a resolution of 1080x720 at a framerate of 15FPS and with a resolution of 320x240 at a framerate of 25FPS. (Bhanse and Jaybhaye, 2018)

Tripathy and Daschoudhury also carried out research regarding the framerate of face detection on the Raspberry Pi. However, they found that using OpenCV with the Viola Jones face-detection algorithm they could carry out face detection and tracking at a framerate of 30FPS. (Tripathy and Daschoudhury, 2014)

Arva and Fryza carried out a study on existing motion and object detection algorithms that can be implemented on the Raspberry Pi. They examined the available face detection algorithms of OpenCV, as this is the most common used Computer Vision library for the Raspberry Pi. OpenCV has uses the Viola Jones face-detection algorithm for face detection but allows for this to be trained either using Haar Cascade classifiers or Local Binary Patterns (LBP) ([See Chapter 5](#)). LBP is an algorithm that matches the binary equivalent of an input image to that of a known set of images. The authors show that when training using LBP it takes around a quarter of the time than the Haar Cascade Classifier does. (Arva and Fryza, 2017)

Daryanavard and Harifi carried out research regarding face detection from an Unmanned Aerial Vehicle (UAV) system such as a drone. Their system was built on a Raspberry Pi that used the Viola Jones face-detection algorithm via OpenCV. They used a UAV facial image dataset to train the algorithm and found they had 98%, 93%, 86% and 80% True Positive for 1.5, 3, 4 and 5 meters height

of the camera from the ground respectively. However, their experimental results were not evaluated on a real time system. (Daryanavard and Harifi, 2018)

Umm-e-Laila et al. compared different face recognition algorithms that are available via OpenCV that can be implemented on a raspberry pi. They concluded that the Raspberry Pi is fast enough processor wise to detect and recognise faces in real time, but they don't comment on the frame rate, resolution or speed at which this is achieved. (Umm-e-Laila et al., 2017)

Gupta et al implemented a face detection and recognition system that could run on a Raspberry Pi. They used the Viola Jones face-detection algorithm to detect faces, and then passed the detected faces through a PCA based model in order to recognise faces. They claim that their solution can run in real time, but they don't comment on framerate, resolution or achieved detection speed. (Gupta et al., 2016)

## Conclusions

The task of detecting the emotional state of a person in a video frame is somewhat complex, but there are many different solutions to this. A common theme between implementations is the detection of a face prior to attempting to recognise the emotion in the face. A commonly used face detector is the Viola Jones face detection algorithm because of its simplicity and computational efficiency. In the literature above, multiple solutions utilise either the FisherFace or EigenFace technique to detect emotions once a face has been detected.

Almost all of the solutions on the Raspberry Pi examined above rely on the Viola Jones face detection algorithm for face detection. With this being run on the Raspberry Pi, but the resulting face is then sent to a more powerful machine for processing and emotion recognition. Most of these solutions implement the Viola Jones algorithm through OpenCV. OpenCV is an open source computer vision library that can run on the Raspberry Pi.

As the aim of this project is for the solution to be implementable on a Raspberry Pi, I wanted the Raspberry Pi to do the processing without developing an IoT based solution. After examining OpenCV I realised that it offers both a face detection and recognition module, with the detection module implementing the Viola Jones Algorithm, and the recognition module implementing either the FisherFace technique, the EigenFace technique or LPB. This led to the assumption that I could develop a system that can run natively on the Raspberry Pi based on OpenCV. Face detection could be carried out with the Viola Jones algorithm, and emotion recognition could be carried out by adapting any of the techniques available in the facial recognition module of OpenCV. As it is all implemented utilising

OpenCV, it should be deployable natively to the Raspberry Pi. (Assuming that it is computationally powerful enough to run the solution in real time)

## Chapter 3

### Problem Description and Specification

#### Problem Specification:

The first stage of this project was clearly specifying the problem that I aimed to solve. The project proposal gives an outline of the necessary functionality of the system. This is a starting point for requirements gathering, that could be expanded on. The project proposal follows:

“The aim of this project is to develop an application that can recognize emotions from facial features and display these in a meaningful manner (Statistics, Charts, etc.) These can then be used to replace questionnaires and interviews that are currently used as a market research tool to determine users’ opinions on new products. This will be developed using an embedded device (Raspberry Pi) to carry out the emotion capture and detection in order to demonstrate that portable solutions that don’t require excessive processing power can be developed for this type of system.”

#### Requirements:

The project proposal highlights four essential functions required by this system. The system must:

1. Recognise emotions in an input video stream
2. Display the recognized emotions in a meaningful manner (Have a User Interface)
3. Make use of an embedded device
4. Process an input video stream

Although the expected functionality of the system was outlined, it wasn’t nearly enough to start thinking about how the system would be designed, so requirements gathering had to be carried out. This would determine the functional requirements of the system, which is what the design of the system is based on.

#### User stories:

Gathering requirements can be done in many ways, but in the case of my system I didn’t have a customer or a specific user that I could interview or ask to fill in a questionnaire. This meant that I had to utilize a requirements gathering technique that didn’t rely on interaction with a potential user. The technique I followed was to develop User Stories, a tool that is commonly used in industry to determine additional requirements, remove assumed requirements that cannot be justified or help to restructure existing requirements. The form of the user stories I developed was: “As <type of user>, I need <functionality> so that <outcome of functionality>”. It is clear that by doing this any

requirements determined by the user stories have to be justified, meaning that they are all necessary for the system.

The User stories that I developed follow:

- As a User I need to know what the users being tested are feeling, so that I don't need to ask them their opinions.
- As a User I need to be able to store the recording of the video before and after the processing so that I can re-examine what the people like or dislike from what they see.
- As a User I need to be able to view statistics of how people that have seen the clip/product feel about it overall.
- As a User I need to log in to the application, so that I can protect the privacy of people being recorded.
- As a User I need to be able to see emotions of people in real time so that I can ask them about the things they didn't like (So that I know what they would like me to change).
- As a User I need to be able to export the overall results so that I can use them in market research.
- As a User I need to be able to view a library of stored videos so that I can replay the ones I choose.
- As a User I need to be able to test multiple people at the same time so that I can save time.
- As a User I need to be able to delete stored videos so that I can deal with mistakes.
- As a User I need to be able to take the application with me so that I can carry out Demos in different places. (Need a Portable solution)

Once the User stories had been developed, the functional requirements of the system could be derived as any requirement now being determined would be justified.

#### Functional Requirements:

A list of functional requirements based on the User stories was now determined, and the functionality of the system was determined as developing an application that would:

1. Recognise emotions in an input video stream in real time
2. Recognise emotions in a pre-recorded video stream
3. Display the recognised emotions in a meaningful manner
4. Make use of an embedded device (Portability)
5. Record a video stream
6. Store the recorded video stream (Create a Library)

7. Delete stored videos (From the Library)\*
8. Display the emotions of users in a video stream as the video is being displayed. (either real time or pre-recorded video clip)
9. Save an edited copy of the input video stream that displays the emotion detected.
10. Display statistics of the emotions a user in the video shows for the duration of the video. (happy 56% of the time they were in the video)
11. Allow the creation of sub-libraries of videos (For individual projects)\*
12. Generate an overall statistic of the emotions shown by multiple users in a library. (For one marketing project, 10 users where tested. These 10 Users appeared to be happy for 43% of the time they were in the video) \*
13. Export statistics so they can be used out with the application\*
14. Be capable of recognising emotions of multiple people at once
15. Have a User-friendly Interface
16. Have User accounts\*

These 16 requirements outlined the necessary functional requirements of the complete system, but at the current time some of them were deemed out of scope simply because of time restrictions. All requirements marked with a (\*) are deemed out of scope at the current time. Out of the requirements that were going to be implemented, prioritization was required to determine the most important functionality. The most important features would be implemented before any “non-high priority” requirements.

#### Requirements Prioritisation:

In order to prioritise the requirements of the system, the MoSCoW method was used. This is essentially a technique that splits the requirements into “Must Haves”, “Should Haves”, “Could Haves” and “Wont Haves (at this time)”. The way that the requirements where prioritized was based on what would prove that this concept would be feasible. So, the “Must Have” (highest priority) requirements where the ones that where necessary for the system to function and demonstrate that the concept is indeed feasible. The “Should Haves” were the requirements that would aid in solidifying the proof of concept but where not necessary for the system to function. The “Could Haves” were any additional features that would be nice to have but didn’t necessarily change the functionality of the system. The “Wont Haves” where the requirements that were deemed as additional features that where unnecessary for the system at the current time.

The way that the requirements where prioritized is demonstrated below:



*Must Have:*

1, 2, 3, 4, 5, 6, 8, 15

*Should Have:*

7\*, 9, 10, 14

*Could Have:*

11\*, 12\*, 13\*

*Won't Have:*

16\*

It was important to prioritise the requirements as the project was ambitious, and it would be extremely hard to satisfy all requirements in the available time frame. By prioritizing the requirements as mentioned above, it was possible to determine when the project would be “finished”. If all “must have” requirements are met, then the project would be in an acceptable state to prove that the concept is viable.

#### Development Methodology:

In the first stage of development, the project was split into submodules to be developed individually and then encapsulated into a web-application. Once individual modules were identified, an iterative approach to development was utilized, allowing for testing of individual components alongside development. Unit testing was carried out throughout ensuring that the expected output from individual components was the actual output.

## Chapter 4

### System Design:

This section of the report details how the system was designed before development commenced. In this section, the high-level design, low level design and system architecture are detailed. To begin with, high level design was tackled, allowing for sub-modules of the system to be identified. The low-level design covers the languages, frameworks and libraries that were used in development, but also justifies their selection. Finally, the system architecture is documented, which explains in more detail how sub-modules identified in the high-level design would interact.

### High Level Design:

At this stage, in order to produce a high-level design, the requirements that were determined earlier were revisited. The requirements determined the functionality of the system, which made it easier to produce a list of submodules that would implement this. This stage of development was vital because it essentially determined which modules would be developed to tackle each task.

Below is the list of identified submodules to carry out each task:

- Module that carries out Emotion Recognition:
  - Submodule for Face Detection
  - Submodule for Emotion Recognition
- Library Module: Record, Store, Delete video clips, Create additional sub-libraries
- Module that displays the results of the emotion recognition in video during playback
- Module that displays statistics of emotions shown for the duration of a video clip
- Module for exporting Statistics to usable format out with of the application
- Module of User Accounts: Create, Add, Delete, Log in
- User Interface Module
- Video Input Module (Either through a camera feed or a pre-recorded video clip)
- Video Recording Module

### Low Level Design:

It is important to remember that the solution being designed needed to be deployable on a Raspberry Pi too. This partially determined the development languages, tools and libraries that would be utilised in the development of the system, as everything needed to be compatible with the Raspberry Pi.

## Programming Languages:

### *Python:*

Python is a high-level, interpreted general purpose programming language. The selection of Python as the programming language was made for a few reasons. Python is a language commonly used in Data Analytics, Machine Learning and Computer Vision, meaning that there would be lots of support available if needed. OpenCV has a python interface, so importing OpenCV to the project would be easy to do. Having the Raspberry PI in mind, the language needed to be compatible with this and Python is pre-installed out of the box with Raspbian (Raspberry Pi's operating system).

### *HTML:*

Hypertext Mark-up Language (HTML) is the standard language used to design how webpages are displayed in a web-browser. HTML is the language that was chosen to develop the user interface as it has cross-browser support, it is relatively simple to understand and interfaces with Flask and Jinja2 templates.

### *Jinja2:*

Jinja2 is a Python templating engine that is used by flask and is necessary for flask to run.

### *JavaScript:*

JavaScript is an interpreted object-oriented scripting language used to create effects within web-browsers. JavaScript was used to interact between different sections of the application. Specifically, for handling requests to start and stop recording the emotion detection. The statistics that have been developed utilised a JavaScript library too.

## Libraries & Frameworks:

### *OpenCV:*

OpenCV is an open source computer vision library that was used for the Computer Vision Section of the system. It has been developed over time and is currently in a maturity stage where there is a big enough community so resources can be found with ease. Many of the projects examined in the literature review use OpenCV, and I have used this in the past so I was familiar with this and preferred it over other Machine Learning libraries that could be adapted for Computer Vision tasks.

### *Flask:*

Flask is a lightweight python web application framework that is designed to get a simple web application running quickly and easily. As the focus of this project was oriented towards emotion detection in real time, this was the ideal framework for the task.

#### *Chartist:*

Chartist is a JavaScript chart building library that was very simple to set up and implement on the system.

#### *Hardware:*

##### *PC-MacBook Pro:*

For the development of the project although it would have been possible to develop entirely on the Raspberry Pi, I didn't want to rely only on this especially when it came to re-training facial recognition algorithms. This is because although the Raspberry Pi model that I used was computationally powerful, the processing power of it couldn't compare to that of the processor in any modern-day computer.

##### *Raspberry Pi 3b+:*

The Raspberry Pi is a microcomputer that is small enough to fit in a pocket, meaning that a solution implemented on this would be portable. The Raspberry Pi is used by many hobbyists and researchers because of how lightweight and affordable it is. This means that there is a large community offering lots of support when needed.

##### *Raspberry Pi camera:*

The need for video input to the system led to the purchase of a Raspberry Pi camera module. This can easily be installed and set up and is affordable yet also produces video at a high enough quality to be used for the system. The Raspberry Pi camera has an 8 megapixel native resolution sensor, and supports video capture at resolutions of 1080p, 720p and 640x480 at framerates of 30, 60 and 90 FPS respectively.

#### *Development Tools:*

##### *PyCharm:*

PyCharm is a Python Development Environment used industry wide that I am familiar with and comfortable using, so it made sense to use this as the IDE for developing and debugging any code.

#### *System Architecture:*

Once the high-level design had been completed, I knew what submodules existed within the system and could therefore determine how they interacted between each other. Figure 1 below shows the interactions between the different modules of the system. This was helpful to understand how internal components of the system would be integrated and how the development of each would be implemented as it showed what dependencies exist between submodules.

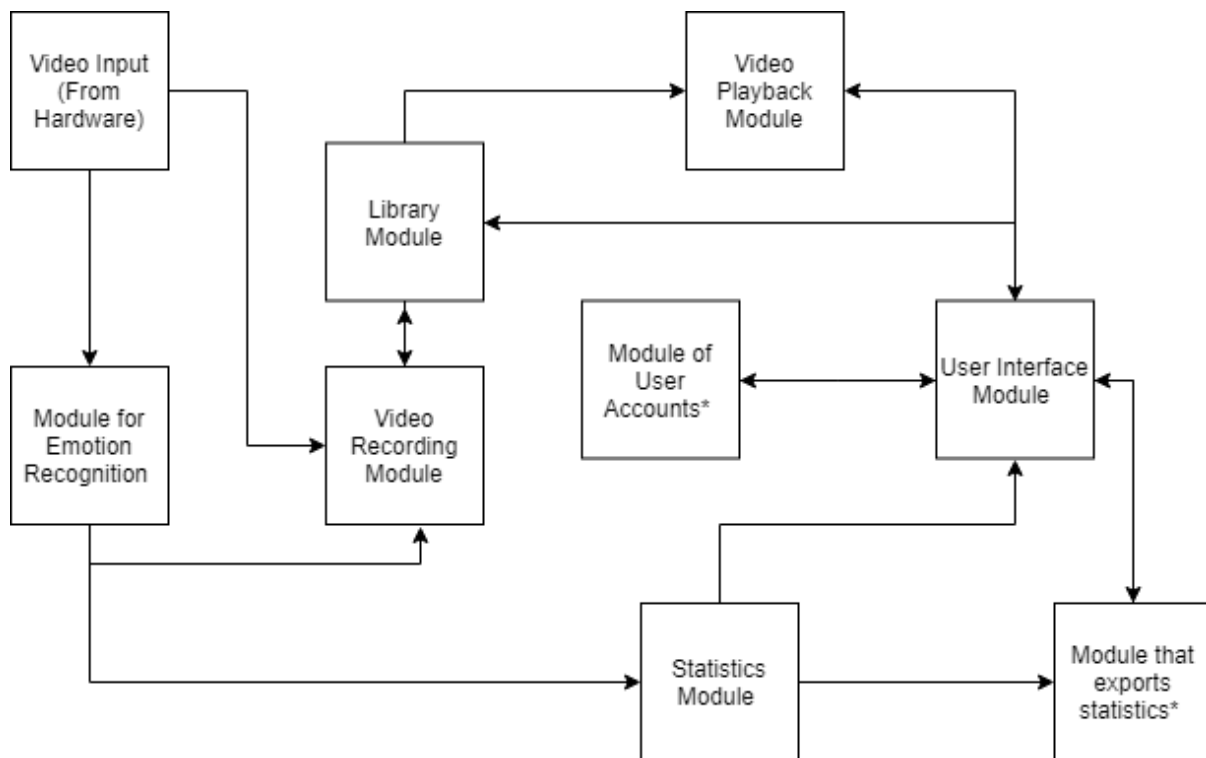


Figure 1 - Interaction of the systems Sub-Modules

As seen in Figure 1, the Video input module is used as an input source. The video that is inputted to the system can be sent to the emotion recognition module or the video recording module. The emotion recognition module uses the inputted video and attempts to recognise the emotion expressed by the person in the input. This can then be sent to the video recording module that saves the video clip to the library. The statistics module takes the output from the emotion recognition module and calculates the statistics of the emotions expressed in the throughout the duration of the video clip. The statistics can either be displayed to the user through the user interface module, or sent to the module in charge of exporting the statistics to a format that can be used out with the application. The library module holds all of the recorded video clips, either directly from the video input module or from the emotion recognition module. The contents of the library can be displayed in the user interface, but when a video clip is selected, it can be played in the video playback module, which is then displayed in the user interface. The module that holds the user accounts only interacts with the user interface module, as its only purpose is to authenticate the users attempting to use the system. As expressed in the requirements section of this report ([See Chapter 3](#)) the User Accounts Module and the Module that exports the statistics will not be developed at this time. (Any modules flagged with a (\*) is out of scope at the current time)

## Chapter 5

### Detailed Design and Implementation:

In this section of the report, the system design and implementation are explained. The biggest challenge of this project was to successfully develop an emotion detection module that could run in real time and be deployed to the Raspberry Pi. This meant that the design had to be split into two sections, the Emotion detection module and the Web-Application and for them to be developed separately.

#### Emotion Detection:

The literature review examined earlier showed that multiple solutions to the emotion detection problem were implemented using deep learning, the FisherFace or the EigenFace algorithms. Choosing OpenCV as the computer vision library to be utilised for this project meant that I had access to both face detection and recognition algorithms. For face detection, deep learning or the Viola Jones algorithm can be used whilst for recognition, either the FisherFace, EigenFace or LBP algorithms can be used. My goal at this point was to retrain the face recognition module to recognise emotions rather than any individuals face.

#### Methodology:

The literature review showed that the methodology followed by many of the authors when attempting to recognise emotions is the same. There are 6 essential steps that are taken:

1. Split video clip into individual frames for processing
2. Individual Frame gets passed to face detector.
3. Face is detected and extracted by the face detector.
4. Extracted face is passed to emotion recognition module.
5. Emotion recognition module identifies current emotion of face passed to it.
6. Repeat from step 1.

It made sense for me to follow the same steps as the authors in the literature, as I was attempting to build a system that could carry out the same tasks as theirs.

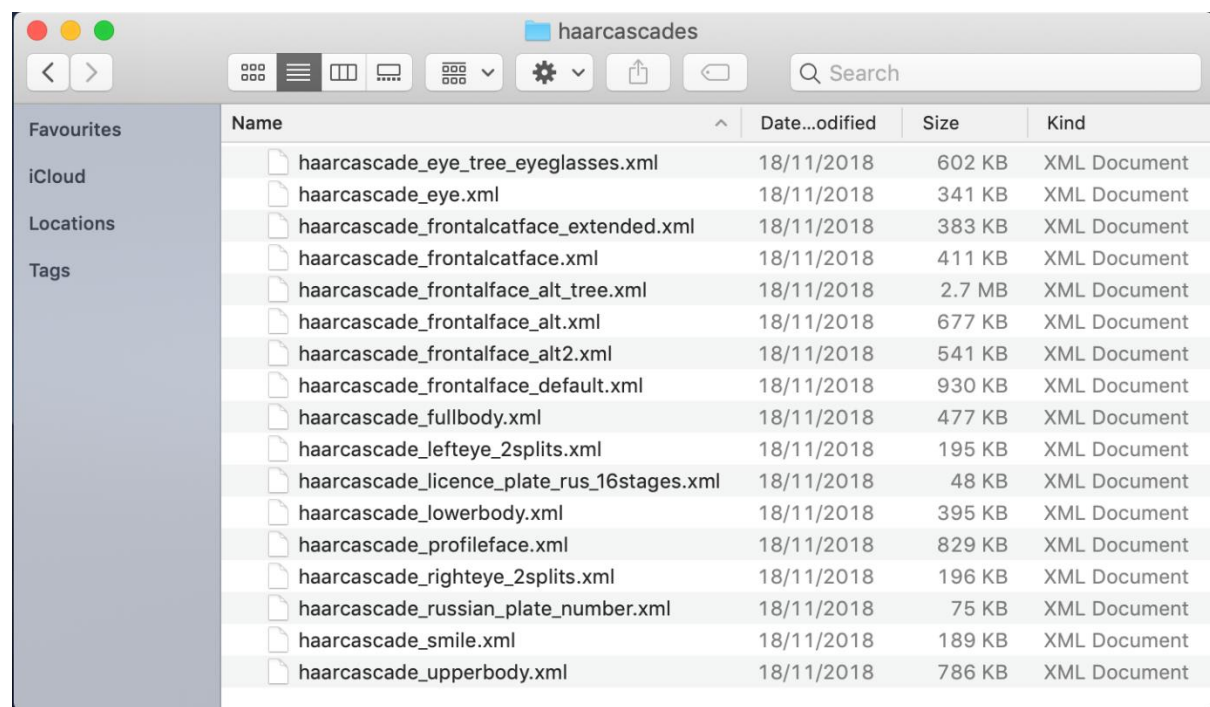
#### Face Detection:

The first step in tackling the problem of emotion recognition is face detection. I was using OpenCV, so I searched and found the algorithms that are available and pre-trained to tackle this task. I found that

there are two available, the Viola Jones face detection algorithm and a deep learning algorithm adapted for detecting faces.

#### *Viola Jones:*

The Viola Jones algorithm is based on Haar-Cascades classifiers, and OpenCV comes with both a Haar-Cascade trainer and detector. This means that it is possible to train your own classifier to detect any object that you like, but also run the object detector that implements the model that has been trained. The nice thing about OpenCV is that with installation, several pre-trained classifiers are installed and can be used by the Cascade Classifier module. Upon inspection of the pre-trained classifiers available with installation, I found that there are 17, of which 5 are face detection classifiers. For this project, I used the default face detection classifier, as posts and blogs such as <https://www.pyimagesearch.com/>, <https://towardsdatascience.com/> and reddit that I had read regarding this seemed to find that this was the most reliable one.



Name	Date...odified	Size	Kind
haarcascade_eye_tree_eyeglasses.xml	18/11/2018	602 KB	XML Document
haarcascade_eye.xml	18/11/2018	341 KB	XML Document
haarcascade_frontalcatface_extended.xml	18/11/2018	383 KB	XML Document
haarcascade_frontalcatface.xml	18/11/2018	411 KB	XML Document
haarcascade_frontalface_alt_tree.xml	18/11/2018	2.7 MB	XML Document
haarcascade_frontalface_alt.xml	18/11/2018	677 KB	XML Document
haarcascade_frontalface_alt2.xml	18/11/2018	541 KB	XML Document
haarcascade_frontalface_default.xml	18/11/2018	930 KB	XML Document
haarcascade_fullbody.xml	18/11/2018	477 KB	XML Document
haarcascade_lefteye_2splits.xml	18/11/2018	195 KB	XML Document
haarcascade_licence_plate_rus_16stages.xml	18/11/2018	48 KB	XML Document
haarcascade_lowerbody.xml	18/11/2018	395 KB	XML Document
haarcascade_profileface.xml	18/11/2018	829 KB	XML Document
haarcascade_righteye_2splits.xml	18/11/2018	196 KB	XML Document
haarcascade_russian_plate_number.xml	18/11/2018	75 KB	XML Document
haarcascade_smile.xml	18/11/2018	189 KB	XML Document
haarcascade_upperbody.xml	18/11/2018	786 KB	XML Document

*Figure 2 - Available Haar Cascade Classifiers*

#### *Deep Learning:*

The Deep Learning module of OpenCV is based on the “tiny-dnn” framework, which is a dependency free C++ open source deep learning library. The deep learning module of OpenCV also comes with some pre-trained models such as object detection, text detection and face detection. Again, the model of interest for this project is the face detector, so in order to access the pre-trained model, I had to run a script located in the installation files of OpenCV on my machine. This script basically initiated a

download for a file containing the calculated weights of the Deep Learning module that had been trained for face detection.

Name	Date...odified	Size	Kind
classification.cpp	18/11/2018	5 KB	C++ Source
classification.py	18/11/2018	4 KB	Python Source
CMakeLists.txt	18/11/2018	681 bytes	Plain Text Document
colorization.cpp	18/11/2018	7 KB	C++ Source
colorization.py	18/11/2018	3 KB	Python Source
common.hpp	18/11/2018	4 KB	C++ Header Source
common.py	18/11/2018	5 KB	Python Source
custom_layers.hpp	18/11/2018	11 KB	C++ Header Source
edge_detection.py	18/11/2018	3 KB	Python Source
face_detector	27/07/2019	--	Folder
fast_neural_style.py	18/11/2018	2 KB	Python Source
js_face_recognition.html	18/11/2018	7 KB	HTML document
mask_rcnn.py	18/11/2018	5 KB	Python Source
mobilenet_ssd_accuracy.py	18/11/2018	5 KB	Python Source
models.yml	18/11/2018	3 KB	Document
object_detection.cpp	18/11/2018	10 KB	C++ Source
object_detection.py	18/11/2018	10 KB	Python Source
openpose.cpp	18/11/2018	6 KB	C++ Source
openpose.py	18/11/2018	5 KB	Python Source
README.md	18/11/2018	5 KB	Markdown Document
segmentation.cpp	18/11/2018	8 KB	C++ Source
segmentation.py	18/11/2018	5 KB	Python Source
shrink_tf_graph_weights.py	18/11/2018	2 KB	Python Source
text_detection.cpp	18/11/2018	6 KB	C++ Source
tf_text_graph_common.py	18/11/2018	10 KB	Python Source
tf_text_graph_faster_rcnn.py	18/11/2018	11 KB	Python Source
tf_text_graph_mask_rcnn.py	18/11/2018	9 KB	Python Source
tf_text_graph_ssd.py	18/11/2018	12 KB	Python Source

Figure 3 - Available deep learning models

### Emotion Detection:

At this stage of development, it was clear that I would need to re-train a face recognition algorithm to recognise emotions instead of face, so before diving into this I had to do some background research on machine learning. In summary, machine learning is the science of teaching a computer to learn from data. (Géron, n.d.) There are many different machine learning algorithms, and different algorithms are fit for different tasks, but regardless of the algorithm, they all have common elements:



- Data: All machine learning algorithms rely on data, as this is used to train the system. (Paluszek and Thomas, 2017) In my case, the training data was pictures of faces expressing an emotion.
- Model: The model of a machine learning algorithm is a mathematical framework developed for learning. (Paluszek and Thomas, 2017) In my case, the model was either one of the three face recognition algorithms provided by OpenCV.
- Training: In the sense of machine learning, training is the task of modifying the model so that for a given input, an output can be mapped. For example, a face detection algorithm can be fed thousands of pictures of faces (training data) and be told that these pictures all contain faces. Then it can be fed thousands of pictures of anything but faces and be told that these pictures do not contain any faces. Once the training has finished, the model should be modified so that when a new picture is input to the system it should be able to tell on its own if this picture contains a face. (Paluszek and Thomas, 2017)

## Learning phase

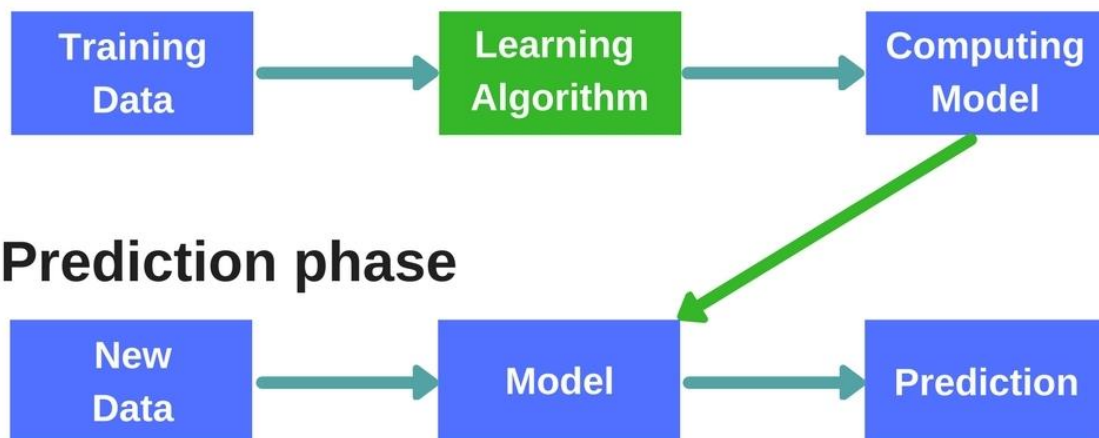


Figure 4 - Machine Learning training and Predicting (Onix-systems.com, 2019)

For the emotion recognition module of the system as no model is included, a dataset was needed, the algorithm was provided and then the training needed to be carried out. The adopted approach was as follows. First data must be collected, this needed to be images of people's faces expressing an

emotion. The data needed to be sorted, so that subsets of data could be produced that contained an emotion. For example, a subset was “happy” and this contained pictures of people that were happy. Next the faces needed to be detected and extracted from the dataset and fed into the face recognition algorithm with its labels. So, all faces extracted from the “happy” subset were fed to the face recognition algorithm with the label of happy. Once training had finished, when a new happy face was fed into the system, it should have given an output indicating that “happy” was detected. In a sense I attempted to “trick” a face recognition algorithm, by telling it that every happy face corresponds to an individual labelled happy.

#### *Data Collection:*

The first step in training the emotion recognition model was collecting data that could be used for training. This had to be images of people that would clearly show the emotions that they are feeling, for example someone that was feeling angry needed to be frowning and indicating that they are indeed angry. Luckily, this area of research is not new, so there are several existing datasets that are freely available for academic research. After some online searching, I came across a website called [www.face-rec.org](http://www.face-rec.org), which is essentially a website with lots of information regarding face recognition, with lots of source code, examples and academic literature related to this field. It also has lots of academic datasets that are available to download and use for academic research. A lot of the datasets require proof that this will be used for academic purposes, so a sign up with a university email address is needed to access the dataset. The sets of data that I used in my case are:

- The Cohn-Kanade (extended) facial expression database: This is a database developed by Takeo Kanade, Jeffrey F. Cohn and Yingli Tian, that contains labels of emotions expressed by the subjects in images along with their facial action units. (Kanade, Cohn and Yingli Tian, n.d.) Facial action units refer to actions taken by the face when expressing an emotion. For example, when smiling people’s mouth, cheeks and eyes tend to move. The database contains 593 images and is available here: <http://www.consortium.ri.cmu.edu/ckagree/>
- The Japanese Female Facial Expression Database: This is a database developed by Michael J. Lyons, Shigeru Akemastu, Miyuki Kamachi and Jiro Gyoba, that contains images of Japanese females expressing different emotions. Each picture has an explanation of the current emotion shown by each subject in the name of the file corresponding to the picture, so all pictures of people expressing happiness had the tag “HA” in their file names. The database contains 214 images and is available here: <http://www.kasrl.org/jaffe.html>
- A database developed by Paul van Gent, that contains images found using google image search that he shared in a blog post about an emotion detection project. He was kind enough to share the dataset with the public, which meant that it could be downloaded and used by anyone.

The database contains 181 images and is available here: <http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/>

- The FACES dataset: This is a dataset developed by Natalie C. Ebner, Michaela Riediger, and Ulman Lindenberger, that contains images of six individuals, three male and three female. Of each of these three male and female subjects, one is young, one is middle aged and one is old. The dataset contains images of each individual expressing 6 emotions (neutrality, sadness, disgust, fear, anger, and happiness) twice, for a total of 72 images. This dataset is publicly available here: <https://faces.mpg.de/imeji/collection/IXTdg721TwZwyZ8e?q=>

#### *Data Sorting:*

Now that I had access to data, it was time to sort the data so that it could be used to train the emotion recognition algorithms.

The Cohn-Kanade database contains 593 images, but only a subset of these have labels for the emotions expressed in the images. This meant that the database had to be sorted so that only the images expressing emotions were used. To do this, I adapted a script written by Paul van Gent that would do just this. When downloading the Cohn-Kanade database, there is a folder that contains the images and a folder that contains the emotion code for the corresponding image (if that image has an emotion code). The emotion codes are: 0=neutral, 1=anger, 2=contempt, 3=disgust, 4=fear, 5=happy, 6=sadness, 7=surprise. This meant that for each subject, the corresponding emotion is recorded, and therefore by looping through the two directories, images can be moved to directories that correspond to emotions. For example, I could copy all images with the emotion code 1 to a directory named "Anger". The adapted script can be found in [Appendix B](#)

The JAFFE database was much easier to sort, as the name of the file gave the emotion of the individual in that picture. This meant that in order to find all images of happy people, I could just search the directory containing the images for all files with the characters "HA" in their names. I then manually copied these into new directories corresponding to each emotion, so that they matched the structure of the sorted Cohn-Kanade dataset.

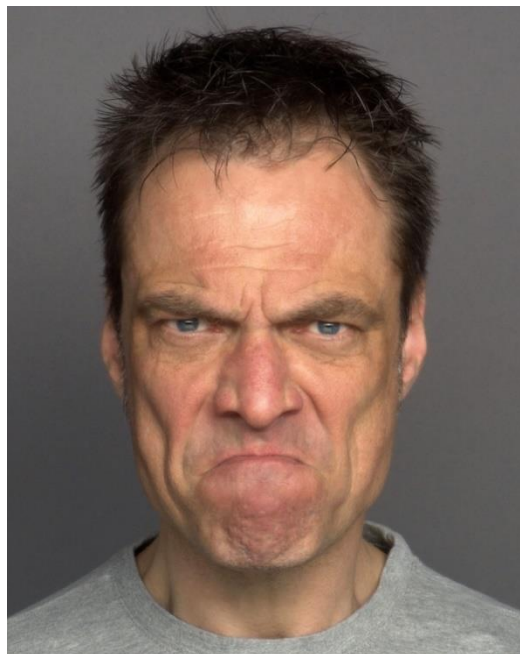
The FACES dataset could be downloaded in batches of each emotion, so when downloading the images they were pre-sorted by emotion expressed by individual meaning that no further sorting was carried out.

The database developed by Paul van Gent was already sorted into subdirectories, so no additional sorting was necessary.

The difference between the datasets is that the Cohn-Kanade dataset contains images of people expressing contempt whilst the other three do not. For the system being developed, I deemed that contempt is unnecessary, because faces expressing contempt closely resemble those expressing anger. Additionally, the other three datasets didn't contain images of people feeling contempt, so there would be an uneven number of images in the training data. All datasets contained images of people feeling the following 6 emotions: neutrality, anger, disgust, fear, happiness, sadness and surprise. Therefore, these were the emotions that were selected to be detected in the system.

#### *Face Detection and extraction:*

The next step in developing the emotion recognition module was to extract the faces from the datasets so that they could be used for training the emotion recognition model. Faces needed to be extracted from the datasets because the approach to emotion recognition relies on extracting a face from an image and then the corresponding emotion of that face to be detected. This means that the model expects a specific sized image of a face. So, in order to ensure that all images of faces were the same size I used the different algorithms available with OpenCV to detect and extract faces.



*Figure 5 - Original face*



*Figure 6 - Extracted face*

As previously explained, with the installation of OpenCV there are two pre-trained face detectors, the Viola Jones Haar Cascade-Classifier and the Deep learning model adapted for face detection. For it to be possible to implement either of these methods, and make sure that they were usable to detect faces, at first I wrote two simple face detection scripts, that would open a camera feed, break the video clip into individual frames then detect a face in the frame and draw a rectangle around the face. Once I was sur that these scripts ran successfully, I adapted them so that they would extract the face detected in the input image. This was done by cropping the input image to the dimensions of the rectangle containing the face at the coordinates of the rectangle in the image.

All three face recognition algorithms expect a specific input image size for a detected face, which meant that when cropping the images, I had to specify the size of the extracted face. If the sizes didn't match, then there would be errors when attempting to re-train the face recognition algorithms.

#### [Viola Jones:](#)

Once I had successfully adapted the scripts mentioned above, I was in a position to carry out the face extraction for each image in the datasets that I had acquired. When using the Viola Jones face extraction script, for each input image a corresponding output image was created. This meant that the Viola Jones face extraction script had a 100% success rate, which was optimal for the system.

#### [Deep Learning:](#)

Next I ran the face extraction script that used deep learning to detect faces in images and extracted all the faces detected in the previously acquired datasets. I found that the deep learning module failed to detect faces in many of the input images, and also wrongfully detected faces in many of the input images.

Figures 7 and 8 below demonstrate an example of a wrongfully extracted face.



*Figure 7 - Original face*



*Figure 8 - Wrongfully extracted face*

Before extracting faces using either of the algorithms, I expected that the Deep Learning face detector would outperform the Viola Jones algorithm, but when it came to detecting faces in images the Viola Jones algorithm appeared to be superior. It was interesting to see this because initial tests using the two face detectors showed that the Deep Learning face detector was more robust than the Viola Jones when detecting a face in a video stream. The Deep Learning face detector handled differences in lighting and occlusion of the face better than the Viola Jones when testing it with a video input. As the methodology that I was adopting to detect emotions in a video stream relied on splitting the video into individual frames, and then detecting the face in the frame before any further processing was carried out, the decision was made not to use the Deep Learning face detector for the project. The deep learning module also required more processing than the Viola Jones algorithm and as this project was intended to be as lightweight as possible, it made sense to use the Viola Jones algorithm.

#### *Emotion Detection Training:*

Now that I had extracted the faces from the original datasets, I was left with four new datasets containing images of faces expressing emotions with the images sorted into their corresponding

directories. The next step in the process was to train the face recognition model to recognise emotions with the “newly” obtained datasets. The face recognition module of OpenCV comes with three pre-trained face recognition algorithms. These are:

- EigenFaces: This is a technique that extracts the variance in facial features for each image in the training set. Then when the model is used, it tries to match the features extracted from the training set to the actual input data’s features.
- FisherFaces: Fisherfaces works in a similar way to EigenFaces, but with the difference that with the FisherFaces technique, instead of extracting the variance in facial features, it extracts the features that differentiate one person from another.
- LBP: This is a technique, where a 3x3 filter is passed over each pixel in an image that returns the binary equivalent value of the pixels within the filter. This binary value is dependent on the intensity value of each individual pixel. Then a Histogram is created that holds all the binary values for each face used in the training set. Then when a face is to be recognised, once it is passed through the LBP recognition model, the model attempts to match the current face’s LBP with one from the training set.

#### The training plan:

In order to develop the best performing emotion recognition module, I had to train a series of different models with different training data. This way I could evaluate the performance of the algorithms and determine the best combination for the application. Initially, I intended on using both face detection modules to see which one would yield in better results, but as explained above the deep learning module wrongfully detected too many faces, so it wasn’t possible to use this as the face detector of the system. What was left to compare was the different face recognisers and different combinations of datasets.

#### FisherFaces vs EigenFaces vs LBP:

In the first stage of training the emotion recognition model, I started out by using the entire dataset to train all three facial recognition algorithms. This way, I would be able to determine the best performing algorithm and use this as my emotion recognition model. So as stated above, I now had a training dataset that contained all the images of all the emotions in their respective folders (folder named happy contained all the images of happy faces, folder named sad contained all the sad faces, etc.) What was left to do was to train the emotion recognition algorithms and compare their performance. Figure 9 below shows the method by which this was carried out. All algorithms were trained using the same datasets to make the comparison fair. Once the algorithms were trained, I was left with three emotion recognition models that could be tested.

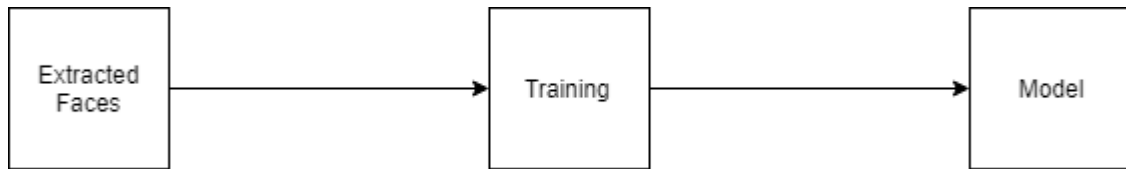


Figure 9 - Training emotion detection model

The approach to training the algorithms was as follows. First all the images for each emotion are shuffled and split into two separate sets, a test set and a training set. Then the training set is passed to the face recognition trainer with the corresponding label of each emotion in the set. Once the training has finished, the now trained model is used to try and predict the emotion in the prediction set. If the prediction is correct it is added to a variable that holds the correct prediction score, if it is incorrect then it is added to a variable that holds the incorrect prediction score. This is then used to calculate the accuracy of the algorithm. The process is repeated 10 times to make sure that the calculated prediction score is consistent. Finally, the emotion detection model is saved. The script that was run to train the models can be found in [Appendix C](#).

The output of the system once the training was complete is shown in figure 10 below.

```

/Users/yenji/.virtualenvs/cv/bin/python3.6 -m trainAlgorithms
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 71.01449275362319 (FisherFace) percent correct. Got 55.55555555555556 (EigenFace) percent correct. Got 57.48792270531401 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 73.91304347826087 (FisherFace) percent correct. Got 56.03864734299517 (EigenFace) percent correct. Got 57.00483091787439 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 71.98067632850241 (FisherFace) percent correct. Got 55.072463768115945 (EigenFace) percent correct. Got 56.52173913043478 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 73.42995169082126 (FisherFace) percent correct. Got 54.106280193236714 (EigenFace) percent correct. Got 55.55555555555556 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 75.3623188405707 (FisherFace) percent correct. Got 57.00483091787439 (EigenFace) percent correct. Got 57.48792270531401 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 69.56521739130434 (FisherFace) percent correct. Got 53.6231884057971 (EigenFace) percent correct. Got 55.072463768115945 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 65.70048309178743 (FisherFace) percent correct. Got 51.690821256038646 (EigenFace) percent correct. Got 51.207729468599034 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 77.77777777777777 (FisherFace) percent correct. Got 54.106280193236714 (EigenFace) percent correct. Got 56.03864734299517 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 71.4975845410628 (FisherFace) percent correct. Got 53.14009661835749 (EigenFace) percent correct. Got 56.03864734299517 (LBPH) percent correct
training FisherFace, EigenFace and LBPH classifiers
size of training set is: 837 images
predicting classification set
Got 71.4975845410628 (FisherFace) percent correct. Got 51.207729468599034 (EigenFace) percent correct. Got 54.106280193236714 (LBPH) percent correct

end score:(fisher) 72.17391304347825 percent correct!

end score:(eigen) 54.15458937198067 percent correct!

end score:(lbph) 55.56217391304348 percent correct!

Process finished with exit code 0
  
```

Figure 10 - System output after training



It can be seen that the FisherFaces algorithm outperforms the other two by roughly 20%. This is a significant number especially when the correct prediction rate jumps from around 55% to 75%. This is a difference of 1 in 5 predictions, which is significant for this system. Because of this big variance in results, the decision was made to only use the FisherFaces based model for the system. I believe that this is justifiable simply because a correct prediction score of around 55% means that the system will predict the wrong emotion around half of the time, which would make the system useless. Even if the correct prediction rate can be increased by changing the training set, the difference in results would not be significant enough to justify using either of the other two algorithms for this system.

#### Entire Dataset vs (Entire Dataset – JAFFE):

The next step was to attempt and increase the performance of the emotion recognition algorithm. At the current time, the only way that this could be done is by changing the training dataset. I made the assumption that the JAFFE database was responsible for some of the wrong predictions when training the algorithm, simply because of the difference in appearance of Japanese faces in comparison to those of Caucasian people. So, in order to test this hypothesis, I retrained the emotion detection algorithm, but this time by removing the JAFFE database from the training set. This meant that I had to restructure my training dataset and retrain my emotion detection model again, but at this time I had already decided that I was only going to use the FisherFace algorithm as this gave the best results in my previous tests. I tweaked the training script, so that it would be the same as the original, but with the difference of only training one algorithm rather than three.

The output of the newly trained algorithm is shown in Figure 11 below.

```

/Users/yenji/.virtualenvs/cv/bin/python3.6 -m train1Algorithm
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 67.87878787878788 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 71.51515151515152 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 74.54545454545455 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 73.33333333333333 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 71.51515151515152 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 70.3030303030303 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 80.0 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 68.48484848484848 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 75.15151515151516 percent correct!
training fisher face classifier
size of training set is: 668 images
predicting classification set
got 69.0909090909091 percent correct!

end score: 72.18181818181817 percent correct!
Process finished with exit code 0

```

*Figure 11 - System output after training*

The performance increase when removing the JAFFE database from the equation was not significant, as seen above. However, in real life test scenarios the emotion detection system seemed to perform better when this was removed. I suspect that this is because I had only tested the outcome of the algorithm on myself, a Caucasian. I believe that by removing the Japanese dataset, it made it easier for the algorithm to make predictions because it was only trained with pictures having similar facial features as the people testing it (Myself).

#### More improvements:

The reliability of a machine learning algorithm usually depends on the dataset that is used to train it. Typically, with more data the algorithm gets better, but at this time, I couldn't get hold of more training data and therefore could not attempt to increase its performance by growing the dataset. Another way that this could be improved is to remove data that is similar or merge it into one section of the dataset. For example, someone looking surprised can easily be confused with somebody looking happy, or somebody looking disgusted could be confused with someone looking angry. As demonstrated in Figures 12 and 13 below.



*Figure 12 - Surprised face*



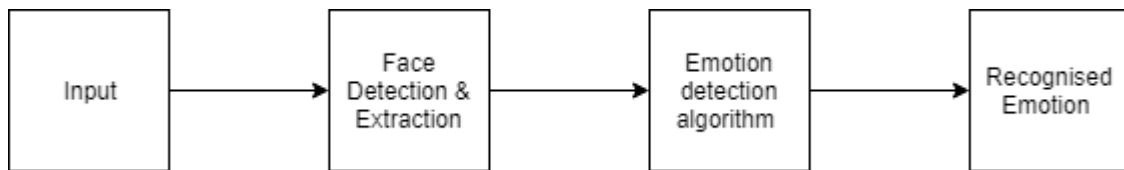
*Figure 13 - Happy face*

However, by removing similar emotions from the set, or by merging the data in the subsets again it would take away from the performance of the system as a whole. There is a trade-off between correct detection rate and number of emotions that can be detected, but I was happy with a correct detection rate of around 60%, so I deemed it unnecessary to remove emotions from the datasets. The purpose of this application was to prove that this concept is achievable, so developing a system that predicts the emotion correct 3 out of 5 times still does just this.

#### [Final Emotion Detection Module:](#)

Once I was happy with the performance of the emotion recognition model, I had to incorporate it to the system, which meant that I had to develop the subsystem that would be in charge of emotion detection. As shown by Figure 14 below, the general idea was for this module to accept an input video stream, or an input from the camera. From this input the face detector detected and extracted faces that were fed to the emotion recogniser. This then output the (predicted) recognised emotion of the

face detected in the video input. Figure 14 below shows how the emotion detection module of the system functions.



*Figure 14 - Emotion detection methodology*

This was then re-structured, so that instead of it being an executable script it became a class that could be imported by other modules of the system, by creating an emotion detection object and calling its subsequent methods.

### Web Application:

As stated at the start of this chapter, the development methodology was geared around developing a two-part system, with the second part of the system being based on the first. This meant that now that I had the emotion recognition section complete, I could move on to develop the Web-Application that housed the entire system.

### Methodology:

As this was a web-application system, it meant that it had to be split up into a front and back end. The back end of the system is where all the functionality of the modules identified in the design chapter would go, with the front end allowing for interaction between modules or displaying the systems outputs.

### Back end:

The back end of the system was comprised of the following sub-modules:

- Emotion detection module: This was the module that had just been developed. It was in charge of detecting emotions from a given input video stream.
- Video recording module: When first developing this module of the system, it seemed like a fairly straightforward task because OpenCV offers the functionality of recording the output video stream after image processing has been carried out. However, with OpenCV the method of stopping the recording is with a hardware instantiated interrupt (usually pressing the 'q' key). In order to adapt this so that it could be stopped with a software instantiated action, I had to run the recording module on a second processing thread. This meant that when I

wanted to start recording, I would start the second thread and to stop recording I would end the process of the second thread.

- Library module: For the prototype system, the library module was a local directory that was used to store the recordings of the video recording module. This was essentially a directory containing video clips that could be accessed by the Video Playback Module.
- Video Playback module: This module was in charge of playing the recorded video clips. It would take the file-path of the desired video clip to be played and would play the video clip using OpenCV. This was the chosen method of playing a video clip as it meant that I didn't have to import any additional libraries to deal with video playback.
- Statistics module: As the name suggests, this module was in charge of outputting the statistics of a recorded clip. In the current version of the system, the emotion detection module saves the times that each emotion is detected in the input stream to an array that is returned once emotion detection has finished. The array that is returned holds the timings of each emotion displayed in an index that corresponds to each emotion. For example, index 0 was used to hold the overall duration that the neutral emotion was detected in the input stream. This local array was then accessed by the statistics module and a chart would be built using chartist, a JavaScript library designed to build interactive charts very easily.

#### Front end:

The front end of the system was a HTML based front end, used to switch navigate different pages of the system. JavaScript was used to deal with interaction between the front and back end of the system. Specifically, to handle a request to start and stop recording the output of the emotion recognition module. There were 5 pages developed for the front end of the system:

- Home Page: At the current time, the homepage was a page that contained navigation buttons to the other pages of the system with a title explaining what page it is.

Home Library Emotion Detection Statistics

**This is the Home Page!**

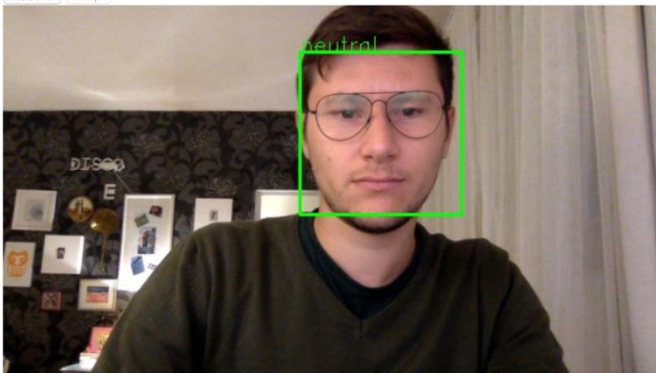
*Figure 15 - Home page*

- Emotion Detection Page: This is the page that emotion detection was carried out on. On this page, real time emotion recognition is carried out and the output of it is displayed on the page. There are also two additional buttons, one to start recording and one to end it. Because the output of the emotion detection is displayed on this page, it meant that the video clip currently being recorded was displayed as it was being recorded.

Home Library Emotion Detection Statistics

**This is the emotion detection page**

Record Stop



The detected emotion is shown in the rectangle around the face.

*Figure 16 - Emotion detection page*

- Library Page: For every video clip stored in the library a HTML button was generated that would redirect the application to the video playback page, where the video that was clicked on would be played in the browser.

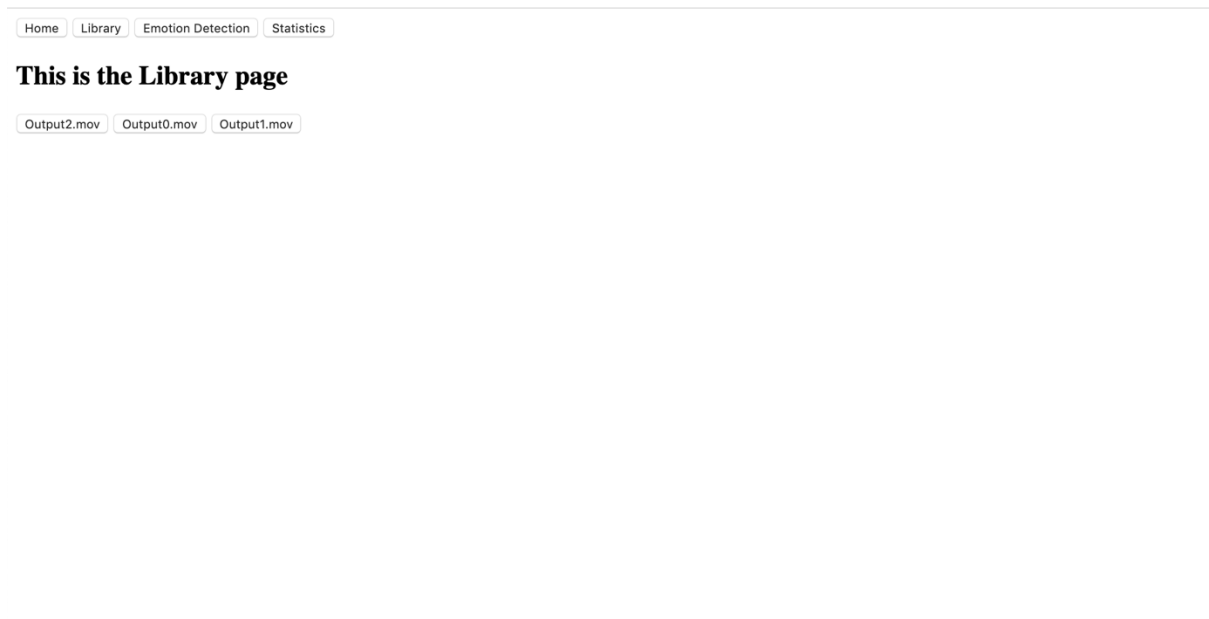


Figure 17 - Library page

- Video Playback Page: This page was generated once a video was clicked on in the library. If no videos existed in the library, then no video playback pages existed either. Once a video was clicked on and this page was displayed, it contained a video player in the browser that would play the video that was selected.

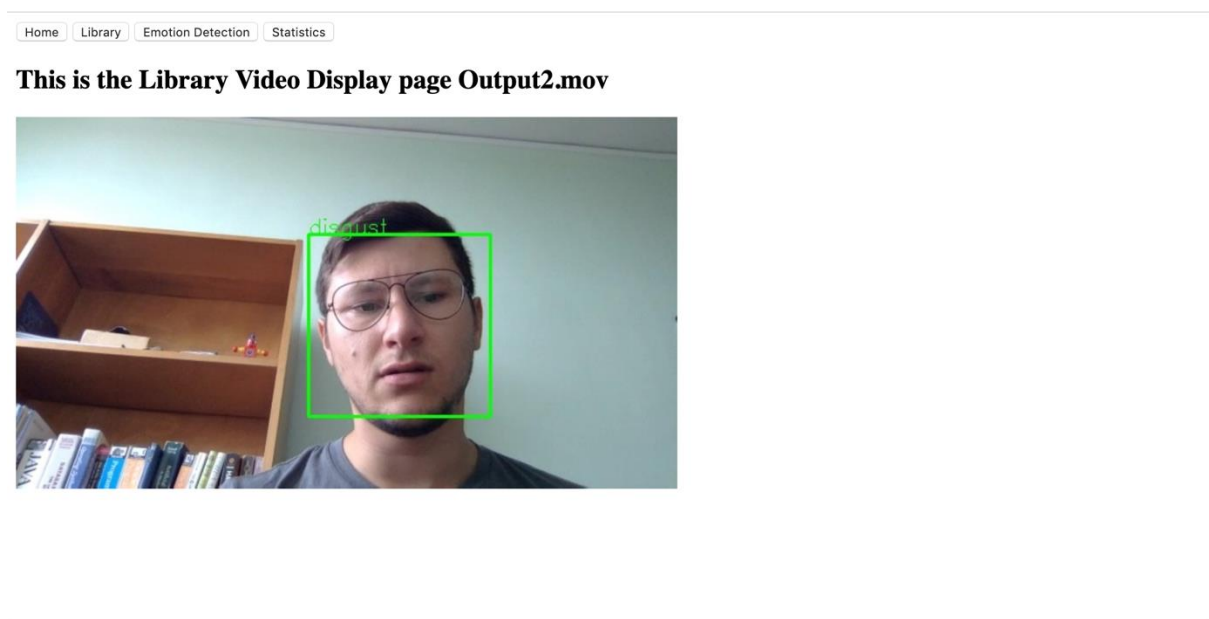


Figure 18 - Video playback page

- Statistics Page: This is the page where the duration of each emotion felt was displayed in a bar chart. The X axis of the chart displayed the different emotions that could be detected, with the Y axis displaying the time duration that emotions had been detected.

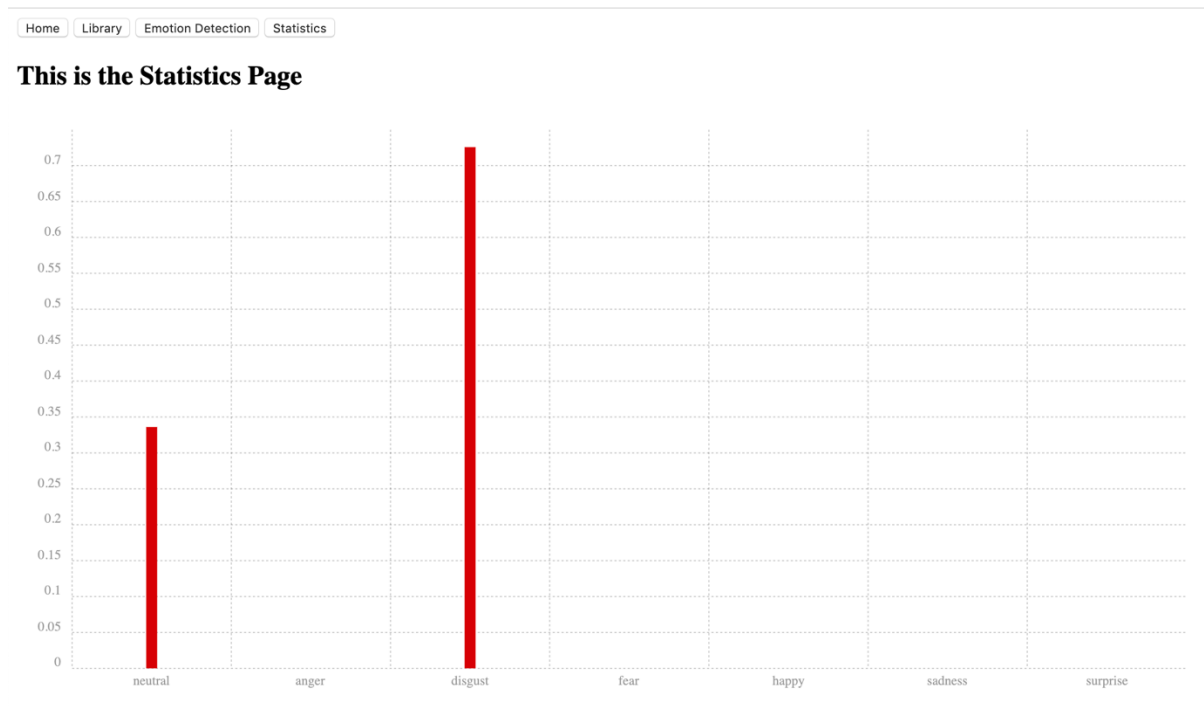


Figure 19 - Statistics page

By using Flask to build the system, I was able to use the so called “Jinja2” template engine. This meant that I could build a HTML page that other HTML pages inherited from. I used this to build the homepage and then each other page in the system would inherit from it. As it only contained the navigation buttons, it meant that all subsequent pages of the system would now contain the navigation buttons too.

#### Integration of back end with the front end:

Flask was used to build the web-application, as this allowed for the system to be hosted on a local web-server without the need of much configuration. As previously explained, Flask was chosen to develop the web-application because of how lightweight and straightforward it is to set up. Because this is a python web-application framework it meant that I could import and run all of my previously developed python code, without having to change too much of the code.

The development approach followed an MVC pattern, which is a development pattern that splits an application into three components, the Model, the View and the Controller as seen in Figure 20. With MVC, the model represents all of the back-end functionality of the system, where any operations on data are carried out. The View represents all of the front-end functionality, which is essentially the



user interface. And the controller is basically an interaction method between the model and the view. In the case of the system that I developed, the Model was all the back end modules of the system, the View was the front end modules, and the controller was the Flask application, as this would handle user requests and share data between the back and front end.

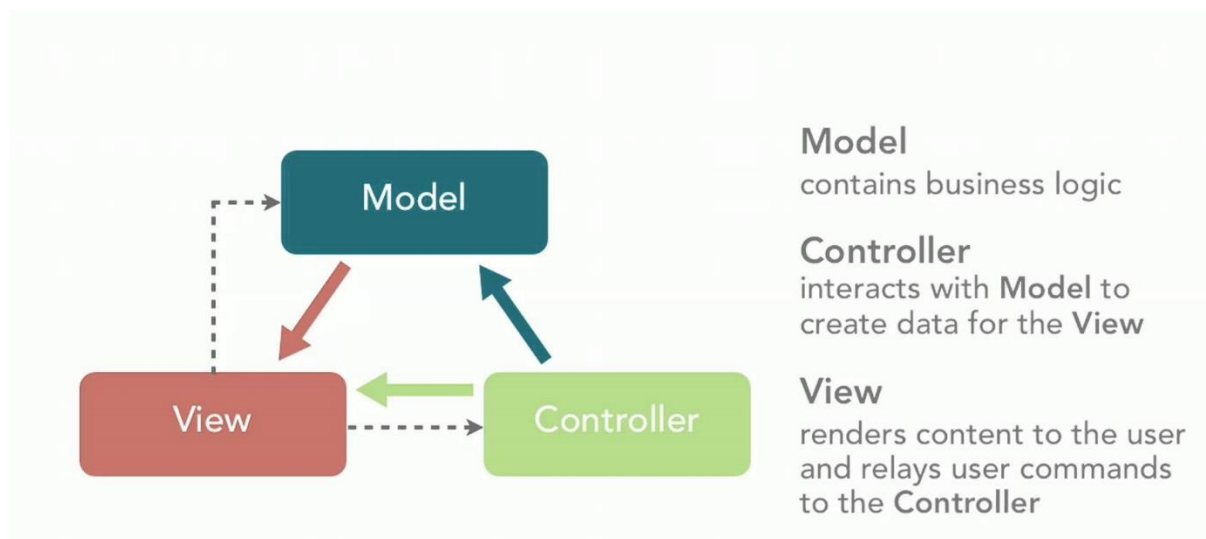


Figure 20 - MVC architecture (Firebirdsql.org, 2019)

### Raspberry Pi Development:

The requirements specified that this system needed to be deployable to an embedded device, as I had selected the Raspberry Pi to act as the embedded device of the system it meant that I had to develop for this. Initially, development started on the Raspberry Pi, so I had installed all of the necessary libraries for the task and ran a couple of tests (face detection scripts) to verify that the Raspberry Pi was powerful enough for the task. However, at this stage of development, I didn't have access to an external monitor which meant that any development on the Raspberry Pi was carried out through the remote command line via SSH. I also used VNC, which is a Virtual Desktop tool that allowed me to view the desktop of the Raspberry Pi through the network. Unfortunately, this meant that the output displayed on the desktop of the Raspberry Pi was restricted by the Network capabilities, which were not fast enough to display video playback in real time. So, any tests that I had ran with regards to face or emotion detection would run with huge delays. This made it impossible to continue developing natively on the Raspberry Pi.

Because of the issues outlined above, I transferred all of my code to my MacBook pro and continued developing on that for the remainder of the time. When transferring code from Raspberry Pi to the MacBook Pro, there were no configuration errors. This led to the assumption that anything developed on the MacBook Pro could then be re-transferred to the Raspberry Pi and the performance of the system could be evaluated when a monitor was available to connect to it.

## Chapter 6

### Verification and Validation:

This section of the report gives details about how the system was verified and tested against the defined requirements of the application. As an iterative development methodology was followed, unit tests were carried out alongside development, but what follows is the final acceptance testing. The system was developed as two separate builds, the emotion detection module and the web application module. This meant that the first tests carried out had to ensure that the emotion detection module met the specification. Once the required functionality of the emotion detection module was developed, a Flask web-application was built that could stream the emotion detection module to ensure integration between the developed module and a Flask web-application was achievable. This was also done for the face detection sub-module of the emotion detection module. When this was successful, the entire web-application could be developed and then both integration and unit tests could be carried out on the complete system.

### Test Plan Overview:

There were two types of tests carried out on the system, Unit Testing and integration testing. First Unit testing was carried out and then integration testing. This way I could ensure that individual components of the system (Units) functioned as necessary before incorporating them into the complete system. Once all components were integrated into the complete system, integration testing could be carried out to ensure that components interact successfully.

### Unit Testing:

Unit testing was carried out on each individual module that the system was comprised of. Although Python has an automated Unit testing framework (Unittest), this was not used to test modules because it would not have been possible to test the actual output of some of the modules of the system against the expected output. For example, it would not have been possible to write an automated test to test that when a face is detected by the face detection module that it is indeed a face. It would not be possible to test that when an emotion is detected by the emotion detection module that the emotion detected is the correct emotion. This meant that Unit testing was carried out manually for this system. Unit testing was predominantly carried out on the Emotion detection module.

### Integration Testing:

Integration testing was carried out once Unit testing had finished, as the modules of the system needed to be verified before attempting to integrate them into a complete system. Integration testing was predominantly carried out on the web-application, as this is where all the sub-modules of the system interacted between themselves.

### Emotion Detection Tests:

Before tests were carried out on the emotion detection module, the requirements were re-visited, so that I could be certain that the module functioned as the expected by the specification. The requirements related to the emotion detection module are:

- Recognise emotions in an input video stream in real time
- Recognise emotions in a pre-recorded video stream
- Be capable of recognising emotions of multiple people at once
- Display the emotions of users in a video stream (either real time or pre-recorded) as it is being played back

However because of the architecture of the emotion detection module, ([see Figure 14](#)) the first section of the emotion detection module is a face detector, so this also needed to be tested to make sure that it would function as required for the emotion detection module. The requirements of the face detector are:

- Detect a face in an input video stream in real time
- Detect a face in a pre-recorded video stream
- Be capable of detected multiple faces at once

The face detector had to also be able to extract the face that was detected so that it could be sent to the emotion detection algorithm. This meant that the face detector had to be able to crop a frame containing a face to the dimensions of the face that had been used to train the emotion detection algorithm. This formed the first integration test of the emotion detector. The face detector and extractor had to integrate with the emotion detector successfully before the emotion detector could run. Both the face detector and the emotion detector needed to be able to be streamed to a web app, to make sure that this could be incorporated into the complete system. Streaming the face detector and emotion detector to a web app were the other two integration tests at this time.

The complete list of test cases relevant to the emotion detection module follows:

Module	Test Title:	Description	Pass/ Fail	Test type
FD	Detect Face Live	Detect a face in a live video input	Pass	UT
FD	Detect Face Pre-recorded	Detect a face in a pre-recorded video input	Pass	UT
FD	Detect Multiple Faces Live	Detect multiple faces in a live video input	Pass	UT
FD	Detect Multiple Faces Pre-recorded	Detect multiple faces in a pre-recorded video input	Pass	UT
FD	Export Cropped Face	Export a cropped image containing only a face	Pass	UT
FD	Detect Face – Lighting	Detect a face with variance in lighting	Pass	UT
FD	Detect Face – Occlusion	Detect a face when it is occluded (wearing glasses or a hat or both)	Pass	UT
FD	Detect Face – Pose	Detect a face with varied poses (45-degree angles)	Pass	UT
FD	Stream Face Detector	Stream the output of the Face Detector to a web-application page	Pass	IT
FD&ED	Detect Face & Emotion	Detect a face, crop it to the emotion detector size and detect the emotion in the face	Pass	IT
ED	Detect Emotion Live	Detect the emotion expressed in a face in a live video input	Pass	UT
ED	Detect Emotion Pre-recorded	Detect the emotion expressed in a face in a pre-recorded video input	Pass	UT
ED	Detect Emotion Multiple Faces Live	Detect the emotion expressed in multiple faces in a live video input	Pass	UT
ED	Detect Emotion Multiple Faces Pre-recorded	Detect the emotion expressed by multiple faces in a pre-recorded video input	Pass	UT
ED	Detect Emotion – Lighting	Detect the emotion expressed in a face with variance in lighting	Pass	UT

ED	Detect Emotion – Occlusion	Detect the emotion expressed in a face when the face is occluded (wearing glasses or a hat or both)	Pass	UT
ED	Detect Emotion – Pose	Detect the emotion expressed in a face with varied poses (45-degree angles)	Pass	UT
ED	Stream Emotion Detector	Stream the output of the Emotion Detector to a web-application page	Pass	IT

*Table 1 - Emotion detection acceptance tests*

FD = Face Detector, ED = Emotion Detector, FD&ED = Face Detector and Emotion Detector, UT = Unit Test, IT = Integration Test

Details of tests:

Additional information regarding the success criteria of the Validation tests of the Emotion Detection module follows.

*Face Detection:*

For face detection, all tests were carried out on myself or images from within the training dataset. For detection of multiple faces in a live video input I held a picture of a face in the frame, so that the detection would have to be carried out on my face and the second face in the video frame. The success criteria for detection was a detection rate above 70% and was evaluated using timers. Evaluation of exporting a face was carried out manually by comparing an input image to an output image and was carried out both on images and video clips. Evaluating the integration of the face detection module with a Flask web-application was done by manually examining that the output of the face detector was streamed to the page and displayed in real time.

*Emotion Detection:*

For emotion detection, again all tests were carried out on myself or images taken from the training dataset. Multiple faces were evaluated using myself holding a picture of a face expressing an emotion. The success criteria were a correct detection rate above 50%, this is because when training the algorithm, the predicted success rate was around 70% meaning that the expected real-life success rate would have to be lower than 70%. Integration between the face and emotion detection modules was evaluated by checking that the emotion detector would detect an emotion (regardless of whether the detected emotion is correct) for every face detected in an image. Evaluating the integration of the emotion detection module with a Flask web-application was carried out manually by examining that

the output of the emotion detection module was successfully streamed to the page and displayed in real time.

Once the tests mentioned above had been completed and all test cases passed successfully, I was left with a fully functioning Emotion Detection Module that could be streamed to a web application. This meant that the rest of the system could be developed, and the emotion detector could be incorporated into it. In the design section of the report, it is explained that the emotion detector module had been transformed into an emotion detection class, so by importing the emotion detection class to the complete system all I needed to do was create an emotion detection object and call its methods to start or stop the emotion detection in the web application.

### Web application Tests:

In the same way that the test strategy for the emotion detector was developed, so was the test strategy for the Web-Application. This meant that the requirements were re-visited before forming a series of test cases. The requirements related to the Web-application not including the emotion detector are:

- Display the recognised emotions in a meaningful manner
- Record a video stream
- Store the recorded video stream
- Display the emotions of users in a video stream (either real time or pre-recorded) as it is being played back
- Store an edited video stream that displays emotions of users as the video is being played back
- Display statistics of the emotions a user in the video shows for the duration of the video
- Have a User-Friendly Interface

At this time, because additional modules had to be developed to meet the requirements of the system, each additional module (such as the module that saved the recorded video stream to file) had to be unit tested, then integration tested when added to the complete system.

The complete list of test cases relevant to the web-application follows:

Module	Test Title:	Description	Test type
UI	Display Page	Display the front-end page for each page of the corresponding functionality of the system	UT

UI	Navigation	Navigate between different pages of the front end	IT
SV	Save to file	Save a video clip to file. (In this case, in the directory of the library)	UT
SV	Save to file – Web	Save a video to file using the web-application	IT
LIB	Save to Library	Use the Save Video module to store the video saved in the directory of the library	IT
LIB	Display videos – Web	Display the videos currently stored in the library directory in the web-application	UT/IT
PV	Play video from file	Play a video saved in the library directory	UT/IT
PV	Play video – Web	Play a video saved in the library directory and stream it to the web application	IT
ED	Detect Emotions – Web	Detect emotions in a video stream and display the results in the web-application	IT
STAT	Record Statistics	Record statistics related to the emotions detected (Duration that each emotion is detected)	UT
STAT	Display Statistics	Display the recorded statistics in the Web-Application	IT

*Table 2 - Web-Application acceptance tests*

UI = User Interface, SV = Save Video, LIB = Library, PV = Play Video, ED= Emotion Detector, STAT = Statistics, UT = Unit Test, IT = Integration Test

#### Details of tests:

All modules of the system undertook integration testing, where a manual test was carried out to ensure that all individual modules of the system integrated with one another. As seen Table 2 above, each individual module undertook Unit tests too. Explanations of the validation tests follow.

#### *User Interface:*

The User Interface's Unit test evaluated the functionality of the User Interface. The User Interface had to display the contents of each page to the User as well as the outputs of each individual functional unit. To verify this, I navigated between each page and examined the contents of the page

The User Interface's Integration test evaluated the interaction between the User Interface and the corresponding functionality of the system. For example, clicking on a video from the Library page was expected to navigate to a page that displayed the video being played.

#### *Save Video:*

The Save Video modules Unit test evaluated the functionality of the Save Video module. This had to record a video and save the video clip to the library directory. To evaluate this, I recorded a video and checked that the new video was in the library directory, and was the video that I had just recorded.

The Integration test carried out on the Save Video module evaluated that the module integrated with the system. To evaluate this, the same Unit test as above was carried out, but this time in the Web-Application.

#### *Library:*

The Library modules Unit test was developed to evaluate its functionality. This had two functions, to display the videos currently stored in the library. To evaluate this, I recorded a video clip and then navigated to the library page to see if it was displayed. This was done three times, and if all three videos were displayed in the directory then test was successful.

The Integration tests carried out on the Library module were intended to verify that the functionality tested above was reflected in the web-application. So, the same test as above was carried out in the Web-Application, and if the results were as expected (Display the videos recorded using the Save Video module in the Library page) then the module functioned as expected.

#### *Play Video:*

The Play Video module was evaluated by playing a video from the library directory first in a separate window that the module would open, and then in the web-application. If the selected video was played (In a window and in the Web-Application) then this test was successful.

#### *Emotion Detector:*

The Emotion Detection module had already been Unit Tested, so only an integration test was carried out that would ensure that the emotion detection module could interact with the save video module and the user interface. To evaluate this, when opening the Web-Application, first the emotion detection page was visited. This would initiate emotion detection and display the resulting video on the page. Then, the record video module was used to save a video clip of the emotion detection that was running in real time. Finally, the library page was visited to ensure that the video clip that had been recorded was the emotion detection that had just commenced.

#### *Statistics:*

The Statistics Module recorded the durations that different emotions were detected, and used these values to display the duration an emotion was detected in the User Interface. The Unit test that was carried out on the Statistics Module was created using timers. The duration that an emotion was



expressed was timed and then the results of the statistic module were compared to the recorded values. If they were the same then the functionality was implemented successfully. The integration test tested that the results recorded by the module were displayed in the Web-Application. This was done by first visiting the Emotion Detection page, then recording a video clip from that page and finally, navigating to the statistics page to verify that the results were displayed on the Statistics page.

Upon completion of the test cases stated above, I had a fully functioning prototype of the application that met the requirements that were defined previously. All the user stories and requirements were met, and the different modules within the application integrated successfully. All submodules were developed separately and tested as individual scripts before being transformed into classes that could be imported and ran by the web-application. Using Flask allowed me to create the different objects corresponding to the classes, and then display the output of these classes in the web application.

### Raspberry Pi Tests:

The final step in the verification and validation process was to test the application on the Raspberry Pi. In the design document, it is explained that development did not happen natively on the Raspberry Pi but instead on PC, with the assumption that because the Raspberry Pi had been set up the same way that the PC had with regards to Python Versions, OpenCV versions and Flask version, the application should be compatible with the Raspberry Pi. The only change that would have been needed to be made was the input source, but as OpenCV's default video input source is the camera module of the device that it is running on, this wouldn't be an issue. On PC the default input source is the web-camera, and on the Raspberry Pi the default input source is the camera module of the Raspberry Pi (assuming that a camera module has been successfully installed). So, the only thing left to do was to run the application on the Raspberry Pi natively and see if the Raspberry Pi had enough processing power to successfully run the application in real time. If this was not the case, then the system would have to be re-designed so that video capture and face detection could be carried out on the Raspberry Pi, with the extracted face being sent to a server for processing.

To determine that the application ran successfully on the Raspberry Pi, a few test cases were developed that would test all of the functionality of the application. At this time, the test cases are in the form of User Tests, but with the difference that the tests were run by the developer and not an external user. The steps followed to test that the application could run on the Raspberry Pi are:

1. Switch between different pages in the application – This tested that navigation functioned as required

2. Navigate to the emotion detection page and examine the output of real time emotion detection – This tested that emotion detection could run in real time and be streamed to the web-application
3. Click the “record” button, detect emotions then press the “stop recording” button in the emotion detection page – This tested that emotion detection could be recorded and stored to the library.
4. Navigate to the library page and examine the files currently in library – This ensured that the previous test was successful.
5. Click on the video currently in the library to watch the video in the browser – This tested that the newly recorded video clip could be played and displayed in the web-application.
6. Navigate to the statistics page and examine the statistics – This tested that the statistics module would run successfully.

When following the test steps mentioned above, there were mixed results on the Raspberry Pi. The User Interface was displayed as it was on PC and navigation between pages was successful. When navigating to the emotion detection page, where emotion detection in real time was displayed the system could display the video file being processed, with a rectangle around the face that was detected, and the emotion detected was also displayed. But the framerate at which this was happening seemed to be reduced (Around 15FPS) compared to that of the system running on a PC. Videos could be recorded using the “record” and “stop recording button”. The library page then displayed the videos that were recorded, but when the video that had just been recorded was displayed, it was played with a normal framerate (Around 30FPS), but appeared to be sped up because of the reduced video frames outputted from the emotion detection module. A video recorded at 15 FPS, when played at 30FPS has half the duration of the original video clip. Then when navigating to the Statistics Page, it correctly displayed the statistics as they were displayed on PC.

## Chapter 7

### Results and Evaluation:

In this section of the report, the final application is evaluated with regards to the original specification and requirements. To begin with, the final application is evaluated by first inspecting the user interface and then the functionality of the system. Next, the Emotion detection module is evaluated separately. This is the only submodule individually evaluated because the entire system relied on this being successfully developed. Finally, the original specification is examined again, and evaluation is carried out to ensure that all the requirements were met upon completion of the project.

#### Final Application:

Upon completion of this project, a fully functioning prototype of the web-application was developed. The focus of this project was the emotion detection, therefore the User Interface that was implemented was basic but functional. The appearance of it was not of concern at this time, which is why testing was revolved around functionality and integration rather than end Users. The aim was to develop a user interface that could be used to navigate between different pages of the application and display the outputs of the different modules within the system in order to demonstrate that the system functioned as expected.

#### User Interface/Front end:

In order to evaluate the User interface, three aspects of it were reviewed. Its ability to display the results of the back-end functionality, its ability to navigate between pages and its ability to initialise back end functionality from within the user interface.

#### Display Results:

In order to evaluate the User interfaces ability to display results, each page that would be used to display, or render anything from the back end had to be evaluated. The evaluation procedure for each page follows:

- Emotion Detection Page: First the emotion detection page was visited as this would display the results of emotion detection in real time upon navigating to this page. This successfully displayed the input from the web-camera or camera module, with the emotion detection being carried out in real time. See Figure 21 below:

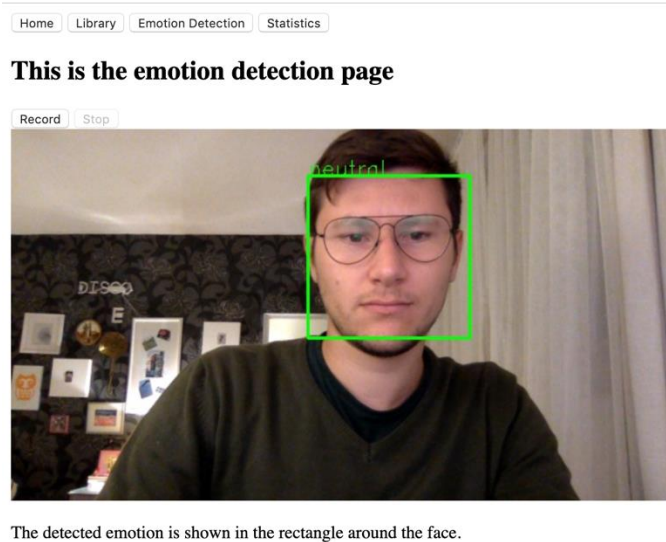


Figure 21 - Output of emotion detection

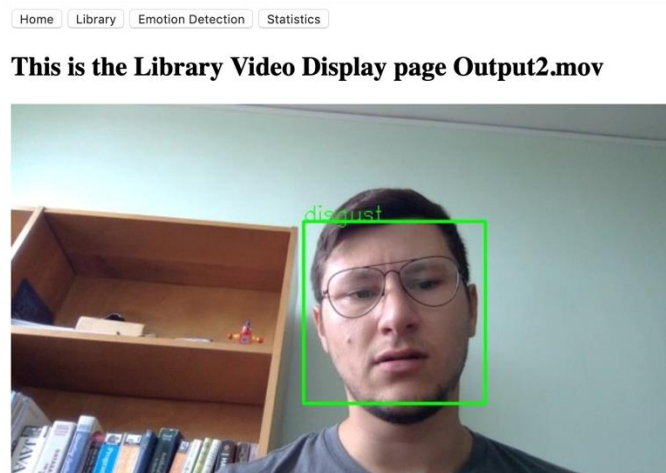
- Library Page: Next, the library page was visited. The library already contained videos that had been previously recorded, so upon navigating to it the buttons corresponding to each file in the library had been created. To evaluate this, the library directories contents were compared against those displayed. Every file in the library appeared as a button on the library page. See Figure 22 below:



Figure 22 - Output of Library page

- Play Video from Library: When visiting the library page, buttons corresponding to each video on the file should have been created. When each of these buttons are clicked, the Web-Application then navigates to the Video Display page of the file corresponding to the button.

On this page, the video that has been selected is played. To verify that this functionality was implemented successfully, I populated the library with four video clips. Then I checked all files could be played in the web-application. This was a form of boundary testing, where the files at the beginning end and middle of the directory were tested. See Figure 23 below



*Figure 23 - Output of Video display page*

- Statistics Page: Finally, the statistics page had to be visited to ensure that the statistics recorded could be displayed. Upon opening this page one of two things happened: If a video had been recorded in the current session then the statistics associated with it were displayed. If a video had not been recorded in the current session, then the statistics page would appear to be empty. To verify that the statistics being displayed were the same as those being recorded, I printed the statistics that were recorded in the terminal that the application was running in and evaluated based on the printed output. See Figure 24 for successful output and Figure 25 for unsuccessful output.

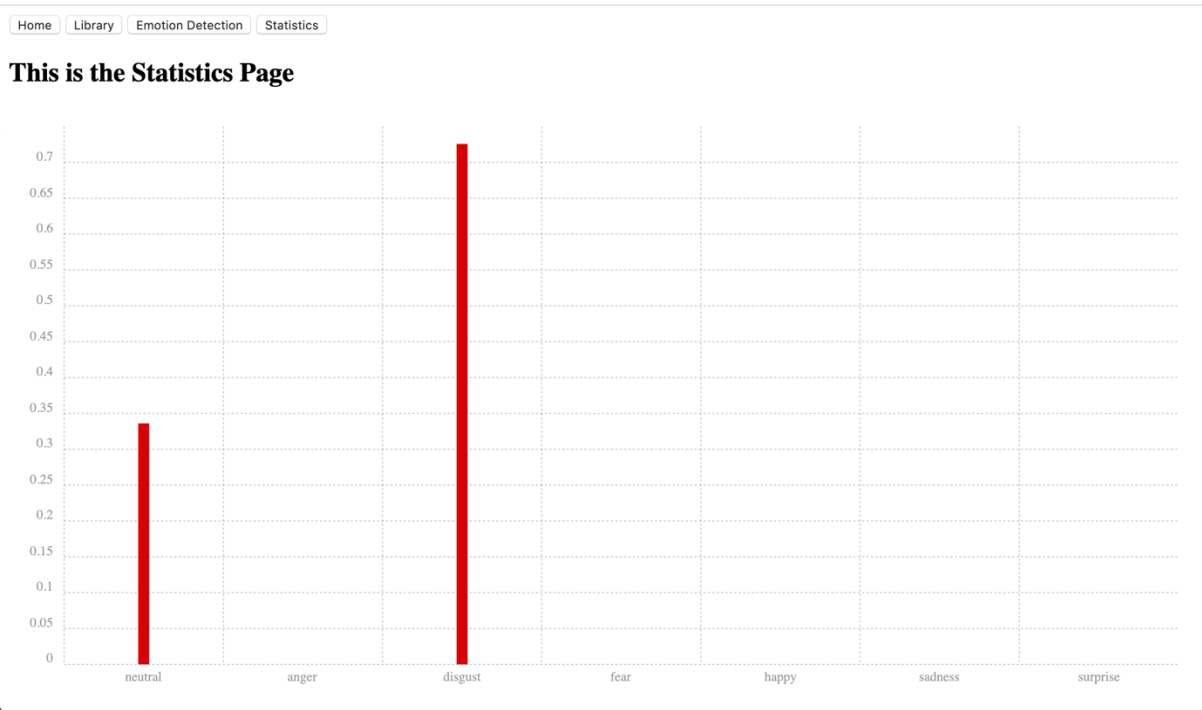


Figure 24 - Successful Output of Statistics page



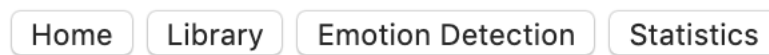
Figure 25 - Unsuccessful Output of Statistics page

Evaluating the applications ability to display results was successful, except from the statistics page that would function as required if a video had been recorded in the current session whilst if no video had been recorded then it would not.

#### Navigation:

In order to evaluate the applications navigational abilities, the navigation buttons of each page were tested. The expected outcome of navigation was the same for every page because the buttons were

inherited from the base.html file. This meant that if the navigation from the homepage did not work, it would not work in any of the other pages either, but if it did work then it was expected to work for all of the other pages too. Figure 26 below shows the navigation section. To test the navigation functioned as expected, each button was clicked from each page which resulted in navigation between pages as required. This was unnecessary because all pages inherited from the base.html file, but I wanted to make sure that the inheritance was successful as I had not come across inheritance in HTML before.



*Figure 26 - Navigation section*

#### *Recording a video clip:*

Evaluating the user interface on its ability to control back end functionality from the front end meant that the buttons used to start and stop recording had to be evaluated. These were the only buttons with functionality other than navigation associated with them. All other actions start with navigation to the page responsible for that action. To evaluate the outcome of these buttons, once the Emotion Detection page was visited, the buttons were clicked and the results of clicking them were recorded. When the Emotion Detection page is loaded, the “Stop” button is unavailable as seen in Figure 27 below. Once the “Record” button is pressed, the “Record” button becomes unavailable and the “Stop” button becomes available as seen in Figure 28 below. This meant that it was not possible to start recording whilst recording was currently running or to stop recording if no recording was currently running. So, the only thing left to evaluate was if by pressing the record button and then the stop recording button a video clip was actually recorded. To do this, the application was navigated to the library page after a video had been recorded, and as expected the library page displayed a new recording. Once this new recording was played back it was confirmed that it was indeed the video clip recorded by using the buttons. For my tests, I recorded different lengths of video clips, spanning from 10seconds to 10 minutes. The system was designed without any limitation of length of video clips, but with the intention of video recordings having a maximum duration of 10 minutes. This is because the system is intended to be used as a replacement for interviews and questionnaires that typically take this long to conduct.



*Figure 27 - Unavailable Stop button*



Figure 28 - Unavailable Record button

### Functionality/Back end:

The next section of evaluation with regards to the final application was to evaluate the back-end functionality. The functionality implemented by the back end is:

- Emotion Detection in real time
- Record emotion detection from input camera
- Display a list of recorded video clips
- Play any video clip from the library
- Display the statistics of the emotion detection

When evaluating the User Interface, the back-end functionality was also evaluated as the User Interface relied on this to function. The functionality of the internal components was evaluated, but not their performance. This meant that the performance of the Emotion Detection and Face Detection had to be evaluated.

### Emotion Detection:

The next step in evaluating the developed system was to evaluate the Emotion Detection module. The methodology adapted to evaluate this was to first evaluate it on its ability to detect a face in a video stream. Next, if a face is detected to evaluate whether an emotion is detected for every face that is detected and finally to evaluate if the detected emotion is correct.

### Face Detection:

The Face Detection module was evaluated during development, as it had 100% success rate in detecting and extracting images from the training data set, this did not have to be re-evaluated on images. When detecting a face in a video stream I found that it met the requirements in ideal conditions (lighting, pose, etc.) but in non-ideal conditions it had a success rate of around 70%, which was acceptable for the application.

### Emotion Detection:

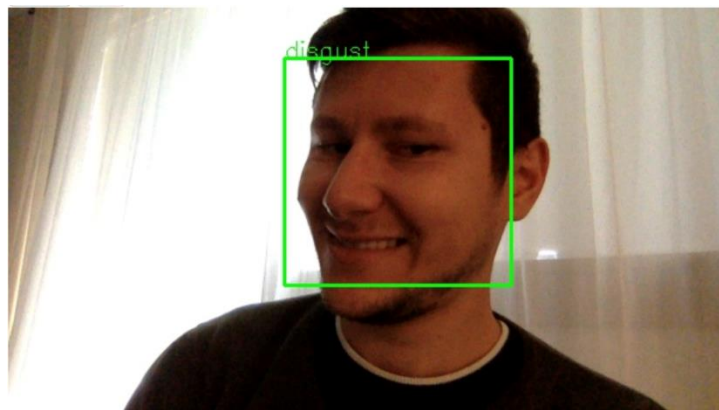
Next, I had to evaluate if the Emotion Detection module actually detects an emotion for every face detected in the video stream. In order to do this, I displayed the emotion that was detected in the rectangle surrounding the face that was detected. I found that every time a face was detected an emotion associated with it was also detected.



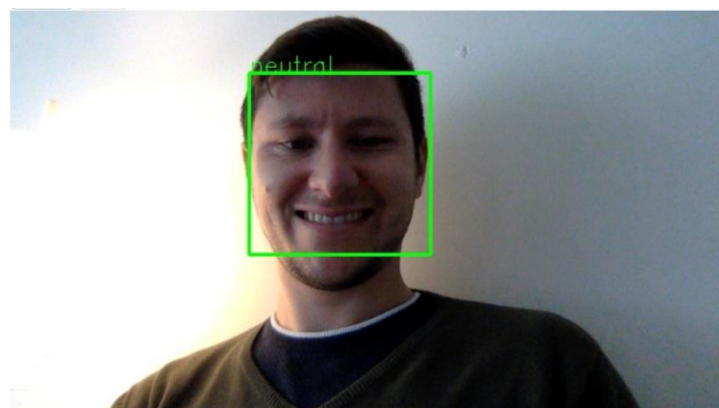
### Emotion Detection Performance:

To evaluate the success rate of the emotion detection module, different evaluations were carried out. First, I would express an emotion towards the camera module. This was then recorded and an attempt to detect the emotion expressed was carried out. The initial test was undertaken in perfect conditions (perfect lighting and face looking directly at the camera). The outcome of the emotion detection was then evaluated. I found that the emotion detector was successful most of the time when testing under ideal conditions.

Next, I carried out the same tests as above, but with varied lighting and facial pose. I found that the emotion detector did not perform well under non-ideal conditions. Figures 29 and 30 below show examples of incorrectly detected emotions due to variance in illumination and pose.



*Figure 29 - Wrongfully identified emotion due to variance in pose*



*Figure 30 - Wrongfully identified emotion due to variance in illumination*

Finally, I carried out the same tests again, but this time wearing glasses. I found that at this time, under ideal conditions the emotion detector would work almost as well as the first tests undertaken. I also found that when there was a glare on my glasses due to the reflection of my monitor the face detector couldn't detect my face correctly as shown in Figure 31 below.

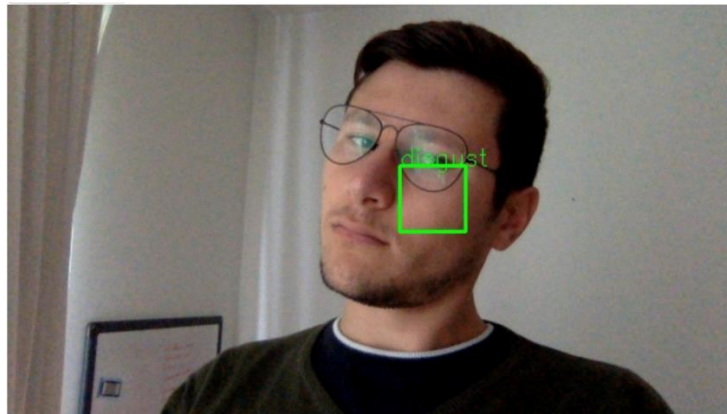


Figure 31 - Wrongfully identified emotion due to monitor reflection on glasses

### Comparison with Specification:

The final section of the evaluation is to compare the developed system with the original specification and requirements. The list of requirements is re-visited below, alongside the outcome of development.

Requirement	Result
Recognise emotions in an input video stream in real time	Successfully Implemented
Recognise emotions in a pre-recorded video stream	Successfully Implemented
Display the recognised emotions in a meaningful manner	Successfully Implemented
Make use of an embedded device (Portability)	Successfully Implemented
Record a video stream	Successfully Implemented
Store the recorded video stream (Create a Library)	Successfully Implemented
Display the emotions of users in a video stream as the video is being displayed. (either real time or pre-recorded video clip)	Successfully Implemented
Save an edited copy of the input video stream that displays the emotion detected.	Successfully Implemented

Display statistics of the emotions a user in the video shows for the duration of the video. (e.g., happy 56% of the time they were in the video)	Partially implemented. The output of the statistics module was only displayed if a video was recorded in the current session.
Be capable of recognising emotions of multiple people at once	Successfully Implemented
Have a User-friendly Interface	Un-Evaluated. No User tests were carried out because the focus of this project was emotion detection.

*Table 3 - Evaluation of requirements*

From the originally specified requirements, all requirements were met except from the ones that were deemed as out of the scope of the project at this time. The statistics module was partially implemented, and how this could be fully implemented is discussed further down in this report ([See Chapter 8](#)) The User Interface had still to be evaluated because User Tests were not carried out at the current time, but in the future a user centred approach would be followed when designing the Interface.

## Chapter 8

### Summary and Conclusions:

In this chapter of the report, a reflection on the project as a whole is given, then reflection on the emotion detection module. Future work is discussed including ideas regarding improving the current system and finally, a conclusion.

#### Summary/Reflection:

**Project as a whole:** This project was developed with great emphasis given to the requirements, as these outlined the features and functionality required by the project. Overall, the requirements that were inside the scope for the project were successfully satisfied. A web-application that can carry out emotion recognition in real time but can also record and save a video clip of the emotion detection was developed. This included a library page that contained all the recorded video clips recorded by the emotion recognition module. It also had playback functionality for files within the library. And finally, a chart that displays the statistics recorded by the emotion detection module. The solution was proven to be deployable to the Raspberry Pi too, even though the performance of real time emotion detection was impacted by the limitations of the Raspberry Pi's processing power. In theory by upgrading to the newest model of the Raspberry Pi (Currently Raspberry Pi 4 Model B), the increase in processing power should allow for the real time emotion recognition system to run at an acceptable framerate.

**Emotion Detection module:** The emotion detection module was highly accurate in ideal conditions despite being lightweight. This was essentially a face recognition algorithm trained to recognise emotions rather than faces. The methodology for detecting faces once the model had been trained was relatively straightforward. If a face was detected in a frame of a video clip, it was extracted, resized and passed through the (re-trained) emotion detection model, and the emotion expressed in the face was matched to a known emotion of the model. When a face was detected under uneven lighting, or with a pose slightly off-centred from the camera, the emotion detection module struggled to predict the correct emotion expressed by the face, but this could be overcome with more training of the model.

Overall, this was a successful project that completed all the predefined goals and objectives and showed that although emotion detection and recognition is a complex task, there are relatively lightweight solutions that can be developed and deployed to portable computers. The system that

was developed is robust enough to be used as a proof of concept but would need some further work to be carried out before it could be released for commercial use.

#### Future Work:

In this section an insight into the next steps of the project, along with ideas to improve its performance are discussed. To begin with, the requirements that were out of the scope of the current project would have to be implemented. The highest priority requirement, which would need to be completed next is the statistics module. As it stands it is implemented but would require some form of database that could store the data regarding the statistics. Next, the user accounts module would need to be created. As a database would have already been implemented, it would be easy to add a table that could hold the user account credentials. This would then have to be encrypted to keep the details of user accounts private.

#### Web Application:

The current version of the web-application is good enough to prove the concept but needs to be made visually appealing and user friendly. So, this would be the next step in development of the web-application. As far as the functionality of the web-application is concerned, everything works nicely. The main scope for improvement and development that exists for the web-application revolves around the appearance of the application. User testing would need to be conducted in order to determine potential future users' requirements with regards to design.

#### Emotion Detection:

The next steps regarding the emotion detection module are based on improving its performance. The functionality of the module meets the requirements of the specification, and no additional features need to be added. As far as improving the performance of the module, the first step is to increase the training data sets. More images for each emotion need to be added to the set, including images of each emotion in different poses towards the camera and under different illumination conditions. This should in theory make the current system more robust. Another approach is to re-develop the system, either by training a deep learning model to detect emotions or exploring different available algorithms that could be fit for the purpose of this project. However, by increasing the complexity of the algorithm being used, an increase in processing power would be required. So, in a sense there is a trade-off between algorithm robustness and necessary processing power. Different implementations of the emotion recognition system need to be tested in order to determine how well they perform when deployed to a portable computer.

## Conclusion:

This was a successful project, where all the requirements for the current time were met and there is now proof that the concept is viable. The emotion recognition module works well under ideal conditions, and the whole system was deployable to the Raspberry Pi. This shows that the solution is lightweight enough to be deployed to portable computers and embedded devices. There are improvements that could be made with some suggestions documented in the report.

## Appendix A

### References:

- Arva, G. and Fryza, T. (2017). Embedded video processing on Raspberry Pi. *2017 27th International Conference Radioelektronika (RADIOELEKTRONIKA)*. [online] Available at: <https://ieeexplore.ieee.org/document/7937598> [Accessed 3 Jun. 2019].
- Bartlett, M., Littlewort, G., Fasel, I. and Movellan, J. (2003). Real Time Face Detection and Facial Expression Recognition: Development and Applications to Human Computer Interaction. *2003 Conference on Computer Vision and Pattern Recognition Workshop*. [online] Available at: <https://ieeexplore.ieee.org/document/4624313> [Accessed 9 Jun. 2019].
- Beltrán Prieto, L. and Komínková-Oplatková, Z. (2017). A performance comparison of two emotion-recognition implementations using OpenCV and Cognitive Services API. *MATEC Web of Conferences*, [online] 125, p.02067. Available at: [https://www.matec-conferences.org/articles/mateconf/abs/2017/39/mateconf\\_csc2017\\_02067/mateconf\\_csc2017\\_02067.html](https://www.matec-conferences.org/articles/mateconf/abs/2017/39/mateconf_csc2017_02067/mateconf_csc2017_02067.html) [Accessed 8 Jun. 2019].
- Bhanse, V. and Jaybhaye, M. (2018). Face Detection and Tracking Using Image Processing on Raspberry Pi. *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. [online] Available at: <https://ieeexplore.ieee.org/document/8597246> [Accessed 4 Jun. 2019].
- Dang, K. and Sharma, S. (2017). Review and comparison of face detection algorithms. *2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*. [online] Available at: <https://ieeexplore.ieee.org/document/7943228> [Accessed 6 Jun. 2019].
- Daryanavard, H. and Harifi, A. (2018). Implementing Face Detection System on UAV Using Raspberry Pi Platform. *Electrical Engineering (ICEE), Iranian Conference on*. [online] Available at: <https://ieeexplore.ieee.org/document/8472476> [Accessed 3 Jun. 2019].
- Farahani, F., Sheikhan, M. and Farrokhi, A. (2014). Facial Emotion Recognition Using Gravitational Search Algorithm for Colored Images. *Artificial Intelligence and Signal Processing*, pp.32-40.
- Firebirdsql.org. (2019). The ASP.NET MVC Platform. [online] Available at: [https://firebirdsql.org/file/documentation/reference\\_manuals/fbdevgd-en/html/fbdevgd30-dot-net-mvc.html](https://firebirdsql.org/file/documentation/reference_manuals/fbdevgd-en/html/fbdevgd30-dot-net-mvc.html) [Accessed 9 Aug. 2019].
- Ghandi, B., Nagarajan, R. and Desa, H. (2010). Real-time system for facial emotion detection using GPSO algorithm. *2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA)*. [online] Available at: <https://ieeexplore.ieee.org/document/5679500> [Accessed 7 Jun. 2019].
- Gupta, I., Patil, V., Kadam, C. and Dumbre, S. (2016). Face detection and recognition using Raspberry Pi. *2016 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*. [online] Available at: <https://ieeexplore.ieee.org/document/8009092> [Accessed 4 Jun. 2019].
- Laboreiro, V., Maia, J. and de Araujo, T. (2012). Face Segmentation Using Projection Pursuit for Texture Classification. *Intelligent Data Engineering and Automated Learning - IDEAL 2012*, [online] pp.237-244. Available at: [https://link.springer.com/chapter/10.1007/978-3-642-32639-4\\_29#Bib1](https://link.springer.com/chapter/10.1007/978-3-642-32639-4_29#Bib1) [Accessed 27 May 2019].

- Laytner, P., Ling, C. and Xiao, Q. (2014). Robust face detection from still images. *2014 IEEE Symposium on Computational Intelligence in Biometrics and Identity Management (CIBIM)*. [online] Available at: <https://ieeexplore.ieee.org/document/7015446> [Accessed 28 May 2019].
- Lee, H., Chen, R. and Wei, D. (2018). Building Emotion Recognition Control System Using Raspberry Pi. *Lecture Notes in Electrical Engineering*, [online] pp.36-45. Available at: [https://link.springer.com/chapter/10.1007%2F978-981-10-7398-4\\_4](https://link.springer.com/chapter/10.1007%2F978-981-10-7398-4_4) [Accessed 9 Jun. 2019].
- Li, M., Yu, C., Nian, F. and Li, X. (2015). A Face Detection Algorithm Based on Deep Learning. *International Journal of Hybrid Information Technology*, 8(11), pp.285-296.
- Magalhaes, J., Ren, T. and Cavalcanti, G. (2012). Face Detection under Illumination Variance Using Combined AdaBoost and Gradientfaces. *Intelligent Data Engineering and Automated Learning - IDEAL 2012*, [online] pp.435-442. Available at: [https://link.springer.com/chapter/10.1007%2F978-3-642-32639-4\\_53](https://link.springer.com/chapter/10.1007%2F978-3-642-32639-4_53) [Accessed 2 Jun. 2019].
- Onix-systems.com. (2019). What Do You Need to Know About the Limits of Machine Learning?. [online] Available at: <https://onix-systems.com/blog/what-do-you-need-to-know-about-the-limits-of-machine-learning> [Accessed 7 Aug. 2019].
- Sajjad, M., Nasir, M., Ullah, F., Muhammad, K., Sangaiah, A. and Baik, S. (2019). Raspberry Pi assisted facial expression recognition framework for smart security in law-enforcement services. *Information Sciences*, [online] 479, pp.416-431. Available at: <https://www.sciencedirect.com/science/article/pii/S0020025518305425?via%3Dihub> [Accessed 10 Jun. 2019].
- Shakhnarovich, G., Viola, P. and Moghaddam, B. (n.d.). A unified learning framework for real time face detection and classification. *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*. [online] Available at: <https://ieeexplore.ieee.org/document/1004124> [Accessed 5 Jun. 2019].
- Sharifara, A., Mohd Rahim, M. and Anisi, Y. (2014). A general review of human face detection including a study of neural networks and Haar feature-based cascade classifier in face detection. *2014 International Symposium on Biometrics and Security Technologies (ISBAST)*. [online] Available at: <https://ieeexplore.ieee.org/document/7013097> [Accessed 1 Jun. 2019].
- Singla, R. and Bansal, S. (2011). A NEW APPROACH FOR MOOD DETECTION VIA USING PRINCIPAL COMPONENT ANALYSIS AND FISHERFACE ALGORITHM. *Journal of Global Research in Computer Science*, [online] 2(7), pp.88-92. Available at: <https://www.semanticscholar.org/paper/A-NEW-APPROACH-FOR-MOOD-DETECTION-VIA-USING-AND-Singla-Bansal/1812d9e1db904ca7ca9652938ae5d32177e269c3> [Accessed 7 Jun. 2019].
- Suchitra, Suja P. and Tripathi, S. (2016). Real-time emotion recognition from facial images using Raspberry Pi II. *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*. [online] Available at: <https://ieeexplore.ieee.org/document/7566780> [Accessed 10 Jun. 2019].
- Thuseethan, S. and Kuhanesan, S. (2014). Eigenface Based Recognition of Emotion Variant Faces. *SSRN Electronic Journal*, [online] 5(7), pp.31-37. Available at: <https://www.semanticscholar.org/paper/Eigenface-Based-Recognition-of-Emotion-Variant-Thuseethan-Kuhanesan/e88364a478c8e2278b1aa5c4d7d539562c059549#citing-papers> [Accessed 8 Jun. 2019].



- Tripathy, R. and Daschoudhury, R. (2014). Real-time Face Detection and Tracking Using Haar Classifier on SoC. *International Journal of Electronics and Computer Science Engineering*, [online] p.175. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.447.1739&rep=rep1&type=pdf> [Accessed 9 Jun. 2019].
- Umm-e-Laila, Khan, M., Shaikh, M., bin Mazhar, S. and Mehboob, K. (2017). Comparative analysis for a real time face recognition system using raspberry Pi. *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)*. [online] Available at: <https://ieeexplore.ieee.org/document/8311984> [Accessed 1 Jun. 2019].
- Viola, P. and Jones, M. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2), pp.137-154.
- Wati, D. and Abadianto, D. (2017). Design of face detection and recognition system for smart home security application. *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*. [online] Available at: <https://ieeexplore.ieee.org/document/8285524> [Accessed 2 Jun. 2019].
- Yang, J. and Waibel, A. (n.d.). A real-time face tracker. *Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96*. [online] Available at: <https://ieeexplore.ieee.org/document/572043> [Accessed 2 Jun. 2019].

## Appendix B

Sort images script:

```
"""Script to sort Cohn-Kanade(CK) Dataset. Available at: http://www.consortium.ri.cmu.edu/ckagree/
Citing:
- Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis.
Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00),
Grenoble, France, 46-53.
- Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., & Matthews, I. (2010).
The Extended Cohn-Kanade Dataset (CK+): A complete expression dataset for action unit and emotion-specified
expression.
Proceedings of the Third International Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB
2010),
San Francisco, USA, 94-101.

Script adapted from: van Gent, P. (2016). Emotion Recognition With Python, OpenCV and a Face Dataset.
A tech blog about fun things with Python and embedded electronics. Retrieved from:
http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/ """

import glob
import natsort
from shutil import copyfile

# All directories are the directories in which my files are stored, and would change accordingly.

emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"] #Define emotion
order
participants = glob.glob("/Users/yenji/Desktop/Emotion-Detection/source_emotions/*") #Returns a list of all
folders with participant numbers
print(participants)
for x in participants:
    part = "%s" %x[-4:] #store current participant number
    print("Current Participant:", part)
    for sessions in glob.glob("%s/*" %x): #Store list of sessions for current participant
        for files in glob.glob("%s/*" %sessions):
            current_session = files[60:-30]
            print(current_session)
            file = open(files, 'r')
            emotion = int(float(file.readline())) #emotions are encoded as a float, readline as float, then convert to
integer.
            imagesPath = natsort.natsorted(glob.glob("/Users/yenji/Desktop/Emotion-
Detection/source_images/%s/%s/*" % (part, current_session))) #get path for images and sort them
            sourcefile_emotion = imagesPath[-1] #last image in directory contains the emotion
            sourcefile_neutral = imagesPath[0] #first image in directory contains neutral image
            dest_neut = "/Users/yenji/Desktop/Emotion-Detection/sorted_set/neutral/%s" %sourcefile_neutral[63:]
#Generate path to put neutral image
            dest_emot = "/Users/yenji/Desktop/Emotion-Detection/sorted_set/%s/%s" %(emotions[emotion],
sourcefile_emotion[63:]) #Do same for emotion containing image
            copyfile(sourcefile_neutral, dest_neut) #Copy file
            copyfile(sourcefile_emotion, dest_emot) #Copy file
```

## Appendix C

### Model training scripts:

#### Model training script – multiple:

```
""" Script that trains the Fisherface, EigenFace and LPBH Algorithms. This runs 10 times and stores the trained
Models that have been adapted to recognise emotions instead of faces.
Script adapted from: van Gent, P. (2016). Emotion Recognition With Python, OpenCV and a Face Dataset.
A tech blog about fun things with Python and embedded electronics. Retrieved from:
http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/
"""

import cv2
import glob
import random
import numpy as np
emotions = ["neutral", "anger", "disgust", "fear", "happy", "sadness", "surprise"] #Emotion list - Removed
Contempt

fishface = cv2.face_FisherFaceRecognizer.create()
eigenface = cv2.face_EigenFaceRecognizer.create()
lpbh = cv2.face_LBPHFaceRecognizer.create()
data = {}

def get_files(emotion): #Define function to get file list, randomly shuffle it and split 80/20
    files = glob.glob("/Users/yenji/Desktop/Emotion-Detection/datasetHaar1/%s/*" %emotion)
    random.shuffle(files)
    training = files[:int(len(files)*0.8)] #get first 80% of file list
    prediction = files[-int(len(files)*0.2):] #get last 20% of file list
    return training, prediction

def make_sets():
    training_data = []
    training_labels = []
    prediction_data = []
    prediction_labels = []
    for emotion in emotions:
        training, prediction = get_files(emotion)
        #Append data to training and prediction list, and generate labels 0-7
        for item in training:
            image = cv2.imread(item) #open image
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #convert to grayscale
            training_data.append(gray) #append image array to training data list
            training_labels.append(emotions.index(emotion))
        for item in prediction: #repeat above process for prediction set
            image = cv2.imread(item)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            prediction_data.append(gray)
            prediction_labels.append(emotions.index(emotion))
    return training_data, training_labels, prediction_data, prediction_labels

def run_recognizers():
    training_data, training_labels, prediction_data, prediction_labels = make_sets()
    print("training FisherFace, EigenFace and LBPH classifiers")
```

```

print("size of training set is:", len(training_labels), "images")
fishface.train(training_data, np.asarray(training_labels))
eigenface.train(training_data, np.asarray(training_labels))
lpbh.train(training_data, np.asarray(training_labels))
print("predicting classification set")

cnt = 0
correct = 0
incorrect = 0

correct_fisher = 0
incorrect_fisher = 0

correct_eigen = 0
incorrect_eigen = 0

correct_lpbh = 0
incorrect_lpbh = 0

for image in prediction_data:
    pred_fisher, conf_fisher = fishface.predict(image)
    pred_eigen, conf_eigen = eigenface.predict(image)
    pred_lpbh, conf_lpbh = lpbh.predict(image)

    if pred_fisher == prediction_labels[cnt]:
        correct_fisher += 1
    else:
        incorrect_fisher += 1
    fisher_score = ((100* correct_fisher)/(correct_fisher + incorrect_fisher))

    if pred_eigen == prediction_labels[cnt]:
        correct_eigen += 1
    else:
        incorrect_eigen += 1
    eigen_score = ((100* correct_eigen)/(correct_eigen + incorrect_eigen))

    if pred_lpbh == prediction_labels[cnt]:
        correct_lpbh += 1
    else:
        incorrect_lpbh += 1
    lpbh_score = ((100 * correct_lpbh) / (correct_lpbh + incorrect_lpbh))
    cnt += 1

return fisher_score, eigen_score, lpbh_score

#Now run it
metascore_fisher = []
metascore_eigen = []
metascore_lpbh = []

for i in range(0,10):
    correct_fisher, correct_eigen, correct_lpbh = run_recognizers()
    #print ("got", correct, "percent correct!")
    print ("Got ", correct_fisher, "(FisherFace) percent correct. Got ", correct_eigen, "(EigenFace) percent correct. Got ", correct_lpbh, "(LBPH) percent correct")
    metascore_fisher.append(correct_fisher)
    metascore_eigen.append(correct_eigen)

```

```

metascore_lbph.append(correct_lbph)

print("\n\nend score:(fisher)", np.mean(metascore_fisher), "percent correct!")
print("\n\nend score:(eigen)", np.mean(metascore_eigen), "percent correct!")
print("\n\nend score:(lbph)", np.mean(metascore_lbph), "percent correct!")
fishface.save('/Users/yenji/Desktop/Emotion-Detection/emotion_detection_model_Haar(fisher).xml')
eigenface.save('/Users/yenji/Desktop/Emotion-Detection/emotion_detection_model_Haar(eigen).xml')
lbph.save('/Users/yenji/Desktop/Emotion-Detection/emotion_detection_model_Haar(lbph).xml')

```

Model training script – individual:

```

""" Script that trains the Fisherface, EigenFace and LPBH Algorithms. This runs 10 times and stores the trained
Models that have been adapted to recognise emotions instead of faces.
Script adapted from: van Gent, P. (2016). Emotion Recognition With Python, OpenCV and a Face Dataset.
A tech blog about fun things with Python and embedded electronics. Retrieved from:
http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/
"""

import cv2
import glob
import random
import numpy as np

emotions = ["neutral", "anger", "disgust", "fear", "happy", "sadness", "surprise"] #Emotion list
fishface = cv2.face_FisherFaceRecognizer.create() #Initialize fisher face classifier
data = {}

def get_files(emotion): #Define function to get file list, randomly shuffle it and split 80/20
    files = glob.glob("/Users/yenji/Desktop/Emotion-Detection/datasetHaar/%s/*" %emotion)
    random.shuffle(files)
    training = files[:int(len(files)*0.8)] #get first 80% of file list
    prediction = files[-int(len(files)*0.2):] #get last 20% of file list
    return training, prediction

def make_sets():
    training_data = []
    training_labels = []
    prediction_data = []
    prediction_labels = []
    for emotion in emotions:
        training, prediction = get_files(emotion)
        #Append data to training and prediction list, and generate labels 0-7
        for item in training:
            image = cv2.imread(item) #open image
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #convert to grayscale
            training_data.append(gray) #append image array to training data list
            training_labels.append(emotions.index(emotion))
        for item in prediction: #repeat above process for prediction set
            image = cv2.imread(item)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            prediction_data.append(gray)
            prediction_labels.append(emotions.index(emotion))
    return training_data, training_labels, prediction_data, prediction_labels

def run_recognizer():
    training_data, training_labels, prediction_data, prediction_labels = make_sets()
    print ("training fisher face classifier")
    print ("size of training set is:", len(training_labels), "images")
    fishface.train(training_data, np.asarray(training_labels))
    print ("predicting classification set")

```

```

cnt = 0
correct = 0
incorrect = 0
for image in prediction_data:
    pred, conf = fishface.predict(image)
    if pred == prediction_labels[cnt]:
        correct += 1
        cnt += 1
    else:
        incorrect += 1
        cnt += 1
return ((100*correct)/(correct + incorrect))
#Now run it
metascore = []
for i in range(0,10):
    correct = run_recognizer()
    print("got", correct, "percent correct!")
    metascore.append(correct)
print ("\n\nend score:", np.mean(metascore), "percent correct!")
fishface.save('/Users/yenji/Desktop/Emotion-Detection/emotion_detection_model_Haar_Fisher1.xml')

```