CommuniTech Wall

Mood Analysis Sub-System

Fraser Campbell
Candidate Number: 11590
Word Count: 7,576

Table of Contents

Table of Figures

Introduction

CommuniTech Wall: System Overview

The CommuniTech Wall is a groundbreaking product designed to revolutionise communal spaces within educational and corporate settings through the integration of state-of-the-art interaction technologies and eco-friendly elements. Consisting of a physical wall and digital display this project aims to foster a sense of unity and interaction by merging advanced technology with practical applications, thereby enriching community culture and interaction.

The CommuniTech Wall is equipped with a static physical display, utilising conductive paint for interactive capabilities, alongside a dynamic digital display driven by a CPU. This CPU is powered through solar panels as part of the solar energy sub-system and operates multiple software applications including plant health monitoring, gesture control, and mood analysis. These functionalities are consolidated by the data integration sub-system, which channels data to the user interface system for visual display and interactive engagement.

Design Process

The design process for the CommuniTech Wall and mood analysis sub-system was methodically structured around the waterfall method, a sequential design approach emphasising planning and step-by-step execution as seen in Figure 1. This method was chosen to ensure that each phase of development—from user needs and requirement gathering to system design, implementation and testing—was completed before moving on to the next. From the onset, the project team engaged in rigorous requirement analysis for the overall system, derived from an initial design brief and extensive user needs assessments (Appendix A/B). These activities informed the development of functional and non-functional requirements for the mood analysis software, that aimed to ensure the sub-system was both technically robust and aligned with user expectations.

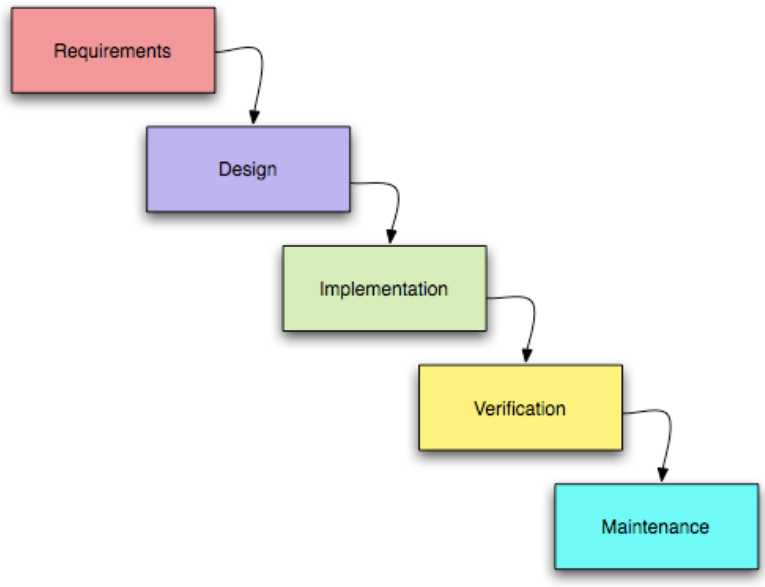


Figure 1: Waterfall design process *[1].*

Overview of the Mood-Analysis Sub-System

At the core of the CommuniTech wall is the mood analysis sub-system, which utilises advanced image processing and machine learning techniques to detect and respond to user emotions. The sub-system also incorporates a web framework to facilitate seamless integration with the CommuniTech UI

software, enabling efficient interfacing and data exchange. This sub-system is integral to the CommuniTech Wall, contributing significantly to its ability to offer personalised interactions. The development of this sub-system involved researching design alternatives that fit the requirements, implementation of designs, rigorous testing, and continuous refinement to ensure its effectiveness and reliability.

Integration within the System

Collaborative efforts across a multi-disciplinary team were crucial in integrating the mood analysis sub-system within the broader architecture of the CommuniTech Wall. This sub-system primarily interfaces with the wall display (UI) sub-system, transmitting data for presentation. Additionally, the mood analysis software shares CPU resources with the gesture control system. This report will explore the interdependencies between the sub-system and other components of the wall, highlighting how seamless integration was achieved to ensure cohesive functionality.

Overall, this report will detail the intricate processes and methodologies that underpin the design, implementation, and functionality of the mood analysis sub-system within the CommuniTech Wall. The forthcoming sections will delve deeper into the technical specifications, system architecture, implementation details, system solution lifecycle, and the comprehensive verification & validation strategies employed.

Design Specifications and Evaluation

Requirements

Functional Requirements

Functional requirements are essential for the sub-system's core operational functionalities [2]. Not meeting these implies the sub-system may fail to perform its designed tasks effectively, leading to a non-functional or only partially functional system. The table below lists the primary functional requirements, along with additional enhancements that, though not crucial, would significantly enhance performance and user experience.

Table 1: Sub-system functional requirements & target specification.

Functional Requirements	Target Specification
Face Detection	The system must reliably detect facial features using object detection algorithms to accurately delineate the area of video input containing the user's face.
	Additional spec: The system should be capable of performing face detection in varying lighting conditions and orientations.
Emotion Recognition	The system must detect and classify user’s emotions from 4 basic states (happy, sad, neutral, surprise).
	Additional spec: The system should be capable of performing mood recognition in varying lighting conditions and orientations.
Response Output	The system must transmit the emotional data to a web framework that interfaces this system with the UI and Display sub-system.

Non-Functional Requirements

Non-functional requirements define the quality attributes needed for the sub-system to be effective, ensuring it operates reliably, efficiently, and securely [2]. If these are not met, the sub-system may function but not satisfy user expectations, resulting in a system that operates but falls short of user needs. Table 2 details the key non-functional requirements

Table 2: Sub-system non-functional requirements & target specification.

Non-Functional Requirements	Target Specification
Performance	The mood detection algorithm should process input and provide output to the wall display within **1 second.**
Accuracy	The mood detection feature should achieve at least an **85% accuracy rate.**
Scalability	The system should be capable of handling simultaneous mood detection for up to **2-3 users.**
Security and Privacy	All personal data must comply with relevant data protection regulations such as the **General Data Protection Regulation** (GDPR).

User Needs

The functional and non-functional requirements and target specifications for the mood analysis sub-system were derived from the initial design brief, the team's user needs analysis, and the initial project specifications. The table below shows how the requirements above were derived from the user needs, ensuring the sub-system is designed to meet identified expectations. For detailed documentation on the user needs, please see the appendix.

Table 3: Relevant User Needs.

User Need	Requirement	Justification
Novel user interaction methods with user and environment (**N2**)	Facial Detection	Implementing edge detection algorithms for reliable facial detection caters to the need for innovative interaction methods and is required for mood recognition.
Novel user interaction methods with user and environment (**N2**) Cultivate a positive culture (**N5**)	Emotion Recognition	By recognising and reacting to a range of emotions, the system contributes to cultivating a positive culture through personalised user engagement.
Receive information over the Internet (**N13**)	Response Output	Real-time data transmission to the CommuniTech Wall's web framework meets this need.
State-of-the-art user interface (**N1**)	Performance **(1 Sec)**	The requirement for the mood detection algorithm to process input and provide output within 1 second caters to this need.

State-of-the-art user interface (**N1**)	Accuracy **(85%)**	Achieving at least an 85% accuracy rate in mood detection is required for providing a reliable user interface.
Adaptable to different locations (**N4**)	Scalability **(2-3 users)**	The system's ability to handle simultaneous mood detection for multiple users is essential for scalability and adaptability.
All data collected/displayed to comply with GDPR (**N14**)	Security and Privacy **(GDPR)**	Ensuring that all personal data comply with GDPR, and other relevant data protection regulations addresses the critical need for security and privacy.

Solutions & Alternatives

Face Detection

Haar Cascade Classifier:

The idea for a Haar Cascade classifier was first introduced by Paul Viola and Michael Jones in their 2001 paper 'Rapid object detection using a boosted cascade of simple features' [3]. In this paper, they outlined a machine learning-based approach for object detection where a cascade function is trained on a large data set of positive and negative images [3]. In the context of face detection positive images are images of faces, whereas negatives are images without faces.

To facilitate face detection, an image is first converted into greyscale. The Haar Cascade algorithm then utilises Haar-like features to analyse the intensity contrasts between adjacent regions of black and white pixels within the image to determine facial features [4]. The term 'cascade' refers to the use of multiple classifiers arranged in a sequence. Each classifier in the sequence is tasked with determining whether a face is present in the image [4]. This approach allows for quick rejection of non-face regions at early stages, while more complex calculations are reserved for promising face-like regions as the process progresses through the cascade. Figure 2 illustrates the Haar-like features and their applications in face detection algorithms, while Figure 3 presents an example diagram of a Haar Cascade Classifier.

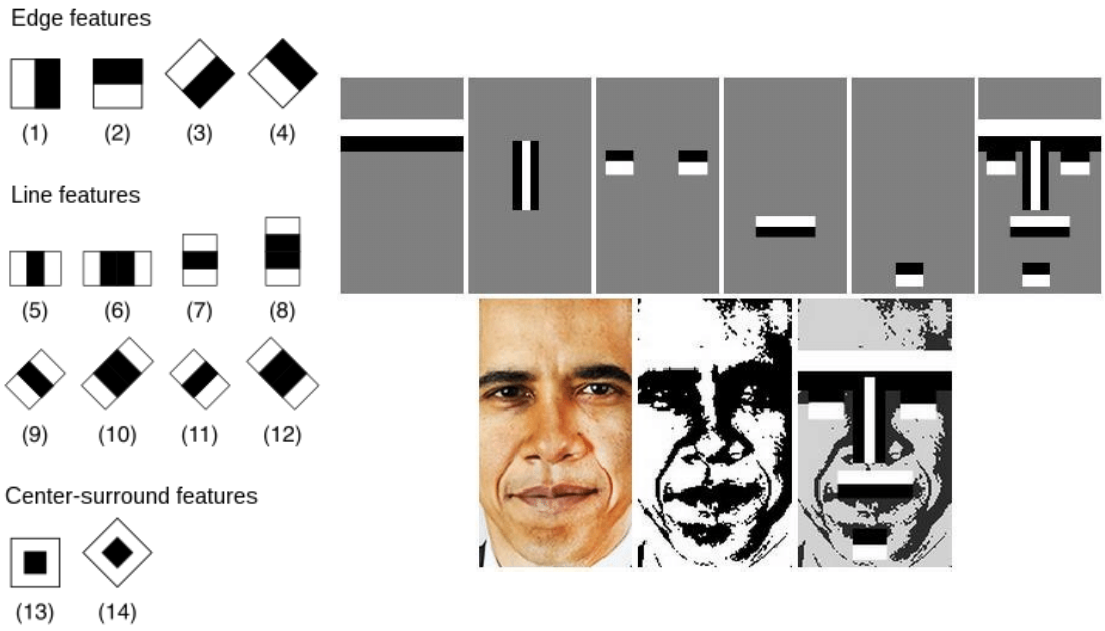


Figure 2: Haar feature sets and example in face detection [5] [6].

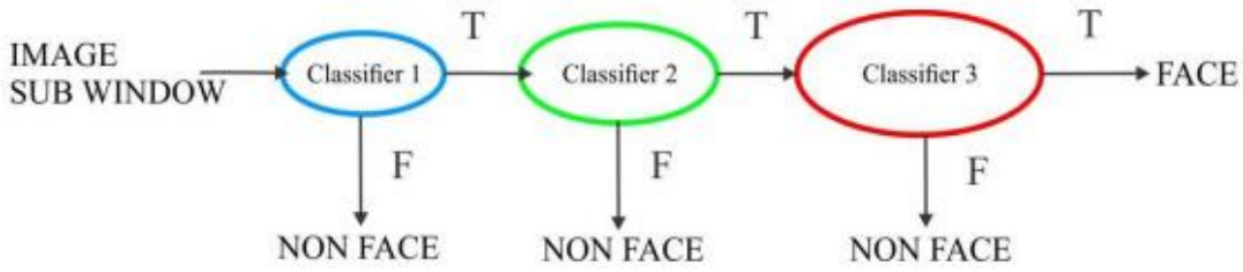


Figure 3: Cascade of classifiers flowchart [4].

Local Binary Patterns:

Local Binary Patterns (LBP) are a simple yet highly efficient texture descriptor that is widely used in image processing, particularly for face detection. First proposed by Timo Ojala in 1996 [7], LBP operates by comparing each pixel in an image with its surrounding 3x3 neighbourhood pixels. The process involves thresholding the neighbourhood of each pixel with the value of the centre pixel and considering the result as a binary number [7]. For instance, If the centre pixels value (intensity) is greater than the neighbour's value, write '0', if the neighbouring pixels value is greater than or equal to the centre's write '1'. The binary sequence of neighbouring pixels is then converted into a decimal number which becomes the LBP value for the centre pixel. These LBP codes are calculated across the entire image to create a feature histogram. The histogram of LBP codes forms a feature vector that describes the face in the image. This vector is what a classifier uses to determine whether a region contains a face based on learned patterns from training data. Figure 4, provides an example of Local Binary Patterns

X_0	X_1	X_2
X_7	X_c	X_3
X_6	X_5	X_4

2	1	3
7	4	1
9	3	6

0	0	0
1		0
1	0	1

Binary Sequence: 00001011
Decimal: 11

Figure 4: LBP operator example.

Histograms of Oriented Gradients:

The Histogram of Oriented Gradients (HOG) classifier is a feature descriptor used primarily in image processing for object detection tasks such as face detection. This technique involves dividing an image into small regions known as cells, computing the gradient direction or edge orientation for the pixels within each cell, and compiling these into a histogram [8]. These histograms are then normalised over larger, overlapping blocks of cells to improve accuracy and to provide resistance to changes in illumination. Figure 5, demonstrates the application of the Histogram of Oriented Gradients method in facial detection.

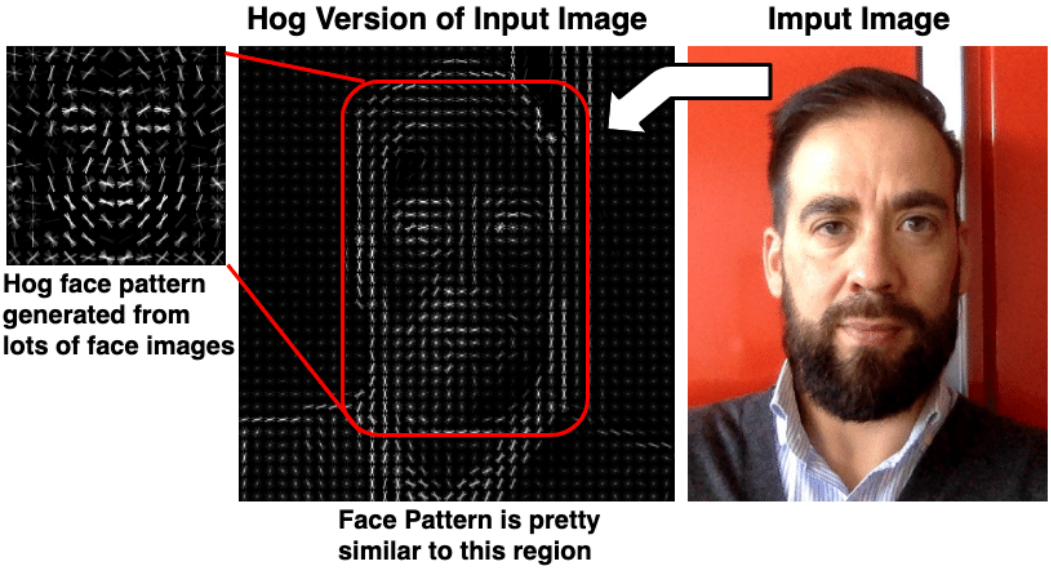


Figure 5: Application of HOG for face detection [9].

Comparative Summary – Face Detection

Table 4: Face detection comparison.

Haar Cascade		LBP		HOG	
Advantages	**Disadvantages**	**Advantages**	**Disadvantages**	**Advantages**	**Disadvantages**
Fast detection speed	Poor detection in uncontrolled environments (variations in lighting etc.)	Maintains accuracy in uncontrolled environments	Poor performance in complex backgrounds	Excellent at capturing edge details	High computational requirements
High accuracy for frontal faces	Low accuracy for non-frontal faces	Low computational requirements	Struggles with face orientation	Maintains accuracy with different face orientations	Difficult to implement
Low computational requirements	Can overfit characteristics if training data isn't diverse enough	Simple to implement	Sensitive to noise	Can handle diverse categories without extensive training	Slow detection speed

**Table not included in word count*

Table 5: Face detection criteria weighting.

Criteria	Haar Cascade	LBP	HOG	Weight (%)
Performance (Detection Speed)	3	2	1	25
Accuracy	2	1	2	25
Robustness	1	2	2	20
Computational Efficiency	3	3	1	20
Ease of Implementation	3	2	1	10
Overall Score	2.35	1.95	1.45	
Suitability	Best Option	Viable Alternative	Not suitable	

**Table not included in word count*

Based on the evaluation criteria and the specific requirements of the sub-system, the Haar Cascade Classifier stands out as the most suitable choice for face detection in the CommuniTech Wall's mood analysis sub-system. This decision is driven by its fast detection capabilities, minimal computational demands, and straightforward implementation process. Although the Haar Cascade Classifier does face challenges in handling varying lighting conditions and is less effective at detecting non-frontal faces, the operational context of the CommuniTech Wall largely mitigates these drawbacks. In the typical use-case environments of universities and corporate settings, users will predominantly interact with the wall face-on, and these locations generally provide consistent, adequate lighting. Therefore, the Haar Cascade Classifier not only meets but should excel in the expected operational conditions, making it the best choice for integrating into the mood analysis system.

Emotion Recognition

Convolutional Neural Network:

Convolutional Neural Networks (CNNs) are a class of deep learning models highly effective in image processing and analysis [10], making them particularly suitable for emotion recognition from facial expressions. CNNs operate by learning spatial hierarchies of features through multiple layers. The process begins with an input layer that takes an image, such as a human face. This is followed by several convolutional layers where feature extraction occurs through convolution operations. Each convolution layer applies an activation function like ReLU (rectified linear unit) to introduce non-linearity, enabling the network to capture complex patterns in the data. Pooling layers reduce the spatial dimensions of the data, summarising the outputs extracted in the convolutional layers into fewer elements. The learning process in CNNs is driven by backpropagation—a method for updating the kernels (set of learnable parameters [10]) to minimise the difference between the predicted and actual outputs [11]. Fully connected layers towards the end of the network integrate these learned features into high-level reasoning, culminating in an output layer that can classify the emotional state of the face into categories such as sad, happy, or neutral. Figure 6, illustrates the architecture of a CNN and illustrates the use of backpropagation in the training process.

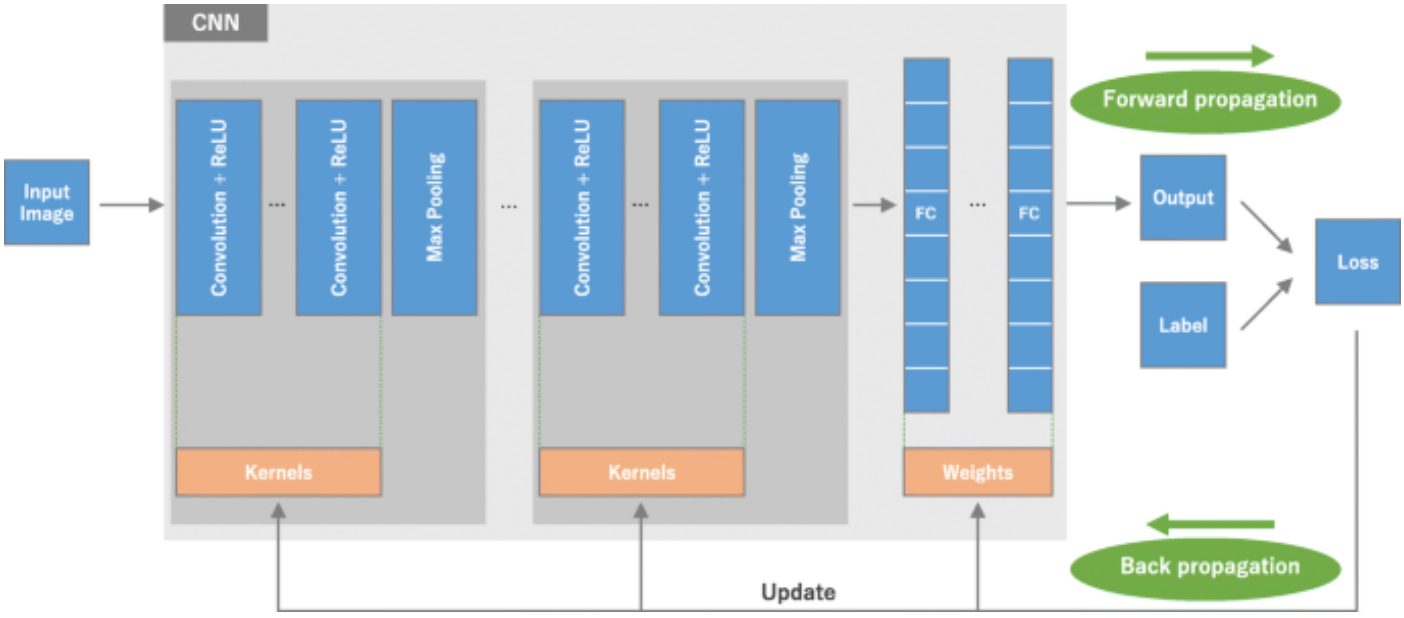


Figure 6: Overview of CNN architecture and training process [10].

Support Vector Machines:

Support Vector Machines (SVMs) are a machine learning algorithm that employs supervised learning [12] to tackle classification challenges. SVMs operate by transforming features to determine the boundaries between data points of predefined classes. The primary objective of SVM application is to establish a hyperplane that effectively separates and classifies data points into distinct classes, which in the context of emotion recognition, correspond to various emotional states derived from facial expressions. The critical elements in this configuration are the support vectors, which are the data points lying closest to the decision boundary [12], playing a pivotal role in defining the hyperplane. For emotion recognition, where expression features exhibit complex patterns, implementing a non-linear or Kernel SVM is necessary due to the non-linear relationships among the data. Figure 7, shows a graphical representation of a linear and non-linear SVM.

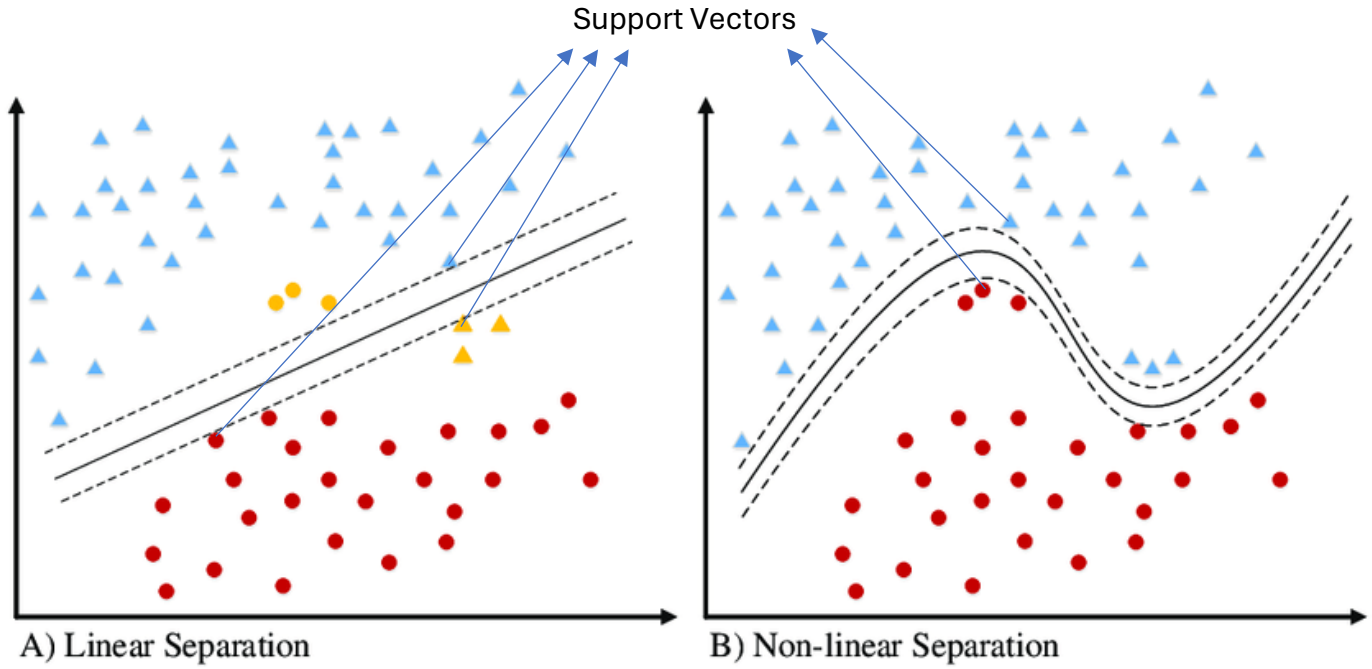


Figure 7: Linear and Non-linear (Kernel) SVM [13].

K-Nearest Neighbours:

K-Nearest Neighbours (KNN) is a straightforward machine learning method predominantly used for classification tasks. Each sample within a dataset is depicted as a data point within an N-dimensional space [14]. N being defined by the number of features such as pixel intensity, texture, and expression characteristics. When classifying a new sample, KNN calculates the distance from this sample to all others in the training dataset. The number of nearest neighbours, denoted as 'K' [14], is a critical parameter set by the programmer that significantly influences the algorithm's performance. For instance, if K is set to 4, the classification of the new sample will be determined by the majority among its four nearest neighbours; thus, if two neighbours are classified as 'happy', one as 'sad', and one as 'neutral', the new sample would be classified as 'happy'. Figure 8, demonstrates a visual representation of KNN.

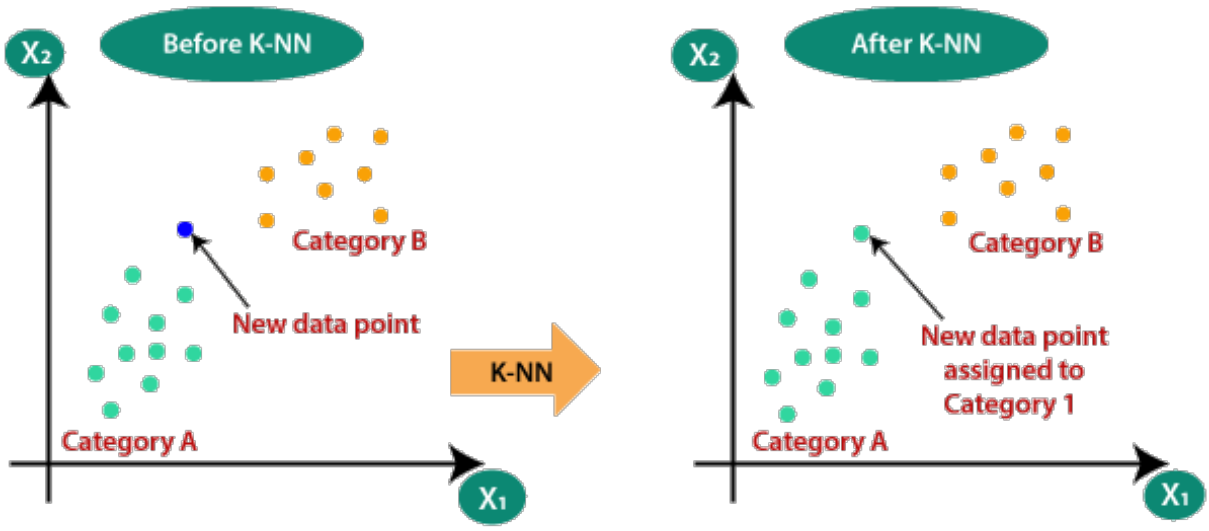


Figure 8: K-Nearest Neighbour method [14].

Random Forests:

Random Forests is an ensemble machine-learning technique that works by constructing multiple decision trees [15]. This method builds each decision tree from a random subset of the training data, and at each decision node within these trees, a random subset of features is chosen to split on. This randomness is called bootstrapping [16] and introduces diversity among the trees, reducing the risk of overfitting that is often seen with single-decision trees [15]. For tasks such as emotion recognition, each tree in the forest independently predicts a class for a given sample, and the final output class is determined by a majority vote among all the trees (Figure 9).

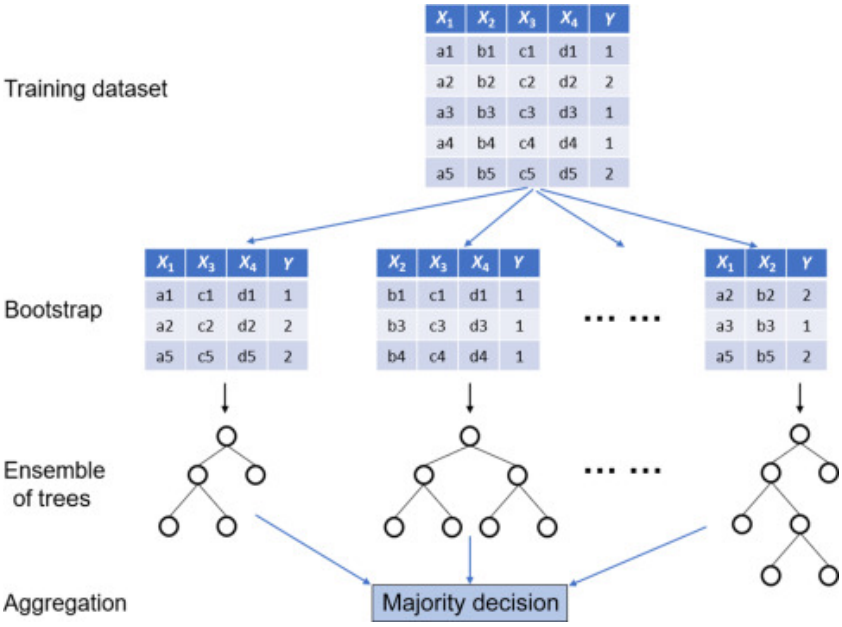


Figure 9: Random Forests visual [15].

Comparative Summary – Emotion Recognition

Table 6: Emotion recognition comparison.

	Advantages	Disadvantages
CNN	- High Accuracy - Robust - Fast detection speed	- High computational requirements - Requires large amount of data for training
SVM	- High Accuracy - Effective in high dimensional spaces	- Not suitable for large data sets - Sensitive to noise
KNN	- Simple to implement - Doesn't require training	- As dataset grows detection speed decreases - Performance highly dependent on K parameter
Random Forests	- Less prone to overfitting - High accuracy - Provides information about the importance of each feature	- High memory consumption - Training time can be very long for complex models (e.g., emotion recognition)

**Table not included in word count*

Table 7: Emotion recognition criteria weighting.

Criteria	CNN	SVM	KNN	Random Forests	Weight (%)
Performance (Detection Speed)	3	3	2	2	25
Accuracy	3	3	2	3	25
Robustness	3	2	1	3	20
Computational Efficiency	1	2	1	2	20
Ease of Implementation	2	2	3	2	10
Overall Score	2.50	2.50	1.70	2.45	
Suitability	Best Option	Viable Alternative	Not suitable	Viable Alternative	

**Table not included in word count*

After reviewing the alternatives, CNNs stood out as the best choice, primarily due to their fast detection speed and high accuracy. These attributes are crucial for reaching the required target specifications, hence their higher weighting. CNNs excel at processing complex image data and learning detailed patterns in facial expressions, which are vital for accurately capturing subtle emotional cues in real time. While SVMs and Random Forests offer viable alternatives with their respective strengths, they do not match the performance of CNNs in key areas. SVMs are particularly effective in high-dimensional spaces and offer good generalisation from limited training data. KNN is valued for its simplicity and ease of implementation, advantageous in systems where computational simplicity is prioritised.

While CNNs do present certain challenges such as high processing demands and the need for extensive training data, these issues are addressed within the project's infrastructure. The hardware specifications designated for the CommuniTech Wall are robust enough to meet the processing demands of CNNs, ensuring smooth and efficient operation. Additionally, the availability of extensive emotion recognition datasets online facilitates the training process, providing the necessary data to effectively train the CNN model. This accessibility to resources substantially mitigates the potential drawbacks of using CNNs, solidifying their status as the preferred choice for high-accuracy and real-time emotion recognition in the CommuniTech Wall project.

Response Output

Flask:

Flask is a lightweight and flexible micro-framework for Python [17]. It is designed to be simple and easy to use, making it ideal for small to medium web applications. Flask provides the basics like routing and request handling and allows developers to add other components as needed.

Flask can handle video streaming effectively using its response generation capabilities. For this system, a generator function could be used to yield video frames, which Flask can send to the client as a continuous stream using MIME multipart content type, typically used for streaming. Flask is well-suited for creating RESTful APIs, making it easy to implement an endpoint that clients such as the UI software can periodically poll to fetch the latest emotion data as JSON, which can then be dynamically displayed on a webpage.

Pyramid:

Pyramid is a web framework that can build web applications with minimal initial setup and can scale up to complex applications [18]. It is highly flexible and allows for the use of various templating and database solutions.

For video streaming, while not as straightforward as Flask, Pyramid can also be configured to stream video by handling HTTP range requests appropriately but implementation of this is slightly more complex. Like Flask, Pyramid can support the creation of JSON APIs that can serve emotion data. Its flexibility in handling various types of responses makes it a good candidate for this application.

Django:

Django is a high-level Python web framework that facilitates the rapid development of web applications [19]. As open-source software, it is well-regarded for its robust security features and scalability for Python-based web applications [19]. Among the three alternatives discussed, Django offers the most comprehensive suite of built-in features, including an administrative panel and user authentication, which can significantly accelerate development processes.

Whilst Django can handle streaming media files, it will require third-party packages since its built-in capabilities are more tuned towards serving HTTP content rather than handling continuous data streams like live video, similar to Pyramid. Django's robust framework includes capabilities for handling real-time data using Django Channels [20], which extends Django to handle WebSockets in a way that allows for scalability.

Comparative Summary – Response Output

Table 8: Web framework comparison.

	Advantages	Disadvantages
Flask	- Easy to implement - Flexible - Lots of extensions	- Scalability - Little built-in features
Pyramid	- Good for projects with hierarchical content - Flexible	- Complex implementation - Little built-in features
Django	- Fully featured - High level framework	- Complex implementation - Not very flexible

**Table not included in word count*

Table 9: Web framework criteria weighting table.

Criteria	Flask	Pyramid	Django	Weight (%)
Ease of Implementation	3	2	1	25
Flexibility	3	3	1	10
Built-in Features	1	1	3	20
Scalability	2	2	3	20
Suitability for Small Projects	3	2	1	25
Overall Score	2.40	1.90	1.80	
Suitability				

**Table not included in word count*

After reviewing web framework options for the response output requirement, Flask is the preferred choice to serve as the interface to the UI system. Its simplicity and ease of implementation align well with the system's needs for a basic framework primarily hosting a single page for video streaming and emotion data display. Although Flask may lack the scalability and built-in features of more comprehensive frameworks like Django, these limitations are minor for the project's scope. Flask's lightweight and adaptable nature efficiently supports the requirements, and its extensive documentation and support streamline troubleshooting and enhancements, making it highly suitable for the specific needs.

Technical Description

System Architecture

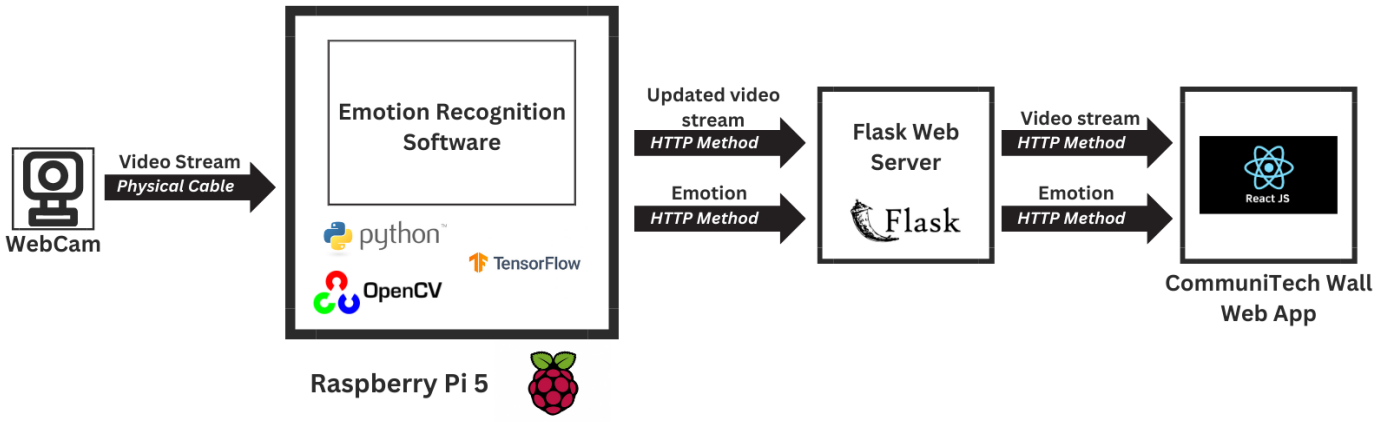


Figure 10: High-level sub-system architecture and data flow diagram.

Data Flow – Hardware

The mood analysis sub-system primarily relies on software therefore the hardware requirements are minimal. The necessary hardware components related to this sub-system include:

- Raspberry Pi 5 (8 GB model)
- Raspberry Pi Camera Module 3 with a Wide-Angle Lens & Cable

The Raspberry Pi 5 was selected as the foundational hardware platform for the data integration sub-system. This choice was driven by its sufficient computational capabilities and compatibility, which satisfy the performance and resource demands of the software for both gesture control and emotion recognition.

Figure 10, illustrates the hardware architecture and data flow mechanisms implemented within the emotion recognition system. Specifically, the architecture employs a webcam interfaced directly with a Raspberry Pi 5 via a physical cable. This Raspberry Pi is configured to run the full suite of emotion recognition software. Live video data captured by the webcam is transmitted to the Raspberry Pi, where it undergoes real-time processing by the emotion recognition application. This process involves the analysis of each frame for facial expressions and subsequent emotion detection. Additionally, the face detection and emotion recognition markers are superimposed onto the video stream before outputting. The updated video stream, along with the identified emotional data, is then forwarded to a Flask web server. This server facilitates the access of the processed video and emotion data so it can be requested by the CommuniTech Wall's user interface software.

Data Flow – Software

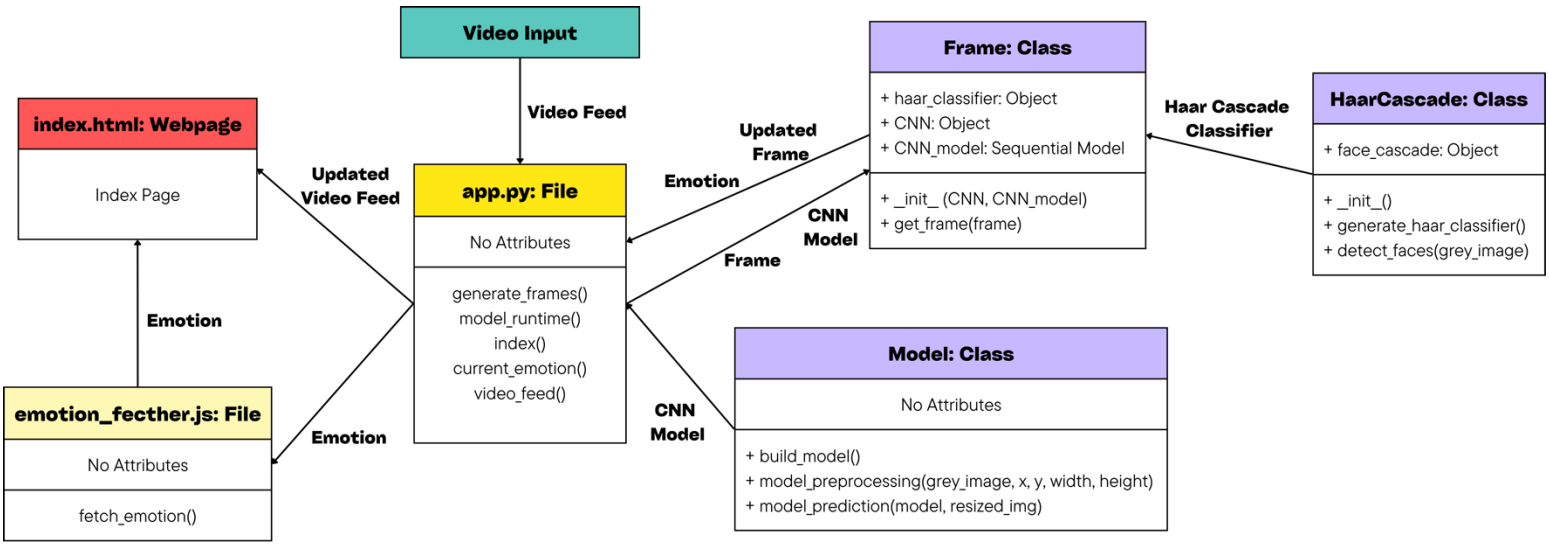


Figure 11: Emotion recognition UML and data flow diagram.

The software data flow for the mood analysis sub-system is depicted in the provided UML diagram (Figure 11), which illustrates the interactions between the system's components. For the full code, please see the "Mood Analysis Sub-System Code" section in Appendix C, as only excerpts are included in the technical description. Below is a detailed explanation of how data flows through the system:

1. **Video Input:**
   - Data flow begins with the video stream input from a webcam, which is physically connected to a Raspberry Pi. The video feed is continuously captured and fed into the Flask application (**app.py**).
2. **Face Detection:**
   - As the video stream is captured in **app.py,** its sent frame-by-frame to the **Frame** class.
   - The **Frame** class utilises **HaarCascade** to detect faces within each frame. The **HaarCascade** class, applies the Haar Cascade classifier to the image to identify facial regions.
3. **Emotion Recognition:**
   - Once faces are detected, the **Frame** class processes each facial region using the **Model** class. The **Model** class is responsible for the emotion recognition.
   - As seen in figure 11 the **Model** class is instantiated within **app.py** and passed to **Frame**, this is so the CNN model is not rebuilt for every frame.
   - It then pre-processes the facial images and then uses the model to predict emotions. This involves resizing the images to fit the model's input requirements and then running the prediction.
   - The CNN model's output is then used to annotate the original video frame with labels indicating the detected emotions.
4. **Data Output:**
   - The annotated video frame and emotion predicted are sent back to **app.py** where they are made available via a HTTP endpoint**.**
   - **emotion_fecther.js** is used to periodically fetch the emotion data from its HTTP endpoint and pass it to **index.html.**

- **index.html** directly requests the video feed from its endpoint and acts as the front end which displays the live video feed and emotion. Both video feed and emotion data is can then be requested by the CommuniTech wall UI software.

Haar Cascade Classifier – Face Detection

Design

As previously discussed, the Haar Cascade Classifier is a robust and efficient face detection method that utilises a pre-trained model. Rather than manually programming and training this, the OpenCV library offers a pre-defined Haar Cascade model, which is optimised for quick and effective face detection.

The OpenCV Haar Cascade operates through a series of classifiers arranged in a cascade, trained with numerous positive and negative images. This training enables the model to detect faces based on the learned features. A key component of this system is the haar_frontal_face XML file, which contains the data defining the Haar features used for face detection. This file is crucial as it outlines the parameters and features that the classifier uses to distinguish faces from the background, ensuring accurate detections within various environments and lighting conditions.

Implementation

Defining Haar Cascade Classifier & Detecting Faces – haar_cascade.py

Code

```
4   class HaarCascade:
5       def __init__(self):
6           self.face_cascade = self.generate_haar_classifier()
7
        1 usage
8       def generate_haar_classifier(self):
9           face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
10          return face_cascade
11
        1 usage
12      def detect_faces(self, grey_image):
13          faces = self.face_cascade.detectMultiScale(grey_image, scaleFactor=1.1, minNeighbors=5,
14                                                     minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)
15          return faces
```

Code Description

Constructor (__init__) - method:

- This method initialises an instance of the **HaarCascade** class. Upon instantiation, it immediately calls the **generate_haar_classifier** method to load the Open CV Haar Cascade classifier (line 6).

generate_haar_classifier - method:

- **Process:** This method creates an object of **cv2.CascadeClassifier** (line 9), loading the pre-trained Haar model from OpenCV's data repository.
- **Output:** OpenCV Haar Cascade Classifier (face_cascade).

detect_faces - method:

- **Input:** A grayscale image **(**gray_image**).**

- **Process:** The method applies the **face cascade** to detect faces within the image (line 13). It uses the **detectMultiScale** method of the cascade, which is crucial for identifying varying sizes of faces based on defined parameters such as scaleFactor, minNeighbors, and minSize.
- **Output:** Returns coordinates of detected faces in the form of rectangles (x, y coordinates, and width, height of the bounding box).

Frame Processing – frame_setup.py

Code

```
11      def get_frame(self, frame):
12          gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
13          faces = self.haar_classifier.detect_faces(grey_image=gray)
14
15          prediction = "Null"
16
17          for (x, y, w, h) in faces:
18              prepped_img = self.CNN.model_preprocessing(gray, x, y, w, h)
19              prediction = self.CNN.model_prediction(self.CNN_model, prepped_img)
20              cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 2, cv2.LINE_8, 0)
21              cv2.putText(frame, prediction, org: (x, y), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1,
22                          color: (255, 255, 255), thickness: 2, cv2.LINE_AA)
23
24          return frame, prediction
```

Code Description

get_frame - method:

- **Input:** frame from the video feed (frame).
- **Process:** First converts the incoming video frame to grayscale using OpenCV's **cvtColour** function (line 12), then calls the **detect_faces** method of the HaarCascade instance to return a list of rectangles containing a face (line 13). This list is then iterated through, and a rectangle is drawn on the frame in the area(s) that contain a face (line 20).
- **Output:** Returns the updated frame (frame) and its emotion prediction (prediction).

Convolutional Neural Network – Emotion Recognition

Design

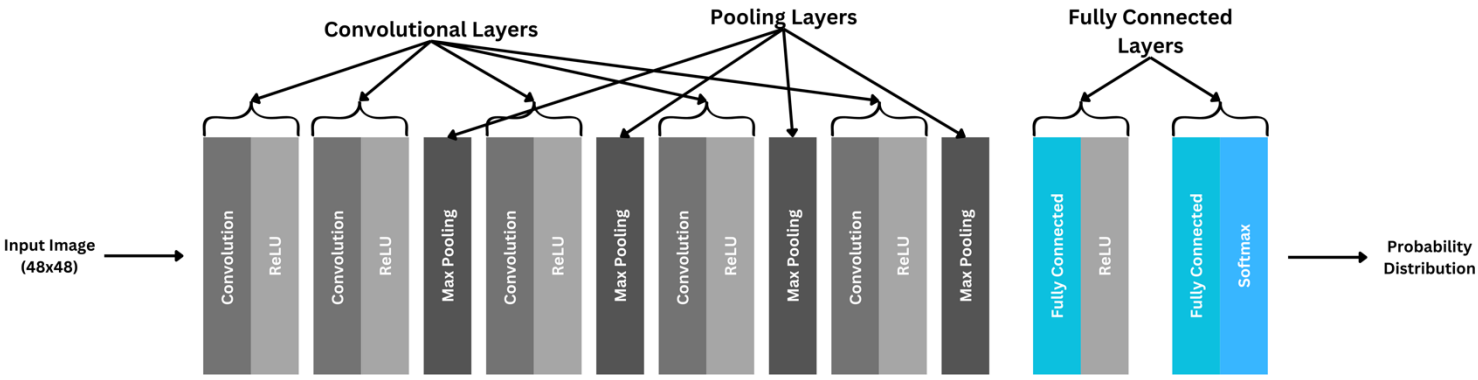


Figure 12: Emotion Recognition Convolutional Neural Network Architecture.

The CNN uses a Sequential model, the simplest neural network architecture available through TensorFlow (TF) and Keras. The Sequential model is ideal for tasks with a straightforward stack of layers, each with one input and output tensor [21], making it efficient for common image classification tasks. This model was chosen for its ease of implementation and effectiveness in handling linear layer stacks.

As seen in Figure 12, the network design includes five convolutional layers. Each convolutional layer is responsible for extracting various features from the images, such as edges, textures, and other relevant facial characteristics. In each convolutional layer an activation function is applied, these functions are used to calculate the output of each layer based on inputs and their weights [22], this output is then fed to the next layer as it's input. ReLU was chosen as the activation function for the convolutional layers due its effectiveness in non-linear transformations without demanding more computational power (Figure 13).

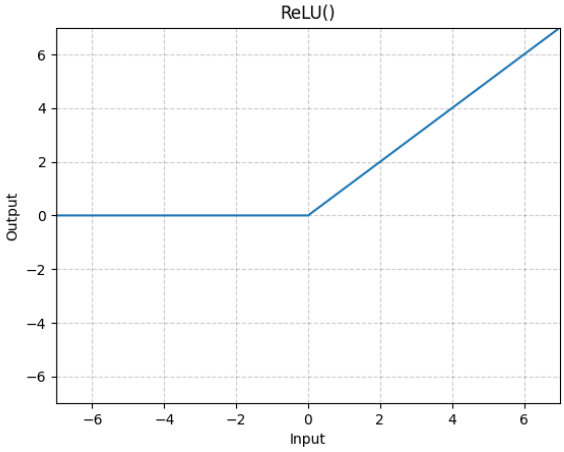


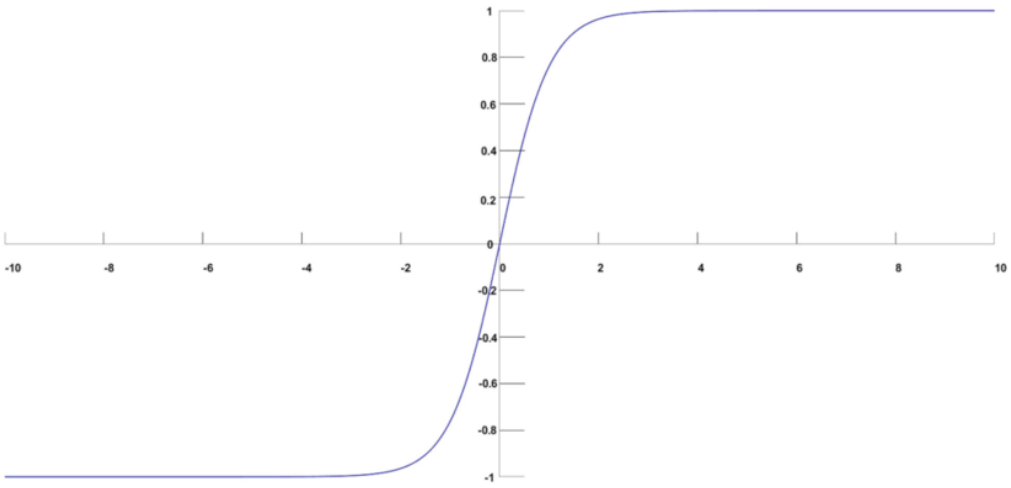
Figure 13: Rectified Linear Units (ReLU) graph [37]

$$f(x) = \begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0 \end{cases} - \textit{ReLU Function Formula} \qquad (1)$$

Pooling layers are spread between convolutional layers to reduce the spatial dimensions of the input (feature maps) for the next convolutional layer [23]. Max pooling is commonly used to achieve this reduction by selecting the maximum value from the feature maps obtained from convolutional layers. The design in Figure 12 includes four pooling layers, these layers work in reducing the computational load and the number of parameters in the network, helping to reduce model overfitting; when the model fits too closely to the training data and does not generalise well with new data [24].

Following the convolutional and pooling layers, the design includes two fully connected layers. The initial dense layer utilises the ReLU activation function. The final layer, which serves as the output layer, employs the softmax activation function (Figure 14). This function is particularly suited for classification as it transforms the outputs into a probability distribution consisting of values between

0 and 1 [25]. This configuration allows the model to predict the likelihood of each emotion based on the input facial features



$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

– Softmax Formula (2)

Figure 14: Softmax Function [38]

To conclude, the design of the CNN has been created to balance complexity and performance, whilst trying to ensure that the network is robust enough to recognise various emotions accurately, while being efficient enough to run in real-time applications like the CommuniTech Wall.

Implementation of CNN Model Training

CNN Model Training Process:

Before initiating the model training process, it is essential to have a well-prepared dataset that includes both training and validation subsets. Fortunately, numerous extensive datasets specifically designed for emotion recognition are readily available online. Model training involves adjusting the weights of the neural network based on error feedback from the predictive output compared to the actual output. Training the model requires several iterations over the entire dataset to minimise the loss function, the loss function quantifies the difference between the predicted and true values.

Training parameters:

Batch Size: Determines the number of training samples to work through before the model is updated [26]. Larger batch sizes provide a more accurate estimate of the gradient, but smaller batches often offer a lower generalisation error. A batch size of **64 images** has been chosen to balance accuracy of the gradient and generalisation error.

Epochs: One epoch consists of one full cycle through the training dataset [26]. More epochs mean more iterations over the data, which can lead to a more accurate model but also a higher risk of overfitting if not monitored with validation data. For this model **50 epochs** are to be iterated through.

Training and Validation Datasets:

Training Data: The dataset is iteratively presented to the model, with adjustments made to reduce the loss after each batch. The training data for the emotion recognition model is made up of **28,709 images**.

Validation Data: Not used directly for training but for evaluating the model's performance whilst training. It helps to monitor and tune the model's capacity to generalise to unseen data, helping to prevent overfitting. The validation data is made up of **7,178 images**. For examples of the training and validation data check Figure 15.

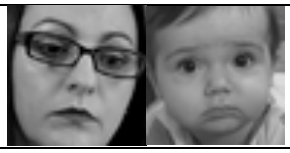
Happy	Neutral	Sad	Surprised
			

Figure 15: Example of images used in model training and validation [27].

Training and Validation Data Processing – train_model.py

Code

```
17  datagen = ImageDataGenerator(rescale=1./255)
18
19  train_data = datagen.flow_from_directory(
20          train_dir,
21          target_size=(48, 48),
22          batch_size=batch_size,
23          color_mode="grayscale",
24          class_mode='categorical')
25
26  val_data = datagen.flow_from_directory(
27          val_dir,
28          target_size=(48,48),
29          batch_size=batch_size,
30          color_mode="grayscale",
31          class_mode='categorical')
```

Code Description

The code snippet above shows how the training and validation images are loaded into the program using Keras's **ImageDataGenerator - flow_from_directory** method (line 19). First the data is normalised by rescaling the image pixels by a factor of $\mathbf{1/255}$ (line 17), this converts all pixel values to a range of 0 to 1. After normalisation the training and validation data is loaded into variables **train_data** and **val_data**, respectively (line 19-31). This also sets the image size to 48x48 pixels, processes images in grayscale, batches them, and treats labels in a categorical manner, which is necessary for classification tasks with multiple classes such as emotion recognition.

Model Training Execution – train_model.py

Code

```
58      model_info = model.fit(
59              train_data,
60              steps_per_epoch=num_train//batch_size,
61              epochs=num_epoch,
62              validation_data=val_data,
63              validation_steps=num_val//batch_size,
64              verbose=1)
65
66
67      model.save_weights('model_weights.h5')
```

Code Description

The **fit** function (line 58-64) trains the model for 50 epochs, on the dataset configured via the training and validation generators. It also provides parameters to define the number of steps per epoch and validation steps, which determine how many batches of samples to use from the generator per epoch for training and validation, respectively. After the model has been trained the network weights are saved into file '**model_weights.h5**', so they can be used when the model is built in the program.

Implementation of CNN Model

Building Model – model.py

Code

```
4 usages
12  class Model:
        2 usages
13      def build_model(self):
14          model = Sequential()
15
16          model.add(Conv2D( filters: 32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
17          model.add(Conv2D( filters: 64, kernel_size=(3, 3), activation='relu'))
18          model.add(MaxPooling2D(pool_size=(2, 2)))
19          model.add(Dropout(0.25))
20
21          model.add(Conv2D( filters: 128, kernel_size=(3, 3), activation='relu'))
22          model.add(MaxPooling2D(pool_size=(2, 2)))
23          model.add(Conv2D( filters: 128, kernel_size=(3, 3), activation='relu'))
24          model.add(MaxPooling2D(pool_size=(2, 2)))
25          model.add(Conv2D( filters: 128, kernel_size=(3, 3), activation='relu'))
26          model.add(MaxPooling2D(pool_size=(2, 2)))
27          model.add(Dropout(0.25))
28
29          model.add(Flatten())
30          model.add(Dense( units: 1024, activation='relu'))
31          model.add(Dropout(0.5))
32          model.add(Dense( units: 4, activation='softmax'))
33          model.load_weights('model_weights.h5')
34
35          return model
```

Code Description

Build_model – method:

- **Process:** As per the design the model includes five convolutional layers, four pooling layers, and two fully connected layers (**Dense**). The first convolution layer (line 16) acts as the input layer, set with dimensions of 48x48 to match the training images. After building the model, the pre-trained weights are loaded (line 33).
- **Output:** Fully configured model (model).

Image Processing & Model Prediction – model.py

Code

```
37      def model_preprocessing(self, grey_image, x, y, width, height):
38          roi_grey = grey_image[y:y + height, x:x + width]
39          resized_img = np.expand_dims(np.expand_dims(cv2.resize(roi_grey, dsize: (48, 48)), -1), axis: 0)
40          return resized_img
41
        1 usage (1 dynamic)
42      def model_prediction(self, model, resized_img):
43          emotion_dict = {0: "Happy", 1: "Sad", 2: "Neutral", 3: "Surprised"}
44          prediction = model.predict(resized_img)
45          max_index = int(np.argmax(prediction))
46          return emotion_dict[max_index]
```

Code Description

model_preprocessing – method:

- **Input:** greyscale image (grey_image), coordinates of area containing a face (x, y, width, height).
- **Process:** This method initially extracts the region of the grayscale image that contains the face (line 38) and resizes the image to 48x48 (line 39).
- **Output:** Processed image (resized_img).

model_prediction – method:

- **Input:** Configured model (model), processed image (resized_img)
- **Process:** This method, is deployed for making emotion predictions. Parameters passed into the method include the model and the processed image. The model evaluates the emotion of the face in the processed image (line 44), and this prediction is subsequently converted into an integer ranging from 0 to 3 (line 45). This integer index is used to retrieve the corresponding emotion description from the emotion dictionary (line 43).
- **Output:** Emotion detected (emotion_dict[max_index]).

Flask Application – Response Output

Design

The Flask application serves as the central hub for interfacing the mood analysis sub-system with the user interface displayed on the CommuniTech Wall. The Flask application is structured with a modular design, segregating different functionalities in distinct components, this design approach helped simplify the development process by allowing components to be developed, tested, and updated independently.

A fundamental design requirement for the Flask application is the real-time processing and transmission of video and emotion data. This capability is vital for maintaining the application's performance and responsiveness, aligning with the target specifications detailed in Table 1. This objective has been realised by encoding each video frame and transmitting it to the webpage

(**index.html**) as a continuous stream of bits, effectively minimising latency and increasing responsiveness.

Implementation

Flask App initialisation – app.py

Code

```
8   app = Flask(__name__)
9   cors = CORS(app)
10  app.config['CORS_HEADERS'] = 'Content-Type'
11  CORS(app, resources={r"/*": {"origins": ["172.26.79.99", "127.0.0.1"]}})
```

Code Description

Flask Instance: An instance of the Flask class is created, named app (line 8). This instance acts as the central object through which all request handling is coordinated.

CORS Setup: Cross-Origin Resource Sharing (CORS) is configured for the Flask app to allow HTTP requests from specified origins (line 9-11), ensuring that the frontend can securely interact with the backend without cross-origin issues.

Frame Generation – app.py

Code

```
15  def generate_frames():
16      global emotion
17
18      CNN, CNN_model = model_runtime()
19
20      while True:
21          success, frame = cam.read()
22          setup = frame_setup.Frame(CNN, CNN_model)
23          frame, emotion = setup.get_frame(frame)
24
25          if not success:
26              break
27          else:
28              ret, buffer = cv2.imencode( ext: '.jpg', frame)
29              frame = buffer.tobytes()
30
31              yield (b'--frame\r\n'
32                     b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Code Description

generate_frames – function:

- **Process:** This function assigns the configured model to CNN_model (line 18). It processes the video feed frame-by-frame (line 21), using the Frame class's get_frame function to annotate frames with face detection and emotion recognition markers and determine the predicted emotion (line 23). Frames are then encoded into JPEG format and converted to bytes for HTTP streaming using OpenCV's imencode function (lines 28-29).
- **Output:** Encoded frames are yielded in a format that is compatible with HTTP multipart streaming (line 31-32).

Flask Routes – app.py

Webpage – ‘index’ route

```
43    @app.route('/')
44    @cross_origin()
45 <> def index():
46        return render_template('index.html')
```

Code Description

- This route serves the main HTML page where the video stream and emotion data are displayed.

Emotion – ‘current_emotion’ route

```
49    @app.route('/current_emotion')
50    @cross_origin()
51    def current_emotion():
52        return jsonify(emotion=emotion)
```

Code Description

- Provides the latest emotion detected as a JSON response, allowing the frontend (**emotion_fetcher.js**) to fetch and display this data dynamically.

Video Feed – ‘video_feed’ route

```
55    @app.route('/video_feed')
56    @cross_origin()
57    def video_feed():
58        return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
```

Code Description

- Streams the processed video frames to the client. The route uses the **generate_frames()** function to provide a continuous stream of JPEG images, ensuring the video is displayed in real-time on the frontend.

System Integration and Final Output

This section demonstrates the successful culmination of the mood analysis sub-system components. This includes the integration of face detection, emotion recognition, and the Flask web framework. Each element, developed in previous phases, now works in concert to provide a cohesive and functional output as seen in Figure 16.

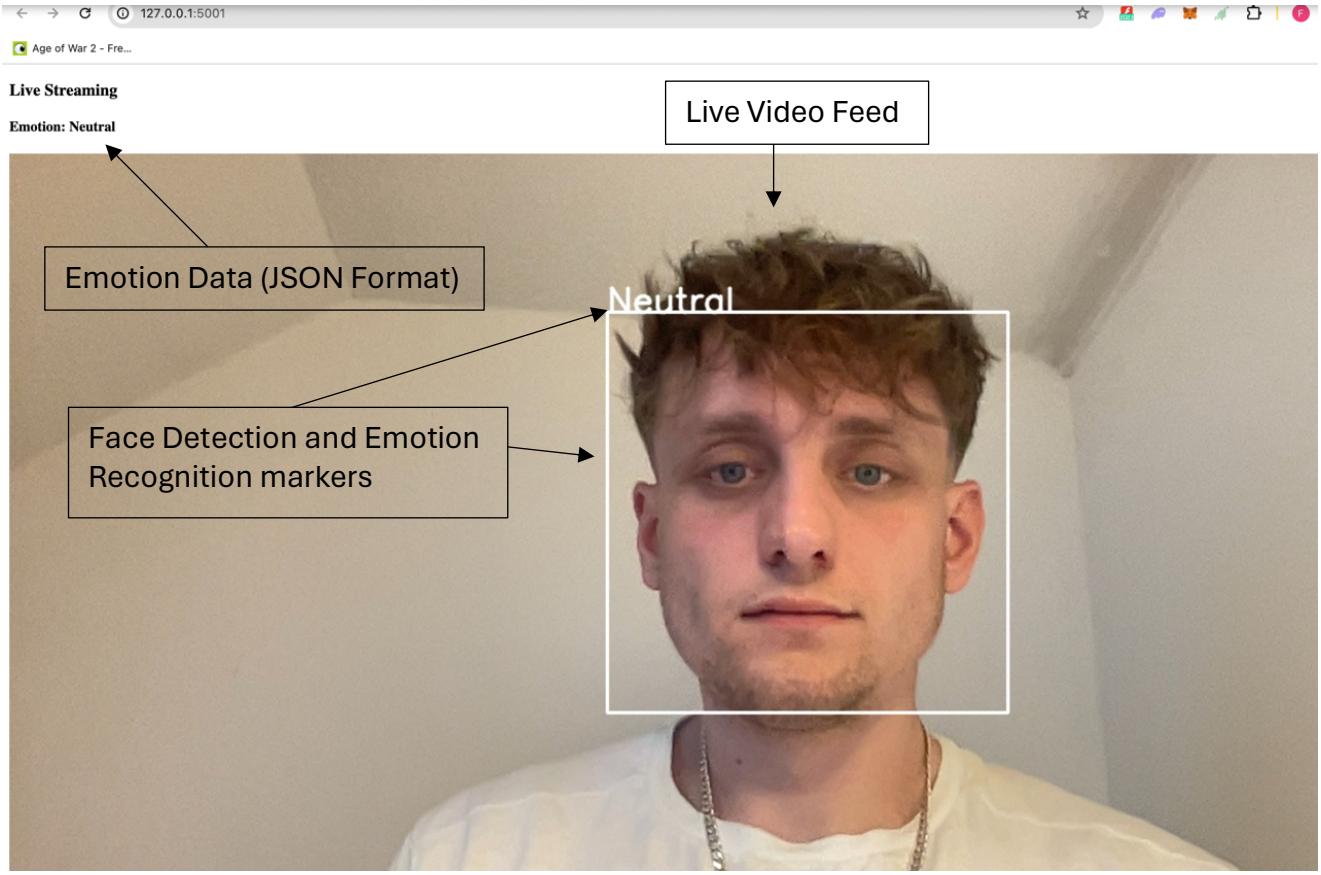


Figure 16: Mood Analysis Sub-System Final Output.

Testing - Validation & Verification

Unit Testing - Validation

Unit testing is a form of software testing that refers to the testing of individual functions or units isolated from the source code [28]. The primary purpose of unit testing in this context is to ensure that each individual part, such as image pre-processing routines and emotion prediction algorithms, operate correctly and as expected, independent of external integrations.

For conducting these tests, the standard Python unit testing framework '**unitttest**' was used. This framework allows execution of automated tests that can simulate use-case scenarios for each function in the system. In the code, not every function necessitates extensive unit testing. Functions with straightforward operations, such as the **generate_haar_cascade** method described in the Haar Cascade Classifier Implementation section, may not require rigorous testing. This approach is adopted to efficiently allocate testing resources where they are most needed, ensuring a focus on more complex functionalities. The example in Appendix D 'Unit Testing' shows an example of how testing was performed for the image processing and emotion prediction functionalities as well as the test results.

Integration Testing - Validation

Integration testing is a critical phase in software development where individual software modules are integrated and tested as a group. The primary goal of this testing phase is to evaluate the system's overall functionality and compliance by examining the compatibility and interoperability of the combined modules [29]. Integration testing is essential for identifying defects that may arise from interactions between integrated modules, which are not detectable when modules are tested in isolation.

For this system an incremental integration testing approach has been used. This strategy starts with the integration of two modules such as the **Frame** class and **HaarCascade** class, and gradually incorporating other modules such as **Model**. To illustrate the application of integration testing within the mood analysis sub-system, the examples in Appendix E 'Integration Testing' demonstrate how incremental testing was employed to ensure effective integration of system components.

Verification

In testing, verification refers to the process of evaluating the system to ensure that it adheres to the defined requirements and target specifications. This section will evaluate the sub-systems performance and capabilities against the functional & non-functional requirements (Tables 1/2).

Verification of Functional Requirements:

Face Detection:

- **Objective:** Verify the systems capability to detect facial feature under various conditions.
- **Method:** Test the face detection software in multiple environments and lighting conditions.
- **Results:** The verification results showed that the Haar cascade classifier reliably detected frontal faces in various environments. However, it occasionally missed faces in complex backgrounds and environments with minimal lighting. While it met the primary requirements and specifications, it fell short in handling complex backgrounds, suggesting the need for further refinement (Figure 17).

Figure 17: Face Detection Testing.

Emotion Recognition:

- **Objective:** Ensure the system can classify the user's mood and test its capability in various conditions.
- **Method:** In alignment with the approach used for face detection, the system will undergo user testing across diverse environments and lighting conditions.
- **Results:** The test showed that the system consistently identified user moods across different environments. The model's performance was stable even in complex backgrounds and varying lighting, if faces were detected. This reliability is likely due to the extensive training dataset, which helped the model generalise well across scenarios. The system met the specified requirements and targets (Figure 18).

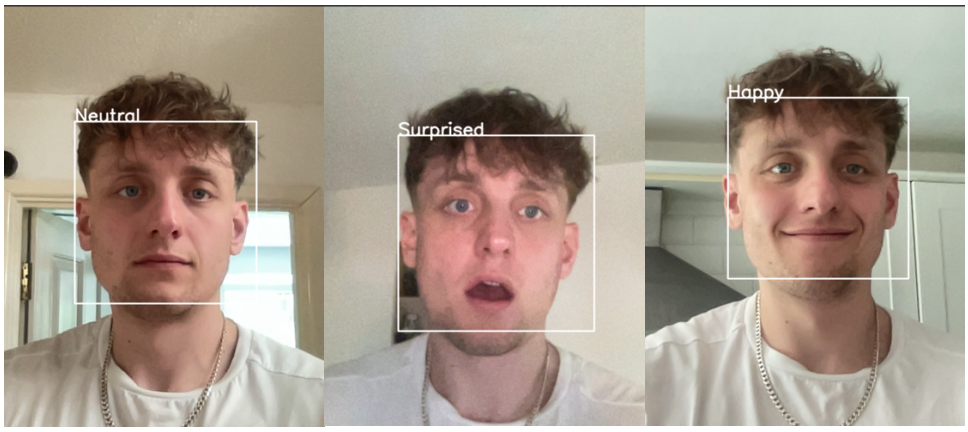


Figure 18: Emotion Recognition Testing.

Response Output:

- **Objective:** Confirm the system's ability to send live video feed and emotion data out via HTTP methods with minimal latency.
- **Method:** The system was tested for its ability to transmit the live video feed and emotion data efficiently via HTTP. The test involved measuring the latency and integrity of data transmission from the software to the server webpage (**index.html**) under normal operational conditions, ensuring real-time updates without significant delays.
- **Results:** The results found that the system was able to efficiently upload real-time emotional data and a live video to the **index.html** page with minimal latency. The results show that the requirement and specification have been met.

Verification of Non-Functional Requirements:

Performance:

- **Objective:** Test the real-time responsiveness of the mood detection algorithm.
- **Method:** To evaluate this requirement, a timer was integrated into the code utilising Python's standard time library. This timer starts just before the model begins making predictions and stops once the prediction has been output. The recorded times are stored in a list and subsequently used to plot a graph, allowing for an assessment of the model's performance across multiple emotion detection iterations.
- **Results:** The test outcomes were unexpectedly positive, with the model's prediction time averaging **60 milliseconds** per detection, significantly faster than the anticipated 1 second. This performance far exceeds the initial target specification (Figure 19).

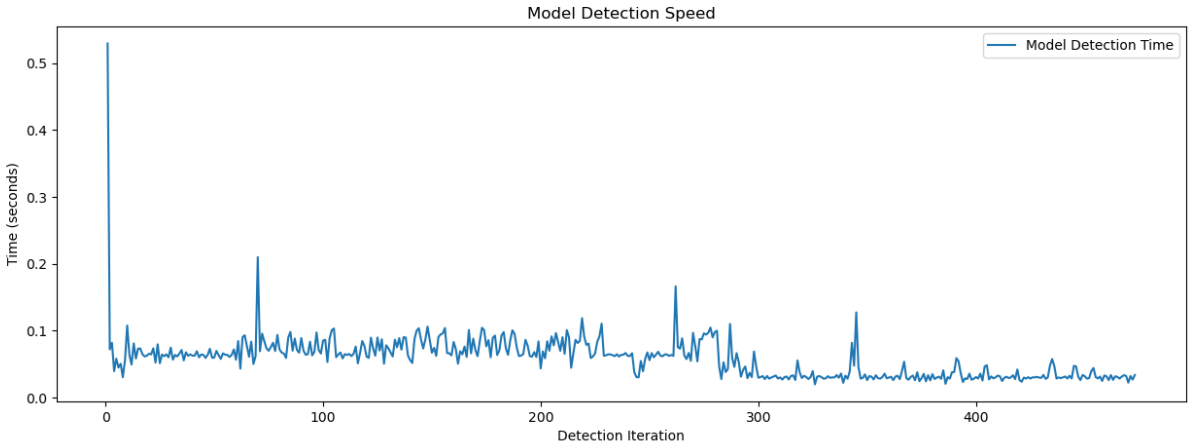


Figure 19: Model Emotion Detection Speed.

Accuracy:

- **Objective:** Test the accuracy of the mood detection algorithm.
- **Method:** During each testing epoch, the model evaluates the validation data, yielding measures of validation accuracy and loss. These metrics are a good indication on how the model will perform in use.
- **Results:** The results showed the training accuracy reached about **80%,** confirming the model is learning effectively from the training data. However, the more crucial validation accuracy, which gauges generalisation to new data, stood at **56%.** This indicates that while the model can generalise to some extent, it does not meet the target specification and requires further improvement (Figure 20).

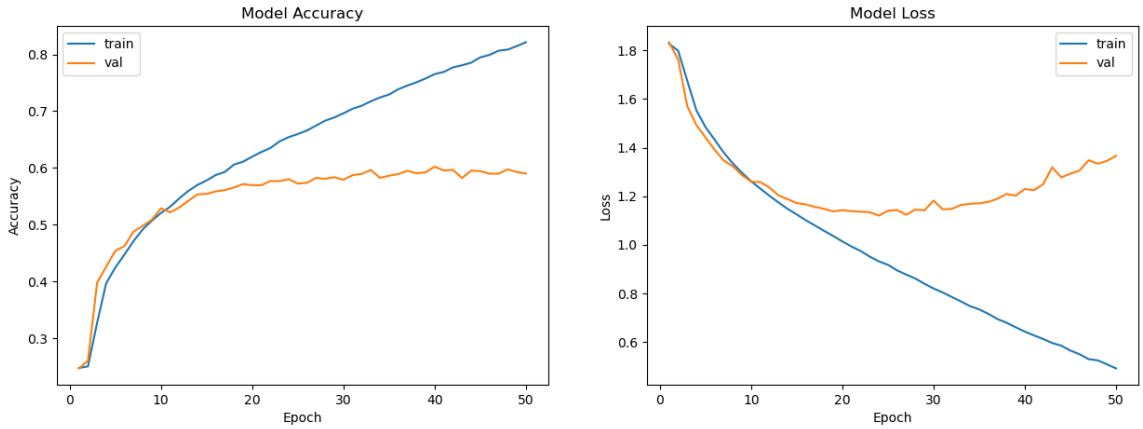


Figure 20: Model Accuracy & Model Loss.

Scalability:

- **Objective:** Measure the systems capability when performing with multiple users.
- **Method:** Conduct comprehensive user testing of the entire system with at least two users to assess the scalability of the sub-system and its performance under multi-user conditions.
- **Results:** The results demonstrated that the system maintained consistent performance with 1-3 users. However, it was observed that the system occasionally became slow and unresponsive when handling more than three users, likely due to increased CPU usage required to process multiple faces simultaneously. Despite these occasional slowdowns, it was determined that the target specification for scalability has been met.

Security and Privacy:

- **Objective:** Ensure all data complies with GDPR
- **Method:** Conduct a thorough audit of the software and data to ensure compliance with GDPR standards, particularly concerning the handling of personal data, including images and emotion-related information.
- **Results:** Under the GDPR, 'personal data' includes any information relating to an identifiable person [30], meaning images with faces qualify as such. While the mood analysis sub-system does not store data, avoiding many GDPR requirements, it still processes personal data.

Therefore, it must comply with two specific GDPR requirements:

Consent – Users must provide explicit consent for their data to be processed [30]. In the context of this system, this means implementing a clear mechanism for users to consent to their facial and emotional data being captured and analysed.
Data Minimisation – The system should only collect data that is necessary for the specified purpose. The software only needs to analyse emotions for real-time interaction and does not store this data, ensuring that no unnecessary data is retained.

Solution Lifecycle

Costing Analysis

The mood analysis sub-system of the CommuniTech Wall was developed with a keen focus on cost-effectiveness. A detailed cost estimation process was integral to the project's management strategy, ensuring that all financial resources were allocated efficiently. Overall, the project prototype was allocated a budget of £500, which was intended to cover expenses for all sub-systems. Fortunately, the mood-analysis sub-system primarily consists of software components, although some hardware was necessary to support the software's functionality (part of the data integration sub-system).

Detailed below are the hardware components utilised and their associated costs:

- Raspberry Pi 5 (8 GB model) - **£62.99** [31]
- Raspberry Pi Camera Module 3 with a Wide-Angle Lens - **£27.56** [31]

Once the CommuniTech Wall enters production, additional expenses will be incurred in relation to the mood analysis sub-system. Compliance with GDPR requires a data protection fee payable to the Information Commissioner's Office (ICO), which ranges from **£40 to £60** for small to medium-sized enterprises [32]. Additionally, a server will be necessary to facilitate the web framework that manages data exchange between the mood analysis sub-system and the Wall UI system. The cost for virtual private server (VPS) hosting typically varies between **£10 and £71** [33].

Purchasing and Logistics

The mood analysis sub-system required minimal purchasing and logistics compared to hardware-intensive systems. The development focused on utilising open-source libraries and tools that are freely available, eliminating the need for extensive procurement processes. This approach significantly reduced the logistical complexities typically associated with negotiating licenses and managing vendor relationships. By leveraging existing software solutions, the project streamlined the integration process, ensuring a smooth development flow without the delays that can accompany license acquisition and vendor coordination.

Maintenance

To guarantee the long-term viability of the mood analysis sub-system, a comprehensive maintenance plan was developed, which includes regular updates and routine checks to ensure optimal performance.

Software

To maintain performance and functionality, the software requires regular checks and updates in line with the latest releases of Python and TensorFlow. Both the programming language and this crucial dependency undergo periodic updates approximately every few months, with major updates typically occurring once a year [34] [35]. These major updates may necessitate modifications to the code to ensure compatibility with the latest versions, confirming that all components remain current and fully functional. To achieve this, conducting a maintenance check and updating the software, if necessary, every six months should be sufficient to keep pace with changes and ensure optimal performance.

Hardware

Although the hardware is technically part of the data integration sub-system, it is important to include it in this report. While there is no definitive lifespan for a Raspberry Pi 5, user experiences suggest that it can be expected to last between 4 to 8 years. This means that each site equipped with a CommuniTech Wall will need to update this hardware within that timeframe to ensure continued functionality and performance. To ensure this happens, CommuniTech will need to send a maintenance team to replace this hardware every 4 years.

Sustainability

The sustainability of the mood analysis system is contingent on the durability of the hardware running it and its power consumption. In alignment with the proposed maintenance plan, hardware is scheduled for replacement every 4 years. To enhance sustainability practices, the decommissioned hardware will be recycled through Okdo, which provides a Raspberry Pi recycling service which aims to mitigate the UK's E-waste problem [36]. Additionally, the hardware will be powered by solar panels, a key component of the solar energy sub-system. This approach not only minimises the CommuniTech Wall's carbon footprint but also curtails energy consumption from non-renewable sources, further bolstering the system's environmental sustainability.

Conclusion

In conclusion, the development of the mood analysis sub-system within the CommuniTech Wall project has successfully demonstrated the application of advanced technology to enhance interactive communal spaces. By integrating state-of-the-art machine learning techniques, this sub-system has fulfilled its intended purpose, significantly contributing to the overall effectiveness and appeal of the CommuniTech Wall.

The structured application of the waterfall method in the project’s development ensured a disciplined and systematic approach for the whole team, facilitating the completion of each phase—from the initial requirement analysis through to the final testing and integration stages. This method proved instrumental in maintaining a clear progression of project milestones. For project management, the team adopted an Agile methodology, complementing the structured phases of the waterfall design process with its focus on incremental delivery. This approach involved conducting regular weekly stand-up meetings to monitor progress and a weekly sprint review to discuss tasks, receive feedback, resolve conflicts, and assign new tasks to each team member. To manage workload equitably among

team members, a points-based system was implemented. Each task was assigned a specific number of points reflecting its complexity and effort required. This ensured that each week, all team members were assigned a balanced amount of work, preventing any individual from being overburdened or underutilised.

Like every project, the mood analysis sub-system faced inherent risks, including software bugs, maintenance challenges, and data management risks amongst others. Mitigation strategies were deployed to minimize these risks effectively.

Robust Testing and Quality Assurance: To reduce the risk of software bugs, extensive testing, including unit, integration, and user testing, was conducted. This rigorous testing process will need to continue post-production with each software update.

Compliance Audits: To ensure adherence to GDPR, audits of software and data management practices were conducted, focusing on the handling of personal data. These audits will remain critical throughout the system's lifecycle.

Quality Components and Training: High-quality, durable components were chosen to minimize maintenance demands. Additionally, comprehensive training programs should be developed to ensure that maintenance staff are well-equipped to manage and maintain the system efficiently.

Future work

Advanced Emotion Recognition Algorithms

To increase the accuracy of the existing emotion recognition algorithm, currently at **56%,** the development of a more sophisticated deep learning algorithm is recommended. Given additional development time, this new algorithm could significantly improve the software's accuracy. Incorporating body language analysis into the algorithm could further refine emotion detection capabilities. However, this enhancement would necessitate a more complex deep learning model with additional layers, which would, in turn, require substantially greater computational resources.

Real-time Adaptation

Rather than relying solely on a pre-trained model using historical training data, developing an algorithm that adapts in real-time to individual user feedback could be explored. This adaptive approach would enhance the model's accuracy progressively, refining its performance based on ongoing user interactions.

Bibliography

[1] D. Hughey, “Comparing Traditional Systems Analysis and Design with Agile Methodologies,” 2009. [Online]. Available: https://www.umsl.edu/~hugheyd/is6840/waterfall.html. [Accessed 8 April 2024].

[2] “Functional vs Non Functional Requirments,” geeksforgeeks, 22 May 2023. [Online]. Available: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/. [Accessed 24 04 2024].

[3] P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features,” 2001.

[4] C. Rahmed, “Comparison of Viola-Jones Haar Cascade Classifier and Histogram of Oriented Gradients (HOG) for face detection,” IOP Conference Series: Materials Science and Engineering, 2020.

[5] R. Lienhart, “An extended set of Haar-like features for rapid object detection,” IEEE, 2002.

[6] K. Kadir, “Haar-like features application,” August 2014. [Online]. Available: https://www.researchgate.net/figure/Haar-like-features-application_fig3_308836179. [Accessed 24 April 2024].

[7] T. Ojala, “A comparative study of texture measures with classification based on featured distributions,” *Pattern Recognition,* vol. 29, no. 1, pp. 51-59, 1996.

[8] K. Mittal, “A Gentle Introduction Into The Histogram Of Oriented Gradients,” Medium, 20 December 2020. [Online]. Available: https://medium.com/analytics-vidhya/a-gentle-introduction-into-the-histogram-of-oriented-gradients-fdee9ed8f2aa. [Accessed 25 April 2024].

[9] L. Piardi, “ResearchGate,” July 2019. [Online]. Available: https://www.researchgate.net/figure/Face-detection-using-the-HOG-algorithm_fig3_338941941. [Accessed 25 April 2024].

[10] R. Yamashita, “Convolutional neural networks: an overview and application in radiology,” *Insights into imaging,* vol. 9, pp. 611-629, 2018.

[11] J. Li, “Brief Introduction of Back Propagation (BP) Neural Network Algorithm and Its Improvement,” *Springer Link,* vol. 169, pp. 553-558, 2012.

[12] D. Pisner, “Support Vector Machine,” *Machine Learning,* pp. 101-121, 2020.

[13] S. Premanand, “The A-Z guide to Support Vector Machine,” Analytics Vidhya, 16 November 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/support-vector-machine-better-understanding/. [Accessed 26 April 2024].

[14] A. Christopher, “K-Nearest Neighbour,” Medium, 2 February 2021. [Online]. Available: https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4. [Accessed 26 April 2024].

[15] S. Misra, “Noninvasive fracture characterization based on the classification of sonic wave travel times,” *Machine Learning for Subsurface Characterization,* pp. 243-287, 2020.

[16] “Bootstrap Aggregation, Random Forests and Boosted Trees,” Quantstart, [Online]. Available: https://www.quantstart.com/articles/bootstrap-aggregation-random-forests-and-boosted-trees/#:~:text=techniques%20to%20work.-,The%20Bootstrap,with%20a%20machine%20learning%20model. [Accessed 27 April 2024].

[17] “Flask Documentation,” [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/. [Accessed 22 April 2024].

[18] “Pyramid,” [Online]. Available: https://trypyramid.com/. [Accessed 22 April 2024].

[19] “Django,” [Online]. Available: https://www.djangoproject.com/start/overview/. [Accessed 22 April 2024].

[20] “Django Channels,” [Online]. Available: https://channels.readthedocs.io/en/latest/. [Accessed 22 April 2024].

[21] TensorFlow, [Online]. Available: https://www.tensorflow.org/guide/keras/sequential_model. [Accessed 27 April 2024].

[22] S. Sharma, “Activation Functions in Neural Networks,” *International Journal of Engineering Applied Sciences and Technology,* vol. 4, no. 12, pp. 310-316, 2020.

[23] P. Sarmah, “Facial identification expression-based attendance monitoring and emotion detection—A deep CNN approach,” *Machine Learning for Biometrics,* pp. 155-176, 2022.

[24] X. Ying, “An Overview of Overfitting and its Solutions,” *Journal of Physics: Conference Series,* vol. 1168, no. 2, 2022.

[25] C. Enyinna, “Activation Functions: Comparison of Trends in Practice and Research for Deep Learning,” 2018.

[26] “Epochs, Batch Size, & Iterations,” 2020. [Online]. Available: https://machine-learning.paperspace.com/wiki/epoch. [Accessed 28 April 2024].

[27] “Emotion Detection,” 2021. [Online]. Available: https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer/data. [Accessed 5 April 2024].

[28] P. Runeson, “A Survey of Unit Testing Practices,” *IEEE Software,* 2006.

[29] “Integration Testing in 2024,” AIMultiple, 3 January 2024. [Online]. Available: https://research.aimultiple.com/integration-testing/. [Accessed 3 May 2024].

[30] intersoft consulting, “General Data Protection Regulation,” 27 April 2016. [Online]. Available: https://gdpr-info.eu/. [Accessed 16 May 2024].

[31] F. Campbell, H. Owen, J. Luke, A. Nyaupane, S. Singh and A. Keung, “CommuniTech Wall Commercial Viability Report,” 2024.

[32] “Sale of goods and services and data protection,” [Online]. Available: https://www.gov.uk/data-protection-register-notify-ico-personal-data. [Accessed 14 May 2024].

[33] M. Parker, “How much does website hosting cost? Average cost and pricing factors,” CNN, 15 May 2024. [Online]. Available: https://edition.cnn.com/cnn-underscored/money/website-hosting-cost#:~:text=On%20average%2C%20you%20can%20expect,volumes%20and%20enhanced%20security%20considerations. [Accessed 20 May 2024].

[34] TensorFlow, “TensorFlow Releases,” [Online]. Available: https://github.com/tensorflow/tensorflow/releases. [Accessed 15 March 2024].

[35] Python, “Python Releases,” [Online]. Available: https://www.python.org/download/releases/. [Accessed 4 April 2024].

[36] okdo, “The E-waste Crisis in the UK and How to Recycle: An Infographic,” [Online]. Available: https://www.okdo.com/blog/e-waste-crisis-in-the-uk-infographic/#:~:text=You%20can%20get%20a%20%C2%A3,renewed%20ready%20for%20another%20life. [Accessed 14 May 2024].

[37] “ReLU,” PyTorch, [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html. [Accessed 28 April 2024].

[38] S. Lexian, “Fall Detection System Based on Deep Learning and Image Processing in Cloud Environment,” 2019.

Appendix A – Design Brief

Brief:

The goal of this project is to create a smart wall that merges state-of-the-art interaction technologies with eco-friendly elements, aiming to enhance the community culture. With features like gesture control, mood analysis, solar energy, and plant health monitoring, this project demonstrates the tangible benefits of modern electrical engineering. As universities and companies continuously seek ways to cultivate cohesive and vibrant environments, the value of such a system could naturally capture wider interest, presenting opportunities for expansion and adaptability.

a. **Gesture Control and Interface Development:** This area will involve the integration of cameras or infrared sensors to recognize and interpret user gestures. The student responsible for this can delve deep into image processing, digital electronics hardware development, and application software development to create a gesture-based interface.

b. **Mood Analysis and Personalised Interaction:** Building on image processing capabilities, this area will use facial recognition techniques, combined with machine learning models, to assess a user's mood. Based on the detected mood, the wall could display encouraging messages, play uplifting music, or even adjust its ambient lighting.

c. **Solar Energy Harvesting and Management:** This area will involve the incorporation of solar panels and energy storage systems. The student in charge will focus on power electronics, electrical power systems, and possibly wireless power transfer. They will ensure the wall is energy-efficient and can run or be supplemented by solar energy, including managing power storage and distribution.

d. **Plant Health Monitoring and Proactive Care:** Using a combination of sensors (humidity, light, temperature, and perhaps even cameras), this domain will aim at ensuring the plants' wellbeing. The system will be capable of notifying when watering is needed, or if the plant is showing signs of distress, possibly even recognizing early symptoms of diseases or pest infestations. Embedded software development, control systems engineering, and possibly image processing (if cameras are used for plant health visuals) will be the key technical areas.

e. **Embedded Systems and Overall Integration:** This domain is crucial, as it ensures that all the other systems work in harmony. The student in charge will be responsible for integrating the various modules, ensuring data flow between them is smooth, and also working on the overall firmware of the wall. This would involve a lot of embedded software development, digital electronics hardware development, and potentially wireless communication (to integrate with other smart devices or the internet).

Appendix B – User Needs and Initial Target Specifications

Needs analysis		Target specifications			Technical solution
Ref Need	Description	Ref Spec	Description	Objectives measurable + tolerance	Possible solutions
N1	State-of-the-art User Interface	SP1.1	Must have reasonable response time to inputs	200ms-1s	Web app to display the data, conductive paint with microcontroller to detect touch inputs
		SP1.2	Should use modern UI frameworks/technologies	Modern JavaScript/CSS frameworks	Svelte, React, Vue
		SP1.3	Should have a modern and intuitive visual design		web app allows any type of design to be created
		SP1.4	Should be on and showing information whenever users are in the room/general area	System turns on whenever users are in the room	Motion/proximity sensor
N2	Novel User Interaction methods with user and environment	SP2.1	Must have gesture-controlled UI	System is able to be controlled by gestures	Gesture control using mediapipe/leapmotion/edge impulse
		SP2.2	Must have a backup UI control method	System has a secondary method of control available	conductive paint allowing touch control, small touchscreen with buttons on for accessiliblity, possibly a joystick or trackpad as a thrid option
		SP2.3	Should be able to respond to the mood or expression of a user	Must be able to react to at least 3 different mood states	Using face detection combined with emotions analysis ML model (deepface, OpenCV, OpenFace)
N3	Eco-friendly elements	SP3.1	Must have a sustainable method of powering/supplementing power to the wall	At least 50% power is from renewable sources	solar panels supplementing the power supply
		SP3.2	Should avoid using harmful materials and processes		
		SP3.3	Should have a low-power/sleep mode	Has a mode that uses < 40W	Turn off high power draw modules, keeping something displayed using e-paper
		SP3.4	Should be energy-efficient	Minimise energy lost as heat and sound	using modern devices, and services which only use lots of energy when actually in use
		SP3.5	Must be able to be turned off	Device has off mode	off button on the central compute unit, allowing the system to be turned off completely
N4	Adaptable to different locations	SP4.1	The system won't need to be operated outside	Warn user to not use product outside	warning on packaging/manual
		SP4.2	The system must be able to be wall-mounted self-standing	System supports wall and free-standing mounts	system mounted to a sheet of acrylic, or other suitably strong and cheap material which could then be mounted to a wall or floor stand
		SP4.3	The system should support varying sizes of display	supports standard monitor mounts	use VESA compatible monitor, or eliminate the req by use of a projector

N5	Cultivate a positive culture	SP5.1	The system should have a vibrant user interface	Bright colours and designs used	Bear conductive paint can be painted over with different colours, UI can be designed to be vibrant
		SP5.2	The system must be visually attractive and sleek	No exposed electrical components, any cables tidily hidden	Hide cables/electronics where possible: mounted behind the material sheet
		SP5.3	Features should be geared towards community engagement		
		Sp5.4	Features which promote multiple user interaction	Should be able be used by 3 people or less at the same time	Multiple forms of control, multiple areas that can be separately interacted with. Possibility of 2 or more player games
N6	Modular features to adapt to varying client needs	SP6.1	Should have additional features available at a cost	At least 3 distinct price levels	Subscription Price tiers depending on number of accessible features, upfront cost for additional equipment (paint, hardware required)
		SP6.2	Features should be self-contained and independant of other modules	Every module can operate with or without any other module present	
		SP6.3	Should have the physical and hardware capabilities for additional modules		Conductive paint and projector allows for infinite scalability
N7	Low Maintenance	SP7.1	Must be sufficiently physically durable	>10% under weight limit of all components	Apply safety factor relevant to mounting method
		SP7.2	Should be able to remotely update software	Software editable remotely	remote access server running on the system, which supports altering its codebase: SSH server, RDP server, or alternatively, have the code served/hosted from cloud
		SP7.3	Must be able to be hard rebooted	Has a reboot/restart option	Physical/paint restart button
		SP7.4	Software must be robust	Errors only shown through custom error screen	Design custom error screen, program should be robust & account for all edge cases
N8	Reasonable cost	SP8.1	Initial prototype must be within budget (£500)	prototype cost <£500	prototype cost <£500
		SP8.2	Must be priced competitively, according to available features	At least 3 distinct price levels	Subscription Price tiers depending on number of accessible features, upfront cost for additional equipment (paint, hardware required)
		SP8.3	Should be mainly built from COTS products	50% of products are COTS	Minimum 50% of products are COTS
		SP8.4	Should have an overall low power consumption	Peak power usage should not exceed 80W	Peak power usage should not exceed 80W
N9	Conform to packaging regulations	SP9.1	Must conform to electronic packaging standards	Comply with UKCA/CE packaging standards	

		SP9.2	Must conform to eletrical product disposal standards	Comply with WEEE standard https://www.gov.uk/electricalwaste-producer-supplier-responsibilities	ensure any purchased & created products all comply with WEEE (mandatory)
N10	Conform to all necessary safety standards	SP10.1	Must conform to safety standards	Comply with UKCA/CE safety standards	Comply with UKCA/CE safety standards
N11	Intuitive to use and accessible for users	SP11.1	Content must be displayed in an accessible way	Must comply with WCAG	Must comply with WCAG
		SP11.2	Should be able to onboard new users quickly	Time to access the wall and any information is under 30s	Learning time for new features <30s , should be tested
		SP11.3	Should have a quick and clear tutorial	Time to complete tutorial under 30 seconds	Design a simple tutorial for new users
N12	To sell to universities and corporations	SP12.1	Should have features marketed towards university and business communal spaces	marketing study	Survey to see if universities would buy such a product
N13	Receive information over the internet	SP13.1	Should be able to connect to internet servers	System features an internet connection	Use raspberrry Pi with wifi chip, or arduino with wifi chip, or mini pc, as the 'hub' for the system
N14	All data collected/displayed to comply with GDPR	SP14.1	Any personally identifying data should be encrypted in transit	All communications and storage of sensitive data is encrypted to high standard (TLS v1.3 etc.)	Comply with TLS v1.3 standard or other
		SP14.2	No personally identifying data should be stored in the system	No personally identifying data written to permanent storage, and any data held in variables/state wiped after exiting the feature it is used in	
		SP14.3	Any personally identifying data must only be used for the features and processes it is required for	No personally identifying data to be shared between features	
		SP14.4	The system must provide user agreements and consent option for users before collecting/using any personally identifying data	User agreement provided and must be accepted before collection of personally identifying data	physical notice of use of data displayed, or a tick box/form to be filled out before using those features
N15	Wirelessly connect to other devices	SP15.1	Should be able to connect to other IoT client devices	Should be able to connect to at least 3 IoT devices	Connect using main hub raspberry PI
		SP15.2	Could act as a hub for other IoT devices	Acts as a hub for IoT devices	Use main raspberry pi as a hub for IoT devices, software such as HomeAssistant, or simlply a communication hub like an MQTT broker
		SP15.3	Should be able to interface with various commonly used IoT protocols	Interfaces with various commonly used IoT protocols	Raspberry PI can interface with commonly used protocols, use ZigBee USB dongle, or Z-Wave hub software/device

Appendix C - Mood Analysis Sub-System Code

app.py

```
from flask import Flask, render_template, Response, jsonify
import cv2
import frame_setup
from flask_cors import CORS, cross_origin
import model

cam = cv2.VideoCapture(0)
app = Flask(__name__)
cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'
CORS(app, resources={r"/*": {"origins": ["172.26.79.99", "127.0.0.1"]}})
emotion = ''

def generate_frames():
    global emotion

    CNN, CNN_model = model_runtime()
    while True:
        success, frame = cam.read()
        setup = frame_setup.Frame(CNN, CNN_model)
        frame, emotion = setup.get_frame(frame)
        if not success:
            break
        else:
            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()

            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


def model_runtime():
    CNN = model.Model()
    CNN_model = CNN.build_model()

    return CNN, CNN_model

# FLASK FUNCTIONS

@app.route('/')
@cross_origin()
def index():
    return render_template('index.html')


@app.route('/current_emotion')
@cross_origin()
def current_emotion():
    return jsonify(emotion=emotion)


@app.route('/video_feed')
@cross_origin()
def video_feed():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')


if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=5001)
```

frame_setup.py

```
import haar_cascade
import cv2


class Frame:
    def __init__(self, CNN, CNN_model):
        self.haar_classifier = haar_cascade.HaarCascade()
        self.CNN = CNN
        self.CNN_model = CNN_model

    def get_frame(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = self.haar_classifier.detect_faces(grey_image=gray)

        prediction = "Null"

        for (x, y, w, h) in faces:
            prepped_img = self.CNN.model_preprocessing(gray, x, y, w, h)
            prediction = self.CNN.model_prediction(self.CNN_model, prepped_img)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 2,
cv2.LINE_8, 0)
            cv2.putText(frame, prediction, (x, y), cv2.FONT_ITALIC, 1,
                        (255, 255, 255), 2, cv2.LINE_8)

        return frame, prediction
```

haar_cascade.py

```
import cv2


class HaarCascade:
    def __init__(self):
        self.face_cascade = self.generate_haar_classifier()

    def generate_haar_classifier(self):
        face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
        return face_cascade

    def detect_faces(self, grey_image):
        faces = self.face_cascade.detectMultiScale(grey_image, scaleFactor=1.1,
minNeighbors=5, minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)
        return faces
```

model.py

```
import numpy as np
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D



class Model:
    def build_model(self):
        model = Sequential()
```

```
        model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48,
48, 1)))
        model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))
        model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(1024, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(4, activation='softmax'))
        model.load_weights('model_weights.h5')

        return model

    def model_preprocessing(self, grey_image, x, y, width, height):
        grey_face = grey_image[y:y + height, x:x + width]
        resized_img = np.expand_dims(np.expand_dims(cv2.resize(grey_face, (48, 48)), -
1), 0)
        return resized_img

    def model_prediction(self, model, resized_img):
        emotion_list = {0: "Happy", 1: "Sad", 2: "Neutral", 3: "Surprised"}
        prediction = model.predict(resized_img)
        max_index = int(np.argmax(prediction))
        return emotion_list[max_index]
```

train_model.py

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.optimizers.legacy import Adam
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils import plot_model

import plot_models

train_dir = 'data/train'
val_dir = 'data/test'

num_train = 28709
num_val = 7178
batch_size = 64
num_epoch = 50

datagen = ImageDataGenerator(rescale=1./255)

train_data = datagen.flow_from_directory(
        train_dir,
        target_size=(48, 48),
        batch_size=batch_size,
        color_mode="grayscale",
        class_mode='categorical')

val_data = datagen.flow_from_directory(
        val_dir,
        target_size=(48,48),
        batch_size=batch_size,
        color_mode="grayscale",
```

```
        class_mode='categorical')

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.0001,
decay=1e-6)
              , metrics=['accuracy'])

model.summary()

model_info = model.fit(
        train_data,
        steps_per_epoch=num_train//batch_size,
        epochs=num_epoch,
        validation_data=val_data,
        validation_steps=num_val//batch_size,
        verbose=1)


model.save_weights('model_weights.h5')
plot_models.plot_model(model_info.history)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Emotion Detection</title>
</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-lg-8  offset-lg-2">
                <h3 class="mt-5">Live Streaming</h3>
                <h4 id="emotion">Emotion: None</h4>
                <img src="{{ url_for('video_feed') }}" width="100%">
            </div>
        </div>
    </div>
    <script src="/static/emotion_fetcher.js"></script>
</body>
</html>
```

emotion_fetcher.js

```
function fetch_emotion() {
        fetch('/current_emotion')
            .then(response => response.json())
```

```
            .then(data => {
                document.getElementById('emotion').innerText = 'Emotion: ' +
data.emotion;
            })
            .catch(console.error);
}

setInterval(fetch_emotion, 100);
```

Appendix D - Unit Testing

Unit Testing Example: Image Processing and Emotion Prediction

Code

```python
from model import Model
import unittest
import numpy as np
import cv2


class TestImagePreprocessing(unittest.TestCase):
    def test_resize_image(self):
        model = Model()
        input_image = cv2.imread( filename: 'data/test/angry/im1.png', cv2.IMREAD_GRAYSCALE)
        resized_image = model.model_preprocessing(input_image, x: 0, y: 0, width: 48, height: 48)
        self.assertEqual(resized_image.shape, second: (1, 48, 48, 1))


class TestEmotionPrediction(unittest.TestCase):
    def test_emotion_prediction(self):
        model = Model().build_model()
        model.load_weights('model.h5')
        test_img = np.random.rand(1, 48, 48, 1)  # Simulated preprocessed image
        prediction = model.predict(test_img)
        self.assertTrue(np.sum(prediction) - 1.0 < 1e-5)  # Checks if output is a probability distribution

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

Results

```
============================= test session starts ==============================
collecting ... collected 2 items

testing.py::TestImagePreprocessing::test_resize_image
testing.py::TestEmotionPrediction::test_emotion_prediction PASSED             [ 50%]

============================== 2 passed in 6.40s ===============================
```

The results from unit testing provide concrete evidence about the reliability and correctness of individual modules within the mood analysis sub-system. For example, the **test_resize_image** ensures that the **model_pre-processing** function accurately resizes input images to the required dimensions. Similarly, **test_emotion_prediction** confirms that the emotion prediction output is a valid probability distribution. This selective approach to unit testing was systematically applied across the system to confirm that each function performs according to its specifications.

Appendix E - Integration Testing

Face Detection to Frame Setup

Test Case: A continuous video stream is input to test if the **HaarCascade** module works in detecting faces and updating the frame within the **Frame** module.

Expected Outcome: A window displayed showing the facial detection markers overlayed onto the video feed.

Integration Testing Example: Face Detection to Frame Setup

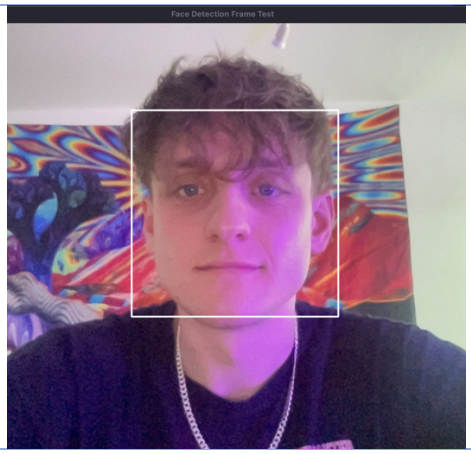
Code: Frame Setup

```python
import haar_cascade
import cv2


5 usages
class Frame:
    def __init__(self):
        self.haar_classifier = haar_cascade.HaarCascade()

    3 usages
    def get_frame(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = self.haar_classifier.detect_faces(grey_image=gray)

        prediction = "Null"

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 2, cv2.LINE_8, 0)

        return frame
```

Code: Test

```python
from frame_setup import Frame
import cv2

cam = cv2.VideoCapture(0)

while True:
    success, frame = cam.read()
    setup = Frame()
    updated_frame = setup.get_frame(frame)

    cv2.imshow( winname: 'Face Detection Frame Test', updated_frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cam.release()
cv2.destroyAllWindows()
```

Result



In this test case, the Frame module is specifically configured to include only the functionality for face detection, integrating solely with the HaarCascade module. The test involves streaming video from a webcam directly into the Frame module, where the Haar Cascade classifier is employed to identify faces within each frame. The results demonstrate that the expected outcomes have been achieved, confirming the effective integration of the HaarCascade module with the Frame module for face detection.

Emotion Recognition to Frame Setup

Test Case: A continuous video stream is input to test if the **Model** module for emotion recognition works in conjunction with **HaarCascade** for face detection and updating the frame within the **Frame** module.

Expected Outcome: A window displayed showing the facial detection and emotion recognition markers overlayed onto the video feed.

Integration Testing Example: Emotion Recognition to Frame Setup

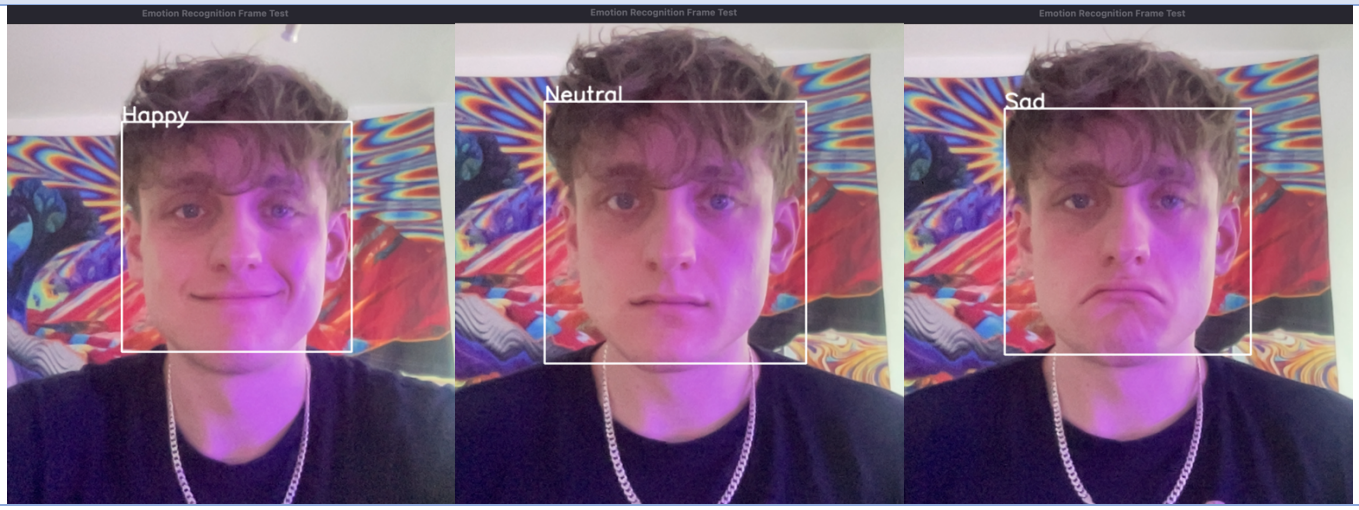
Code: Frame Setup

```
import haar_cascade
import cv2


3 usages
class Frame:
    def __init__(self, CNN, CNN_model):
        self.haar_classifier = haar_cascade.HaarCascade()
        self.CNN = CNN
        self.CNN_model = CNN_model

    2 usages
    def get_frame(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = self.haar_classifier.detect_faces(grey_image=gray)

        prediction = "Null"

        for (x, y, w, h) in faces:
            prepped_img = self.CNN.model_preprocessing(gray, x, y, w, h)
            prediction = self.CNN.model_prediction(self.CNN_model, prepped_img)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 255, 255), 2, cv2.LINE_8, 0)
            cv2.putText(frame, prediction, org: (x, y), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1,
                        color: (255, 255, 255), thickness: 2, cv2.LINE_AA)

        return frame, prediction
```

Code: Test

```python
from frame_setup import Frame
from model import Model
import cv2

cam = cv2.VideoCapture(0)
CNN = Model()
CNN_model = CNN.build_model()

while True:
    success, frame = cam.read()
    setup = Frame(CNN, CNN_model)
    updated_frame, emotion = setup.get_frame(frame)

    cv2.imshow( winname: 'Emotion Recognition Frame Test', updated_frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cam.release()
cv2.destroyAllWindows()
```

Result



In this test case, the code from the initial face detection test has been updated to include emotion analysis following the face detection step. The results provide three examples where the model successfully recognises various emotional states, and these identified emotions are then dynamically overlaid onto the video feed in real-time. The results confirm that the system effectively meets the expected outcomes, accurately recognising and displaying emotions as intended, ensuring seamless data interaction between three modules – **Frame, HaarCascade** and **Model**.