University of the West of England

Testing the viability of consumer grade peripherals to predict user confidence when making decisions.

By

Harrison J. Bennion

17012546

BSc (HONS) Computer Science

UFCFXK-30-3 Digital System Project

April 2021

Word Count: 11197

Gitlab Link: https://gitlab.uwe.ac.uk/hj2-bennion/DSP-Project

Abstract

With artificial intelligence, and by extension active learning, becoming more and more prevalent among certain industries, the need for humans to accurately label images to feed the active learning algorithm is becoming a larger problem to tackle. However, humans aren't always completely confident in their answers, and instead their confidence will waver with each answer or every image labelled. From previous work by Smith *et al* (2019), it was shown that a high-end eye tracker could be used to track user's gaze, which in turn could be used to predict the confidence of participants as they labelled images.

Although an incredibly useful discovery, the eye tracking method required the use of both an extremely expensive eye tracker and a tightly controlled environment to track user's gaze. From this, the question was prompted: Could a cheaper alternative be used in a casual environment to predict user's confidence? In this paper, a tool was developed to begin working to answer this very question, adopting the use of a consumer-grade Tobii eye tracker 4C alongside the active learning environment, ActiVAte.

Acknowledgements

As a part of this project, I'd like to say my thanks to both my family and friends who have always taken an interest in this project, motivated me to push onwards and achieve the best I can.

Table of Contents

Table of figures

Table of tables

Abbreviations

Abbreviation	*Definition*
AOI	Area Of Interest
CIFAR-10	Canadian Institure For Advanced Research – 10 categories
CNN	Convolutional Neural Network
EEG	Electroencephalogram
GB	Gigabyte
GUI	Graphical User Interface
IDE	Intergrated Development Evnironment
MoSCoW	Must, Should, Could, Would
OS	Operating System
RAM	Random Access Memory
SDK	Software Development Kit
UWP	Universal Windows Platform
Venv	Virtual ENVrionments

1 Introduction

This chapter of the report aims to introduce the reader to the report's premise, discussing the overall importance of the work completed within, the questions it plans to look at and the objectives it wishes to complete. Finally, this chapter contains a brief overview of each of the other chapters to lay down the structure this report will follow.

1.1 Overview

With the rise of supervised machine learning, one strain called active learning has become increasingly popular as the required dataset is usually smaller in comparison to other methods. During the completion of active learning, a human-labelled dataset is required as part of the process, but it is uncommon to find a pre-labelled dataset for a required task. As a result, humans must complete the time-intensive tasks of labelling the data, in which it is common for humans to make mistakes or become distracted due to the environment they are in. This project aims to look at using peripheral devices to measure the concentration and the confidence of their answers whilst they label images as part of an active learning process.

1.2 Why is this research important?

During the training of an active learning model, the computer requires a human to label chosen pieces of data with the desired output. Without this first fundamental step, the model will get nowhere as it requires labels to begin the learning process. As stated, the human is a key part of this process, but trusting the human to always make the right decision may not be the best decision as humans are prone to errors or mistakes. In order to detect the level of confidence a user has in their answer, it is possible to simply ask them how confident they feel; however, this obviously further increases the time intensity of an already time-intensive task. As a result, in order to automate this process, it is possible to monitor a user's gaze path and their areas of fixation using a highly accurate research eye tracker to predict how confident they are in each of their answers, however this isn't viable as an option for most commercial or consumer groups due to the sheer cost and setup required to accurately monitor user's gaze in this fashion. Resulting from this, research if required to look into the

use of consumer grade peripherals to predict the confidence of others when making decisions in the workplace.

One example of applying this to the real world could be in the case of a doctor labelling x-ray imagery where there is a chance that they may miss a small detail due to distraction or may feel unconfident in their answer, which they could then use when completing a report or deciding what a correct diagnosis is.

1.3 Research Purpose

The primary purpose of this research is to study the potential use of commonly available peripherals, with a focus on an eye tracker to predict the confidence and distraction of user whilst they complete visual labelling tasks. This project focuses on the development of a simple tool which can be quickly setup and used collect and analyse the data from the eye tracker whilst a participant labels images as part of an active learning task.

1.3.1 Research Questions

Primary: Can a consumer grade eye tracker be used to produce a gazemap which accurately predicts the confidence of a user during a labelling task?

Secondary: Can the gazemap generation be extended to work with a consumer grade EEG headset alongside ActiVAte.

1.4 Aim and Objectives

1.4.1 Aim

The aim of this research is to test the viability of using a consumer grade eye tracker alongside a readily available EEG headset to predict the confidence and distraction of a user to a level that's comparable to the eye tracker used in the paper by Smith *et al.* (2018).

1.4.2 Objectives

1. Explore the literature and previous papers which have investigated this area of study.
2. Use the eye tracker alongside their developer libraries to produce a gazemap from the omitted data.

3. Update and adapt ActiVAte to allow users to label CIFAR-10 images whilst their gaze is tracked.
4. Test the application of the MindWave EEG headset as an additional source of information.
5. Implement an adaptive convolutional neural network which will use the produced gazemaps to predict the user's confidence on future labelling tasks.
6. Interpret the gazemaps that are produced to deduce the decision-making process.

1.5 Report Structure

This section defines and quickly describes the different chapters in this report to allow users to select certain sections to read and to improve the ease of referencing different sections if required:

1. Introduction: Outline the basis and relevance of this paper, as well as depicting the goals it wishes to be completed by the end.
2. Literature Review: Research and summarise previous pieces of literature, gaining a greater insight into the work that has been completed before in this subject area.
3. Requirements: Explicitly define the functional and non-functional requirements that will be used to assess the of the project.
4. Design Methodology: Describe the procedure used to achieve this project's aims, including the software development practises.
5. Program Design: Detail the programmatic design of the program, including both the tools and languages in which it will be implemented with.
6. Implementation: Outline the actual software implementation and the thoughts behind the design, as well as discussing its limitations and whether the program has met the requirements
7. Project Evaluation: Discuss the work that was undertaken as part of this project, the scope of the requirements and future ideas for work that could be done.
8. Conclusion: Presents a summary of the project, picking out key details that it wishes to convey, and providing a closing to the report.

1.6 Chapter Summary

This chapter has outlined the basis of this paper, writing about its importance and the reasoning behind why it is being written. Following on from the overview, the research question, aims, and objectives were detailed to help guide the work that completed during this paper. Finally, a short synopsis of each of the chapters was included to allow the reader to learn about the overall flow and layout of the paper for each reference or selective reading.

2 Literature Review

This chapter of the report aims to explore a variety of literature relating to this project. The beginning of this chapter will primarily investigate the previous work by Smith *et al*. (2018), before studying other previous efforts at automating the detection of confidence, the effects that visual decision-making tasks have on people, before finally studying the use of EEG devices to measure both confidence and awareness.

2.1 Background information

As artificial intelligence becomes increasingly popular and more readily available, various industries are taking it up in a variety of forms to improve their workload or automate certain tasks. However, some industries such as defence or healthcare require a much greater level of accuracy in many aspects of their function, and so the increasing the accuracy of machine learning models is a requirement before these algorithms can be fully relied upon. As explained by Smith *et al.* (2018), there is a fundamental assumption made about humans when completing tasks, such as image labelling, that their answers are correct all of this time, when this is highly likely to be incorrect and instead, they will have a varying level of confidence in each of their answers.

Lughofer *et al*. (2009) studied the effects of capturing and using user confidence values during vision-based labelling tasks. They found that there was a reduction in error rates up to 50% when using the participant's confidence values; from this it seems clear that measuring the confidence values of users when gathering data makes sense as it can provide a much clearer prediction model. However, this paper still recognised a few flaws regarding the recording of user confidence values. Due to the presence of a GUI shown in Figure 1, which required attention from the users after every question, it was common for the users to become distracted and divert their attention away from the task at hand. The extra time required to record their confidence caused an issue with the scalability of this approach, as it had the potential to require a significant portion of time when completing each task which would stunt the completion rates on larger projects.

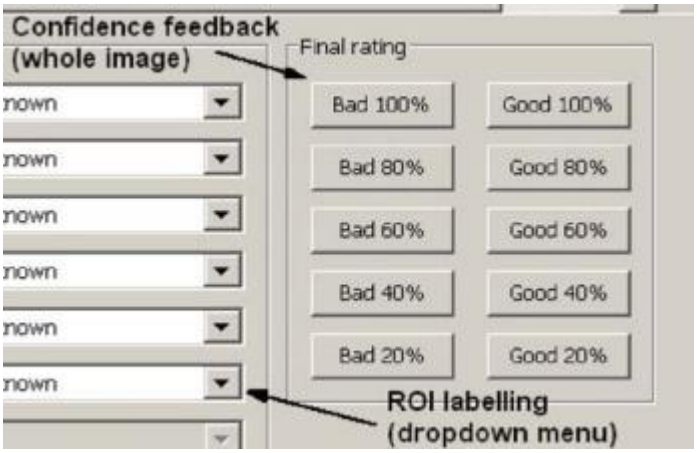

Figure 1 - Lughofer *et al.* (2009) GUI

From this conclusion, the idea was proposed to collect information about the user's confidence without interrupting their workflow. To do this, Smith *et al.* (2018) hypothesised that an eye-tracker could be used to monitor the eye movements of users when collecting information to predict their confidence without interrupting or imposing on their work. Previous work by Steichen, Conati and Carenini (2014), as well as Roderer and Roebers (2010) demonstrated how the fixation location, dwell time, saccadic movements or smooth movements along a scan-path can provide useful analysis when trying to understand human perception. From these papers, the idea presented by Smith *et al.* (2018) has sound reasoning behind the experiment and so taking a deeper look into this study would be highly advantageous.

2.2 Previous work automating detection of confidence values

The paper by Smith *et al.* (2018) investigated the use of a high-end eye tracker within a highly controlled environment to monitor the eye motion and fixation data of users as they completed a visual labelling task. During this paper, Smith *et al.* covered a range of different techniques to demonstrate that a model could trained to accurately identify a user's confidence based on certain eye movements and fixations. Although, this method was able to accurately predict the user's confidence, due to the extremely limited design and requirements that needed the participants to rest their chin on a platform, alongside the cost of the eye-tracker, it would be very difficult to employ this technique on a much larger scale. From this,

it makes sense to potentially use a similar technique on a consumer-grade eye-tracker instead with the aim to compare the twos performance.

Alongside Smith *et al*. (2018), there were previous attempts at using an eye tracker to monitor and study concentration and confidence. The book by Chen *et al.* (2016) studied how cognitive load could be measured using a variety of techniques which included using an eye tracker under a few conditions. Chen *et al.* (2016, cited by Smith *et al.* 2018) found that various features of eye movement could predict the user's cognitive load, however this research looked at eye movement at a much more macro level, rather than capturing finer-grain information such as the sequence of eye fixations. The work by Smith *et al*. (2018) and by extension this report plans to capture the finer details that can be found when someone is considering a visual task. Furthermore, it should be noted that the work by Chen *et al.* (2016) focussed on cognitive load rather than on confidence or uncertainty estimation and so their methods can't be exactly translated to work within this paper.

Jradi and Frasson (2013) authored a paper investigating how a student's uncertainty could be predicted using a multitude of sensors. They used electroencephalogram (EEG), skin conductance (SC), and blood volume pulse (BVP) sensors to gather various pieces of information. This paper found that this combination of sensors proved to be highly effective at predicting user uncertainty which could in turn be extended to monitor the student's confidence levels too. However, although this combination provided accurate predictions, placing BVP and SC sensors can be invasive and take a while to set up, whereas with the "growing progress in developing portable, convenient, and low-cost EEG headsets" it is more feasible to operate within a greater degree of environments without requiring complex setup operations. As a result, as a part of this paper, an EEG helmet work make sense to potentially accompany an eye-tracker as an additive measure when predicting user confidence values.

Another article that studied the prediction of confidence values was performed by Roderer and Roebers (2010). This paper found that "During recognition, age-dependent patterns of fixation time allocation were found for easy versus difficult items" as shown in Figure 2. This essentially means that there was a pattern between the fixation duration and whether the participant found the task challenging or not. Although an extremely useful discovery for

monitoring how difficult a user may find the task, this approach struggled to differentiate when the users were completing a task of middling difficulty. Furthermore, due to the limited scope of this approach, many details of the eye's motion are missed, however these were covered in the paper by Smith *et al.* (2018).

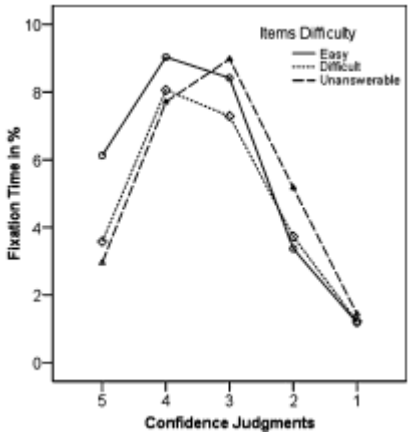

Figure 2 - Roderer and Roebers (2010) Graph

2.3 Visual decision making

As a part of the decision-making process, it is common for most of the information gathering to be completed visually as those trying to solve the problem will study the issue using their vision to quickly assess the situation. A practical example of this is driving; whenever someone is driving, they are primarily using their sight alongside their previous training to assess the road and traffic conditions followed by them deciding what to do next.

One of the paper's inspecting how visual decision making was completed was written by Ais *et al.* (2016). Ais *et al.* monitored how an individual's confidence varied across multiple sessions when completing similar tasks. They found that everyone had, what they called, a "subjective fingerprint" when completing tasks of the same form. This meant that, when the participants completed contextually similar tasks, they would produce a very similar distribution of confidence scores as their confidence "fingerprint" would cause their cognitive function to act in a very similar fashion when doing the same thing.

An article by Grimaldi *et al.* (2015) collated and analysed research regarding confidence, decision making in the brain, and how they interact with one another. They found evidence supporting a range of hypotheses, one of which was that confidence and decision making are

linked within the brain, however it was also theorised that they are instead generated as part of a shared loop and so it is not surprising that the two values were seen within the same areas of the brain when monitored. If it is the case that both tasks are completed in the same, or closely related areas, it's useful to know that the brain may be lighting up in the area whilst collecting data from an EEG device.

2.4 Previous EEG usage

In addition to using a consumer grade eye tracker, this paper has the potential to use an EEG headset as further means to predict user confidence and distraction. A previous paper by Mohamed *et al.* (2018) investigated the prospect of using a laboratory-spec, 14-channel wearable EEG headset to detect the attention and memory of the wearers. This paper demonstrated that during a cognitive test, they were able to predict the three levels of focused attention under controlled conditions. This shows that it's possible to detect a user's attention using just an EEG headset, and so it will be worth investigating to see whether a consumer grade alternative can produce comparable results.

Another study by Maskeliunas *et al.* (2016) completed a deep dive into the use of consumer-grade EEG devices and how accurate their measurements are during concentration tasks and when blinking, they tested two devices against each other which are shown below in Figure 3, one of which was a predecessor to the model of EEG which will be used in this paper. After being suitably tested, it was found that the EEG devices had significant problems due to the technical limitations and the weak feedback they could detect. To paraphrase the conclusion of the paper, it was recommended that a combination of EEG readings, combined with other inputs such as gaze, body posture and facial expressions should be considered to create a more accurate measurement, from this it makes sense to include gaze data as another medium of detection. It should be noted that the MindWave headset, used in this paper, was a predecessor who was released in 2010/11, rather than the MindWave Mobile 2 which was released in June 2018 (NeuroSky, 2018). As there has been a significant gap between their releases, this headset has the potential to be much more accurate and stable for recording data.

Figure 3 - MindWave headset shown left, Emotiv EPOC shown right.

2.5 The specifics of measuring gaze data

As previously detailed, the paper by Smith *et al.* (2018) experimented with a high-end eye tracker to predict confidence of users during a visual decision-making task. In order to begin recording data from the Tobii eye-tracker, an understanding of what to record is required first and so an analysis of the process used in the paper by Smith *et al.* (2018) is provided below. As the users completed their labelling tasks, they provided a confidence score between one and five whilst their gaze was recorded. A variety of methods for predicting the user's confidence were produced including the use of summary statistics, which included the mean duration of fixation on each place of interest, count of stimuli fixations and time taken to make a decision. All of these features were selected and derived manually from the dataset and prove themselves to be an accurate measure of confidence, however deriving these values is incredibly time consuming and so a faster method would be more suitable for the scope of this project.

Another method used to predict confidence, was to generate an image called a gazemap. A gazemap is a compilation of points which correspond to the user's gaze and fixations during their labelling attempts; an example of the gazemap from the paper by Smith *et al.* (2018) is below in Figure 4. These images were able to represent values such as fixation time length,

inter-fixation duration and what kind of feature a user is looking at by using simple features like point size, line width and point shape to change the gazemaps design. From this, using a convolutional neural network, Smith *et al.* was able to accurately predict the confidence of the users to an extremely similar level when compared to summary statistics. As this method is much easier to transfer to a newer system, as is the case with this report, it makes sense for this report to implement this method rather than deriving individual statistics from the collected data.

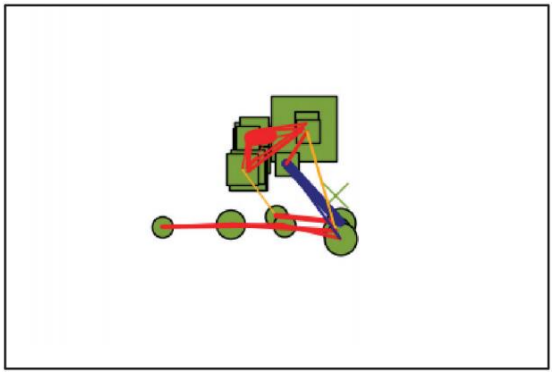

Figure 4 - Smith *et al.* (2018) gazemap

NEEDS TO DISCUSS ACTIVATE

2.6 Chapter summary

what new papers did he find
I hadn't cited already?

In summary, this chapter has explored a range of studies and articles which relate to this project. This chapter has covered a variety of topics, starting at the general background information around this project in which the primary source of this project was outline and briefly investigated. Following off from this, the next section took a deeper dive into the detection of user confidence via various methods such as the use of EEG and eye monitors in an attempt to understand previous attempts to automate the prediction process. The third section focussed specifically on visual decision making and how it linked to the brain's ability to produce a feeling of confidence as well as its proximity to one another. Finally, a deeper dive took place, investigating the specifics of how an EEG headset and an eye tracker can be used to predict participant confidence at a slightly closer attention to detail.

3 Requirements

This chapter will identify several formal requirements for the application using the MoSCoW requirements capture method, which are then split into functional and non-functional requirements following the below definitions from the book by Sommerville (2016):

- **Functional requirements:** These are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.
- **Non-functional requirements:** These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, and constraints imposed by standards. Non-functional requirements often apply to the system as a whole rather than individual system features or services.

3.1 Functional requirements

F 1. The tool must use a current version of TensorFlow.

F 2. This tool must automatically connect to the eye tracker.

F 3. The tool must collect data from the eye tracker.

F 4. The tool must generate a gazemap of the user labelling images.

F 5. The tool should train a convolutional neural network using the gazemaps as data input.

MUST be able to predict in real time?

F 6. The tool could train to predict the confidence values of the users in real time.

F 7. The tool could interface with the MindWave headset.

F 8. The tool could detect basic brain activity using the MindWave headset.

F 9. The tool could store CNN models for predicting future tasks.

F 10. The tool could be scalable to work with different datasets.

3.2 Non-functional requirements

NF 1. The tool must be easy to setup and start on a window's computer.

NF 2. The tool should be easy to maintain and easy to update.

NF 3. The tool should be simple to use and navigate.

NF 4. The tool should start quickly to start recording eye tracking data.

NF 5. The tool should come with documentation to start the program with ease.

NF 6. The tool should scale to work with different data collection lengths.

NF 7. The tool should successfully implement a convolutional neural network.

NF 8. The tool should collect fixation data when the fixation length is greater than 0.1 seconds.

NF 9. The tool could produce reliable confidence scores.

NF 10. The tool could run on Windows, Mac OS, and Linux operation systems.

NF 11. The tool could record movement data based on a window's position.

NF 12. The tool won't store eye positional data past processing.

VERY SHORT - methodology rather than actuall design

4 Design Methodology

This chapter of the report depicts the design methodology that underpins the development of this project. During this chapter, the software development techniques that will be used will be explained and justified, followed a brief outline of the ethical considerations. Afterwards, the testing process shall be explained before the time planning process is revealed to round off the chapter.

4.1 Software Development

During chapter one of this report, the general aims and objectives were laid out to help guide the development of this tool, which was then paired with the requirements in chapter three to detail the key functionality. With this in consideration, it was decided that using multiple elements from the Agile software development methodology as listed by Sommerville (2016) would be best. The first method which was used during the development was a Kanban board, which is a looser version of a Scrum board; this tool allowed the student to break down the delivery of software into manageable tickets, whilst also allowing for new ideas to be included and accounted for with ease with minimal planning externally. As development continued, each aspect of the project was gradually broken down into smaller and smaller tasks, with stretch goals and priorities being assigned to each on a case-by-case basis to help deliver the product in a progressive manner. Provided below is a screenshot of the in-progress Kanban board inside of Figure 5:

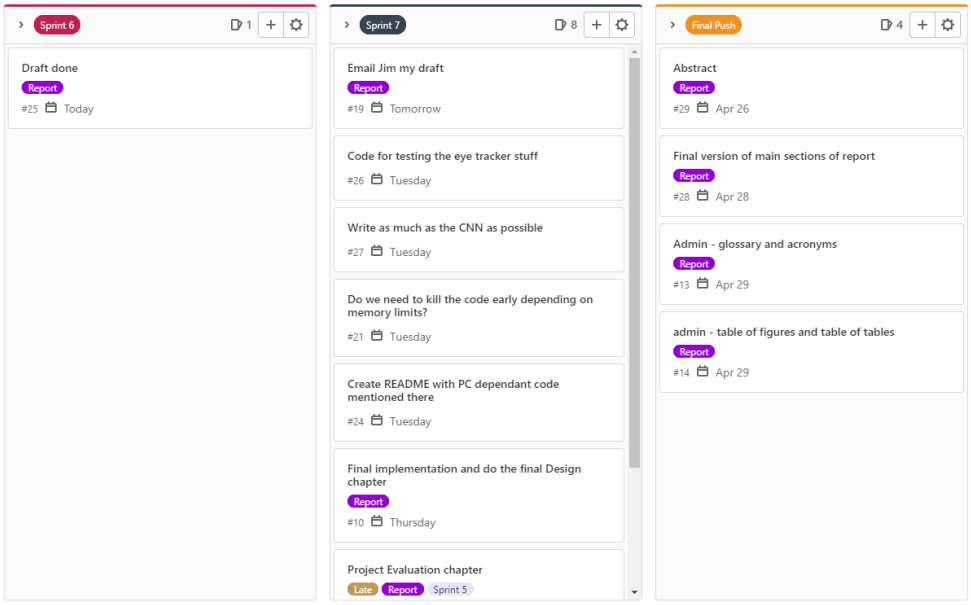

Figure 5 - Kanban board

As shown in Figure 5, the student decided that the use of short sprints would be highly advantageous for developing different iterations of the tool as more research is completed. Using the Kanban board, at the beginning of each sprint, several tasks relating to the product and this report were allocated to the sprint to be completed by the end of said week. These tasks were much more manageable to complete in comparison to the overall task and so this method is very useful for both manging time and expectations. Using this divide and conquer tactic, it allowed for each day's goals to be visualised and thus allowed for the objectives to be achieved on time without over-working becoming an overbearing issue.

4.2 Ethical Considerations

As a part of this report, an ethical review checklist was completed to determine whether a full ethical review would be required as part of the project. It was deemed that the project was a low risk and thus wouldn't require a full ethical review provided the project remained on its current path. As the code won't store information on the user's, nor does it store eye tracking information past processing, it was deemed that there was low risk of any ethical concerns.

4.3 Testing procedure

During the design of this program, test cases were written to testing the functionality of certain elements of the codebase, ensuring they work in both isolation and combination. These tests cases are defined in Appendix A. In future iterations, it may prove useful to automate this process using a testing framework, although tests that require human input will still require intervention. In order fully test the gazemap without using ActiVAte and the eye tracker, an output faking program was required. This application contained a whole host of variables which can be altered and shifted to produce both gaze data and confidence data for labelled images to test the gazemap production which was very useful during development.

4.4 Time Planning

Although a Kanban board was used to plot individual tasks alongside week-long, short development sprints, a Gantt chart was used in addition to help plan the work that took place across the whole year. The aim of this tool was to provide a macro look at the development across the whole year.

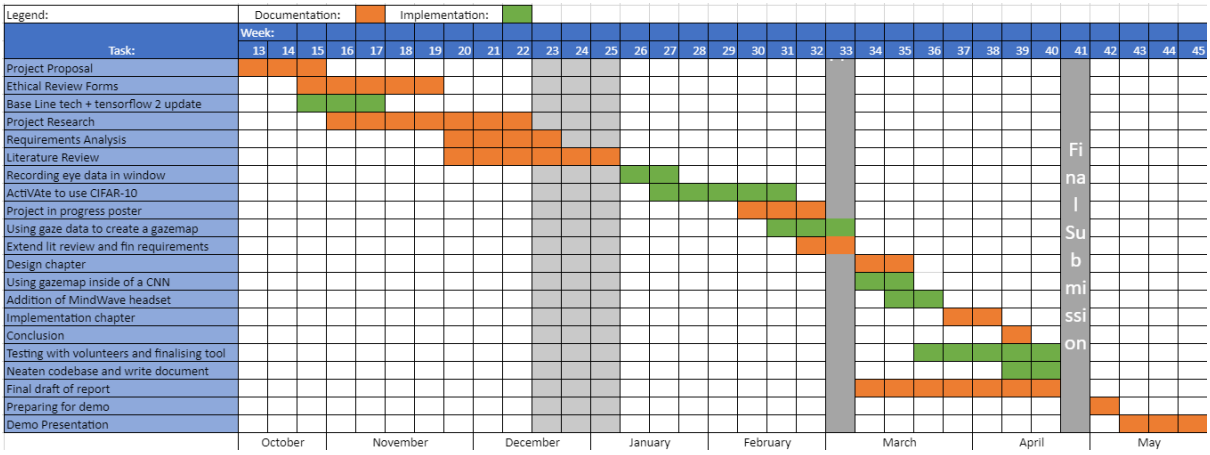

Figure 6 - Gantt chart

4.5 Chapter Summary

In summary, this chapter covered the over-arching design methodology and the choices that took place in the background prior to work beginning on the work. The first section covered the software development strategies that were being used, which demonstrated an

emphasis on the Agile software development methodology. Shortly afterwards, the ethical considerations were briefly mentioned, but due to low-risk factor, this was mostly skimmed over. Finally, the testing procedure was detailed in order to allow readers to gain perspective regarding the process that took place during development to ensure a proper product was made.

5 Program Design

This chapter aims to detail the general program design, detailing design choices and limitations it came under. To begin with, the pre-existing programs and tools will be explored, covering ActiVAte and the libraries provided by Tobii for use with their eye tracker. Moving on, the use of programming languages and development environments will be discussed, highlighting some of the good details about them. Finally, part of the system architecture and general flow of the program will be displayed to gain a greater understanding of the engine running under this tool.

5.1 Pre-existing Tools

5.1.1 ActiVAte and Gazemap generation

As this project continued on from the previous work by Smith *et al.* (2018), there were prior pieces of software which were already developed and thus could expanded on top of to support this project aims. This main piece of software was called ActiVAte, which is fully detailed inside of the report by Legg, Smith, and Downing (2019). Firstly, ActiVAte needed to be refracted as it was using an older version of TensorFlow which contained outdated functionality, alongside multiple hard-coded values for the dataset which was replaced with as much automation as possible to allow for simple updates. Secondly, ActiVAte required a clean-up in a few places as there were a few unnecessary older files which increased the difficulty of maintaining the project, as well as multiple system dependant lines of code which were changed to their system independent counterparts.

When considering the gazemap generation, the student decided that using an adapted version of the original code used during the Smith *et al.* (2018) paper would be the best approach. The output of the Tobii eye tracker could easily be adapted to use with this code, but it would require some modifications as things like pupil width would need to be removed as it can't be recorded on the Tobii device.

5.1.2 Tobii Product Integration Libraries

Alongside their eye tracker, Tobii provide multiple code libraries which let developers create applications using their hardware. There were three options available to be used, which were listed on the Tobii Developer Zone site. The first of which was the Tobii Stream Engine, this low-level SDK is designed to allow tight control over interaction and produced exact coordinates of where the user was looking during usage. Although responsive and fast, it was decided that this SDK wasn't appropriate for the task as deciding what counted as a fixation on a particular point was going to be particularly difficult and thus out of scope for this project. Much like the second library, this SDK is only available in C and C++.

The second library listed on the Tobii website was the Interaction Library; a robust and lightweight library which allowed for certain areas on screen to be nominated as areas of interest. This library would be much more suited to the task at hand as being able to quickly and easily determine where the user is looking at a source of stimulus or would be highly advantageous during gazemap generation. Furthermore, the library is portable and works with Windows (64-bit), MacOS (64-bit) and Linux (64-bit) operating systems. However, the one weakness is that the library only support C++, C and C# as options and so required additional development to work with ActiVAte, as that would need to run separately on Python.

Finally, the Tobii website links a tutorial to work with UWP applications as a developer, however this is designed to work with UWP windows application and thus wasn't applicable for this project.

5.2 Development Tools

5.2.1 Programming Languages

As mentioned in section 5.1.2, ActiVAte was built using Python as the primary language, with support from HTML, CSS and JavaScript providing a front end as a Flask application on the local host. Due to the highly data-driven nature of this project, and the pre-existing sections of code, it made sense to continue using Python as the primary programming language as it's both concise and can fulfil most of the program's needs. Within python, there

are modules, such as NumPy, TensorFlow and Matplotlib, which are readily available to developers to allow them to manipulate data as well as train and display models. When looking at some of the specific requirements of this project, in order to predict the confidence of the users, a Convolutional Neural Network would be required to process the gazemaps and so using the TensorFlow module was highly advised due to its simple, yet powerful design for this specific task.

Alongside ActiVAte, this project also required development of a program which would work with the eye tracker. Due to the limitations emplaced by the eye tracker library, which was discussed in section 5.1.2, this part of the application will be developed using C++. Due to the mixing of languages, a form of communication will be required between the C++ and Python which will be discussed in greater detail later in this chapter. Although a required language, C++ also allows for an object-orientated design when dealing with the data from the tracker, which creates clean and functional code. On top of this, C++ allows for a development in which the tracker's performance hit is minimal as it allows for close control on the memory usage from the program.

5.2.2 Programming Environments

The development of ActiVAte, the gazemap generation tool and the convolutional neural network will all require the use of Python and a sufficient coding environment. It was decided that PyCharm would be the ideal IDE due to its vast number of features and its availability for use by students. One of the key features that PyCharm provides is the ability to easily create and edit Python Virtual Environments (venv), whilst providing the ability to store the module versions so they can be run on another system later. This was particularly useful during the early stages of development as updating ActiVAte to work with newer versions of TensorFlow required constant flicking between environments.

To develop the eye tracker section of the program, another development environment would be required to provide a development space as well as a simple to use compiler. After experimenting with multiple tools including Visual Studio 2019 and Code::Blocks, it was found that Visual Studio Code would be the best fit due to its dynamic build system. Using the build system, it was much simple to add the custom header files from the Tobii Interaction

Library. Unfortunately, as Visual Studio 2019 for Windows provided the compiler, there may be compilation issues when using this code with a different compiler on another operating system in future iterations, from this, it could be theorised that using a universal build system, such as CMake, may be a better alternative to this solution, although this system still comes with trade-offs relating to time and maintain this section of code.

5.3 System Architecture

The system as a whole is made up of multiple constituent parts, all working independently to provide different services and functionality. During the early stages of development, to help picture the design of the program, a flow diagram was created as shown in Figure 7. This simple flow diagram allowed the student to being picturing the overall design of the tool and what work would be needed to meet the requirements.

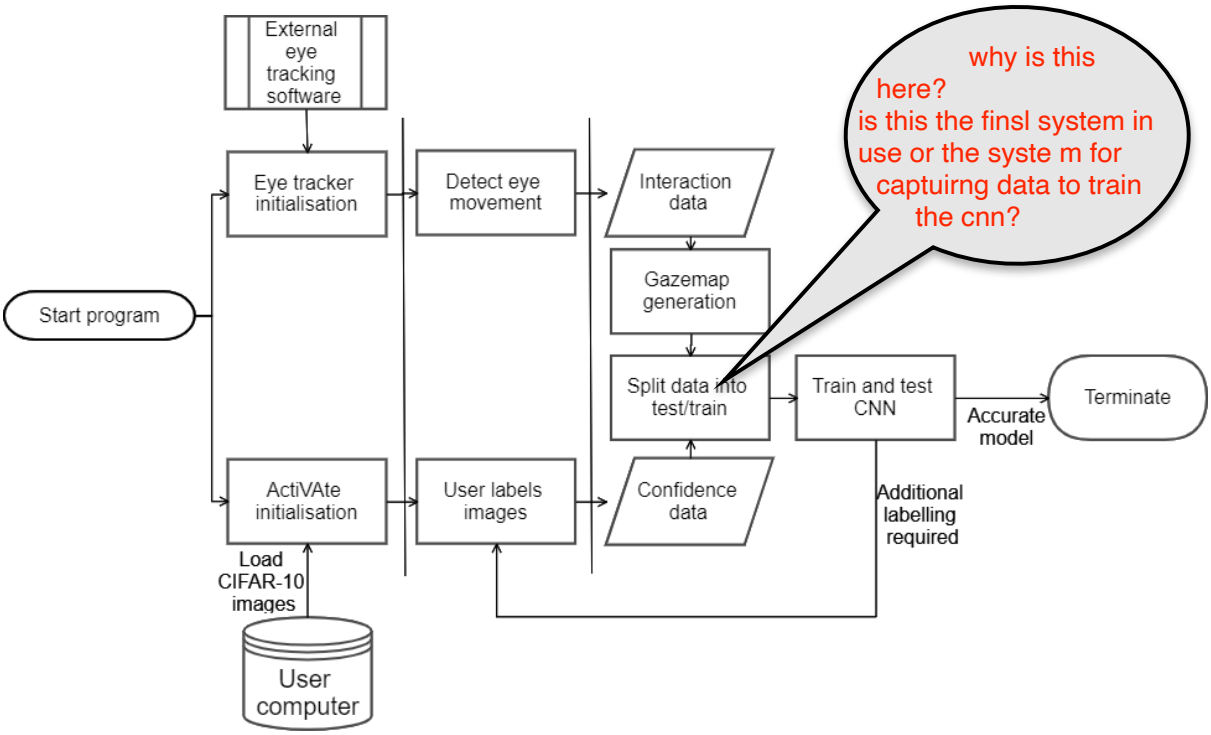

Figure 7 - Pre-design flow diagram

Towards the end of the development time, another flow diagram was created as shown below in Figure 8. Although similar, due to the multiple design constraints which are discussed later in section 6, there had to be a fair few change in order to bring this tool to life. The largest change being a separation of form, where the gazemap and CNN are all separate

tools to the labelling within the project, however, it's possible these could be added to the overarching script for ActiVAte if it's wished that these programs run in series after the labelling finishes.

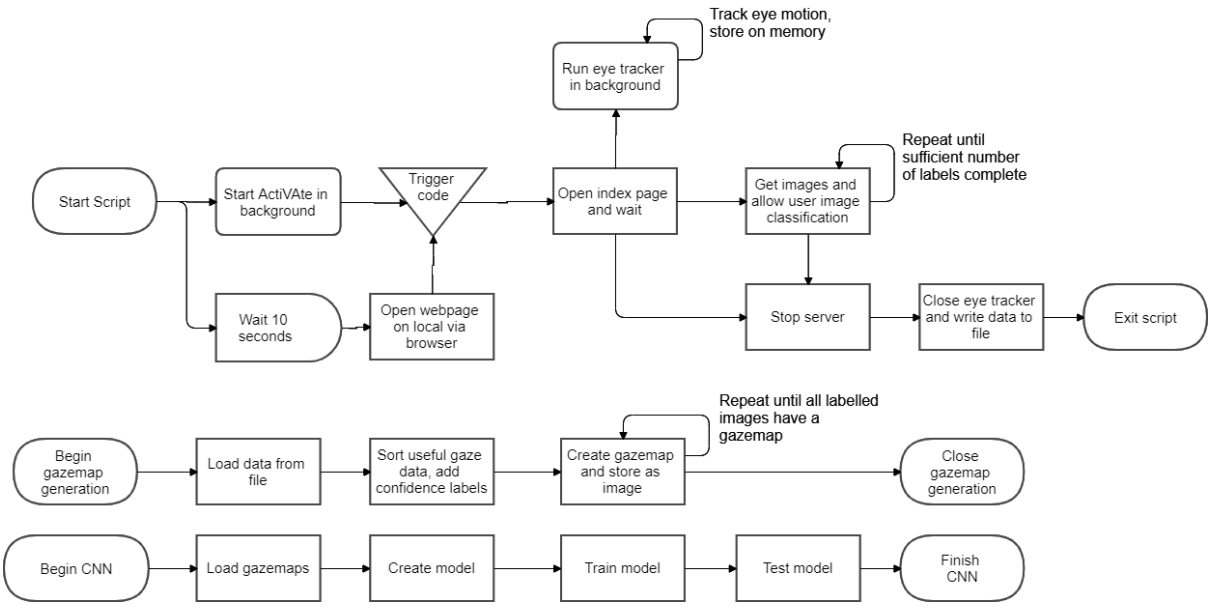

Figure 8 - Finished program flow diagram

5.4 Eye Tracker Program Structure

5.4.1 Class and Data Structure Design

Classes and how data is handled for optimal use? Class diagram?

As described in section 5.2.1, C++ was required to work with the eye tracker due to library limitations. However, with the language, there came a chance to optimise the data usage to minimise the performance of the program. In order to read data from the eye tracker, the program must subscribe to the data stream in a simple loop format. During the running of this loop, whenever data is output by the tracker, a simple object is created and written to a vector where it is held on memory until completion of the tracking process. The GazeEvent class of the object is shown below in Figure 9, this minimal design meant that the eye tracker didn't run up a huge memory usage as described in the final iteration (section Final Iteration6.5).

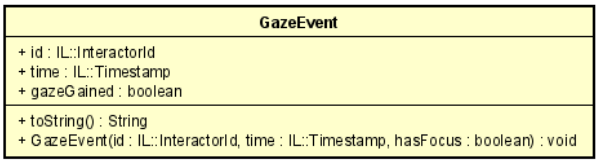

Figure 9 - GazeEvent class diagram

5.4.2 Program Sequences

This section will briefly detail a typical run through of ActiVAte and the eye tracker. As the script is initiated, ActiVAte is initiated in the background causing the script to wait ten seconds. After this time has passed, it will then open a webpage in the browser that is being hosted by ActiVAte. The script part ends here as the program is then reliant on ActiAVte. As the webpage opens, ActiVAte begins the eye tracker as a subprocess running underneath it, which remains idle until the user requests images to label. At this point ActiVAte creates a file to start the eye tracker and returns the images to be labelled to the server. As the user labels images, the data is recorded in a file to be used during the gazemap generation. Both ActiVAte and the eye tracker continue to run until the user is done labelling images, where they then shutdown the server. ActiVAte destroys the trigger file, causing the eye tracker to stop tracking and instead write its data to file for use later, before quickly shutting and sending a signal to ActiVAte to shut down too. This process is also shown below in Figure 10.

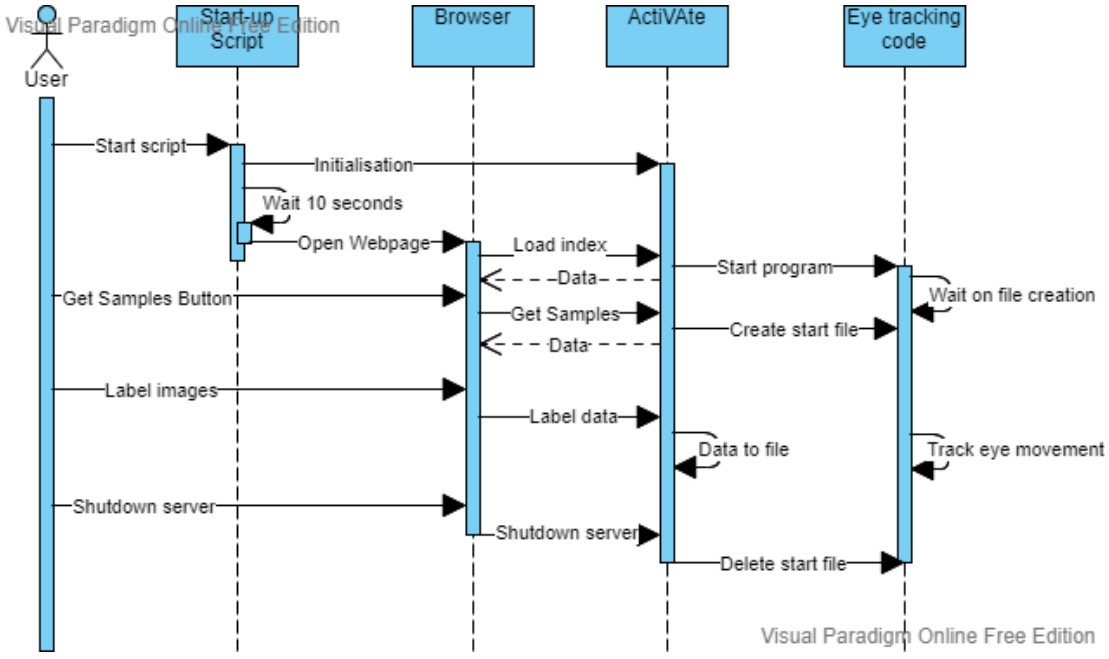

Figure 10 - ActiAVte sequence diagram

5.5 Chapter Summary

This chapter has covered the overall design of the program. The first section focussing on ActiVAte and the work that needed to be done on it and the limitations being presented by the eye tracker, which included the language limitations. How these issues were progressively combatted are discussed in the implementation section below. Finally, the architecture of the program was discussed, including a comparison of the original design plans in contrast to the end product.

6 Implementation

This chapter will cover the code's implementation, walking through the code's design choices from the proof of concept to the final design. Inside of the various sections, the code's limitations and how they meet the requirements will be discussed to gain a greater insight into how the development cycles took place. Finally, certain limitations will be discussed before the requirements are checked through at the end of this chapter.

6.1 Proof of Concept

As development of the product began, a first iteration was required in order to demonstrate the baseline technologies which were going to be used during development. This consisted of two major sections of work, the first of which was to utilise the libraries provided by Tobii to track eye fixations to begin generating gazemaps. Using the Interaction Library described in section 5.1.2, it's possible to create interactors in the shape of rectangles which detects eye motion that moves into and out of the defined region. Combining multiple interactors across the screen, it is possible to create a grid which can record when eye movement fixates on different regions within the screen. The granularity of the grid can easily be varied by increasing the number of interactor objects on screen. Using this method, it was possible to generate the first basic gazemap over a twenty second time step as shown below in Figure 11

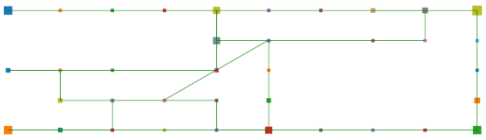

Figure 11 - Gazemap proof of concept

This method had multiple limitations including the inability to interact with ActiVAte, nor detect when ActiVAte was up and running. This method was also unable to detect what sort of thing the user was looking at and so all fixations were temporarily assigned to be stimulants during gazemap generation.

The second section of work required an update to ActiVAte to bring the code in line with modern TensorFlow 2.x, as deprecated functionality was being used which caused errors within the interpreter. Alongside this update, the way in which the dataset was loaded was updated to reflect the modern approach, which fortunately allows ActiVAte's under laying dataset to be updated with ease, as it can now be swapped for any other TensorFlow Keras dataset in one place rather than many. Finally, as part of this update, the images from the new dataset needed to be downloaded to be stored locally for ActiVAte to display them online, but the correct image dataset repository for CIFAR-10 had become unavailable online. A multi-use mini application was created to load and then save the dataset to the local file structure which could be used again to download a separate dataset if required.

This proof of concept already began to tackle some of the requirements at the base of this project. Requirement F 1 is met here as the application will automatically connect to the eye tracker provided it's both plugged in and turned on, on top of this, requirement F 3 is met as the interaction library's implementation means the data from the eye tracker can be gathered and analysed during usage. These requirements will be discussed in greater details towards the end of the chapter after the final implementation is documented. This iteration was taken place under sprint one and a significant portion of pre-sprint work.

6.2 Testing Implementation

As a part of the early development stage, a small program was created to test the gazemap generator. This tool replicated the labelling of images within ActiVAte whilst the eye tracker would be running; it produces the same files the programs would and so they can be used to create gazemaps as a result, although the output from the gazemap generator wouldn't be very good to test out the CNN due to their random nature rather than direct input from a human.

6.3 First Iteration

As development started on the first iteration, a meeting was had to discuss the most needed features for the application. The first of which was to set up the code so that both the eye tracker and ActiVAte worked in tandem to create gazemaps for each labelled image. As

this was the first iteration, simplicity was a driving factor when designing the code. It was decided that a batch script would be used to initiate the two programs together, using primitive manual commands to start the eye tracker code when required. During this step, the granularity of the gazemaps were stepped up and they could create much more accurate models as shown below in Figure 12. This increase in granularity meant that the produced gazemaps are a lot close to the ones shown in the paper by Smith *et al.* (2018).

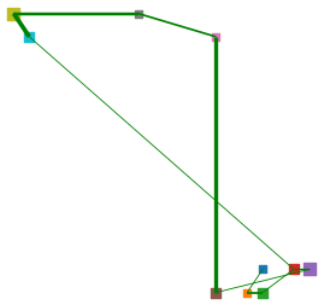

Figure 12 - First iteration gazemap

Although this method could generate gazemaps, it had significant drawbacks, the largest being that the eye tracker could only run for a set length of time which meant it wouldn't meant requirement NF 6 as that required a dynamic run-time to meet this standard. On top of this, the fact that the tracker had to be started manually meant that tool wasn't very user friendly and thus would hard to start, not meeting NF 3. These issues were further investigated in later iteration after the basic functionality was implemented during this stage.

It should be noted that the timing of the eye tracker was investigated here, as the timer is based around microseconds ($1 \times 10^{-6} s$), this needed to relate this to the system's clock to show when the program started and to compare these eye tracker timings to the labelled images from ActiVAte. The way in which timings are calculated on each computer do vary, however as a fixation is only counted as an event when the gaze is longer than 0.1 seconds, we can discount this error as trivial in comparison.

Finally, it should be mentioned that the gazemap generation was further developed at this stage so that a gazemap could be generated from the output from the eye tracker for each labelled image. This works using a separate program to ActiVAte and can be run after the

labelling has finished. This development works to fulfil requirement F 4. This work took place over sprints two to four.

6.4 Second Iteration

During the second iteration's production, the primary focus was to remove the ill-functioning initialisation of the eye tracker, as well as allowing for it to run continuously until the ActiVAte user stops labelling the images and shuts the program down. To do this, the python code with ActiVAte would need to both initialise, signal and shutdown the eye tracking code. It was decided that the subprocess module would be best here as the eye tracking program could be started in the background of ActiVAte, where it would simply wait on a prompt from ActiVAte to begin recording data, before finally writing the data to file after ActiVAte's server is shutdown.

As mentioned in the previous paragraph, in order to begin gathering eye tracking data, a signal needed to be passed between ActiVAte and it's subprocess. The first thing the student considered was that the subprocess module had a function called send_signal which allowed a signal to be sent to the subprocess, however, to receive this signal, the eye tracker would have needed some OS dependant functionality which should be avoided where possible. As a result, a much simpler system was employed using a file's creation and destruction to indicate when to begin and halt the recording of the eye tracker.

Although a simple process, this design was created with multiple requirements in mind, namely: NF 3, NF 4, NF 6 and NF 10. What this meant is that the app is easy to use as possible for someone with basic technical knowledge, it quickly begins recording eye positional data when required, the data collection scales for as long as the participants are required to label images and finally this dual operation should work on Linux, Mac OS, and Windows systems.

Regarding NF 10, the requirement to have the code potentially run on the three primary operating systems, during this development stage, multiple changes to the app took place to standardise the code. One point of which took place inside of ActiVAte in which all file paths were standardised to work with all operating systems using Path from the pathlib module within python. This work took place over sprints four and five.

6.5 Final Iteration

During the final few sprints, working up to the final iteration, one of the questions raised regarded the eye tracker and its memory usage. As the program looped whilst the participant labelled images, more and more gaze events are stored in memory and so there was a concern that the eye tracker may only be able to run for a short period of time due to the memory limitations. The eye tracker was tested by having 150 images labelled, which took just under 10 minutes to complete, using the windows resource monitor the student was able to witness the computer's memory usage over time. It was found that the memory usage looked as if it plateaued, however when the code finished, there was a noticeable drop of approximately 20% of 32gb of RAM as shown below in Figure 13. There are a number of potential causes here which will be looked into during the evaluation chapter.

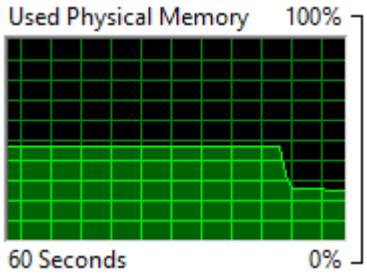

Figure 13 - Eye tracker memory usage

Another part of the final iteration was to create a README file which could be used by a new individual to utilise this tool. The README contained details of the Python interpreter and the C++ compiler which were used during the project, including pointers to recreate the same environment that was used. Following this, the four sections of code are described and the process to update the dataset used with in ActiVAte is detailed to. This README should help any future users to utilise the project and its available functionality.

The largest section of work completed during this time focussed on the convolutional neural network. In order to work with the CNN, it was decided that implementing it to predict in a binary fashion would be more appropriate for a starting design. To achieve this, the confidences file from ActiVAte required work to change the confidence score to be a binary value rather than a continuous value. As the CNN was a starter concept to be built on, a simple split of 80% training data and 20% testing data was created just to see if the CNN would be

possible, however this could be changed within the gazemap generator to provide a different split. The accuracy of the model will be discussed in the evaluation section below. An output of one run is shown below in Figure 14.

```
Epoch 1/8
2021-04-26 14:55:24.930580: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library cub
4/4 [==============================] - 1s 285ms/step - loss: 92.6943 - accuracy: 0.7361 - val_loss: 87.3126 - val_accuracy: 0.8000
WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at leas
Epoch 2/8
4/4 [==============================] - 1s 169ms/step - loss: 36.6711 - accuracy: 0.6944
Epoch 3/8
4/4 [==============================] - 1s 170ms/step - loss: 0.9627 - accuracy: 0.5556
Epoch 4/8
4/4 [==============================] - 1s 147ms/step - loss: 0.6932 - accuracy: 0.4306
Epoch 5/8
4/4 [==============================] - 1s 168ms/step - loss: 0.6926 - accuracy: 0.7917
Epoch 6/8
4/4 [==============================] - 1s 147ms/step - loss: 0.6917 - accuracy: 0.8194
Epoch 7/8
4/4 [==============================] - 1s 146ms/step - loss: 0.6907 - accuracy: 0.8194
Epoch 8/8
4/4 [==============================] - 1s 146ms/step - loss: 0.6897 - accuracy: 0.8056
```

Figure 14 - Output of CNN program

6.6 Development Considerations

During development of the application there were a number of running concerns which will be detailed here. The first of which regards the eye tracker. Due to the high frequency access speeds of the tracker, it was taken into consideration that writing the gaze data to memory would likely be a better bet using the GazeEvent class which was described in Figure 9. The alternative would have required many more reads from memory as well as writing to file on the hard drive which is renowned for being a lot slower than memory. As a result, the design choice to store all gaze events on memory until the labelling finished remained in place to prevent a bottleneck. From this, a memory allocation question was raised, however as described in the last section, there was a minimal impact on the memory storage over a ten-minute period.

Due to the requirement "NF 10 - The tool could run on Windows, Mac OS, and Linux operation systems." There were multiple design choices and limitations that had to take place, including tracking the window position during the first iteration. As a result of this, functionality to the read the positional data of ActiVAte's browser window couldn't be implemented due to the OS reliance. From this, requirement NF 11 couldn't be completed without compromising NF 10. From this, the tracker instead covers the whole screen.

Finally, according to NF 2 - The tool should be easy to maintain and easy to update. From this requirement, multiple refractors of existing code and clever use of variables was required to both limit the amount of repeated code, as well as reducing the number of hard-coded values within it.

6.7 Requirements

This section will cover the requirements and will discuss whether the implementation has reached them. For easier reference, the requirements have been re-written here:

- F 1 The tool must use a current version of TensorFlow.
- F 2 This tool must automatically connect to the eye tracker.
- F 3 The tool must collect data from the eye tracker.
- F 4 The tool must generate a gazemap of the user labelling images.
- F 5 The tool should train a convolutional neural network using the gazemaps as data input.
- F 6 The tool could train to predict the confidence values of the users in real time.
- F 7 The tool could interface with the MindWave headset.
- F 8 The tool could detect basic brain activity using the MindWave headset.
- F 9 The tool could store CNN models for predicting future tasks.
- F 10 The tool could be scalable to work with different datasets.
- NF 1 The tool must be easy to setup and start on a window's computer.
- NF 2 The tool should be easy to maintain and easy to update.
- NF 3 The tool should be simple to use and navigate.
- NF 4 The tool should start quickly to start recording eye tracking data.
- NF 5 The tool should come with documentation to start the program with ease.
- NF 6 The tool should scale to work with different data collection lengths.
- NF 7 The tool should successfully implement a convolutional neural network.
- NF 8 The tool should collect fixation data when the fixation length is greater than 0.1
- NF 9 The tool could produce reliable confidence scores.
- NF 10 The tool could run on Windows, Mac OS, and Linux operation systems.
- NF 11 The tool could record movement data based on a window's position.
- NF 12 The tool won't store eye positional data past processing.

Working from the top down, we can say the 4 functional musts have been met successfully as ActiVAte now works with TensorFlow 2.x, turns on, connects the eye tracker automatically and can produce gazemaps after it finishes. F 5 has been met, as the convolutional neural network now trains using the gazemaps, however F 9 couldn't be met on time and so it fell

outside of scope. Due to the current design of the eye tracker, the gaze data isn't being written out during the tracking and so, fulfilling F 6 would be particularly hard unless it stopped occasionally to write to file. Both F 7 and F 8 required interfacing with the MindWave headset, however this part may have been very ambitious and so will be discussed later in the project evaluation. F 10 was a fairly straight forward requirement after ActiVAte was updated to work with CIFAR-10 and TensorFlow 2.x, as a result simply downloading the new image dataset, providing the class names to the JavaScript, and updating ActiVAte's config file as detailed in the README will quickly allow for a different dataset to be used.

Moving onto the non-functional requirements, significantly more success was had. Requirements NF 1 through NF 8, and NF 10 to NF 12 were all met by the end of the production cycles, however NF 9 would be debateable as although a predictable confidence score can be produced after training the model, these aren't overly accurate due to the lack of tweaking of the model.

6.8 Chapter Summary

This chapter covered the implementation of this project, beginning with the proof of concept many months, progressing through the development iterations until finally reaching the final implementation. This chapter has covered many limitations of the design, discussing things that were out of scope and finally discussing whether this application has reached all of the requirements. Whether the requirements have been met or not, they will be further discussed in the next chapter when the project as a whole will be evaluated.

7 Project evaluation

This is the penultimate chapter of this report and will be covering the project's evaluation. This section will break down many sections of this report, the development of the codebase and whether this project managed to meet the aims and requirements set out towards the beginning of this report.

7.1 Project Management

The management of this project was a mixed affair. There were a whole host of limitations and factors that effected the work, but all of the planning that took place towards the start of this project really helped in the long run to ensure work was done on time. At the beginning, a Gantt chart was created; this chart aimed to cover the over-arching work which was lined up for completion over the next few months. Although incredibly useful towards the beginning of this project to get things started, due to the macro implementation, its usefulness fell off towards the end of the project in favour of the Kanban board instead as it allowed for a much more granular approach. From this thought, it would be interesting to see how a Gantt chart compares to a Kanban board on a much shorter time scale, or whether the rigidity or the Gantt chart will still be too large of an issue.

Trying to find the right balance for time allocation to this project was very difficult. Although the Gantt chart allowed for task allocation and a simple way to set deadlines, it was much more difficult once it got to actually doing the work, especially in the first semester due to the overloading of modules and deadlines around this period. As a result, it would make sense to consider the implication of known external forces when planning another project like this, like in this case it would be module deadlines.

During the various planning phases of this project, the meetings were a great aid to steer the project's design as well as assisting with the prioritisation of the implementation. From this, the primary lesson learnt was that communication with supervisors, or with clients is key during development as they will have a good idea of what they want done first, or how to implement certain features.

7.2 Literature Review

The literature review section of this paper allowed the student to gain a significantly greater insight into the paper by Smith *et al.* (2018) and the surrounding topic area. The field itself is ever changing and evolving, so an in-depth literature review was required towards the beginning of this project to allow the student to identify current and previous attempts at using peripherals, such as an eye tracker, to predict user confidence and awareness. One of the most interesting points was raised by Ais *et al.* (2016) regarding the idea everyone has a "subjective fingerprint" which they use when making a decision; attempting to capture the various elements of their 'fingerprint' is a fascinating task.

Overall, the research went well as it allowed the student to gain a great appreciation of the work that had previously taken to derive confidence and other values using high-end eye trackers. From this, they were able to identify the requirements that were going to be needed to successfully utilise the Tobii Eye Tracker as well as beginning to idealise how the MindWave could be used in conjunction.

7.3 Design Methodology

As described in the design methodology chapter, this project utilised a number of agile practises including the use of a Kanban board alongside short sprints. These methodologies were only implemented part way through the project after the student discussed with a supervisor how best to plan their time and distribute their upcoming workload. The student found that the Kanban board was incredibly useful, utilising the provided one within Gitlab to narrow down the required steps to finish each cycle of implementation, as well as being able to set deadlines for completion. Looking forward to the future, utilising a similar board to this would be highly recommended due to the sheer utility of function it brings.

In conjunction, the use of sprints was useful to keep pace through the weeks as it set achievable deadlines over a shorter period of time, however these sprints weren't overly planned, nor were they very similar to those described by Sommerville (2016). In contrast, Sommerville outlined the idea of task allocation among groups during longer periods of time which were usually at least two weeks long. However, these shorter sprints kept things much

more manageable and flexible, plus shorter time frame meant a sense of urgency was carried through development iteration.

Looking at the ethical considerations of this project, little was changed and so the project has clearly maintained it's low-risk status, with the timely removal of gaze data after gazemap generation. Although it would have been good to gather some feedback and data from users regarding the eye tracker, this would have had implications on the ethical clearance of this paper, alongside limitations provided by Covid-19 which limited this idea too.

As a part of the system design, basics tests were written to help with the design of the program which were later adapted and edited to contain steps relevant to the end design. These tests were useful as a framework to define what the program needed to do, but, using an automated tool would have been useful to make the tests more official and would have enforced a clearer pass/fail mark rather than their current informal design.

7.4 Program Design

The design of this project evolved with time, as certain limitations were uncovered, and different ideas were tested out. From the initial design, certain aspects were quite limited as this tool built on top of ActiVAte's already existent design, however using pre-existing software significantly cut down the development time as a GUI was already provided for everything the tool needed to do, as well as providing the entire active learning backend. Although a challenge to take on an existing tool due to its complex design, this unique viewpoint of utilising an existing tool was a great learning experience for the student.

On top of this, due to the programming language barrier presented by the eye tracking library, a mix of programming languages was required to get this tool up and running. This wasn't a problem that the student had faced before and so solving this problem was entirely novel; the knowledge gained from this could potentially be very useful in the future, although avoiding crossing languages would be highly advantageous in most situations due to the difficulties it presents.

Following on from this, the use of the interaction library for the eye tracker allowed for a simple creation of an eye tracking tool due to its simplistic design. After discussing the

alternative with a supervisor, the trying to cluster viewpoints into a fixation would be incredibly tough by comparison if the stream library were chosen instead. Furthermore, the interaction library allows for dynamic implementation depending on the granularity of the tracking required.

Looking at the program's end design, it was clear that the tool ended up as a collection of programs rather than one continuous system as envisioned during the proof-of-concept stage. Although currently disconnected, the programs could be combined using the script to succinctly allow for labelling and tracking, followed by gazemap generation and finally the convolutional neural network could be trained once operational. However, for the purpose of testing and allowing users to only label images, these programs were kept separate for the final submission for testing purposes as shown in section 5.3.

7.5 Implementation

When looking at the implementation of the project, the use of iterative design was incredibly useful to show a clear development of thoughts and ideas as the software was created. This idea of creating milestones after key aspects of the product were added was incredibly useful and is highly recommended to continue using on other development projects. The first proof of concept stage was pivotal in the creation of the tool as it laid the framework from which it was built including the choice of eye tracking library, from this it was learnt that doing your research correctly in this stage makes life a lot easier in the later stages of development.

As mentioned, a few times before, working with previous code was a very different experience when compared to creating something from scratch. Although ActiVAte was an incredibly tool, in order to comply with many of the requirements, an update was need to some of the codebase. The update from TensorFlow 1.x to 2.x brought with it a variety of benefits, including the ability to quickly swap datasets, as well as allowing the student to gain a much greater appreciation of the dataset and how they're constructed. On top of the update to TensorFlow, there were multiple hard-coded values and system dependant paths which were updated to much more friendly system independent lines, however it wasn't a perfect

fix, as certain files like the JavaScript which is loaded on the server requires manual input of the image file location and the image labels from the dataset.

During the second phase of development, it become very apparent that the requirements set out for this project were simply too vast to accommodate during the available time. During this step, it was decided that the headset would simply have to be ignored in favour of the eye tracker as the primary peripheral. This scope shift was quite a shame, but it taught a valuable lesson to not plan larger than what can be accommodated.

As mentioned in the Final Iteration section of the report, the memory usage of the eye tracker was tested using the simple resources monitor tool available on Windows. As the application was developed on a machine with high memory capacity, the memory usage wasn't overly concerning, however on reflection the drop of approximately 6 GB of RAM after memory after running for 10 minutes is quite concerning. From this, it could be concluded that the eye tracker may need a slight redesign, so that the memory flushes to drive intermittently rather than storing all of the gaze data on memory during the run time.

As defined in the requirements, this project required a CNN which could be used to predict the user's confidence provided an input of a gazemap. Although this was implemented, this tool didn't have much in the way of tuning or testing and so was left in a state which lends itself to be manipulated and adapted for future use. Although a shame the model wasn't tuned at all, the tool should be a great basis for future development.

7.6 Aims, objectives and requirements

Overall, the requirements devised at the beginning of this project were created with too broad of an idea in mind. With the time available, finishing all of the functional requirements would have likely left significant holes in the code. On top of this, when trying to work with the headset provided, the connectivity was found to be very temperamental and so creating an application to interface with it may have become too much of a challenge before anything meaningful was created. With all this said and done, the requirements were still challenging and pushed the student to cover a wide range of topics, learning from them all, but potentially reigning in the requirements scope may have been required in order to finish them all.

The aim of this project was as follows: "To test the viability of using a consumer grade eye tracker alongside a readily available EEG headset to predict the confidence and distraction of a user to a level that's comparable to the eye tracker used in the paper by Smith *et al.* (2018)." Much like the requirements, the scope of this research aim was simply too large. Instead, the groundwork was laid to interface with the eye tracker with ActiVAte, but the ability to predict the confidence or distraction of user proved too much to complete during this project's timeframe.

7.7 Improvements and future work

As described in section 6.7, there are a number of functional requirements which weren't met, the most important of which would be interfacing with the MindWave headset. This headset could potentially greatly improve that accuracy and modelling potential of the CNN. After which, further developing and tuning the CNN would be next priority to begin properly predicting the confidence of the users during labelling to finally see whether these tools could be used in place of their much more expensive counterparts.

In order to test the model, future work could use volunteers, provided it's ethically cleared, to label images and then train the model to work with new individuals. It would be interesting to see whether the "subjective fingerprint" could be detected here, with each participant requiring a separate confidence model, or whether a general-purpose model could be designed if enough participants were presented to the system?

It was mentioned in a meeting, that the eye tracker could potentially always be left on, provided the user can consented to this, to continue tracking eye data and predict how confidence the user is when completing other tasks, or to simply turn on whenever the user starts looking at stimulus on screen. However, this may have further memory implications as discussed in the last section and so should be approached with care.

Conclusion

In conclusion, the overall development of the app was a success, with multiple new applications and tools being developed which are ready to be used to test out the usefulness of the Tobii eye tracker in comparison to the research tracker used in the Smith *et al.* (2019) paper. On top of this, this tool should provide a useful platform for adding the MindWave headset in addition. It should also be noted that this project was an incredibly useful learning experience, providing a variety of experience learning about project management and general software development, although it should be said that over-stretching the requirements is something to be aware to prevent things falling out of scope in the future.

References

Ais, J., Zylberberg, A., Barttfield, P. and Sigman, M. (2016) Individual Consistency in the Accuracy and Distribution of Confidence Judgments. *Cognition* [online]. 146, pp. 377-386. [Accessed 29 December 2020].

Chen, F., Zhou, J., Wang, Y., Yu, K., Arshad, S.Z., Khawaji, A. and Conway, D. (2016) *Robust Multimodal Cognitive Load Measurement* [online]. : Springer International Publishing. [Accessed 23 December 2020].

Grimaldi, P., Lau, H. and Basso, M.A. (2015) There Are Things That We Know That We Know, and There Are Things That We Do Not Know We Do Not Know: Confidence in Decision-making. *Neuroscience & Biobehavioural Reviews* [online]. 55, pp. 88-97. [Accessed 29 December 2020].

Jradi, I. and Frasson, C. (2013) Student's Uncertainty Modelling Through a Multimodal Sensor-based Approach. *Journal of Educational Technology & Society* [online]. 16 (1), pp. 219-230. [Accessed 23 December 2020].

Krizhevsky, A. (2009) Learning Multiple Layers of Features From Tiny Images. [online]. [Accessed 16 February 2021].

Legg, P., Smith, J. and Downing, A. (2019) Visual Analytics For Collaborative Human-machine Confidence in Human-centric Active Learning Tasks. *Human-centric Computing and Information Sciences* [online]. 9 (5) [Accessed 05 September 2020].

Lughofer, E., Smith, J., Tahir, M.A., Caleb-solly, P., Eitzinger, C., Sannen, D. and Nuttin, M. (2009) Human–machine Interaction Issues in Quality Control Based on Online Image Classification. *Ieee Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* [online]. 39 (5), pp. 960-971. [Accessed 22 December 2020].

Maskeliunas, R., Damasevicius, R., Martisius, I. and Vasiljevas, M. (2016) Consumer-grade Eeg Devices: Are They Usable For Control Tasks?. *PeerJ* [online]. 4, p. 1746. [Accessed 02 March 2021].

Mohamed, Z., El Halaby, M., Said, T., Shawky, D. and Badawi, A. (2018) Characterizing Focused Attention and Working Memory Using Eeg. *Sensors* [online]. 18 (11), p. 3743. [Accessed 02 March 2021].

NeuroSky (2018) *MindWave Mobile 2 Available Now! Improved Comfort.* Available from: http://neurosky.com/2018/06/mindwave-mobile-2-available-now-improved-comfort/ [Accessed 06 March 2021].

Roderer, T. and Roebers, C.M. (2010) Explicit and Implicit Confidence Judgments and Developmental Differences in Metamemory: An Eye-tracking Approach. *Metacognition and Learning* [online]. 5 (3), pp. 229-250. [Accessed 23 December 2020].

Smith, J., Legg, P., Matovic, M. and Kinsey, K. (2018) Predicting User's Confidence During Visual Decision Making. *Acm Transactions on Interactive Intelligent Systems* [online]. 8 (2) [Accessed 05 September 2020].

Sommerville, I. (2016) *Software Engineering* [online]. Edition 10. Place of publication: Pearson Education. [Accessed 18 April 2020].

Steichen, B., Conati, C. and Carenini, G. (2014) Inferring Visualization Task Properties, User Performance, and User Cognitive Abilities From Eye Gaze Data. *Acm Transactions on Interactive Intelligent Systems* [online]. 4 (2), pp. 1-29. [Accessed 23 December 2020].

Tobii (no date) *Product Integration – Tobii Developer Zone.* Available from: https://developer.tobii.com/product-integration/ [Accessed 25 March 2021].

Bibliography

Haapalainen, E., Kim, S., Forlizzi, J.F. and Dey, A.K. (2010) Psycho-physiological Measures For Assessing Cognitive Load. *Proceedings of the 12th Acm International Conference on Ubiquitous Computing* [online]., pp. 301-310. [Accessed 23 December 2020].

Appendix A

These cases were created under the principles provided by Sommerville (2018) and are split into three constituent parts: Unit testing, Component testing, and System testing. There are no component tests within this code.

1 Unit tests:

Test case ID	Description	Steps	Expected results
1.	GazeEvent initialisation	1) Run eye tracker 2) Get IL::InteractorId from interactor 3) Get IL::Timestamp from interactor 4) Get bool hasFocus from interactor 5) Create GazeEvent using values 6) Print id, time and gazeGained attributes from GazeEvent object by access	id, time and gazeGained is printed to console
2.	GazeEvent copy	1) Follow initialise from Test Case 1 2) Create new GazeEvent 3) Assign original value into new GazeEvent object 4) Print new object's id, time and gazeGained attributes by access	id, time and gazeGained is printed to console
3.	GazeEvent to_string	1) Create GazeEvent object as defined in Test Case 1 2) Using std::cout, print the object's content	id, time and gazeGained is printed to console

4.	Print GazeEvent list	1) Create multiple GazeEvents using initialisation listed in Test Case 1 2) Use DumpListData to print list of events	id, time and gazeGained of all events are printed to console
5.	Fixation initialisation	1) Create two gazeEvents using initialisation from Test Case 1 2) Get duration using obj2.time – obj1.time 3) Set var interFixDur to float between 0 and 5 4) Set horzPos and vertPos using GetCoordsFromId(obj1.id) 5) Set AOI to the obj1.id 6) Set AOIname to "Stim" + str(AOI) 7) Initialise Fixation using AOIname, obj1.time, duration, obj2.time, interFixDur, horzPos, vertPos and AOI 8) Print values set using step 7 using direct access	Fixation's AOIname, startTime, duration, stopTime, interFixDur, horzPos, vertPos and AOI print to console
6.	Fixation copy	1) Create fixation object as defined by Test Case 5. 2) Create a new Fixation object. 3) Assign original fixation object to the new object. 4) Print values set during step 1 using direct access	Fixation's AOIname, startTime, duration, stopTime, interFixDur, horzPos, vertPos and AOI print to console

7.	Inline BoolToString true check	1) Create bool var = true 2) Call BoolToString(var) with std::cout call	true printed to console
8.	Inline BoolToString false check	1) Create bool var = false 2) Call BoolToString(var) with std::cout call	false printed to console
9.	Calculate the centre of an interactor object	1) Create an array of rectangle interactor with shape cols x rows using the window width and heigh to find the boxWidth and boxHeight as shown in the code provided within main.cpp 2) Set std::pair<float, float> var using GetCoordsFromId(0, cols, rows, boxWidth, boxHeight) 3) Print var.fist and var.second	var.first and var.second are printed to console. var.first == boxWidth / 2 and var.second == boxHeight / 2
10.	Check file exists inline with true	1) Create file called “filename” 2) Set bool var using call to fileExists(filename) 3) Print var	true printed to console
11.	Check file exists with false	1) Set bool var using call to fileExists(filename) 2) Print var	false printed to console

Table 1 - Table of unit tests

2 System tests:

Test Case ID	Description	Steps	Expected results
1.	EyeTrackerCode runs successfully	1) Set size of screen and number of columns and rows on screen, compile code 2) Run main.exe 3) Create file called “start.txt” next to main.exe 4) Look around screen 5) Destroy “start.txt”	Check Student_data.csv file has been created with data in GazemapGen folder
2.	ActiVAte labelling images runs successfully	1) Run activate_app.py 2) Open the webpage printed to console 3) Press “Get Samples” button 4) Label images in correct areas according to how confident you feel 5) Press “Shutdown server” button	Check “confidences.txt” was created and contains details with sync up to the images labelled in ActiVAte.
3.	GazemapGen test run using spoof data	1) Run ../testing/GazeGenSpoof.py 2) Check StudentData.csv and ../Activate/confidences.txt files exist 3) Run main.py	Check test and training folders contain gazemaps

4.	ConfidenceCNN test run using gazemaps produced in test case 3	1) Follow test case 3 to produce gazemaps. 2) Check there are conf and non_conf folders which contain data within ../GazemapGen/test/ and ../GazemapGen/training/ 3) Run CNN.py	Confidence over iterations graph is shown on screen.

Table 2 - Table of system tests