CS and AI
COC257
F126783

Development of Algorithms for Effective Human-Robot Interaction

by

Leo Shen

Supervisor: Dr Haibin Cai

Department of Computer Science
Loughborough University

May/June 2024

Abstract

Technology has rapidly been improving within the last decade. Standing at the forefront of this has been the field of AI and robotics and, thus, the realm of Human-Robot Interaction. As AI and digitization see unprecedented growth and integration within human society, the importance of developing a natural, smooth experience between people and robots is highlighted.

This project aims to facilitate synergistic communication between humans, robots, and in turn AI, specifically through integrating OpenAI's ChatGPT into LuxAI's QTrobot, creating an intuitive human-robot interaction between the two systems. This enables it to engage with users in real-time conversations by utilizing OpenAI's Natural Language Processing and QTrobot's sensors and movement to create an interactive, smoother, and natural-seeming conversation.

Acknowledgements

I would like to express my deepest gratitude to several individuals whose support was invaluable throughout this project. I would like to first thank my family, who have provided me with unwavering support and encouragement. I am also immensely grateful to my friends, their companionship and understanding, especially during challenging times, have help drive me. Finally, a special thanks goes to my supervisor, Dr Haibin Cai, whose expertise and guidance have been critical to my research.

Contents

List of Figures

List of Tables

Abbreviations

Term	Definition
AI	Artificial Intelligence
NLP	Natural Language Processing
PnP	Perspective-n-Point
P	Camera Intrinsic Matrix
3D	three dimensional
2D	two dimensional
HRI	Human Robot Interaction
CLT	Cognitive Load Theory
AET	Affective Events Theory
UES	User Engagement Scale
NP	Numerical Python
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
Adam	Adaptive Moment Estimation
FER-2013	Facial Emotion Recognition 2013
DNN	Deep Neural Networks
GPT	Generative Pre-trained Transformer
VAD	Voice Activity Detector
TTS	Text To Speech
RAF-DB	Real-world Affective Faces Database

Chapter 1

Introduction

AI research focused on rule-based systems to imitate intelligent behavior from its inception. However, these systems were limited by their inability to learn and adapt. AI overcame these limitations with the conception of machine learning, a branch of AI designed to learn and adapt, thus allowing a more dynamic and responsive form of AI. Machine learning can be further developed into deep learning, a concept inspired by the human brain of employing complex neural networks to learn from large amounts of data and adapt to it. Naturally, deep learning has been particularly successful in understanding, interpreting, and generating human language, also known as the field of natural language processing (NLP).

ChatGPT, a product of OpenAI, is a unique implementation of a deep learning algorithm. It utilizes a variant of the generative pre-trained transformer (GPT) architecture, specifically designed for generating and understanding human-like text. What sets ChatGPT apart is its ability to learn from a vast dataset of example languages deeply, allowing it to familiarize itself with the patterns and nuances of human language. This breakthrough in NLP enables ChatGPT to produce coherent and contextually relevant text based on the inputs it receives, creating an immersive conversational experience.

The field of robotics has witnessed rapid advancements in recent years, and I am privileged to work with one such innovation, the QTrobot. Developed by LuxAI, this humanoid robot is specifically designed to engage and interact with people in various settings, from educational to therapeutic environments. Equipped with a wide range of technologies, including cameras, microphones, speakers, and a screen, as well as motors for enhanced interaction,

the QTrobot holds immense potential for human-robot interaction.

Connecting ChatGPT with the QTrobot allows a person to communicate verbally with ChatGPT using the QTrobot's microphone. The robot can also reciprocate body language using motors and a screen to convey the tone of the conversation more clearly. This is a pivotal aspect of HRI: the ability to perceive, interpret, and reciprocate body language. This can be achieved by using ChatGPT's NLP to convey an emotion after each sentence and then displaying it using the screen or gestures. Additionally, to avoid miscommunication between the robot and a person, the robot needs to understand the direction a person faces, a core part of human body language. A solution to this is to utilize its camera to detect if the person is attempting to interact with it by detecting the direction the person's head is facing. This then allows the robot to engage more meaningfully by initiating interaction, responding to user commands, and avoiding accidental triggers.

1.1 Aims and Objectives

The main aim of this project is to create a seamless connection between the QTrobot and ChatGPT, enhancing human-robot interaction and allowing QTrobot to interpret and respond to human emotions and actions in real-time while also leveraging ChatGPT for natural language understanding and generating coherent responses.

- **Implement facial emotion recognition** using convolutional neural networks that will allow the QTrobot to perceive and understand human emotional states.
- **Integrate user engagement technology:** monitor the user's attention and change the robot's behavior to maximize the interaction experience.
- **Integrate the emotion and action recognition systems with ChatGPT** enabling the robot to produce and recognize the contextually relevant verbal responses based on the emotional and physical cues of the user.
- **Optimization and validation of the recognition algorithms**, testing them harshly with all kinds of datasets, making the performance robust in various scenarios and demographics of users.
- **User Experience Optimization:** fine-tune the robot's speech style, reaction time, and overall interaction quality based on feedback through testing in the real world, keeping the user engaged.

- **Real-time Interaction:** optimize for real-time communication between the user and the robot. The interactions are timely and fluent.
- **Contextual Understanding:** develop algorithms that allow the robot to consider the interaction context, making its responses relevant and appropriate within its context.
- **Gesture Detection:** Explore the possibility of incorporating more modes of communication, such as gestures, to accommodate different user preferences and needs.
- **Integration with QTrobot:** Find a way to integrate all the tools developed with ChatGPT into QTrobot meaningfully and comprehensively.

Chapter 2

Literature Review

2.1 Foundations of Artificial Intelligence and NLP

AI is the idea of developing algorithms and methods that mimic human intelligence, replicating how humans think and learn. In this section, the aim is to delve into the origins and evolution of AI, understanding both its theoretical parts and its modern applications. Over time, AI has evolved from rule-based systems to machine learning and deep learning. Exploring this offers a comprehensive understanding of the methods and algorithms that constitute AI's core.

As AI evolved from theoretical to practical applications, machine learning is an instrumental factor. Machine learning is a primary method for developing AI applications as it uses statistical techniques to learn and improve from experiences. One of the most prevalent machine learning methods is supervised learning, which involves training a model on a labeled dataset. Telling the model when it is wrong as the model is taught using a dataset that contains inputs paired with correct outputs, hence supervised. The goal is to learn a mapping from inputs to outputs that can predict the output for new inputs, similar to creating a formula for a line of best fit. Some examples of supervised learning are support vector machines and neural networks such as a Multi-Layer Perceptron. Supervised learning is used for image recognition and speech recognition, where pre-existing data can be extrapolated to form predictions.

On the other hand, while supervised learning has proven effective for tasks with clear tar-

get outcomes, unlike supervised learning, unsupervised learning allows AI to uncover hidden structures and patterns on its own, involving working with unlabeled data. In unsupervised learning, the model is expected to infer the natural structure within a set of data points. Typically, the idea is to find patterns or groupings within data; an example is the clustering of customers based on their purchasing behavior. Another way is to detect anomalies that deviate from the norm. Standard unsupervised learning techniques are k-means clustering and principal component analysis, which are used to reduce dimensionality.

In contrast to supervised and unsupervised learning, reinforcement learning, where an agent learns to behave in an environment by performing certain actions and receiving rewards or penalties in return, introduces an active learning process where decisions are based on feedback from the environment. The agent must discover the right and wrong actions through trial and error, guided by a reward system. This is ideal for applications such as robotics, games, and navigation, where the goal is to learn a policy for choosing actions that will maximize the sum of the rewards.

However, a vital component of the effectiveness of machine learning depends significantly on the quality and quantity of the available data. Where bias, imbalance, or insufficient variety of the data can severely impact model performance. In data with overrepresented features, bias can lead to poor generalization of real-world scenarios. In addition, models trained on imbalanced data could perform better on underrepresented classes. However, some tools to address include data augmentation and the use of synthetic data generation.[2]

The need to handle vast and complex datasets paved the way for deep learning, which uses multi-layered neural networks called deep neural networks (DNN). Strictly speaking, a neural network is considered a DNN when it has three or more layers. By utilizing multiple layers of neural networks, deep learning models can learn complex patterns in vast amounts of data, pushing the advancements in image and speech recognition and NLPs. Each layer in a DNN consists of interconnected nodes or neurons, and each layer is designed to perform specific transformations on its inputs. The three types of layers of a DNN that the data goes through are the initial input layer and, usually, multiple hidden layers before ending in an output layer. Each neuron in a layer is connected to several others in the next layer, and these connections/weights are adjusted during training to produce an output that, as closely as possible, matches the desired output.

On-type deep learning is Convolutional Neural Networks (CNNs). These neural networks are used primarily to process data with a grid-like structure, such as images, which are grids of pixels. CNNs use a mathematical operation called convolution, which involves sliding

a filter over the input to produce a feature map. This allows CNNs to be exceptionally good at picking up on features regardless of their spatial position in the image. Especially regarding image recognition, CNNs have become the standard for tasks ranging from facial recognition to medical image analysis.

Deep learning also extends into natural language processing, where DNNs have been adapted to handle text data, capturing contextual relationships and deeper meaning in text-solving tasks such as sentiment analysis, translation, and text summarization. Deep learning's ability to adapt to features without intervention allows them to perform more traditional models that rely heavily on human intervention.[10]

Natural Language Processing (NLP) is part of the interaction between computers and humans through language with AI. Aiming to enable computers to understand, interpret, and produce human language in a relatable way. The main challenge, however, is the complexity of the human language and its flexible rules, leading to its need to be clearer and contextual. The evolution of AI and the development of models like RNNs, LSTMs, and transformer architecture are great at capturing the context and multiple levels of meaning in the text. This allows a deeper machine understanding of a language's syntax, semantics, and pragmatics, advancing machine translation, text summarization, and the generation of human-sounding text.[37]

The evolution of AI from simple rule-based systems to the complexity of deep learning reflects a shift from merely understanding structured data to further understanding complex, unstructured data. Key milestones in this journey are machine learning and, subsequently, deep learning, which opens up a range of insights and capabilities, from image and speech recognition to natural language processing.

2.2 Evolution of Chatbots and GPT Models

Chatbots are programs devised to have conversations with human users by copying human conversation models naturally and intuitively. Chatbots are responsible for responding to a variety of queries that people have, ranging from the products and services they need to more complex transactions and support issues. However, over time, chatbots branched into being capable of learning and adapting from interactions, allowing them to become an integral part of enhancing user experience and streamlining operations. This section explores the evolution of chatbots, highlighting key developments to deepen understanding of the nature

of chatbots.

The first chatbots were created as rule-based systems and worked by scanning user inputs for keywords or fixed phrases and retrieving pre-defined responses from a script; this enabled them to manage simple dialogues or respond to specific user queries based on keyword recognition. Some early examples are ELIZA (developed by Joseph Weizenbaum in the mid-1960s), considered the first chatbot, and PARRY (developed by psychiatrist Kenneth Colby in the early 1970s), designed to simulate a patient with paranoid schizophrenia. Despite the innovations brought by these systems, a key issue was that they needed more ability to understand or generate language in a genuine sense and could not learn or adapt from conversations over time. The reliance on pattern-matching meant they struggled particularly with context and nuances within a conversation. Although, there is no denying that these chatbots paved the way for more sophisticated computational techniques.

As the limitations of rule-based systems became apparent, there was a shift towards more dynamic machine-learning techniques that could learn and evolve from user interaction. The advent of machine learning marked the departure from the static, rule-based systems of the past to more dynamic, learning-oriented models. Initially, chatbots began with integrating statistical models such as Hidden Markov Models (HMMs) that could analyze patterns in large datasets of conversational logs; this allowed them to generate new responses based on probabilities derived from past interactions. However, they still needed to be helped by the complexity of the human language as it is impossible to record and analyze all the possible legal compositions of words. This led to the introduction of neural networks, one of the most impactful advancements from machine learning; an example of a neural network was sequence-to-sequence (seq2seq) models, which enabled chatbots to process and generate human-like text by learning to map sequences of input data to sequences of output data, improving on subtleties of language that statistical models might have missed.[1]

Finally, building on the machine learning techniques, Generative Pre-trained Transformer (GPT) models were introduced, representing one of the most significant advancements in chatbots and natural language processing. The core technology behind these models was the adaptation of the transformers, which improved on previous sequence-based models like RNNs and LSTMs instead of processing all words or tokens in parallel while using self-attention mechanisms to maintain a contextual relationship between them. This permitted the model to weigh the importance of different words in a sentence, irrespective of their distance from each other in the text, saving computation time and giving chatGPT the capacity to capture long-distance dependencies within the text. This consequently solved the issue of earlier models only generating somewhat mechanical responses and enabled

GPT-based chatbots to create responses that can maintain the context over more extended conversations and adapt tone and style.

Inevitably, with GPT's introduction, the practical applications and use cases of chatbots expanded beyond theoretical advancements influencing chatbots in various sectors. This is most noticeable in the customer support sector, where chatbots have been revolutionary by providing support that is not only reactive but also proactive, offering relevant and personalized solutions and suggestions. In the field of HRI, by integrating GPT into robotic systems, developers can equip robots to engage in more natural dialogues with humans. Assisting in developing companion robots in healthcare settings and customer service robots in retail environment.[35]

This section demonstrates that the broader trend in AI is towards creating systems that not only simulate human conversation but also understand and respond to the nuances of human language in a meaningful way, reflected in the gradual development of chatbots from rudimentary, rule-based models into more sophisticated, contextually aware GPT models. Because of such evolution, they are trending towards revolutionizing Human-Robot Interaction for the better, making it more natural and intuitive, and the future of chatbots and AI looks toward pioneering new applications that can be seamlessly integrated into daily human activities.

2.3 Robotics and QTrobot

Robotics is a field of study that stems from a fusion of mechanical engineering, computer science, and electrical engineering, which is dedicated to creating, structuring, and operating robots. Robots are defined as self-governing devices meant to execute a range of functions traditionally undertaken by humans, notably those that are unsafe or those that require a lot of repetition and call for high precision. This section aims to research the technical components of robotic design and operation and discuss the critical applications of robots in society today.

Delving into the historical development of robotics, robotics has advanced from basic mechanized figures to today's intelligent systems. The concept of robotics can be traced back to the ancient Greeks, who developed steam-powered automated machines and the intricate automata of the medieval Islamic world, which included musical automata and mechanical servers. However, in the 20th century, what is recognized as modern-day robotics

began to take shape. The invention of Unimate in 1954 by George Devol was the first digitally operated and programmable robot and marked the start of modern robots. Following this came the microprocessors in the '70s, which became vital because they ensured robots had enough computational power for advanced calculations. Next, with the advancements in sensor technology, robots cancel their environments more accurately. And lastly, the integration of artificial intelligence in the late 20th and early 21st centuries saw the forming of autonomous robots that could learn from and adapt to their environments. So, AI-equipped robots could finally move past repetitive tasks and engage in complex decision-making processes. They were ushering in robots capable of navigating varied terrains, recognizing and manipulating objects, and interacting naturally with humans.[18]

Today, robots are highly specialized for a wide range of applications, and an example of this is the QTrobot (created by LuxAI), designed to make interactions as engaging and human-like as possible. The QTrobot is equipped with a screen acting as an expressive humanoid face capable of displaying various emotions. It also comes with pre-programmed gestures and expressions that can be used during interactions to convey feelings and reactions. These features are complemented by a high-quality speaker system, camera, and various sensors to help the robot perceive its environment. Due to these features, QTrobot has shown significant potential in interventions for children with autism spectrum disorders (ASD), as the robot's consistency and predictable interactions help ASD children, who may find human emotions and social cues challenging to interpret. QTrobot can perform repetitive social training exercises, providing a patient and controlled environment where children can learn social skills. The robot can also lead interactive educational games, tell stories, or even teach language and handwriting skills through its engaging interface and autonomous movement [7]

The development of robots like QTrobot stands as a testament to the positive impacts that thoughtful robotic design can have on improving human lives, particularly for those with special educational needs. It highlights how, as robotics advances, robots can be extended beyond industrial applications. It is imperative to note that their development should be guided by thoughtful consideration of ethical implications, including privacy, security, and socio-economic impact.

2.4 Evaluating User Engagement in HRI

User engagement regarding Human-Robot Interactions (HRI) means whether or not the user is paying attention (engaged with the robot); in other words, if the human is involved with the robot. Studying engagement is vital to HRI, not only for taking into account when designing and implementing the interfaces but also for allowing more complex interfaces capable of adapting to users to be developed[21]. This is particularly true when integrating chatGPT with an advanced robot like the QTrobot. When designing the integration of the two systems, the quality of a user's interaction and involvement with a system, encompassing aspects like attention, interest, and emotional participation, must be considered. Engagement in HRI is about the human interaction with the robot and the perception of the robot's responsiveness and intelligence.

Engagement can be split into many different types. Understanding these different types is instrumental in understanding the complex levels of user engagement, especially with a robotic system. These various types can be defined as cognitive engagement, emotional engagement, and behavioral engagement. Of course, there are more branches to be considered, but for simplicity, focusing on these three can provide the most efficient results[16].

Cognitive engagement refers to the active intellectual effort and attention a user might invest into something. This usually occurs when a user understands and processes key information. This will be important as when users interact with a robot, they are likely trying to understand and process the data the robot relays [27]. Understanding Cognitive Load Theory (CLT) can help further understand cognitive engagement; CLT identifies the difference between three types of cognitive load: intrinsic (meaning task complexity), extraneous (meaning the way the information is presented), and germane (meaning the process of learning)[25]. CLT theories state that high cognitive load levels can impede learning andand reduce engagement. This is because users try to spend more time understanding a system than interacting with the system. Considering CLT when designing the integration of chatGPT and the QTrobot means that it is crucial to design interactions that minimize cognitive load. To do this, the robot's engagement with the user must be simple, clear, and concise. Perhaps supplementing visual cues and emotions through the screen could assist in reducing cognitive load. Cognitive engagement can be measured by observing how users interact with the system, how long it takes a user to get a question across or complete an interaction, or through user feedback.

Emotional engagement indicates how a user feels about an interaction, i.e., whether

a user is feeling frustrated or mistrusting of an interaction. One theory that delves into understanding emotional engagement is the Affective Events Theory (AET); this is the understanding of how environmental events trigger emotional reactions, which in turn influence attitudes and behaviors[34]. In the context of HRI, AET is essential for understanding how interactions with the robot and chatGPT can cause certain responses; for example, if chatGPT does not understand a user's question, this can cause the user to be frustrated. These emotional reactions can influence the user's perception of the robot in the long term. The robot conveying any emotions or regarding any interaction should be context-sensitive; for example, the robot responding empathically can elicit positive emotional reactions. In summary, AET can help explain how users may develop emotional connotations to the robot; measuring this through user surveys can ensure the robot's emotional engagement is positive.

Behavioral engagement can be observed through how a user acts towards the robot. This can entail the frequency at which the user interacts with the robot and the nature of these interactions. Behavioral engagement can be measured through the User Engagement Scale (UES) based on observable behaviors and subjective experiences. UES typically factors in things such as focused attention, usability, aesthetic appeal, and reward. As mentioned before, behavioral engagement can be measured by interaction frequency, the user's responsiveness to the robot's cues, and feedback. UES can help determine specific behaviors that indicate engagement and can be adapted to observe how the AI's responses influence user behavior[3].

Detecting user engagement in HRI can be done through various methods, especially by,by utilizing QTrobot's sophisticated sensors. The main way to observe engagement is through behavioral indicators, facial expressions, and verbal interaction patterns, and observing where the user is facing when interacting with the robot, An example of this could be sustained focus and attention on the QTrobot, which can indicate high levels of user engagement. Surveys and feedback can be used to gather user feedback on their engagement level; this can even be automated so that QTrobot can ask questions after interactions regarding the reaction and record the response. Combining these can give a comprehensive insight into user engagement.

Chapter 3

Methodology

3.1 Detecting User Engagement

The idea of detecting user engagement is to take an image of a person and, using their head pose and facial features, calculate whether or not the person is facing the camera. This can be extrapolated by applying the same method frame per frame to work out user engagement in real time. One of the main difficulties with calculating user engagement is perspective. This is commonly known as the Perspective-n-Point problem (PnP). PnP is the problem of estimating the relative pose between an object, in this case, a person's face and a calibrated camera. Solving this problem returns a rotation, and a translation that provides minimal re-projection error from the corresponding three-dimensional (3D) points onto the two-dimensional (2D) points [26]. In other words, the objective is to figure out how to map a two-dimensional perspective the camera gives onto a three-dimensional perspective of the world and vice versa. The PnP can be mathematically defined as

$$p_i \sim K[R \mid t]P_i$$

- Where P_i is a set of 3D points in the world coordinate system.
- p_i is the corresponding 2D projections of these 3D points on the camera's image plane.
- K is the camera's intrinsic matrix.
- The goal is to find the rotation matrix R and translation vector t that describe the camera's pose.[38]

The first thing to understand is the relationship between the 2D and 3D perspectives and the role a camera plays. A camera functionally maps the 3D world onto a 2D image, which can be mathematically understood using the pinhole camera model shown by figure 2. The pinhole model assumes a single point (the pinhole) through which light from the scene passes to form an image on the camera sensor. Using figure 2, the problem proposed is: what is the equation using the image coordinate x_i in terms of X_i[17]?

Simplifying this problem and looking at the image from a side-on perspective as shown in figure 3,f represents the camera's focal length, the distance between the pinhole (or lens) and the image sensor. The focal length indicates how much an image is magnified when projected onto the sensor. If the focal length is provided, using the relationship between similar triangles, the height of the smaller triangle can be calculated as $\frac{fY}{Z}$ shown in figure 3. This can be repeated from a top-down perspective to get $\frac{fX}{Z}$. Combining these two perspectives creates the coordinates of a point in a 3D space relative to the camera $\begin{bmatrix} XYZ \end{bmatrix}^T \mapsto \begin{bmatrix} \frac{fX}{Z} \frac{fY}{Z} \end{bmatrix}^T$.

To map $\begin{bmatrix} XYZ \end{bmatrix}^T$ on to $\begin{bmatrix} \frac{fX}{Z} \frac{fY}{Z} \end{bmatrix}^T$, a Camera Intrinsic Matrix, commonly denoted as P, is needed. P mathematically represents a pinhole camera model in a matrix form. The projection from 3D to 2D

$$P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The focal length in both the x and y planes indicating the scaling of the 3D world onto the 2D image plane in both the horizontal and vertical directions. Before using P, the 3D coordinates, $\begin{bmatrix} XYZ \end{bmatrix}^T$, need to be converted a homogeneous coordinates becoming $\begin{bmatrix} XYZ\,1 \end{bmatrix}^T$ to enable matrix multiplication. After performing the matrix multiplication:

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix}$$

Since the result returned is 3 component vector to achieve $\begin{bmatrix} \frac{fX}{Z} \frac{fY}{Z} \end{bmatrix}^T$ a perspective division can be performed. This means dividing the first two components, fX and fY, by the third Z. Providing $\begin{bmatrix} \frac{fX}{Z} \frac{fY}{Z} \end{bmatrix}^T$.

However, the P provided above does not consider that the image origin and camera origin might differ. In a practical scenario, the origin of the image coordinate system might

not align with the center of the image. To account for this, the principle point needs to be considered; the principle point is the point at which the camera's optical axis intersects the image plane. In figure 2, the point at the end of the dotted line. To adjust for the shifting of the origin P needs to be rewritten as:

$$P = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Where (p_x,p_y) are the principle point coordinates. Adding these principal point coordinates allows for a more accurate mapping of 3D world points onto the 2D image plane. This adjustment accounts for when the camera's coordinate system does not naturally align with the world coordinate system and allows for the projection to adjust regardless of the differences in the origins and orientations of these coordinate systems.

P can be decomposed into two matrices:

$$P = K[I \mid 0]$$

This represents a particular case where I is the identity matrix to indicate no camera rotation relative to the world of coordinates. The zero vector represents no translation, meaning the camera is at the origin of the world coordinate system [17].

$$P = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Now that K is defined mathematically, the focal point and the principle point need to be obtained to define K practically. Typically, the principle point can be estimated by calculating the midpoint of the image dimensions, assuming the optical axis is centered in the image. The focal length can be calculated using the formula

$$f = \frac{sensor\,width/2}{\tan(\frac{FoV}{2})}$$

Where FoV is the field of view in radians, the field of view is the extent of the observable world that is seen at any given moment. In a camera, it is the part of the world that fits into the image frame. The formula is derived from trigonometry, using the tangent of an angle in a right triangle, which is the ratio of the opposite side to the adjacent side. Since the tangent function can be arranged to create the focal length formula[4].

$$\tan(\frac{FoV}{2}) = \frac{w/2}{f}$$

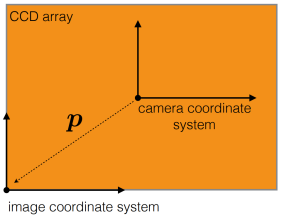

Figure 1: The misalignment of the origin between the Image Coordinate System and the Camera Coordinate System, where p is the translation vector to align the two systems.

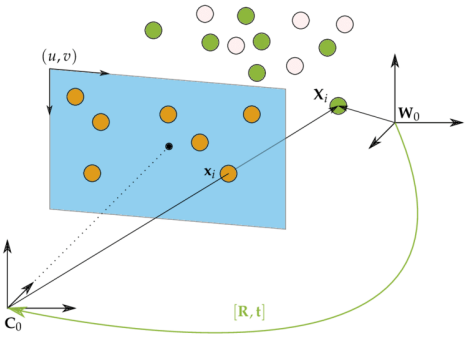

Figure 2: The Perspective-n-Point problem is the issue of estimating an image pose given a set of 3D points of a scene and their corresponding 2D points. The image pose of a camera with center C 0 is given by a rotation matrix R and translation vector t concerning a global reference frame W 0 . The image pose is computed from corresponding 2D points (orange) and 3D points (green)[20].

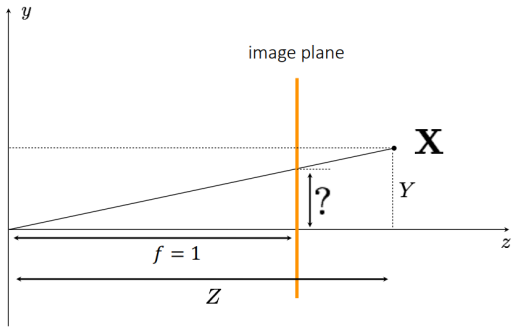

Figure 3: Shows the pinhole camera model from a side perspective. [17]

Now that the focal length and the principle points are obtained, to solve the PnP problem, two more variables to account for, P_i (the set of 3D points in the world coordinate system) and p_i (the corresponding 2D projections of these 3D points on the camera's image plane). DetectThe set of 3D points would be a face model for detecting user engagement. All that is necessary when calculating a user's engagement is understanding where the face is facing. So, to create a 3d coordinate system model of a face, essential features or facial landmarks need to be taken into account. To track how a head is facing, only six landmarks are necessary: the nose tip, chin, left eye corner, right eye corner, left mouth corner, and right mouth corner. To create the 3D model, the nose tip can be used as the origin since it is prominent and easily recognizable. The rest of the point can then be estimated relative to the nose. For example, the chin point can be set below the nose (in the negative Y direction), and a negative Z value sets the chin slightly back from the nose tip along the depth axis. A table, 1, can be created for the coordinate system [8].

Feature	X	Y	Z
Nose tip	0	0	0
Chin	0	-330	-65
Left eye corner	-225	170	-135
Right eye corner	225	170	-135
Left mouth corner	-150	-150	-125
Left mouth corner	150	-150	-125

Table 1: 3D model coordinates of a face

Finally, the last variable needed is the corresponding 2D projections of these 3D points from the image. To do this, the landmark features must be mapped onto the face in the image, and then the relevant coordinates must be retrieved. Once all these variables are obtained, plugging in these variables the following is obtained[23]:

$$p_i = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} [R \mid t] P_i$$

Where f, c_x, c_y (p_x, p_y were changed to c for clarity), p_i and P_i are given. This can then be used to calculate the rotation matrix and the translation vector.

After working out both the rotation matrix and the translation vector, the translation vector can be ignored as when determining if a person is facing the camera, the orientation of the head (which is provided by the rotation vector) is all that is necessary since the translation vector regards the position in space. However, the problem arises that the rotation matrix returned provides angles in an arbitrary coordinate system. These angles must be decomposed to Euler angles to obtain rotation angles along the three principal axes (x, y, and z). The rotation matrix written out is [23]

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Using this, the Euler angles can be calculated with the following formulas [22]:

- Yaw (rotation around the z-axis): $\theta = \text{arctan2}(r_{21}, r_{11})$
- Pitch (rotation around the y-axis): $\psi = \text{arctan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2})$
- roll (rotation around the x-axis): $\phi = \text{arctan2}(r_{32}, r_{33})$

Looking at figure 4 can help further understand Euler angles. Here, yaw represents the head rotating around the y-axis (the head moving from side to side), pitch represents rotating around the x-axis (the head nodding, i.e., the up and down motion), and roll represents rotating around the z-axis (head tilt) [6]. Noticing from 4 roll is not necessary as head tilt does not indicate that the user is not engaged. Once the Euler angles are calculated, it simply dictates the threshold at which the angle should be to where it is considered that the user is facing the camera and thus engaged.

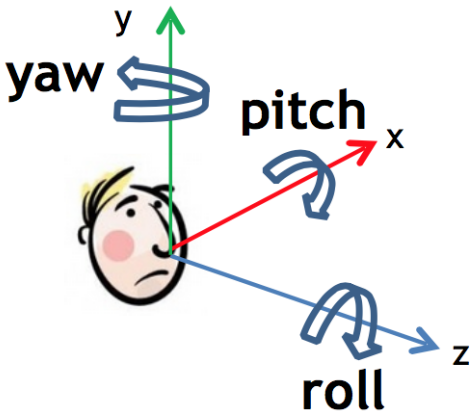

Figure 4: Euler angles defined in the head coordinate system

3.2 Emotion detection

Emotion detection usually involves analyzing facial expressions to determine a range of emotions such as happiness, sadness, anger, fear, disgust, and surprise. In this project's context, computer vision and neural networks are used to achieve facial emotion recognition and allow the robot to react more emotionally intelligently in real time. This can be achieved by building on the foundation of image processing technologies and using a convolutional neural network (CNN) model tailored explicitly for emotion recognition.

A CNN class of deep learning neural networks analyzes visual imagery. The idea of CNN is to learn spatial hierarchies of features through backpropagation adaptively. This is done using convolutional layers, pooling, and fully connected layers. The convolutional layer applies filters over the image, which create feature maps; these filters are small matrices of weights that slide over the image and are designed to detect specific features, such as edges and textures at various locations. The filter computes the dot product of the filter values and the input values under the window, transforming the data into a form the neural network can use to classify the image. Combining these filters aggregates these basic features into more complex representations, allowing the CNN to learn from the data and adapt to high levels of abstraction and complexity in the data.

Typical parameters of the convolution layer are the filter size (typically three by 3 or 5 by 5) and the stride length, which refers to the number of pixels by which a filter moves across the input image during the convolution process. These parameters influence how the input

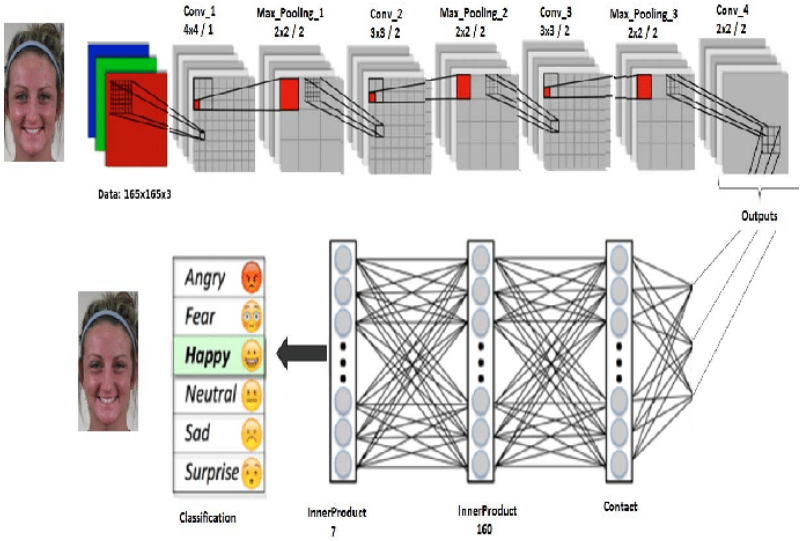

Figure 5: Architecture of the convolutional neural network.[9]

image is transformed into feature maps. These feature maps are a condensed version of the input image where the desired features are emphasized. For example, a filter designed to detect edges would activate strongly in areas of the image where edges are present. Another element of convolution layers to consider is padding, typically applied before convolution; padding involves adding layers of zeros around the image borders, allowing the filter to fit entirely on the input image, even at the borders, ensuring that features at the edges are not missed and maintaining the size of the feature map. This ensures that information at the edges of the image is not lost in the convolution process. After convolution, the feature maps are typically passed through a non-linear activation function like ReLU (Rectified Linear Unit). This introduces non-linear properties into the network; the activation function enables the neural network to solve non-linear problems. Non-linear properties mean the output is not a direct, proportional, or linear response to the input. This introduction helps the model combine the feature maps generated by each convolution layer in an increasingly complex way, allowing the neural network to learn and model more complex patterns. Combining multiple convolutional and activation layers enables the network to deeply understand the visual content, leading to the effective classification of facial expressions.[5]

In deep learning models, particularly in complex architectures like CNNs, batch normalization is crucial in improving the training process and achieving higher performance. Batch Normalisation involves normalizing the inputs of each layer within a network to have a mean of zero and a standard deviation of one. This addresses the issue where the distribution of

each layer's inputs changes during training as the parameters of the previous layers change, which can cause the gradients to explode or vanish. This normalization allows for higher learning rates, accelerating the network's convergence during training. Additionally, batch normalization can also act as a regulariser, and using batch normalization often results in improved training speed and performance and more stability. Typically, batch normalization is placed after convolutional layers, which helps to maintain the optimal range of values.

Another layer to understand is max pooling; this type of pooling operation downsamples the input by applying a max filter on distinct sections of the feature map that do not share any pixels defined by pool size. A typical pool size is two by two, which means that for a pool size of (2, 2), the operation selects the maximum value from each two by two blocks of the feature map. By doing this, max pooling reduces the size of the feature map because, from each 2x2 block, only one value is chosen, and the rest are ignored, effectively shrinking the feature map's dimensions. This reduces the number of parameters and computations in the network, enhancing the network's ability to generalize from the training data. Adding a max pooling layer after convolutional layers in the neural network effectively summarizes the most prominent features while discarding irrelevant data.

Integrating regularisation, such as dropout and Gaussian noise, prevents overfitting. Dropout is done by randomly setting a proportion of input units to 0 at each update during training time. Randomly eliminating a fraction of the units helps break up patterns that are not representative of the general data. Forcing CNN not to rely on specific features to make predictions helps the model generalize the data better. A typical dropout size starts small, for example, 25%

After forming the structure of each convolution block, the next step is to connect all the layers; this is accomplished by flattening the output and then adding fully connected layers. Flattening the layers involves converting the multidimensional output of the previous layers into a single, long, continuous linear vector. This is important as the input to the fully connected layers is a one-dimensional array. The role of fully connected layers is significant as it allows the network to learn deep representations of the data, integrating all the features extracted by the convolutional and pooling layers. This is because the layer consists of a high number of neurons at first and then gradually narrows down to a layer with seven neurons, corresponding to the number of classes to predict. Usually, in between each layer, a drop-out is added to prevent overfitting. The activation function softmax, which takes a vector of real numbers as input and normalizes it into a probability distribution, allows the model to give a predicted class based on which output neuron has the highest probability.

The next step is the optimizer, the learning rate, and the loss function. A famous optimizer is an Adam (Adaptive Moment Estimation) optimizer, which maintains separate learning rates for each parameter by calculating the first moments (the mean of the gradients) and the second moments (the uncentered variance of the gradients) of the gradients. This combines the advantages of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). The learning rate dictates the step size at each iteration while moving toward a minimum loss function. Adam helps manage the learning rates, making them responsive to the specific behaviors of the data and the model. Finally, the loss function measures the disparity between the predicted probabilities and the actual class labels, aiming to minimize this gap during training. When classifying, a widely used loss function is categorical cross-entropy, which measures the distance between the model's predicted probabilities assigned to each class and the proper distribution, which is the one-hot encoded vector of the class labels.

The last thing to discuss when building a CNN is callbacks, which are used to view the internal states and statistics of the model during training and help manage and optimize this process dynamically. The three callbacks to consider are model checkpointing, early stopping, and reducing the learning rate on a plateau. Model checkpointing is used to save the model at different stages during the training by measuring the validation loss during training and saving the model when the validation loss has improved. When used with early stopping, this helps combat overfitting as early stopping stops the training prematurely if the validation loss stops improving after a certain amount of epochs, giving the model some leeway to overcome local minima in optimization. The last callback reduces the learning rate by a set amount when the validation loss has stopped improving after several epochs, fine-tuning the model when it is close to its optimal configuration.[15]

It is equally critical to consider how the performance of CNNs is quantified, evaluating the effectiveness and accuracy of a model in classifying data. Below is an explanation of the purpose of several key metrics, such as accuracy, Recall, precision, F1 score, and confusion matrix, which are commonly utilized to provide a comprehensive assessment of a model's performance:

- **Accuracy**: This is a basic performance measure that is simply a percentage of all correctly predicted results over all predicted results.
- **Precision**: Also known as a positive predictive value, measures the accuracy of positive predictions by using the percentage of correct predictions of a class to the correct predictions summed with the wrong prediction of different classes that the model la-

beled as that class. It is important when the costs of False Positives are high.

- **Recall**: This measures the ability of a model to find all the relevant cases within a dataset and is the percentage of correct predictions of a class over the correct predictions and the wrong predictions of that class that the model labeled as a different class. This is crucial in situations where Missing a Positive (a true case) is costlier than getting a False Positive.
- **F1 Score**: This is the weighted average of Precision and Recall, maintaining a balance between Precision and Recall.
- **Confusion Matrix**: Provides a matrix of accurate labels against predicted labels as output and gives insight into the errors being made by a classifier and the types of errors being made.

3.3 Integration with ChatGPT

With emotion detection and engagement tracking, the next step is to merge and feed this data into ChatGPT. This allows ChatGPT to understand and respond to the user's emotional states and engagement levels dynamically, enhancing the interaction quality and overall user experience. The first component to create a seamless engagement with ChatGPT is to use speech recognition and Text-to-Speech (TTS) to imitate a real conversation.

Speech recognition involves processing human speech into a usable text format, and some common everyday applications are voice-activated assistants and real-time communication translation. Speech recognition involves several key steps. First, noise is processed to filter out and adjust the signal for optimal clarity. The processed signal is then analyzed to extract meaningful features that represent phonetic components; the next step is to use machine learning techniques like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) trained on vast datasets of spoken language to learn variances in pronunciation, accent, and dialect to extract features further. After recognizing the basic sounds, the technology utilizes language models to predict and form coherent sentences; these interpreted words are then output as written text.[36]

On the other hand, TTS focuses on transforming a written text into spoken words, enabling a more natural and accessible way to communicate with users.TTS involves preprocessing and normalizing the text to standardize various forms of words to a common base

form and segmenting the text into manageable parts, such as sentences and words. Then finally, actually synthesising speech can be done in three ways:

- Concatenative TTS: The system draws from a database of prerecorded speech segments and joins them to form the final speech output. This can yield high-quality results but requires a large database.
- Parametric TTS: This contrasts Concatenative TTS by synthesizing the sound waves directly from the text, requiring less storage.
- Neural TTS: An approach that uses deep neural networks to generate speech

When the user's speech is successfully extracted, the following challenge is to implement this in real-time. A Voice Activity Detector (VAD) can be used to monitor the audio stream and detect the presence of speech. Audio frames detected by VAD believed to contain speech are stored immediately, while non-speech frames are temporarily stored in a buffer. Buffering helps retain context by preserving a few segments of audio before the speech begins, ensuring that the beginning of spoken sentences is not clipped off. Using a VAD helps the program identify what speech is so thatspeech is so the algorithm can begin speech recognition when there is no detected speech,.[14]

Building upon real-time speech recognition, emotion detection, and engagement detection, the next step is defining the conversation logic provided to chatGPT and utilising the insights gained from the user's verbal and non-verbal cues to shape the flow of the conversation and the system's responses. An example of this is if engagement detection indicates that the user seems disinterested, by looking away from the screen, the conversation logic can trigger, and a prompt can be fed to chatGPT to trigger a change in topic or to get chatGPT to ask a question to re-engage the user. On the other hand, if the system detects signs of sadness or frustration in facial expressions, ChatGPT can be prompted to modify its responses to be more empathetic or supportive. An example reply to sadness might be prompting chatGPT to ask what is wrong and whether it could be of any assistance.

Chapter 4

Design and Implementation

4.1 Detecting User Engagement

When implementing the user engagement detection feature, what libraries are necessary need to be considered; libraries are a great tool that offers pre-written functions to allow programmers to focus on the intricacy of the software instead of re-writing code. Regarding detecting user engagement, since it requires camera use, the library OpenCV is a natural and intuitive choice. OpenCV is a computer vision library that focuses on real-time applications; this makes it perfect for detecting user engagement as it has various functions on 3D reconstruction[29][22]. Another valuable library to consider is the Google library MediaPipe; MediaPipe offers comprehensive face mesh and face landmark detection functions [12].

Before coding anything, referring back to the methodology, first, the camera intrinsic matrix (P) is needed, and to create P, both the focal length and the principle point are necessary. The principle point can be assumed to be the center of the image, so it can be calculated by taking half of the image's width and height. In Python code, this can be written as:

```
principle_point = (image_shape[1]/2, image_shape[0]/2)
```

To get the focal length using the formula from the methodology as code, this can written as:

```
focal_length = principal_point[0] / np.tan(60 / 2 * np.pi / 180)
```

The image width can be assumed as the sensor width, and so the first element of the principle point is used to create the numerator. To fill the denominator, the field of view angle needs to be defined. A typical field of view angle used is 60 degrees, as it represents a narrow-angle lens generally used by most cameras. This field of view angle is then converted to radians.

With both the focal length and the principle point, the camera intrinsic matrix can be formed as a numerical Python (NP) array following the methodology from above. The code is as follows:

```
camera_matrix = np.array(
    [[focal_length, 0, principal_point[0]],
     [0, focal_length, principal_point[1]],
     [0, 0, 1]], dtype = "double"
)
```

Where the data type is specified as a double.

Next, to solve the PnP problem, a model face needs to be created so that the image face can be mapped later. As the 3D coordinate model face was already defined in methodology, implementing is a simple process of defining an NP array to hold the coordinates:

```
model_points = np.array([
    (0.0, 0.0, 0.0),          # nose tip
    (0.0, -330.0, -65.0),     # chin
    (-225.0, 170.0, -135.0),  # left eye left corner
    (225.0, 170.0, -135.0),   # right eye right corner
    (-150.0, -150.0, -125.0), # left Mouth corner
    (150.0, -150.0, -125.0)   # right mouth corner
])
```

With this model, all that is needed now is to obtain the points from the 2D image provided by the camera to be mapped. To do this, google's media pipe provides a face landmark tool to detect facial landmarks. The necessary coordinates can be obtained using a media pipe to create a mesh on the user's face, as shown in the image in 1. Since the landmark points are expressed as a decimal between 0 and 1, as the landmark's relative

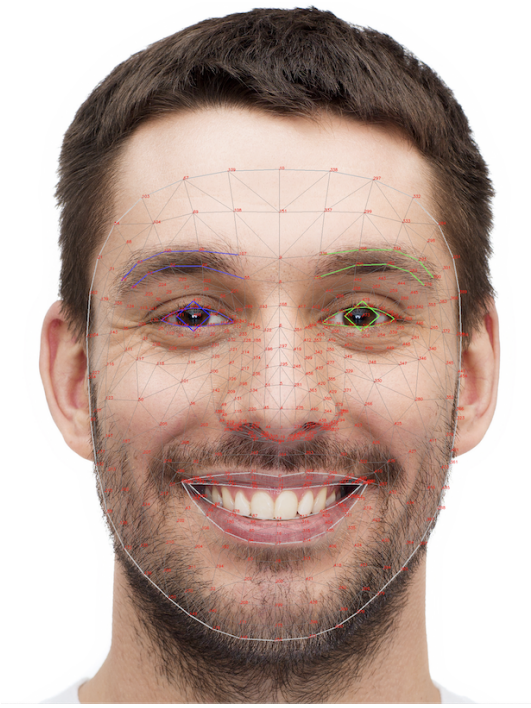

Figure 1: An example face mesh with coordinates at each landmark provided by media pipe[12]

position is in proportion to the image dimensions, the landmarks need to be scaled to the image's actual dimensions in pixels by multiplying the normalized coordinates by the image's width and height.

```
# Using media pipe face gets the point corresponding to the model face
image_points = np.array([
    (landmarks[1].x * image_shape[1], landmarks[1].y * image_shape[0]), # nose
        tip
    (landmarks[152].x * image_shape[1], landmarks[152].y * image_shape[0]), #
        chin
    (landmarks[226].x * image_shape[1], landmarks[226].y * image_shape[0]), #
        left eye left corner
    (landmarks[446].x * image_shape[1], landmarks[446].y * image_shape[0]), #
        right eye right corner
    (landmarks[57].x * image_shape[1], landmarks[57].y * image_shape[0]), # left
        Mouth corner
    (landmarks[287].x * image_shape[1], landmarks[287].y * image_shape[0]) #
        right mouth corner
```

```
], dtype="double")
```

With this, assuming there is no lens distortion by passing in a vector 0s, the openCV'sOpenCV'sP function returns a rotation and a translation vector. Only the translation vector is needed since only the head pose is calculated. However, the rotation vector is defined arbitrarily and needs to be converted to obtain meaningful rotation angles. To do this, openCV provides another function that converts a rotation vector to a rotation matrix and then to Euler angles that return a pitch, yaw, and roll. Since, as mentioned in the methodology, the roll is not necessary, all that is needed is to define thresholds for the pitch and yaw, which determines whether or not the user is looking at the camera.

The next step is to integrate with real-time; this is a simple process of creating a try and attempt to catch a keyboard interrupt to terminate the process and then using a while loop to catch frame by frame from the camera. These frames are then fed into the function, which determines the head position and returns the pitch and the yaw. This value can then be evaluated to measure if the user is looking at the camera.

4.2 Emotion detection

In the design and implementation phase of the emotion detection system, the first step is identifying which library to use to help construct the CNN. The framework selected was TensorFlow (a popular library developed by Google and released in 2015) as it is designed for both production and research, offers high levels of flexibility, and allows for easy model iteration and rapid prototyping[24]. Before constructing the CNN, a dataset is required, and the dataset selected for the project is the Facial Emotion Recognition 2013 (FER-2013) dataset found on a popular dataset website, Kaggle. The FER-2013 dataset consists of 32,298 processed grayscale images of faces, each labeled with one of seven emotion categories: anger, disgust, fear, happiness, sadness, surprise, and neutral. Using the FER-2013 dataset provides a representative sample of human emotions, allowing the model to better generalize across real-world scenarios.[28]

Although the FER-2013 dataset does provide a good range of emotions, after visualizing and graphing the dataset, the use of Matplotlib and Seaborn revealed that there was a significant disparity between disgust and the rest of the classes across both training and testing, as shown in table 1. This poses the risk that the model will not learn to recognize 'disgust' as effectively as more frequently represented emotions. This is vital when evaluating

Emotion Class	Total Amount of Images
Anger	4953
Disgust	547
Fear	5121
Happy	8989
Neutral	6198
Sad	6077
Surprise	4002

Table 1: The amount of images for each class in the training/validation and testing datasets

Figure 2: Some random sample images from the training data

the model to understand why it performs the way it does. However, visualizing the images shows various expressions, confirming that the labeled emotions correspond correctly to the expressions displayed. It also shows that all the images are close-up images of the face, which is vital to remember when integrating the model with real-time.

The first step in constructing a CNN is to use a combination of OpenCV and numpy to begin, preprocess the data, transform, and format the images to meet the requirements for the CNN to adapt. Preprocessing involves normalizing the data and ensuring that all the images are the same size and are all grayscaled; this ensures that the CNN does not learn from unnecessary features such as color and image size as these do not affect the emotion's class and helps to achieve better convergence while learning. After preprocessing, the data is initially encoded so that each image corresponds to its label and then is split into training, validation, and testing sets.

4.2.1 First Implementation

The CNN is built as a sequential model, meaning a linear stack of layers that makes a multi-layer neural network, where each layer has exactly one input tensor and one output tensor. The model consists of three convolution blocks, each consisting of a convolution layer, a

batch normalization, another convolution layer, and a max pooling of 2 by a drop-out layer of 25%. After the three blocks, the model is flattened, and the layers are fully connected with a drop out of 50%; each convolution layer has a stride of one and a three-by-three filter as well, as padding is set to the same, which adds just enough padding to ensure that the output feature map has the exact dimensions as the input image when the stride is 1. Both the convolution and the dense layers are activated by ReLU, as mentioned in the methodology, except for the last dense layer, which is activated by softmax to return a prediction.

Layer (type)	Output Shape	Number of Parameters
conv2d	(None, 48, 48, 64)	640
batch normalization	(None, 48, 48, 64)	256
conv2d 1	(None, 48, 48, 64)	36928
max pooling2d	(None, 24, 24, 64)	0
dropout (0.25)	(None, 24, 24, 64)	0
conv2d 2	(None, 24, 24, 128)	73856
batch normalization 1	(None, 24, 24, 128)	512
conv2d 3	(None, 24, 24, 128)	147584
max pooling2d 1	(None, 12, 12, 128)	0
dropout 1 (0.25)	(None, 12, 12, 128)	0
conv2d 4	(None, 12, 12, 256)	295168
batch normalization 2	(None, 12, 12, 256)	1024
conv2d 5	(None, 12, 12, 256)	590080
max pooling2d 2	(None, 6, 6, 256)	0
dropout 2 (0.25)	(None, 6, 6, 256)	0
flatten	(None, 9216)	0
dense	(None, 1024)	9438208
dropout 3 (0.5)	(None, 1024)	0
dense 1	(None, 512)	524800
dropout 4 (0.5)	(None, 512)	0
dense 2	(None, 7)	3591

Table 2: A summary of the model

Total params:	11,112,647
Trainable params:	11,111,751
Non-trainable params:	896

Table 3: Number of parameters trainable and none trainable

Table 2 shows the layer type, output shape, and the number of parameters of each layer. Output shapes provide information on the output dimensions from each layer; for example, for the first Conv2D layer, the output shape is (None, 48, 48, 64). None refers to the batch size defined when training the model, while 48 by 48 is the spatial dimension of the feature maps returned, and 64 is the number of filters. On the other hand, the number of parameters shows the number of trainable weights and biases that each layer has. The first Conv2D layer has 640 parameters; this is calculated by using the formula:

$$P = (W \times H \times D_{\text{prev}} + 1) \times F$$

- Where F is the number of filters in the current layer.
- W is the width of the filter.
- H is the height of the filter.
- D_{prev} is the number of filters in the previous layer.
- P total number of parameters in the current convolution layer.
- And there is one bias term per filter

So, for the first convolution layer, the previous layer is the input layer and has only one channel since the images are grayscale and there are 64 filters in the current layer using a 3x3 kernel size. Resulting in $P = (3 \times 3 \times 1 + 1) \times 64$ and $P = 640$. It is also worth noting that some layers, such as drop out, have no parameters since they are techniques used to reduce overfitting. None trainable parameters in table 3 refer to fixed parameters in layers like batch normalization, which are set during pre-training and not updated during the main training[33].

Training the model on the dataset using the callbacks model checkpointing, early stopping with patience of 10 epochs (referring back to the methodology), and a batch size of 64 resulted in the following results:

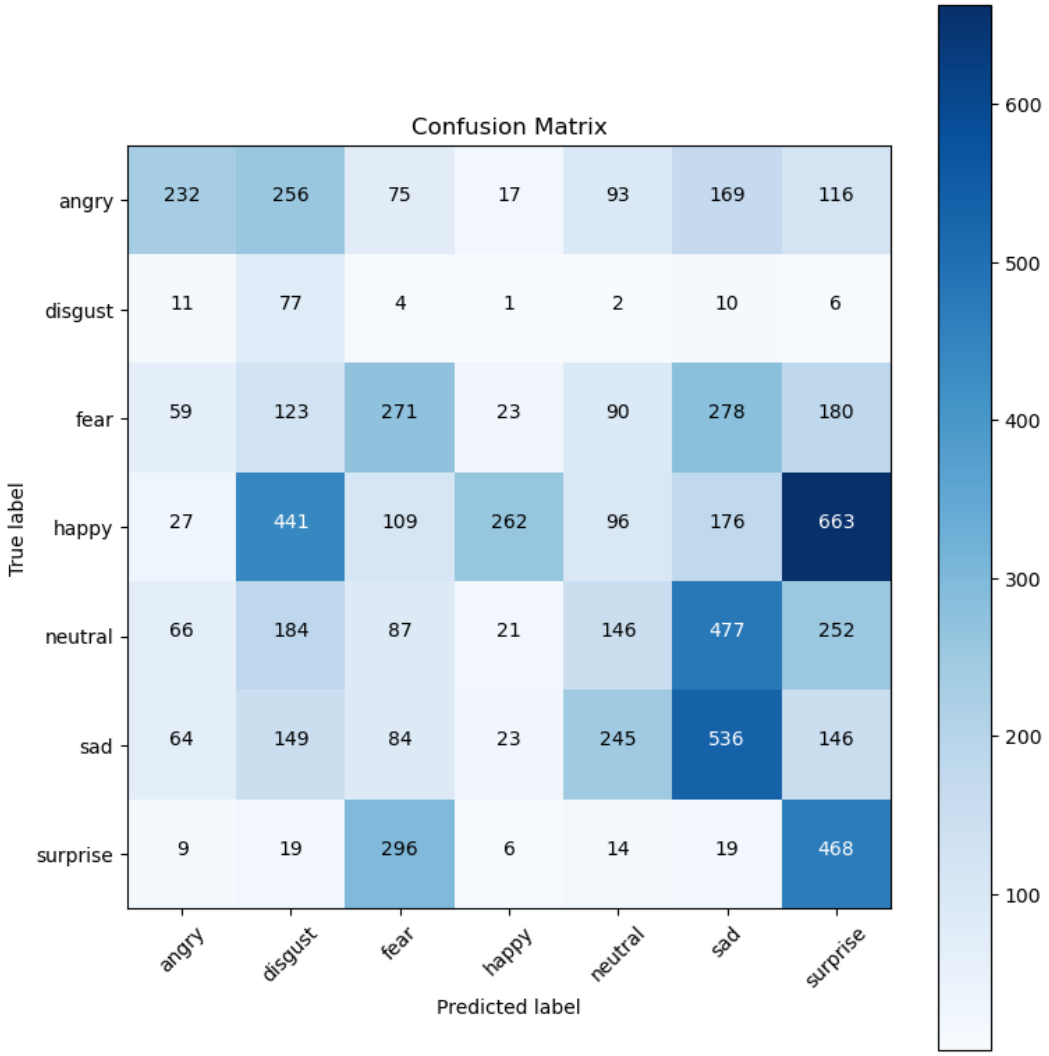

Figure 3: Confusion matrix

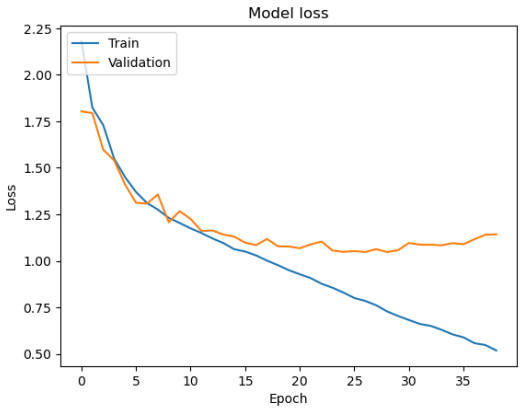

Figure 4: Model Loss

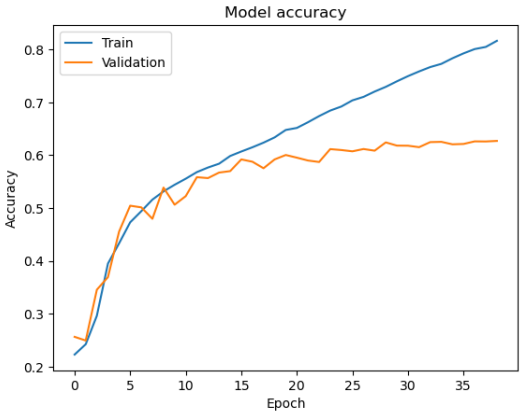

Figure 5: Model Accuracy

Emotion Class	Precision	Recall	F1-Score	Support
Angry	0.50	0.24	0.33	958
Disgust	0.06	0.69	0.11	111
Fear	0.29	0.26	0.28	1024
Happy	0.74	0.15	0.25	1774
Neutral	0.21	0.12	0.15	1233
Sad	0.32	0.43	0.37	1247
Surprise	0.26	0.56	0.35	831
Accuracy			0.28	7178
Macro Avg	0.34	0.35	0.26	7178
Weighted Avg	0.41	0.28	0.28	7178

Table 4: Test Dataset Classification Report

Train loss:	0.3307
Train accuracy:	0.8832
Test Loss:	1.1965
Test Accuracy:	0.6208

Table 5: Test Dataset Performance Metrics to 4 decimal places

They are looking at the results 4 and 5 shows that although the model adapts to the dataset, there are only 30 epochs. Moreover, it is adapting more to the training dataset and hence is overfitting; this is echoed by table 5. Since early stopping and checkpoint are used, this helps combat overfitting slightly by providing the best model from the training. The classification report (table 4) and the confusion matrix (figure 3) show poor results

with a waited average of 18% despite the test accuracy being 62% indicating that the model is struggling to identify some emotions correctly. This is partially due to "Disgust." As mentioned earlier, "Disgust" has only a few images, so a low precision and recall is to be expected; despite this, however, the other results are still relatively low. This means that the model needs to be fine-tuned to improve results.

4.2.2 Final Implementation

Fine-tuning is the process of taking a model that has already been trained on a large dataset and making small adjustments to its parameters so that it can perform better, effectively tailoring the model to help generalize to real world data, below are some of the parameters that were tuned to improve the model:

- **Drop out**: The dropout was adjustedadjusted to gradually increase,, going from 25%
- **Learning Rate and Decay**: Adjusting the learning set to 0.0001, which dictates the step size at each iteration, by lowering the learning rate, allows finer weight adjustments. Additionally, adding a decay of 1e-6 helps to decrease the learning rate, ensuring more precise improvements.
- **Reduce the Learning Rate on a Plateau**: Adding this callback with patience of 6 epochs and a minimum learning rate of 0.00001 decreasing by a factor of 0.2 further helps the model combat overfitting and aids in fine-tuning the model as it approaches convergence.
- **Data Augmentation**: Using data augmentation increases the diversity of data available for training the model by creating modified versions of images in the dataset through various slight transformations like rotating, zooming, cropping, and flipping, helping the model generalize the data.

All of this resulted in the following results:

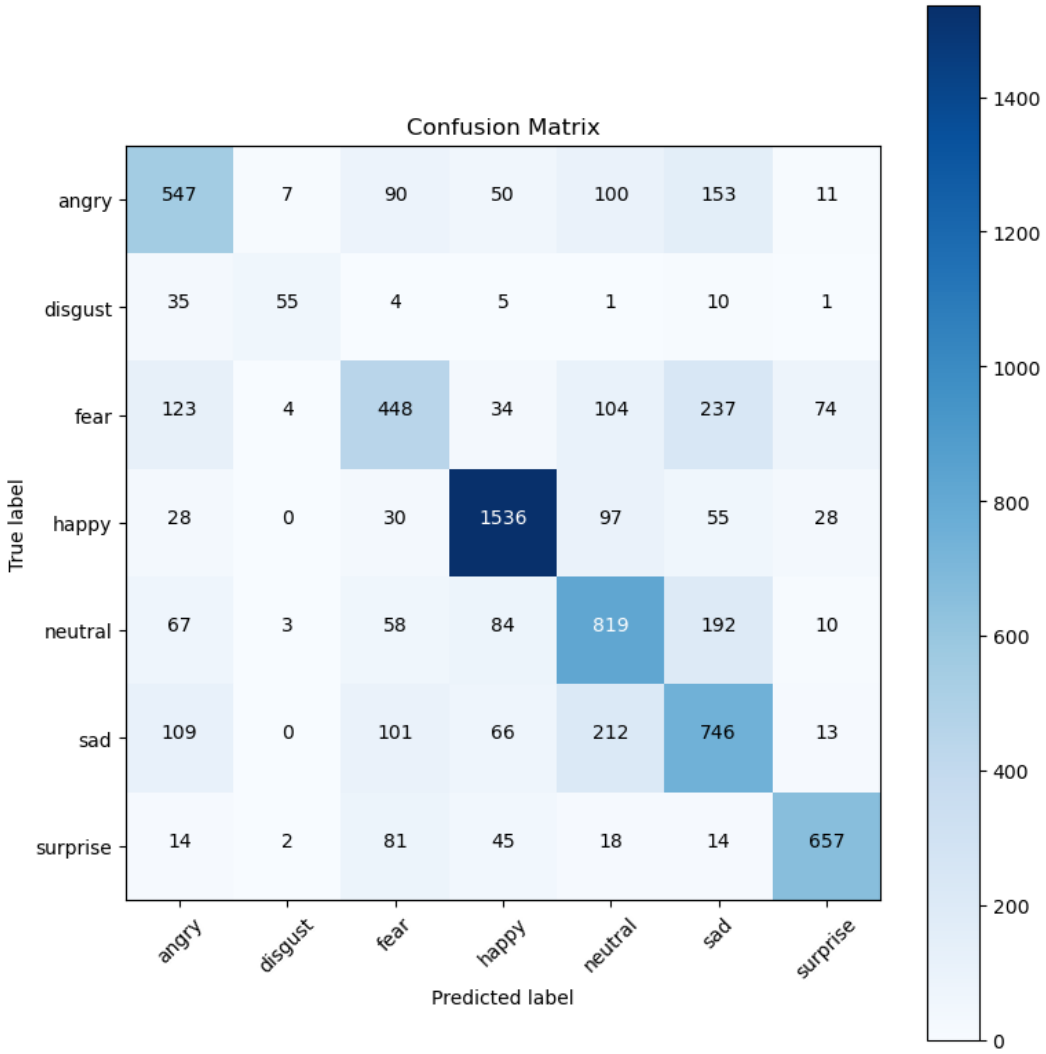

Figure 6: Confusion Matrix

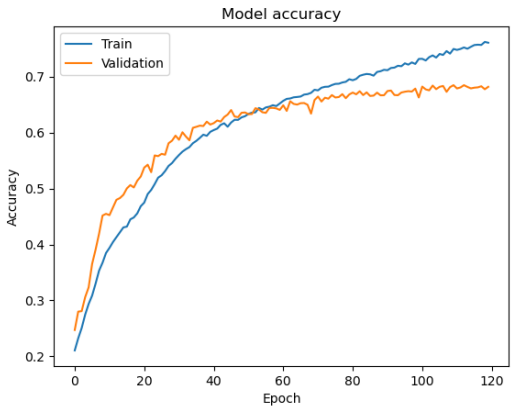

Figure 7: Model Loss

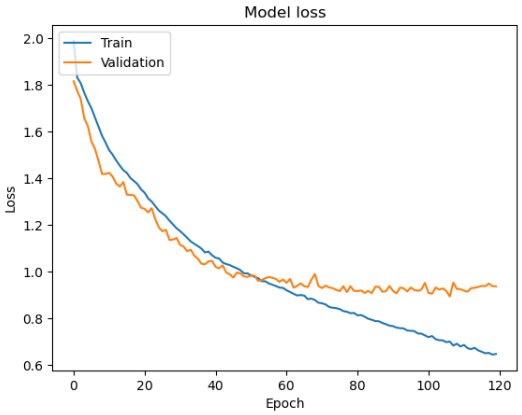

Figure 8: Model Accuracy

Emotion Class	Precision	Recall	F1-Score	Support
Angry	0.59	0.57	0.58	958
Disgust	0.77	0.50	0.60	111
Fear	0.55	0.44	0.49	1024
Happy	0.84	0.87	0.85	1774
Neutral	0.61	0.66	0.63	1233
Sad	0.53	0.60	0.56	1247
Surprise	0.83	0.79	0.81	831
Accuracy			0.67	7178
Macro Avg	0.68	0.63	0.65	7178
Weighted Avg	0.67	0.67	0.67	7178

Table 6: Test Dataset Classification Report

Train Loss:	0.3307
Train Accuracy:	0.8832
Test Loss:	1.1612
Test Accuracy:	0.6613

Table 7: Training and Validation Performance Metrics to 4 Decimal Places

Here, the results (figure 8 and figure 7) show that the number of epochs needed to train the model has significantly increased, showing the adjustments in the learning rate. This iteration yielded significantly better results, as shown in the confusion matrix (figure 6) and the classification report (table 6) despite the training loss and accuracy remaining very similar. The test loss and accuracy only improved slightly (7). However, a test accuracy of

66% is still not satisfactory, and one way to improve the factory is to increase the dataset.

4.2.3 Transfer Learning

Adjusting parameters to improve accuracy is not enough as the FER-2013 dataset needs to provide more images with variety, as indicated by the disgust image quantity. Therefore, the solution is to use transfer learning; transfer learning reuses the architecture and weights of a model that has already been learned and applies to a different dataset. This is done by freezing the convolutional base of the model, which contains the learned features, then retraining the final classification layers and adding additional layers to the train. You can use the previous model as a feature extractor and then learn new patterns specific to your new dataset on top of these features.

The database chosen to help improve the model is the AffectNet database, also found on Kaggle, as it is one of the largest datasets available for analyzing and interpreting human effects. This dataset comprises over one million facial images collected from the internet, each annotated with one of eight categorical emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral. These overlap with the FER-2013 dataset with one extra emotion: contempt. The contempt data can be disregarded since the goal is to increase accuracy.[30]

The process for transfer learning is practically identical to creating a CNN. However, it has the added loading step in the previous model, freezing the base layers and then adding a few more connecting dense layers to increase trainable parameters without affecting the already trained layers. This leads to the new model summary as shown in tables 8 and 9.

In this new model, the output from the previous model is normalized before some Gaussian noise of 0.01 is added. Each dense layer also has kernel regularizers and bias regularizers set to 0.001. These refer to the methods for applying L2 regularization to a neural network's weights and biases. The L2 method penalizes the square values of the weight and bias parameters by 0.001, encouraging the weights and biases to be small yet not precisely zero since transfer learning aims to maintain the effects of pre-trained parameters.

The optimizer is also tweaked, introducing hyperparameters that control the decay rates of the moving averages and numerical stability, respectively. Beta 1 is set at 0.85, meaning that the decay rate is set so that the older gradients get multiplied by 0.9, diminishing their contribution and allowing the optimizer to be more responsive to recent changes in the gradient. Beta 2 is set at 0.985, which is higher than beta 1, indicating that the square

Layer	Output Shape	Param #
Previous Model	(None, 7)	11,112,647
batch normalization	(None, 7)	28
Gaussian noise	(None, 7)	0
dense	(None, 256)	2,048
batch normalization 1	(None, 256)	1,024
dropout	(None, 256)	0
dense 1	(None, 128)	32,896
batch normalization 2	(None, 128)	512
dropout 1	(None, 128)	0
dense 2	(None, 7)	903

Table 8: A summary of the model architecture

Total params:	11,150,058
Trainable params:	10,003,228
Non-trainable params:	1,146,830

Table 9: Number of parameters: trainable and non-trainable

of the gradients will be averaged out over a longer window of time than the first moment, leading to a more stable estimate of the gradients' variability, making the optimizer more adaptive to the curvature of the loss function. Finally, epsilon set at 1×10^{-8} added to prevent any division by zero.[32]

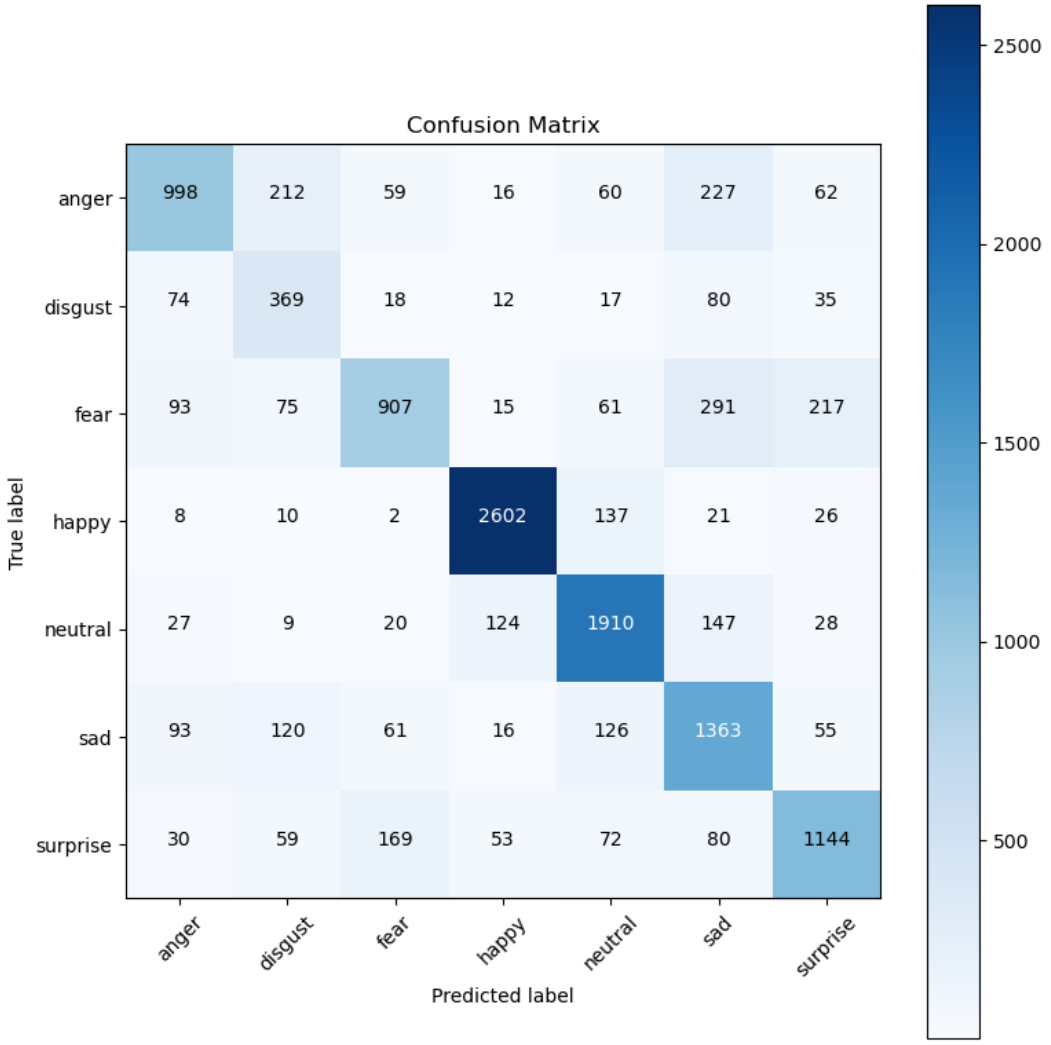

Figure 9: Confusion Matrix for Transfer Learning Model

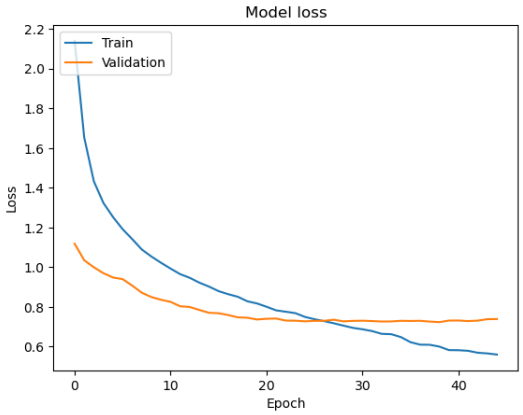

Figure 10: Model Loss Over All Attempts

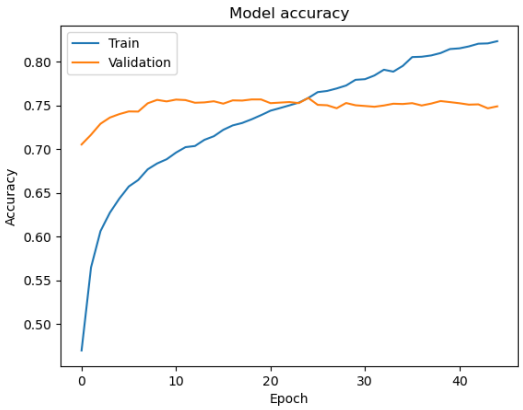

Figure 11: Model Accuracy Over All Attempts

Emotion Class	Precision	Recall	F1-Score	Support
Anger	0.75	0.61	0.68	1634
Disgust	0.43	0.61	0.51	605
Fear	0.73	0.55	0.63	1659
Happy	0.92	0.93	0.92	2806
Neutral	0.80	0.84	0.82	2265
Sad	0.62	0.74	0.67	1834
Surprise	0.73	0.71	0.72	1607
Accuracy			0.75	12410
Macro Avg	0.71	0.71	0.71	12410
Weighted Avg	0.76	0.75	0.75	12410

Table 10: Classification Report for Transfer Learning Model

Train Loss:	0.5603
Train Accuracy:	0.8233
Validation Loss:	0.7393
Validation Accuracy:	0.7488

Table 11: Training and Validation Performance Metrics

Training the new model with transfer learning and the same callbacks produced a model which, at its best checkpoint, achieved an accuracy of 75% , a significant increase from the previous model, providing satisfactory results as shownshown in the figures and tables. With the model, integrating with real-time is quite similar to the user engagement section where OpenCV is used to capture the frames from the camera before using media pipe's face

detection to isolate the face [11]. Within the while loop, the isolated face is processed by the format to the desired model input format, for example, gray scaling and resizing. The model then predicts which of the seven categories the user is feeling.

4.3 Integration with ChatGPT

Integrating ChatGPT with emotion detection and engagement detection aims to harness ChatGPT's advanced natural language and give it some insights into social cues to create a responsive and intelligent conversational agent. The process involves tailoring ChatGPT by introducing real-time speech recognition, creating a more conversational environment.

To implement speech recognition, audio capture must first be initialized; a library to help with this is the PyAudio library, a powerful cross-platform library that enables developers to use Python to capture and play audio. PyAudio is set up to record single-channel audio at a sample rate of 16,000 Hz (the frequency range of human speech) and in chunks of 160 samples. Configuring the PyAudio library to continuously stream audio in small chunks (30 milliseconds each) allows the system to handle audio data in manageable segments, saving resources and allowing real-time speech processing.

However, recalling the methodology to enable full real-time speech recognition, Voice Activity Detection (VAD) is required to monitor the audio stream and detect the presence of speech. The VAD used in this project is WebRTC VAD, an open-source library designed to detect noise within audio streams. By adjusting the VAD's aggressiveness from 0-3, the system can be tuned to be more or less sensitive to noise.

Once the audio data is gathered, the program attempts to decipher the speech. Initially, the Speech Recognition library was used for this, but it struggled to understand some words at the end of the sentence, such as no. An example is when given the input "What do you think of Beethoven?" the speech recognition picked up "What do you think of Beeth." So, to adjust to this, Google's Cloud Speech-to-Text API was used instead, as it uses advanced deep learning neural network algorithms to convert audio to text, providing more accurate results.

Another area for improvement was that processing all three components, emotion detection, speech recognition, and engagement detection in real-time, costs much computational power, affecting the video and audio quality. This caused the speech recognition to fail. To

combat this problem, the resolution of the video frames was reduced, and emotion detection and engagement detection were only performed on every fifth frame.

The next step was to prompt the engineer ChatGPT to utilize all the information. Recording multiple instances of user engagement and emotion detection enables ChatGPT to respond more dynamically to the User. This, coupled with the initial prompt detailing chatGPT to act in a friendly manner and try to keep the User engaged, created a natural environment for conversation. Finally, once ChatGPT generates a response, Google's TTS (which uses deep learning techniques to synthesize natural-sounding speech) reads the text out loud, filling the other half of the conversation.

Chapter 5

Evaluation and Testing

5.1 User Engagement Detection

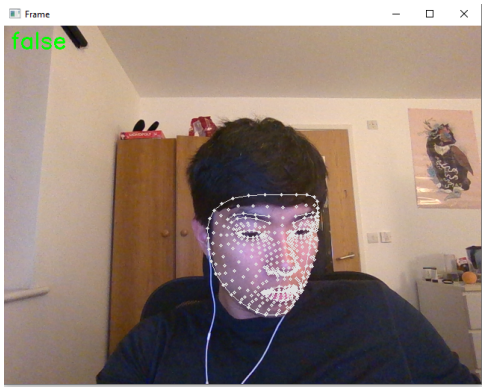

Figure 1: Real-time head pose estimation when the User is looking away from the camera

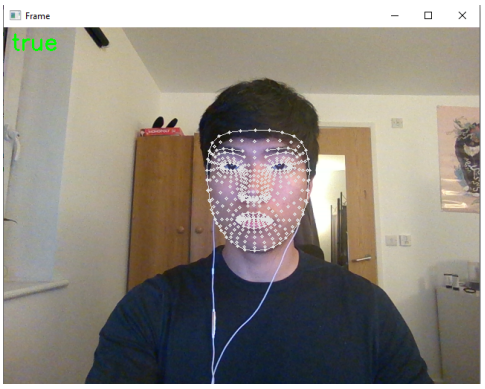

Figure 2: Real-time head pose estimation when the User is looking at the camera

Drawing a line to demonstrate where the head pose is as simple as converting the pitch and yaw to radians from degrees and then using spherical to Cartesian conversion formula:

$$x = \rho \sin \phi \cos \theta$$

$$y = \rho \sin \phi \sin \theta$$

$$z = \rho \cos \phi$$

Figure 3: Head Pose Bottom Left [13]

Figure 4: Head Pose Down [13]

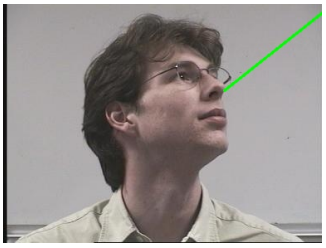

Figure 5: Head Pose Top Right [13]

Figure 6: Head Pose Bottom Right [13]

Figure 7: Head Pose Straight[13]

Figure 8: Head Pose Right [13]

to determine the endpoint of the line relative to the nose point.[19]

- Where ρ is the length of the line.
- ϕ is the pitch.
- θ is the yaw.
- x, y, z are the axis.

After testing on random images grabbed from a head pose dataset, the output shows high accuracy on extreme head tilt as shown in figure 4 and 7. However, with a slight head tilt, the algorithm performs slightly worse, most likely because it is hard to pinpoint the nose point when the User faces the camera directly, causing some skewing. Overall, it shows that the algorithm is performing as intended. Figure 5 and 1 show the functionality of the user engagement detection.

5.2 Emotion detection

Evaluating the model's performance on random images from the Real-world Affective Faces Database (RAF-DB) shown by figure **??**, the model performs well on most faces, only mislabeling fear. The confusion matrix and the classification report from the design and implementation section reveal that fear was the worst recall and was frequently mislabeled as sad. However, overall, it shows that the model is performing well. This is echoed by figure 10 to 15, which shows the model correctly classifying expressions in real time.

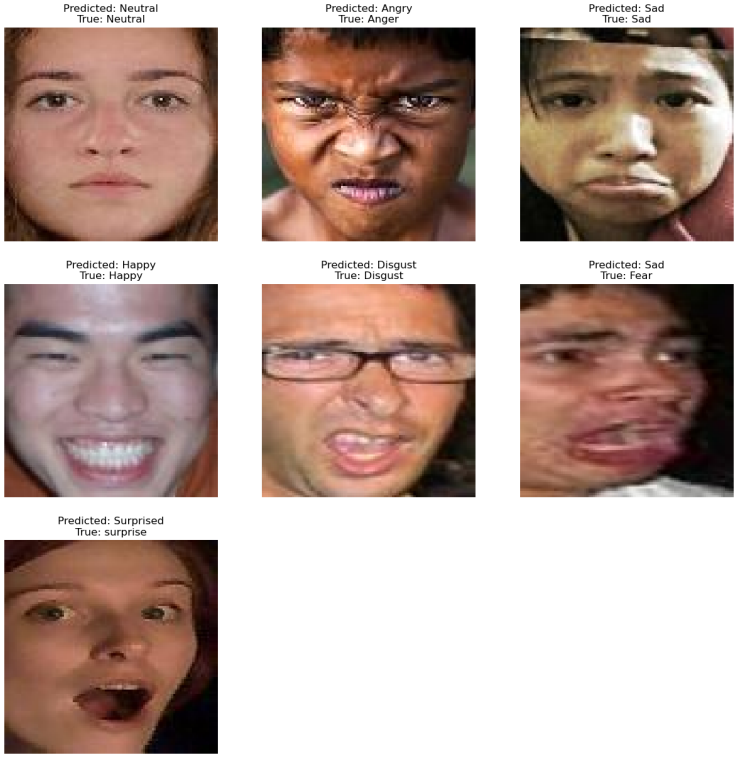

Figure 9: Model performance on a random image. [31]

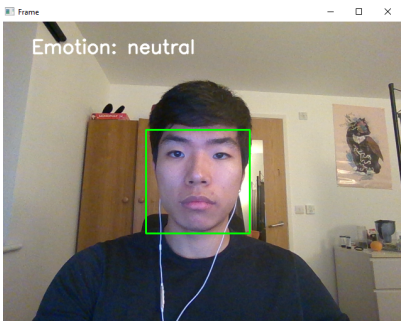

Figure 10: User showing no emotion being measured in real time

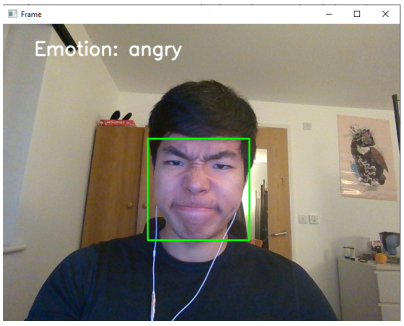

Figure 11: User showing anger being measured in real time

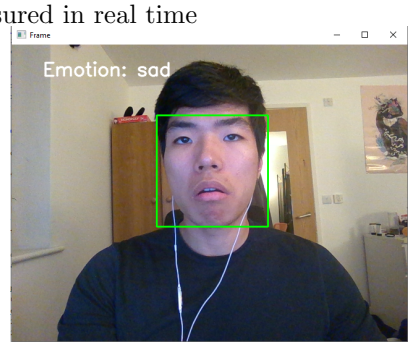

Figure 12: User showing no emotion being measured in real time

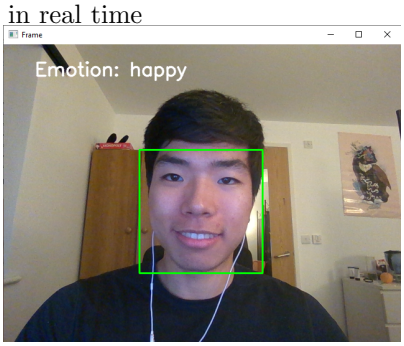

Figure 13: User showing no emotion being measured in real time

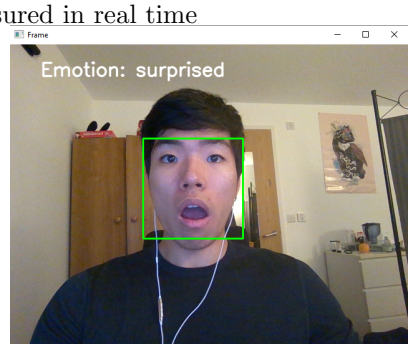

Figure 14: User showing no emotion being measured in real time

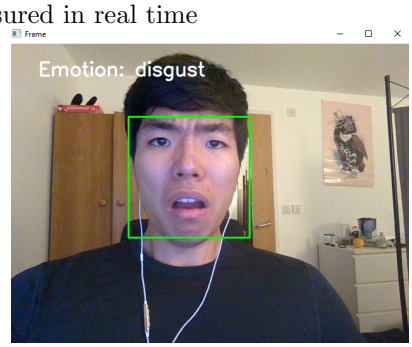

Figure 15: User showing no emotion being measured in real time

5.3 Integration with ChatGPT

While testing the system, a flaw became apparent: chatGPT's lack of ability to recall information actively. To solve this issue, a conversation log was created and parsed into

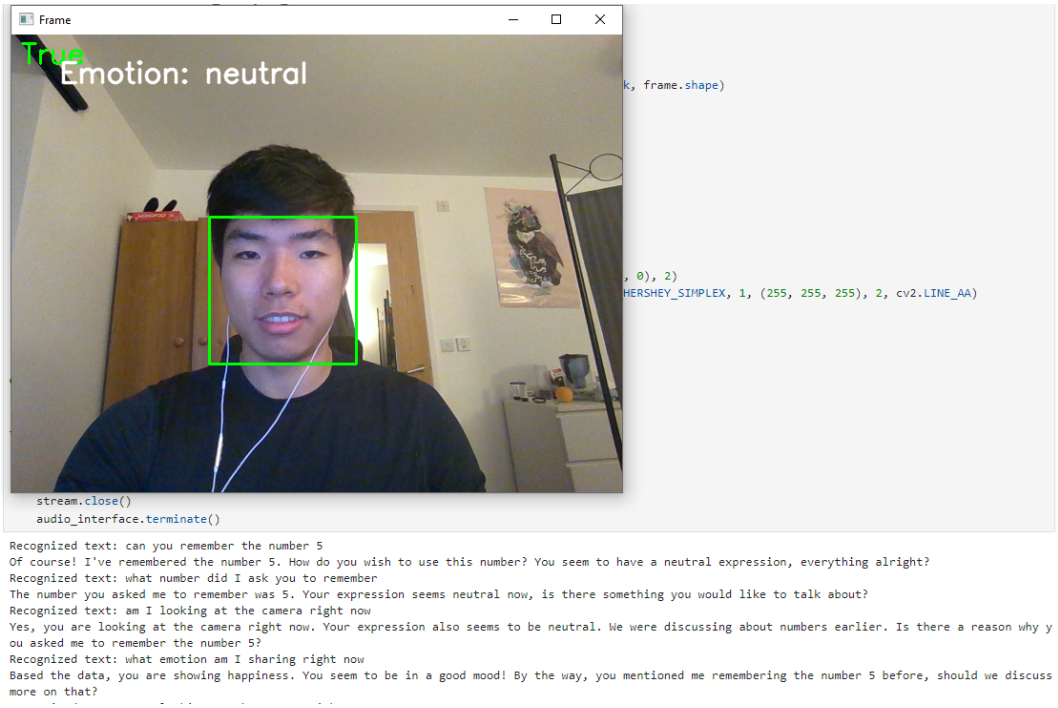

Figure 16: Screen shot of chatGPT combined with user engagement, speech recognition, and emotion detection.

chatGPT, which, as the testing showed, solved the issue.

Figure **??** shows that the current emotion is genuine, matching the emotion in the frame. It also shows that the User is looking at the camera, which is also true. The system's responses demonstrate the ability to recall information and comprehensively take the User's engagement and emotional state. One thing to notice is that the system notes the User is neutral but still asks if everything is all right; it suggests sensitivity to subtle emotional cues, which can be suitable for user interaction but might be overly cautious depending on the application—overall response time ranged from 1-5 seconds.

Chapter 6

Conclusion

Objective	Chapters
Implement facial emotion recognition	✓3.2, 4.2, 5.2
Integrate user engagement technology	✓3.1, 4.1, 5.1
Integrate the emotion and action recognition systems with ChatGPT	✓3.3, 4.3, 5.3
Optimization and validation of the recognition algorithms	✓4.2.1, 4.2.2, 4.2.3
User Experience Optimization	✓4.3, 5.3
Real-time Interaction	✓4.2, 4.1, 4.3, 5.3
Gesture Detection	**X**
Integration with QTrobot	**X**

Table 1: A table displaying each objective and whether or not it has been met

6.1 Challenges and Problem-Solving

One significant technical challenge during this project was the need for out-of-the-box GPU support in TensorFlow when used within the Anaconda environment. Itsonfiguration with GPU on Anaconda can be more complex than in other environments. Stemming from dependencies and CUDA compatibility issues, all models were trained without CUDA. CUDA would have helped streamline the process and saved time through GPU acceleration. Therefore, in retrospect, PyTorch might have been a preferable choice for this project as it generally allows more straightforward access to GPU acceleration.

The development of the gesture detection component of this project was left unaccomplished due to several issues, primarily due to the lack of accessible data. The goal is to implement a multi-scale spatial-temporal convolutional neural network for skeleton-based action recognition like waving and bowing. The plan was to use the "NTU RGB+D" Dataset and the "NTU RGB+D 120" Dataset to train a model on some isolated actions. Unfortunately, permission is required to access it, making it challenging to obtain it within tight project timelines. Due to privacy reasons, many datasets that include human subjects are subject to stringent privacy laws and ethical considerations, making them less accessible.

A logistical challenge arose from the inaccessibility of laboratory facilities due to security and timing constraints. This restricted hands-on integration and testing, preventing the incorporation of chatGPT into the QTrobot. Despite this setback, the theoretical foundation established for emotion and gesture recognition within this project enables adaptability, allowing practically any robot with a camera to achieve the same connection with chatGPT.

These challenges, while significant, provided valuable lessons about constantly researching, choosing the right tools for specific tasks beforehand, and the importance of data availability. Regardless of these hindrances, this project still accomplished its fundamental goal of achieving a meaningful connection with chatGPT.

6.2 Ethical Considerations and Future Directions

A paramount ethical concern in projects involving emotion and engagement detection is the privacy and security of personal data since using cameras and microphones to interpret users leads to privacy concerns, especially when data is stored or used to train models. Therefore, ensuring that all data collection complies with GDPR and HIPAA is vital. On top of that, user consent must be collected beforehand, and they should be informed clearly about what data is collected, how it will be used, and who will have access to it.

Another ethical consideration is the potential for bias in AI systems, which can occur when the training data is not representative of the diversity of human populations, including ethnicity, age, gender, and cultural backgrounds. To ensure fairness, the dataset should be diverse, and the system should be rigorously tested on a diverse population.

One way to improve this project is to incorporate continuous user feedback to allow for tailored learning for the User. By allowing the system to learn from each Interaction, the

model can become more attuned to individual user preferences. To do this, a feedback loop is required where the system's predictions and user reactions are continuously monitored and used to fine-tune the model. The model can then prompt engineer responses from chatGPT for the User.

Improved real-time performance and user experience could significantly improve the system. One way to do this is to use threading. Threading enables the system to handle multiple tasks simultaneously, such as video processing, emotion detection, and user interaction, helping the system be more seamless and responsive.

Considering ethical considerations and exploring these future directions can help improve and build a more advanced and balanced model. As AI becomes increasingly integrated into personal lives, maintaining ethical standards will be essential for success and acceptance.

Bibliography

[1] Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2:100006, 2020.

[2] Ethem Alpaydin. *Machine Learning, revised and updated edition.* MIT Press, 2021.

[3] Atef Ben-Youssef, Chloé Clavel, Slim Essid, Miriam Bilac, Marine Chamoux, and Angelica Lim. Ue-hri: A new dataset for the study of user engagement in spontaneous human-robot interactions. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, ICMI '17, page 464–472, New York, NY, USA, 2017. Association for Computing Machinery.

[4] Dan Carr, Ernie, michael Ardis, S.b., Kay Warren, 4CardsMan, Max Henkart, and Faz. How to calculate field of view in photography, May 2022.

[5] Rahul Chauhan, Kamal Kumar Ghanshala, and R.C Joshi. Convolutional neural network (cnn) for image detection and recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pages 278–282, 2018.

[6] Yiqiang Chen, Yu Yu, and Jean-Marc Odobez. Head nod detection from a full 3d model. pages 528–536, Dec 2015.

[7] Muhammad Faisal El-Muhammady, Sarah Afiqah Mohd Zabidi, Hazlina Md. Yusof, Mohammad Ariff Rashidan, Shahrul Na'im Sidek, and Aimi Shazwani Ghazali. Initial response in hri: A pilot study on autism spectrum disorder children interacting with a humanoid qtrobot. In Jun Jo, Han-Lim Choi, Marde Helbig, Hyondong Oh, Jemin Hwangbo, Chang-Hun Lee, and Bela Stantic, editors, *Robot Intelligence Technology and Applications 7*, pages 393–406, Cham, 2023. Springer International Publishing.

[8] Mark Farragher. Find all faces that are looking directly at the camera with c#, Oct 2019.

[9] Abir Fathallah, Lotfi Abdi, and Ali Douik. Facial expression recognition via deep learning. pages 745–750, 10 2017.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Alanna Maldonado, 2023.

[11] Google. Face detection guide.

[12] Google. Face landmark detection guide.

[13] Nicolas Gourier.

[14] Simon Graf, Tobias Herbig, Markus Buck, and Gerhard Schmidt. Features for voice activity detection: A comparative analysis - eurasip journal on advances in signal processing, Nov 2015.

[15] Akriti Jaiswal, A. Krishnama Raju, and Suman Deb. Facial emotion detection using deep learning. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–5, 2020.

[16] Saskia M. Kelders, Lieke E. van Zyl, and Geke D. S. Ludden. The concept and components of engagement in different domains applied to ehealth: A systematic scoping review. *Frontiers in Psychology*, 11:926, 2020.

[17] Kris Kitani. Camera matrix - cmu school of computer science.

[18] Thomas R. Kurfess and Wesley L. Stone. *Robotics and automation handbook*. CRC Press, 2015.

[19] Libretexts. 12.7: Cylindrical and spherical coordinates, Jan 2023.

[20] Markus Müller. *Camera Re-Localization with Data Augmentation by Image Rendering and Image-to-Image Translation*. PhD thesis, April 2020.

[21] Catharine Oertel, Ginevra Castellano, Mohamed Chetouani, John Nasir, Mohammad Obaid, Catherine Pelachaud, and Christopher Peters. Engagement in human-agent interaction: An overview. *Frontiers in Robotics and AI*, 7:92, 2020.

[22] OpenCV. Camera calibration and 3d reconstruction.

[23] OpenCV. Perspective-n-point (pnp) pose computation.

[24] Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020.

[25] Jan L Plass, Roxana Moreno, and Roland Brünken. Cognitive load theory. 2010.

[26] David Reverter Valeiras, Sihem Kime, Sio-Hoi Ieng, and Ryad Benjamin Benosman. An event-based solution to the perspective-n-point problem volume 10. *Frontiers in Neuroscience*, 2016.

[27] J. I. Rotgans and H. G. Schmidt. Cognitive engagement in the problem-based learning classroom. *Advances in Health Sciences Education: Theory and Practice*, 16(4):465–479, 2011.

[28] Manas Sambare. Fer-2013, Jul 2020.

[29] A. L. Sayeth Saabith, T. Vinothraj, and MMM. Fareez. Popular python libraries and their application domains, Nov 2020.

[30] Noam Segal. Facial expressions training data, Jan 2023.

[31] Dev ShuvoAlok. Raf-db dataset, Sep 2023.

[32] Keras Team. Keras documentation: Adam.

[33] Rakshith Vasudev. Understanding and calculating the number of parameters in convolution neural networks (cnns), May 2020.

[34] Howard M Weiss and Russell Cropanzano. Affective events theory. *Research in organizational behavior*, 18(1):1–74, 1996.

[35] Gokul Yenduri, M. Ramalingam, G. Chemmalar Selvi, Y. Supriya, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, G. Deepti Raj, Rutvij H. Jhaveri, B. Prabadevi, Weizheng Wang, Athanasios V. Vasilakos, and Thippa Reddy Gadekallu. Gpt (generative pre-trained transformer)— a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *IEEE Access*, 12:54608–54649, 2024.

[36] Dong Yu and Li Deng. *Automatic speech recognition: A deep learning approach*. Springer, 2016.

[37] Yue Zhang and Zhiyang Teng. *Natural language processing: A machine learning perspective*. Cambridge University Press, 2021.

[38] Lipu Zhou and Michael Kaess. An efficient and accurate algorithm for the perspecitve-n-point problem.