Facial Fatigue recognition (real-time and static images)

GitHub: https://github.com/irfanessa2/CSI-Dissertation

23/24, BSc (Hons) Computer Science Project

Irfan Essa

Supervised by

Enrico Grisan

London South Bank University

Department of Computer Science, Engineering

Declaration

“This dissertation is my own original work and has not been submitted elsewhere in fulfilment of the requirements of this or any other award. Any passages taken from my own previous work or other people's work have been quoted and acknowledged by clear referencing to author, source and page(s). Any non-original illustrations are also referenced. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree as a whole.”

Signature: ______________

Abstract

Facial recognition technology offers a promising tool for identifying signs of fatigue, such as increased blink rate, yawn frequency, and other micro-expressions. This dissertation describes a facial fatigue detection system which has wide-ranging applications from transport drivers, machine operators, aircraft pilots, etc.

The aim of this project is to develop two distinct machine-learning software tools designed for identifying signs of fatigue through facial analysis.

The first tool, which is real-time, focuses on the changes in blink and yawn rates to detect early signs of fatigue using two webcams. This software identifies signs of early fatigue, by detecting an increase in blink and yawn dates which are signs of tiredness. The second tool classifies static images into three discrete categories/levels of alertness. This classifier was trained on a dataset of 1,200 images.

For the first tool (real-time), the software successfully locates the subject's eyes and mouth. It also tracks eye and mouth movement in real-time while the subject's face moves in all three axes. i.e. backwards/forward, left/right, up/down. This software was able to successfully track eye and mouth movement (hence blink and yawn rates) under variations in brightness, background movement, significant head movement and with subjects wearing (or not wearing) spectacles in a real-time driving environment.

For the second tool, which uses static images, a validation accuracy of up to 75% was achieved. This was increased to 85% (0.85) by incorporating metrics which combined precision and recall. This is significantly higher than the random classification of 33.3% which would be expected with three classes.

Overall, this project was successful in that it achieved significantly higher true positive rates for static images as compared to both random classification and currently available commercial software. It was also successful for real-time video in that it was able to measure blinks and yawn rates in controlled and real driving environments, with movement in the background, significant movement in the subject's head and also with the subject's wearing spectacles.

Acknowledgements

I would like to express my gratitude to my supervisor Enrico Grisan, for his guidance and recommendations throughout the course of this paper. His deep understanding of the subject matter and insightful feedback was instrumental in guiding the direction and substance of this work.

List Of Figures

List Of Tables

Glossary & Abbreviations

- BPM – Blinks per minute
- YPM - yawns per minute
- CNN- Convolutional neural network
- HC - Haar Cascades
- CPU – Central Processing Unit
- GPU – Graphical Processing Unit
- AI – Artificial Intelligence
- YOLO – You Only Look Once
- CUDA – Computer Unified Device Architecture
- FPS – Frames Per Second
- ML – Machine Learning
- WSL – Windows Subsystem for Linux
- VM – Virtual Machine
- MTCNN - Multi-Task Cascaded Convolutional Neural Network
- AR - Aspect Ratio
- OOP - Object Oriented Programming
- RAM – Random Access Memory
- EAR – Eye Aspect Ratio
- MAR – Mouth Aspect Ratio
- AR – Aspect Ratio

Table of Contents

Chapter 1. Introduction

This chapter introduces the critical issue of fatigue in various high-risk environments where sustained attention and alertness are essential. It discusses the conventional methods of detecting fatigue and their limitations, highlighting the need for innovative solutions. The chapter then outlines the potential of using advanced computer vision and machine learning techniques to detect signs of fatigue through facial analysis. Each section is designed to provide foundational information, propose a new solution, and discuss its broader implications such as ethical and legal complications.

1.1. Project Background & Overview

Fatigue, especially in contexts requiring sustained attention and alertness, poses a significant risk to both personal and public safety. The consequences of fatigue are particularly acute in professions such as driving, piloting, and operating heavy machinery etc where lapses in concentration can lead to catastrophic outcomes. Traditional methods of fatigue detection have relied heavily on subjective self-assessment or physiological measurements, which can be intrusive or impractical in many working environments.

Recent advancements in computer vision and machine learning modelling have opened new avenues for non-invasive, real-time monitoring of fatigue indicators. Among these, facial analysis technology offers a promising tool for identifying signs of fatigue, such as increased blink rate, yawn frequency, and other micro-expressions indicative of tiredness. This technology has the potential to provide real-time, objective assessments of fatigue and a significant opportunity to enhance safety protocols across a range of industries.

By accurately identifying signs of fatigue, the system outlined here allows for timely interventions, reducing the risk of accidents, enhancing overall workplace safety and the costs associated with such.

1.2. Project Purpose and Rationale

The use of ECGs and similar physiological monitoring devices for the use of fatigue detection is often hindered by the requirement for physical attachments (i.e. wires and sensors), which can restrict user movement and introduce safety hazards in various operational contexts. On the other hand, the reliability of subjective questionnaires is compromised by their vulnerability to manipulation, either intentional or accidental, by respondents.

(NHTSA, 2021) states that drowsiness is a major factor in approximately 2.4% to 20% of all road accidents, though these figures can vary due to differences in analysis. The U.S. National Highway Traffic Safety Administration (NHTSA, 2021) also estimates that drowsy driving is responsible for more than 100,000 crashes annually, leading to 1,550 deaths, 71,000 injuries, and $12.5 billion in monetary losses.

In industries that involve heavy machinery, fatigue can be a critical safety issue, contributing to approximately 15% to 30 of accidents % (OSHA,2022).

A study by the Federal Aviation Administration (FAA, 2020) indicates that fatigue is a contributing factor in 4% to 7% of aviation incidents.

Furthermore, The UK Department of Transport quotes that 20% of all accidents are caused by fatigue (Jackson. P et al., 2011). Blinking and yawning rates are considered early symptoms of fatigue (Tipprasert. W et al., 2019).

The rationale is that by accurately and non-invasively identifying early signs of fatigue, this system could reduce the risk of accidents and increase safety in the workplace. Thus, increasing productivity and reducing costs.

1.2.1 Problem with existing systems

The problem with existing implemented systems is threefold.

1. The majority of current systems, for example, one identified by (Yang et al., 2017) and discussed in section 2.1.4 are aimed at identifying a range of facial expressions e.g. smiling, sad etc. They are not targeted primarily at detecting signs of fatigue.

2. Secondly, most systems, which are currently based on machine learning, are limited to a controlled environment. The lighting conditions are controlled within a narrow range, the background movement is minimal and the images are often accurately cropped.

3. Third, because most current systems are demonstrated (and trained) in a controlled environment, the orientation of the face (with respect to the camera) is within very strict limits and not 'negotiable'. This restriction would be intolerable in a 'real-time' work or study environment.

1.2.2 Proposed solution

This project is motivated by the need to overcome the above (see sec 1.2.1) limitations of traditional/existing facial expression detection systems.

This project proposes the development of a system dedicated to the early and accurate detection of facial fatigue. The system proposed here is a twofold solution. Firstly, a real-time setup that utilises a camera-based facial analysis system. Secondly, to analyse static images, again using machine learning algorithms. Both approaches are non-invasive, cost-effective, and universally applicable solutions to fatigue detection in the real world. Both systems, proposed here, are to be tested (and trained) in a real work/driving environment with varying lighting conditions, significant background and head movement and subjects wearing spectacles. Furthermore, it is also proposed to include calibration (of the subject's face) in the real-time setup to further enhance accuracy.

1.3. Aims and Objectives

The **aim** of this project is to develop and evaluate a two-tier (Real-time video and static images) facial fatigue detection system that can accurately identify early signs of fatigue.

Objectives

Real-Time Video:

A) Allow two webcams (inexpensive) and video streams to be captured

B) Allow first-time users to calibrate their own eye and mouth (yawn) movement characteristics (ranges)

C) Allow the user to save their calibration data for future use

D) Allow the user to load their profiles containing their calibration information/ranges

E) Detect facial landmarks on the face, i.e. eyes, mouth, etc.

F) Process real-time video to measure blink and yawn frequency

G) Flag the user when a fatigue threshold has been reached/triggered based on F (see above)

H) Give option to output data (.csv) file containing blink and yawn rate

Static-Image classification

A) Create a well-balanced and data-rich training set with appropriate labels classifying three levels of fatigue.

B) Automate face cropping using MTCNN (Multi-Task Cascaded Convolutional Neural Network)

C) Standardise the training set for the same image size, aspect ratio, contrast etc, and implement data augmentation to add variety to training set

D) Develop an ML model to determine three fatigue levels, and fine-tune to prevent overfitting through using techniques (see above) and others such as early stopping.

E) Provide/output model performance metrics (F1 score, loss, accuracy, precision, recall, confusion matrix, etc.).

F) Allow the user to input a cropped image of a face and give an output of one of three fatigue levels with a confidence score.

General

A) Keep the hardware costs (of real-time system) as low as possible, so it may be inexpensively incorporated into as many applications as possible.

B) Keep the graphical user interface (real-time system) simple to use with minimal messaging and inputs. Therefore, allowing this system to be used safely without prior training.

1.4. Scope

1.4.1 Project milestones

- Identify currently available solutions.
- Highlight problems (of current available solutions) for facial fatigue recognition
- Propose solutions to problems outlined above

Real-Time

- Generate code for blink and yawn analysis aided through existing libraries.
- Develop a user-friendly GUI with calibration for real-time. Include employee employer login and retrieve profile containing pre-saved calibration data.
- Output results on GUI, and also generate tabulated results.

Static Images

- Gather the data training set (static images) and crop and classify the images as required.
- Generate code to create a classifier model to distinguish between three levels of fatigue.
- Generate performance metrics.

1.4.2 Intended Environment for Real-time System

For video, this system is intended to be used to monitor only one face. For optimal results, the camera position should remain constant. This system should only be used where there are not excessive head movements.

1.4.3 Machine learning (static images)

The accuracy of the system will be proportional to the training set size. This can be increased with time. The quality of the training set is also a consideration. The classification, how the images are cropped, lighting conditions and the general head position will also all influence the results.

1.4.4 Factors Affecting Accuracy

Both systems are intended to be used for analysis of a single face. Therefore, in some real-time environments, e.g. Where other people are passing in the background, the accuracy will be reduced. Other local environmental changes will also affect/reduce accuracy E.g. When a bus driver on a cloudy day sees the sun, he may blink. The background light conditions will also affect the accuracy.

1.4.5 Hardware

The camera position should remain constant relative to the user for optimal results

1.4.6 Software

For static images: The classifier (machine learning) requires a photo of a single face correctly cropped. The quality of images will also have a significant impact on the performance of the model (i.e. motion blur, light reflections, fisheye lens effect, etc).

For real-time system: The variations in brightness, and background movement will degrade accuracy.

1.5. Commercial and economic benefits

Pilots

Fatigue detection systems in aviation can drastically reduce the risk of accidents attributed to pilot fatigue, which, according to the Federal Aviation Administration (FAA), plays a role in 6-13% of aviation accidents (Caldwell et al., 2009). By ensuring pilots are sufficiently rested, airlines can avoid the potential costs associated with accidents, which, according to the National Transportation Safety Board (NTSB), average $242 million for a commercial aviation accident. Furthermore, improving pilot alertness can enhance overall flight efficiency and reduce operational delays, saving airlines an estimated $150,000 per avoided delay (National Aeronautics and Space Administration, 2014).

Drivers

In the trucking and transportation sector, fatigue detection systems can lead to substantial reductions in road accidents. The American Trucking Associations (ATA) report that fatigue-related accidents cost the industry approximately $9 billion annually (American Trucking Associations, 2012). Implementing fatigue detection technologies, such as in-cab monitoring systems, has been shown to reduce long-haul truck accidents by up to 20%, leading to significant savings on insurance premiums and accident-related costs (FMCSA, 2014). This will also be true with millions of other drivers on busy roads.

Heavy Machinery Operators

For operators of heavy machinery, fatigue detection systems can prevent costly accidents and equipment damage. The Occupational Safety and Health Administration (OSHA) estimates that fatigue-related accidents in the construction and manufacturing sectors result in annual costs exceeding $1 billion, including lost productivity, medical expenses, and equipment replacement costs (OSHA, 2015). Implementing fatigue monitoring systems can reduce these incidents by up to 30%, representing substantial financial savings for companies.

Desk Employees

In office environments, fatigue detection systems that monitor computer usage patterns and physiological signs can enhance employee productivity and health. The Centres for Disease Control and Prevention (CDC) suggest that fatigue results in a 65% reduction in cognitive performance among office workers, leading to an estimated loss of $2,000 per employee annually due to decreased productivity and increased sick leave (Centres for Disease Control and Prevention, 2016). Implementing fatigue management programs can improve productivity by 10-20%, translating into significant cost savings for employers.

The implementation of fatigue detection systems across various sectors not only enhances safety and well-being, but also presents significant economic benefits by reducing costs associated with accidents, improving operational efficiency, and increasing productivity.

1.6. Legal, social, ethical, and professional concerns

The (Data Protection Act 2018) states that it is legal to monitor employees (including CCTV) if monitoring is lawful, fair and transparent. It is worth noting that data collected should be specific, and explicit and not be kept for longer than necessary. It is vital to note that employees should be made aware of any recording and that the employer is required to gain consent from any staff. Employees also have the right to access personal data, have it erased and stop or restrict their data from being processed. Since the data stored for the proposed system is temporary (which is wiped when the software is closed), it complies with the criteria. The data collected on the user's facial information will also be limited to specific attributes such as eye, mouth, and nose movement (to detect if looking up or down). Additional irrelevant data such as hair colour will not be stored, therefore ensuring that it complies with the requirement that data collected must be specific.

Though the project is ethical as parties are being made aware of monitoring and the data being collected by the software, there is a risk that the high use of monitoring could change social norms where monitoring is considered normal

As per professional issues, the software being used by driver employees would result in a decrease in accidents and therefore an improvement in the company's image. However, it is important to note that its implementation of a facial monitoring system could negatively impact the company's image as it may raise concerns about intrusiveness.

Chapter 2. Literature and Technical Review

The following sections outline a literature review (section 2.1) into the background of effective study methods, existing solutions, and an insight into machine learning neural networks. This is then followed by a technical review (section 2.2) which explores various libraries, frameworks and models used to achieve the real-time and still image facial fatigue detection software. It discusses their advantages/disadvantages and the relevance to this project. The section is then concluded by choosing particular methods, frameworks, models and libraries which are used for the project.

2.1 Literature Review

The method of research used to evaluate the needs, impacts and value of this project involved the exploration of various peer-reviewed research papers and articles produced by established institutions (and provided by LSBU library and Google Scholar) that utilised a diverse range of research techniques to provide reliable results.

2.1.1 Current existing solutions

2.1.1.1. Yawn Based Driver Fatigue Level Prediction,

(Haider A. K, 2020) Describes a driver fatigue system which classifies drivers into one of three fatigue levels. i.e. Alert, early fatigue, and fatigue. This is based purely on the number of yawns detected. ‘Alert’ implies no yawning. ‘Fatigue’ is triggered with frequent yawning (more than once a minute) and ‘early fatigue” is flagged between these two states. The system uses the Yawdd dataset provided by (Abtahi et al., 2020) which has a large set of videos depicting both male and female subjects yawning, while talking, in a calm environment.

A deep CNN model is used as the classifier. The model is first trained using a training set and then tested using randomly chosen cropped video from the same dataset.

<u>Disadvantage</u>

Although this system boast high accuracy of yawn detection, by using only yawn detection to identify fatigue, the system can be easily fooled for example by the subject simply opening their mouth for a long period of time. The other drawback here is that real-time video capture is not demonstrated.

2.1.1.2. Driver Drowsiness Detection Based on Face Feature and PERCLOS

A paper by (Suhandi. J and Habibullah. A 2018) proposes a system using video capture (not real-time). The system initially detects both eyes, and then calculates PERCLOS. Perclos is a ratio metric measurement comparing the closed eye time to the time the eye is open.

The system is evaluated using the YawDD video dataset and not real-time video capture. The system does not incorporate a method of calibration and therefore the results are presented using 3 fixed threshold values (P60, P70 & P80) for both eyes. A threshold value of P60 implies that the eye is closed 60% during a fixed measurement time. The system found that the PERCLOS value is lower when the driver is drowsy. The paper highlights ‘head movement’ as a serious limitation when using this system.

Advantage: The system is simple

Disadvantage: No real-time video, no method of calibration (for different subjects) and a measurement that concentrates on only one feature the eye)

2.1.1.3. Driver's Fatigue Detection Based on Yawning Extraction

(Nawa. A, 2014) describes a system that detects/identifies yawning by measuring physical changes occurring in the driver's mouth using circular Hough transform (CHT). A 'normal' (non-specialized) video camera is used to capture the image. The system is based on based on support vector machine (SVM) technique and executes various mathematical functions to achieve the final result. The paper boasts accuracy rates in excess of 90% for yawn detection. It is also worth noting that the paper does not highlight any limitations for using ONLY yawn rate to classify fatigue.

Advantage: The system is simple and uses real-time video from an inexpensive camera.

Disadvantage: Only Yawn analysis, no method to calibrate for different subjects. Also, the system can be easily fooled by the subject just opening their mouth for an extended period.

2.1.1.4 Emotion recognition as a means to detect fatigue

A university paper written by (Yang et al., 2017) at the University of Sydney outlines the development of an emotion recognition software which is used in a virtual learning environment. Its goal is to provide teachers with real-time feedback on their student's emotions allowing them to adapt their teaching methods in real-time. (Yang et al., 2017) divides emotions into six categories; sadness, happiness, surprise, fear, anger and disgust.

D. Yang et al. / Procedia Computer Science 125 (2018) 2–10

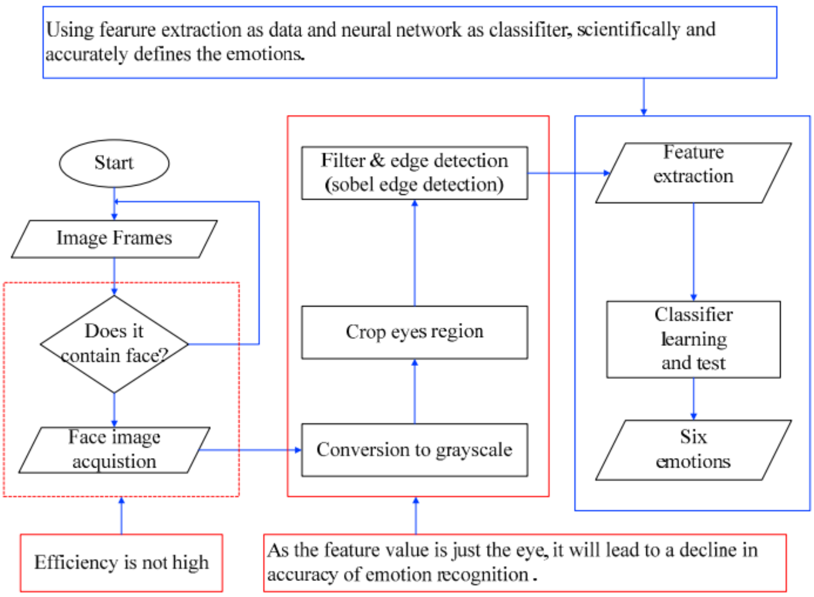


Figure 1: Flow chart solution proposed by (Yang et al., 2017)

The proposed software solution (see Figure 1) first checks whether a face can be identified, if it can then it saves the image (this process is inefficient). If a face is not identified, the software loops until one has been found. It then converts the relevant image to black and white. The person's eyes are then cropped out of the image, this is done to improve accuracy. The software then detected edges utilising the Sobel edge detection method. Facial features are then extracted and fed into the classifier for testing and recognition of the six emotions.

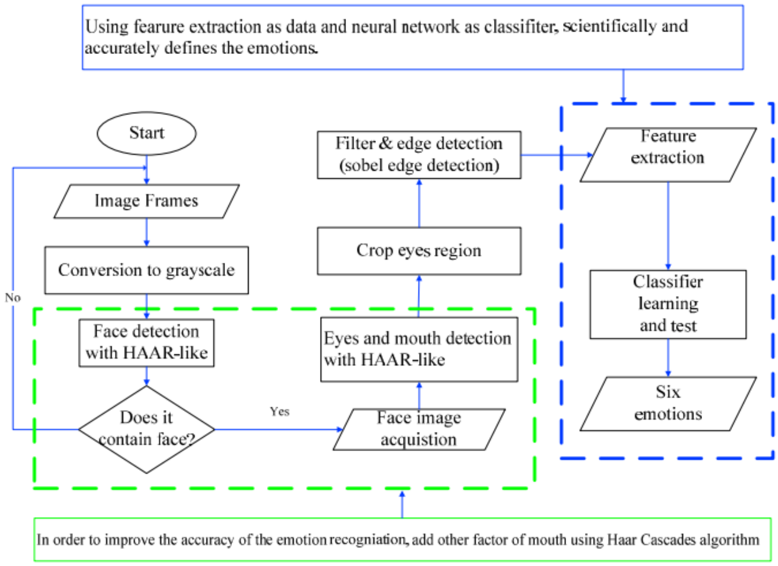


Figure 2: Improved flow chart solution proposed by (Yang et al., 2017)

To improve accuracy, the new model (see Figure 2) produced includes the implementation of Haar Cascades. Furthermore, to improve the performance issues outlined in Figure 2, the images are converted to greyscale prior to facial detection.

Advantage: Simple as it uses emotions (i.e. happy and sad) to gauge fatigue.

Disadvantage: Not really suited for facial fatigue recognition. No calibration is available for real-time video.

2.1.1.5 Facial Expression Recognition in Multiple Smiles

Another more recent paper cited by (Zhijia Jin, 2023) proposes and implements a system to classify a "smile" into 6 distinct classes. The system utilises a training set of 1,100 images. Some images were cropped from online photos and the majority were individually photographed in a controlled environment often using a verbal phrase or sentence to invoke a particular smile. Because the majority of the images were photographed in a controlled environment, the light conditions were controlled. Also, the images were "Full facial" i.e. the face was directly looking into the camera when photographed. The system requires greater processing as it utilises a convolution algorithm to focus on particular (pre-determined) important areas/features of the face and also "suppress" background noise. With 6 classes and their controlled photographic environment, they achieved an accuracy rate exceeding 80%, which is impressive.

Advantage: Good accuracy

Disadvantage: Only demonstrated in a "controlled" environment. This is not applicable in everyday practical environments e.g. driving, work environment, hazardous areas etc.

2.1.1.6 Viola–Jones object detection framework

A much earlier paper written by [viola jones, 2001] uses a machine learning algorithm to classify facial expressions into 2 distinct classes using facial landmark features. The system only copes with 2 fps. Other constraints include (but not limited to) images of fixed resolution, a narrow range of image brightness, fixed face orientation (looking directly forward), and only two classes. The system does use a cascade feature which allows the two classes to be further subdivided. Although the machine learning algorithm is advanced for its time, it cannot be compared to the software technology available today. Furthermore, at the time the paper was published, the system was run on a standard 700mHz Pentium processor.

Advantage: This system was "ahead" of its time.

Disadvantage: Few features can be applicable to the system proposed (and implemented) in this report.

2.1.1.7 Steering pattern monitoring

Steering pattern monitoring is a method used in automotive safety to detect driver fatigue by analysing steering behaviour as an indicator of alertness levels. This approach is based on the premise that changes in driving behaviour, particularly steering actions, can signal the onset of fatigue (Salvucci, 2012). The technique involves collecting data from the vehicle's steering system through sensors that track steering wheel angle, speed, and the frequency of steering adjustments. This data is then processed by algorithms to identify patterns that indicate fatigue. For example, a fatigued driver may show increased steering wheel corrections due to a diminished capacity to maintain a steady vehicle trajectory, leading to greater variability in steering inputs (Dong, 2017). These systems aim to mitigate the risk of accidents by suggesting breaks or other interventions (Li, 2015).

Advantage: Simple and precise measurement

Disadvantage: Doesn’t take subject and person into account

2.1.1.8 Real-time driver fatigue detection system with deep learning on a low-cost embedded system

A driver fatigue system proposed by (Esra. C, 2023) classifies drivers in to one of four fatigue levels. The system utilizes two CNN models to detect driver fatigue. Using two CNN models to monitor the eyes and mouth greatly improves accuracy. The images are resized 150 x150 prior to training/classification. The system is tested using the Yawdd dataset provided by (Abtahi et al., 2020) which has a large set of videos and operates at 6 fps. The authors boast a accuracy in excess of 90% while keeping costs down. Head tilt, background movement, lighting conditions and real time environments are not discussed in any detail.

Advantage: Relatively low cost – Although the actual cost is not given/discussed, good accuracy

Disadvantage: Only operates at 6 fps, not demonstrated in a real environment with background movement, varying lighting conditions or with subject wearing spectacles.

2.1.1.9 Drowsiness Detection Based on Yawning Using Modified Pre-trained Model MobileNetV2 and ResNet50

The system proposed by (Hepatika . L , 2023) combines the Haar cascade classifier with MobileNetV2 and ResNet50 models. The images, taken from a camera, are classified as either yawning or non-yawning. They found that results based around the ResNet50 model yielded more accurate results. A total of approx 5,000 images (224 x 224 format) were used. Split into 60% for training and 20% each for validation and testing. Accuracies in excess of 95% are quoted for both models.

Advantage: Low cost, very good accuracy with both models. Simple to implement

Disadvantage: Only demonstrated with static images. Their findings also quote that the training set was “finely tuned”. No video (real time or otherwise)

2.1.1.10 Real-time-Driver-Drowsiness-Detection-System-Using-Deep-Learning

A paper produced by (Pratham . M, 2023) uses two data sets. One for eyes, called “closed eyes in the wild”, (open or closed) and the second called YawDD for yawning (yawning or not yawning). Images are cropped 145 x 145. Uses a Haar cascade classifier and a convolution neural network built using Keras sequential model. Accuracies of up to 97% are quoted.

Advantage: Good accuracy although the term “up to” is used.

Disadvantage: No real time video. Not demonstrated (no results) with head tilt

	Title	Author	Year
1	**Yawn Based Driver Fatigue Level Prediction**	Haider A. Kassem1	2020
	Approach: Deep CNN model classifier		
	Advantage: Good accuracy		
	Disadvantage: Only uses yawn. No real-time video		
2	**Driver Drowsiness Detection Based on Face Feature and PERCLOS**	Suhandi Junaed	2018
	Approach: Perclos. Measures "eye closed" percentage time		
	Advantage: Simple system. Good accuracy		
	Disadvantage: Only eye. No real-time video. No calibration		
3	**Driver's Fatigue Detection Based on Yawning Extraction**	Nawal Alioua,	2014
	Approach: SVM		
	Advantage: Uses camera. Good accuracy		
	Disadvantage: Only Yawn analysis		
4	**Emotion recognition as a means to detect fatigue**	Yang et al.	2017
	Approach: Virtual learning (classifier)		
	Advantage: 6 different emotions		
	Disadvantage: Not suited to fatigue recognition. No calibration		
5	**Facial Expression Recognition in Multiple Smiles**	Zhijia Jin	2023
	Approach: ML Classifier		
	Advantage: Good accuracy		
	Disadvantage: Controlled environment. No background movement		
6	**Viola–Jones object detection framework**	viola jones	2001
	Approach: ML to classify facial expression		
	Advantage: Good when published. Now mostly out of date		
	Disadvantage: Not really relevant here. E.g. only copes with 2 FPS		
7	**Steering pattern monitoring**	Salvucci	2012
	Approach: Collect steering wheel data with ML		
	Advantage: Simple, precise		

	Disadvantage: No really relevant here. Doesn't take the subject/person into account		
8	**Real-time driver fatigue detection system with deep learning on a low-cost embedded system**	Esra Civik	2023
	Approach:2 CNN models to monitor eyes/mouth		
	Advantage: Low-cost		
	Disadvantage: Operates at 6 FPS		
9	**Drowsiness Detection Based on Yawning Using Modified Pre-trained Model MobileNetV2 and ResNet50**	Hepatika Zidny Ilmadina	2023
	Approach: Classifies yawning and not yawning		
	Advantage: Low cost, simple to implement		
	Disadvantage: Only works for static images		
10	**Real-time-Driver-Drowsiness-Detection-System-Using-Deep-Learning**	Pratham Metha	2023
	Approach: Eyes/Mouth opening and closing using Haars cascade		
	Advantage: Good accuracy, but says “up to” 97%		
	Disadvantage: No real-time video. No test with head tilt		

Table 1: Table of papers reviewed

2.1.2. The requirements (and benefits) for this project

The requirement for this facial fatigue recognition system is clear. detecting early signs of fatigue can significantly reduce the risk of accidents in numerous environments. Some environments include but are not limited to, Driving/piloting, use of machinery, using hazardous chemicals/tools, hazardous environments (e.g. cooking) etc.

By using this system for early fatigue recognition, everyday environments (work and otherwise) can be made significantly safer, and the costs associated with such accidents can be reduced. In a business environment, this would result in increased profits. In a non-business environment, the organisation e.g. government, council, or education department would benefit.

2.2 Technical Review

This technical review will detail the hardware and software requirements for two key components of the system: a real-time facial fatigue detection software, and a still image classifier. For each component, we will outline essential specifications for performance and discuss the necessary software frameworks and tools. This document aims to provide developers with a clear understanding of the technical needs to efficiently develop and deploy both systems.

2.2.1 Real-Time Facial fatigue detection

This system should be highly responsive and capable of handling live video feeds effectively to detect signs of fatigue in a user's facial eye and mouth movements.

2.2.1.1. Hardware Requirements (Real Time)

- Two USB 420p+ cameras.
    - Two webcams with a resolution of 420p or higher are required to capture clear facial video from two angles (when looking up and when looking down). The two cameras are spaced vertically in order to compensate for pitch (head tilt up and down). Only one of the two cameras is selected at any one time, depending on the pitch of the head.
- 2.3 GHz CPU or higher
    - A CPU with a minimum speed of 2.3 GHz is recommended to handle the real-time processing and analysis of video data without significant latency.
- 4GB RAM
    - 4GB of RAM is recommended to efficiently manage the concurrent operations of video capture, processing, analysis and other GUI features. This ensures that the system can handle multiple tasks and data streams without performance degradation. Again, the amount of memory is minimal to keep costs down.
- **CPU supporting Hyper-V** (Hypervisor) or base Debian distribution i.e. Ubuntu
    - The software utilises environment variables (for video inputs) and Udev rules to keep camera inputs consistent regardless of the order of being

plugged in, regardless of if plugged in first or last. This requires the use of Linux where udev rules are configured in /etc/udev/rules.d/.

Reasoning: The main reason for the above hardware choice is to keep the system cost to an absolute minimum without sacrificing significant results. By keeping the hardware costs low, the system usage can be maximised in as many applications and environments as possible. The secondary reason is the availability of the above hardware components. Easily available hardware components are used to ensure maximum system usage/coverage.

2.2.1.2 Software Requirements (Real Time)

USE	Chosen	Alternative Considered
Facial Landmarks	MediaPipe	Dlib
GUI	PyQt5	Tkinter
Vision Tool	CV2	Image Tool Kit (IKT)
Language and Threading	Python	Java

Table 2: Real-time software requirements overview

i) Facial Landmarks (MediaPipe vs Dlib)

MediaPipe is an open-source framework developed by Google, designed to facilitate the development of machine learning pipelines for media processing. It supports applications relevant to this project such as facial features tracking (i.e. recognising eyelids, lips, forehead, etc.). This is very useful for fatigue detection as it allows analysis of changes in blink rate, gaze detection and yawning which are indicators of fatigue and drowsiness. It also provides hand detection.

It works by using a graph-based structure where each graph represents a media processing pipeline (MediaPipe Developers, 2021), guiding data through various processing nodes. Also, these nodes handle tasks like machine learning inference and image processing and enable efficient real-time data processing essential for interactive applications.

Figure 3: MediaPipe example

It works by assigning each facial point (landmark) a value, i.e. nose tip = 1. This will allow the tracking of facial points, for example eye lids, and measure their movements using pixels as reference points therefore allowing detection of blinks (an increase in frequency being an indication of fatigue). The same is also true for the number of yawns.

Pros	**Cons**
Efficiency: Optimised for real-time performance, MediaPipe processes video streams quickly and efficiently, essential for real-time applications like fatigue detection.	**Limited Customisation**: MediaPipe's pre-built solutions offer limited customisation, posing challenges for developers needing specific adjustments beyond standard configurations.
Robust Tracking Capabilities: Excels in accurately tracking multiple facial features with high accuracy and low latency, crucial for effective fatigue detection.	**Steep Learning Curve:** Its graph-based processing architecture can be complex for newcomers, potentially requiring a significant time investment to master.
Integrated Visualisation Tools: Features built-in visualisation tools that aid in debugging and refining models by providing real-time visual feedback.	**Dependency on External Libraries:** Heavily relies on external libraries like TensorFlow and OpenCV, where any limitations or issues in these frameworks could impact MediaPipe's performance.

Table 3: Pros and Cons of MediaPipe for Facial Landmarks

Dlib is a machine learning and computer vision library that features a facial landmark detection model with 68 points, mapping key facial features such as eyes, nose, and mouth. It works by predicting these points on faces, aiding in tasks like recognition and emotion analysis. However, Dlib isn't as effective for real-time applications as MediaPipe, mainly because it lacks the specialised pipeline architecture that efficiently manages continuous data flow and resource utilisation necessary for optimal real-time performance.

Pros	Cons
Detailed Facial Mapping: Dlib provides an accurate mapping of 68 facial points, beneficial for recognition and emotion analysis.	**Less Suitable for Real-Time Applications**: Lacks efficient real-time data processing capabilities due to its traditional architecture.
Versatility: It supports various machine learning and computer vision tasks, offering broad utility beyond facial analysis.	**Higher Resource Utilisation:** Tends to consume more computational resources, which can be a drawback for continuous real-time analysis.
Strong Community and Support: Dlib benefits from a robust user community and comprehensive documentation, facilitating easier implementation and troubleshooting.	**Limited Mobile Support:** While powerful, Dlib offers limited optimisation for mobile environments, affecting its performance and usability on mobile devices.

Table 4: Pros and Cons of Dlib for facial landmarks

Final Decision and reasoning (MediaPipe as facial landmarks)

MediaPipe is ideal for real-time facial fatigue recognition because it accurately tracks facial features (i.e. eyes, mouth, head position, etc) which is essential for identifying signs of drowsiness. It also processes data very efficiently, which results in a responsive real-time software. **This is the main reason it was decided to use MediaPipe in this project.**

ii) GUI (Pyqt5 vs Tkinter)

PyQt5 is a set of Python bindings for the Qt application framework, enabling GUI development through Python. It works by employing an event-driven architecture where user or system actions trigger events. Here, its event-driven architecture ensures that the software can promptly respond to changes in facial expressions detected using camera inputs, updating the GUI in real-time as it processes data. Communication between these elements is facilitated through a signals and slots mechanism which allows for efficient communication between back-end facial recognition processing (MediaPipe) and the front-end user interface (GUI)

Pros	**Cons**
Event-Driven Architecture: Ensures real-time responsiveness in the GUI, essential for applications requiring immediate feedback.	**Steep Learning Curve:** Complex features that can be challenging for new users to master quickly.
Python Integration: Seamlessly combines Python's ease of use with Qt's robust GUI capabilities.	**Resource Intensive:** May consume substantial system resources, affecting performance on lower-end devices.
Efficient Communication: Uses signals and slots for effective backend-to-frontend communication, enhancing application interactivity.	**Limited Mobile Compatibility:** Better suited for desktop environments, with suboptimal performance on mobile platforms.

Table 5: Pros and Cons of PyQt5 as GUI

Tkinter is more popular, however, it was not the best choice for the real-time facial fatigue recognition project due to its limitations (see below):

a**) Single-threaded Model:** Tkinter operates on a single-threaded model, which can lead to performance bottlenecks in applications requiring concurrent operations like video processing and GUI updates.

b**) Limited Widgets:** Tkinter offers a basic set of widgets, which may not suffice for building sophisticated, user-friendly interfaces needed in dynamic real-time applications.

c) **Performance Issues:** Compared to PyQt5, Tkinter's performance and rendering capabilities are less robust, affecting the responsiveness and efficiency of the application.

Pros	**Cons**
Ease of Use: Tkinter is known for its simplicity, making it accessible for beginners to quickly start developing GUI applications.	**Single-threaded Model:** Tkinter's single-threaded nature can cause performance bottlenecks in applications that require handling multiple operations simultaneously.
Standard Library Integration: As part of Python's standard library, Tkinter doesn't require separate installation and works seamlessly within the Python environment.	**Limited Widgets:** The toolkit provides a basic selection of widgets, which might not be adequate for creating more complex or modern-looking GUIs.
Lightweight: Tkinter is a lightweight option that's less demanding on system resources, and suitable for simpler applications.	**Performance Issues:** Generally, Tkinter has inferior performance and rendering capabilities compared to more robust frameworks like PyQt5, impacting the responsiveness of dynamic applications.

Table 6: Pros and Cons of Tkinter for GUI

Final Decision and Reasoning (PyQt5 as GUI):

PyQt5 is crucial for the real-time facial fatigue recognition project because its event-driven architecture ensures immediate responsiveness to facial expression changes detected via webcam. Additionally, PyQt5's signals and slots mechanism enables efficient communication between the back-end facial recognition processing (handled by MediaPipe) and the front-end GUI. This integration allows for dynamic GUI updates that can promptly alert users about detected signs of fatigue, enhancing both safety and system reactivity. **The reasons outlined above are the main reasons why PyQt5 was chosen as the main GUI framework.**

iii) Vision Tool (OpenCV vs ITK)

OpenCV (Open-Source Computer Vision Library) is a widely utilised library in computer vision and image processing. OpenCV works by representing images as multi-dimensional arrays through the 'Mat' object in C++, encapsulating the pixel data of images where each pixel may encode grayscale or colour intensities across various channels. For example, converting an RGB image to grayscale is compressed into a single function that computes a weighted sum of the colour channels. This is a key example of the library's efficiency (Bradski, G., 2000)

Pros	Cons
High Performance: OpenCV is highly optimised for real-time operations, making it suitable for applications that require immediate processing like facial fatigue detection in drivers or operators.	**Steep Learning Curve:** For beginners, the breadth of functionalities in OpenCV can be overwhelming, making it challenging to start complex projects without prior experience in computer vision.
Robust Algorithm Support: It offers robust algorithms for face detection and tracking, which are essential for detecting signs of fatigue such as eye closure duration and frequency, blink rate, and yawning.	**Limited High-Level Functionality:** While highly effective for low-level vision tasks, OpenCV lacks some high-level capabilities directly out of the box, such as advanced deep learning functionalities which are often needed for precise fatigue detection.
Community and Resources: OpenCV has a vast community and a plethora of tutorials and resources which facilitate development.	**Dependency Management:** Integrating OpenCV with other libraries and managing dependencies can be cumbersome, particularly in environments that require consistent real-time processing capabilities.

Table 7: Pros and Cons of OpenCV as vision tool

Image Toolkit (ITK) is a Python library designed for image manipulation that leverages FFmpeg for video processing tasks. It works by employing techniques like convolution, interpolation, and matrix transformations to apply various image-processing operations. These operations are executed at a low level and directly modify pixel values in memory, ensuring efficient and fast processing of image data. Although it can handle various image and video formats, ITK is not specifically optimised for real-time video processing tasks.

Pros	**Cons**
High Precision Algorithms: ITK is equipped with sophisticated algorithms designed for accuracy and robustness, making it highly effective for critical applications like medical imaging where precision is paramount.	**Complex Implementation:** The complexity of ITK's architecture can pose challenges, making it difficult to integrate and maintain within real-time processing systems where simplicity and speed are crucial.
Extensive Community and Documentation: ITK benefits from a large, active community. This support network provides extensive documentation and user guides, facilitating easier problem-solving and implementation.	**Performance Limitations:** While ITK is powerful for many image processing tasks, its high computational demands can be a drawback in real-time applications where quick processing is required.

Table 8: Pros and Cons for Image Took Kit as vision tool

Final Decision and Reasoning (CV2 as vision):

OpenCV is relevant to real-time facial fatigue recognition software as it efficiently handles image data allows for fast and accurate analysis of facial expressions and changes, which are critical for detecting signs of fatigue. It also allows for easy integration of webcams, allowing real-time software to easily capture video feeds directly from cameras and video/image inputs (via file path). ITK was considered by not used due to its lack of support for real time.

iv) Threading Language (Python vs Java)

The **Python** threading library is a component of the standard library designed to facilitate the concurrent execution of code through threads. A thread represents the smallest sequence of programmed instructions that can be managed independently by a scheduler, this allows for simultaneous execution of code blocks. The creation of a new thread in Python using the threading library typically involves subclassing the Thread class and overriding its run() method. The thread starts its execution when its start() method is called, running in concurrently to the main program (Summers. E, 2015).

The threading library provides several synchronisation mechanisms, including:

- Locks: Allow only one thread to access a segment of code or data at a time.
- Rlocks: Permit a thread to acquire a lock it already holds, useful for recursive functions.
- Semaphores: Enable a fixed number of threads to enter a critical section.
- Conditions: Facilitate the pausing of threads until notified by other threads.
- Events: Signal between threads that certain conditions or events have occurred (Beazley, D., 2010).

Pros	**Cons**
Ease of Use: Python's threading is user-friendly, simplifying the implementation of multi-threading with high-level interfaces.	**Global Interpreter Lock (GIL):** Python's GIL limits performance in multi-threaded CPU-bound applications, as it allows only one thread to execute at a time.
Supportive Libraries: Python offers numerous libraries that aid in threading and concurrency, making it versatile for managing multiple threads.	**Limited CPU Efficiency:** Python is not optimal for CPU-intensive multi-threading due to the GIL, which restricts its ability to fully utilise multi-core processors.

Table 9: Pros and Cons of Python for Threading

JAVA was considered due to its improved multithreading and OOP support. However**,** Python was preferred for its integration with MediaPipe due to its comprehensive libraries like TensorFlow and OpenCV, which facilitate efficient image processing.

Pros	**Cons**
Built-in Threading Support: Java has built-in support for multi-threading with extensive API support, such as java.lang.Thread and concurrent packages, allowing efficient thread management and synchronisation.	**Complexity in Thread Management:** Despite its capabilities, managing Java threads can be complex, requiring careful handling of synchronization and thread safety to avoid issues like deadlocks and race conditions.
Performance: Java threads are managed directly by the JVM, which can efficiently handle many threads, optimising performance across multi-core systems.	**No Native Support for MediaPipe:** Java does not natively support MediaPipe, a popular library for building multimedia processing pipelines, which could be a limitation in projects requiring advanced video and image processing.

Table 10: Pros and Cons of Java for Threading

Final Decision and reasoning (Python Threading)

The Python threading library is crucial for the real-time facial fatigue recognition software as it enables concurrent execution of tasks like image processing and GUI updates. This ensures the system remains responsive and efficient, with mechanisms like Locks and Semaphores to manage resource access and prevent conflicts, essential for maintaining stable operation during real-time data analysis.

2.2.2 Static Image facial fatigue detection

2.2.2.1. Hardware Requirements (Static Image)

- 4GB GPU or higher (Nvidia recommended)
    - A GPU with at least 4GB of memory is recommended to efficiently handle the computational load of images

2.2.2.2. Software Requirements (Static Image)

USE	**Chosen**	**Alternative Considered**
Base Framework	TensorFlow	PyTorch
Visualisation	MatPlotLib, SKlearn	
Parallel Processing	CUDA	OpenCL
Base Model	MobileNet	AlexNet, ResNet50

Table 11: Static-Image Software Requirements Overview

i) Base Framework (TensorFlow vs PyTorch)

TensorFlow is Google's competitor to PyTorch, like PyTorch, it is an open-source framework for AI with a focus on deep learning. TensorFlow works by utilising static computation graphs which gives much more control over optimisation and resource distribution due to its fixed structure.

Pros	**Cons**
Built-in Parallelism: TensorFlow is designed to handle parallel processing natively, effectively managing operations across multiple cores and GPUs for significant performance boosts.	**Steep Learning Curve:** TensorFlow can be complex, especially for beginners, due to its extensive and detailed API.
Scalability: It scales well from single devices to large clusters, making it suitable for both small and large-scale systems.	**Less Pythonic:** Its static graph definition is less intuitive compared to PyTorch's dynamic nature, which might hinder rapid development and experimentation.

Table 12: Pros and Cons of TensorFlow as main framework

PyTorch is an open-source machine learning library developed by Facebook's AI Research lab. Unlike TensorFlow which uses static computation graphs, PyTorch works through a dynamic graphing approach, (known as define-by-run paradigm), where the graph is built on the fly during execution. This feature allows for more natural coding. **However, computational graphs have significant performance overhead** due to the fact that the computational graph needs to be rebuilt at each iteration. Dynamic graphing also makes debugging significantly more difficult.

Pros	**Cons**
Dynamic Computation Graphs: PyTorch uses dynamic computation graphs which are easier to work with for real-time updates and debugging.	**GPU Utilisation:** While PyTorch is efficient, its dynamic nature can sometimes lead to less efficient GPU utilisation compared to TensorFlow's static graphs.
User-Friendly: It offers a more intuitive, Pythonic interface, making it easier to learn and use, especially for Python developers.	**No Built-In Support for Large-Scale Distributed Systems:** Unlike TensorFlow, PyTorch has less out-of-the-box support for large distributed systems, though extensions like PyTorch Lightning are helping bridge this gap.

Table 13: Pros and Cons of PyTorch as main framework

Final Decision and Reasoning (TensorFlow as base framework)

TensorFlow is chosen over PyTorch for image-based fatigue analysis due to its static computation graphs, which optimise the execution efficiency of complex models, hence making it suitable for deployment in production environments where performance and scalability are critical (i.e. in workplaces).

ii) Visualisation (SKlearn)

Scikit-learn includes algorithms for tasks such as classification, regression, clustering, and dimensionality reduction. However, in this case, it will be used to create analysis metrics such as a confusion through combination of the seaborn library to create the correlation matrix heatmap. This is used to analyse the accuracy of specific classes.

Pros	Cons
Integration with Matplotlib: Scikit-learn integrates seamlessly with Matplotlib, facilitating the easy creation of visualisations for model evaluation and data analysis.	**Limited Visualisation Capabilities:** Direct visualisation capabilities within Scikit-learn are limited; it primarily relies on external libraries for more sophisticated visualisations.
Helpful Visual Tools: It provides tools like confusion matrices, ROC curves, and other plotting utilities that are useful for quickly visualising complex data relationships and model performance.	**Not Specialised for Real-Time Visualisation:** Scikit-learn is not designed for real-time visualisation tasks. It's more suited for static data analysis and requires additional tools for dynamic, real-time data visualisations.

Table 14: Pros and Cons of SKlearn for Visualisation

Reasoning and relevance to project (SKlearn)

Creating confusion matrices using SKlearn will allow for analysis of the accuracy of specific classes. This will allow us to determine which class is underperforming and requires further fine-tuning/adjustments. For example, images in one class may be noisier than images in another.

iii) Visualisation (Matplotlib)

Matplotlib is a widely used Python library for creating static, interactive, and animated visualisations in Python. It works by constructing figures or plots with layers of elements, such as axes, lines, text, and other graphical representations.

Pros	**Cons**
Flexibility: Matplotlib offers extensive customisation options, allowing you to create a wide variety of plots and charts.	**Complexity with Interactivity:** While capable of interactive plots, Matplotlib can be complex and less intuitive when creating highly interactive or real-time visualisations.
Wide Usage: It's widely used in the Python community, supported by comprehensive documentation and examples.	**Performance:** It may not perform well with very large datasets or in scenarios requiring high-speed updating without optimisation.

Table 15: Pros and Cons of Matplotlib for Visualisation

Reasoning and relevance to project (Matplotlib)

Matlibplot will be used to create graphs which visually show performance. I.e. recall, precision, accuracy, loss etc.

iv) Parallel Processing (CUDA vs OpenCL)

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by (NVIDIA, 2007). It provides the ability to direct GPU acceleration for specific tasks, which improves computing performance for applications that are suitable for parallel processing (i.e. image analysis). It works by enabling numerous threads in parallel and distributing them across multiple GPU cores. This performance improvement is vital in matrix and vector computations.

Pros	Cons
High Performance: CUDA enables high-performance computing by allowing direct access to the virtual instruction set and parallel computational elements of NVIDIA GPUs, maximising throughput for complex calculations.	**NVIDIA Hardware Required:** CUDA is specifically designed for NVIDIA GPUs and does not work on hardware from other manufacturers, limiting its applicability.
Wide Adoption: It is widely adopted in both academia and industry for GPU-accelerated computing, supported by extensive documentation and a robust ecosystem.	**Complexity:** Programming in CUDA can be complex due to the need to manage memory and optimise parallel execution paths explicitly, which requires a deep understanding of both the hardware and the problem domain.

Table 16: Pros and Cons of CUDA for parallel processing

OpenCL (Open Computing Language) is a framework that enables parallel programming across various compute devices like CPUs, and GPUs. It achieves parallel processing by defining kernels, which are functions written in a C-like language that execute across multiple compute units simultaneously on devices such as GPUs. It is not bespoke to Nvidia GPUs (it also works for AMD).

Pros	Cons
Cross-Platform: OpenCL provides a framework for writing programs that execute across heterogeneous platforms including CPUs, GPUs, and other processors.	**Complexity:** Writing efficient OpenCL code can be complex and requires a good understanding of parallel computing concepts.
Flexibility: It supports a wide range of devices from various manufacturers, allowing for broad hardware compatibility.	**Performance Variability:** Performance can vary significantly between different hardware implementations, which may lead to inconsistent results across platforms.

Table 17: Pros and Cons of OpenCL for parallel processing

Final Decision and reasoning (CUDA as parallel processing)

CUDA is used because it enables parallel processing on NVIDIA GPUs (RTX 2060 used in this case), which significantly accelerates computations of analysis tasks. The use of CUDA will lead to faster image processing and analysis times. CUDA is also specifically made for Nvidia GPUs making it more optimised than OpenCL. CUDA was chosen over OpenCL because it is specifically optimised for Nvidia GPUs which has been used in this case. It is also closely integrated with Nvidia graphics drivers, whereas OpenCL has a broader architecture.

v) Base Model (MobileNet vs AlexNet vs Resnet)

The **MobileNet** architecture utilises depth-wise separable convolutions, a technique that breaks down the standard convolution into a depth-wise convolution and a pointwise convolution. This approach reduces the model's complexity and computational cost by significantly decreasing the number of parameters and the computational burden, compared to traditional convolutional networks.

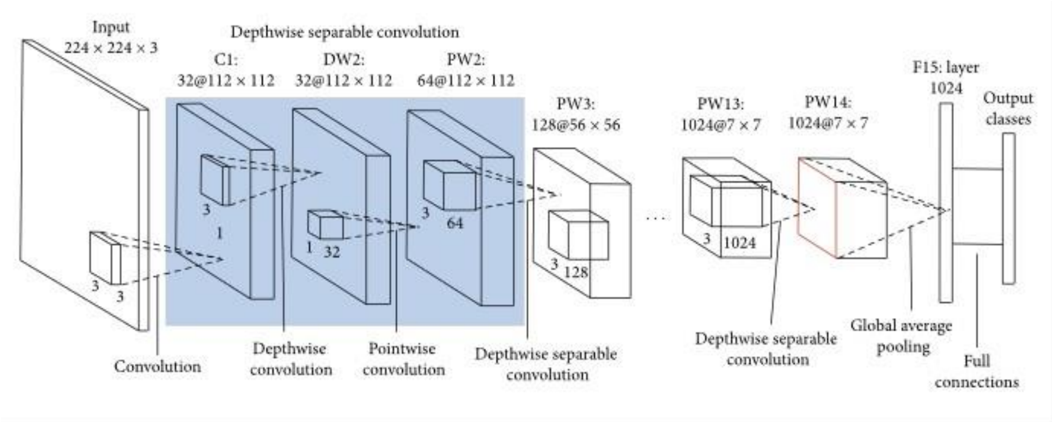


Figure 4: MobileNet Architecture

Pros	Cons
Small Minimum Input Size: MobileNet is designed with small image sizes and a streamlined architecture, making it ideal for environments with limited computational resources, such as mobile and embedded devices.	**Accuracy Trade-off:** While MobileNet is compact and efficient, it typically offers lower accuracy compared to larger, more complex models, which may be a critical drawback for some applications.
Efficiency: Optimised for speed and low power consumption, MobileNet performs well in real-time applications on devices with limited hardware capabilities.	**Limited Customisation:** The streamlined nature of the model can limit the extent to which it can be customised or fine-tuned for specific tasks outside of general image classification.

Table 18: Pros and Cons of MobileNet as the base model

AlexNet, introduced by (Krizhevsky et al. 2012), consists of 8 layers, whereas ResNet (introduced by He et al., 2015) presented a much deeper architecture with variants ranging from 18 to 152 layers. The depth of ResNet allows it to learn more complex patterns and achieve higher accuracy on challenging datasets. Another disadvantage of AlexNet is its lack of residual blocks which help combat the vanishing gradient problem. Hence the final choice of MobileNet as the base model.

Pros	**Cons**
High Accuracy: AlexNet is known for its high accuracy in image classification tasks, making it a reliable choice for applications requiring robust visual recognition.	**Resource Intensive:** AlexNet requires significant computational resources, which can be a limitation in real-time or resource-constrained environments.
Influential Architecture: As one of the pioneering deep learning architectures, AlexNet has influenced numerous subsequent models, and its design is well-understood in the computer vision community.	**Outdated for Some Tasks:** Newer architectures may offer improved efficiency and accuracy, making AlexNet less optimal for cutting-edge applications.

Table 19: Pros and Cons of AlexNet as base model

RESNET50 (He et al., 2015) first introduced the idea of Residual networks and deep residual learning in his seminal research paper for Microsoft. The below outlines the difficulties previously faced by deep learning, then followed by the solution.

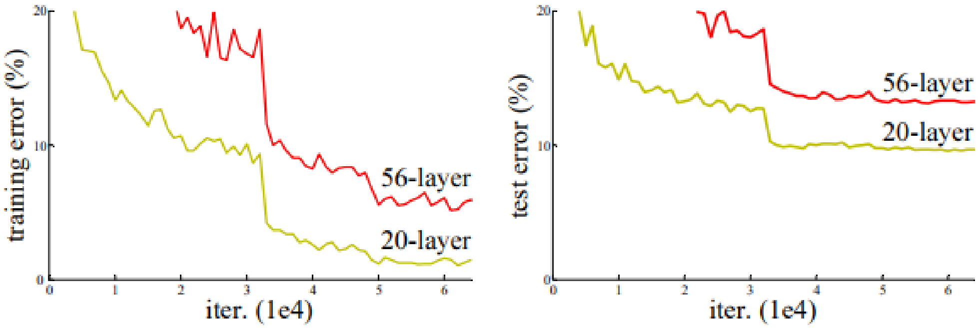


Figure 5: Error rates of 20 layers vs 56 layers (He et al., 2015)

The graph shown above outlines a visual representation of the fact that a deeper layered network results in a much lower rate of error, however, the increase in layers usually leads to a significant decrease in performance due to degradation.

This paper then outlines the solution to be:

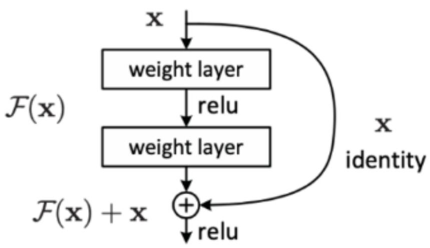


Figure 6: Residual learning block (He et al., 2015)

The paper introduces the idea of a residual learning block. This allows the neural network to learn residual functions, rather than attempting to learn the expected output directly, resulting in a reduction in error rates.

It works as the following:

H(x) represents the desired underlying mapping we wish to learn. Instead of attempting to learn H(x) directly, you would learn the residual function F(x). This is the difference between the desired output and the input:

F(x) = H(x) - x

Now, the output of the residual block is not H(x), instead the input x plus the residual F(x):

Output = x + F(x)

Pros	Cons
Deep Learning Efficiency: ResNet-50 introduces residual learning to alleviate the vanishing gradient problem, allowing it to train deeper networks effectively and efficiently.	**Computational Demand:** While efficient for a deep network, ResNet-50 still demands considerable computational resources, which can be challenging for real-time applications without powerful hardware.
Widely Adopted: It's widely used and proven across various industries, offering strong performance and generalisability for a broad range of image classification tasks.	**Complexity:** The depth and complexity of the model may make it overkill for simpler tasks, where simpler models could suffice with less resource expenditure.

Table 20: Pros and Cons of Resnet50 as the base model

Final Decision and reasoning (MobileNet as base model)

Though resnet50 and AlexNet are more accurate than MobileNet, they require a minimum input image size of 224 * 224. Upon experimenting the relatively small dataset combined with a 224 * 224 image input led to overfitting as key facial features were not as emphasised as they could have been. MobileNet requires a minimum image size input of 128 * 128 which significantly reduces the emphasis on irrelevant information (in image) such as a person's eyebrow size.

Chapter 3. Methodology/Requirements/Design

This chapter delves into the structured methodologies, requirements, and design strategies implemented in the development of the facial fatigue detection system. It covers the selection and application of software development methodologies, project management tools, and detailed design processes tailored to both real-time and still-image components of the project.

3.1 Methodology and Project Management tools

This section discusses the methodologies and tools used for project management and development. It provides insights into the decision-making process for choosing specific approaches and tools, considering the project's complexity and the need for flexibility and iterative development.

3.1.1. General Methodologies

Here, we explore the broader software development methodologies considered for the project. This subsection outlines various approaches, such as Waterfall and Agile, and discusses their applicability, strengths, and weaknesses in the context of developing a facial fatigue detection system. The focus is on how these methodologies can support the project's goals of adaptability and responsiveness to feedback.

3.1.1.1. Waterfall Methodology

The **waterfall methodology** is a renowned SLDC and is known for its rigidness and inability to make changes based on client (supervisor recommendations in this case) needs.

Pros	**Cons**
Structured and Predictable: The Waterfall model is highly structured with clear milestones and deadlines. This predictability helps in planning and allocating resources efficiently.	**Inflexibility:** Once a phase is completed, it's difficult to go back and make changes without redoing the entire subsequent work. This makes it poorly suited for projects where requirements are expected to evolve.
Simplicity and Clarity: Each phase has specific deliverables and a review process, which makes it easy to understand and manage projects with well-defined requirements that are unlikely to change.	**Late Testing Phase:** Testing occurs late in the process, which means any problems or bugs found are only discovered at the end of the development cycle, potentially leading to significant delays and increased costs.
Strong Documentation: The requirement of comprehensive documentation at each phase ensures that every aspect of the project is well-documented, facilitating easier handovers and references.	**Poor Adaptation to Change:** The Waterfall model assumes that all requirements can be specified upfront and will not change, which is often unrealistic in dynamic environments where requirements evolve based on market or technological changes.

Table 21: Pros and Cons of waterfall as methodology

3.1.1.2. Agile Scrum Methodology

Agile Scrum is a subset of agile that emphasises flexible development and the idea of adapting development, design, etc based on client changes. The process involves development cycles (sprints) which conventionally last two weeks. User stories are prioritised as needed, and as a result, client satisfaction increases. Scrum also generally consists of daily standups in professional environments where developers are required to stand up to ensure that meetings are kept short and get straight to the point.

Pros	Cons
Flexibility and Adaptability: Agile allows for changes in project requirements and scope, adapting to feedback from stakeholders and changes in the market environment throughout the development process.	**Less Predictability:** Agile projects can be less predictable in both timeline and budget since changes and adaptations are expected and encouraged throughout the project lifecycle.
Continuous Improvement: Frequent iterations and regular feedback loops with stakeholders ensure continuous improvement and refinement of the product, which can lead to higher customer satisfaction and better end results.	**Requires More Client Involvement:** Agile demands significant client involvement throughout the development process, which can be a challenge if clients are not available, unwilling, or unable to commit the necessary time.
Faster Time to Market: By focusing on the delivery of individual features in short cycles or sprints, Agile can reduce the time to market, allowing organisations to benefit from earlier product launches and adjustments based on user feedback.	**Scope Creep:** Due to its flexible nature, there's a risk of scope creep where features or requirements continually evolve, possibly leading to projects moving away from original goals or becoming unmanageable without proper control mechanisms.

Table 22: Pros and Cons of agile as methodology

3.1.1.1. Reasoning for choosing methodology (Agile)

Agile was chosen for its advantage of flexibility in requirements. It has been taken into account that the strict deadline must be met.

3.1.2. Project management tools

This section evaluates the project management tools (Jira and Trello) which can be used to streamline the development of the facial fatigue detection system, focusing on their features, usability, and effectiveness in managing the project's workflow and timelines.

3.1.2.1 Jira

Jira is a project management tool developed by Atlassian, primarily used for issue tracking, bug tracking, and agile project management (Atlassian, 2023). It supports both agile (scrum) and kanban. It also allows teams to create user stories, issues, epics and sprints.

Advantages

- **Detailed Task Management:** Ideal for a complex project like facial fatigue detection software, where you may need to manage various tasks such as coding, testing, and documenting algorithms.
- **Agile Project Management:** If adopting an Agile methodology, Jira supports this with features like scrum boards and sprints, which could be beneficial for iterative testing and refinement of your detection algorithms.
- **Integrations:** Jira offers integrations with various development tools that you might use for coding and version control, such as Git, which can streamline workflow.

Disadvantages

- **Complexity:** Jira might be overkill for a solo project due to its complex setup and features designed for larger teams. The learning curve could divert time away from actual project development
- **Cost:** While there is a free tier, if additional features are required, Jira can become costly for a student or individual.
- **Performance Overhead:** The comprehensive features of Jira might slow down its performance, particularly if using a lower-powered computer

3.1.2.2 Trello

Trello is a web-based Kanban-style list-making application, also owned by Atlassian. Through experience I have noticed it is much simpler to configure and use, however comes with minimal customisation features.

Advantages

- **Simplicity and Visual Organisation:** Trello's Kanban boards provide a visual overview of the project at a glance, making it easy to track progress on different tasks such as data collection, algorithm development, and testing.
- **Ease of Use:** With Trello, you can quickly set up a board and start adding tasks without a steep learning curve, which is ideal for a solo project where you want to minimise setup time and start working immediately.
- **Flexibility:** You can easily adapt Trello boards to fit the project's needs such as tracking research phases, software.

Disadvantages

- **Limited Built-in Features for Software Development:** Trello lacks some of the advanced project management features like burndown charts or detailed issue tracking, which might be useful for more complex software development projects.
- **Scalability**: If the project expands or we decide to add more features or even collaborate with others later, Trello might not scale as well as Jira in managing increased complexity.
- **Minimal Reporting Tools:** Trello's basic reporting tools might not provide enough insight into project metrics or performance, which could be a limitation if a detailed analysis of progress and productivity is needed.

3.1.3. Real-Time software Methodology

This section focuses specifically on the method applied directly (rather than broader methodologies such as waterfall) to develop the real-time software component of the facial fatigue detection system.

1. **Video Input Capture**: The system begins by capturing video input through one of two webcams. This will be achieved using a simple video capture library that can interface with camera hardware, such as OpenCV for Python.

2. **Face Detection**: The software must first detect the presence of a face in the video frames. This can be done using pre-trained models or algorithms that identify human faces in images, i.e. Haars Cascade.

3. **Facial Landmark Detection**: Once a face is detected, the next step is to identify specific points on the face known as facial landmarks. These landmarks help in tracking the movements of the eyes and mouth. MediaPipe will be used for this.

4. **Tracking Eye and Mouth Movements**: With facial landmarks identified, the software focuses on the landmarks around the eyelids and the mouth to monitor for signs of fatigue:
   - Blink Detection: The system tracks the distance between the eyelid landmarks to determine when the eyes are open or closed. An increase in blink rate or longer duration of eye closures can indicate fatigue.
   - Yawn Detection: Similarly, the software measures the distance between the landmarks around the mouth. A significant increase in this distance might indicate yawning, another sign of fatigue.

5. **Analysis and Decision Making**: The system continuously analyses the data from the landmark tracking to calculate the frequency of blinking and yawning. Based on predetermined thresholds (like a certain number of yawns or blinks per minute), the system can decide if the person is showing signs of fatigue.

6. **Alert Generation**: If the system detects signs of fatigue, it can then trigger alerts to notify the user or take other predefined actions. These alerts can be visual, auditory, or sent as messages depending on the application's design.

7. **User Interface**: A user-friendly interface can be developed to display real-time analysis, settings for sensitivity (thresholds for yawning and blinking), and alerts. This interface helps in managing and configuring the software according to different environments and user preferences.

8. **Continuous Improvement**: Incorporate feedback mechanisms to refine detection algorithms and improve the accuracy and reliability of the system. This might involve gathering more data on user behaviour or tweaking the algorithm based on user feedback and performance.

3.1.4. Still-Image classifier Methodology

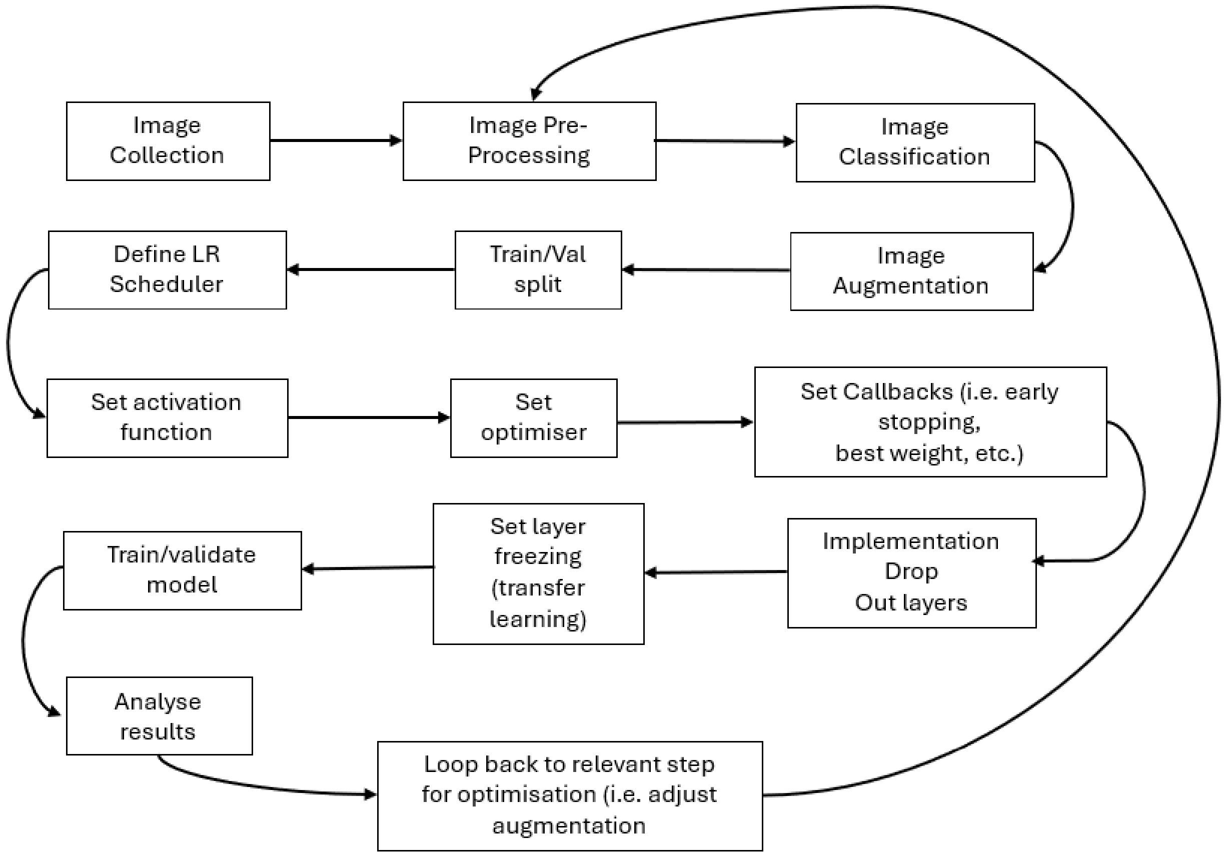


Figure 7: Still-Image methodology

The CRISP-DM methodology is generally followed when conducting data mining tasks, however, in this case, a custom methodology may be easier to understand and follow. The methodology outlined for the Still-Image Classifier employs a structured approach, starting with Image Collection, followed by Image Preprocessing and Classification. Within the workflow, learning rate scheduling and activation functions are defined early, setting the stage for model optimisation. The model is to be fine-tuned with transfer learning, and results are analysed for performance evaluation. This cycle iterates as necessary, emphasising flexibility in returning to previous steps for adjustments.

3.3 Requirements and analysis

3.3.1. General (for both Real-Time and Static Images)

- Gather detailed requirements, both functional and non-functional by identifying problems and shortfalls with currently available solutions.
- Define clear objectives for the software, understanding the user needs and the problem space.
- Document and agree upon the requirements with all stakeholders to ensure clarity and alignment.

3.3.2. Real-Time software

This section will outline the function and non-functional requirements for the real-time software. The functional requirements are generally specific, whereas the non-functional requirements are the requirements that do not impact the function of the system as much (i.e. cost)

3.3.2.1. Functional and nonfunctional requirements

Functional Requirements

Functional Requirement	**Description**
Camera and Video File Input	• Support real-time video capture from two different webcam models, ensuring compatibility with inexpensive webcams. • Allow input of video files in common formats (e.g. .mp4, .Avi) for analysis via environment variables.
User Calibration	• Provide a user-friendly interface for first-time users (and existing users) to calibrate their eye and mouth movements, establishing personalised thresholds for blink rate and yawn frequency.
Calibration Data Management	• Enable users to save their calibration data/profiles on the system. • Allow users to easily load and switch between saved profiles containing their unique calibration information.
Fatigue Alerts	• Notify users in real-time when a predefined fatigue threshold is reached, using visual, auditory or messaging alerts. • Allow users to customise the fatigue threshold levels and the type of alert received.
Data Output	• Provide an option to export detailed session data, including blink rate and yawn frequency, to a .csv file for further analysis or record-keeping.

Table 23: Real-Time Functional Requirements

Non-Functional Requirements

Non-Functional Requirement	**Description**
Performance	• Ensure real-time processing of video data with minimal latency to provide timely fatigue alerts. • Optimise facial landmark detection algorithms for speed and accuracy, even on lower-end hardware.
Scalability	• Design the software to easily incorporate additional fatigue indicators in the future, such as head pose estimation.
Usability	• Develop an intuitive user interface that can be easily navigated by users of all technical skill levels.
Usability	• Develop an intuitive user interface that can be easily navigated by users of all technical skill levels.
Reliability	• Ensure the software runs consistently over extended periods without crashes or significant performance degradation. • Implement error handling to manage and log failures or anomalies in video processing.

Table 24: Real-Time Non-Functional Requirements

3.3.3. Static image

This section outlines the functional and non-functional requirements for the static image classifier model.

3.3.3.1. Functional and nonfunctional requirements

Functional Requirements

Functional Requirements	**Description**
Data gathering	• Gather a large dataset of images and categorise them to respective classes
Data Preprocessing	• Resize or crop images to match the input size expected by MobileNet (224x224 pixels is a common choice, but fine-tuning suggested otherwise (discussed in implementation)). • Augment the dataset through techniques like rotation, zoom, flip, etc., to increase the diversity of the training data.
Training Process	• Split the 1200 images into training and validation sets (70% training, 30% validation). • Employ a suitable loss function that can handle multi-class classification (e.g., categorical cross entropy). • Implement early stopping to prevent overfitting, and monitoring validation loss with a patience parameter.
Transfer Learning Setup	• Utilise a pre-trained MobileNet as the base model, keeping its convolutional base frozen during the initial training phase. • Add a custom classification head on top of the MobileNet base to distinguish between the three levels of fatigue.
Evaluation Metrics:	• Report accuracy, precision, recall, and F1-score, etc. • Provide confusion matrices to understand misclassifications between the fatigue levels.
Deployment	• Be exportable to a format suitable for deployment in a production environment (TensorFlow Saved Model).

Table 25: Static-Image Functional Requirements

Non-Functional Requirements

Non - Functional Requirements	**Description**
Performance	• Achieve a classification accuracy of at least 55% on the test set. • Process a single image and return a prediction within a specified time frame suitable for the application's context (under 10 seconds).
Scalability	• Be capable of retraining with additional data without significant degradation in performance. • Efficiently utilise GPU resources during training and inference if available.
Maintainability	• The codebase should follow industry-standard coding practices and guidelines for readability and maintainability.
Ethics and Privacy	• Ensure that the model does not inadvertently introduce or perpetuate bias towards any group of individuals. • Comply with data protection and privacy regulations relevant to the application's use case and geography, especially regarding the use of personal or sensitive images.

Table 26: Static-Image Non-Functional Requirements

3.3.4 Project schedule (Gantt chart)

This section presents the project schedule using a Gantt chart, detailing the timeline for each phase of the facial fatigue detection system's development. It visualises the start and end dates for tasks such as design, development, testing, and deployment, providing a clear overview of project milestones and deadlines.

	Task Name	Start	Finish	Duration	
1	Requirements and analysis	1st Nov 2023	30th Nov 2023	4 weeks	
2	Design System	24th Nov 2023	10th Jan 2023	6 weeks	
3	Implement system	20th Dec 2023	15th Feb 2024	8 weeks	
4	Testing	10th Feb 2024	8th March 2024	4 weeks	
5	Deployment	7th March 2024	31st March 2024	3 weeks	
6	Maintenance and updates	1st April 2024	15th April 2024	2 weeks	
7	Project Report	15th Feb 2024	20th April 2024	9 weeks	

Figure 8: Implementation plan Gantt chart

The development of this project started with a 4-week period to understand what was needed and to lay out a plan. After that, the 6 weeks were spent working on the design of the system. The first 9 weeks covered planning and design, followed by an 8-week phase where the actual system was built. This was followed by a 4-week period for testing, to make sure everything worked correctly. 2 weeks were spent making any necessary adjustments and improvements after the system was deployed.

3.4 System Design

This section outlines some system design, program logic and UI design. The code design (the logic behind implementation) is discussed in the implementation section with code referenced to the appendix (as advised by primary supervisor)

3.4.1. Real-Time Software

The following section will provide a low-fidelity view of the real-time system, along with the program logic and the camera set-up

3.4.1.1. Low fidelity view

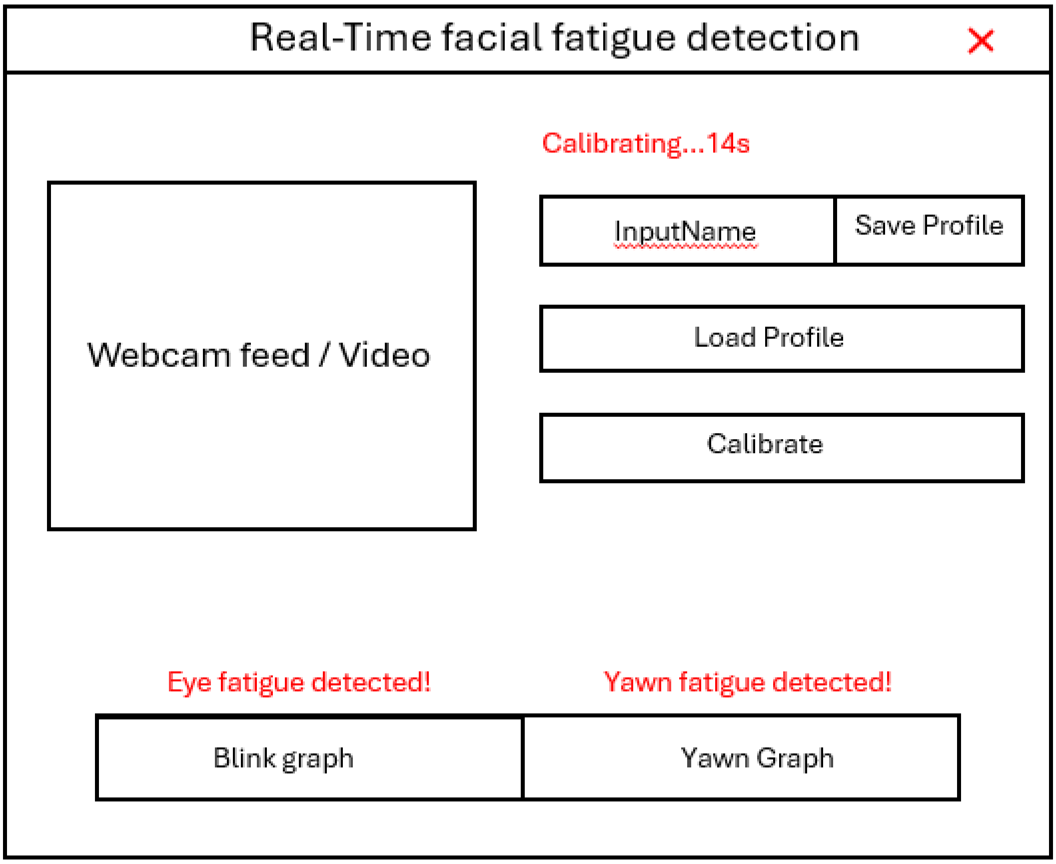


Figure 9: Low fidelity plan for Real-Time

The wireframe outlines the facial fatigue detection software's UI, with a section for webcam input to track the user's face, and options to input, save, and load user profiles for customised settings (blink/yawn calibrations). It includes a calibration countdown, necessary for setting baseline measurements of the user's blinking and yawning. The UI also features alerts for eye and yawn fatigue detection and corresponding graphs that plot blink and yawn frequencies, indicating these metrics are continuously analysed and logged by the system for fatigue monitoring.

3.4.1.2. Program Logic

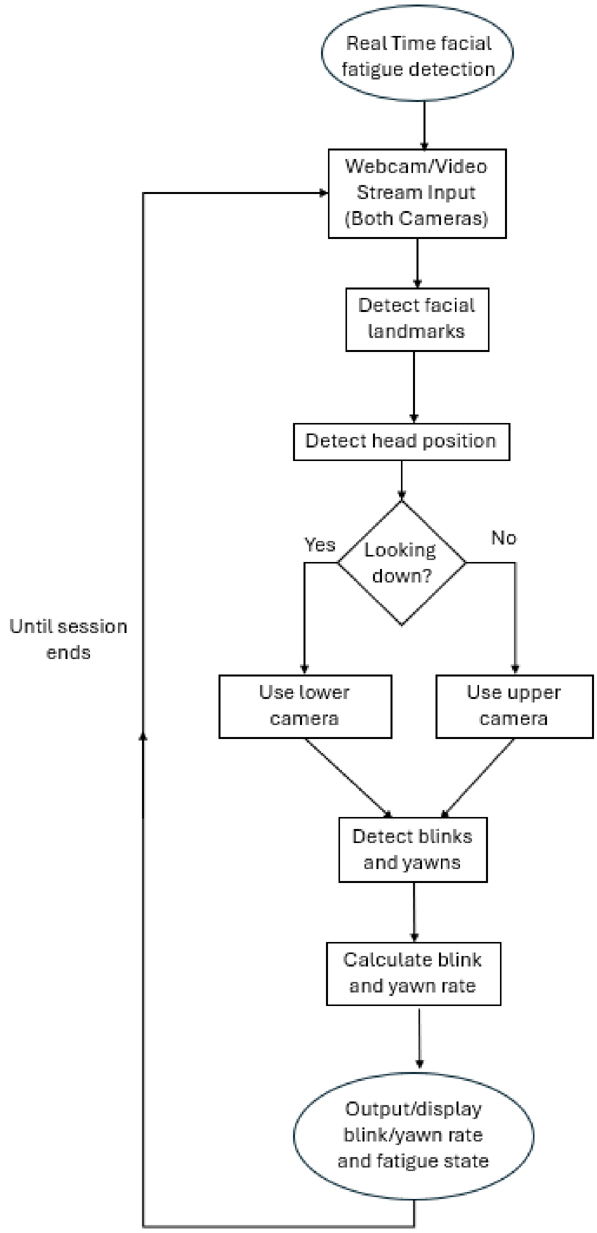


Figure 10: Overview of Program Logic

The system employs computer vision to detect facial landmarks. Two webcams or one video can be inputted. It concurrently assesses the head's position to determine the user's eye/gaze direction. If the head is angled downwards, the system switches to a secondary camera positioned to better capture the face from a lower angle. Otherwise, it maintains or switches to the primary top camera. The software (in parallel) identifies eye blinks and yawns, subsequently calculating the blink rate in blinks per minute (bpm) over an initial two-minute interval. A threshold is set at 1.2 times the detected bpm, and the system continuously compares the current bpm against this threshold. If the bpm exceeds the threshold, an alert is triggered to indicate potential eye fatigue; if not, the system remains in a monitoring state without action.

3.4.1.3. Choosing best Webcam (top or bottom) for head tilt

It is important to select the best camera depending on the 'pitch ' of head (ie tilted up or down). Selecting the correct webcam will give the best view of the eye and mouth and hence improve the measurement accuracy of blink and yawn frequency.

The Webcam can be selected by measuring the distance between the eye and nose tip (see Figure 11 below).

If $d_t < d_b$ then bottom webcam is selected

If $d_t > d_b$ then top webcam is selected

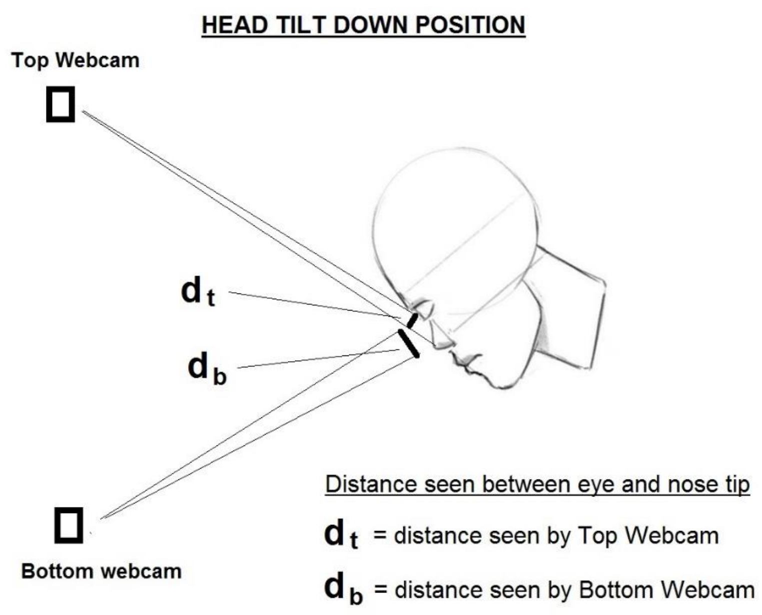


Figure 11: Choosing webcam (top or bottom)

3.4.2. Static image classifier (ML)

The following section will outline the model design, including model inputs for training, outputs for predictions, and frozen layers for transfer learning.

3.4.2.1. Model Design

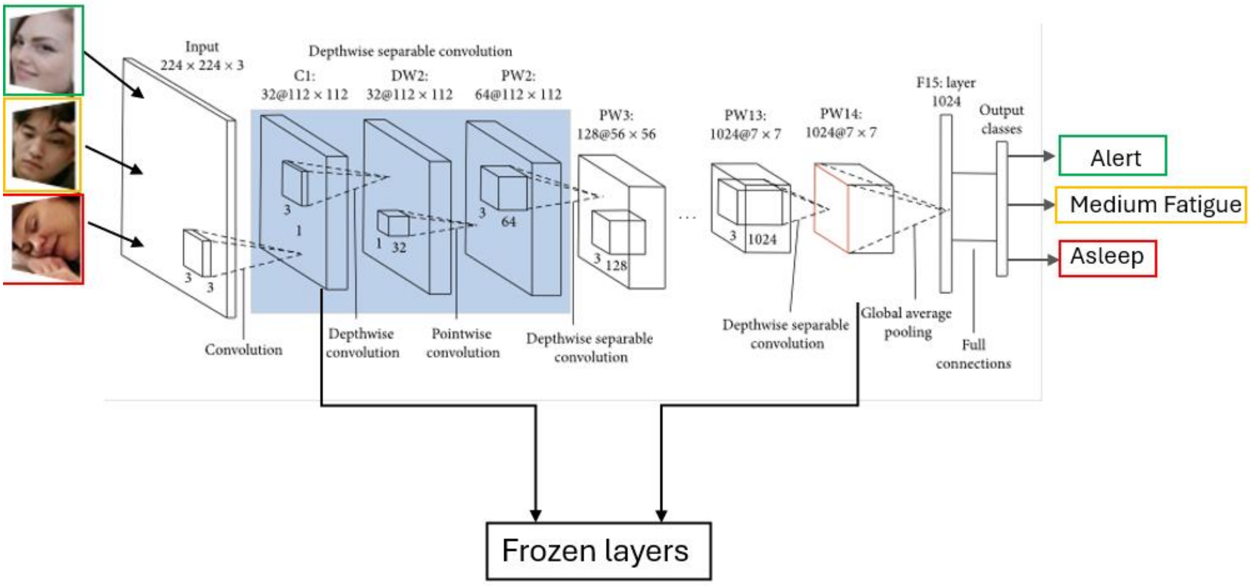


Figure 12: Still image model plan/design

The above depicts the design of the convolutional neural network (CNN) to be used with a specific focus on utilising frozen layers for feature extraction. Initially, input images of size 128x128 pixels are processed through a series of depth-wise separable convolutions, an efficient mechanism that splits the convolutional process into two parts to reduce computational requirements. The network leverages pre-trained weights up to the last depth-wise separable convolution layers, as indicated by the 'Frozen layers' label, meaning these weights are not updated during training. This frozen state allows the network to utilise previously learned feature detectors from large datasets to recognise general patterns and shapes within new images. It is also frozen to feature extraction. After the frozen layers, the network moves to a global average pooling layer followed by a fully connected layer (not frozen), meaning they are trainable. These layers will be fine-tuned to classify input images into one of three categories: Alert, Medium Fatigue, and Asleep.

3.4.2.2. Data Pre-Processing and optimisation

In the development of the machine learning model for detecting signs of fatigue from still images, a critical component of the system design is the **preprocessing** of image data, which enhances model accuracy and robustness. To address variability in the input data and to simulate a range of conditions under which fatigue might be detected, image augmentation techniques are extensively utilised. These techniques include horizontal flipping, which helps the model generalise across different orientations of facial features, and random shifts in height and width, which train the model to recognize fatigue signs even when parts of the face are slightly occluded or misaligned due to movement or camera angle.

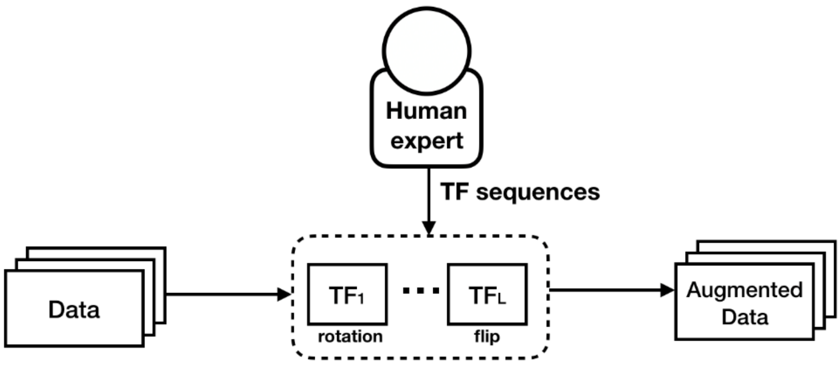


Figure 13: Data Augmentation plan

The diagram provided by (Sharon. L, 2020) shows a data augmentation process in machine learning. A human expert uses a series of transformations, like rotation and flipping, to manipulate original data, creating a larger and more diverse dataset. This enriched dataset is then better suited for training robust machine learning models.

These augmentations are implemented using a pipeline that applies transformations randomly but consistently across images, ensuring that the model is exposed to a diverse set of variations. This approach not only aids in preventing overfitting by broadening the diversity of the training data but also enhances the model's ability to perform reliably in real-world scenarios where perfect alignment and conditions are rare.

In **optimising** the machine learning model for detecting facial fatigue from still images, several advanced techniques were employed to enhance both efficiency and accuracy. A dynamic learning rate scheduler is to be implemented to adjust the learning rate during training phases, starting with a higher rate for rapid convergence and reducing it progressively to fine-tune model weights as the training progressed. This approach will help in avoiding the common pitfalls of overshooting the minimal loss point and facilitated more granular updates to model parameters.

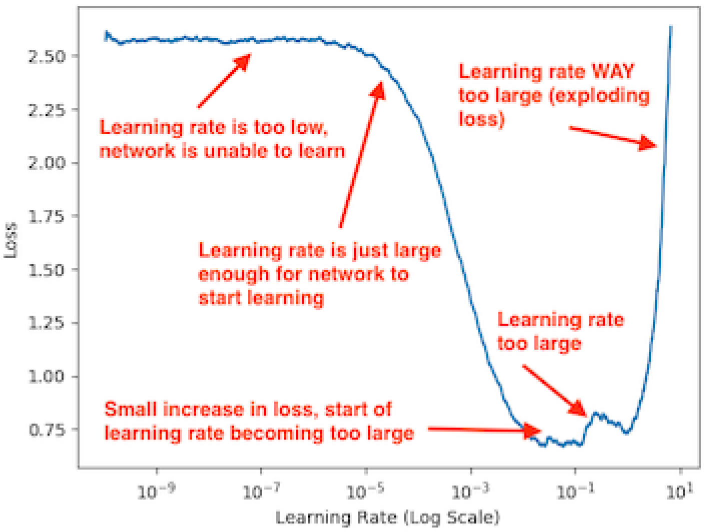


Figure 14: Learning rate guide/plan

The diagram illustrates why selecting the right learning rate is crucial: a learning rate that's too low results in slow or no learning, while a rate that's too high leads to unstable training with exploding loss. An optimal rate allows efficient and stable training, minimising loss effectively.

3.4.2.3. Class Representation

One-Hot encoding

Alert	Medium Fatigue	Asleep
1	0	0
0	1	0
0	0	1

Table 27: One-Hot encoding representation

The table illustrates a one-hot encoding scheme for the three categorical states: 'Alert', 'Medium Fatigue', and 'Asleep'. In this encoding, each state is represented by a binary vector where only one element is '1' (indicating the presence of that state), and all other elements are '0' (indicating their absence). For instance, 'Alert' is encoded as [1, 0, 0], 'Medium Fatigue' as [0, 1, 0], and 'Asleep' as [0, 0, 1].

Categorical Encoding

Fatigue Level	Categorical Encoding
Alert	1
Medium Fatigue	2
Asleep	3

Table 28: Categorical encoding representation

The table shows a simple categorical encoding for three levels of fatigue: 'Alert', 'Medium Fatigue', and 'Asleep'. Each level is assigned a unique integer value: 'Alert' is encoded as 1, 'Medium Fatigue' as 2, and 'Asleep' as 3. This type of encoding is used to transform categorical labels into a numerical format that can be easily understood and processed by machine learning algorithms.

Chapter 4 Implementation/Findings/Analysis

This section outlines the implementation design decisions. The section splits the Real-Time and static programs into two subheadings. The results evaluate both programs together. Code is not shown in the implementation section but is instead referenced in the appendix with code in fixed-width font as advised by supervisor. Full code in on GitHub (See title page).

4.1. Real-Time facial fatigue detection

This section uses diagrams and narratives to show the implementation of the real time software. The implementation for this has been done using the methodology outlined in section 3.1.2.

4.1.1. GUI

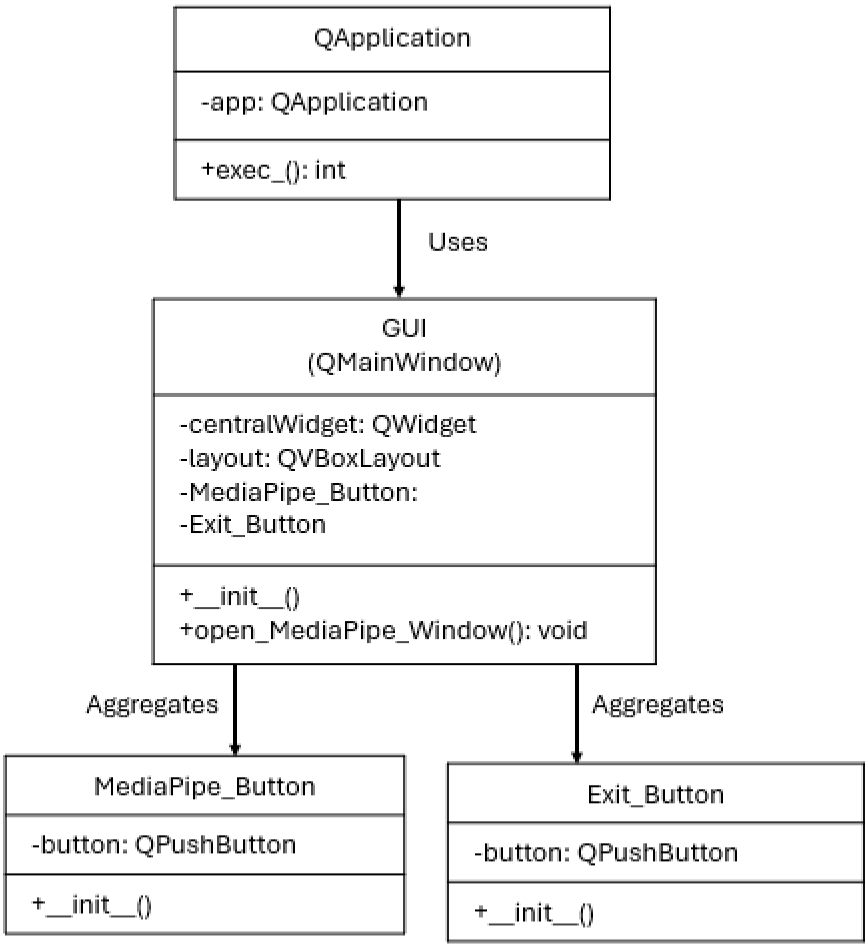


Figure 15: GUI class diagram

- QApplication: Manages the GUI application's control flow and main settings. It has a method exec_() to start the application loop.
- GUI: The main window class inheriting from QMainWindow. It aggregates two widgets: MediaPipe_Button and ExitButton. It also has a method to open a secondary window (MediaPipeWindow).
- MediaPipe_Button and ExitButton: These are custom widgets each containing a QPushButton. The ExitButton has a method connected to QApplication.quit() to close the application.
- MediaPipeWindow: Another QMainWindow that can be launched from MediaPipe_Button.

4.1.2. Threading & Parallel Processing overview

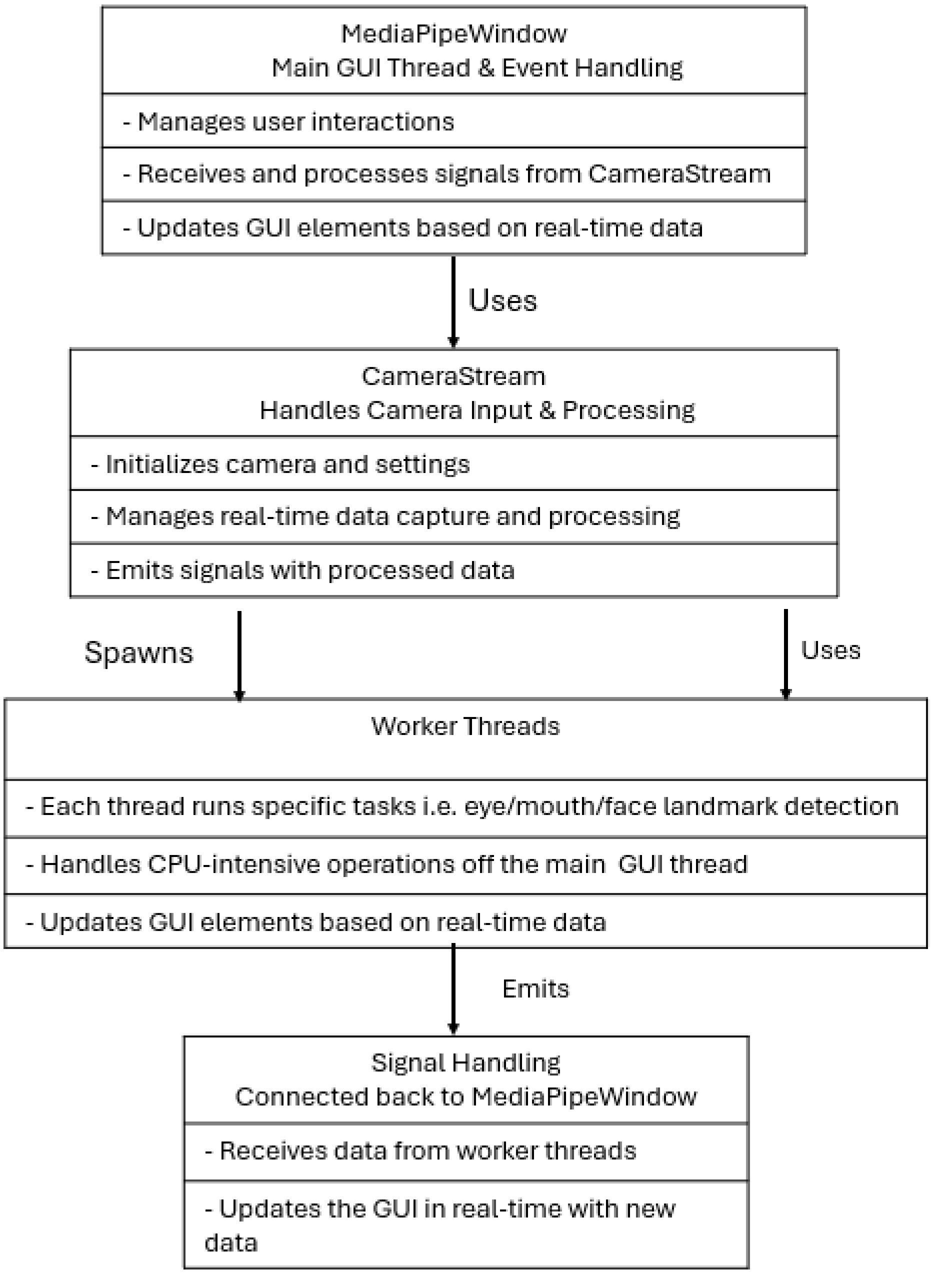


Figure 16: Threading diagram

run(): Starts the camera capture and processing in a loop, which is managed by a QThread. This separation ensures that the GUI remains responsive by offloading the heavy processing to another thread.

service_thread(): Manages the creation and running of threads for specific tasks. It checks if a thread is alive and, if not, starts it. This is crucial for tasks that need to be repeated or maintained throughout the application's lifecycle.

face_feature_detection_worker(): A CPU-intensive function that processes the video frames to detect facial features. It runs in a separate thread managed by service_thread() to ensure that it does not block the GUI.

Signal Handling: Processes signals emitted from the CameraStream threads. Since PyQt signals are thread-safe, they can safely update GUI elements even when emitted from a background thread.

This threading architecture is designed to maximise responsiveness and efficiency in your real-time application by ensuring that the main GUI remains responsive while heavy processing tasks are handled in parallel.

4.1.3. Detecting blinks and yawns

Notation used for head (face) movement

For the purpose of this report, we will use Roll, Pitch and Yaw notation to define the head movement. See diagram x below. The 'forward' arrow defines the front of the face, while the 'back' refers to the back of the head. Using this notation a change in the yaw angle would correspond to a face moving left to right (or vice versa).

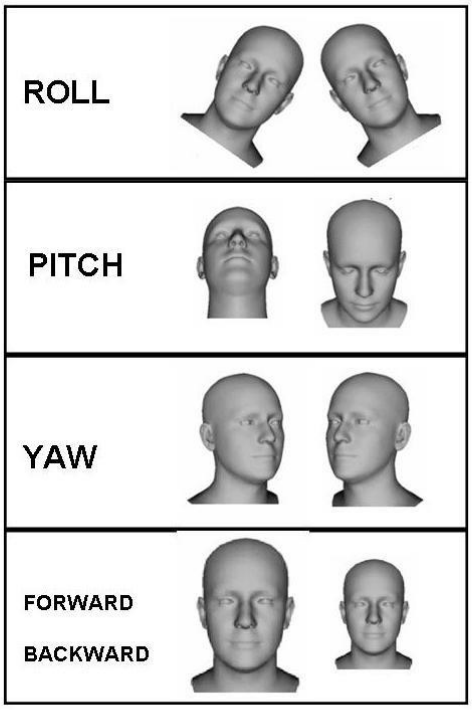


Figure 17: Head movement notation used

4.1.3.1. Initial design (Stage 1)

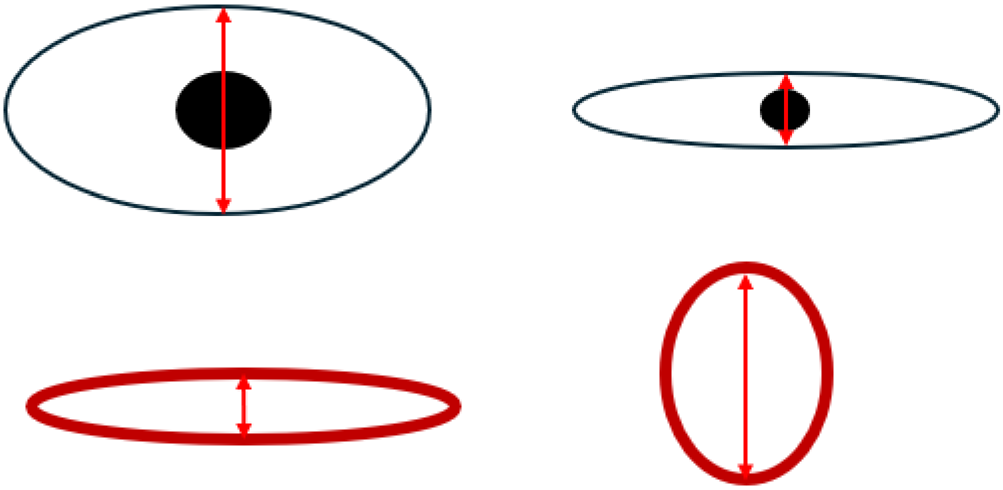


Figure 18: First design to detect blinks

The initial design to detect blinking was to measure the Euclidean distance between the middle of the top and bottom eyelids, calibrate for open and closed eyes, and set the blink flag to true, once a certain threshold had passed. Though this would work in theory, a key problem encountered is by measuring the distance through pixels, if a user were to move away from the camera after calibration, the system may determine it as a blink since the number of pixels between the two points would have decreased.

4.1.3.2. Intermediate design (Stage 2)

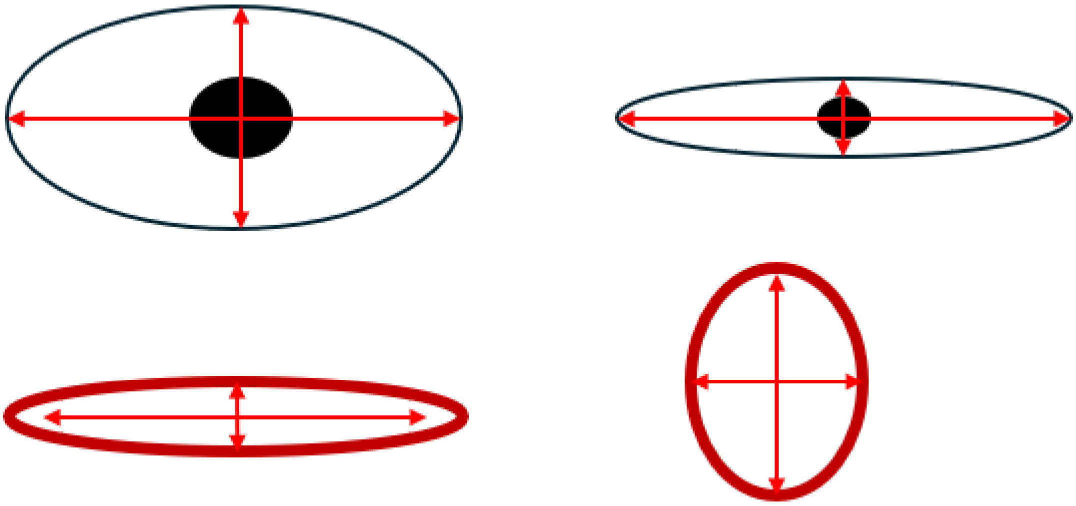


Figure 19: Second design to detect blinks

A better solution is to measure the aspect ratios of the eyes and mouth (see Figure 19). Height/Length. By employing the aspect ratio, the system gains invariance of the user's position relative to the camera. For example, the horizontal axis will decrease proportionally to the vertical axis if the user was to move away from the camera. Additionally, the aspect ratio can be further calibrated over time to account for different lighting conditions, facial expressions, and individual variances in eye and mouth geometry.

However, a key problem with using the aspect ratio as a key indicator of blinks and yawns (when changed past a certain threshold) is that if a user were to tilt their head, it would greatly change the EAR/MAR values, as shown in the diagram below

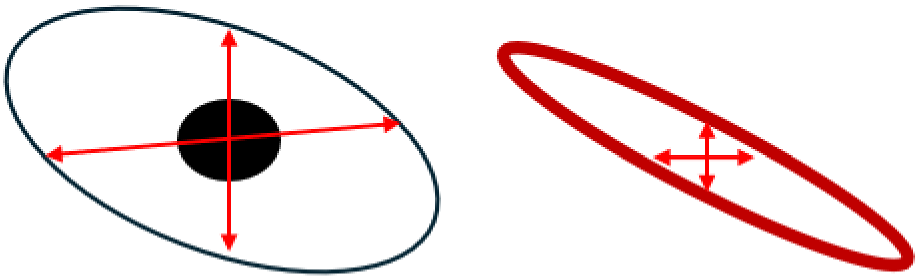


Figure 20: Problem with the second design to detect blinks

If a user were to tilt their head left, the Y axis would increase and the X axis would decrease. This is shown more significantly in the right image where the mouth is closed (compared to figure)

4.1.3.3. Improved final design (Stage 3)

Detecting blinks and yawns using eyes/mouth aspect ratio (AR) AND taking into account head tilt

Top Point =The uppermost landmark on the vertical axis of the eye.

Bottom Point: =The lowest landmark on the vertical axis of the eye.

Left Point = The leftmost landmark on the horizontal axis of the eye.

Right Point = The rightmost landmark on the horizontal axis of the eye.

Distance Calculation:

dx is defined as the Euclidean distance between the top and bottom points. This measures the vertical span of the feature.

dy is defined as the Euclidean distance between the left and right points. This measures the horizontal span of the feature.

Aspect Ratio Calculation:

If dy (the horizontal span) is greater than zero, the aspect ratio (ar is calculated as dx / dy. This represents the ratio of the vertical to horizontal dimensions, which is a straightforward measure of how elongated the feature is vertically relative to its horizontal extent.

The tilt is calculated using the **arctangent** of the slope between the left and right points.

The tilt provides a measure of the rotation of the feature around the frontal axis of the face, reflecting how much the head is tilted. Code in sections 7.1.5 and 7.1.6.

Effect of head movement on aspect ratio

Because the camera is depicting the 3-dimensional head as 2D images, the aspect ratio will change due to head movement. The table below outlines the effect of head movement on aspect ratio.

Changes in aspect radio due to head movement.

	Eye/Blink Measurements			Mouth/Yawn Measurements		
Movement	Width	Height	Aspect Ratio	Width	Height	Aspect Ratio
-						
Roll	same	same	same	same	same	same
Pitch (up-down)	same	change	change	same	change	change
Yaw (left-right)	change	same	change	change	same	change
Forward/Backward	change	change	same	change	change	same

Table 29: Effect of head movements on AR

4.1.4. Calibrating for different eyes/mouth movements

Initially, blinks and yawns were counted using an arbitrary trigger threshold for movement. However, this method was found to be highly inaccurate as everyone's eyes/mouths are different and move by varying amounts when blinking and yawning. Therefore, it was decided early in the project to include a calibration procedure to detect the minimum and maximum eye and mouth openings over a 30-second calibration period. It was decided to limit the calibration period to 30 seconds for practical reasons.

Once the minimum and maximum eye/mouth openings have been measured, a formula is then used to calculate the trigger threshold using a preset sensitivity constant (see section x). The higher the sensitivity constant (σ), the more the eye or mouth has to open to trigger a blink or yawn.

$$\text{Trigger Threshold} = AR_{min} + \left[\frac{[(AR_{max} - AR_{min})}{2} \times \sigma \right]$$

AR = Aspect Ratio for eye/mouth

σ = Sensitivity (1 = full eye/mouth opening)

The calibration period (30 seconds) is used to measure the aspect ratio of the user's bespoke eye and mouth when blinking and yawning.

The calibration routine measures the minimum and maximum eye and mouth opening over a 30-second calibration period. The formula calculates a threshold by adding 30% of the difference between the maximum and minimum aspect ratio (AR) observed over 30 seconds to the minimum AR. This threshold is set at the 30th percentile of the observed AR range and is used to detect significant changes, like blinks and yawns, which signal fatigue.

An array will collect the user's eye and mouth aspect ratios over a 30-second period.

Code in appendix 7.1.6

4.1.5. Average Blink/Yawn Calibration

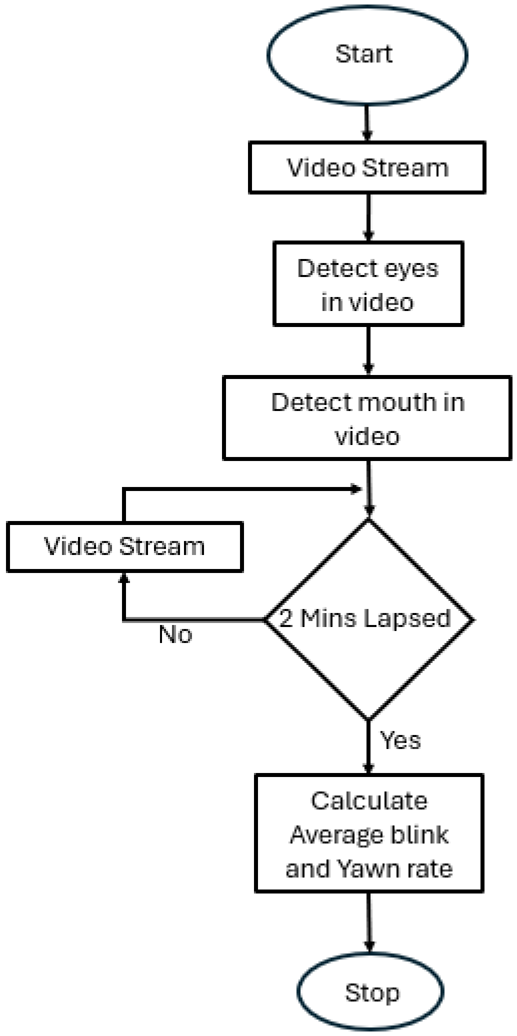


Figure 21: BPM/YPM implementation

The flowchart describes a process that begins with starting a video stream, which then continuously detects eyes and mouths within the video. This is likely part of a monitoring system designed to observe facial features to assess states like fatigue or alertness. After initiating the video stream, the system separately identifies eyes and mouths, probably using image recognition technologies.

The process checks whether 2 minutes have lapsed since the start. If not, it continues to monitor the video stream. Once 2 minutes have passed, it proceeds to calculate the average blink and yawn rates based on the data collected during that time frame. Finally, the process stops after these calculations.

In such a context, an increase in blinks per minute is a critical metric. This increase can indicate fatigue, as frequent blinking is commonly associated with tiredness and the need to refresh the eyes. Monitoring blink rates over time, therefore, can be an effective way to detect early signs of fatigue, particularly in scenarios where alertness is critical, such as in driving or operating heavy machinery.

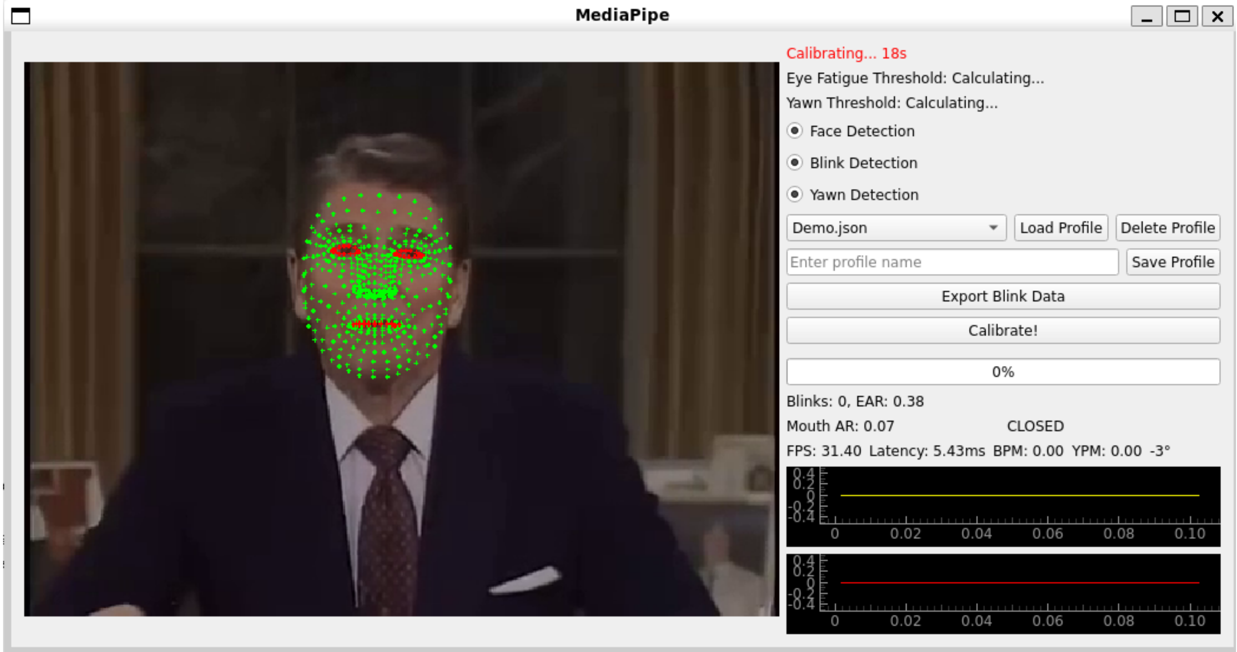
4.1.6. Detecting if user is looking down

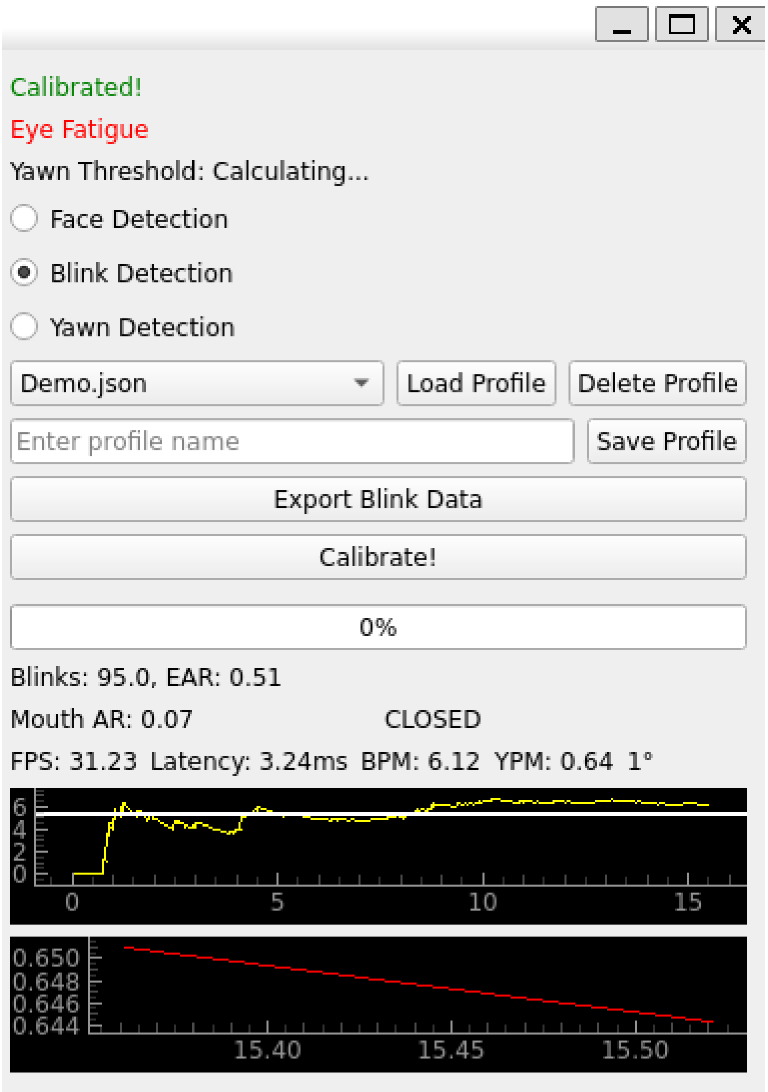
This has been done because if blinking is calibrated when looking up, **if someone were to look down, their eyes would seem squinted, therefore triggering a false positive blink**. The system detects when the user is looking down by focusing on the **normalised** Euclidian distance between the eyes and nose. It calculates the average vertical position of the eyes and defines a threshold for looking down. If the average eye position is lower than the tip of the nose by the defined threshold, it means the person is looking down (code in section 7.1.4)

$$\textbf{Looking Down} = \textbf{Eye Y} < \textbf{Nose Tip Y} - \textbf{Threshold}$$

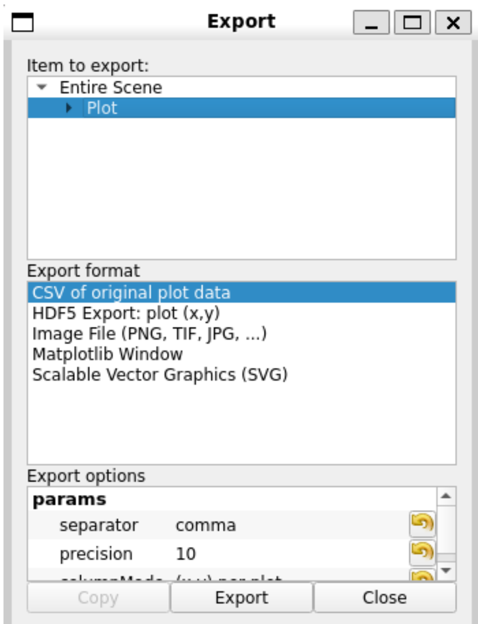
Eye Y is the average y-coordinate of all the landmarks for the left and right eyes.
Nose Tip Y is the y-coordinate of the nose tip.
Threshold is a predefined value.

4.1.7. Final Software






```
{
    "name": "Demo",
    "ear_threshold":
        0.27638681747090854,
    "yawn_mar_threshold":
0.52193726382328
}
```

The real-time software implemented (see section 4.1.7) is designed to measure and track facial dynamics such as blinks and yawns, utilising multi-threading to ensure responsive operation. It accurately displays frames per second (FPS), which is crucial as lower FPS may lead to missed blink detection. The software also calculates and displays blinks per minute (BPM) and yawns per minute (YPM), alongside graphs showing these metrics over time. Additionally, latency metrics are provided to monitor processing delays. The top graph on the GUI shows the BPM, whereas the bottom graph shows the YPM.

Calibration is tailored for individual faces, enhancing detection accuracy, and this configuration can be saved in JSON format for subsequent uses, bypassing the need for recalibration. The software allows for the export of detailed graphical data and Excel files containing session results.

Thresholds for detecting eye and yawn fatigue are set to 1.2 times the BPM and YPM after an initial 120 seconds of monitoring. If current metrics exceed these thresholds, the system triggers alerts for potential eye and yawn fatigue, ensuring timely interventions. The program is set up to switch cameras based on whether the user is looking down. However, the code uploaded on GitHub and in the appendix has been commented out due to testing on pre-recorded videos.

4.2. Still-Image Facial Fatigue Detection

This section will outline the implementation of the still image facial fatigue detection (three states). The subsections are presented as so to follow the methodology presented in section 3.1.3.

4.2.1. Image Classification and Cropping (utility)

After 12,000 stock images had been downloaded for the (initially) four classes, 1,600 were initially manually labelled and put into their respective classes using (CVAT), where bounding boxes were drawn around faces for each image. However, due to the large nature of the dataset and the number of classes (not just binary) and the high computational load of CVAT, a better solution was to create a utility tool (code in appendix 7.2.1) which would display downloaded images and allow you to move them to their respective folders via one click. It was created using Tkinter and all images were resized smaller so that the GUI wouldn't move with each image, making it easier to classify. The faces in these images were then be cropped using a model MCNTT to reduce background noise (code in appendix 7.2.2 and output in figure 24).

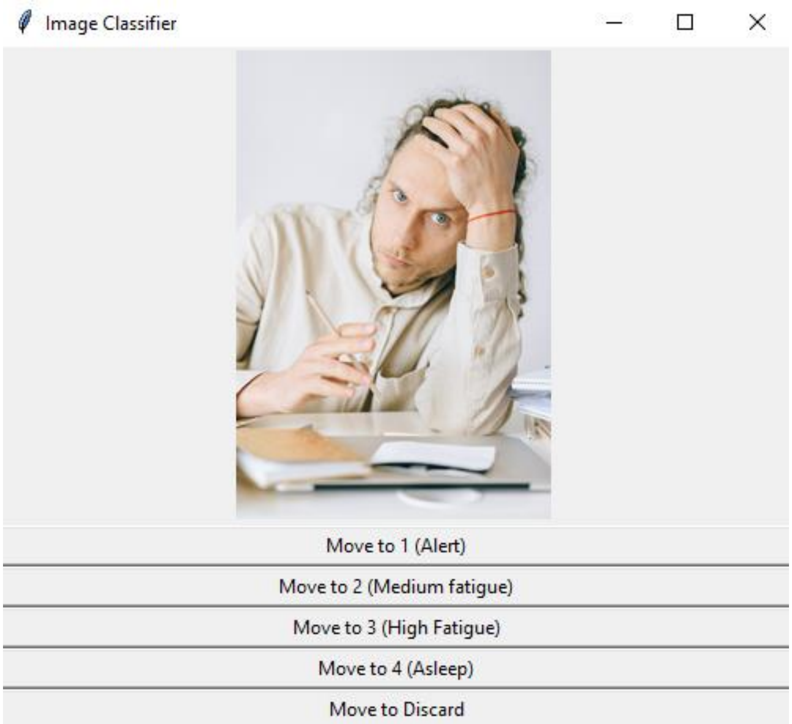


Figure 22: Custom still image classifier utility

4.2.2. Class Distribution and sample images

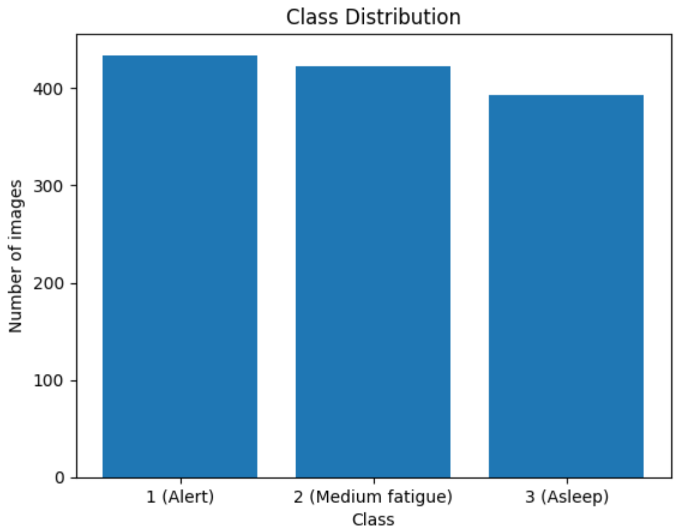


Figure 23: Class Balancing

The class distribution shown in the bar chart is for the facial fatigue detection system, which categorises images into three classes: Alert, Medium fatigue, and Asleep. From the model's perspective, having an even distribution of classes is beneficial because it ensures that the model is not biased toward any particular class due to overrepresentation. This helps in achieving a balanced training process where the classifier learns to recognise each class with equal importance, which can lead to better generalisation when making predictions on new, unseen data.

The distribution for the 'Alert' and 'Medium fatigue' classes has a similar number of images, which is desirable for the reasons mentioned. However, the 'Asleep' class has a slightly lower count. The reason given for this adjustment is that during testing, the model was biased toward the 'Asleep' class. This suggests that the model was more likely to incorrectly classify images into the 'Asleep' category.

To correct this bias, the dataset for the 'Asleep' class was reduced. This was an effective approach because it forced the model to become more certain about the features that are indicative of the 'Asleep' state, rather than overfitting to the overrepresented class.

However, this method of addressing class bias should have been complemented with other techniques such as weighted loss functions, where the model's mistakes on underrepresented classes are penalised more heavily during training, or by using synthetic data augmentation to balance the classes without losing valuable data.

Figure 24: Random sample image from each class

The sample images are shown above (random image from each class). The faces are cropped to focus on the facial features that are most relevant for fatigue detection, such as the eyes, eyebrows, and mouth. Cropping reduces extraneous background information and variability in the dataset, helping the detection algorithm to concentrate on the key indicators of alertness or fatigue. This can (and did) improve the accuracy of the model by minimising distractions and standardising the input data. The close framing also ensures a consistent field of view across all classes, which is crucial for reliable feature extraction and classification in image processing tasks.

4.2.3. Image Augmentation

Augmentation type	Value
Rescale	1./255
Rotation Range	30
Wide Shift Range	0.1
Height Shift Range	0.1
Sheer Range	0.2
Zoom Range	0.2
Horizontal Flip	True
Vertical Flip	True
Fill Mode	Nearest
Contrast	1.8

Table 30: Image Augmentation values

Figure 25: Data augmentation output

The table presents parameters used for data augmentation in image processing. It lists the types of augmentations applied, like rescaling, rotation, shifting, shearing, zooming, flipping, and contrast adjustment (code in appendix 7.2.4). The image augmentation process enhances the diversity of the training dataset by applying various transformations to the original images, which can help improve the robustness and generalisation of a machine learning model. **A key image augmentation step is the image contrast adjustment**. This adjustment makes the difference between the lighter and darker parts of the image more amplified which plays a significant role in preventing overfitting. The contract was increased to make features within the images more pronounced. By emphasising edges and textures, the contrast adjustment helps the model focus on the structural attributes of objects, which are crucial for recognition tasks. It is worth noting that overuse or extreme contrast was tested and led to loss of detail in images.

4.2.4. Train/Val Split

Train	Validation
70%	30%

Table 31: Train and Validation split

The dataset was split into two subsets: 70% for training and 30% for validation (code in appendix 7.2.4). Initially, a 80/20 split was tested, however, during experimentation, the 10% increase in the validation split significantly helped training accuracy and validation overfitting. This split was done to allocate most of the data for the model to learn from while reserving a significant portion to validate the model's performance. The training set is used to fit the model's parameters, and the validation set is used to provide an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters. This practice helped in generalising the mode to new, unseen data.

4.2.5. Learning Rate Scheduler

Epochs	Learning Rate
0-10	0.001
11-20	0.0001
21-30	0.00001
31+	0.000001

Table 32: LR scheduler implementation

This table shows the learning rate schedule used that dynamically adjusts the learning rate based on the epoch number during training (code in appendix 7.2.5). Initially, the learning rate (lr) is set to 1e-3 (0.001). As training progresses, the learning rate is decreased at specific epochs to smaller values, employing a decay strategy:

- After 10 epochs, the learning rate is reduced by a factor of 10 (lr *= 1e-1), setting it to 1e-4.
- After 20 epochs, it is suggested to further reduce the learning rate by a factor of 100 (lr *= 1e-2) from the initial rate, intending to set it to 1e-5.
- After 30 epochs, the reduction factor increases to 1000 (lr *= 1e-3), aiming for a learning rate of 1e-6.

The rationale behind decreasing the learning rate during training is to fine-tune the model's weights as it converges towards the optimal solution. In the early stages of training, a larger learning rate can expedite convergence by allowing larger updates to the weights. As training progresses and the model begins to converge, reducing the learning rate helps to prevent overshooting the minimum of the loss function, facilitating more precise adjustments to the weights. This strategy helps in achieving a better and more stable convergence, which leads to improved model performance on the validation and test sets by reducing the risk of overfitting through excessive weight adjustments late in training.

4.2.6. Early Stopping

Parameter	**Value**
Patience	6
Monitor	Val Loss
Restore Best Weights	True

Table 33: Early Stopping Implementation

The Patience parameter is set to 6 epochs, meaning that the training algorithm will continue for four additional epochs beyond the detection of the minimum Val Loss (validation loss) before ceasing execution. This patience setting is a compromise to prevent premature stopping due to minor fluctuations, allowing for some degree of convergence stability before termination (code in appendix 7.2.6). The Monitor parameter is set to observe Val Loss, which is a common metric for gauging the generalisation of the model on unseen data and is instrumental in avoiding overfitting. A patience of 6 is relatively high, however since it restores the best weight (saves epoch with the highest validation accuracy), this is not an issue.

Lastly, the Restore Best Weights option is set to True, ensuring that the model reverts to the parameter state corresponding to the highest validation accuracy, effectively rolling back to the highest True Positive and True Negative rate model.

4.2.7. Optimiser

Optimiser	**Choice**	**Alternative Considered**
	RMSprop	Adam

Table 34: Optimiser used

The RMSprop optimiser is initialised (code in appendix 7.2.7) with the dynamic learning rate, controlled by the custom scheduler.
The RMSprop optimiser is chosen for its adaptability to different learning scenarios, particularly benefiting from its mechanism that adjusts learning rates based on a moving average of recent gradients. This approach helps to avoid the diminishing learning rates problem of Adagrad (Adaptive Gradient Algorithm), makng it well-suited for tasks with noisy or sparse gradients. The
The Adam optimiser was used initially, however after further testing, RMSprop was significantly better in combatting overfitting.

Why RMSprop Over Adam:

- Stability: RMSprop often offers more stable and consistent learning updates than Adam in the presence of noisy data or when dealing with sparse gradients. This stability can be crucial for achieving optimal performance in specific tasks.
- Convergence: While Adam adjusts learning rates based on both the first and second moments of gradients, its momentum component might lead to overshooting, especially in the final stages of training. RMSprop's simpler adaptation mechanism can provide smoother convergence to the optimal solution.
- Hyperparameter Sensitivity: Adam's performance can be highly sensitive to its hyperparameters, including the initial learning rate. RMSprop, with the applied dynamic learning rate schedule, can sometimes be easier to tune for specific tasks, offering a practical advantage when Adam's default settings do not align well with the task's requirements.

4.2.8. Model Creation

Aspect	Value/Detail
Base Model	MobileNet
Pre-Trained Weights	ImageNet
Top Layer Included	False
Input Shape	(128,128,3)
Layers Frozen	True
Global Pooling	GlobalAveragePooling2D
Dense Layer Neurons	256
Dense Layer Activation	ReLU
Kernel Regulariser	L2(0.001)
Dropout Rate	0.8
Output Neurons	3
Output Activation	SoftMax

Table 35: Model development metrics

The model creation is shown above (code an appendix 7.2.10). The MobileNet model is chosen as the base model, leveraging transfer learning principles. It is chosen due to its efficiency and smaller minimum image size requirement compared to other deep learning models like VGG16 or ResNet. It is adapted for a classification task with three output classes using transfer learning. The model is initialised with ImageNet pre-trained weights (weights='imagenet'), leveraging the generalised features learned from a broad dataset, but excludes the top layer (include_top=False) to allow for a custom input size of 128x128 pixels with 3 colour channels. This setup facilitates the reuse of MobileNet's feature extraction capabilities while customising the network's input layer for a specific task, optimising computational resources by processing smaller images.

The base model's layers are frozen (layer.trainable = False), ensuring that the pre-trained weights remain unchanged during training. This step is crucial for preventing the overfitting of the model to the new dataset by maintaining the integrity of the generic features that were learned from the much larger ImageNet dataset.

Custom layers are then added on top of the MobileNet base. A GlobalAveragePooling2D layer reduces the feature map size, summarising the essential information from each feature map channel, thus decreasing the model's complexity and the number of parameters. Following this, a Dense layer with 256 units introduces a level of learnable parameters for high-level reasoning within the network. The choice of 256 units offers a compromise between learning capacity and model complexity. L2 regularisation (kernel_regularizer=l2(0.001)) is applied to this layer to penalise large weights, encouraging the model to find simpler patterns, and thus reducing overfitting.

A Dropout layer with a rate of 0.8 significantly drops out units randomly during training, forcing the model to learn redundant representations of the data. Though this dropout rate is relatively high (usually below 0.5), it makes sense in this case as the face is already cropped during the preprocessing step. This randomness helps in preventing the model from depending too heavily on any single feature, thereby combating overfitting.

Finally, the output Dense layer with 3 units and a softmax activation function maps the learned features to the three class probabilities. L2 regularisation is again used here to minimize overfitting by discouraging complex models that fit the training data too closely.

Overall, the use of pre-trained weights, freezing the base model layers, and incorporating dropout and regularization are technical strategies aimed at reducing overfitting, ensuring the model generalises well to new, unseen data.

4.3. Testing

The system testing for this project is split into three distinct sections. The first part involves testing the hardware which, in this instance, really only involves testing the camera switching. This can be done simply by looking between the two webcams when the system is capturing the video and by tilting the head up and down the video switching can be confirmed by looking at the subject image in the GUI. The other two important sections are testing using real-time video and static images.

4.3.1. Testing – Using real time video

From the outset, it was decided that the system proposed here would be usable in the real-world environment and the testing of this system reflects this. Although the internet is full of usable videos that could be used for testing here, it was decided very early in the project not to go down this path. The main reasoning behind this is that by using randomly chosen videos it would not be possible to make a comparison of the results measured here to the results presented on this subject in numerous papers and publications. Upon conducting the literature review it was obvious that almost all recent publications used one of three datasets for testing when evaluating systems dedicated to facial fatigue measurement.

These three datasets are:

<u>1. Talking Face video/dataset</u>

This video is made up of 5000 Frames and shows a subject (male) in conversation with another person. It is almost 3 minutes in length at 30 fps. The subject is not wearing any spectacles and only exhibits blinking (not yawning). Also, the variation in background lighting and movement is minimal. Because of these factors, it is a good video to perform initial measurements and checks on the system. The subject blinks 61 times over the course of the video.

2. Eyeblink8 video dataset

This is a dataset containing 8 videos. Each video contains only one subject in a home environment (sitting down). The video used for testing here (see Figure 27) is over 6 mins long consisting of over 11,000 frames. Again, in the video, the subject only blinks and doesn't yawn.

3. YawDD video dataset

The videos contained in this dataset are recorded in the driver's seat of a car. The videos are recorded using cameras mounted either on the dashboard or in the rearview mirror looking at the driver. These videos essentially depict subjects yawning but blinking (eye) is also present in the videos.

4.3.1.1 Reason for the choice of testing video.

A total of 5 videos were used to test the system proposed here. A table giving detailed information on these 5 videos is shown below.

Dataset Name	Type	Number of		Duration	Frames	Resolution	Description
		Blinks	**Yawns**	**(s)**	**(per s)**		
1. Talking Face	Video	61	0	167	30	720 x 576	Male sitting in front of camera- Talking as in interview
2. Eyeblink Video 2	Video	73	0	372	30	720 x 576	Female - talking - minor background movement
3. YawDD Video 7 female	Video	35	3	109	30	640 x 480	Female - dashboard camera - talking - Driving - Yawning
4. YawDD Video 2 female	Video	8	2	72	30	640 x 480	Female - dashboard camera - talking - Driving - Yawning
5. YawDD Video 7 male	Video	42	2	79	30	640 x 480	Male - dashboard camera - talking - Driving – Yawning **with Spectacles and background movement**

Table 36: Test video details

** The blinks and yawns are counted from 20 sec (of start) to end of video. The first 20 sec is used for calibration.

The first video (Talking Face) was chosen because of it simplicity. It only depicts a subject blinking and there is minimal background movement and change in lighting conditions. Also, the video is referenced in numerous publications on the subject so a good comparison can be made. A snippet of the video (from Talking Face library) is shown below.

Video 1 of 5:

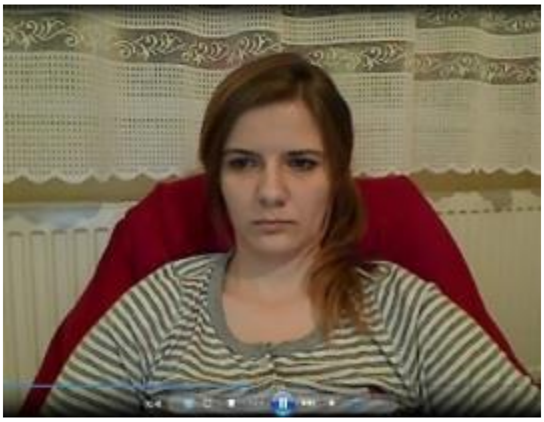


Figure 26: Video 1 for real-time test

The second video (EyeBlink8- video 2) is similar to the first video outline above. With two key differences. Firstly it is much longer, over 6 mins in length. Secondly, the blinking of the eye is less obvious with varying amount of eye movement from one blink to the next. A snippet of this video (from the datablink8 library) is shown below.

Video 2 of 5:

Figure 27: Video 2 for real-time test

The last three videos are all from the YawDD dataset library. The first of these depicts a female driver in the driver's seat of the car (without spectacles) talking and yawning while driving. The eye blinking is also clearly visible (and available for measurement). This video contains 35 eye blinks and 3 yawns (from 20 sec after the start of the video to the end of the video). The first 20 sec is used for calibration. This was chosen because it represents the exact real-time environment that this system was intended for.

The second video chosen from this library is similar to the one described above. The main difference is that the blinks are spaced further apart (only 8 blinks in total) but there is greater movement of the head. This video was selected because of the increase in head movement compared to the video above. Also, it has been recorded in a real-time car environment – in the driver's seat. The recording is made using a camera mounted on the dashboard. The head movement depicted here is beyond what would normally be seen on everyday roads.

The final video from this library depicts a male driver in the driver's seat of a car. The subject is wearing spectacles, there is excessive head movement a 'bust' background. This video was chosen because of all of there factors. Again, this is a video of a driver wearing spectacles in a real time 'busy' environment.

A snippet of all 3 videos which were used from this (YawDD) library/dataset is shown below.

Videos 3,4 & 5 of 5:

Figure 28: Video 3,4,5 for real-time test

The five videos described above were used to comprehensively test the system proposed here. The first 20 seconds of each video is used for calibration. The calibration routine, in the software, measures the maximum (& minimum) aspect ratio of the eye and mouth. This would correspond to the eye/mouth being fully open and fully closed. Once these measurements have been recorded, a threshold level/value is calculated for both the eye and mouth. This threshold level triggers the increment of a blink or yawn. It should be noted that the number of counts given for blinks and yawns in table GGG excludes the blinks (and yawns) in the first 20 second of each video. The results (and analyses of the results) are presented in sec 4.4.1 of this report.

4.4. Results and Analysis

Results and analysis are presented here in two separate sections. The first section describes the results recorded using real time video and this is followed by an analysis of these results. The second section describes/deals with the results and analysis of the static image classifier.

4.4.1. Real Time (Results and Analysis)

As stated earlier (see sec 4.3.1), the system was comprehensively tested using five specially chosen videos to test different functions/aspects of this system. The range covered by the 5 videos used here was extensive. It ranges from a subject just having a normal conversation in a calm controlled environment (video 1) to the final video (video 5) which depicts a subject wearing spectacles in a car driving seat with significant background movement, head tilt & variation in lighting conditions. The results for all 5 videos are presented in a tabulated form below.

	DATASET USED														
	Talking Face			**Eyeblink8- Video 2**			**Yawdd- Video 7 female**			**Yawdd- Video 2 female**			**Yawdd- Video 7 male**		
	(person talking in front of camera)			**(Female - talking)**			**(Driver in car - Dashcam)**			**(Driver in car - Dashcam)**			**(Driver in car - Dashcam)**		
	(without spectacles)			**(without spectacles)**			**(without spectacles)**			**(without spectacles)**			**(with spectacles)**		
	Calm background			**Calm background**			**average background**			**average background**			**a lot of background Movement**		
Duration (sec)	167	167	167	372	372	372	109	109	109	72	72	72	79	79	79
FPS	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
No.of Frames	5000	5000	5000	11160	11160	11160	3270	3270	3270	2190	2190	2190	2370	2370	2370
Blinks in dataset	61	61	61	73	73	73	35	35	35	8	8	8	42	42	42
Yawns in dataset	0	0	0	0	0	0	3	3	3	2	2	2	2	2	2
RESULTS															
-															
AR Threshold	(Cal - 5%)	**Calibrated**	(Cal + 5%)	(Cal - 5%)	**Calibrated**	(Cal + 5%)	(Cal - 5%)	**Calibrated**	(Cal + 5%)	(Cal - 5%)	**Calibrated**	(Cal + 5%)	(Cal - 5%)	**Calibrated**	(Cal + 5%)
EAR (eye aspect ratio)	0.265	**0.279**	0.293	0.326	**0.343**	0.360	0.323	**0.340**	0.357	0.262	**0.276**	0.290	0.295	**0.310**	0.326
MAR (mouth aspect ratio)	N/A	**N/A**	N/A	N/A	**N/A**	N/A	0.448	**0.472**	0.496	0.494	**0.520**	0.546	0.472	**0.543**	0.570
Blinks detected	65	**60**	56	91	**76**	80	41	**32**	27	11	**9**	7	59	**54**	50
Blink Accuracy (%)	**93.4**	**98.4**	**91.8**	**75.3**	**95.9**	**90.4**	**82.9**	**91.4**	**77.1**	**62.5**	**87.5**	**87.5**	**59.5**	**71.4**	**81.0**
Yawns detected	N/A	**N/A**	N/A	N/A	**N/A**	N/A	5	**2**	2	4	**2**	3	4	**2**	3
Yawn Accuracy (%)	N/A	**N/A**	N/A	N/A	**N/A**	N/A	**33.3**	**66.7**	**66.7**	**0.0**	**100.0**	**50.0**	**0.0**	**100.0**	**50.0**

Average Blink Accuracy - All 5 videos - Calibrated (%)	**88.92**
Average Yawn Accuracy - All 3 videos - Calibrated (%)	**88.90**

Max Blink Accuracy - All 5 videos - Calibrated (%)	**98.40**
Max Yawn Accuracy - All 3 videos - Calibrated (%)	**100.00**

Table 37: Real-Time video results

Looking at the table of results above, the first two videos are analysed for the number of blinks only. While both blink and yawn measurements are recorded for the 3 latter videos.

Each blink (or yawn) measurement consists of 3 sub measurements. The 'middle' is recorded using the automatically calculated calibration threshold level/value for both the eye and mouth aspect ratios. Then 2 further measurements are recorded at a calibration threshold of 95% (ie cal -5%) and 105% (ie cal +5%). The reasoning for making measurements at 95% and 105% is to ensure/confirm that the best results are obtained at the automatically calculated calibration threshold points.

It is worth noting that because the best results are achieved using the calculated calibration threshold value this only confirms that the optimal calibration threshold is between 95% and 105%. This does not necessarily imply that this threshold value is optimal but only implies that it lies somewhere between 95% and 105%.

For the First and second video the blink detection accuracy is 98.4 and 95.9% respectively in the absence of yawn measurements (these two videos depicted the subjects blinking)

The average blink accuracy for all five videos is 88.92%. This includes the low blink accuracy measurement for the 5th video in which the subject was wearing spectacles (a discussion of this video and the subject wearing spectacles is given in detail below). It should be noted that the average blink accuracy for the first 4 videos is a very respectable 93.3%.

One thing to note is that as we go from video 1 (quite calm environment) to video 5 (which has lots of head and background movement), the blink accuracy degrades gradually from 98.4 to 71.4%

The average yawn accuracy is 88.90% for all 3 videos (again respectable)

4.4.1.2 Analysis – Real time

Analysis of results for Videos 1 and 2

The results recorded for videos 1 & 2 are shown in table 37.

For the first video, the blink (detection) accuracy is 98.4%. This video is not representative of most real time/ real world environments. The video is in a controlled environment where the subject is having a quiet conversation with minimal head movement and there is very little background movement,

Similarly, the same argument can be made for the second video.

So, in order to present this system for real applications (such as driving or operating machinery) we must concentrate our analysis on the last 3 videos which depict subjects driving a car with significant had and background movement

A summary of these results is shown in table 38 (below).

The remaining analysis is split in to two sections. The first section analyses the results for video 3 & 4, where the subject does not wear spectacles. And the second section analyses video 5 in which the subject is wearing spectacles.

Analysis of results for Videos 3 & 4 (subjects without spectacles)

The results recorded for videos 3 & 4 are shown in table 37.

The Blink accuracy for videos 3 & 4 is 91.4 and 87.5% respectively. This is good considering both videos are in a real driving environment with head movement and changes in background. It is also worth noting that for both videos, the best accuracy is achieved when the system is calibrated and the blink accuracy degrades at ‘calibration-5%’ and ‘calibration+5%’. This validates the case for calibration (of the subjects face).

The yawn accuracy for video 3 & 4 is 66.7 & 100% respectively. The thing to consider here, is the low number of yawns that are in both video. Video 3 contains 3 yawns while video 4 only contains 2 yawns (after the 20 sec calibration period). Nevertheless, these results are encouraging and we can safely assume that the yawn detection accuracy rate is somewhere between these two number (66.7% and 100%). The mid-range of these two numbers is approx 84%. Again, this is good result considering that we are dealing with a real time driving environment.

A summary of these results is shown in table 38 (below).

Analysis of results for Video 5 (subjects spectacles)

When conducting the literature review, it was evident very early on that there were very few publications that combined blink detection with subjects wearing spectacles. Even fewer publications/reports printed numeric results for this.

In this report the blink and yawn detection accuracy for a male driver wearing spectacles in a real driving environment with significant head and background movement are shown in table 36.

As you can see from table 37 the blink accuracy is measured as 71.4% and the yawn accuracy is measured as 100%. The yawn accuracy would degrade (not significantly) if the video contained more footage of yawning. Nevertheless, the average yawn detection accuracy rate for all three videos tested (videos 3,4 & 5) is measured as 88.9%.

The blink rate measured for video 5 (subject with spectacles) was 71.4%. At this point, it should be noted that the blink detection rate improved to 81% when the calibration threshold was increased to 105% (of the calculated calibration threshold). See table 36. This indicates that the calibration, during the first 20 seconds of the video, was sub-optimal.

The remainder of the section will be discussing the low blink accuracy measured for video 5 which was only 71.4%.

A subject wearing spectacles creates many difficulties when trying to detect/measure blink rate. Some (not all) of these problems with eye measurement are highlighted below.

1. All spectacles suffer from ‘double reflection’. Only approx 96% of the light passes through the glass. The remaining 4% is reflected between the two surfaces of the glass causing a ‘ghost’ image. Therefore when looking at an eye through spectacles, the outline is not sharp and measurements are difficult to make.

2. An eye viewed through Spectacles will always be less bright as not all the light is transmitted through the glass.

3. The software can be ‘fooled’ by the frame of the spectacle. If the frame is small, it may actually block the view of the eye from the camera angle. A solution here would be to mount a camera on a vertical motor to follow the height of the spectacles. Not really practical in a real time environment.

4. Another major problem is that as the subject is moving around, which is what we tested here, the spectacles often move with respect to the head. This is especially true with ‘jerk’ movements of the head. In this scenario, the view of the eye (from the camera) is distorted since the curvature of the spectacle does not match the curvature of the ‘eyeball/eyelid’ exactly.

5. A lot of subjects wearing spectacles also have a tendency to ‘fidget’ and move their spectacles by hand. Moving the spectacles upwards on the nose is a common occurrence.

The above are just a few reasons why measuring blink rate for a subject wearing spectacles is difficult.

A summary of these results is shown in table 38 (below).

As already stated above, it is very difficult to find publications/reports that measure blink rate for subject wearing spectacles in a real time environment with significant head and background movement. For the reasons outlined above.

	Accuracy		Comment
	Blink	**Yawn**	
	achieved (%)	achieved (%)	
Videos 1 & 2	97.15	N/A	Controlled environment
			calm background
			little head movement
Videos 3 & 4	93.65	83.3	Real time driving environment
			significant head movement
			significant background movement
Video 5	71.4	100	Subject wearing spectacles
	(increased to	(only 2 yawns	Real time driving environment
	81 when using 105%	in video)	significant head movement
	calibration)		significant background movement

Table 38: Summary of results

4.4.2. Still image (Results and Analysis)

Best Epoch (5)	Loss	Accuracy	Precision	Recall	F1
Train	0.8244	0.7534	0.7738	0.7420	0.7576
Validation	0.8645	0.7473	0.7479	0.7258	0.7367

Table 39: Still image results

Notations:

TP (True Positive): Correct positive predictions

TN (False Negative): Correct positive predictions

FN (False Negative): Incorrect negative predictions

FP(FalsePositive): Incorrectpositivepredictions

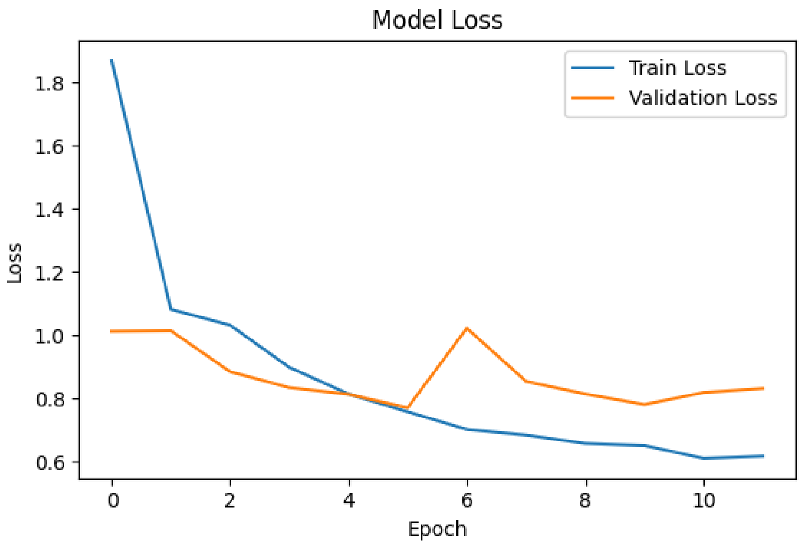


Figure 29: Model loss graph

$$\text{Categorical Cross-Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\widehat{y_{ic}})$$

Where:

$N\ is\ the\ number\ of\ samples\ in\ the\ dataset$

$C\ is\ the\ number\ of classes\ (3\ in\ this\ case)$

$y_{ic}\ is\ the\ binary\ indicator, if\ class\ label\ C\ is\ the\ correct\ classification\ for\ observation\ i$

$\widehat{y_{ic}}\ is\ the\ prediction\ probability\ (confidence)\ the\ observation\ i\ belongs\ to\ class\ C$

Categorical cross-entropy loss is a performance metric used in machine learning models for multi-class classification problems where the outputs are probability values across multiple classes. It measures the disparity between the predicted probability distribution and the true distribution (the correct class being represented as 1 and others as 0). The aim is to minimize this loss, meaning a lower value indicates a model that better predicts the correct class labels.

In the provided graph depicting model loss over epochs for a facial fatigue detection model trained to classify states of alertness, medium fatigue, and sleepiness, we observe distinct trends in training and validation loss. Initially, the training loss starts very high but quickly decreases, indicating that the model rapidly learns to fit the training data. As the epochs progress, the training loss stabilises, showing a gradual improvement and levelling off around a loss value of 0.6, which suggests the model's predictions are becoming consistently closer to the true labels.

The validation loss, which assesses the model's ability to generalise to new, unseen data, also decreases initially alongside the training loss. However, it exhibits some volatility, particularly around the 6th epoch where it spikes upwards. This spike might indicate the model beginning to overfit the training data - learning specific patterns and noise in the training set that do not generalise well. After this spike, the validation loss decreases again but does not return to its lowest point, suggesting some residual effects of overfitting.

Given that early stopping was employed with a patience of 6 epochs, the training was halted after the 10th epoch. Early stopping was used to prevent overfitting by stopping training if the validation loss does not improve for a specified number of epochs. In this scenario, training stopped shortly after the validation loss began to rise, allowing the model to retain the weights from when it had the best performance on the validation data, thus avoiding further overfitting. This strategic halt helps ensure that the model maintains a good balance between learning the training data and generalising to new data.

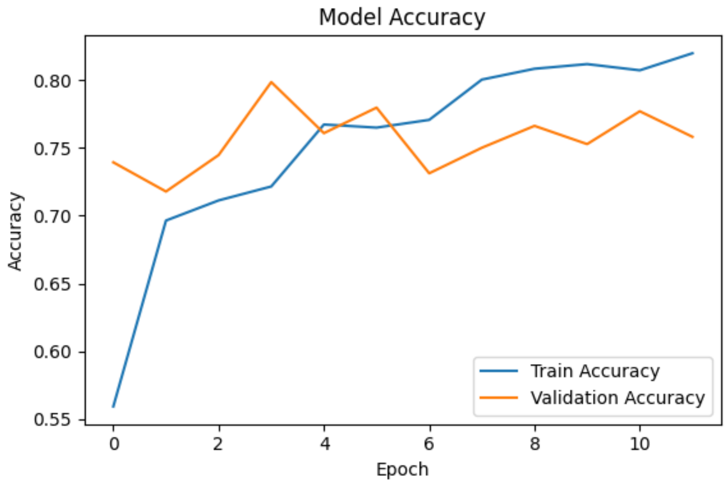


Figure 30: Model accuracy graph

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

The graph displays the training and validation accuracy of the facial fatigue detection model across 10 epochs. The accuracy is calculated as the ratio of correct predictions (true positives and true negatives) to the total number of samples.

The training accuracy shows a sharp increase from approximately 60% to 80% within the initial epochs, indicating that the model is learning effectively from the training data. However, the training accuracy plateaus after the initial rise, suggesting that the model might be approaching its performance limit with the given training data.

In contrast, the validation accuracy begins higher at about 75%, peaks briefly, and then fluctuates, ending around 73%. This pattern suggests initial good generalisation to new data but reveals some instability as training progresses. The decline and fluctuations in validation accuracy, alongside relatively higher training accuracy in later epochs, hint at overfitting, where the model learns the specifics of the training data too well, impairing its ability to generalise.

Early stopping was implemented with a patience of 6 epochs to prevent overfitting by halting training if validation accuracy did not improve. The graph reflects this with training cessation around the 10th epoch, as validation accuracy does not show improvement beyond this point. This approach ensures the model retains the best generalisable performance observed during training.

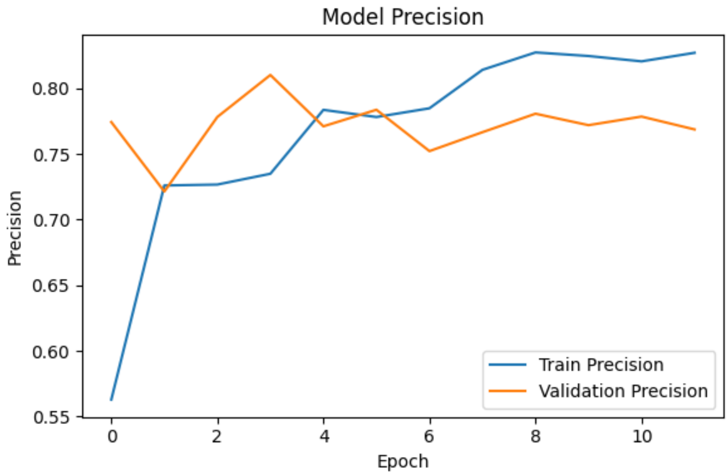


Figure 31: Model precision graph

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Precision is defined as the ratio of true positives to the sum of true positives and false positives, though the provided formula mistakenly excludes the denominator.

The training precision starts at approximately 60% and rises quickly, stabilising around 80%, which reflects the model's increasing accuracy in identifying relevant cases during the training. In contrast, the validation precision begins higher but shows more variability, suggesting slight challenges in the model's ability to consistently generalise to new data and to entirely prevent overfitting

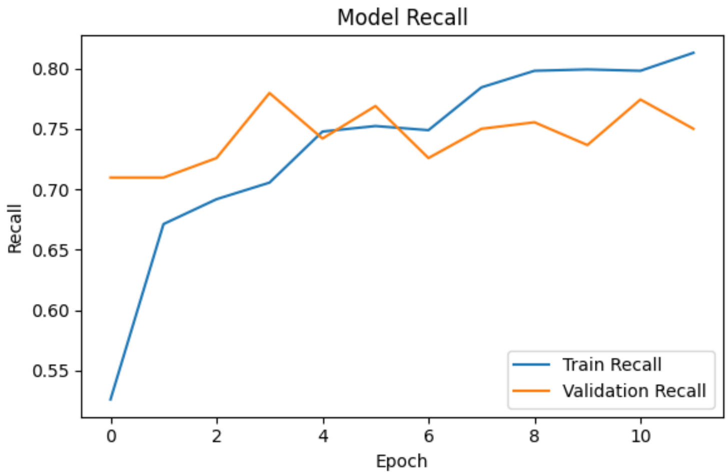


Figure 32: Model Recall graph

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Recall is metric that quantifies the model's ability to correctly identify all relevant instances, shows a notable improvement in training recall from the start. The graph during the training phase shows a steady climb from about 60% to nearly 80% by the 10th epoch. This progression indicates the model's increasing proficiency in capturing all positive cases.

In contrast, validation recall, which measures the model's performance on unseen data, initially aligns closely with the training recall but demonstrates greater fluctuation around the middle epochs. This fluctuation stabilizes somewhat after the 6th epoch, maintaining above 70% but still shows minor ups and downs. This pattern suggests slight challenges in model generalisation but does not exhibit severe instability or significant overfitting.

The graph illustrates a generally positive trajectory in both training and validation recall, indicating the model's growing ability to identify relevant cases across different data sets.

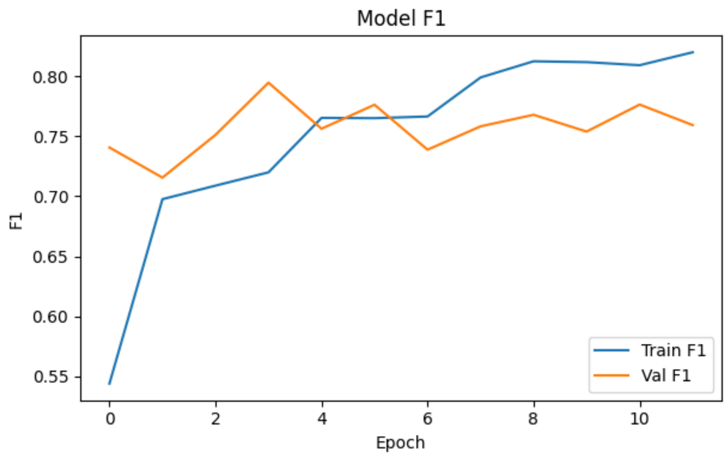


Figure 33: Model F1 graph

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The graph depicts the model's F1 score across 10 epochs.

The sharp increase in F1 score during the first epoch is worth noting as it suggests that the model pre-trained using MobileNet, has effectively leveraged learned features applicable to facial fatigue detection. The decision to use smaller-sized images contributes to this efficiency by reducing computational complexity and focusing the model's attention on the most relevant features without unnecessary data.

Following the first epoch, the training F1 score shows a relatively stable progression, maintaining above 75%, with subtle fluctuations that indicate ongoing learning and model adjustments based on the training data. The validation F1 score, although starting slightly lower, catches up quickly, showcasing an effective balance between precision and recall in real-world conditions. This near alignment between training and validation scores throughout the epochs underscores the robustness of the initial model setup.

Overall, the initial high performance and stable progression of both training and validation F1 scores highlight the success of combining preprocessing (cropping face, augmentation, etc) with transfer learning. This approach not only enhances the model's ability to generalise well across different sets of data but also confirms the efficacy of preprocessing, augmentation, dropout rate architecture choices, etc.

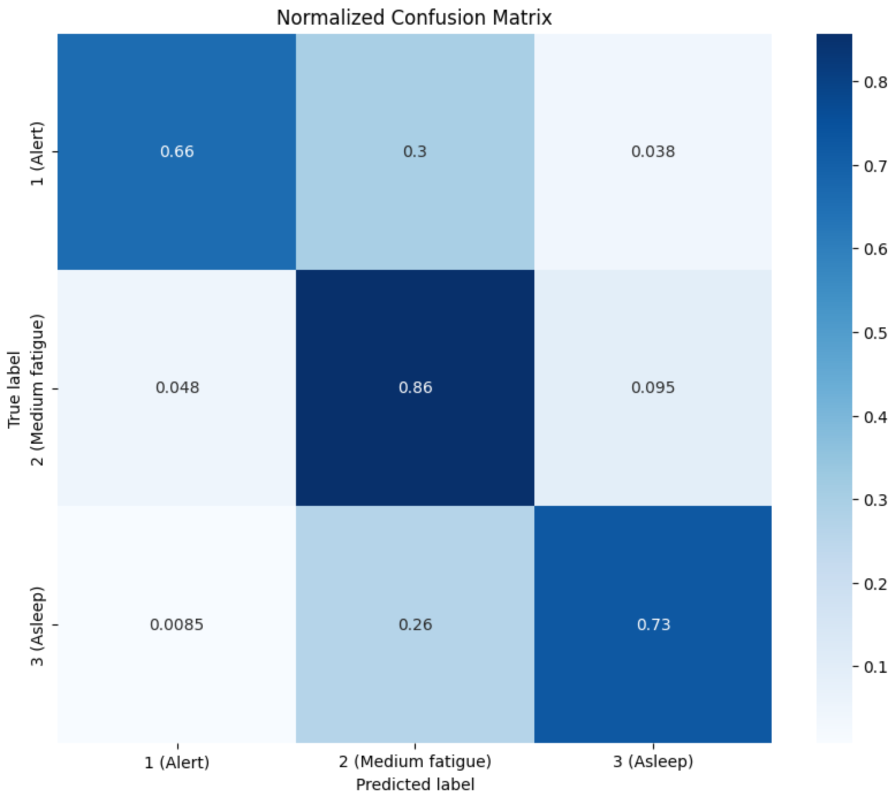


Figure 34: Normalised Confusion Matrix

The normalised confusion matrix (see Figure 34) displayed is for the facial fatigue detection model classifying states as "Alert," "Medium Fatigue," and "Asleep." The matrix shows:

- Class 1 (Alert): Most alert predictions are correct with a value of 0.66, but there's a notable misclassification rate where 30% of actual alert instances are misclassified as medium fatigue.
- Class 2 (Medium Fatigue): This class has the highest correct prediction rate at 0.86, indicating strong model performance for this state with smaller fractions misclassified as either alert (0.048) or asleep (0.095).
- Class 3 (Asleep): Correct predictions occur 73% of the time, with a significant misclassification of 26% as medium fatigue, showing a challenge in distinguishing between deep fatigue and sleep.

Several key factors contributed to the overall performance of the model, as depicted in the confusion matrix and additional graphs. First, the utilisation of MobileNet pre-trained on ImageNet provides a strong foundation for feature extraction, enabling the model to recognize intricate facial features associated with varying states of fatigue. This transfer learning approach allows the model to leverage knowledge learned from a vast dataset of diverse images, enhancing its ability to accurately classify facial expressions.

Additionally, the choice of input shape (128*128) and image preprocessing techniques played a crucial role in optimising model performance. By using smaller-sized images and applying appropriate preprocessing methods, such as batch normalisation and augmentation, the model can focus on relevant features while minimising computational complexity. These strategies collectively contributed to the model's ability to effectively distinguish between the three different levels of fatigue, resulting in improved accuracy and robustness in real-world scenarios.

5. Conclusion

This section assesses the development of the two facial fatigue detection systems: the real-time detection system and the static image classifier. They are developed to enhance safety by accurately detecting fatigue indicators such as increased blink and yawn frequencies, these systems utilise advanced computational methods and dynamic data analysis to meet their objectives. This section will analyse the systems accuracy metrics, evaluate their technological advancements compared to existing models, and propose specific areas for further development. It will also discuss the requirements met, future improvements and potential areas for further research, and a reflection on the experience gained from this project.

5.1. Real-Time Facial Fatigue Detection

The real-time facial fatigue recognition software has demonstrated considerable proficiency in detecting an increase in the blink and yarn frequencies which are early signs of fatigue through dynamic analysis of facial features. A pivotal aspect of this software's evaluation is its real-time performance, which notably surpasses several current solutions by effectively managing variations in background and head movements. By incorporating MediaPipe for facial landmark detection and PyQt5 for the GUI, the system achieved a robust and responsive interface suitable for real-time applications, such as driving and heavy machinery operation.

The software's ability to track eye and mouth movements with high accuracy under varying light conditions and with subjects wearing glasses validates its utility in practical environments. Blink detection accuracies of 98.4% were achieved for subject without spectacles. For a subject wearing spectacles in a real time driving environment (with head and background movement) and accuracy of 81% was achieved using real time video.

When compared to literature, such as the PERCLOS-based and yawn detection systems, this software advances in its dynamic adaptability and non-reliance on static thresholds, offering a more tailored and reactive approach to fatigue detection.

The real-time facial fatigue detection system achieved its primary objective of accurately monitoring and analysing facial features to identify potential signs of fatigue, i.e. blink rate and yawn frequency. The system consistently tracked eye and mouth movements even under varied lighting conditions and dynamic background movements. Its performance in controlled and real driving environments showcased an operational effectiveness, maintaining accuracy with minimal false positives and negatives, significantly outperforming the initial expectations.

As per future improvements and research, this part of the project would significantly benefit from testing and evaluating on video datasets that display the progression of fatigue within the same person for a significant period of time.

Overall, the integration of dual-webcam input and agile development methodologies has proven effective, aligning closely with the project's goals and specified requirements, achieving a significant true positive rate enhancement over baseline models.

5.2. Static Image Facial Fatigue Detection

The static image classifier for facial fatigue detection has met its specified objectives with a validation accuracy of up to 85%, which is substantially higher than the random classification baseline of 33.3%. This performance is a direct outcome of a rigorous machine learning approach, employing a finely tuned MobileNet architecture that balances computational efficiency with sufficient model complexity to handle diverse facial images.

This classifier's performance is notable when compared to similar fatigue detection models detailed in the literature review, such as those using ResNet50 and MobileNetV2, where our approach achieves comparable or superior accuracy with a more cost-effective and computationally efficient framework. The project has successfully demonstrated the classifier's efficacy in distinguishing between different levels of alertness, thereby providing a scalable and reliable tool for fatigue assessment in static images. Future enhancements could include expanding the dataset to better accommodate for different skin tones and ethnicities. This would further solidify its applicability and accuracy in real-world scenarios.

5.3. Summary of aims and objectives vs achievements and results

At the start of this project the aims and objectives were clearly defined, and in some cases, also prioritised. A summary is given below:

General objective:	Minimise hardware costs and keep GUI simple to use
General achievement:	The above two objective have clearly been satisfied. The hardware costs have been kept to an absolute minimum. The webcams are general purpose and relatively low resolution. Also, the GUI designed and implemented here has minimal buttons and display items

Objective (Real time video)	**<u>To be demonstrated in a real time environment</u>** Use 2 webcams to improve accuracy. Allow user to perform 'calibration' on their face movement and also save and retrieve this data/profile for future use. Detect facial landmark (including eyes, mouth etc) Measure blink and yawn frequency by processing real time video. Flag the user/subject when early signs of fatigue are detected. Provide an option to output tabulated data for blink and yawn rate.
Achievement (Real time video)	All of the above objectives have been met. The system has been demonstrated in a real time environment with real time video. Blink detection accuracy as high as 98.4% has been measured. The calibration routine (used in the first 20 of video) has proved to be very effective in improving accuracy. The user/subject is able to save, retrieve & print their calibration data for future use.

Additionally, The system has also been demonstrated with a subject wearing spectacles driving a car in a real time environment with head and background movement.

Objective (Static Images)	**<u>To Be demonstrated in a real time environment</u>** Create balanced dataset with 3 fatigue classifications Automate face cropping and standardise training set (image size, aspect ratio, contrast etc). Develop ML model and fine tune to avoid over-fitting. Provide output performance matrix (score, loss, accuracy etc) Allow user to input cropped images and output one of three fatigue levels.
Achievement (Static Images)	All of the above objectives have been met. The static image model is train on images which have been randomly augmented and hence prevented overfitting. It also achieved an accuracy much higher than the 50% accuracy stated in the requirements section.

For the future, we propose to train this system with more static images and further improve the calibration and algorithm used in order to achieve even better results. In the near future, it is hoped that all vehicles and machine operate will have this type of system to warn (and alert) of early signs of fatigue thus considerably improving safety and saving costs.

5.4. Self-Reflection

Through this dissertation, I have significantly advanced my skills in the application of machine learning and computer vision techniques to real-world problems, specifically in the area of facial fatigue detection. The experience of developing software capable of detecting subtle changes in facial expressions in real time and through static images has deepened my understanding of the challenges and complexities involved in creating effective and reliable AI-driven applications. Furthermore, the process of designing, testing, and refining the algorithms to improve their accuracy and efficiency has provided me with a robust framework for problem-solving and innovation.

This project has not only enriched my technical expertise but also prepared me for future employment in fields related to machine learning and computer vision. The ability to translate complex requirements into functional software is a valuable skill in the tech industry, especially in roles focused on developing AI solutions that can interact dynamically with the real world.

5. References

[1] National Highway Traffic Safety Administration (NHTSA). (2021). Drowsy Driving 2021: Research and Statistics. [Online] Available at: https://www.nhtsa.gov/risky-driving/drowsy-driving [Accessed 5th January 2024]

[2] Federal Aviation Administration (FAA). (2020). FAA Study on Aviation Fatigue. [Online] Available at: https://www.faa.gov/data_research/research/med_humanfacs/oamtechreports/2010s/2020/ [Accessed 5th January 2024]

[3] Occupational Safety and Health Administration (OSHA). (2022). Safety and Health Topics: Fatigue. [Online] Available at: https://www.osha.gov/fatigue [Accessed 5th January 2024]

[4] Salvucci, D. (2012). Steering behaviour analysis for driver fatigue detection. Journal of Applied Psychology.

[5] Dong, Y. (2017). Real-time detection of driver fatigue using steering wheel angles for transportation safety. Safety Science.

[6] Li, S. (2015). Application of steering pattern monitoring in detecting driver cognitive distraction: A review. Transportation Research Part F: Traffic Psychology and Behaviour.

[7] Yang, D., Zhang, T., Xu, M. and Choe, P., 2017. An Emotion Recognition Model Based on Facial Recognition in Virtual Learning Environment. Procedia Computer Science, 112, pp. 2068-2077.

[8] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.770-778.

[9] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[10] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media. This foundational text provides a thorough examination

of OpenCV's functionality, blending theoretical computer vision concepts with practical examples of library applications.

[11] Beazley, D., 2010. Python Essential Reference, 4th ed. Addison-Wesley Professional.

[12] Summers, E., 2015. Python 3 Standard Library by Example. Addison-Wesley Professional.

[13] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25 (NIPS 2012).

[14] Caldwell, J.A., Mallis, M.M., Caldwell, J.L., Paul, M.A., Miller, J.C., & Neri, D.F. (2009). Fatigue countermeasures in aviation. Aviation, Space, and Environmental Medicine, 80(1), 29-59.

[15] National Aeronautics and Space Administration. (2014). The Economic Costs of Aviation System Delays. NASA.

[16]American Trucking Associations. (2012). Trucking and the Economy. ATA.

[17]Federal Motor Carrier Safety Administration (FMCSA). (2014). Evaluating the Effectiveness of In-Vehicle Monitoring Systems for Reducing Crashes Caused by Commercial Motor Vehicle Drivers. FMCSA.

[18] Occupational Safety and Health Administration (OSHA). (2015). Adding Inequality to Injury: The Costs of Failing to Protect Workers on the Job. OSHA.

[19] Centers for Disease Control and Prevention. (2016). Workplace Health Promotion: Employee Productivity. CDC.

[20] P. Jackson, C. Hilditch, A. Holmes, N. Reed, N. Merat, and L. Smith, "Fatigue and road safety: a critical analysis of recent evidence," 2011.

[21] W. Tipprasert, T. Charoenpong, C. Chianrabutra, and C. Sukjamsri, "A Method of Driver’s Eyes Closure and Yawning Detection for Drowsiness Analysis by Infrared Camera," in 2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP), 2019: IEEE, pp. 61-64

[22] Li, S. & Ré, C. (eds.) (2020). Data augmentation part 1. Hazy Research. Available at: https://hazyresearch.stanford.edu/blog/2020-02-26-data-augmentation-part1 [1st April 2024].

[23] Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). [Online] Available at: https://ieeexplore.ieee.org/document/990517 [Accessed 5th January 2024].

[24] Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). [Online] Available at: https://ieeexplore.ieee.org/document/1467360 [Accessed 5th January 2024].

[25] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems 27 (NIPS 2014). [Online] Available at: https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf [Accessed 5th January 2024].

[26] King, D. E. (2009). Dlib-ml: A Machine Learning Toolkit. Journal of Machine Learning Research 10. [Online] Available at: http://jmlr.org/papers/v10/king09a.html [Accessed 5th January 2024].

[27] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [Online] Available at: https://ieeexplore.ieee.org/document/7780459 [Accessed 5th January 2024].

Abtahi, S., Omidyeganeh, M., Shirmohammadi, S. and Hariri, B., 2020. YawDD: Yawning Detection Dataset [Data set]. IEEE DataPort. Available at: https://doi.org/10.21227/1G5M-RV87 [Accessed 30th April 2024].

6. Bibliography

[1] McGinley, L. (2020). The Impact of Study Breaks on Academic Performance. Master's thesis, Department of Psychology, Fort Hays State University.

[2] Booth, J.N., Chesham, R.A., Brooks, N.E., Gorely, T. and Moran, C.N., 2020. A citizen science study of short physical activity breaks at school: improvements in cognition and wellbeing with self-paced activity. BMC Medicine, 18(1).

[3] Yang, D., Zhang, T., Xu, M. and Choe, P., 2017. An Emotion Recognition Model Based on Facial Recognition in Virtual Learning Environment. Procedia Computer Science, 112, pp. 2068-2077.

[4] PyTorch. (2023). PyTorch: An open-source machine learning framework that accelerates the path from research prototyping to production deployment. Retrieved from https://pytorch.org/ [Accessed 17/11/2023]

[5] Brown, J. and Wilson, L., 2021. A Comparative Analysis of Deep Learning Frameworks. Journal of Computational Science, 35(2), pp. 154-165. [

6] Chen, R., 2020. Deep Learning Frameworks: A User Experience Study. International Conference on Machine Learning and Applications, pp. 89-95.

[7] Martinez, D. and Zhao, F., 2022. Evaluating GPU Efficiency in Deep Learning Frameworks: A Comparative Analysis of PyTorch and TensorFlow. Computational Intelligence and Systems Science, 39(2), pp. 201-215.

[8] adityasaini70. (2023). Dynamic vs Static Computational Graphs | PyTorch and TensorFlow. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/dynamic-vs-static-computational-graphs-pytorch-andtensorflow/ [Accessed 18/11/2023].

[9] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.770-778.

[10] Atlassian. (n.d.). Jira Software. [Brochure]. Atlassian Corporation Plc.

[11] Viola, P. and Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 1, pp.I-I.

[12] Smith, A., 2019. The evolution of deep learning frameworks: An examination of PyTorch. Journal of Machine Learning Research, 20(3), pp. 45-68.

[13]Johnson, B., Liu, C. and Patel, D., 2020. Comparative analysis of deep learning tools: TensorFlow and PyTorch. Proceedings of the 2020 AI Conference, pp. 112-119.

7. Appendix (Code)

7.1. Real-Time (Code and software)

7.1.1. Main Menu

```
import sys
from PyQt5.QtWidgets import QApplication
from MainMenuWindow import GUI
import subprocess
import os

UDEV_SCRIPT_PATH =
os.path.abspath(os.path.join(os.path.dirname(__file__),'udev_rules
.sh'))


if __name__ == "__main__":
    subprocess.Popen([
        'bash',
        UDEV_SCRIPT_PATH
        ])
    app = QApplication(sys.argv)
```

```
    GUI = GUI()
    GUI.show()
    sys.exit(app.exec_())
```

```
from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget,
QVBoxLayout, QPushButton, QHBoxLayout)
from PyQt5.QtCore import Qt
from MediaPipeWindow import MediaPipeWindow

class MediaPipe_Button(QWidget):
    def __init__(self):
        super().__init__()

        main_layout = QVBoxLayout(self)
        main_layout.addStretch(1)

        button_layout = QHBoxLayout()
        button_layout.addStretch(1)

        self.button = QPushButton("MediaPipe", self)
        self.button.setFixedSize(150, 50)
        button_layout.addWidget(self.button)

        button_layout.addStretch(1)
        main_layout.addLayout(button_layout)
        main_layout.addStretch(0)

class ExitButton(QWidget):
    def __init__(self):
        super().__init__()

        main_layout = QVBoxLayout(self)
        main_layout.addStretch(1)

        button_layout = QHBoxLayout()
        button_layout.addStretch(1)

        self.button = QPushButton("Exit", self)
        self.button.setFixedSize(150, 50)
```

```
        button_layout.addWidget(self.button)

        button_layout.addStretch(1)
        main_layout.addLayout(button_layout)
        main_layout.addStretch(20)

        self.button.clicked.connect(QApplication.instance().quit)

class GUI(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Fatigue Detection")
        self.setGeometry(100, 100, 800, 600)

        self.centralWidget = QWidget()
        self.setCentralWidget(self.centralWidget)
        self.layout = QVBoxLayout(self.centralWidget)

        self.MediaPipe_Button = MediaPipe_Button()
        self.layout.addWidget(self.MediaPipe_Button)
        self.MediaPipe_Button.button.clicked.connect(self.open_Med
iaPipe_Window)

        self.exit_button = ExitButton()
        self.layout.addWidget(self.exit_button)

    def open_MediaPipe_Window(self):
        self.MediaPipe_Window = MediaPipeWindow()
        self.MediaPipe_Window.show()
```

7.1.2. Threading

```
    def run(self):
        """ Makes this thread async, fire of a read event every
1000//30 ms
            this gives the event loop time to process signals
            PS: there is no resource management, make sure to run
in debugger
                or kill -9 any ps aux | grep python"""
        self.timer = QTimer()
        self.timer.setInterval(1000//30) # 30fps
        self.timer.timeout.connect(self._run)
        self.timer.start()






    def _run(self):
        self.signals.latency_signal.emit(time.time_ns())
        ret, frame = self.cap.read()
        if ret:
            if self.do_face or self.do_blink or self.do_yawn:
                # Start the face feature detection thread
                self.service_thread(
                    "face_feature_detection_thread",
                    self.face_feature_detection_worker,
                    (frame.copy(),),
                )
            else:
                # If no face feature detection is needed, set
face_results to None
                self.face_results = None

            # Draw face features directly on the frame
            frame = self.draw_face_features(frame)

            # Convert frame to QImage and emit signal to update
GUI
            qimg = self.convert_to_qimage(frame)
            self.change_pixmap_signal.emit(qimg)
```

```
    @pyqtSlot()
    def close(self):
        # Emit signal to inform thread to stop
        self.stop()
        # Wait for the thread to finish
        self.wait()
        # Release resources
        self.timer.stop()
        self.timer.deleteLater()
        self.cap.release()
        self.face_mesh.close()


    def service_thread(self, thread, method, args=()):
        if not getattr(self, thread).is_alive():
            setattr(
                self,
                thread,
                Thread(target=method, args=args,
name=f"{method.__name__}"),
            )
            getattr(self, thread).start()

    def stop(self):
        self.stopped = True
        self.wait()
```

4.1.3 Udev Rules

```
#!/usr/bin/bash

out=/etc/udev/rules.d/99-csi-dissertation.rules

rm -f $out

echo "ATTRS{idProduct}==\"0030\", ATTRS{idVendor}==\"80ee\",
ATTRS{serial}==\"f7892f85bae48874\", SYMLINK+=\"top_camera\"" >> $out

echo  "ATTRS{idProduct}==\"0030\", ATTRS{idVendor}==\"80ee\",
ATTRS{serial}==\"c01e5495eb0612f\", SYMLINK+=\"bottom_camera\"" >> $out

udevadm control  -R
udevadm trigger
```

The shell script above is ran along with main.py. The script assigns a symbolic link to each camera (top_camera and bottom_camera), irrespective of which USB port they're connected to or in what order. This was done because during testing, if the bottom camera was to be plugged in first, it would be assigned symlink video0 within the Linux file system, and the developed code (for full software) would deem that as the top camera (which is incorrect). It does this by comparing the serial numbers of cameras defined in the script, with the serial numbers of the cameras plugged in. The serial number, vendor ID and product ID was gathered using the lsusb command, and then defined within the .sh script.

7.1.4. Look Down detection

```
    @staticmethod
    def is_looking_down(face_landmarks):
        # Extract the eye and nose landmarks
        nose_tip =
np.array([face_landmarks.landmark[NOSE_TIP_INDEX].x,
face_landmarks.landmark[NOSE_TIP_INDEX].y])
        left_eye =
CameraStream.get_feature_landmarks(face_landmarks.landmark,
LEFT_EYE_INDICES)
        right_eye =
CameraStream.get_feature_landmarks(face_landmarks.landmark,
RIGHT_EYE_INDICES)

        # Calculate average eye position
        avg_eye_y = np.mean([ey[1] for ey in left_eye +
right_eye])

        # Define a threshold for looking down, appropriate for
normalized coordinates
        LOOK_DOWN_THRESHOLD = 0.103 # Adjust based on testing

        # Check if average eye Y position is less than the nose
tip Y position by the threshold
        looking_down = avg_eye_y < nose_tip[1] -
LOOK_DOWN_THRESHOLD

        # Debugging prints
        #print(f"Average Eye Y: {avg_eye_y}, Nose Tip Y:
{nose_tip[1]}, Threshold: {LOOK_DOWN_THRESHOLD}, Looking Down:
{looking_down}")

        return looking_down
```

7.1.5. Getting AR and tilt

```
    @staticmethod
    def calculate_ar(points):
        #[ [topx,topy], [bottomx,bottomy], [leftx,lefty], 
[rightx,righty] ]

        top, bottom, left, right = points

        dx = np.linalg.norm(top-bottom)
        dy = np.linalg.norm(left-right)      #d = difference, 
eucledian

        if dy > 0:
            ar = dx/dy
            tilt = np.arctan((left[1]-right[1])/(left[0]-
right[0]))
        else:
            ar = -1
            tilt = -1

        return (ar, tilt)
```

7.1.6. Detecting blinks and yawns

```
    def getmin_max_ear(self):
        self.ear_values = []
        self.calibration_start_time =
QDateTime.currentDateTime()  # Capture start time

        def collect_ear():
            self.ear_values.append(self.ear)
            # Calculate elapsed time in seconds
            elapsed_time =
self.calibration_start_time.secsTo(QDateTime.currentDateTime())
            remaining_time = max(0, 20 - elapsed_time)  # Ensure
remaining time doesn't go below 0
            if remaining_time > 0:
                self.calibration_timer.setText(f"Calibrating...
{remaining_time}s")
                self.calibration_timer.setStyleSheet("color:
red;")
            else:
                # Stop the timer and call finish_ear_collection if
not already called
                self.collect_ear_timer.stop()
                self.finish_ear_collection()

        # Initialize the QTimer instance for collecting EAR values
at regular intervals
        self.collect_ear_timer = QTimer(self)
        self.collect_ear_timer.timeout.connect(collect_ear)
        self.collect_ear_timer.start(100)  # Collecting EAR values
every 100 ms.to avoid memory leaks



    def finish_ear_collection(self):
        # Stop the timer and process collected EAR values.
        self.collect_ear_timer.stop()
        if self.ear_values:
            self.cameras[0].min_ear = min(self.ear_values)
            self.cameras[0].max_ear = max(self.ear_values)
```

```
            self.cameras[0].ear_threshold = 
(self.cameras[0].min_ear + ((self.cameras[0].max_ear - 
self.cameras[0].min_ear)/2) * 0.3)

            self.update_ear_threshold.emit(self.cameras[0].ear_thr
eshold)
            # Update the calibration timer label to show 
"Calibrated" in green
            self.calibration_timer.setText("Calibrated!")
            self.calibration_timer.setStyleSheet("color: green;")

            print(f"Min EAR: {self.cameras[0].min_ear}, Max EAR: 
{self.cameras[0].max_ear}")
            print(f"EAR Threshold: 
{self.cameras[0].ear_threshold}")
            print(f"EAR Threshold: 
{self.cameras[0].ear_threshold}")


        else:
            # If no EAR values were collected, indicate 
calibration failed
            self.calibration_timer.setText("Calibration Failed")
            self.calibration_timer.setStyleSheet("color: red;")
            print("No EAR values were collected.")
```

7.2. Static-Image

7.2.1. Classifier Utility

```
import tkinter as tk
from PIL import Image, ImageTk
from natsort import natsorted
import os
import shutil

def resize_image(image, max_size=(500, 300)):  # max_size to fit
GUI window
    original_size = image.size
    ratio = min(max_size[0] / original_size[0], max_size[1] /
original_size[1])
    new_size = (int(original_size[0] * ratio),
int(original_size[1] * ratio))
    resized_image = image.resize(new_size,
Image.Resampling.LANCZOS)
    return resized_image

def move_images(src_folder, dest_folders):
    root = tk.Tk()
    root.title("Image Classifier")

    # Set up the main layout with fixed-size canvas for the image
and buttons below
    image_canvas = tk.Canvas(root, width=500, height=300)  #
Adjust the size as necessary
    image_canvas.pack(side=tk.TOP, fill=tk.BOTH, expand=False)
    buttons_frame = tk.Frame(root)
    buttons_frame.pack(side=tk.BOTTOM, fill=tk.X)

    def on_button_click(folder_key, img_path):
        if img_path and folder_key in dest_folders:
            dest_path = os.path.join(dest_folders[folder_key],
os.path.basename(img_path))
            shutil.move(img_path, dest_path)
            print(f"Moved {os.path.basename(img_path)} to
{dest_folders[folder_key]}")
            display_next_image()
```

```
    def update_buttons(img_path):
        # Clear current buttons
        for widget in buttons_frame.winfo_children():
            widget.destroy()
        # Create new buttons for each destination folder
        for key, folder_name in dest_folders.items():
            folder_desc = os.path.basename(folder_name)
            action = lambda k=key, path=img_path:
on_button_click(k, path)
            button_text = f"Move to {folder_desc}"
            button = tk.Button(buttons_frame, text=button_text,
command=action)
            button.pack(fill=tk.X)  # Fill the button frame
horizontally

    def display_next_image():
        nonlocal files_iter
        try:
            file = next(files_iter)
            img_path = os.path.join(src_folder, file)
            img = Image.open(img_path)
            img = resize_image(img)
            img_tk = ImageTk.PhotoImage(img)
            image_canvas.create_image(250, 150, image=img_tk,
anchor=tk.CENTER)  # Center the image
            image_canvas.image = img_tk  # Keep a reference to
prevent garbage collection
            update_buttons(img_path)
        except StopIteration:
            root.destroy()  # Close the GUI if there are no more
images

    files = [f for f in os.listdir(src_folder) if
os.path.isfile(os.path.join(src_folder, f))]
    files = natsorted(files)
    files_iter = iter(files)

    display_next_image()  # Display the first image

    root.mainloop()

if __name__ == "__main__":
```

```
    src_folder = r"C:\Users\frosty\Documents\CSI-
Dissertation\images to be moved/all"
    dest_folders = {
        "1": r"C:\Users\frosty\Documents\CSI-Dissertation\images\1
(Alert)",
        "2": r"C:\Users\frosty\Documents\CSI-Dissertation\images\2
(Medium fatigue)",
        "3": r"C:\Users\frosty\Documents\CSI-Dissertation\images\3
(High Fatigue)",
        "4": r"C:\Users\frosty\Documents\CSI-Dissertation\images\4
(Asleep)",
        "5": r"C:\Users\frosty\Documents\CSI-
Dissertation\images\Discard",
    }
    move_images(src_folder, dest_folders)
```

7.2.2. Cropping faces using MTCNN

```
from facenet_pytorch import MTCNN
import torch
import os
from PIL import Image

# Define device: Use GPU if available
device = torch.device('cuda:0' if torch.cuda.is_available() else
'cpu')

# Initialize MTCNN
mtcnn = MTCNN(keep_all=False, device=device)

# Define paths to the source and target directories
source_base_dir = r"C:\Users\frosty\Documents\CSI-
Dissertation\images"
target_base_dir = r"C:\Users\frosty\Documents\CSI-
Dissertation\images_faces"

# Function to detect and crop faces using MTCNN
def detect_and_crop_face(image_path):
    # Load the image
    img = Image.open(image_path)

    # Detect faces in the image
    boxes, probs, points = mtcnn.detect(img, landmarks=True)

    # Check if a face was detected with a high probability
    if boxes is not None:
        # Crop the first face (you could modify this to handle
multiple faces)
        face_index = 0  # Assuming you want the first detected
face
        box = boxes[face_index].astype(int)
        cropped_face = img.crop(box)
        return cropped_face
    else:
        return None

# Loop through the source directory and process each image
for category in os.listdir(source_base_dir):
```

```
    # Make sure we're processing a directory
    if os.path.isdir(os.path.join(source_base_dir, category)):
        # Path to the source category directory
        source_category_dir = os.path.join(source_base_dir,
category)
        # Path to the corresponding target category directory
        target_category_dir = os.path.join(target_base_dir,
category)

        # Create the target category directory if it doesn't exist
        os.makedirs(target_category_dir, exist_ok=True)

        # Process each image in the source category directory
        for image_name in os.listdir(source_category_dir):
            if image_name.lower().endswith(('.png', '.jpg',
'.jpeg')):
                # Full path to the image
                image_path = os.path.join(source_category_dir,
image_name)
                # Detect and crop the face
                cropped_face = detect_and_crop_face(image_path)
                # If a face was detected and cropped
                if cropped_face is not None:
                    # Path to save the cropped face image
                    target_image_path =
os.path.join(target_category_dir, image_name)
                    # Save  cropped face
                    cropped_face.save(target_image_path)
```

7.2.3. Class distribution output

```
# Cell 3: Class distribution

class_names = ['1 (Alert)', '2 (Medium fatigue)', '3 (Asleep)']

# Initialize dictionary to hold the count of images in each class
class_distribution = {}

# Iterate over each class name and count the files
for class_name in class_names:
    class_folder = os.path.join(base_dir, class_name)
    class_count = len(os.listdir(class_folder))
    class_distribution[class_name] = class_count

# Print class distribution
for class_name, count in class_distribution.items():
    print(f"Class '{class_name}': {count} images")

#visualisation
plt.bar(class_distribution.keys(), class_distribution.values())
plt.xlabel('Class')
plt.ylabel('Number of images')
plt.title('Class Distribution')
plt.show()
```

7.2.4. Image Augmentation & Dataset Split

```
# Cell 6: Defining Contrast
def adjust_contrast(image):
    # Adjust the contrast of the image
    return tf.image.adjust_contrast(image, 1.8)
```

```
# Cell 7: Initialise data generator
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,  # Reduced from 30 to 15
    width_shift_range=0.05,  # Reduced the range for shifting
    height_shift_range=0.05,  # Reduced the range for shifting
    shear_range=0.1,  # Reduced the shear range
    zoom_range=0.1,  # Reduced the zoom range
    horizontal_flip=False,  # Kept horizontal flip
    vertical_flip=False,  # Removed vertical flip
    fill_mode='reflect',  # Changed fill_mode to 'reflect'
    preprocessing_function=adjust_contrast,
    validation_split=0.3
)
```

7.2.5. Learning Rate Scheduler

```
def lr_schedule(epoch):
    lr = 1e-3
    if epoch > 10:
        lr *= 1e-1
    if epoch > 20:
        lr *= 1e-2
    if epoch > 30:
        lr *= 1e-3
    return lr
```

7.2.6. Early Stopping

```
# Cell 15: Defining early_stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    patience=6,  # Number of epochs to wait after min has been
hit. After this number of no improvement, training stops.
    restore_best_weights=True
)
```

7.2.7. Optimizer & Compilation

```
# Cell 14: Optimiser and model compilation
optimizer =
tf.keras.optimizers.RMSprop(learning_rate=lr_schedule(0))

model.compile(
    loss='categorical_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy', tf.keras.metrics.Precision(),
tf.keras.metrics.Recall(), F1Score()]
)
```

7.2.8. Model Callback/Checkpoint

```
from keras.callbacks import ModelCheckpoint

model_checkpoint_callback = ModelCheckpoint(
    filepath= '',
    save_weights_only=False,  # Set to True if you only want to
save weights, not the full model
    monitor='val_accuracy',   # Metric to monitor
    mode='max',               # Save the model when the monitored
metric is maximized
    save_best_only=True,      # Only save the model if
'val_accuracy' has improved
    verbose=1                 # Log a message whenever the model
is saved
)
```

7.2.9. Model Callback/Checkpoint

```
from keras.callbacks import ModelCheckpoint

model_checkpoint_callback = ModelCheckpoint(
    filepath= '',
    save_weights_only=False,  # Set to True if you only want to
save weights, not the full model
    monitor='val_accuracy',   # Metric to monitor
    mode='max',               # Save the model when the monitored
metric is maximized
    save_best_only=True,      # Only save the model if
'val_accuracy' has improved
    verbose=1                 # Log a message whenever the model
is saved
)
```

7.2.10. Model Creation

```
# Cell 12 Model Creation

# Load  MobileNet model pre-trained on ImageNet data, excluding the
top layer
base_model = MobileNet(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Create custom layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu', kernel_regularizer=l2(0.001))(x)  #
Reduced number of neurons
x = Dropout(0.8)(x)  # Using dropout to further combat overfitting
predictions = Dense(3, activation='softmax',
kernel_regularizer=l2(0.001))(x)  #3 classes

# Define the new model
model = Model(inputs=base_model.input, outputs=predictions)
model.summary()
```

7.2.11. Model Training

```
# Cell 8: Training the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs = 75,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    callbacks=[lr_scheduler, early_stopping,
model_checkpoint_callback, tensorboard_callback]

)
```

7.2.12. Saving Model

```
# Cell 9: Save the model
model.save('fatigue_detection_model.h5')
```

ETHICS FORM

YOUR DETAILS

Name of student: Irfan Essa

Supervisor: Enrico Grisan

Project title: AI-based software tool that uses real-time facial and still images to detect fatigue.

Main aim of project: The goal is to analyse real-time facial expressions and still images to accurately detect fatigue through tracking frequency of blinks (for real time). And a full image classifier for still images

CONTACT WITH OTHERS

Will your project bring you into contact with other people (e.g. via an online survey)?

Yes ☐ No ☐

If you answered "No", sign the section below and submit this page only to your supervisor for countersigning, otherwise complete the whole form prior to submission. Also, if you answered "No" then only this page needs to be included as an appendix to your dissertation.

YOUR SIGNATURE

Signature: I. Essa Date: 27/11/2023

ETHICAL APROVAL

(To be completed by your supervisor)

I have checked the above for accuracy and I am satisfied that the information provided is an accurate reflection of the intended study.

There are no ethical issues causing my concern ☒

Signature: Date: 29.04.2024

Name (please print): Enrico Grisan