UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

BACHELOR IN COMPUTING ENGINEERING

Specialization in Information Technology

BACHELOR DISSERTATION

AIWake
Platform for the analysis of the level of attention in lectures using computer vision and deep learning

Enrique Valverde Soriano

July, 2022

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Technologies and Information Systems

Specialization in Information Technology

BACHELOR DISSERTATION

AIWake
Platform for the analysis of the level of attention in lectures using computer vision and deep learning

Author: Enrique Valverde Soriano

Supervisor: Carlos Gonzalez Morcillo

Co-Supervisor: Santiago Sánchez Sobrino

July, 2022

Este texto ha sido preparado con la plantilla LaTeX de TFG para la UCLM publicada por Jesús Salido en GitHub[1] y Overleaf [2] como parte del curso *«LaTeX esencial para preparación de TFG, Tesis y otros documentos académicos»* impartido en la Escuela Superior de Informática de la Universidad de Castilla-La Mancha.

[1]https://github.com/JesusSalido/TFG_ESI_UCLM, DOI: 10.5281/zenodo.4574562
[2]https://www.overleaf.com/latex/templates/plantilla-de-tfg-escuela-superior-de-informatica-uclm/phjgscmfqtsw

Tribunal:

Presidente: ______________________________

Vocal: ______________________________

Secretario(a): ______________________________

Fecha de defensa: ______________________________

Calificación: ______________________________

Presidente	Vocal	Secretario(a)
Fdo.:	Fdo.:	Fdo.:

TRABAJO FIN DE GRADO - ESCUELA SUP. DE INFORMÁTICA (UCLM)

AIWake

Enrique Valverde Soriano
Ciudad Real, July 2022

Resumen

En los tiempos que corren no es raro encontrarse cada vez mas con sistemas que hacen uso de la visión por computador y la inteligencia artificial para resolver problemas del mundo real, algunos con aplicaciones tan importantes como la medicina o la seguridad y otros que hacen uso de estas tecnologias para resolver problemas relacionados con las emociones humanas ayudando en el tratamiento de algunos trastornos como el autismo o determinando las reacciones de los clientes ante un determinado producto.

Este proyecto toma como base la idea concebida originalmente en el C:TED, un departamento de la Universidad de Castilla-La Mancha encargado de la producción de contenidos digitales para esta misma, dentro de estos contenidos se encuentran las presentaciones dadas por profesores o expertos en una materia a las que, de manera presencial, los alumnos o interesados acudían. El proyecto AIWake toma ese contexto para que mediante el desarrollo de un sistema que se apoye en la visión por computador y técnicas de inteligencia artificial como el deep learning, se obtenga como resultado una herramienta que ayude a que dichos profesores o expertos a través de una grabación del público de sus ponencias puedan obtener métricas y estadísticas referentes a las expresiones mostradas por dicho público durante el transcurro de su ponencia para ayudarles a encontrar posibles mejoras en cuanto a su manera de transmitir información.

El resultado del desarrollo fue un sistema que se apoya en un modelo de clasificación entrenado especificamente para este cometido, una interfaz de usuario diseñada acorde a los principios de Gestalt y de usabilidad y un componente de control interno para la coordinación, todo esto junto dota de funcionalidad al sistema, sirviendo al usuario con gráficas de los datos obtenidos y herramientas para la corrección de posibles errores dados por la propia naturaleza de los problemas de clasificación.

BACHELOR DISSERTATION - ESCUELA SUP. DE INFORMÁTICA (UCLM)

Guided template for TFG

Enrique Valverde Soriano
Ciudad Real, July 2022

Abstract

It is not odd nowadays to increasingly find computer vision and artificial intelligence systems that solve real world problems, some of these systems have important applications on fields like medicine or security and others use this technologies in order to solve human emotions related problems helping with the treatment of disorders like autism or determining the reactions that clients have against a determined product.

This project comes from the originally conceived idea at the C:TED, a department of University of Castilla-La Mancha in charge of producing digital contents for this, regarding this contents there are the lectures brought by professors or matter experts which, in a face-to-face way, the students or interested ones came. AIWake project takes that context in order to through the development of a system supported by computer vision and artificial intelligence techniques such as deep learning to obtain a tool that supports these professors and matter experts which through a recording of their lecture attendants will be able to obtain statistics and metrics regarding the attendants facial expressions during the course of their lectures in order to help them finding improvements on their way of transmitting information.

The development result was a system that is supported on a specifically for this task trained classification model, a user interface designed accordingly to the Gestalt and usability principles and an internal control component for coordination, all of this together provides functionality to the system, serving the user with obtained data charts and tools to rectify possible misfires given by the implicit nature of classification problems.

Agradecimientos

Hay muchas personas que seguramente me dejaré en el tintero a la hora de agradecer el apoyo que me mostraron durante el desarrollo de este proyecto y seguramente necesitaría escribir un libro para mencionarlas pero si que quiero agradecer en especial a mi familia no solo por el apoyo durante el desarrollo del proyecto sino porque a pesar de ser un liante por naturaleza siempre están ahi, en especial a mi madre que si no hubiera sido tan pesada no me habría animado a retomar el proyecto con fuerza de nuevo.

A ese grupo de chavales que conocí en ciudad real y ahora no podemos pasar un año sin hacer mínimo una escapa.

Al Club Waterpolo Valdepeñas, del que orgullosamente formo parte, por hacer que mantuviera la cordura durante este periodo tan estresante de mi vida, no seremos el mejor equipo de la historia pero somos un grupo de amigos increible.

A mis colegas del pueblo que aunque pase el tiempo siempre harán que las cervezas sepan mejor los fines de semana.

A Carlos Gonzalez Morcillo por confiar en mí para este proyecto, darme tantísima libertad creativa a la hora de hacerlo y darme aquella oportunidad en el C:TED, un lugar que a pesar de todo siempre recordaré junto con su gente y los momentos que pasamos grabando aquellas aburridas ponencias de Química.

Enrique Valverde Soriano
Ciudad Real, 2022

Contents

List of Figures

List of Tables

Listings

CHAPTER 1

Introduction

How can we measure if a presentation was interesting, funny, both of them or nothing special at all? for sure someone without the technological knowledge would answer that we should take a look on how the attendants may react, this answer is correct at first, but nowadays we have a lot of resources and with the help of a proper technological knowledge and *image classification algorithms* we could achieve something far away from the fact of just looking how attendants react.

Nowadays data is every where to be used and this makes it easier for us people with the proper technological knowledge and even people that just have a big interest on technology (thanks to the internet anyone can search and learn about anything) to transform this data in something pretty useful, in this project the aim is to create a system that allows the user to determine the performance of a given presentation with the help of an emotion detection model along with the human action through the use of a usable user interface.

The technology and the resources available make this development process an amazing challenge that when fulfilled it will provide a powerful system to be use as a tool for professors, experts or anyone interested on deep learning study.

While development the need of amplifying the developer base of knowledge about topics regarding artificial intelligence, image classification problems, user interface design and programming will be required for accelerating the project development and fulfilling project deliveries deadlines.

All the decisions, methodologies, tools and results will be shown in this document intended to give everyone who reads it a full overview of the mentioned elements, without further delay, lets start with some general aspects of the development such as the motive, background of the project, some facts about the project and the structure of this document.

1.1. MOTIVE

The main way of presenting any information to others implies always that someone who has the knowledge has to present that information to the ones that hasn't have the knowledge or those who want to amplify this knowledge, there are different ways of presenting this knowledge to others and we can classify these in:

- **personal** or **interpersonal**, the first one being the most direct that facilitates the interaction between emitter(s) and receiver(s) and is limited to a receiver or a limited number of them, the second one, *interpersonal*, is directed to many receivers but this way of communicating doesn't allow the interaction between the emitter and the receivers.
- **unidirectional** or **bidirectional**, in the case of *unidirectional* communication we assume that there's no way the receivers would we able to respond to the emitter(s), in the bidirectional case is the opposite.

In this particular case we will focus on the personal and bidirectional ways of communicating, to be more specific we will focus on those situations where there is an emitter (can be a teacher or an expert) and a set of receivers (the pupils or people that are just hungry for knowledge).

After determining the scope of the project it can be stated that using all the mentioned concepts above it would be very interesting to create an system to allow these emitters to *analyze their way of communicating to receivers* through the use of a classification algorithm and deep learning to obtain metrics about their performance and show these metrics in a *usable* and *user friendly* system.

1.2. BACKGROUND

Emotions are expressed during personal interactions, the study of how to read these emotions is a tough task and there are several artificial intelligence studies and solutions that are applied by different institutions nowadays, the definition of the concept *emotion recognition* will be shown later in this document, in this section we will focus in the challenges emotion recognition is facing in the present day.

The following table shows the general characteristics of every emotion, these characteristics are used by emotion recognition models that extract the characteristics of an image as the Table 1.1 shows:

Table 1.1: Main emotions and their representative characteristics

Emotion	Characteristics
Angry	Lowered and burrowed eyebrows, intense gaze and raised chin
Happy	Raised corners of mouth
Surprise	Dropped jaw, raised eyebrows and wide eyes
Fear	Open mouth, wide eyes and furrowed eyebrows
Sad	Furrowed eyebrows and lip corner depressor
Disgust	raised top lip, wrinkled nose and narrowing of the eyes

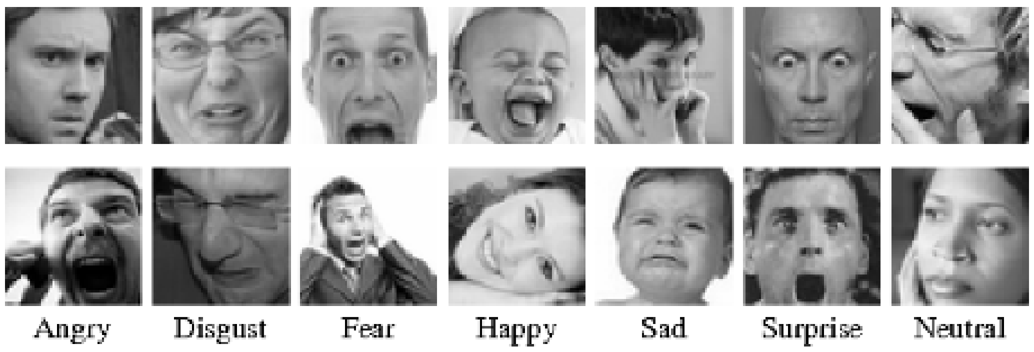


Figure 1.1: Set of images from a sample of FER2013 data-set

Figure 1.1 shows a data sample from the data set *FER2013* which can be found in the *Kaggle* website, this data set was used to train the emotion detection model for this project.

Deep learning is an artificial intelligence function that mimics the human brain by processing data through a set of interconnected nodes usually called *neurons*, for emotion recognition the algorithm is composed by several layers, an input layer, the hidden layer and the output layer, each layer modifies all the input values in order to transform them into the target and preferred output (the emotion a face is showing in this case, deep learning has more uses and we will discuss them later).

1.2.1. Facial recognition

This technology is becoming more advanced every year [1] but at its core the system detects and studies the expressions extracting the characteristics of every face processed and depending on a set of factors in concludes with the emotion as an output, main factors are as follows:

- The location of eyebrows and eyes.
- Position of the mouth.
- Changes on facial features.

An study held in *2012* summarized the system algorithm as shown in Figure 1.2

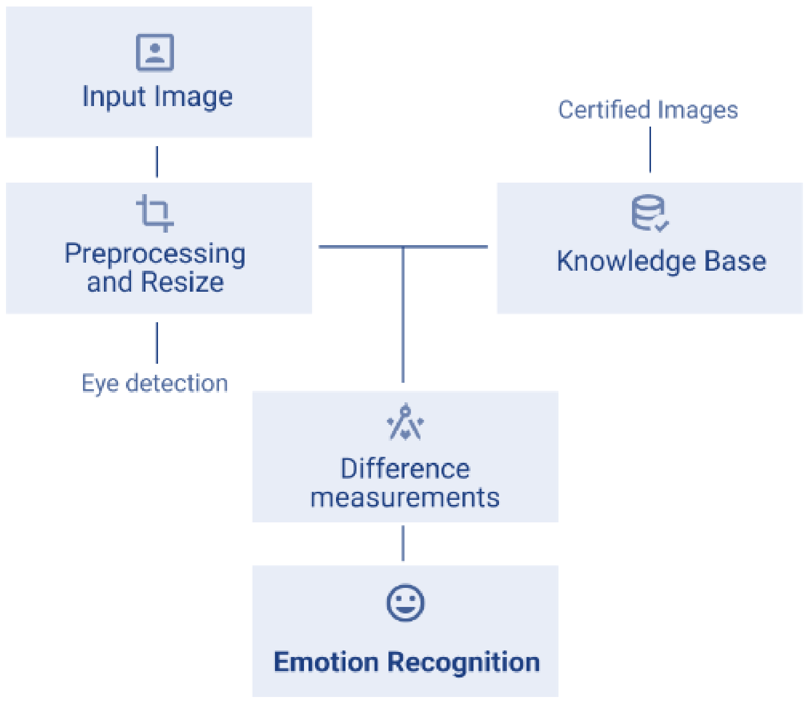


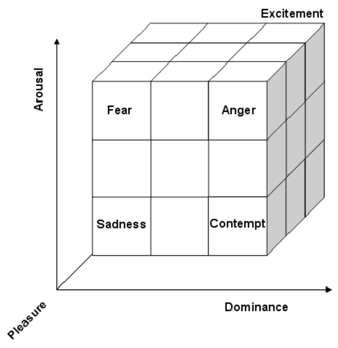
Figure 1.2: Emotion detection algorithms summarized

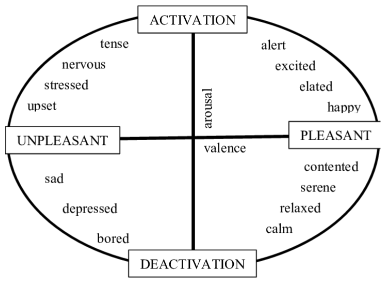
About Figure 1.2 [2] its important to mention the **knowledge base** which is a data base used for comparisons in the **difference measurements** phase which contains a set of images relevant for the emotions to be detected, in the **prepossessing and resize** phase the system enhances the input and removes noise, then the input is resized usually using the eye selection method.

There are two approaches when training a emotion recognition software, the **categorical** way and the **dimensional** way, in the categorical way there is a finite set of emotions that falls into various sets of classes, this is the way the *AIWake* project emotion recognition algorithm was trained, in the dimensional way the emotions are not defined concretely instead they exist on a spectrum based on this approach. *PAD* emotional state has three dimensions while the *Circumplex* model of affect uses two.

The chose of the approach relies on the way the emotion recognition algorithm will be implemented, with a **categorical** approach a classifier has to be implemented while, when the **dimensional** approach is chosen the algorithm must implement the outputs on a sliding scale.

Figure 1.3 shows both approaches schemes of the dimensional way of determining emotions in an emotion detection process.





(a) PAD model uses three dimensions **(b)** Circumplex model uses two dimensions

Figure 1.3: three dimensional model vs two dimensional model for emotion recognition

1.2.2. Emotion recognition

Emotion recognition is the process of identifying human emotion through machines in the same way humans are capable to recognize human emotions by looking at each others faces in a conversation, this section provides a quick view at the nowadays real-world uses of this technique and their impact in the world as emotion recognition is one of the core elements of this project.

- **Is emotion recognition effective?**
  Emotion recognition technologies are not near to be perfect, in simpler terms, emotion recognition consists in the classification of facial expressions, this *classification problem* brings faults and there is not an emotion recognition system nowadays capable of being 100% accurate in its emotion guessing for example: systems could consider subtle emotions and expressions more alarming than those which actually are so it cannot distinguish which emotions are genuine and which are not and could be deceived easily.

  Also there is the cultural problem, different cultures express their emotions differently from others which make emotion recognition systems unable to produce correct conclusions so there is always a certain risk of misinterpretations.

  At the end emotion recognition is a growing technology that becomes better everyday but the problem with *classification problems* still there, one goal of this project is to make the users able to detect these misinterpretations and correct them.

- **The fields of emotion recognition**
  This section encompasses all the fields where emotion recognition solutions are applied in our case we will focus on video and image but this is something to be developed more deeply throughout this document.

  Nowadays emotion recognition is employed in customer service using cameras that compare the customer emotions before and after being attended by the customer service to determine the level of satisfaction of customers, emotion recognition is also used for helping differently abled children for example by using smart glasses in children that aren't capable of recognize other's emotions.

  More solutions brought by emotion recognition are implemented in video game testing, some companies in the video game industry are using emotion recognition on testing phases of new games in order to obtain feedback from the users in real-time.

  Main fields of emotion recognition are as shown on Table 1.2

Table 1.2: Main fields of emotion recognition

Field	Summary
Video	lot of research is still on going to understand how to use video in emotion recognition, at the end video is just a set of images (frames) that can be analyzed using emotion recognition algorithms
Image	One of the major sets available in emotion recognition, some research stated that classification of emotions in pictures could be used to sort video sequences into various genres.
Speech	The goal by some studies is to be able to instead of transcribing speeches into texts (which is not capable of emotion recognition) use these speeches for emotion recognition systems. [3]
Text	DTM (Document-Text Matrix) is the usual structure used for texts, DTM records the frequency of words in a document, as it uses individual words it is not appropiate for determining emotions, the emotion recognition approach is to perceive text based on its tone, punctuation, etc factors that are considered by researchers in the research of new data structures for texts.
Conversation	Focused on adquiring emotions from discussions between two or more individuals, the data-sets used are commonly from free samples from social platforms, one big challenge of this field is the detection of sarcasm, emotional shifts and context.

- **A real world emotion recognition solution to help kids with autism**
  Autism spectrum disorders (ASD) are a group of diverse disorders that are characterized by a certain level of difficulty in social interaction and communication, ASD also shows atypical activity and behaviour patterns, in *2018* [4] it was estimated that almost 2.3% of 8 year old kids had ASD.

  This difficulty in social interaction makes ASD children to have a big lack of social skill, with the arriving of emotion recognition and Google Glass device, a group of researches at the *Standford University School of Medicine* began to study the effects on using *Google Glass* devices along with an *Android* that made use of emotion recognition in order to improve ASD children to improve their social skills.

  The researches called the therapy *Superpower Glass* [5] to help make it appealing to children, it uses flash cards that depict faces with different emotions.

  It uses a categorical approach of emotion recognition having eight core expressions: *happiness, sadness, anger, disgust, surprise, fear, neutral* and *contempt*, it includes a mechanism to allow people involved in the study to calibrate their own neutral faces if necessary.

  The study tested 14 families for an average of 10 weeks each, these families had a child between the ages of 3 and 17 with confirmed ASD diagnosis, the families used the therapy for at least three 20 minutes sessions per week.

  The program was designed to allow three ways to use, *free play* were children wear the glasses while interacting with their families and two game modes *Guess my emotion* and *Capture the smile.*

Results were that 12 of 14 families said that their children made more eye contact after the therapy and the keys were able to realize the clues of feelings in faces, it was proven after evaluating the children's that their *SRS-2* (questionnaire completed by parents to evaluate children's social skills) scores decreased, meaning that the children's had less severe symptoms of ASD and never increasing the scores in any case, 6 out of 14 children's declined their scores so dramatically that they even moved down one step in the severity of their autism classification.

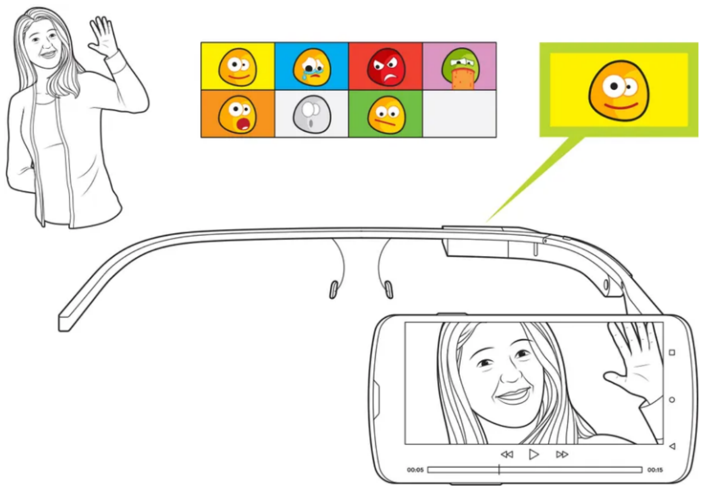


Figure 1.4: Superpower Glass is a therapy that aims to improve ASD children social skills

Figure 1.4 represents the concept behind the Superpower glass project.

1.3. THE AIWAKE PROJECT

In this chapter we have had a brief explanation on emotion recognition and face detection systems and nowadays research, this section encompasses a description of the system that is going to be developed and the reasons behind its development.

The chosen name for the system was *AIWake* from Artificial Intelligence *AI* and the *Wake* verb which refers to the problem it wants to solve: analyzing the emotions of presentation attendants in order to provide the speaker with a set of feedback (which makes able the speaker to analyze the level of attention of attendants, the more attention paid the more awake the attendants).

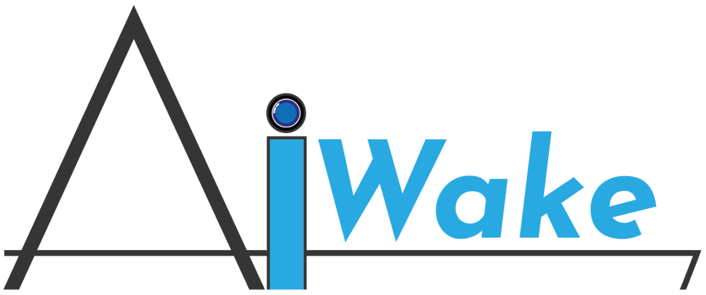


Figure 1.5: the logo chosen for *AIWake* system

Logo shown in Figure 1.5 was inspired on cameras due to the use of video recordings as input for the emotion recognition algorithm, can you spot the camera shutter behind the logo (not the *i* dot)?

So lets give a little context about the project, since *2019* the developer of this project began to be part of *C:TED* which is a department of *UCLM* in charge of recording important events and produce multimedia products that are delivered directly to the *UCLM*, presentations with high level of attendants were filmed during all these years and the idea of developing an system to analyze the attendants expressions started to grow but its development was dramatically slowed because of the arrive of *COVID-19*.

As an emotion recognition project *AIWake* its an opportunity to learn about emotion recognition techniques, there is not much information about other tools out there that are used for emotion recognition in the context of presentations, as an example shown in the last section, some are used for example for retrieving feedback in customer service applications also its important to mention one big challenge of this project: the fact that we are looking to analyze multiple faces in a single image (a frame of a video) which implies the implementation of a top-to-bottom algorithm where we first scan the video for face appearances and then we analyze each detected face individually.

After all this time the system has been changing due to the problems found (mainly *COVID-19* as mentioned before) and since November *2021* the development has follow the steps that are going to be shown in this document, to give an example, the project started as a *C++* and *Intel OpenVino* project in *2020* but the idea was discarded due to lack of flexibility *Intel OpenVino* had (and the fact that it only allowed *Intel* processors to be used in data processing) in a mid-development phase (The final project uses *Python* and *Tensorflow* so we can conclude that a lot of things have changed).

With the acquired knowledge of Artificial Intelligence the development of a system which with the use of a set of frameworks for *computer vision*, *emotion recognition* and *UI development* along with an strong *programming language* in terms of *Artificial Intelligence* fulfilled the objectives specified in this document second chapter and solved the problem, the project can be improved by adding more functionalities to it as the used methodology allows it, improvements that could make this tool an amazing solution for analyzing performance of presentations will be shown in the final chapter of this document.

1.4. DOCUMENT STRUCTURE

The documentation for this project will be following the next structure following the guide lines for final degree project documentation provided by the *Escuela Superior de Informatica of Universidad de Castilla-La Mancha*.

- Chapter 1. Introduction:The actual chapter, it will depict general information about the project, some context and the background
- Chapter 2. Objectives:A brief view of the challenges the project will face, in this section each characteristic of the system to implement will be reviewed.
- Chapter 3. State of the art:Review of topics encompassing the project and quick view over some examples of solutions that employ these topics.
- Chapter 4. Methodology:This chapter will cover the followed development methodology for this project and the tools used for development.
- Chapter 5. System architecture:Detailed design and implementation of the proposed solution.
- Chapter 6. Results:As it name says, this chapter will cover results at each phase of the development and will include the different design choices taken during the development.
- Chapter 7. Conclusions:A more personal review of the project development process, this chapter will show thoughts that came during the development and also will depict some new ideas that could be added to the project in the future.

CHAPTER 2
Objectives

In this chapter the objectives that will lead the development and the purpose of each of this objectives along with the procedures involved on its development will be shown.

2.1. MAIN OBJECTIVE

The main objective is to develop an system that makes a user able to analyze and determine which were the emotions expressed by the attendants of a presentation, a set of frameworks, data sets and libraries will be employed, the tools involved in the development will be depicted in the *Methodology* chapter. To reach this objective *Python* programming language knowledge is required along with concurrent programming and data management, the sub objectives shown in this chapter are all related to the completion of the main objective and will be shown in the following section of this chapter.

2.2. SUB OBJECTIVES

1. **Acquire knowledge of deep learning techniques to be able to generate a model for image classification**. this sub-objective is very important in order to give the system a unique touch but the system must be flexible in order to allow users to use their own classification models.

   In order to achieve this objective, the learning of lot of concepts about Artificial Intelligence from the internet, looking at research papers about emotion recognition models, a data set for model training that will be downloaded from the website *Kaggle* (a community focused on machine learning) and make use of machine learning frameworks such as *Keras* and *Tensorflow*.

2. **Define which technologies will be used in order to achieve the main objective and stablish a proper project planning**. time is the most important resource everyone has so having everything clear from the start will allow a faster development, the technologies and planning used for this project will be discussed in future sections of this document.

   To give a brief summary of the methodology the project will follow its development will consists on different **iterations**, each iteration will focus on a project functionality and the next iteration will extend the functionalities, the idea is to make it modular for example: output of iteration 1 will be the input for iteration 2 so each iteration can run independently making a possibility for the functionality of iteration 2 to be started without running the iteration 1 functionality, the emotion recognition process will depend only on the output of the face detection functionality so we can start from the emotion recognition part if we already have a file containing the data processed by the face detection part.

3. **Create a user friendly system**. in order to fulfill this objective, a good knowledge base about user interface design principles is required to employ usability and Gestalt principles which are necessary in any heavily user oriented system.

We will discuss about usability in following chapters of this document, for the UI design part a popular framework for UI development known as Qt will be integrated in the project development, Qt is a set of C++ libraries that allow the developer to implement UIs for different applications, in order to use Pyhton along with Qt the Python wrappers set for Qt provided by PyQt module will be employed, the results will be shown with a set of screenshots of the final UI of the project in the *Results* chapter.

4. **Establish a style guide to make an unique system**. colors are one of the most important aspects in terms of making something unique, establishing a style guide (buttons of the system, background colors, fonts...) will allow the system to be more appealing, have high contrast to have more readable texts and basically produce a feeling of love at first sight for the user.

   Qt uses QML (Qt Meta Language) which is based on *JavaScript* in order to implement the design aspect of a Qt application, similar to *CSS* we can establish a set of style rules for the elements of the UI an preview these rules directly in the IDE which we will employ for the design of our system *Qt Creator*.

5. **Use principles of parallel and concurrent programming**. the processing power needed by this kind of system is huge, a lot of sub-processes will take place during the main execution of the program so there is a big need of concurrent programming techniques.

   The architecture of the system will implement a set of signals and threads in order to control parallel processes, Qt provides the libraries *QThread* and *QtSignal* this set of signals and threads will provide the system with control methods for the concurrent aspect, the use of threads is mandatory on systems that run over an UI were the main thread is the system container as any different process from the main window would freeze all the UI until the heavy processing is finished.

6. **Give the user a high level of feedback from the system**. this system requires a great processing power due to use of big data models and use of classification algorithms to analyze the images the user will provide, this makes the fact of giving constant feedback like information messages, real time data, etc to the user a priority.

   Taking advantage from the Qt library we can use *QtSignal* module to make the sub-processes of the system able to send messages to main thread, giving feedback to the user is an important aspect of any system in order to make the user less impatient while a high amount of time is required for a process to complete, elements such as progress bars and information labels are our best allies in terms of feedback.

CHAPTER 3

State of the art

This chapter will address the core elements of this project, by core elements i am referring to all the technologies that will see use in this application, their actual state and main uses.

3.1. ARTIFICIAL INTELLIGENCE

Nowadays the world is seeing big changes due to the presence of growing technologies which is considered as a new industrial revolution, artificial intelligence [6][7] has attracted much attention from governments, industries and academia's, a proof of this fact is the amount of articles published in recent years over this discipline that are selected and explored, in this section we will take a look on the state of the art for artificial intelligence and some of its more relevant projects and applications that are present in the real world.

First lets start with some context, artificial intelligence is as know in computing science a discipline that tries to replicate and develop intelligence and its implicit processes through machines, artificial intelligence follows four approaches:

1. Centered in humans.
    a) Systems that **think** like humans.
    b) Systems that **act** like humans.
2. Centered in rationality.
    a) Systems that **think** rationally.
    b) Systems that **act** rationally.

The concept of artificial intelligence began after second world war in *1956* in the *Darthmon conference* by the computer scientist *John McCarthy* and as a science that has been coined since *1956* it has achieve great challenges as the defeat of the chess world champion by the computer *AlphaGo* in early *2016*, you may be thinking on the case of *Deep Blue*, the machine that defeated **Kaspárov** playing chess in *1996*, the main difference is the fact that *Deep Blue* was brute-force programmed this means that the computer was programmed to play chess and due to the use of a massive parallel processing based in *RS/6000* it was able to win (but Kaspárov won 3 matches and 2 resulted on a draw after the *Deep Blue* first win and won 4-2 against the machine) and *AlphaGo* is a **computer program** developed by *Google DeepMind* using the programming language **Go** that employs true artificial intelligence that is able to learn and even achieve a *superhuman* level on the game after only 4 hours of playing (Figure 3.1).

As a curious fact, Kaspárov asked **IBM** about registers of the machine because he thought the machine was being manipulated by a human and **IBM** never gave him those registers which made Kaspárov to declare that the event was organized for propagandist purposes and it was never meant to be an scientific event.

(a) Deep Blue, a *machine*

(b) AlphaGo, a *program*

Figure 3.1: programmed to play vs programmed to learn

Artificial intelligence is a knowledge project that takes the knowledge as the object to acquired knowledge and at the end achieve the effect of simulating human intellectual activities it groups computer science, logic, biology, psychology, philosophy and other disciplines to achieve great results in activities such as:

1. **Speech recognition**: converts human speech from analog to digital form and its powered by *Natural Language Processing* and *Machine Learning*, the digitized speech can be use for applications in smart phones *(Apple, Siri)*, smart homes *(Amazon, Alexa)* and other voice activated solutions.

2. **Image processing**: its in charge of making algorithms that are able to process images like the human brain does, also its the core of this project, there are a lot of libraries that allow image processing programing such as *OpenCV, Tensorflow, PyTorch* or *Caffe* some of then will be discussed deeply in this document.

3. **Natural Language Processing (NLP)**: the ability of a computer program to understand human language as it is spoken and written that has been existing for over 50 years and relies in the field of linguistics, it is not the same as Speech recognition as mentioned above, NLP is focused in the natural language and it can extract meaning of words based on the context, recognize named entities and even generate new texts based on its knowledge.

4. **Intelligent robots**[8]: robots with a well developed artificial "brain" which can arrange actions according to a purpose and the help of sensors and effectors, main uses of this intelligent robots include industrial robots, service robots and mechanical rehabilitation robots, they use simulation environments for testing and researching purpose, some of these environments like *NVIDIA's Isaac Sim* are built on top of *Unreal engine or Unity* platforms, a close example of an intelligent robot is EVA (Figure 3.2) [9] an affecting robot for old and dependant people being developed by MAmI group *(Castilla la mancha University)* in collaboration with CICESE *(Ensenada Public University)* and the *Healthcare Robotics Lab (University of California San Diego).*

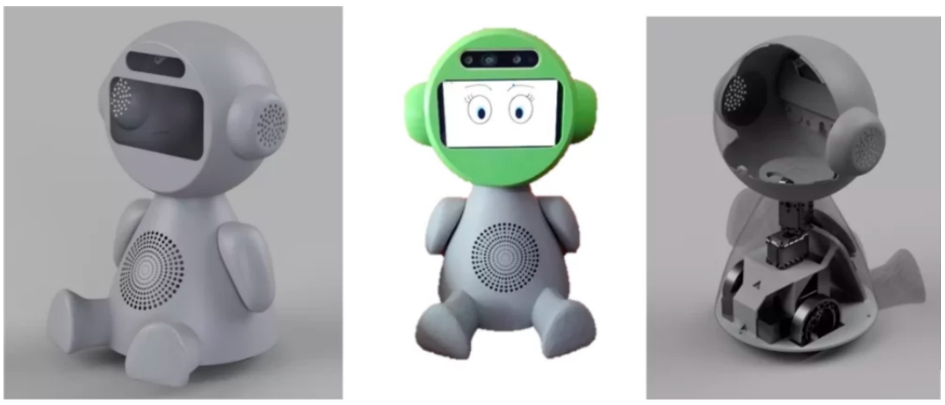


Figure 3.2: EVA, an affecting robot being developed by MAmI group in collaboration with CICESE and *Healthcare Robotics Lab*

3.2. COMPUTER VISION

Computer vision [10] refers to the set of tools and technologies that make systems able to take images from real world, process those images and produce a set of information, it try's to emulate the way the human vision works but translated to a context of machines.

The goals and functions of computer vision can be grouped in these 3 big groups:

1. **Object detection**: takes the idea of computers being able of recognizing patterns found in images to separate and classify them, these objects can be humans, buildings, cars, etc, object detection has many applications including **face detection**, video surveillance and image restoration but one of its most common use case if the facial recognition. It takes advantage of the characteristics any object has, a ball is round, a face has eyes, mouth, nose and hair (or not).
2. **Image analysis**: takes advantage of the capacity a computer has in terms of extracting the characteristics of a big set of images, in terms of the human capability to analyze an image against a computer processing capability a computer can process thousand of images in a second and collect the data in a human readable format.
3. **Statistics**: with the above mentioned about the capacity of a computer to process large sets of images, computer vision can be applied in order to compare the extracted characteristics of a set of images, imagine for example a program which uses a video taken during a day in a motorway were we want to obtain the amount of Ford cars passing through the motorway and compare it with the amount of KIA cars, a human person would need to count the cars appearing in the video manually and spending almost the entire day, a computer vision program could fulfill this task in minutes producing useful data for car vendors, an approximation of this example can be seen in Figure 3.3.

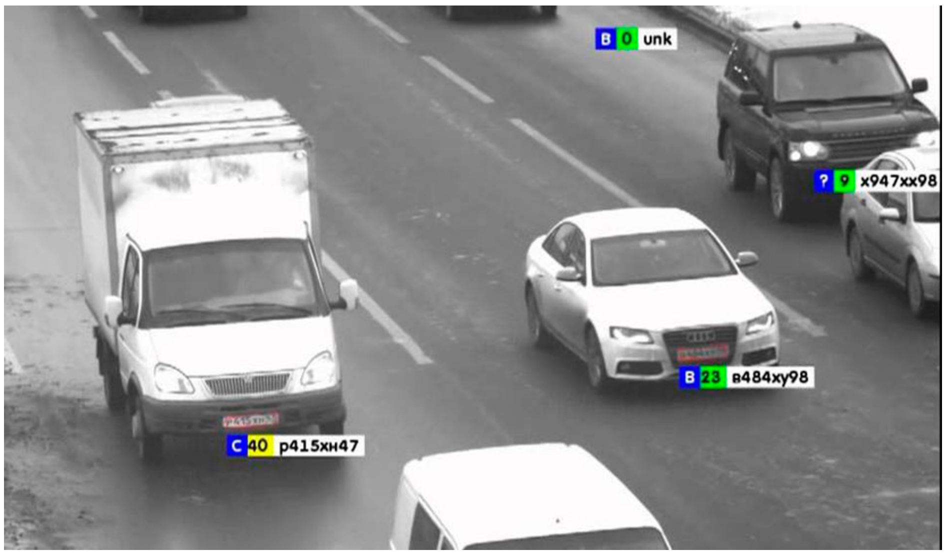


Figure 3.3: A computer vision example of a program that is able to read car plates

The use of machine learning in computer vision [11] is applied when only the recognition of patterns in a set of images is needed, the training process for a computer vision solution through machine learning consists on giving an artificial intelligence system a set of instructions previously classified, ordered and interpreted in order to make the system get used to the patterns and respond properly to them. If we are looking for a more advanced and complex solution then we need to use deep learning [12] techniques to build the system, giving the system a set of rules and let it learn to recognize the patterns by itself which is a slow process but allows the system to recognize new patterns once trained.

3.2.1. Computer vision applications

Making a system that is able to recognize the environment has become an important trend in the industrial context and this systems have come to stay in our daily lives [10].

Here is a list of the main applications computer vision has at the present day, there are a lot more but these are the most present ones specially in the industrial world.

1. **Electronics**: Industries have focused the computer vision approach to the components selection which facilitates the selection of components replacements and the automated welding giving more security in terms of human workforce and allowing the efficient welding of small components and pieces.
2. **Food industry**: Computer visions allows to avoid millionaire losses to agri-food companies by detecting bad batches of products or intruders in the preparation and packing lines which also provides the industry with a higher quality assurance.
3. **Automotive industry**: This was one of the most benefited industries when computer vision came, computer vision introduced the use of industrial robots in the assembly and welding processes, these robots are able to recognize specific pieces and welding supervision which translates in higher quality standards and productivity.
4. **Logistics**: Automation of logistic processes translates in higher productivity, computer vision is applied in product picking, inspections of products in order to fulfill quality standards and the classification of products.
5. **Security**: Stand-alone security systems came along with computer vision, this systems are able to recognise personnel, detect intruders and together with proximity devices these systems are able to event prevent accidents.

A new trend in computer vision is **3D Vision** which makes the system to recognize objects and perceive the environment around this object as an human would do, this has great applications in navigability solutions and in high level industrial processes allowing the system to measure pieces and specify the measurements in order to buy this pieces which is very important in the industry due to the fact that 3D Vision is able to measure even in microscopic scales that are invisible to the human eye, this concept of 3D Vision and computer vision in general collaborated in the emergence of the concept that nowadays is known as **Industry 4.0** (Figure 3.4).

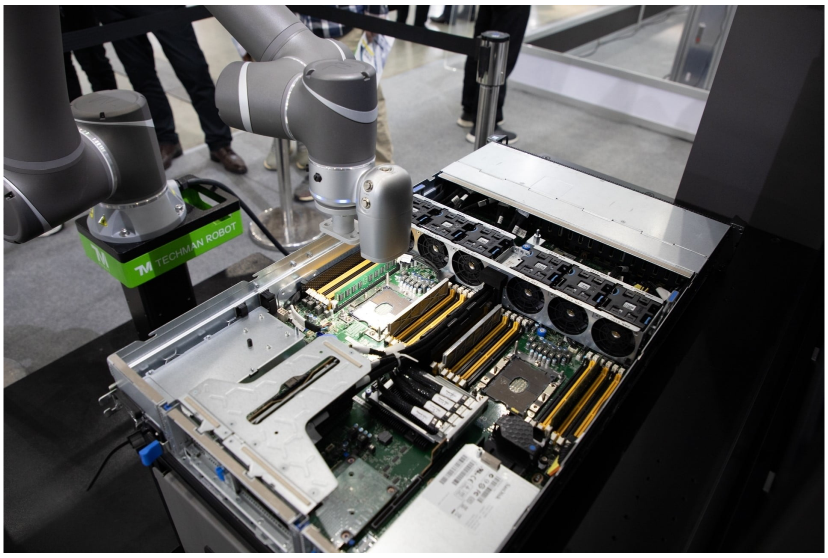


Figure 3.4: Industry 4.0 is supported by automatons that take advantage of computer vision to operate, in this case for motherboard components inspection

3.3. DEEP LEARNING

In this project in order to generate a model for our application we will need to introduce the concept of *deep learning*, *deep learning* is a type of *machine learning* both of them being subsets of *Artificial Intelligence, AI*, which is a science devoted to make machines think and act like humans (in an intelligent way).

The key difference between these two concepts is the approach each one has, *machine learning* aims to make computers being able to perform tasks without the need for explicit programming, with *machine learning* computers are given a set of structured data (Figure 3.5) and they learn to become better at evaluating and acting on that data over time.

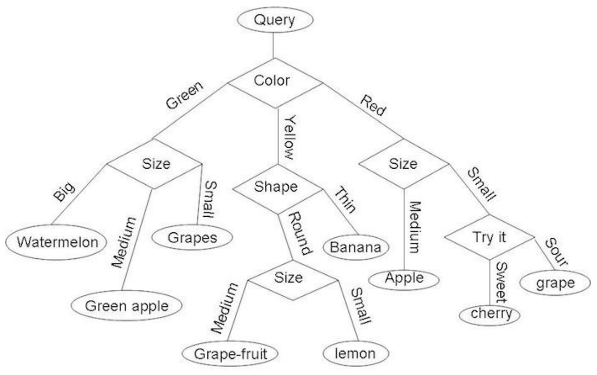


Figure 3.5: A decision tree, the way machine learning acts to determine its outputs

The "problem" with *machine learning* is the fact that even the computer is able to act more accurately over time it will still think and act as a machine so *deep learning* addresses this "problem" by creating a model after the human brain. Complex, multi-layered *deep learning networks* (Figure 3.6) allow the data to be passed between *nodes* that act like *neurons* in a highly connected way.

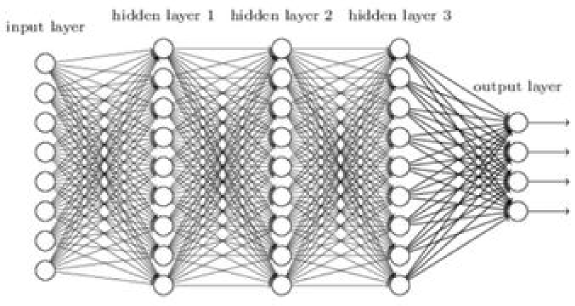


Figure 3.6: A example on how nodes are connected in a deep learning model

3.3.1. Main types of deep learning algorithms

Its is important to mention the two main ways of building a *deep learning network* at the approach both have in order to illustrate how *deep learning* can be applied.

3.3.1.1. Convolutional Neural Networks

are focused on *image classification problems* [13], this type of algorithm is commonly referred as **CNN** and can take an image as input, assign weights and biases to various aspects of the image and by doing this being able to differentiate one image from another.

It is a very interesting fact that convolutional networks are based on the connectivity pattern of the organization of the neurons in the human brain visual cortex.

The way this networks work is by using the pixels that conform an image by applying relevant filters, but why it is called *convolutional*? because it uses convolution layers, the element in charge of the convolution is called **Kernel/Filter, K,** (Figure 3.7) a matrix that goes over the image pixels and gets the convolved feature.

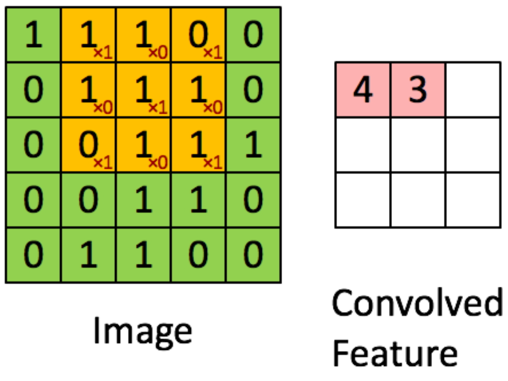


Figure 3.7: A graphic example of a 3x3x1 kernel convoluting a 5x5x1 image

When the whole image is traversed by the kernel the convolved features allows the network to extract high level features such as edges.

There also another layers used in this type of network, the **pooling layer** is similar to the convolution layer but it reduces the spatial size of the convolved feature in order to decrease the computational power required to process the data and the **fully-connected layer, FC Layer** that is used as a cheap way of learning non-linear combinations of the high level features represented in the output of the convolutional layer.

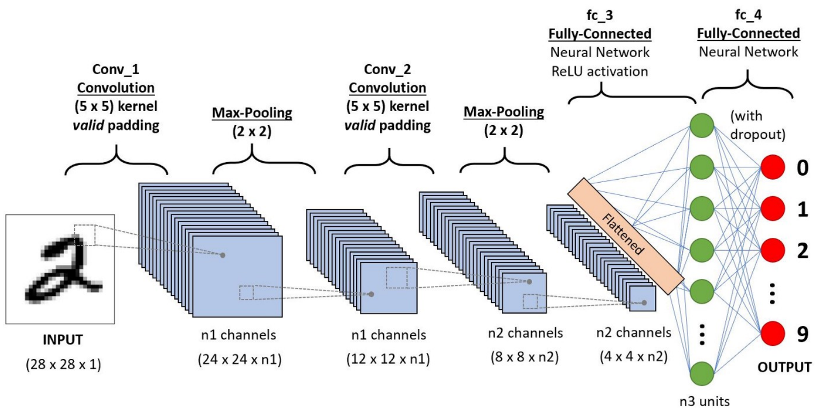

Conv_1 Convolution (5 x 5) kernel *valid* padding
Max-Pooling (2 x 2)
Conv_2 Convolution (5 x 5) kernel *valid* padding
Max-Pooling (2 x 2)
fc_3 Fully-Connected Neural Network ReLU activation
fc_4 Fully-Connected Neural Network
(with dropout)
Flattened
INPUT (28 x 28 x 1)
n1 channels (24 x 24 x n1)
n1 channels (12 x 12 x n1)
n2 channels (8 x 8 x n2)
n2 channels (4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

Figure 3.8: CNN example, notice the use of the different layers mentioned.

As we can see in Figure 3.8 the input imaged is processed layer by layer, the **convolution layers** extract the features, the **pooling layers** reduce the size of these convolved features and the **FC layers** classify those features producing a final output.

AIWake requires the processing of images in order to classify emotions determined by the expressions the attendants of presentations will make so a **CNN** will be required for this purpose, the process of making this **CNN** will be depicted in future sections of this document but in order to fully understand how deep learning tries to mimic human thinking lets briefly talk about the other main deep learning type of algorithm.

The complexity of *deep learning* allows the generation of data models when using a **CNN** that are capable of solving image classification problems like the one we will have to face in the *AIWake* project, so in this case we will need to generate a data model to classify our images, we will talk about the process of generating this model in future sections of this document.

3.3.1.2. Recurrent Neural Networks

also known as **RNN** introduce the concept of memory which makes the computer able to keep data points and decisions in mind and use this in order to review new data.

RNN have a mayor focus on natural language processing work and the connections between its nodes form a directed or uni-directed graph along a temporal sequence which is the main reason why this kind of networks are able to "remember".

As in *AIWake* we are not interested in natural language processing there's no need to expand this section more.

CHAPTER 4

Working method

This chapter will describe the steps took to achieve the final goal of the project, the chosen methodology, the tools used for development and the context around this project, as the project was developed by a team conformed by one person in charge of programming and a supervisor an *agile* methodology based on development iterations focused on specific functionalities that have an established period of time to be developed and then reviewed was chosen.

4.1. METHODOLOGY

As said before, an agile methodology was taken, this methodology will be described in detail in this chapter but in order to summarize the development of this project will consists in several phases/increments, each iteration will focus on one big aspect of the overall functionality, lets take a look at this iterations:

1. **iteration 0**: Generate a deep learning model for solving a classification problem.
   a) Identify the data to be processed by the model
   b) Train the model using a data set
   c) Evaluate the generated model
2. **iteration 1**: Pre-processing of video source, this iteration covers the face isolation aspect of the application.
   a) Allow user to browse for a video source
   b) Load this video source in the application
   c) Develop an algorithm for isolating faces appearing in the video
   d) Generate a *json* file containing all the data obtained after the face isolation processing
3. **iteration 2**: Post-processing of video source using the data obtained in **iteration 1** in order to employ the model generated in **iteration 0** to detect emotions.
   a) Load the data generated in **iteration 1** and saved in a *json* file that contains positional information about each person appearing in the video.
   b) Crop the video source according to the positional information given and send each cropped image to the data model
   c) Generate a *json* file with the obtained data after processing the cropped images through the data model
4. **iteration 3**: Data review, in this iteration most of the UI programming will take place, the idea is to make an UI to allow the users review the data obtained on each iteration allowing them to modify data and some parameters regarding processing.
   a) Integrate Qt with the application and start modeling the UI
   b) Create the UI options for **iteration 1** results review

   c) Create the UI options for **iteration 2** results review
   d) Make the UI fully responsive and establish the styling
5. **iteration 4**: Create charts to allow data analysis in the UI.
   a) Create a chart view for the **iteration 1** processing data
   b) Create a chart view for the **iteration 2** processing data
   c) Allow the user to pick different data representation options.

That's a glance on the steps the development will take, all this steps will be depicted in the following section, its important to mention the use of *Git* along with *GitHub* that was present during all development and can be checked on *https://github.com/Kirkuss/TFG*, the use of *Git* will also be shown in this document but for now all that is needed to know is that every **iteration** was developed following the *Gitflow* [14] model for branch usage.

4.1.1. Iterative and incremental software development

The modular nature of this project creates a perfect situation to follow an iterative and incremental software development methodology where all the **Phases** of the project will be taken as **iterations** and the new functionality added on each **iteration** will be the **increment** for each finished iteration it will be an evaluation of results, if these results are the desired ones that is to say the output of that iteration follows a *deterministic* way we can continue to the next **iteration**.

Agile methodologies are highly used nowadays and there is a lot of reference guides to follow an agile methodology, in this case the end of each iteration will be followed by videos with a brief review of the implemented functionality that will be send directly to the project supervisor (*Carlos Gonzalez Morcillo*) who after a look at the video will share some considerations about the functionality implemented.

Iterative and incremental software development is summarized in Figure 4.1

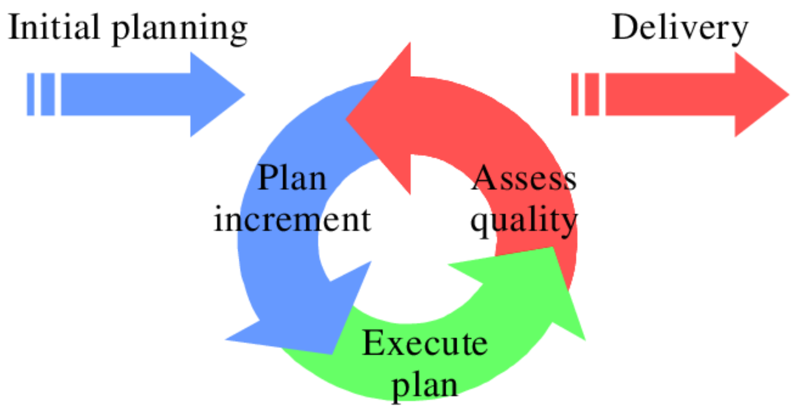


Figure 4.1: Iterative and incremental software development takes each iteration as a cyclic process that goes over different phases

4.1.2. Planning

This section will show the steps that followed the completion of the stated goals in *Chapter 2* with the time taken while developing the main features of the project and some considerations that appeared during the development of each feature.

- **iteration 0, generating a deep learning model for a classification problem**
  This functionality consisted on achieving a certain knowledge level about artificial intelligence and deep learning, the learning process started in *September, 2021* and finished in *December, 2021* it is relevant to say that before starting focusing on the aspects regarding this application

in *September, 2021* there was already some knowledge about artificial intelligence due to the fact of the several attempts of developing the project that got rejected because of problems like *COVID-19*.

After achieving the required knowledge the implementation of a model trainer began and it was deployed in *December, 2021*, the architecture will be depicted in the *Architecture* section, now lets focus on the main tasks involved with this 4-month iteration.

1. **Obtain knowledge about classification models training**: the most demanding task in terms of time, the need of adapt the acquired knowledge of data models to the selected frameworks and programming languages and the chose of a data set for training was a difficult task to complete.
2. **Implement the model trainer**: due to the complex nature of deep learning models keeping it simple in terms of implementation was chosen, overall process took 1 month from *November, 18* to *December, 21*.

The time estimation for this iteration was 3 months but problems with memory management and elements (like *NVIDIA CUDA*) integration delayed the iteration completion with an additional month this delay was palliated with a parallel implementation of **iteration 1** which along with the modular nature of the project was possible (both functionalities were independent) this fact violates the iterative and incremental methodology principles but it was required due to the time demanding development and the modularity of the project, end justifies any means and the results were great.

- **iteration 1, processing the video source in order to detect the faces to be processed**
  As said in the previous iteration planning specification, **iteration 1** was implemented during the **iteration 0** development in a 2-week time lapse that took from *October, 27* to *November, 17*, the estimated time for this iteration was 1 month but at the end it was completed on a 1 week period.

  Hardest part of this iteration was the implementation of an algorithm to determine if the appearance of a face in a new frame of the video belongs to a face that appeared in another frames before in what was called the *Face Isolation Algorithm* its development took less than a week and is one of the core features of the application, this algorithm will be explained in detail in the *architecture* section.

  To process the video source *OpenCV* and *OpenCV haarcascade model* were integrated into the source code, a simple video window allowed the testing of the algorithm output, despite some performance problems that were solved during this period everything went smooth as silk.

- **iteration 2, processing the video source to obtain data regarding emotions**
  As the goal of this iteration stands we can deduce that **iteration 0** and **iteration 1** outputs are needed for the implementation of this iteration, this iteration implementation was similar to the **iteration 1** implementation as it also needed of **OpenCV** to work and the video to be processed but it introduced new problems such as performance and data management, so a 2 week period was established to work on this iteration, the implementation of a first version took from *December, 21* to *January, 3*.

  It was working perfectly, not smoothly, the output of the iteration was the desired one (we were able to obtain data regarding the emotions detected in the video and produce a *json* file regarding that data) so the decision of leaving the development of this iteration as completed but awaiting for a future refactoring in order to obtain better performance results was made.

- **iteration 3, creating an UI for reviewing obtained data**

In this iteration the main approach was to take all the previous phases functionalities and integrate these on a UI, time estimated to obtain a first version of the UI was 2 months but at the end it got expanded another month in order to start some code refactoring (having an UI allowed testing functionalities easily), refactoring that took a little longer than expected, this iteration was divided in sub-phases/iterations:

- **iteration 3.5**: Intended to increase the performance of **iteration 2** execution and the way data was processed at the outputs of each iteration, covered an extensive search for parallelism and signal behaviour of the application, took more time than intended due to the search of a way for image compression which was discarded due to very large size in output.
- **iteration 3.5.5**: A lot of general errors were solved in this sub-iteration and the development of the system's main user interface got more complexity by adding style rules and responsive behaviour.

This iteration including sub-phases was deployed on *April, 27* and it started to show the big potential the application had, but it took more time than intended hardening the project development time restrictions.

In this iteration the integration for *Qt* libraries and the use of *Qt Creator* in the application took place along with some basic data charts that allowed a first contact with the final product.

- **iteration 4, integrate charts in the application to allow detailed data analysis**
  Intended to end in *May, 31* it took 6 days more and ended in *June, 6* due to a problem found in *matplotlib* library that required the implementation of data charts through *QtCharts*.

  Hardest part of this iteration was the processing of the data generated by **iteration 1** and **iteration 2** functionalities as different charts require different ways of processing the data, also mention the performance issues that having different data plotters brought to the application, some additional research in the *QtChart* library documentation allowed the implementation of different ways of improving performance till the point the impact of the data plotters were almost 0.

  Although the implementation of the UI took place on **iteration 3** the UI was so important for this application that it got a continuous development, this means that **iteration 4** also introduced changes to the UI aspect such as style improvements and more responsive sections.

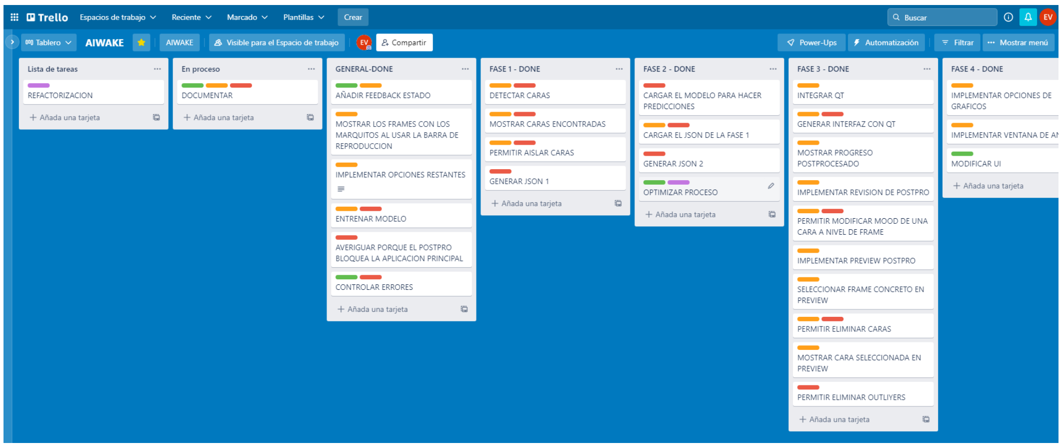
The overall time distribution for each iteration ended like shown in Table 4.1 were the red color represents delays and the yellow color represents that times when some aspects of an iteration were modified to improve the system in general, for example when modifying the user interface.

Table 4.1: Scheme of the phases development

	Sept	Oct	Nov	Dic	Jan	Feb	Mar	Apr	May	Jun
iteration 0	🟦	🟦	🟦	🟥						
iteration 1		🟦	🟦							
iteration 2				🟦	🟦					
iteration 3						🟦	🟦	🟥	🟨	🟨
iteration 4								🟦	🟦	🟥

4.1.3. Development management

All the planning shown above brought a set of tasks to be completed, there are a lot of tools along internet that are free to use that have the purpose of ease the task management of a development process for this project the decision of using **Trello** was made, *Trello* is an online tool that provides a **Kanban** board where the different tasks of a project can be shown and moved through different states, for this project a set of tasks and states were defined during development.



Lista de tareas
- REFACTORIZACION
- Añada una tarjeta

En proceso
- DOCUMENTAR
- Añada una tarjeta

GENERAL-DONE
- AÑADIR FEEDBACK ESTADO
- MOSTRAR LOS FRAMES CON LOS MARQUITOS AL USAR LA BARRA DE REPRODUCCION
- IMPLEMENTAR OPCIONES RESTANTES
- ENTRENAR MODELO
- AVERIGUAR PORQUE EL POSTPRO BLOQUEA LA APLICACION PRINCIPAL
- CONTROLAR ERRORES
- Añada una tarjeta

FASE 1 - DONE
- DETECTAR CARAS
- MOSTRAR CARAS ENCONTRADAS
- PERMITIR AISLAR CARAS
- GENERAR JSON 1
- Añada una tarjeta

FASE 2 - DONE
- CARGAR EL MODELO PARA HACER PREDICCIONES
- CARGAR EL JSON DE LA FASE 1
- GENERAR JSON 2
- OPTIMIZAR PROCESO
- Añada una tarjeta

FASE 3 - DONE
- INTEGRAR QT
- GENERAR INTERFAZ CON QT
- MOSTRAR PROGRESO POSTPROCESADO
- IMPLEMENTAR REVISION DE POSTPRO
- PERMITIR MODIFICAR MOOD DE UNA CARA A NIVEL DE FRAME
- IMPLEMENTAR PREVIEW POSTPRO
- SELECCIONAR FRAME CONCRETO EN PREVIEW
- PERMITIR ELIMINAR CARAS
- MOSTRAR CARA SELECCIONADA EN PREVIEW
- PERMITIR ELIMINAR OUTLIYERS
- Añada una tarjeta

FASE 4 - DONE
- IMPLEMENTAR OPCIONES DE GRAFICOS
- IMPLEMENTAR VENTANA DE A
- MODIFICAR UI
- Añada una tarjeta

Figure 4.2: The state of the Trello page for AIWake

As can be seen in Figure 4.2 a color code for each task was used based on the priority of these tasks, being **red** used for *mandatory* tasks, **orange** for *high priority*, **green** for tasks that can be made over the development, and **purple** for *non-mandatory* tasks that can be implemented in case there's time which due to the time restrictions are the less tasks making an appearance.

4.2. TOOLS INVOLVED ON THE DEVELOPMENT

In this sections the main tools (frameworks, system, IDEs and Libraries) will be described, the entire project development relied on this set of tools and obtaining a good knowledge base for them was mandatory, only mentioned tool that didn't require knowledge was my home pc, all the obtained knowledge about using these tools came from the internet by a set of guides and tutorials in video/text format.

4.2.1. Software

This section will explain the tools selected regarding software the hardest aspect of these software tools was the fact of choosing them due to the amount of tools available for developing artificial intelligence software applications.

- **Windows 10 Operative System:** Operative system used as the development platform, some problems regarding low level system calls in order to apply parallelism in the system execution appeared.
- **Python:** Python [15] is a high level, interpreted and general purpose programming language, being interpreted means that it is not necessary to compile it, it is executed by the interpreter of the computer instead (it is not necessary to "translate" it to machine code), Python includes some built in libraries that allow data representation such as *MatPlotLib.*
  - **MatPlotLib:** Is a library that allows data representation throught the use of lists or arrays of data, it also makes use of the *NumPy* library.
- **Qt:**An object oriented multi-platform framework mostly used in the development of software that needs a user interface, it is also used for console (server-side) applications and command line programs.
  Qt its an open source software developed by the Qt project nowadays a part of Nokia and Trolltech (Norwegian company) are part of the Qt project, Qt is used in KDE Plasma which is a GNU/Linux desktop environment.

  - **PyQt:** PyQt [16] is a comprehensive set of Python bindings for Qt v5, PyQt has more than 35 extensions that enable Python to be used as the development language for Qt, it also allows us to develop using Qt through Python on iOS and Android applications, this set of bindings also can be embbeded in C++ based applications in order to configure or enhance these applications.
  - **QtCharts:** A Qt module that uses the Qt Graphics view framework to integrate charts with modern user interfaces, this Qt module replace the *MatPlotLib* libraries usage at the final stages of the development due to its performance improvement and chart representation.

- **PyCharm:** *PyCharm* is a multi-platform IDE developed by the czech company *JetBrains* for the *Python* programming language that offers a community editions with as its website says *all the python tools just in one place.*
  PyCharm offers advanced code analysis, a graphic debugger and integration with version control systems like *Git*
- **Qt Creator:** An IDE for development of Qt applications, it uses *QML* which is a language based on *JavaScript* created for the design of user interface focused applications *QML* stands for *Qt Meta Language*, Qt creator is programmed in C++ which makes a need for our application to include the PyQt5 libraries into our Qt Creator instance, it was created by Trolltech also the creators of Qt
- **OpenCV:** Is an open source computer vision and machine learning software library built to be one common infrastructure for computer vision applications.
- **TensorFlow:** Tensorflow is an open source end to end platform for machine learning that counts with an integral and flexible ecosystem of tools, libraries and resources coming from the contributions of its community that allows developers and researchers to innovate through the creation of applications using machine learning, it was developed by *Google*.
- **Keras:** Keras is an API (Application Programming Interface) designed as it website claims *for human beings, not machines*, that follows best practices in order to reduce cognitive load minimizing the number of user actions for common use cases when it comes to model training for machine learning applications, it is built over *TensorFlow 2* and comes with an extensive documentation and user guides.
- **NVIDIA CUDA:** *CUDA* is a pararell computing platform and programming model developed by *NVIDIA* for computing on *NVIDIA* GPUs, *CUDA* allows to speed up drastically very demanding computing applications using the power of GPUs, *CUDA* makes the compute intensive portions of programs to be executed in parallel using the thousands of GPU cores available, in terms of processing deep neural networks it comes with the *cuDNN* library.
  - **cuDNN:** A GPU-accelerated library of primitives for deep neural networks that provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization and activation layers.
- **Trello:** A web software for project organization that employs a Kanban system to keep track of activities regarding a project.
- **Adobe Illustrator:** A very powerful tool for graphic design provided by *Adobe* which was used on this project in order to create the logo seen in Figure 1.5.
- **Git:** A software for tracking changes in any set of files, used for the version control while developing the project.
- **GitMind:** A free application for professional mind mapping this application was used in the creation of the design diagrams.

4.2.2. Hardware

This section depicts the hardware tools involved in the development of this project, originally cameras and other multimedia devices were tought to be used but testing and sample gathering was run trough simulations by collecting internet videos, so at the end, only a home pc was involved.

A home pc running on Windows 10 with the following specifications was used in the development.

- **Operative System:** Windows 10 Pro (21H1).
- **CPU:** Intel(R) Core(TM) i5-9600k (3.7 GHz, 6 cores).
- **GPU:** NVIDIA GeForce RTX 2060.
- **Motherboard:** Gigabyte GA-H310M S2H
- **RAM:** 16 Gb.

A *Logitech* webcam was used in order to test real-time processing but this feature was discarded due to time restrictions and will appear in the improvements section at the end of this document.

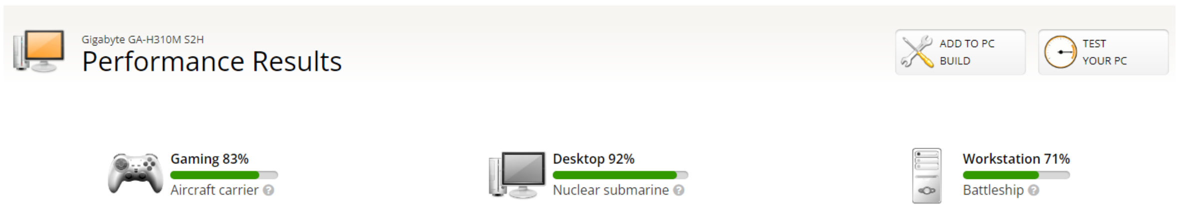


Figure 4.3: The system used for developing the project UserBenchmark scores

Figure 4.3 shows the score of the system employed for developing the project, it obtained high results which its very important due to the high performance required by the project on early stages due to the lack of optimization, having a less powerful system could have led to frustrating development stages due to screen freezes and system failing in order to process the required outputs, for example, in order to predict emotions through the model, the system requires a high performance GPU as it is processing a high volume of video frames.

Having an NVIDIA GeForce RTX 2060 allowed the use of CUDA which had a great impact in terms of improving performance.

4.2.3. Other tools

This section will show tools used that couldn't fit in the previous sections.

- **Kaggle**
  is an online community composed by data scientist and machine learning experts that allows users to find and publish data sets and generate models along these data sets, *Kaggle* also offers working along with other data scientists and engineers and participate in competitions of solving data science challenges against other teams since *2010*.

CHAPTER 5

Project architecture

This chapter will encompass all the development regarding the project's different elements architecture and its relationships between each other, the main goal of this chapter its to give a detailed view on how the elements are set and defined in order to fulfill the system requirements.

The main components of the project are running in different threads that run in parallel each of this threads is in charge of one functionality regarding the system, a general view of the component organization is shown in Figure 5.1:

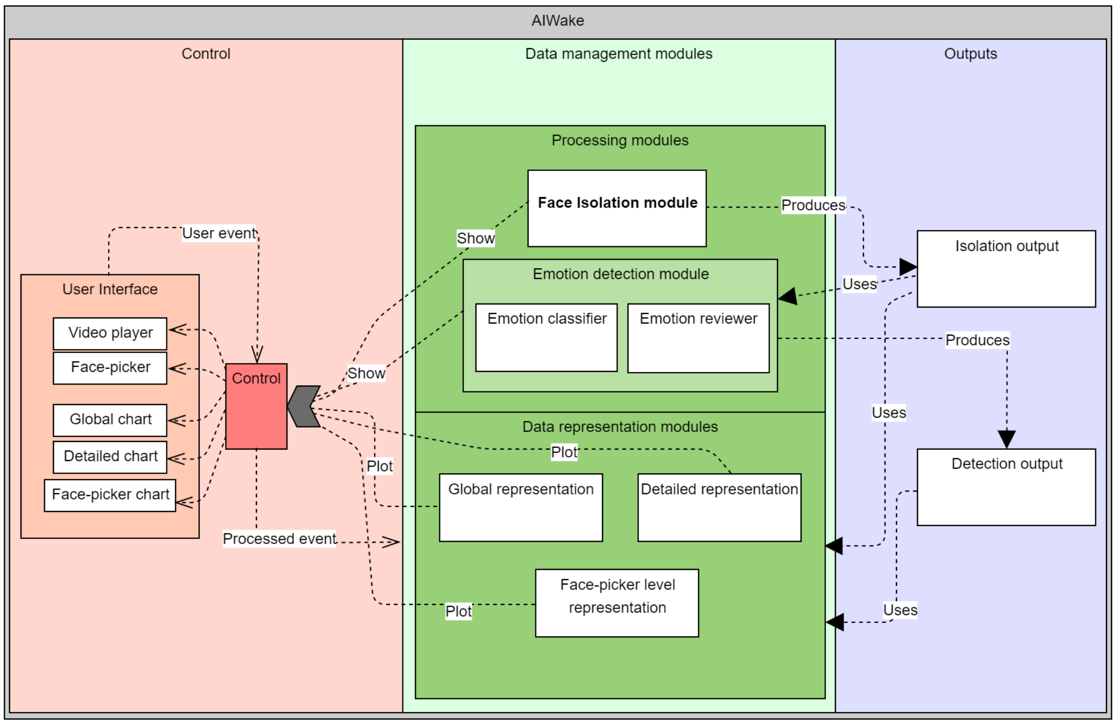


Figure 5.1: Overview of the AIWake system

Figure 5.1 shows a general view of the main components of the system the modules, such as the user interface and the data management modules are running in parallel in different threads, the purpose of each module and a detailed explanation of them will be depicted in this chapter but first lets see a brief summary of the components shown in Figure 5.1

- **Control and UI:** Refers to the main execution, it encompasses the UI mainWindow thread and using the events produced by the interaction of the user with the main window it controls which threads are being executed and what actions are going to be shown through the system, it is the main responsible of having a constant feedback output to the user.

- **Processing:** The core modules of the system data processing, it is in charge of taking the input video source and process it to produce the output files that are going to be processed by each process, it is composed by two processes that refer directly to the development phases mentioned in the methodology.
    - **Face isolation module:** this functionality takes the video source and produces an output json containing data about the found faces, an algorithm for determining which particular face belongs a new face appearing in the video takes place here, a review process takes place after isolating the faces, then, the output json is produced.
    - **Emotion detection module:** by using the model generated in **Phase 0** this module takes the output from the **Face isolating module** and reads it in order to obtain a cropped section of the video source, with this cropped section of the video a new image containing only a particular face its send to the data model in order to predict the emotion of that face in that particular video frame, a review process takes place and then the json output is generated, this output could be used in other programs that use the format or be used again on AIWake to review the data again.
- **Data representation:** Encompasses the data analysis and visualization of the data obtained by the **Processing** threads, the threads running by the purpose of representation of data are constantly running and waiting for new data to plot in the UI, three threads are in charge of this.
    - **Detailed plotter:** It is in charge of representing the data in a detailed way, this means at a frame level, each time a frame of the video is processed the data obtained in that particular frame is plotted in the UI layout containing the chart, this plot is useful when reviewing the data frame by frame and it allows the user to pick between different plots.
    - **General plotter:** It gives a general view of the data that is being processed, this means that the plot is going to plot a cumulative representation of the obtained data, it is useful in order to obtain a general view of the video performance and ruling emotions detected in the presentation attendants.
    - **Face level plotter:** When reviewing the data the UI offers the option of picking a particular face, this fact makes mandatory for the user to be able to review the emotions detected in a particular selected face, it is useful when there is a need to know what a particular attendant was doing when for example, in frame 436 he got angry without a logical explanation (it could be a classification misfire or maybe the attendant had a reason to be angry).

Notice that there is only one thread in charge of UI components, this is due to the fact that in terms of design, AIWake has a UI that is composed by only the main window, the design aims for simplicity using grouping of related options in different sections inside the main window, this reduces the feeling of the system being complex to use by the final user, much applications nowadays make the mistake of overloading its main window with a lot of popups, external windows and options which makes these applications to feel more complex than it really are.

Another important consideration is the fact that the **Face isolation module** and the **Emotion detection module** are not parallel executed, this is due to the fact that the **Emotion detection module** depends on the **Face isolation module** output json this design decision was taken in order to give the system the option of start the execution of the **Emotion detection module** directly if a video source and a json with the valid format are given, which allows the user to continue the process in different periods of time that doesn't need to be continuous but clashes directly with the option of giving the system a continuous video source like a webcam or a camera for real-time emotion detection processing an improvement that could come in the future.

5.1. CONTROL AND USER INTERFACE

In this section we will have a detailed explanation of the components implementation regarding the user interface and control of the threads involved in the system, as said before, the mainWindow thread is the parent for all the threads running the processes of the system, this made mandatory the control of these threads using signals and different control methods.

Both aspects, control and user interface will be explained separately in this section, other aspects of the system like the data representation charts (that also belongs to the UI) are separated in different threads in order to prevent the mainWindow from freezing as these processes are managing a large amount of data and plotting this data through variables that are in a dependency relationship with the components of the main window (the plotters layouts).

5.1.1. User Interface

The design choices for the user interface will be explained here, as mentioned before, as an information technology student i considered the need of an usable system as mandatory so **usability principles** were taken into account for the user interface design, the concept of usability refers to the facility of a user to interact with an system in an easy and comfortable way allowing and helping the user to achieve its goal using the system, this concept is directly related to level of satisfaction that the system will produce on the user after using the system, the higher the usability the higher the level of satisfaction.

Also **Gestalt principles** for UI design [17] were applied in the design of the AIWake UI, these principles are part of a psychological current that appeared in Germany around *1920* and are directly related to the study of how the human brain works when perceiving objects, the system of these principles along with the use of **usability** principles guided the UI design.

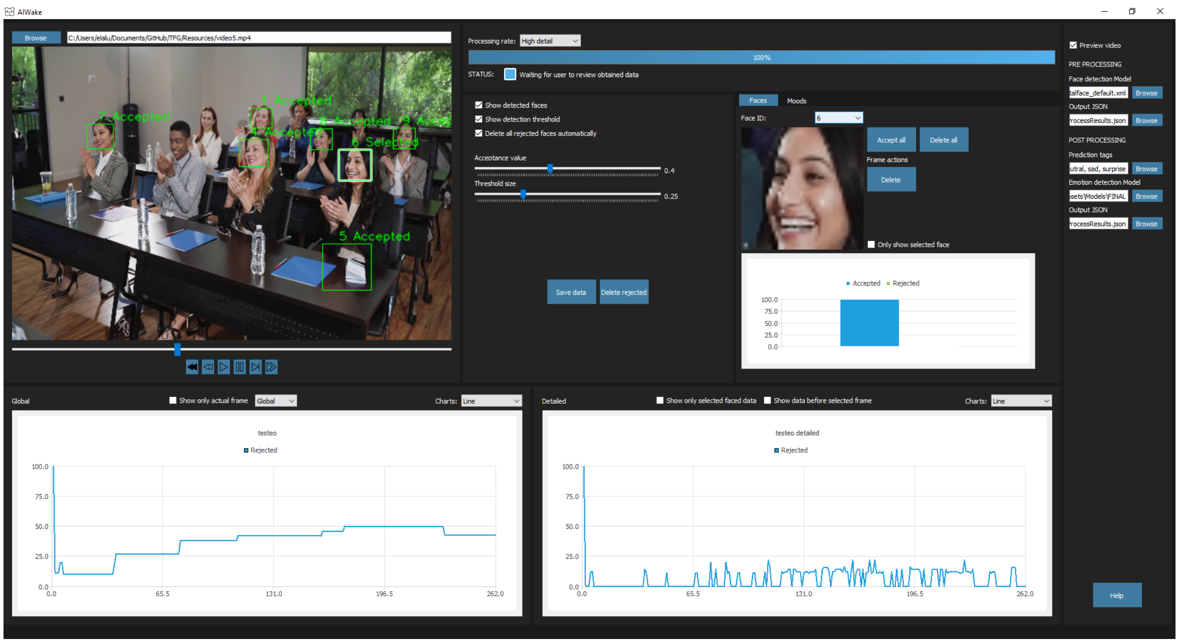


Figure 5.2: AIWake UI during the review phase of the face isolation process.

In Figure 5.2 we can see the AIWake UI, how the mentioned principles were implemented and different designs for the UI will be shown in this section along with some code snippets regarding the user events controls.

Now lets describe how the principles mentioned for UI design were applied, lets start with the **usability**, usability principles are mostly applied in wed design but it can be applied also in desktop applications (web applications are just desktop applications that are able to run over the browser).

The list of **usability** principles applied is the following:

- **Make the system state visible for the user:** This principles refers directly to the action of giving feedback to the user, means like progress bars, status texts and live charts were implemented in the design of the user interface.
- **Give the user control and freedom:** This principle refers to the action of giving the user a big set of options to control the process, AIWake UI gives the user options to change the data model used for processing, manage options regarding the face isolation algorithm (like aceptance levels and accuracy) and also allows the user to correct errors regarding the classification of faces, AIWake was also designed having one important consideration in mind, the fact that as a desktop system not all systems are the same so the processing rate can be modified allowing the system to run in less powerful systems.
- **Consistency:** Users have an internalized set of standards regarding usability such as the actions some buttons will perform after clicking them, this consistency principle is mostly applied in the video player, where the design is based in a common video player like windows media player or VLC would look which are popular video players used nowadays.
- **Recognizing before remembering:** This refers to the structure of the UI, AIWake UI was designed having in mind the distribution of the elements and how the user will perceive them, for example, the options pane regarding the face isolation phase is near the video player as this is the first element the user will need to interact with (user has to provide a video source and press play), the video player is in the upper left corner due to the fact that according to some studies about user interaction upper left corner is the first area of an system the users tend to pay attention to (this is know as the **F pattern**), notice how the first button the user should use is in that area's corner, next the *face picker* situated in the right side immediately updates when starting the reviewing process which changes the focus of the user (which is in the video player) directly to the main tool they have for reviewing the data, the charts are real-time updated and occupy the bottom section of the UI as they are core elements when reviewing the obtained data and giving feedback to the user.
- **Flexibility and use efficiency:** In terms of flexibility, this principle refers to the wide range of users that could use the system, the goal is to make the system adaptable to this wide range of users, AIWake UI was designed with only the necessary number of buttons each one makes the action it represents (video player uses the usual set of icons while system's specify directly the action the button will perform), the user interface is also fully responsive not only at main window level but at section level, this allows the user to modify the overlay of the window and also hide elements or even maximize sections that the user is interested in like the chart views.
- **Aesthetic and minimalist design:** One goal of the design of AIWake UI was to provide the system with a custom design, using QML (the programming language for Qt UI design) i was able to give the system an unique look regarding the main elements, here it is a list of the main characteristics of the visual design the elements of the UI has:
  - **Progress bar:** no border, white color for texts and #7A7A7A as the background color (gray with some light), when the progress bar is showing progress it will be filled with a gradient of blue going from #3F7BA1 to #51B1ED.
  - **Buttons:** no border, max width of 80px and max height of 40px, the background color when not pressed is #3F7BA1 and the text color is white, when pressed the button's background color changes to #51B1ED.
  - **Frames (sections containing grouped actions):** no border, background color #232323.
  - **Frames (sections containing empty spaces):** no border, background color #C0C8CF.
  - **Tab panes (Overall):** no border, background color #232323.

- **Tab panes (tabs):** no border, the selected tab looks bigger than the non selected one by giving the non selected tab a 2px margin-top, selected tab background color is #43A5FA and the non selected tab has a background color of #1A1A1A, when hovering over any tab the background color will change to #3F7BA1.
- **Labels and check boxes:** all text with white color.
- **Main window:** background color of #1A1A1A.
- **Sections separators bars:** bars between sections that allow the user to change the sections size, its color will be always the same as the main window background color in order to maintain consistency.

As can be seen in the list regarding the visual elements style guide the main goal is to achieve consistency through the use of a color palette consisting on a set of gray-blue tones and the use of white texts.

The chosen color palette used in the system user interface employs the above mentioned colors Figure 5.3 shows the final palette used in the user interface.

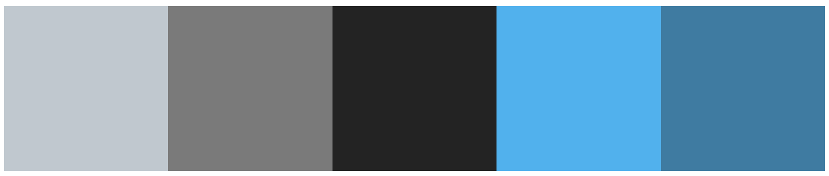


Figure 5.3: Color palette for AIWake in order: #C0C8CF, #7A7A7A, #232323, #51B1ED and #3F7BA1

In terms of being minimalist, the less buttons the better and also the use of reduced number of colors for the palette help achieving this goal.

- **Help and documentation:** this principle is applied through the use of a user manual and the "Help" button in the bottom-right corner which prompts the user with guides to obtain the manual, also, the process run through the system is "guided" by preventing the user of committing mistakes through the execution controlling which sections of the system can be used depending on the process that is running in that particular moment.

Now lets take a look on how the **Gestalt** principles were applied in the design choices of making the UI for our system.

Gestalt most recognized principles are as follows: *principle of closure, principle of common region, principle of multi-stability* and *principle of proximity (emergence)* the way such principles are applied in this system is the following:

- **Multi-stability:** Humans tend to look for stable solid and stable items, this principle is mainly applied for giving the elements of the user interface high contrast, and its applied through the use of a dark background and light white text.

  When this principle is well applied it helps lessen the cognitive load of the users and also help with guiding them through the system tasks in order to complete the process.

- **Closure:** refers to the fact that humans prefer complete shapes when perceiving elements, this principle is applied through the shaping of buttons and the shapes that the different sections of the system use, lets see an screenshot of a section:

  Figure 5.4 shows the face-picker utility section, the **closure principle** is applied by means of using a darker color in the gaps between different section, notice the use of #232323 (dark gray) to make a square figure around the utility buttons, charts, image and combobox, this allows the utility to be distinguish by the user at first sight as a whole element.

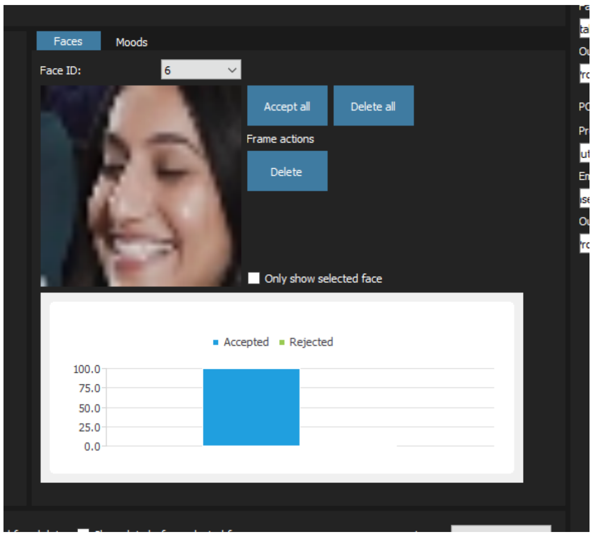


Figure 5.4: Section of the face-picker utility

- **Common region:** this principle stands for having related elements in the same closed area, we could think that Figure 5.4 is also applying this principle and in fact it is correct because the squared section enclosures related elements (those that belong to the face-picker utility) but the principle of common region can be applied in many ways such as the way the video controls are displayed (Figure 5.5) or even the two sections that contains the detailed and general chart views.

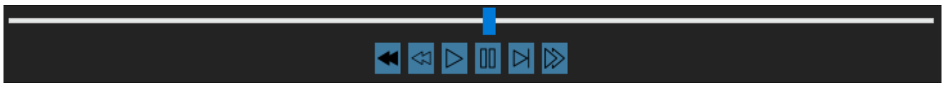
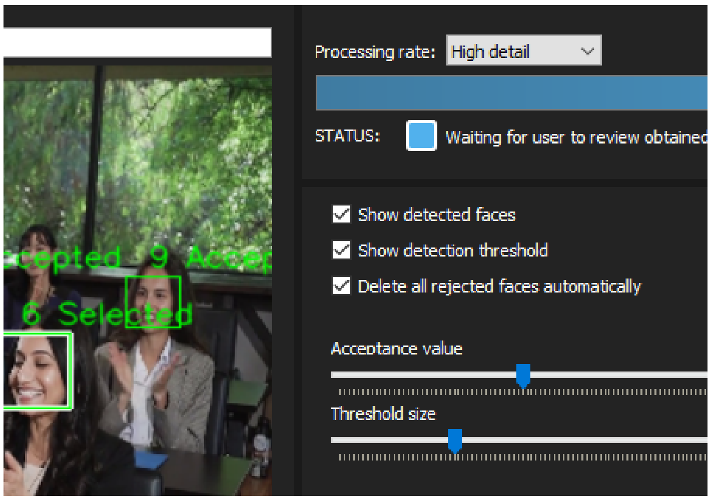


Figure 5.5: Video controls for AIWake

- **Proximity (Emergence):** emergence is easy applied when we take in consideration what elements must be considered as one, for example, all the elements regarding video progress, video options and video player are together and only the gap between them for the **closure** principle to be applied with different sections of the system is what separates them, this helps the user recognize that area of the system as the video controls.

  Lets take a look at the mentioned above area:



Processing rate: High detail

STATUS: Waiting for user to review obtained

- [x] Show detected faces
- [x] Show detection threshold
- [x] Delete all rejected faces automatically

Acceptance value

Threshold size

Figure 5.6: Video area (little section of the overall video area)

Figure 5.6 shows a cropped section of the system, this figure show how the elements regarding feedback about the video being processed, processing rate, detection options and the video player are together, the **proximity** principle is applied in an horizontal way for the video utilities and its applied also for the data visualization area that is in the bottom of the user interface.

- **UI designs**
  The design of the user interface went through four phases, lets take a look at the evolution of the user interface through the development:

  1. **Simplistic approach (Figure 5.7), *27-01-2022*:** this was the first idea in terms of design, the system was though as a main tab bar were each tab would contain each main module of the process, first tab being the face isolation module, second tab would contain the emotion detection module, a third tab containing the data analysis module with the chart views and the last tab for the general options pane.

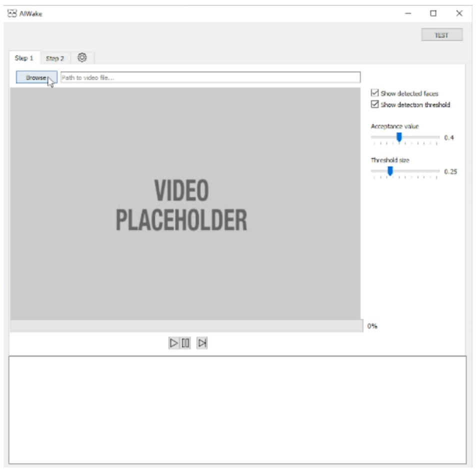


Figure 5.7: First design of the AIWake UI

     This design turned out to be very complex in terms of navigation through the system and a big problem for having a minimalist system, also, it had big problems in terms of making a responsive system.

  2. **First attempts following UI design principles (Figure 5.8), *01-04-2022*:** the first attempts on getting an system that followed the mentioned design principles, it is similar to the final UI and was the core idea for the distribution of the elements in the system but the use of a tab bar for navigation in order to get to the general options of the system and the fact that there is an obvious problem regarding spatial distribution (which is a problem when making a responsive system) resulted on a little redesign.

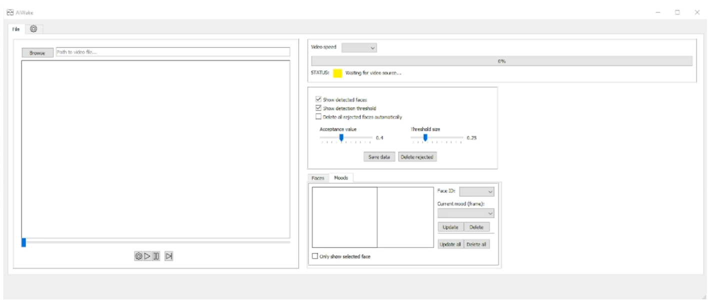


Figure 5.8: Second design of the AIWake UI

  3. **The mock-up (Figure 5.9), *27-04-2022*:** The main approaches of this design were to eliminate the tab pane navigation and to distribute space used in the system there wasn't

a real implementation of this mock-up but it allowed me to confirm which was the best way to distribute the elements in order to make use of the full space the system could use.

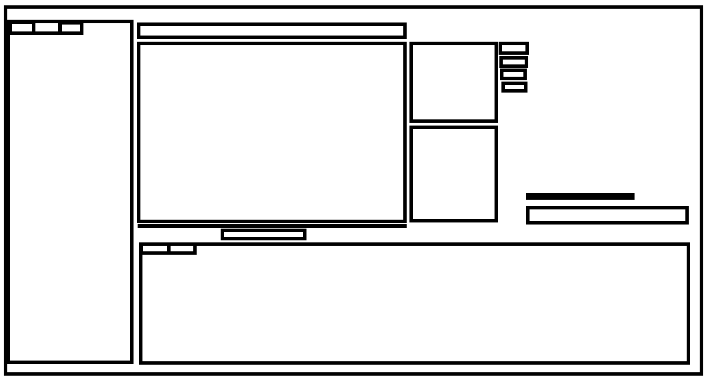


Figure 5.9: A very simple mock-up regarding the spatial distribution of the system

It was a very fast design process in order to allow me to portray the main idea behind the distribution i wanted for the system elements.

4. **Final UI (Figure 5.10), *05-05-2022*:** using the mock-up main ideas and the state the user interface was along with some modifications turned out in the final system interface:

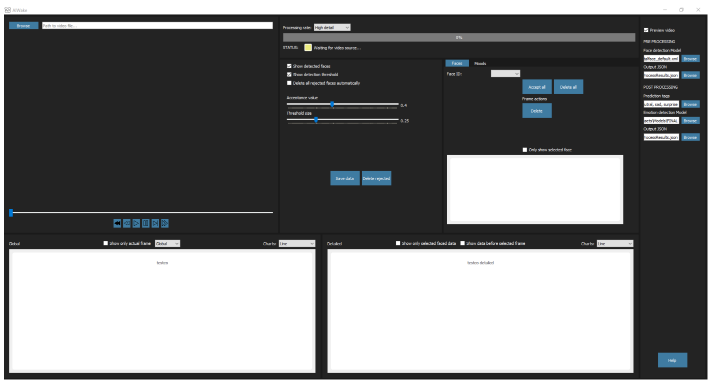

Processing rate: High detail
STATUS: Waiting for video source...
Acceptance value 0.4
Threshold size 0.25
Face detection Model: alface_default.xml
Output JSON: rocessResults.json
Prediction tags: utral, sad, surprise
Emotion detection Model: sets\Models\FINAL
Output JSON: rocessResults.json
Charts: Line

Figure 5.10: Final design of the system

This design modified a little the distribution of the mock-up but followed its principles, the tab navigation was deleted with only one tab panel remaining, the face-picker utility but following the **common region** principle it made no sense to me to split the face-picker section in two and it remained in the same area enclosed by the same square but separated with a tab that is auto-selected when the emotion detection process starts.

- **The .ui file containing the design elements**
  (AIWake_app.ui) is located in the project folder and contains all the scheme of the user interface along with the code, this .ui file extension belongs to **QtCreator** files for ui windows and saves all the user interface configuration in **XML** language including a section for the style of the elements of the system, the file contains over 2000 lines of code regarding the user interface.

5.1.2. Control

This section will depict how the main thread that belongs to the mainWindow process controls the threads executing the processes running during the execution of the system.

The main thread code is located in the *AIWakeUI.py* file, this file contains all the necessary calls and control structures to manage the communication between threads, using **PyQtSignal** library as our main control manager in order for the main thread to receive signals and decide what to do according to the data received from that signals.

lets have a look at the *__init__* function of the main thread which is called from the *main.py* program and its in charge of instantiating all elements regarding user events control, signals control and threads control.

Listing 5.1: init funtion (*AIWakeUI.py*)

```
1    def __init__(self):
2        super(AIWake_UI, self).__init__()
3        uic.loadUi("AIWake_app.ui",self)
4        self.setWindowTitle("AIWake")
5        self.thread = {}
6        self.currentThread = []
7        self.initUI()
```

The control structure for threads is simple, *thread =* is a python dictionary that will contain the instance of each running thread of the system to begin its execution by calling the *start()* function of that thread instance or to change some values managed by a particular thread, the next control structure is *currentThread = []* a python list that contains a set of boolean values that determine which thread should be running at that time.

About the last function call of the init function *initUI()* it maps all the elements of the system with its corresponding events and links them to a function that will perform an action regarding the event, the way **Qt** manages this events is by the use of already implemented signals in the object that our user interface element is inheriting such as *clicked()* signal in the case of buttons or the *release()* signal in the case of sliders, such mapping done by the initUI function is a vast list of code lines, we will take a look at some of these lines for the shake of simplicity.

Listing 5.2: Mapping of user interface events to their respective functions (*AIWakeUI.py*)

```
1    self.playBt.clicked.connect(self.playBtClick)
2    self.showBoxesCb.stateChanged
3        .connect(self.previewChange_showHB)
4    ...
5    self.VideoSpeed.currentIndexChanged
6        .connect(self.setProcessingSpeed)
7    self.VideoSpeed.setCurrentText("High␣detail")
8    ...
9    self.frameSelector.sliderPressed.connect(self.sliderPressed)
10   self.frameSelector.valueChanged.connect(self.sliderMoved)
11   self.frameSelector.sliderReleased.connect(self.sliderReleased)
```

The code shown in Listing 5.2 belong to the *initUI* function and performs the following actions:

- ***Line 1* clicked() signal:** Triggered when a user clicks on a button, in this case the mapping belongs to the play button of the video player, clicking that button would trigger the *playBtClick()* function which starts all the processing of the video or in case that the thread in charge of that functionality (Thread 1, Face isolation or Thread 2, Emotion detection) are already running and in the review phase it would resume the pause event (in case the video is paused).
- ***Line 2* stateChanged() signal:** this signal belongs to the *QCheckBox* component and it is triggered when the state of the check box changes (if it is selected or not) it brings an integer argument to determine which state the check box has, for this example, if the face isolation thread is running it shows the boxes that are drawn in the frame to represent the cropped area of the detected face or hides the boxes depending on its state.
- ***Line 4* currentIndexChanged() signal:** it belongs to the *QComboBox* component and gets triggered when the selected option of the combo box changes, this signal returns an integer which represents the index of the selected element, more values like, the text value of the selected index can be modified or read through getter and setter functions like in **Line 5** where the text value for the selected index (default index in this case) in the VideoSpeed combo box

is changed to "High detail", for this particular case, the signal is mapped to a function that changes the processing rate option for the video processing (allowing the processing to skip some frames in an exchange for level of detail in the charts).

- ***Line 7*** **sliderPressed() signal:** the slider component of the user interface is present in the video player and in the options section regarding the face isolation phase, these sliders come with this signal which is triggered when the knob of the slider is pressed by the user, for this case, it controls the video player frame selector slider and pauses the video player in order to allow the user to pick a new frame of the video to play from.
- ***Line 8*** **valueChanged() signal:** this signal is common in some elements, in this particular case it is being used for the slider that controls the selected frame, but it does the same and gets triggered always in the same way as it name says, when the current value of a selector changes, for this case we are mapping the slider value change event (when we move the slider to right or left sides or the video bar) to the *sliderMoved()* function which tells the threads in charge of showing video being played to change the current frame the video is showing, combined with the *sliderPressed* signal the system has the ability to pause the player and show the frame selected by the slider instantly.
- ***Line 9*** **sliderReleased() signal:** regarding the slider, the system makes use of another signal of this component, the *sliderReleased()* signal, which triggers when the knob of the slider is released by the user and in this particular case it is used when reviewing the data after the emotion processing phase to set current frame for an action that i will explain later.

This are the total set of predefined signals that i made use in the system, *Qt* offers the possibility of configuring our own signals for thread control, the set of these own signals will be shown and briefly explained below and explained in detail in the sections regarding the architecture of the different threads of the system.

The *Qt PyQtSignal* library offers a very easy way to implement signals, so as said, lets explain the set of signals that were created in order to control the threads of the system.

- **Face Isolation thread:** the *FaceIsolator* module contains a set of signals for video preview, management of the face picker utility, feedback and more.
    - **changePixmap(QImage, int) signal:** this signal is used to notify the main thread when a new frame is processed in order to update the video player with the processed frame and cropping boxes in the faces, it sends to the main thread a QImage object that is the required object by the layout that shows the processed images to properly show these images, the int value is an integer that holds the percentage of the video that has been processed by this thread in order to be shown in the progress bar.
    - **setPicker(list) signal:** used to send a list containing the id's of the processed faces to the main thread, this list is processed by the main thread in order to establish the options that the face-picker utility holds in terms of available faces to select for review.
    - **changePixmap_pick(QImage, list) signal:** acts in a similar way than the *changePixmap()* signal but the QImage object is the cropped section of a detected face in order to show it in the face-picker utility, the list argument contains strings with data about the face.
    - **updateStatus(list, int) signal:** as its name says this signal is use to update the status feedback text and the status color, the list contains the strings that hold the status text and the integer argument represents a number which is used for determining which color will show the status, red for error, yellow for wait and blue for processing.
    - **updateFrameSelector(int) signal:** it is used for positioning the slider bar knob in the corresponding position of the frame that is being processed, the integer argument represents the actual frame value.

   - **preprocessDone() signal:** this signal notifies the main thread that the face isolation phase was completed in order for the main thread to begin with the execution of the emotion processing phase, this signal is emitted when the user saves the reviewed data and the json file is generated.
   - **changeSelectedFrame(int) signal:** is triggered when during the reviewing process, the user selects a different frame, the integer argument holds the frame selected by the user.
   - **updatePlotter(list, list) signal:** in order to plot the data that is processed in real time, this signal sends two lists, one contains the information about the cumulative detections that are being processed and the other one holds the label values for the chart, in this case accepted and rejected faces that were isolated during the execution of this thread.
   - **updatePlotterDetailed(dict) signal:** similar to the *updatePlotter()* signal but this one holds the values of the frame that is being processed, it uses a dict for simplicity.
   - **sendDataToPicker(dict) signal:** this signal is used to send a dict structure to the face-picker utility in order for the picker to plot the data regarding the face that is selected in that moment only it is triggered when the selected face value changes.
   - **setFirstFace() signal:** in order to direct the user attention to the face-picker utility when the face isolation process is over, this signal is triggered at the end of the processing and makes the selected face value to automatically change to the first face id in the face-picker combo box and updates the image of the face-picker along with the chart.

- **Emotion detection thread:** this thread is in charge of the emotion detection aspect of the system, the signals that are triggered in this thread are five in total and act in a similar way than in the face isolation thread, these five signals are:
   - **postPbValue(int) signal:** updates the value of the progress bar during the emotion detection process, the integer represents the percentage of frames that have been processed.
   - **done() signal:** similar than the *preprocessDone()* signal in the face isolation thread with the difference that this process ends automatically and doesn't require the user to click any buttom, after this signal is over a post-review process begins that runs in another thread, this decision of separating the emotion processing thread from its review process was made due to tha fact that the emotion detection is a very performance and RAM memory demanding.
   - **updateStatus(list, int) signal:** works in the same way it does in the Face isolation thread.
   - **updatePlotter(dict) signal:** works similar to the face isolation thread signal but it sends a dictionary to the main thread, this is because the fact that labels for emotion detection could change if a user decides to try a new model, this labels are stored in the configuration file as a dictionary that already contains the labels and their values.
   - **updatePlotterDetailed(dict) signal:** same as face detection thread signal.
- **Emotion detection review thread:** the high requirements of the emotion detection thread came with the idea of separating both processes (processing and reviewing) this thread has the same signals of the face isolation thread visual feedback aspects but with some differences.
   - **changePixmap_preview(QImage, int) signal:** works the same way than the *changePixmap()* signal of face isolation thread.
   - **updateStatus_preview(list, int) signal:** used for updating the status feedback, the same as in the already shown threads.
   - **changePixmap_pick(QImage, list) signal:** same as the face isolation thread one with the difference that the face-picker is now allocated in the emotion detection tab (mood tab).

- **updateFrameSelector(int) signal:** updates the slider knob position to the position of the current frame that is being shown in the preview video similar to the face isolation thread *updateFrameSelector()* signal.
- **updateStatus(list, int) signal:** same signal as the one in face isolation thread.
- **setPickerMood(list) signal:** new addition to the signals set, this signal sends to the face-picker utility the emotion that was detected over the selected face in the last reviewed frame that face appeared, it allows the user to select a different emotion in case of misfires from the emotion detection model when predicting that face's emotion.
- **updatePlotter(list, list) signal:** this signal sends the same data as the *updatePlotter()* signal in face isolation thread but it is only used to send the detailed plotter the data regarding the selected frame.

▪ **Data representation threads:** this threads doesn't use any signal, this is because the nature of these threads consists on receiving data from the main thread and these are constantly running in the background during the whole workflow of the system, the data representation threads have access to shared variables containing general data of the processing of the system so they are constantly feeding from this data and waiting for new data to be processed.

In terms of shared variables the file *variables.py* contains a set of variables and values that are modified by different events, the threads of the system make use of this variables in some occasions, for controlling the access to this variables the list *currentThread* acts as a thread-pool initialized in the main thread that controls which threads can access the variables using this list as a **lock mechanism**, as the execution of threads that may write in this variables follows a sequential order there aren't much occasions when a **race condition** can happen, but this control is necessary, for the case of the data representation threads there is no chance a race condition could happen as these threads only read the data from this variables but never write over them.

5.2. PROCESSING

This section will provide a deep explanation on how the main modules in terms of data processing work in order to produce each of their outputs.

5.2.1. Face isolation module

This module is in charge of the first processing step of the video, it takes the video source as input and along with the use of OpenCV it sends the frames to a data model provided by the OpenCV libraries *haarcascade* model, it also uses some parameters such as the processing rate, the threshold value, the acceptance value and some more parameters regarding visualization as inputs, some parameters can be modified in real-time while others like the video source can't.

Figure 5.11 provides a general view of the components involved on this module and their relationships, lets give a detailed explanation of each module appearing in that diagram, some components shown are shared by the emotion processing module such as the jsonManager, variables and AI-WakeUI association relationship, also notice that the *AIWakeUI* acts as a controller for all the execution and user events, in this diagram, the functions only related to the face isolation module are shown, lets take a look at the *FaceIsolator(Qthread)* module:

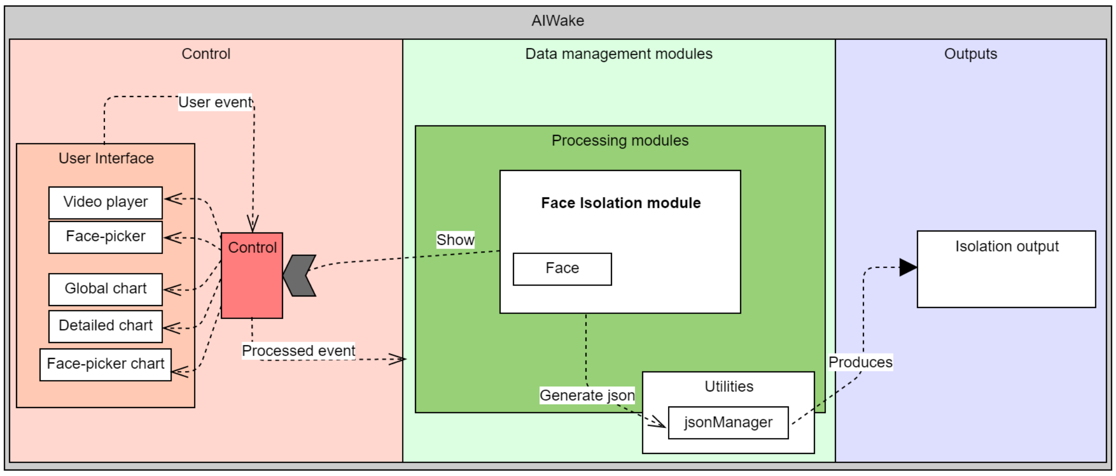


Figure 5.11: components involved in the face isolation module

- **FaceIsolator(Qthread):** this module contains all the main functionality of the face isolation module, it is supported by the implementation of a module that holds an structure for storing the isolated faces data, the *face* module.

  Regarding the variables used on this module, much of them are named on a self explanatory way being for example the pathToVideo variable the variable that holds the file-system path to the source video provided by the user stored on a string as its name says but some of these variables need a little explanation:

  - **previewData:** it holds the isolated faces values before these values are stored in json format, this allows a faster deletion and modification of isolated faces data so instead of storing the data all the time in the json file it is stored only when the user confirms the review (there are some cases where the json file may be updated before saving it definitely).
  - **list:** a temp list that holds isolated faces, this list mostly works when the source video is being processed, the previous list *previewData* holds the values and its used when the user is reviewing the data, *list* variable holds the values during processing and then its used as a temporal variable to manage data between dicts.
  - **showOnlySelected, deleteAuto:** this variables hold the values of the options, user events modify these values through the main thread which changes the values of these variables directly.
  - **finished, selecting, done:** status variables, used to control the process and determine the stage the user is viewing, these values are modified in the main thread and send through signals.
  - **forward, forwardToProcessed, backward and backwardToProcessed:** these variables are modified when the user clicks the video controls via the main thread, these are used to tell the OpenCV *cap.read* function which frame of the video is about to be shown to the user, the *ToProcessed* stands for those cases were the user selects a processing rate different than frame-by-frame processing, clicking backward in these cases will show the next frame going backwards but this frame may be an skipped frame so the *ToProcessed* variable tells the module to move to the last frame that was processed backwards or forwards (depends on the button)

  Now lets talk about the **functions** that are implemented in this module, some of the explanations will include code snippets along with the explanation about the code itself.

- **getFaceIds(list, String, boolean):** it returns a list containing the ids for the isolated faces that are being processed, it is used for updating this list in a dynamic way during the processing and is also called when the user deletes a isolated face, the list argument contains all the isolated faces in order to extract the ids determined by the algorithm, the String argument holds the id of a selected face in case an addition to the ids list occurs (when a new face appears) and the boolean argument tells the function if the action that called this function was a deletion or an addition.

Listing 5.3: getFaceIds() code snippet

```python
def getFaceIds(self, faceList, k, delete):
    idList = []
    if delete:
        for k, v in faceList.items():
            idList.append(k)
    else:
        idList.append(k)
    return idList
```

- **setStatusText(String):** builds a list that will contain the text string to send to the status label, this function is called as an input for the *updateStatus* signal and returns a list containing the text to be displayed in order to provide feedback to the user.
- **deleteFace():** this function removes the face item regarding the *SELECTED_FACE* variable in the variables module from the list that holds the isolated faces and updates the json file, its interesting to take a look at the code and see how the different mentioned functions are called:

Listing 5.4: deleteFace() code snippet

```python
def deleteFace(self):
    self.updateStatus.emit(self.setStatusText
        ("Generating␣preview␣data..."), 1)
    deleted = self.list.pop(str(config.SELECTED_FACE))
    del self.previewData[str(config.SELECTED_FACE)]
    self.js.setDataSerialized(self.previewData)
    self.js.saveJson(config.PATH_TO_JSON_TEMP)
    self.setPicker.emit(self.getFaceIds(self.list, ↩
        ↪ "0", True))
    self.updateStatus.emit(self.setStatusText("Waiting␣↩
        ↪ for␣user␣to␣review␣obtained␣data"), 1)
```

Notice how the *updateStatus* signal is used at the start and end of the process in order to give feedback to the user as the function updates the output json and this process takes very little time but enough to be noticed, it also calls the *setPicker* signal in order to dynamically update the list of available faces of the face-picker utility.

- **drawFaces(String, boolean, Integer, Integer, Integer, Integer, Integer, boolean):** the main goal of this function is to draw a rectangle over the coordinates were a face was detected during the processing of a given frame, the Integer arguments are variables that hold the coordinates x, y and the width and height values of the detected face, the String value refers to the id of the face that is being drawn, two boolean values to hold the accepted value of the face which mean if according to the acceptance ratio determined by the user (by default *40%*) the face was accepted as a valid isolated face which means that the rectangle would be drawn in green color or if it was rejected it will be drawn in red color and the other boolean determines if there is a selected face in order to highlight the rectangle regarding that face (to allow the user to easily find the selected face in the video preview), this function uses the cv2 utilities *rectangle* and *putText* in order to generate draw these figures over the processed frame.

- **deleteAllRejected():** works the same way as the *deleteFace()* function with the difference of iterating through all the list of faces and automatically deleting all the faces that have been rejected.
- **deleteFaceFrame():** this function is in charge of deleting an appearance of a detected face which due to any reason was misfired by the algorithm, there are some cases were the algorithm detects a face with an error and doesn't get the correct coordinates for that face which leads to a cropped image of the area near the face that was isolated that doesn't give enough information about the characteristics to be processed by the emotion detection module, lets see a code snipped of this function:

  Listing 5.5: deleteFaceFrame() code snippet

```
def deleteFaceFrame(self):
    if str(config.SELECTED_FACE) in self.previewData:
        if str(config.SELECTED_FRAME) in self.previewData[str(config.SELECTED_FACE)]:
            del self.previewData
                [str(config.SELECTED_FACE)]
                [str(config.SELECTED_FRAME)]
            self.js.setDataSerialized(self.previewData)
            self.js.saveJson(config.PATH_TO_JSON_TEMP)
```

  Notice the use of *previewData* variable as this function belong to the review stage of this module, notice also the use of the variables *SELECTED_FACE* and *SELECTED_FRAME* to retrieve the data needed to perform the deletion, all this variables are directly modified by the main thread *AIWakeUI*.
- **getNearestProcessed(Integer):** this function returns the frame that contains the nearest processed values, in other words, it looks over the *previewData* variable to obtain the next frame that holds any processed information about the faces, this is useful if there are frames that weren't processed or frames that were skipped due to the use of a processing rate different from the default *frame-by-frame/high detail* one, this function works backwards or forwards depending on the *forwardToProcessed* and *backwardToProcessed* variables values, lets see the case of *forwardToProcessed* value being *True*:

  Listing 5.6: code snippet for getting the nearest processed frame (forward)

```
def getNearestProcessed(self, iterations):
    aux_iterations = iterations
    founds = []
        if self.forwardToProcessed:
            for k in self.previewData:
                while True:
                    aux_iterations += 1
                    str_iterations = str(aux_iterations)
                    if aux_iterations > config.VIDEO_LENGHT:
                        aux_iterations = iterations
                        break
                    if str_iterations in self.previewData[k]:
                        founds.append(aux_iterations)
                        aux_iterations = iterations
                        break
            if len(founds) == 0:
                return -1
            return min(founds)
```

- **getAcceptedAndRejectedNum():** returns a list containing the number of valid isolated faces and non valid faces, this function is called when an iteration of the processing is completed in order to emit a signal that will send this data to the data representation threads.

- **run():** the function that holds the main execution of this module, this function is overwriting the *run()* function of the *QThread* module that this module is inheriting from, it is always called by calling the *start()* function over the instance of a *QThread* object, in this case the *start()* function is called when the user clicks the play button after browsing a video source, this function is complex and has a lot of code, so snippets of it will be shown starting with:

Listing 5.7: Snippet of the code for the isolate algorithm

```python
for (x, y, w, h) in faces:
    newFace = DS.face(x, y, w, h, iterations)
    newFound = True
    if len(self.list) == 0:
        founds += 1
        self.list[str(founds)] = newFace
        newFace.queue(newFace, None)
        self.setPicker.emit(self.getFaceIds(self.list ,str(founds), False))
        frame_stats["Rejected"] += 1
    else:
        for k, v in self.list.items():
            if newFace.equal(config.PROP, self.list[k], frame):
                newFace.occurs += self.list[k].occurs + 1
                if newFace.occurs >= int(iterations_ratio*config.RATIO):
                    newFace.valid = True
                    frame_stats["Accepted"] += 1
                else:
                    newFace.valid = False
                    frame_stats["Rejected"] += 1
                newFace.queue(newFace, self.list[k].list)
                self.list[k] = newFace
                newFound = False
                break

    if newFound and len(self.list) > 0:
        founds += 1
        newFace.queue(newFace, None)
        self.list[str(founds)] = newFace
        self.setPicker.emit(self.getFaceIds(self.list, str(founds), False))
        frame_stats["Rejected"] += 1
```

Notice the use of the temporal list *list* during the data processing and the use of the *face* in order to store the data in an structured way.

Lets take a look at the review stage of this module:

Listing 5.8: Snippet of the code for data review in the isolation stage

```python
ret, frame = cap.read()
frame = cv2.resize(frame, (540, 380), fx=0, fy=0, interpolation=cv2.INTER_CUBIC)

for k in self.previewData:
    if str(config.SELECTED_FRAME) in self.previewData[k]:

        valid =
            self.previewData[k]
            [str(config.SELECTED_FRAME)]["valid"]
        x =
            int(self.previewData[k]
            [str(config.SELECTED_FRAME)]["x"])
        y =
            int(self.previewData[k]
            [str(config.SELECTED_FRAME)]["y"])
        w =
            int(self.previewData[k]
            [str(config.SELECTED_FRAME)]["w"])
        h =
            int(self.previewData[k]
            [str(config.SELECTED_FRAME)]["h"])

        if config.SELECTED_FACE >= 0:
            if k == str(config.SELECTED_FACE):
                if k in self.previewData:
                    cropped = frame[y: (y + h), x: (x ↩
                       ↪ + w)]
                    rgbImage = cv2.cvtColor(cropped, ↩
                       ↪ cv2.COLOR_BGR2RGB)
                    hs, ws, ch = rgbImage.shape
                    bytesPerLine = ch * ws
                    cropped2Qt = QImage(rgbImage.data, ↩
                       ↪ ws, hs, bytesPerLine, ↩
                       ↪ QImage.Format_RGB888)
                    aux_face =
                        self.list
                        [str(config.SELECTED_FACE)]
                    self.changePixmap_pick
                        .emit(cropped2Qt,
                        self.generateFaceInfo(aux_face, ↩
                           ↪ iterations,
                        config.SELECTED_FACE))
                self.drawFaces(k, valid, frame, x, y, ↩
                   ↪ w, h, True)
            elif k != str(config.SELECTED_FACE) and ↩
               ↪ not self.showOnlySelected:
                self.drawFaces(k, valid, frame, x, y, ↩
                   ↪ w, h, False)
        elif not self.showOnlySelected:
            self.drawFaces(k, valid, frame, x, y, w, ↩
               ↪ h, False)

    rgbImage = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    h, w, ch = rgbImage.shape
    bytesPerLine = ch * w
    convertToQt = QImage(rgbImage.data, w, h, ↩
       ↪ bytesPerLine, QImage.Format_RGB888)
    self.changePixmap.emit(convertToQt, 100)
```

The code for the review phase is in charge of drawing the rectangles according to the data contained in the *previewData* dictionary, this variable as told before holds all the data after the processing of the isolation stage ends and provides the system with a set of structured data about the detections of the video, notice also the use of *cv2.resize* function, this is due to the requirements of the model frames sizing and its passed through all the calls referring OpenCV when processing the video, the initial data frames are always 540x380 so the following calls to the video source opening must match this measurements in order to maintain data consistency.

- **jsonManager(object):** this module is contained in the *utilities.py* file as a tool for parsing the json objects and dictionaries, it is in charge of loading the json files containing the data and processing the data into json objects to dump that json objects into files, this module is employed by any module that needs to retrieve data from the output json files or to produce this files, so it is also used in the emotion detection module and the emotion detection review module, this module holds only two variables:

- **__instance:** employed in order to implement a ***singleton pattern*** this pattern is used in order to have one and only one instance of the *jsonManager* module so the files containing the data are opened only one time by the same process avoiding problems like overwriting of the files or having multiple instances of the same file opened at once which could lead to memory problems this pattern usage is not widely implemented on python components but it is employed in some particular cases such as this one, the way to implement the *singleton pattern* is as follows:

  Listing 5.9: The __new__ function returns the instance of the *jsonManager*

  ```
  module jsonManager(object):

      __instance = None
      data = {}

      def __new__(cls):
          if jsonManager.__instance is None:
              jsonManager.__instance = object.__new__(cls)
          return jsonManager.__instance
  ```

  Were the return value of this function is the instance of the *jsonManager*, notice the inheritance of the module from the Python object module in order to use the __new__ function in case there isn't an instance of the *jsonManager* module yet.

- **data:** holds temporary values to be serialized when parsing the generated data in order to generate the json file output resulting from the processing of the different modules of the system.

Now lets take a look at the different functions implemented in this module, remember that *jsonManager* module is employed by the emotion processing module also.

- **__new__(object):** as depicted in the variables this function is used when the instance of the *jsonManager* module is required in order to apply the ***Singleton pattern***.

- **setData(dict, String):** this function takes a dictionary as argument which holds the data regarding the processed frames during the execution of the processing modules and stores the data in this dictionary in the temporary dictionary variable *data*, the string argument stands for the type of data that is being copied but at the moment it only stores data regarding faces, no deeper explanation is required for this function as it simply copies the data and serializes it in order to produce a formatted json output, the serializing function is as follows:

  Listing 5.10: Tool used for serializing the faces information.

  ```
  def serializeFace(face, id):
      serialized = {}
      serialized[str(id)] = face.getSerialized()
      for k in face.list:
          serialized[str(face.list[k].frame)] = ↩
              ↪ (face.list[k].getSerialized())
      return serialized
  ```

  Were the id variable holds the value of the id regarding the face that is getting its information serialized at that moment and the face variable holds an instance of the face module in order to call the function *getSerialized()* which will be described soon.

- **setDataSerialized(dict):** its an auxiliary function that copies an already serialized set of data in order to perform more immediate actions regarding the data.
- **loadJson(String):** takes the path to the file system were the output json are and loads the data into a dictionary which is passed in the return value of this function, loading json data in python is easy by using the *json* library and the function ***load(file)*** which takes a file and parses the data contained in that file to produce a dictionary.
- **saveJson(String):** takes the path to the location specified in the options of the system to store the produced json files and writes the file with the serialized data, the *json* library comes with the function ***dump(dict, file)*** which allows the programmer to easily implement this functionality, after the file is saved, the *data* temporal variable is cleared.

▪ **face** is an auxiliary module implemented in order to maintain a cleaner organization of the faces that are being isolated by the face isolation module, this module holds a set of variables that refers directly to the spatial information of particular faces, it is also in charge of determining if a new face appearance belongs to an already detected face and keeps track of every particular face that has appeared over the course of face isolation processing, the variables appearing in this module are as follows:

- **list:** a dictionary that is used to keep track of the face appearances that were detected and determined to belong to a particular face appearing in the processed video, this dictionary holds pairs of values with the following structure: {"frame":{<face data>}, ...} the frame were it was detected and the data obtained about that face appearance in that frame.
- **x, y, w, h, frame, occurs:** are integers that hold the spatial data regarding that face instance, x and y follow the OpenCV way of organizing the images as a grid of pixels were the x=0 and y=0 point is at the top left corner, the w and h are values regarding the width and height of the frame section that was found to be a face so the rectangle that crops the frame to obtain a single face image is obtained by doing x+w and y+h which gives us a new point, the frame value holds the time when that face appearance was detected and the occurs value holds the number of times that particular face has appeared.
- **valid:** holds a boolean value that determines if the face appeared in an enough number of frames to be considered as "valid" which in other words means that the algorithm determined that face to be a real face, some misfires can occur but the user has the functionality to correct these misfires.

Those are the variables used when processing the isolation of faces appearing in the video source, now lets take a look at the set of functions that are implemented in this module in order to support the operations of the face isolation module.

- **queue(face, dict):** this function is in charge of adding newly detected faces that are proven to belong to an already existing face in order to feed the *list* variable that keeps track of the faces that are being processed, each face has its own list of appearances, the face argument holds the instance of the new face that was detected, in order to avoid overloading the memory the function its implemented as follows:

Listing 5.11: Queue function that acts as a linked list for holding faces track

```
def queue(self, face, queue):
    if queue is not None:
        self.list = queue
    else:
        pass
    self.list[face.frame] = face
```

Were the *if* statement checks if that particular face already holds an instance of the queue, this avoids a particular face to have multiple lists a problem that leads to an exponential growth in memory usage, the queue argument is always holding the first instance of the queue that its created when a new face that has no previous occurrences appears.

- **paintFrame(cv2.frame, Integer, Integer, Integer, Integer):** this function is used in order to draw figures in the frame instance passed in the cv2.frame argument, the integers passed as arguments hold the values for x, y, w, h, the function checks if the option for showing the collision squares mentioned above is selected and if that is the case it draws the figures in the frame instance to be shown in the video player.
- **equal(float, face, integer):** the most interesting function of the *face* module, this function implements an algorithm that determines if a new face appearing belongs to an already processed face, lets take a look at the implementation:

Listing 5.12: The face isolation algorithm

```
def equal(self, conf, item, frame):
    x1, y1 = int(self.x + (self.w * conf)),int(self.y ↩
        ↪ + (self.h * conf))
    x2, y2 = int(self.x - (self.w * conf)),int(self.y ↩
        ↪ - (self.h * conf))

    if x2 < 0: x2 = 0
    elif y2 < 0: y2 = 0

    rx = range(x2, x1 + 1)
    ry = range(y2, y1 + 1)

    self.paintFrame(frame, x2, y2, x1, y1)

    if item.x in rx and item.y in ry:
        return True
    else:
        return False
```

This algorithm looks simple but it took almost two days to be made, it implements an efficient way on determining if the *item* argument which is a new face object is an appearance of an already processed face, it takes the values of the previous face and builds a collision square, if the x,y point of the new detected face is inside the calculated collision square **(Figure 5.12)**the algorithm determines that the new face is actually a new appearance of an already processed face and returns True so the new face is added to the *list* variable of the already processed face, the frame argument holds the instance of the frame that is being processed so a call to the function *paintFrame* allows the function to draw the collision squares used by the algorithm in the video player of the system, showing the collision squares allows the user to modify the threshold size option and visualize it in real time, the threshold value refers to the size of the collision squares and its passed to this function as a float value.

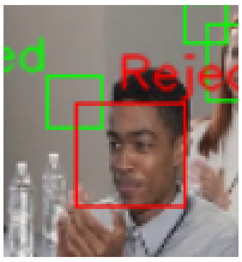


Figure 5.12: Collision squares are shown in the video player in the top-left corner of the faces detected

- **getSerialized():** returns the serialized set of data regarding the face object it is used when there is a need for producing a json output and its mainly called when saving the json file by the *jsonManager* module, it can be also called in case of face deletion or frame level face deletion, this function returns a dictionary of strings.

5.2.2. Emotion detection module

This module is in charge of processing the emotions that the isolated faces detected by the face isolation module have according to the emotion detection model, the execution of this process is very demanding in terms of performance so the review part of this stage is on a separated thread that uses the output produced by this module, the output is also a json produced in the *jsonManager* instance that includes the pair {"frame":<frame id>{"prediction":<emotion detected>}} to the already existing data in the output json of the face isolation module.

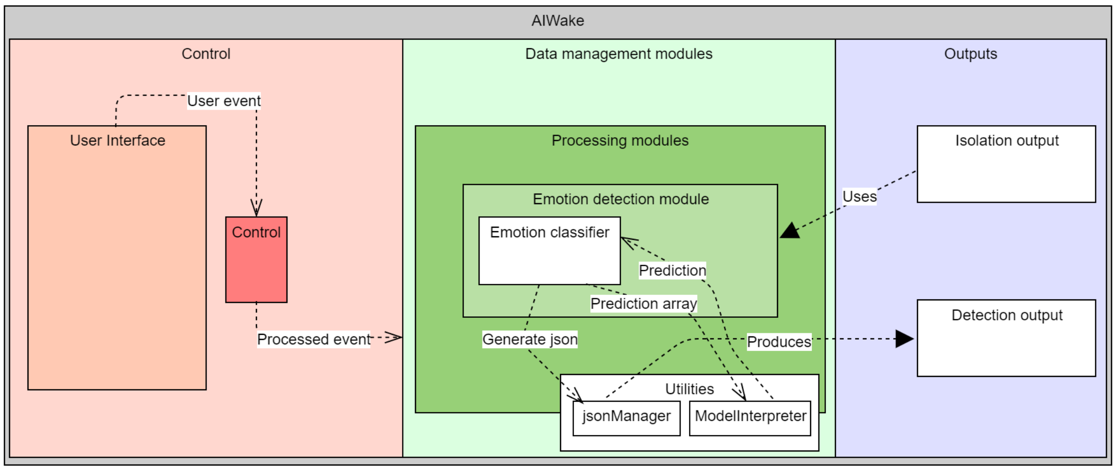


Figure 5.13: Components involved in the emotion detection module

Figure 5.13 shows how the different elements involved in emotion detection are connected, some of them have already being depicted in this document such as the *AIWakeUI* control module and the *jsonManager* which was depicted in the previous section, the new elements added in this module are the *EmotionProc(QThread)* module which executes the core functionality of this module and the *ModelInterpreter* module which is included in the *Utilities.py* file as a tool to support the detection process.

So with the *jsonManager* module already explained, lets take a look at the different variables and functions implemented in the *EmotionProc(QThread)* module and the *ModelInterpreter* module starting with the *EmotionProc(QThread)* module:

- **EmotionProc(QThread):** this module holds the main functionality of the emotion detection module, it makes use of *TensorFlow* libraries in order to use the *TensorFlow-Keras* model interpreter and the *CUDNN* library that is loaded when this module is instantiated, the declaration of this module and the execution of the *start()* function in the *AIWakeUI* control thread is delayed till the first stage of the system completes due to the time that it takes to the *Emotion-Proc(QThread)* module to load all the imported libraries from *TensorFlow-Keras* this measure of delaying the start of the module gives the system a fast start and delays the waiting time to the moment the user saves the json file of the face isolation process which makes the overall flow of the system the sensation of being more dynamic as the user has the sensation the json file is being processed and saved while the libraries of *EmotionProc* module are being loaded, its just a question of timing.

  lets take a look at the variables of this module, most of them are control variables that are modified by the *AIWakeUI* control thread in order to coordinate the processing of the face emotions:

- **pathToVideo:** a string that gets its value from the *PATH_TO_VIDEO* variable this value cant be modified and is determined at the start of the system execution by the user, the module uses this variable to get access to video source file in order to reopen the video and get the faces that were detected in the face isolation module.
- **pathToOutput:** this string holds the value for the output file that this module will produce after its execution, the value is specified by the user in the options pane but it has a folder located in the project files as default location.
- **js:** the *jsonManager* instance, this variable acts the same way as in the face isolation module and its the same instance as the one declared by face isolation module.
- **iterations:** an integer that keeps track of the iteration in which the process is operating, this variable could be named "frames" because it can be equal to id of the frame that is being processed but sometimes there is a need of modifying the value of this variable in a different way the frames does, we could understand each iteration as each time a frame is fully processed.
- **model:** the instance of the emotion recognition model, this module uses the *Keras* model loader to load the model that is located in the specified *PATH_TO_EMODEL* path in the options pane, by default this path points to the model that was trained for the emotion detection stage of this system which is located inside the project files, but the user can provide a new path to a custom model that if it fulfills the requirements of the ***MobileNetV2*** model architecture and providing the set of labels required by its model the user could use a new custom model for emotion detection or any classification problem related to faces (not tested in other classification problem domains, but theoretically it should work).
- **faces:** this dictionary holds the data of the face isolation json output file in order to obtain fast access to the data by the emotion detection module, the dictionary values are obtained by using *jsonManager loadJson()* function.
- **x_data, y_data:** these two list holds the values of the processed emotions that are send to the data visualization plotters in order to plot the processed data in real time, the lists are send through signals to the main thread that forwards these two lists to the data visualization threads.
- **labels:** a dictionary that holds the values corresponding to the predictions made by the model on each emotion, this dictionary holds cumulative values that are incremented when a coincidence is found.

Those were the variables used in the *EmotionProc(QThread)* module now lets take a look at the implemented functions of this module which are easy to understand as the usage of *Keras* library simplifies the implementation of the emotion detection part as it provides all the necessary functions to load and predict data models.

- **setStatusText(String):** the same behaviour as the function implemented in the face isolation module, it takes a string and returns a list that contains the data to be displayed in the status section of the user interface.
- **crop(cv2.frame, Integer, Integer, Integer, Integer):** it takes the frame instance that is being processed and crops a section according to the spatial coordinates specified by the integer arguments of the function which are x, y, w and h, this function returns a frame array which is resized to fit the requirements of the model architecture and normalized in order to have the correct values for each pixel, as this frame array represents an image each cell of the array is a pixel, lets take a look at the code:

Listing 5.13: crop() returns a cropped section of the video frame processed to fit the model interpreter requirements

```python
def crop(self, frame, x, y, w, h):
    cropped = frame[int(y) : (int(y) + int(h)), int(x) : (int(x) + int(w))]
    resized = cv2.resize(cropped, (224, 224))
    resized = np.expand_dims(resized, axis=0)
    resized = resized / 255.0
    return resized
```

As seen in the code snippet, the frame section is extracted directly from the frame array using the python array operands, then it is resized to fit the ***MobileNetV2*** architecture (which is the architecture our model is based) after resizing, the image dimensions are expanded to add a new axis to the image, again, to make it fit to the model architecture, finally, the image is normalized to hold the pixel color values correctly (pixels are represented with three values regarding color, red, green and blue (when RGB format) this values are 8-bit values which have a range that goes from 0 to 255). After all that image processing the function returns a new array that represents a new image extracted from the video frame, this image holds only the processed face section (green rectangles in the video player) and this new image is send to the model in order to predict its emotion.

- **saveProcessedFaces():** this function takes the data contained in *faces* variable and calls the *setDataSerialized(dict)* function of the *jsonManager* module in order to produce and save the output generated in this module execution.
- **generatePlotData():** generates the values for *x_data* and *y_data* to be send to the main thread through a signal emitted after completing an iteration in order to plot data in real time while the processing of emotions is taking place.
- **run():** overwrites the *run()* function of the *QThread* module which is inherited in the *EmorionProc* module and executes the main functionality of this module, it is not as large as the face isolation run function as this module only processes and its not in charge of the reviewing process, the main part of the *run()* function is the following:

Listing 5.14: The section of the run() function in *EmotionProc* module that is in charge of determining emotions

```python
1  if ret:
2      frame = cv2.resize(frame, (540, 380), fx=0, fy=0, interpolation=cv2.INTER_CUBIC)
3      for k in self.faces:
4          if str(self.iterations) in self.faces[k]:
5              faceMat = self.crop(frame,
6                  self.faces[k]
7                  [str(self.iterations)]["x"],
8                  self.faces[k][str(self.iterations)]["y"],
9                  self.faces[k][str(self.iterations)]["w"],
10                 self.faces[k][str(self.iterations)]["h"])
11             predictions = self.model.predict(faceMat)
12             pred =
13                 Utilities.ModelInterpreter
14                 .getClass(n=np.argmax(predictions))
15             self.faces[k][str(self.iterations)]
16                 ["prediction"] = pred
17             config.LABELS[pred] += 1
18             predictions_inst[pred] += 1
```

The snippet shown above has one very important part that also empowers the concept of *Keras* simplicity when programming with deep learning models, it takes just a line (*line 11*) and the trained model which was loaded at the initialization of this module to start predicting the faces emotions, as can be seen, the for loop in *line 3* is in charge of iterating over the dictionary containing the parsed json output of face isolation module, each *k* holds a face structure regarding the spatial data to crop the section for that face in the video frame, *k* is the value of the frame and along with the iteration value it is used to pick the face data regarding the iteration and the frame that are being processed at that moment.

The rest of the function is in charge of emitting signals for status updates and data visualization so there's no need for a deeper explanation.

- **ModelInterpreter:** this module contains a tool that allows the emotion detection module to determine which label belongs to the maximum value detected after a prediction, the return value of calling the *Keras* model *predict()* function is an array that looks like this:

$$\begin{pmatrix} 0.76344 & 0.12334 & 0.00023 & 0.2364 & 0.41 & 0.24213 & 0.00001 \end{pmatrix}$$

  Were the highest value represents the estimated prediction, each position of the array represents a label so a way to translate this position into its corresponding label string value is to pass the index of the highest value to a translator function, for the case of the example array shown above it would be *0.76344* which corresponds to the label *"angry"* in the model trained for this system.

  This module uses only a variable:

  - **components:** this variable is a list that holds the string values of the labels, the position of each of this values corresponds to the positions it should have in the prediction array.

  And have only one very simple function for the translation:

  - **getClass(Integer):** which is called after the prediction of the face cropped image section is processed, the integer argument is given by the use of *np.argmax()* function which returns the index of the highest value inside an array, this function just returns the value holded in the *components* list variable that is in the index specified by the *np* function.

5.2.3. Emotion detection review module

This module is a continuation of the emotion detection module processing, it was designed in order to split the processing load present in this stage of the system flow, this module contains all the necessary elements to manipulate the obtained data coming from the emotion detection module and most of the functions in this module are controlled via the main thread user events by means of a set of boolean variables that act as a lock between actions and coordinate the review process flow.

The diagram for this module (Figure 5.14) is simple as it relays only in the interaction between the user interface and the output data taken from the emotion detection module when it finishes its processing stage, one important feature about this characteristic is the fact that the user could provide the json file already processed and start the system from the emotion detection review module directly.

As we can see in the diagram below, the module is similar to the previous one with the difference of not employing any supporting module related to the purpose of the module scope, the review process makes use of the *jsonManager* module to load the json output produced in the emotion detection module and the variables file in order to access general system variables like the video source path or the current selected face in the face-picker.

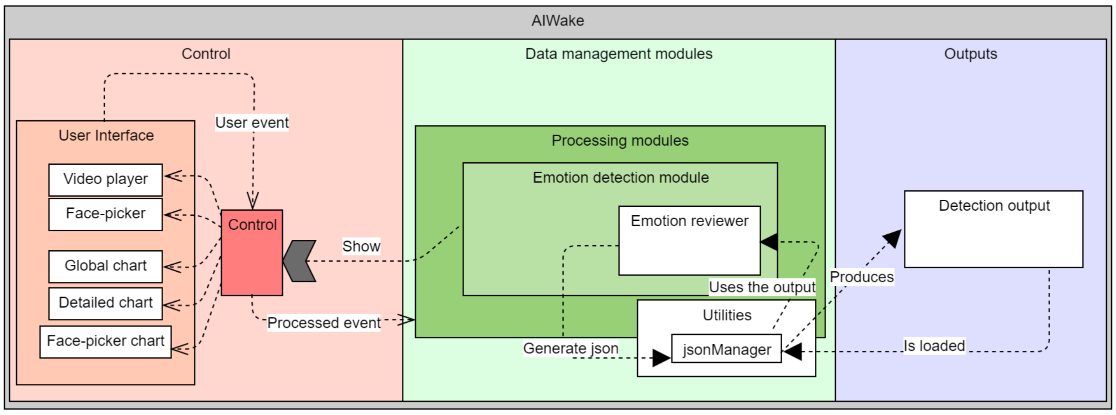


Figure 5.14: Components of the emotion detection review module

Lets take a look at the different elements that are part of the *postPreview(QThread)* module:

- **postPreview(QThread):** first of all lets take a look at the variables that are employed in this module, remember that this module is intended to be a controller for the review stage of the data produced in the emotion detection module and all its functions and variables are implemented in order to achieve an optimal flow when the user arrives to this stage, there are no big processing requiring functions in this module as its main goal its to manipulate data, now lets see the variables involved:
    - **pathToVideo:** same purpose as in the other modules.
    - **js:** instance of the *jsonManager* module in order to read and write json files.
    - **facesInfo:** a dictionary that holds all the information obtained when the json output of the emotion detection module finishes its execution, the file is loaded when the system detects such json file exists in the path specified by the *PATH_TO_JSON_POS* variable.
    - **iterations:** in this module the iterations variable refers to the current frame selected by the user when reviewing the data, it its modified before any processing about the data is made in order to prepare the video player output for playing video at the exact frame the user wants to review, it is also used when there is a need for data to be retrieved about any present frame in the video.
    - **showOnlySelected:** this variable is modified in the main thread and is used for the review process to only draw a square in the frame section were the face that is being reviewed is, its main purpose is to clear the reviewed frame and focus only on the selected face.
    - **pause:** this variable as its name indicates its used for pausing the video player, the pause event can happen when the user holds the slider of the video player or when the user clicks on the pause button.
    - **writeOnPause:** variable used when the video is paused to send feedback to the user only one time (remember we are inside a loop) this variable makes sure that the information is sent to the status label only one time when the pause event occurs.
    - **firstTime:** due to the loop were the review process is happening information can be sent multiple times, this variables is used for sending data to the face-picker utility only one time when a face is selected and the frame being reviewed changes with other frame, it differs from the *writeOnPause* variable because the frames change if the *pause* variable is not set to True.
    - **barReleased:** this variable is used to modify the value of the *iterations* variable when

the slider bar for selecting the frame is released and forces the video player to show the selected frame when this event happens.

- **forward, forwardToProcessed, backward, backWardToProcessed:** this variables behave in the same way as in the face isolation review stage.

Notice how most variables are booleans that control the events flow of the system, all this booleans are modified depending on the events that happen in the user interface due to the user interaction directly from the main thread, lets see a code snippet of a function in the main thread that modifies the value of one of these booleans.

Listing 5.15: Example of the main thread modifying control variables of the emotion detection review module

```
def sliderPressed(self):
    if self.currentThread[2]:
        self.thread[3].pause = True
```

The code seen above checks if the current thread is the emotion detection module one, in this case it refers to the stage the system flow is because the thread of emotion detection was stopped, but as the thread that belongs to the review stage is part of the emotion detection stage of the system this function checks if the event of pressing the slider for selecting the frame is happening during the review phase of the emotion detection stage, if that's the case the *pause* variable value is set to **True** and the video player pauses the video at that frame.

The functions that are implemented in this module are mostly supporting functions for the data manipulation, the *run()* function its in charge of the whole process of controlling which information is displayed and which not and sends a constant feed of signals with the processed frames depending on the *pause* value at that moment, lets take a look at the functions implemented on this module:

- **getFaceIds(dict, string, boolean):** returns a list of the available faces to review to the face-picker utility, it has the same behaviour as in the face isolation module.
- **generateFaceInfo(string):** returns a list of strings that are sent to the face-picker utility to set the values regarding the selected face.
- **drawFaceInfo(string, cv2.frame, Integer, Integer, Integer, Integer, boolean, string):** returns a processed frame, the processing of the frame consists on drawing the corresponding figures regarding the face spatial information along with the prediction label that was determined in that frame for that face in the emotion detection module, lets take a look at the code of this function:

Listing 5.16: Use of cv2.rectangle and cv2.putText to draw information over the processed frame

```
def drawFaceInfo(self, id, frame, x, y, w, h, ↩
    ↪ selected, pred):
    if config.DETAILED:
        if selected:
            cv2.rectangle(frame, (x, y), (x + w, y + ↩
                ↪ h), (255, 255, 255), 2)
            cv2.rectangle(frame, (x, y), (x + w, y + ↩
                ↪ h), (0, 255, 0), 1)
            cv2.putText(frame, id + "␣" + pred, (int(x ↩
                ↪ + (w * config.PROP)) + 5, y - 5), ↩
                ↪ cv2.FONT_HERSHEY_SIMPLEX,
                0.5,
                (0, 255, 0), 1, cv2.LINE_AA)
        elif not self.showOnlySelected:
            cv2.rectangle(frame, (x, y), (x + w, y ↩
                ↪ + h), (0, 255, 0),
                              1)
                cv2.putText(frame, id + "␣" + pred, ↩
                   ↪ (int(x + (w * config.PROP)) + 5, ↩
                   ↪ y - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                            (0, 255, 0), 1, cv2.LINE_AA)
    return frame
```

The frame instance argument is modified by drawing figures over it using the cv2 libraries for image manipulation, depending on the options chosen by the user the way of drawing the figures is different, notice how if the face is selected in the face-picker utility the function will draw another white rectangle with a slightly higher border size to give a highlighting effect on the video player for the selected face.

- **setStatusText(string):** sends a list via signaling the main thread to update the status feedback text.
- **setFaceFrameMood(string):** sets the value of the emotion detected in a face on the selected frame to a new one selected by the user, the value is sent to this module by the main thread regarding the value of the combo box selector in the face-picker utility.
- **deleteFaceFrameMood():** deletes an occurrence of the selected face in the selected frame by user demand.
- **setFaceFrameMoodAll(string):** this function modifies the value of the prediction given by the emotion detection module in all the occurrences of a selected face, its important to know that this option is for cases were the overall predictions from a face are kind of strange according to what the user is seeing in the video player.
- **dleteAll():** deletes all the selected face occurrences, this function can be used when the *setFaceFrameMoodAll* function is not enough to correct a particular face detected emotions misfires, for example if the user states that a particular attendant had a mix of happy, disgust and angry emotions but the predictions made by the model says that the attendant was happy all the time correcting the emotions frame by frame can be an exhausting task so it may be worth it to not consider that attendant in the final results.
- **processFaces(cv2.frame):** it extracts the spatial characteristics of a given face and processes the given frame in order to draw the figures to represent the data, the *iterations* value is used here to determine if a face contained in the *facesInfo* dictionary belongs to the current frame drawing the required figures in the frame if thats the case, lets take a look at the first part of this function, the characteristics extraction:

Listing 5.17: Management of the characteristics of the faces to modify the frame instance

```
def processFaces(self, frame):
    if frame is not None:
        for k in self.facesInfo:
            if str(self.iterations) in self.facesInfo[k]:
                prediction =
                    self.facesInfo[k]
                    [str(self.iterations)]["prediction"]
                x =
                    int(self.facesInfo[k]
                    [str(self.iterations)]["x"])
                y =
                    int(self.facesInfo[k]
                    [str(self.iterations)]["y"])
                w =
                    int(self.facesInfo[k]
                    [str(self.iterations)]["w"])
                h =
                    int(self.facesInfo[k]
                    [str(self.iterations)]["h"])
```

Much of the functions of this system are designed to avoid nested loops which would lead to great complexity algorithms that would have great impact in execution time of the system.

- **getNearestProcessed(Integer):** same behaviour as the function in the face isolation module.
- **run():** the main function of this module it is in charge of picking the next frame to be displayed in the video player and resize this frame in order to maintain the consistency between the modules it is the less complex *run()* function we can found in the processing section as it contains just the execution loop of the thread and the controls to determine the frame to display according to the *SELECTED_FRAME* variable value, it also controls the pause event and the forward and backward events.

  The *run()* function is called when the *start()* function is invoked from the main thread, but the main processing of this function takes part in the *processFaces* function as the main goal of this module is to draw figures regarding the json data of the output from the emotion detection module.

Those were the core components encompassed in the processing aspect of the system, the last but not least is the data representation aspect that will be depicted in the next section.

5.3. DATA REPRESENTATION

This section will show the different modules implemented for data representation in the system, the main concept was to split the different types of representation approaches into modules that when combined the result is a consistent and complete representation of the data that is being obtained by the system, the main approaches are:

- **Global/Cumulative representation:** this module has the approach of producing charts that represent the cumulative values processed by the system, it is intended to show a global overview of the processed data.
- **Detailed/frame by frame representation:** this module gets the data from the instant set of values produced at a particular frame being processed, it is intended to plot charts that represent what values are being obtained at the moment of being processed.
- **Face-picker/face-level representation:** this module provides the face-picker utility with a bar chart that represents the values of a selected face on a certain frame, it is subdivided in two modules, the face isolation stage one and the emotion detection one in order to maintain the structure of having two different layouts were these charts are drawn in the user interface.

Before explaining how the different approaches were designed lets get a brief summary on how the *QCharts* library works, the *QCharts* library provides the programmer with a set of objects capable of holding data in different formats, this way of holding the data enables the transformation of this objects into another objects for different data representation, for example, a pie plot is made through a *QPieSeries* object while a bar plot is done with a *QBarSeries* object so by storing the data on an optimal structure like a python dictionary we can morph the data into the desired *QChartSeries* object and the chart layout will represent the data with that particular *QChartSeries* format because the *QChart* object is mapped to the layout in the user interface and the *QChartSeries* objects are linked to the *QChart* object.

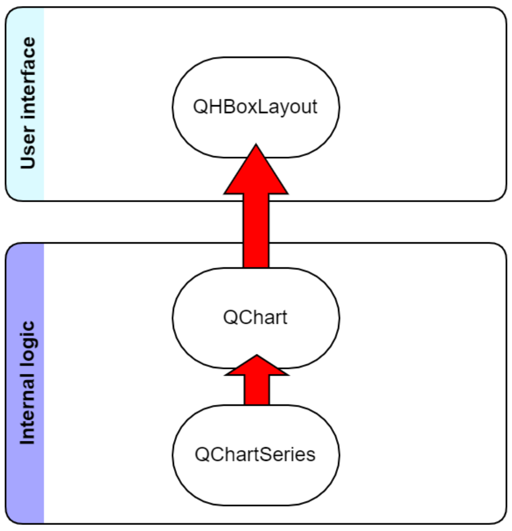


Figure 5.15: QChart objects interaction

The implemented way to directly modify the charts in real time of this system was to send the *QChart* object instance to the representation modules initial arguments so we can modify the *QChart* instance directly in the representation thread instead of doing it in the main thread (Figure 5.15), lets take a look on how these three different approaches are implemented in the system in order to achieve a better understanding of this concept.

5.3.1. Global representation module

In the list above we had an overview of the approaches the representation modules have in this system, this section will break down the different elements involved in the global representation module, this module is in charge of providing the user with cumulative representation of the data that is being processed by the processing modules, the main goal of this cumulative representation is to give an overview on how the presentation developed and which were the most detected emotions during the course of the presentation, for example, the bar chart can give the user an overall view of the obtained data to notice the percentage of times the "happy" emotion appeared in the attendants faces and compare it to the total amount of times the "sad" emotion appeared while the line plot can give the user information about how these emotions evolved during the presentation, for example if in the middle of the presentation a bad joke was said then in the frames regarding that bad joke the angry cumulative percentage incremented while the happy level decreased this way allows to detect portions of the presentation that could be modified in order to maximize the overall happiness level.

This module **(Figure 5.16)** is huge compared to other components of this system this because of the way *QCharts* manages the data structures, as can be seen in the diagram there are no supporting components or external tools used in the execution of this module, the *GeneralPlotManager* module manages the data that it receives from the main thread by itself morphing the data structures into the *QChartSeries* required for the type of plot the user wants, Figure 5.16 shows all the components regarding the data representation module as there is not much difference between these modules, the main difference relies in the way the data is processed to produce the charts.

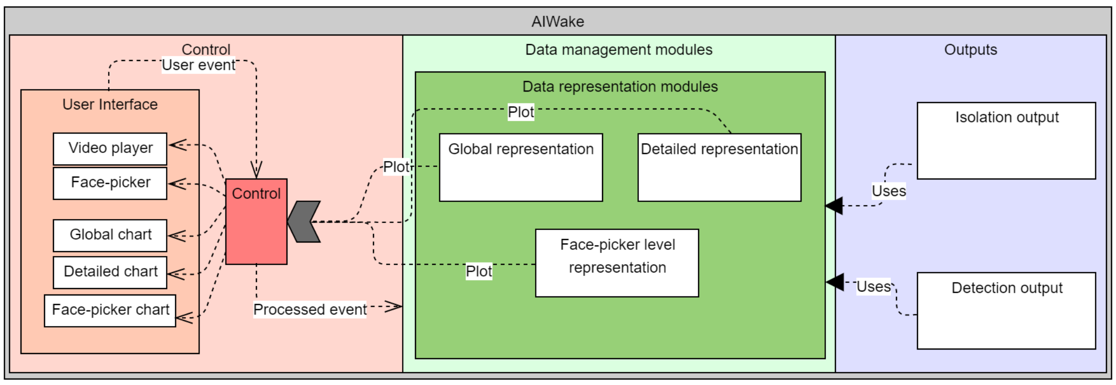


Figure 5.16: Data representation module components relationship

Lets break down all the variables and functions that were implemented in this module:

- **chart:** the instance of the *QChart* object that is linked to the general plot layout, the way the elements are linked to this variable makes possible to modify the chart view directly by modifying the *QChartSeries* data structures that are linked to it.
- **series_Pie, series_Bar, series_Line:** the *QChartSeries* structures were the data that is being feed by the main thread morphs, assigning a *QChartSeries* structure to the *QChart* object changes the way the data is plotted in the layout, each different type of plot requires its correspondent type of *QChartSeries* being in this case *QPieSeries, QBarSeries* and in the case of the line plot the series_line object is a dictionary that contains instances of *QLineSeries* that are assigned a set of *QPoints*, this series are used for the face isolation module data.
- **series_PieProcessing, series_BarProcessing, series_LineProcessing:** the counterpart of the *series_Pie, series_Bar* and *series_Line* this series are used for the emotion detection module data.
- **labels:** holds the value of the labels that are being plotted, for the face isolation module it represents the accepted and rejected strings and in the case of the emotion detection module represents the different emotions in string format.
- **values:** holds the values to be plotted, acts as a supporting variable to feed the hashes that relate the labels and values.
- **labelHash:** a dictionary that holds the related values coming from the *labels* and *values* variables, this variable is used to feed the *QChartSeries* objects regarding the face isolation module data.
- **labelHashProcessing:** the counterpart of *labelHash* dictionary, this variable holds the data pairs regarding the emotion detection module data.
- **change:** a boolean control variable that indicates the module that the selected plot has changed this change morphs the *QChartSeries* object into the desired chart series and does it only one time every change.
- **slices:** a list that holds the *QPieSlice* object that is required for pie plotting in the face isolation module, its emotion detection module counterpart is the **slicesProcessing** variable.
- **firstTimePie, firstTimeBar, firstTimeLine:** control variables that determine if is the first time the user selects a type of plot, this is used to prepare the layout information elements that will be shown like the legend and the axis.
- **processing:** indicates the module that the mode of representing the data has changed to a different processing module.

- **initial:** a boolean used to indicate the module that its the first iteration of the plotting process in order for the module to instantiate the *QChartSeries* objects in order to represent the data.
- **last:** support variable that holds the value of the last frame id that was processed, this way allows the plotters to not plot the data multiple times in the case that the thread executes faster than the iterations are incremented during processing.
- **frame:** the id of the actual frame that is being processed by the processing modules used along the *last* variable to check the condition of not plotting multiple times the same frame data.

Now lets see the functions involved along with some code snippets in order to provide a better understanding of the *QCharts* library usage for achieving great data representation.

- **plotSelect_processing():** the controller that selects the type of *QChartSeries* object the data is gonna be morphed into, lets take a look at a code snippet of the case were the user selects the bar plot:

Listing 5.18: The code executed when the user selects the bar plot

```
1  elif config.SELECTED_CHART == "Bar":
2      if self.change:
3          self.series_pieProcessing.hide()
4          for l in self.series_lineProcessing:
5              self.series_lineProcessing[l].hide()
6          self.change = False
7          if not self.initial:
8              self.chart.axes(Qt.Horizontal)[0]
9                  .setVisible(False)
10             self.chart.axes(Qt.Vertical)[0].setVisible(True)
11         if self.firstTimeBarProcessing:
12             self.barPlotProcessing()
13             self.chart.addSeries(self.series_barProcessing)
14             self.chart.setAxisY(self.chart.axes(Qt.Vertical)[0]
15                 , self.series_barProcessing)
16             self.firstTimeBarProcessing = False
17             self.series_barProcessing.show()
18     else:
19         self.updateValues_barProcessing()
```

In the snippet we can appreciate multiple things:

- **variable *SELECTED_CHART*:** this variable is modified in the main thread when the event of changing the value in the plot selector triggers and its used in this function to determine the type of chart to be plotted, python doesn't support the *switch* statement so an if statement is used instead.
- **Lines 2 to 17:** are used when the change in the plot selector is produced, from line 3 to 5 the function hides the other *QChartSeries* objects in order to clear the layout for the *QBarSeries* object, line 7 to 15 belong to the morphing process of the data, notice the use of the *initial* variable to determine which axes to display in the layout and the use of the *firstTimeBarProcessing* variable to instantiate these axes.
- **lines 18 and 19:** in the case the change has already been produced, the function only updates the values of the *QChartSeries* object as all the required elements are already being displayed.

When any function on a plotter module contains the word *processing* it refers to the fact that it will process the emotion detection module produced data, the counterpart of this selector function is the **plotSelect_preprocess** function and acts in the same way as this one, the reason why there are different functions for each processing module is because of the way *QChartSeries* are managed by the *QCharts* library.

- **calculatePercent(), calculatePercent_alt():** functions used to calculate the percent of the processed data, this functions are required due to the fact that the data coming from the main thread comes in a raw format that needs to be set to its corresponding percentage regarding the total data two functions are used for this purpose because sometimes there is a need of calculating the data in a different way.
- **updateValues_line(), updateValues_pie(), updateValues_bar():** these functions update the values of the *QChartSeries* objects in order to update the charts in real time while the data is being processed, lets see the code implementations of the *_processing()* counterparts of these functions:

Listing 5.19: Three functions to update the three different types of charts that are displayed

```python
def updateValues_lineProcessing(self):
    self.labelsHashProcessing = config.LABELS.copy()
    self.calculatePercent()
    for s in self.series_lineProcessing:
        point = QPoint(config.CURRENT_FRAME, ↩
            ↪ self.labelsHashProcessing[s])
        self.series_lineProcessing[s] << point

def updateValues_pieProcessing(self):
    self.calculatePercent()
    for k in self.series_pieProcessing.slices():
        k.setValue(self.labelsHashProcessing[k.label()])

def updateValues_barProcessing(self):
    self.labelsHashProcessing = config.LABELS.copy()
    self.calculatePercent()
    for k in self.series_barProcessing.barSets():
        k.replace(0, self.labelsHashProcessing[k.label()])
```

The important part of these code snippets is to watch how the data coming from the *labels* variable is morphed into the different elements regarding the *QChartSeries* objects.

- **linePlot(), piePlot(), barPlot():** these functions are only called once and are used to create the starting *QChartSeries* objects to be append to the *QChart* instance, these functions also have its *_processing* counterpart and are similar to the *update* functions with the difference of instantiating the *QChartSeries* objects.
- **setLabels():** a supporting function that creates the data pairs when the data coming from the main thread comes in a list format instead of a dictionary, this situation can happen from time to time in the emotion detection module and most of the time in the face isolation module.
- **run():** as said in previous *run* methods contains the execution loop executed when the *start()* function is called over a thread in the main thread, lets take a look at the code:

Listing 5.20: Run function of the Global representation module

```python
def run(self):
    line = True
    while len(self.labels) <= 0 and len(self.values) <= 0: ↩
        ↪ time.sleep(0.033)
    while True:
        if config.CURRENT_FRAME > 0:
            if line:
                self.linePlot()
                line = False
            if self.processing:
                self.plotSelect_processing()
            else:
                self.plotSelect_preprocess()
        time.sleep(0.15)
```

This function controls the stage of the processing flow of the system, the *sleep* function is used in order to prevent a high usage of the cpu thread this process is running in because the execution time of the representation modules is so fast it saturates the threads were they run, notice on how the thread is running at the start of the system and waiting for data to process, the line plot is instantiated at the start of the representation process in order to maintain data consistency when the user changes the type of chart (line plot is continuous while bar and pie plot are instant).

5.3.2. Detailed representation module

As said in the representation approaches list, this module is focused on data representation at a frame by frame level, the main goal is to give the user an overview of the data that has been obtained during the processing of a given frame, the main idea behind this module is to not only provide the user with a visual representation of the data through the video player but to give the user a graphic representation of this data, this way the interpretation of the processed data at any point is easier, the module diagram of this module is the same as in the **Global representation module** with slight differences which will be shown in this section, the main difference is the way of retrieving the data.

The differences between the Global and Detailed modules reside on its functions, the control mechanism is the same for the variables but there is a big difference between the data feed of both modules, lets take a look at the variables that differ:

- **variables regarding the *QChartSeries*:** the mechanism is the same one set of series for each processing module.
- **control variables regarding the different types of charts:** same mechanism, the main difference is the variable *SELECTED_CHART_DET* which is modified in the main thread when the selector of the detailed plotter changes, having a different selector for the detailed section allows the user to obtain different data plots to compare, for example, the user could want to display the global bar plot while looking at the detailed line plot in order to compare the overall of the detected emotions while looking at the frame by frame evolution of this emotions in a continuous way.
- **labels and values pair:** labels list remains the same but the values come in a different way, these values are reset when a new iteration of the processing module comes.
- **labelHash:** the same applies to the *labelHash* and its counterpart *labelHashProcessing* dictionaries the concept is the same but the data consists on the instant data regarding a processed frame.

Regarding the functions, all the differences reside in the way the data variables feed from the incoming data from the main thread, lets see an example of the **updateValues_lineProcessing(), updateValues_barProcessing()** and **updateValues_pieProcessing()** functions:

Listing 5.21: Way of updating plots in the detailed module

```python
def updateValues_lineProcessing(self, labels):
    self.labelsHashProcessing = labels
    self.calculatePercent()
        for s in self.series_lineProcessing:
        point = QPoint(config.CURRENT_FRAME, ↩
            ↪ int(self.labelsHashProcessing[s]))
        self.series_lineProcessing[s] << point

def updateValues_barProcessing(self):
    self.calculatePercent()
    for k in self.series_barProcessing.barSets():
        k.replace(0, self.labelsHashProcessing[k.label()])

def updateValues_pieProcessing(self):
    self.calculatePercent()
    for k in self.series_pieProcessing.slices():
        k.setValue(self.labelsHashProcessing[k.label()])
```

The only difference resides in the way the *labelHashProcessing* variable gets the data.

5.3.3. Face-picker representation module

This module its intended to represent the data regarding a selected face in the face-picker utility section, it follows the same mechanisms as the detailed and global representation modules but with a simplistic approach as this module doesn't allow the user to pick any chart more than the bar plot representation, due to its simplicity the emotion detection and the face isolation views of the face picker utility use two instances of this module, one for the face picker for face isolation data review layout and the other for the emotion detection review face picker layout.

Lets take a look at the last module of this system module diagram, and the elements that are inside this module.

The face picker representation module is simpler than the other representation modules as it only has to manage data regarding a particular selected face, we could say that this module is implemented in order to support the representation modules to provide the user with a deeper level of detail regarding each face processed data.

Lets take a look at the variables involved in the face picker representation module:

- **draw:** boolean that indicates the module to plot the data of a selected face, the main idea is to make the module plot the data only once along with the index change event of the face selector.
- **chart:** the chart instance linked to the face picker tab layout this follows the same principle of managing the charts inside the module taking advantage of the *QCharts* library.
- **series:** *QBarSeries* object linked to the chart instance.
- **labelHash:** dictionary that holds the pairs regarding the data and the labels of the selected face.
- **sets:** a *QBarSet* object, this object holds key pair values that are appended to the series variable, each set represents a label.
- **firstTime:** same principle as in the other representation modules of instantiating the data only one time.
- **facesInfo:** holds a copy of the data processed by the processing modules, its important to notice that this copy is made when the processing modules finish their execution and enter the review stage, review operations for each processing module requires all the video frames to be processed (or all the frames that are meant to be processed when the user selects a processing rate).
- **dataReady:** indicates the module to start its operation at the start of the review stage when a face is selected.

To end this module elements lets explain the functions of this module:

- **plot():** set all the necessary elements for the *QChart* object to be plotted this means it instantiates the axes, the *QBarSets* and adds the *QBarSeries* object to the *QChart* instance, lets take a look at the code:

  Listing 5.22: All the necessary variables being instantiated to start plotting data.

  ```
  def plot(self):
      self.calculatePercent()
      for k in self.labelsHash:
          set = QBarSet(k)
  ```

```
            set << self.labelsHash[k]
            self.sets.append(set)
        for s in self.sets: self.series.append(s)
        self.chart.addSeries(self.series)
        axisY = QValueAxis()
        axisY.setRange(0, 100)
        self.chart.addAxis(axisY, Qt.AlignLeft)
        self.chart.setAxisY(self.chart.axes(Qt.Vertical)[0], ↩
            ↪ self.series)
```

this function is executed just once when the data is ready to be reviewed.

- **update():** this function updates the *QBarSets* that hold the data, it works the same way as the *updateValues_bar* of the global and detailed representation modules.
- **calculatePercent():** gets the percentage value of the data.
- **getData():** updates the *labelHash* variable value according to the data provided by the main thread.
- **run():** the main goal of this function is to wait till the data is ready, then it enters the execution loop that waits for the *draw* variable to change in order to plot the data regarding a selected face when the index of the face selector changes.

CHAPTER 6

Results

With all the previous chapters viewed it is time to start talking about the results obtained during the development, this chapter will depict the results obtained at each iteration described in the *methodology* chapter and the final result with some screenshots of the application fully working, the data representation modules layouts and the processing modules review stages along with some statistics regarding the project and the implemented code, at the end of this chapter we will also have a look at the project budget estimations.

6.1. PER ITERATION RESULTS

This section will break down the results obtained regarding the project development at each iteration described in the *methodology* chapter along with design choices and schemes that belong to these iterations.

6.1.1. Iteration/phase 0, generating a deep learning model for a classification problem

This iteration is focused on the learning process in terms of programmer knowledge and machine learning, it was the most time demanding iteration and was intended to produce the application own model for emotion detection [18], due to memory management problems caused by the hight ammount of training data used which was about 30000 images of faces taken directly from the *FER-2013* data set that can be found in *Kaggle* website, after some studying about memory management when training models [19] and about the *TensorFlow Keras* libraries and deep learning models the design approach of a continuous training process was discarded and the following processing was implemented:

The resulting process divided the memory load into separated chunks of arbitrary data [20] regarding the image-emotion labels used for model training, it also started using the system's GPU (NVIDIA) in order to increase performance, the size of the chunks of arbitrary data were determined by the batch size variable which was also used for determine the amount of epochs per steps along with the validation steps, the resulting model obtained an **accuracy metric of 63%** and leaved an open window for improvements (that can be achieved with more training [21]).

The model architecture followed the *MobileNetV2* architecture with a slight modification in the output layers that added the following:

1. 128 outputs *Keras* dense layer with the activation function *relu* (Rectified linear unit).
2. 64 outputs *Keras* dense layer with the activation function *relu*
3. 7 outputs *Keras* dense layer with the activation function *softmax* to produce an array of K (k = 7) arbitrary values were each values corresponds to a label, this is the final output of the model.

The additions of these outputs layers is intended to produce an output that corresponds to a set of 7 labels each one representing an emotion, the goal is to reduce the original ammount of outputs the *MobileNetV2* architecture has which is 1000.

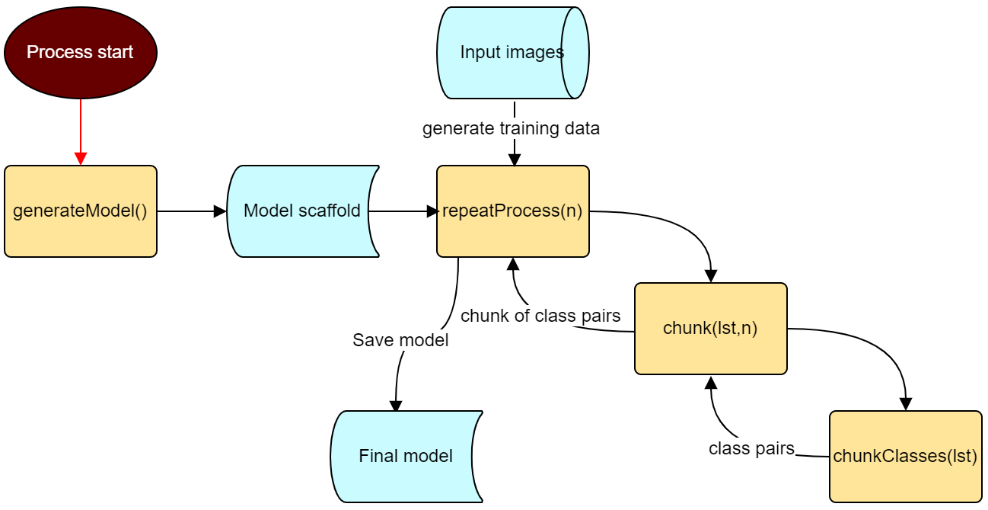


Figure 6.1: Model training process designed to avoid memory issues when the model was trained for this application

The idea behind the decision of using *MobileNetV2* architecture instead of for example *FaceNet* was the *MobileNetV2* approach which as intended for mobile devices provided a lighter model architecture suitable for solving the problem of memory management along with the scheme described in Figure 6.1, the training process took over 2 weeks because of more random memory issues while training that ended up corrupting the produced model, each failure come with an improvement in the model training processing scheme until the process was able to produce a model with the whole *FER-2013* dataset.

6.1.2. Iteration/phase 1, processing the video source in order to detect the faces to be processed

This iteration was implemented during the development of the iteration 0 due to the lack of dependency between these two iterations as the emotion detection model was intended for the iteration 2, as said in the *methodology* chapter the approach of this iteration was to create a module for face isolation of the faces appearing in the video source, the results were the best in comparison with the other iterations as this iteration module gave the less problems and began the core concept of the main user interface for the project.

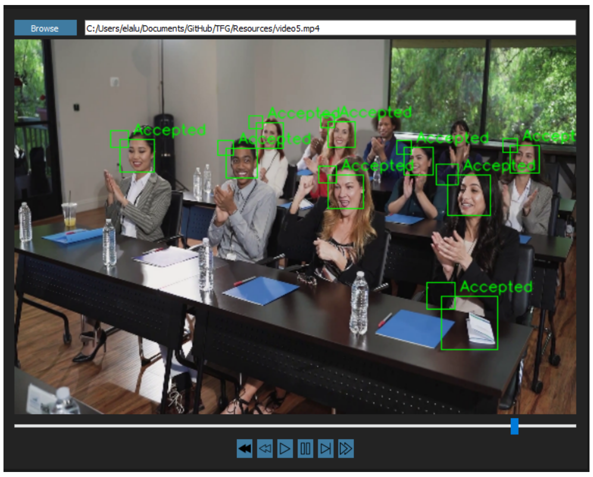
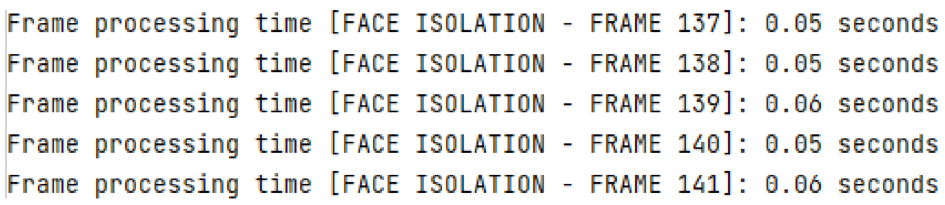


Figure 6.2: A video sample from the website *pexels* being processed in the final application

Notice in Figure 6.2 that there's a misfire done by the algorithm, a paper sheet in the right bottom corner of the frame has been accepted as a face, this type of errors made by the face detection model are made because of the classification problem nature, the application allows the user to delete this model misfires but this functionality belongs to the iteration 3.

The global context at the time the project was being developed didn't allow the gathering of own video samples for testing the algorithms so the video resources were taken from free video repositories like *pexels*, the time took by the algorithm to process a single frame at the face isolation module was as follows:

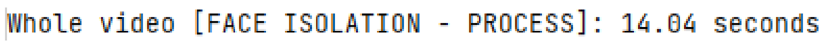


```
Frame processing time [FACE ISOLATION - FRAME 137]: 0.05 seconds
Frame processing time [FACE ISOLATION - FRAME 138]: 0.05 seconds
Frame processing time [FACE ISOLATION - FRAME 139]: 0.06 seconds
Frame processing time [FACE ISOLATION - FRAME 140]: 0.05 seconds
Frame processing time [FACE ISOLATION - FRAME 141]: 0.06 seconds
```

Figure 6.3: Execution time of single frame isolation

The results of Figure 6.3 belong to a video sample of a presentation with 11 attendants the mean **execution time is around 57.5 milliseconds** with is a great latency and makes the algorithm capable of isolating faces in the video source at a speed of **17 frames per second**

In regard of the total video source, in the same sample that has **262 frames** the execution time obtained directly from the application's terminal is shown in Figure 6.4:



```
Whole video [FACE ISOLATION - PROCESS]: 14.04 seconds
```

Figure 6.4: Execution time of the whole processing of the video

This time starts counting from the beginning of the *Run()* function and stops when the *done* signal is emitted, the option to chose the processing rate of the video implemented in later stages of the project, allow the user to skip a fixed number of frames before processing a new frame, this option reduces the execution time of the whole video by a factor of skipped frames in an exchange of obtained data accuracy, its important to remember that any video source is resized to *540x380* when processed so resizing videos has also an impact on the execution time, the samples used while developing the application were *1920x1080* videos which is an standard resolution nowadays and the resolution that was meant for the original way of taking samples using the C:TED (UCLM) infrastructure but due to the pandemic context the project was being developed similar samples were taken from the internet.

Listing 6.1: Sample json output for the face isolation module

```
{
    "1": {
        "1": {
            "x": "281",
            "y": "79",
            "w": "26",
            "h": "26",
            "valid": "False"
        },
        "2": {
            "x": "281",
            "y": "79",
            "w": "26",
            "h": "26",
            "valid": "True"
        },
```

To end this iteration results, Listing 6.1 contains a sample of the output obtained after face isolation module processing, the first level corresponds to the face id, then that face data is stored with the corresponding frame of appearance.

6.1.3. Iteration/phase 2, processing the video source to obtain data regarding emotions

In this iteration things started to get interesting, critical performance issues started to appear and the best design approach as possible for the emotion detection module was chosen, this iteration ended with a very under-powered emotion detection module but the functionality was there to continue with the next iteration.

One thing that changed in the final application was the deletion of the video preview when the emotion detection module is running, this was because it showed the video at a very low frame rate which in user feedback terms tends to cause a negative experience in the user, in this module, the feedback is provided through the progress-bar and the status text.

Although it was under-powered its implementation allowed the testing of the trained model and the review of first results generated by this module (Figure 6.5), the final user interface video output looks the same in the final version but the overall performance of this module was improved in next iterations when an hiatus period for refactoring was established to improve performance.

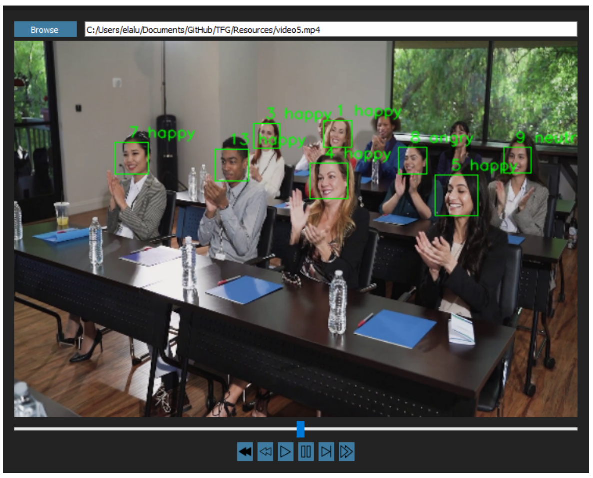
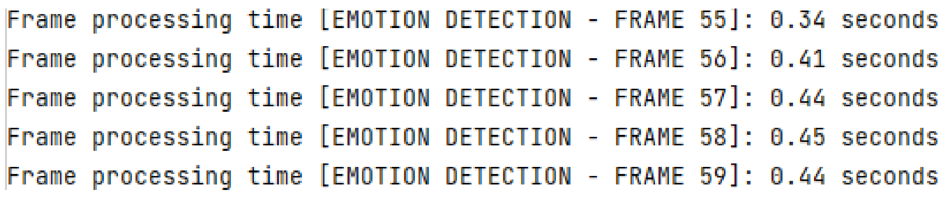


Figure 6.5: Video player during the emotion detection review stage

In the final application the execution time for processing a frame is shown in Figure 6.6:



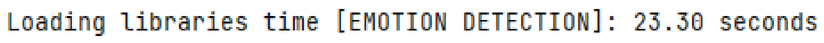
```
Frame processing time [EMOTION DETECTION - FRAME 55]: 0.34 seconds
Frame processing time [EMOTION DETECTION - FRAME 56]: 0.41 seconds
Frame processing time [EMOTION DETECTION - FRAME 57]: 0.44 seconds
Frame processing time [EMOTION DETECTION - FRAME 58]: 0.45 seconds
Frame processing time [EMOTION DETECTION - FRAME 59]: 0.44 seconds
```

Figure 6.6: Sample of execution times for frames emotion detection

which still very high nowadays and leaves a big open window for improvements, the main reason why the processing time for a single frame is so high relies in the model used for emotion detection architecture, due to the amount of memory issues training the model required a lot of tweaking as said in the iteration 0 results, main reason being the mobile architecture of *MobileNetV2*.

The whole process takes over 2 minutes to complete for a 262 frame video with 11 faces, that lead to design choice of implementing the processing rate option that reduced this time by a factor of almost the skipped frames.

Another disadvantage of this module is the time taken by the processing libraries to load (Figure 6.7), this leaded to the decision of delaying the load of these libraries until the start of the emotion detection module.



```
Loading libraries time [EMOTION DETECTION]: 23.30 seconds
```

Figure 6.7: Time required by the CUDNN library to load up

As a result of the implementation of CUDNN library the system is dependant of NVIDIA GPUs to perform properly, this doesn't mean that an AMD graphics card running the system wouldn't be able to run properly, this means that the execution would be very slow, this is the reason why a live demo isn't available at the moment (when executing the system over the CPU, the execution times grow in almost a factor of three times the CUDNN execution times).

Listing 6.2: Sample json output for the emotion detection module

```
{
    "1": {
        "1": {
            "x": "281",
            "y": "79",
            "w": "26",
            "h": "26",
            "valid": "False",
            "prediction": "surprise"
        },
        "2": {
            "x": "281",
            "y": "79",
            "w": "26",
            "h": "26",
            "valid": "True",
            "prediction": "surprise"
        },
```

As said in the previous iteration results, Listing 6.2 contains a sample section of the output generated by the emotion detection module, its very easy to see the difference, only the predicted value for the emotion regarding that frame is added to the previous data.

6.1.4. Iteration/phase 3, Phase 3, creating an UI for reviewing obtained data

This iteration had the approach of creating the required user interface elements for allowing the user to perform a set of actions over the processed data of each module, the face-picker utility was implemented in this iteration along with the video processing options regarding the face isolation module.

The implementation of this user interface elements was entirely done in the *QT Creator* IDE following a layout scheme in order to give the application the capacity of resizing each section element similarly as many modern applications do nowadays.

The final .ui file containing the XML for the user interface layout **contains 2107 lines of code** some of this lines were manually modified and the rest were generated by the IDE, the development of the user interface started in this iteration and then a continuous development of the user interface was followed.

The face-picker utility is contained in a tab pane which has two tabs, one for face isolation (Figure 6.8) module and the other for emotion detection module (Figure 6.9).

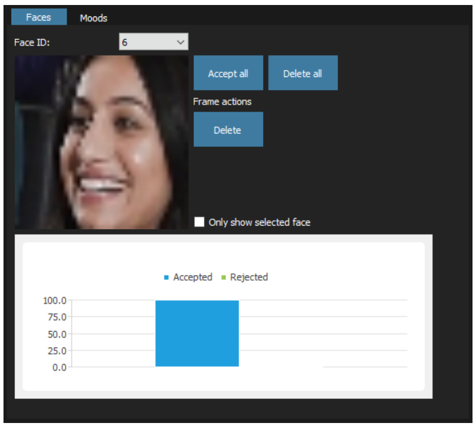


Figure 6.8: Final face-picker utility section in the user interface

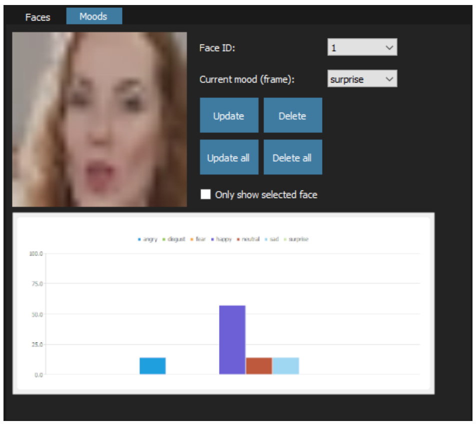


Figure 6.9: Final face-picker utility section in the user interface for the emotion detection module

The overall section of user interface intended to provide review options and options regarding the video processing looked like this before applying the style guide of the application (Figure 6.10):

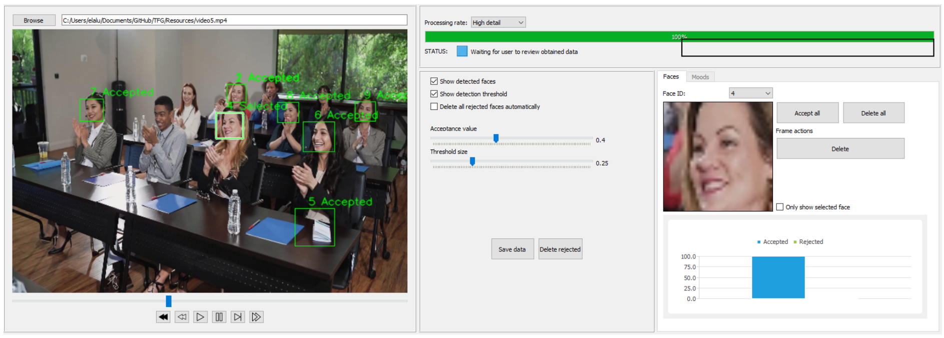


Figure 6.10: The section of the user interface regarding data review when there was no style established

The hiatus period that was taken during this iteration also included the definition of the style guide for the application and the refactoring of the emotion detection module in order to achieve better performance, lets take a look at the results of the sub-iterations of this iteration.

- **Iteration/phase 3.5, refactoring in order to increase emotion detection performance** In this sub-iteration the approach was to achieve a better performance in the emotion detection module the user interface element remained untouched while this sub-iteration, some approaches were tried in order to achieve this performance improvement:

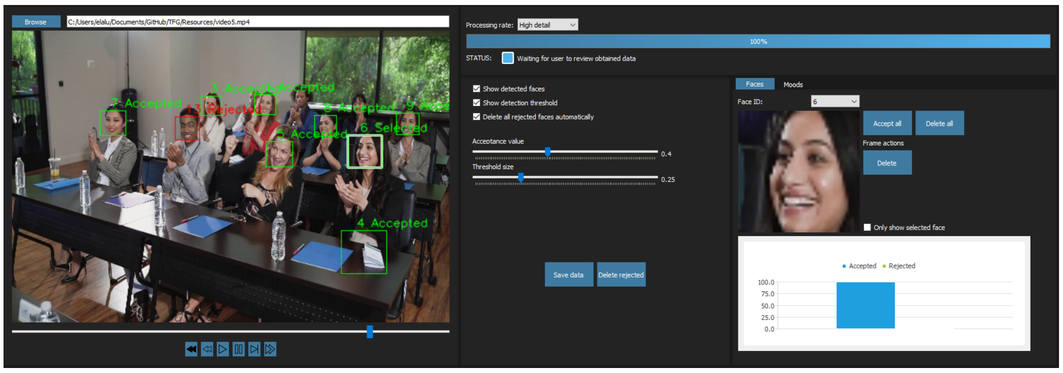
  1. **Application of parallelism:** this resulted in failure due to the lack of low level system calls (the project was developed on windows) like *fork()*, the *multiprocessing* library was

used in order to try to achieve a higher parallelism level but the coordination of the threads to access the model instance resulted in higher execution times, an approach of creating several instances of the model was implemented but resulted on a memory management problem due to the amount of memory required by the CUDNN library provided by CUDA, the option of adding more parallellism to the application was discarded when more knowledge about how the CUDA library works was achieved (the CUDA library assigns frame pixels to each GPU core so there was already a concurrent approach achieved when using CUDNN).

2. **Changing the format of the face isolation module output file:** this doesn't refer to changing the json format of the file, this refers to the way the data was set in the json, in order to try to achieve a emotion detection module that didn't required the video source to be open by passing to the json the complete cropped image of the face and its corresponding id and frame, this approach produced oversized json files and wasn't as good as it sounds, the performance was almost the same but the times taken by the face isolation module to produce the output increased exponentially.
3. **Removing the video preview in the processing phase:** this approach lighten the emotion detection module memory usage and increased performance, so this approach was applied in the final application.
4. **Splitting the review stage and the processing stage:** after the video was processed the review phase of this module kept all the processing libraries open, including the model, the approach of splitting both stages made the application more light keeping the high performance demanding stage of processing into a momentary period were the user is only able to receive feedback from the application.

After all this approaches the emotion detection module saw an medium performance improvement but as this sub iteration was taking so long the decision of stopping it and continuing in other iteration to increase the overall performance and improving the user interface was made.

- **Iteration/phase 3.5.5, 2nd refactoring to increase overall performance** In this sub iteration slight changes to the way the data was processed and the way the threads of the application were controlled were made, the overall performance increased giving the application with a more "smooth execution" feeling, also the style guide for the user interface was applied (Figure 6.11) and the responsive aspect of the application was implemented.


C:/Users/elalu/Documents/GitHub/TFG/Resources/video5.mp4

Processing rate: High detail

100%

STATUS: Waiting for user to review obtained data

☑ Show detected faces
☑ Show detection threshold
☑ Delete all rejected faces automatically

Acceptance value 0.4

Threshold size 0.25

Face ID: 6

☐ Only show selected face

Figure 6.11: The application of the style guide took place in this sub iteration

6.1.5. Iteration/phase 4, integrate charts in the application to allow detailed data analysis

With this iteration came another big design choice, the library to plot the data in the user interface, initially *matplotlib* was used, the integration of the matplotlib into the Qt views was easy but came with a disadvantage, *matplotlib* uses a *matplotlib.Figure* instance to make the plot over the layout, this instance was unique and when trying to plot data while processing the existence of multiple plots required the coordination of this *Figure* instance between the data representation threads.

The *Figure* instance coordination was the most complex implementation in this iteration but there was another disadvantage, the performance of *matplotlib* threads were low when trying to plot frame by frame data and the real time plotting seemed a little snappish, this made *matplotlib* to be discarded as the library used for data analysis.

With a more complex implementation but an exponential performance increase came the *QCharts* library which was chosen as the final application data representation library.

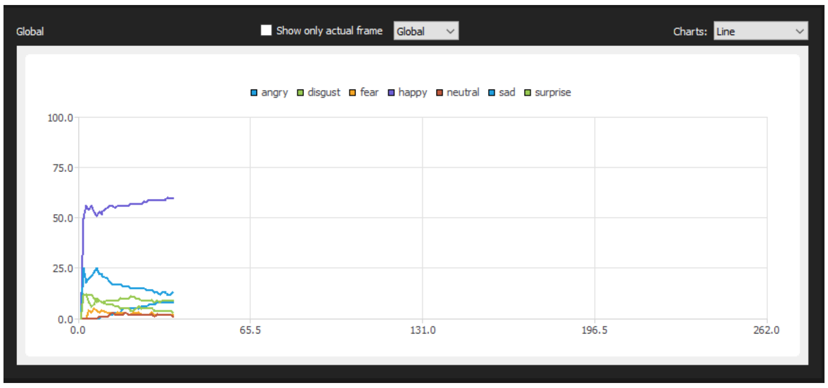


Figure 6.12: The global representation using QCharts library while the data is being processed

This choice allowed the application to have independent threads for data representation and multiple views for the data that was being processed, it also came with a very smooth execution feeling in terms of feedback as the charts were being plotted while the data was being processed (Figure 6.12 and Figure 6.13).

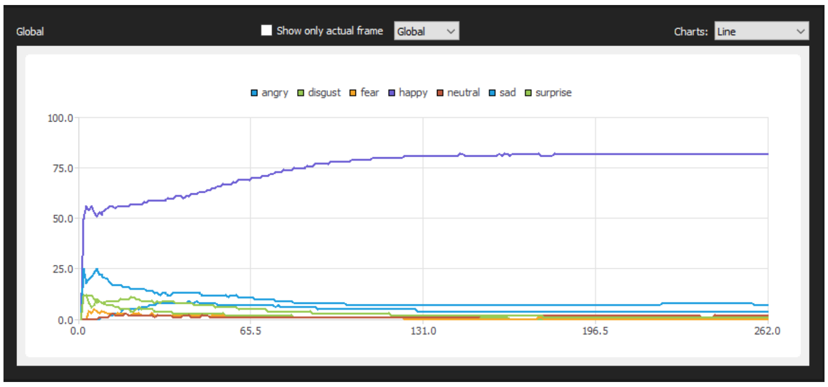


Figure 6.13: The global representation using QCharts library when the data has been processed

The detailed chart view works in the same way but with the data of each frame that has been processed, the chart views are able to plot the data so fast that the execution loops of these threads require to be controlled in terms of execution times with the help of the *time.sleep()* function, in the final application a 0.15 second delay is applied to this loop, this delay can be lowered to achieve more than 60 frames per second (regarding the real time plotting) but as the purpose of this module is to plot charts and this plotting depends on the execution time of the processing modules there is no need for achieving 60 frames per second and having a 4 frame per second rate (0.15 seconds between each plot) seemed smooth enough for the final application.

We have already seen the face picker utility chart, here is a screenshot (Figure 6.14) of the main charts when the data has been already processed by the emotion detection utility:

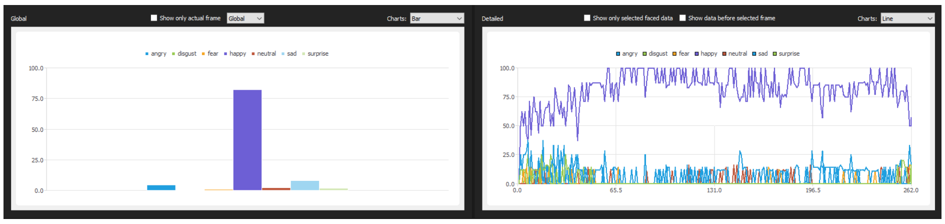

Global
☐ Show only actual frame
Global
Charts: Bar
angry disgust fear happy neutral sad surprise
100.0
75.0
50.0
25.0
0.0
Detailed
☐ Show only selected faced data
☐ Show data before selected frame
Charts: Line
angry disgust fear happy neutral sad surprise
100.0
75.0
50.0
25.0
0.0
0.0
65.5
131.0
196.5
262.0

Figure 6.14: Main chart views of the application

6.2. PROJECT STATS

This section will summarize the statistics regarding the project development, the data that will be shown is taken directly from the *GitHub* repository the project was developed, these statistics refer to the commit amount, the programming language used, the activity and the branches that were created during development.

As seen in the *methodology* chapter the development took from September 2021 to May 2022 with some additional development in June 2022, the activity graph regarding the commits that were made during this period of time according to *GitHub* website is as Figure 6.15 shows:

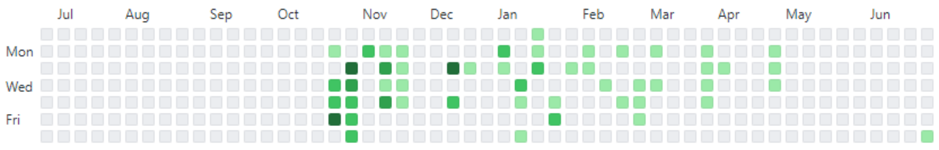


Figure 6.15: Activity according to *GitHub*

were most of the commits were local repository commits and when a relevant feature was obtained it was committed to the remote repository branch, according to *GitHub* a total of **60 commits** were made to the remote.

Overall overview of these statistics can be seen in the *GitHub* repository and are displayed like in Figure 6.16:

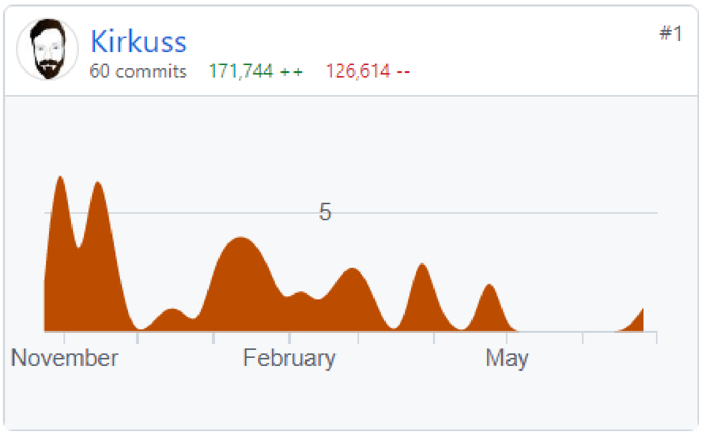


Figure 6.16: Overview of the main committer activity

The branches created in repository followed the methodology specification in order to develop a module per branch, naming of the branches was based on the corresponding phase.

According to *GitHub* the totality of the code was implemented using the python programming language (Figure 6.17), the .ui file regarding user interface uses XML but this file uses the .ui extension and its not recognized as a XML file by *GitHub*.

The total amount of code lines written can be obtained by executing the *find . -name '*.py' | xargs wc -l* command which requires a Linux system to be executed but in windows we have the *Git Bash* utility that allows the execution of the command in the project's folder, executing this command gives a total of **2906** code lines regarding python programming language.

Figure 6.17: GitHub determines that the totality of the project was developed in Python

6.3. BUDGET ESTIMATIONS

In this section we will discuss the budget estimations for this project's development in the case of this project the development team was conformed by only one person that fulfilled the roles of **user interface designer and programmer** the amount of time employed in the project development is difficult to estimate but we can say that a mean of 2.5 hours were employed daily (except weekends), there were days when development was stopped due to external situations and days were the entire available time was employed, but for estimation purposes lets state that a mean of 2.5 hours were expend in daily development during all the project development cycle.

With that said lets take a look on mean salaries for a python developer and a UI designer are in Spain nowadays, we cant forget about including the development hardware equipment which is composed by the system described in the *tools* section of the *methodology* chapter:

- **Python developer:** The mean annual gross salary is 32.000€ for a python developer in Spain, which translates to a net monthly salary of 1700€ and approximately 9€ per hour.
- **UI/UX designer:** The mean annual gross salary is 23.000€, being this approximately 8€ per hour.
- **Development infrastructure:** the amount of money spent in the home pc the project was developed in was around 1500€ taking into account hardware peripherals.
- **Electricity costs:** nowadays electricity costs are important and very relevant to take into account, the mean electricity invoice for one person in Spain is 70€ each two months.

Taking the 2.5 hour per day without weekends lets take the *Methodology* scheme of development to estimate the amount of hours spend while developing the project to estimate the following statements:

1. **from September to January:** period devoted to develop the internal logic of the program, this means time spend as Python developer.
2. **from February to March:** this time period was spend as UI/UX designer.
3. **from April to start of June:** time period were both specializations were working together.

So all these considerations leads us to the following cost estimation for this project:

Resource	Time	Cost
Python developer	325.5 hours	2929.5€
UI/UX designer	157 hours	1256€
Hardware		1500€
Electricity	9 months	315€
TOTAL		5996.5€

CHAPTER 7

Conclusions

In this final chapter of the document some final thoughts, future work approaches and a more personal section regarding final thoughts will be shown, in overall, the development of this project was hard in terms of acquiring knowledge and the global context were the project was developed.

7.1. CONCLUSIONS

In general terms the main objective of the system was fulfilled with all the sub objectives implemented accordingly to the specification although the window left open by the model training aspect that could be better in terms of the model being more accurate and performance friendly.

Lets review the sub objectives established in *Chapter 2* and specify the way these sub objectives were solved:

1. **Acquire knowledge of deep learning techniques to be able to generate a model for image classification:** At the end the model was generated and a lot of new knowledge was acquired by using online guides, documentation of the libraries used and a lot of trial and error, the obtained model had a 63% accuracy after validating with test data and produced acceptable results in the system when it was implemented, the only problem was the architecture and its aim for low memory systems in order to deal with the memory management problems that occurred while training.
2. **Define which technologies will be used in order to achieve the main objective and establish a proper project planning:** the planning part of this objective was the hardest one due to the time restrictions regarding this project, as shown in the *Methodology* chapter, the modularity of the system allowed some work to be done in parallel, this accelerated the development and helped fulfilling time restrictions.
3. **Create a user friendly system:** The use of Gestalt and usability principles was easy due to being familiar to the developer, also creativity was on the side of the project which allowed to get a pretty decent and good looking user interface, it is true that there is always room for improvements but the user interface can wait while the other internal components are improved.
4. **Establish a style guide to make an unique system:** The color palette and style guide of the user interface took some inspiration from other systems like *Adobe Premier*, *Youtube*, *Windows Media Player* and some more, but the design process always aimed to produce an unique style which can be seen in the final user interface.
5. **Use principles of parallel and concurrent programming:** As a highly performance demanding system this objective was considered as critical, but it can be said that regarding the room for improvement left in the project were most part corresponds to concurrent execution improvements this objective was partially fulfilled, signals and locks are powerful techniques but a more complex threading system could be implemented in the future to improve, above all the other components, the emotion detection module.

6. **Give the user a high level of feedback from the system:** The implementation of the video player, the different charts and the feedback section with the status text and the progress bar made the application to not only provide a big amount of feedback to the user but to feel very dynamic and alive.

To end this section lets talk about the design choices regarding the tools used in the project, the set of tools that was chosen for the project development required a lot of knowledge to be useful, the Python programming language is simple at a start but comes with great complexity when we go deeper in it, at the end it was an amazing tool for developing a machine learning system, the tools regarding machine learning TensorFlow and Keras were complex from the start but allowed the generation of the emotion detection model, although the use of CUDNN was required and a lot of thinking was involved in this training process, in general the final results given by the application keeping in mind the knowledge base of the programmer justifies the decision of using this set of tools.

Due to time restrictions there are a lot of open windows for improvements and original functionality ideas, these improvements will be discussed in the next section.

7.2. FUTURE WORK

As said in previous sections there are a lot of improvements that could be made on the system's functionality, lets summarize them:

- **Overall performance:** the performance of the system is acceptable until the emotion detection module execution comes, improvement efforts were made in order to improve the user experience regarding feedback and system stalls along with memory usage the system required in this module's execution but its impossible to hide the amount of improvement that this module requires in terms of internal performance, some modifications to the model architecture could be made along with a more light approach in algorithmic terms regarding the implementation of this module.
- **Refactoring:** the architecture of the system have a lot of flaws regarding some aspects, in general terms there are functions that could be moved to another class in order to act as a set of tools for processing support, actually in the system there is the file *Utilities.py* which contains a set of tools, these functions that are repeated in different classes but with slight changes on its functionality could be moved to this *Utilities* file.
- **Data representation:** a lot of options like filtering the data, more types of plots and a more custom view of the data were discarded due to time restrictions.
- **Slight changes on responsive behaviour:** some sections of the user interface have problems when resizing, this is because the knowledge base about *QT Creator* IDE way of managing layouts, anyway the responsive behaviour of the system its there but making slight changes in it could improve the user experience of the system.

Original ideas from the starting concepts of the system that couldn't be implemented due to time restrictions are now shown as future functionalities that could be implemented:

- **processing of a real time video source:** this refers to for example using a webcam or a camera to send the system a constant feed of video data that is being recorded at real-time, the sequential execution of the face isolation module and emotion detection module makes this option impossible and a big change is required to achieve this functionality, a change that requires the face isolation module and the emotion detection module to work in parallel (remember the memory management problems?), this functionality was meant to be in the original design of the system.
- **Provide more inputs for data processing:** usage of hardware like hearth rate analyzers or add a model for pose detection of the attendants was thought to be in the original design,

these ideas were discarded due to the global pandemic (which affected directly to hearth rate analyzers idea) and the time restrictions (2 week were required to train an emotion detection model).

- **Cloud platform/Web application:** the nature of the system makes it very hardware dependant, this dependency could be solved through the use of a web system with the required infrastructure to act as a platform that depends only on the video source and data files provided by the user.
- **Attention comparator utility:** a functionality that allows the user to compare previous data obtained by the system to new data that is being obtained.
- **Options:** there is a good set of options already in the system but some more options could be added such as modifying the output labels of the emotion detection module in order to ignore particular emotions and much more.

7.3. SPECIALIZATION COMPETENCES FULFILLMENT

This section will be used to justify the fulfilled competences regarding the specialization of *Information Technology*:

- *Capacity of understanding the organization's environment and its needs in the context of information technologies and communications.*
  This competence refers directly to the identification of the problem when designing the system, in this case the environment was originally based on the information regarding the UCLM infrastructure, every teacher nowadays have a computer and the usage of cameras to record the attendants during a presentation is available through webcams or the resources C:TED has, understanding this environmental context guided the design process of the system.
- *Capacity of employing user based methodologies and the development, evaluation, application and systems based on information technologies that ensure accessibility, ergonomic and usable systems management.*
  One of the priorities of this systems was to apply the Usability and Gestalt principles to design the user interface having always the user in mind.
- *Capacity of understanding, applying and managing the system's guaranty and security.*
  This competence was fulfilled when the process of designing the user interface and controls of the internal system logic had the final user actions in mind, in order to prevent user errors the developer must think about the system from the perspective of a final user, that's exactly the approach that was taken when designing the system.

7.4. FINAL THOUGHTS

Lets have a more personal section to talk about my final thoughts about this project, first of all let me use this section as a relief method for myself about the development period of this project, in *2019* Carlos Gonzalez Morcillo (former UCLM professor) came to me with this project's idea as i was working in C:TED (UCLM), a place were we recorded teaching videos and produced multimedia resources for the UCLM.

I saw this idea as a great opportunity for me and my professional development along with a great sense of pride due to working with a teacher i personally admired, this fulfilled me with motivation about the project but then the global pandemic came.

With the pandemic the restrictions came and made impossible to put in practice all the sample gathering for this project's testing phase so a lot of ideas like the hearth rate monitoring or real time attention detection using the C:TED infrastructure were discarded this fact along with additional personal problems that came in this period made me lose all the initial motivation and let me on a state that bordered serious psychological problems.

All of that made me even considering if my studies were on the right direction and almost quitting but life goes on and thanks to the people that surrounded me i was able to find motivation again and take this project seriously once again until now were i see myself writing this final section of my bachelor dissertation.

The development was hard, i decided to specialize myself in information technology were the competences are far away from the computing specialization which most of this project aspects are and made me move out of my comfort zone so i had to study some of this specialization competences in order to understand how deep learning works which required a lot of time.

Also my Python programming language knowledge base was very basic and that's something that can be noticed in the code the classes made at the beginning have more coding mistakes than the classes that were implemented the last and in my opinion with a little more time i could have polished the system's internal code and architecture with a big refactoring which I'm sure i will overtake in the future.

Personally i feel very proud of the system i have made, all the learning process, the great satisfaction moments when some big error was stopping the system execution and solving the error like solving a big puzzle while reading tons of documentation in order to understand why the error was occurring and looking now at the user interface remembering the headache it was to distribute all the elements correctly to see that after all the hours expend it makes the system look as a whole are things that made me understand why i love this profession.

Bibliography

[1] Vishal Rajput. Face detection and recognition capable of beating human beings using facenet. URL: https://www.analyticsvidhya.com/blog/2021/06/face-detection-and-recognition-capable-of-beating-humans-using-facenet/, 2021.

[2] recfaces. Emotion recognition: Introduction to emotion reading technology. URL: https://recfaces.com/articles/emotion-recognition, 2021.

[3] Michael Tupek. Speech recognition using artificial intelligence. URL: https://scionanalytics.com/speech-recognition-using-artificial-intelligence/, 2021.

[4] National Institute of Mental Health. Autism spectrum disorder (asd). URL: https://www.nimh.nih.gov/health/statistics/autism-spectrum-disorder-asd#:~:text=Prevalence%20of%20ASD,-Prevalence%20data%20for&text=Across%20the%20CDC%20surveillance%20sites,all%20racial%20and%20ethnic%20groups., 2022.

[5] Erin Digitale. Google glass helps kids with autism read facial expressions. URL: https://med.stanford.edu/news/all-news/2018/08/google-glass-helps-kids-with-autism-read-facial-expressions.html#:~:text=The%20researchers%20named%20the%20new,depicting%20faces%20with%20different%20emotions., 2018.

[6] Yang Lu Caiming Zhang. Study on artificial intelligence: The state of the art and future prospects. URL: https://www.sciencedirect.com/science/article/abs/pii/S2452414X21000248, 2021.

[7] Bhaskar Mondal. Artificial intelligence: State of the art. URL: https://www.researchgate.net/publication/337400888_Artificial_Intelligence_State_of_the_Art, 2020.

[8] Zhongzhi Shi. Intelligent robots. URL: https://www.sciencedirect.com/topics/computer-science/intelligent-robots, 2021.

[9] Unknown. Intelligent robots. URL: https://esi.uclm.es/index.php/2022/05/26/robots-afectivos-para-ayudar-a-personas-mayores-o-dependientes/, 2022.

[10] edsrobotics. Visión por computador: qué es, objetivos y aplicaciones. URL: https://www.edsrobotics.com/blog/vision-computador-que-es/, 2022.

[11] Vihar Kurama. Ml-based image processing. URL: https://nanonets.com/blog/machine-learning-image-processing/, 2021.

[12] Michael Middleton. Deep learning vs. machine learning — what's the difference? URL: https://flatironschool.com/blog/deep-learning-vs-machine-learning/#:~:text=Deep%20learning%20is%20a%20type,modeled%20on%20the%20human%20brain., 2021.

[13] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018.

[14] cleventy. Qué es git flow y cómo funciona. URL: https://cleventy.com/que-es-git-flow-y-como-funciona/, 2022.

[15] Santander Universidades. Python: qué es y por qué deberías aprender a utilizarlo. URL: https://www.becas-santander.com/es/blog/python-que-es.html, 2021.

[16] Python. Pyqt5: Threading, signals and slots. URL: https://wiki.python.org/moin/PyQt5/Threading%2C_Signals_and_Slots, 2019.

[17] Ramsés Moreno. Principios de gestalt en el diseño de interfaces de usuario. URL: https://www.uxdiario.com/blog/principios-de-gestalt-en-el-diseno-de-interfaces-de-usuario, 2021.

[18] Ringa Tech. Tu primer clasificador de imágenes con python y tensorflow. URL: https://www.youtube.com/watch?v=j6eGHROLKP8&ab_channel=RingaTech, 2021.

[19] Guilherme Duarte Marmerola. Training models when data doesn't fit in memory. URL: https://gdmarmerola.github.io/big-data-ml-training/, 2020.

[20] DIPAYAN MUKHOPADHYAY. Train keras model with large dataset (batch training). URL: https://medium.com/analytics-vidhya/train-keras-model-with-large-dataset-batch-training-6b3099fdf366, 2019.

[21] Unknown. How to set steps_per_epoch,validation_steps and validation_split in keras's fit method? URL: https://androidkt.com/how-to-set-steps-per-epoch-validation-steps-and-validation-split-in-kerass-fit-method/, 2021.

APPENDICES

APPENDIX A

A more detailed overview of the tools employed on this project

A.1. PYTHON

Python is a high level, interpreted and general purpose programming language, being interpreted means that it is not necessary to compile it, it is executed by the interpreter of the computer instead (it is not necessary to "translate" it to machine code).

Figure A.1: Python logo

This programming language was conceived in the late *1980s* by Guido Van Rossum as a successor to the ABC programming language.

It is known as an easy to read and write programming language due to its high similitude with human language and its an open source multi-platform programming language that has been gaining popularity over the years since its creation.

The most important feature and the main reason why i chose Python as the programming language for AIWake is the fact that Python eases working with artificial intelligence, big data volumes and machine learning.

The following is the core philosophy of Python defined with the use of aphorisms:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Nowadays python is in its *Python 3.9* version and is in the top ten of popular programming languages well known organizations like *Wikipedia, Google, CERN, Amazon* and *Spotify* use Python.

A lot of frameworks, libraries, web applications and also video-games use Python as their main source of power.

Indeed some of this python powered elements have been used in this project and we will discuss them later.

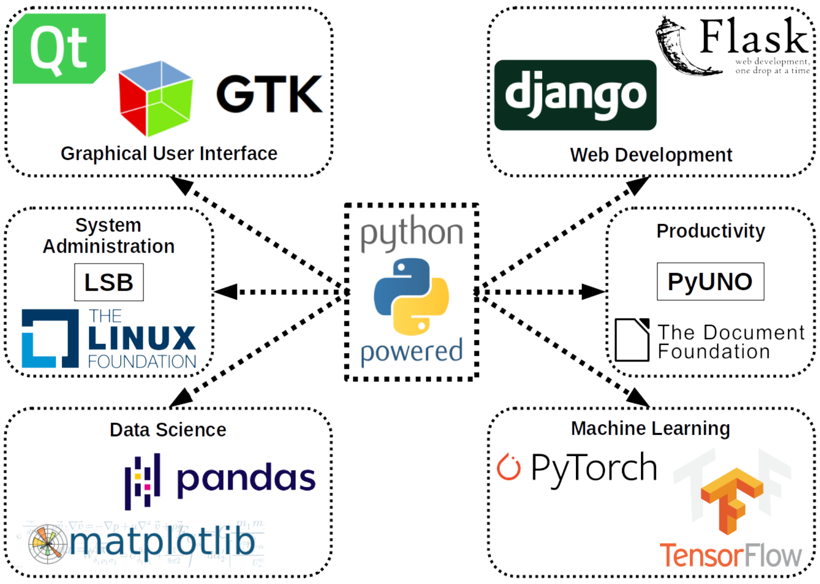


Figure A.2: Main frameworks, libraries and projects that use Python (Python powered)

A.2. QT

An object oriented multi-platform framework mostly used in the development of software that needs a user interface, it is also used for console (server-side) applications and command line programs.

Qt its an open source software developed by the Qt project nowadays a part of Nokia and Trolltech (Norwegian company) are part of the Qt project, Qt is used in KDE Plasma which is a GNU/Linux desktop environment.

Originally a library developed by Trolltech that wasn't totally free, it was actively used between *1996* and *1998* in the development of KDE with great success, this fact made Qt to be seen as a threat against the GNU project (Qt wasn't totally open source) which made the GNU project to start developing GNOME with GTK+ and the development of a open source library called Harmony that was totally compatible with Qt.

After the years it was in *2000* when Trolltech released the 2.0 version which included a license change to the Q Public License (open source).

Nowadays Qt has a triple license, GPL v2/v3 which are open source and free software and the QPL license which is private and focused on the commercial applications.

Figure A.3: Qt logo, the framework that powers AIwake UI

Qt has a lot of bindings to be used along with the most powerful programming languages as it was originally developed as C++ libraries, in this project as we are using Python, the bindings *PyQt5* will be used, for the development of UI also *Qt Creator IDE* will be used.

A.2.0.0.1. PyQt5 binding along with Qt Creator, a powerful combination PyQt5 is a comprehensive set of Python bindings for Qt v5, PyQt has more than 35 extensions that enable Python to be used as the development language for Qt, it also allows us to develop using Qt through

Python on iOS and Android applications, this set of bindings also can be embbeded in C++ based applications in order to configure or enhance these applications.

PyQt uses a GPL v3 license that allows the development of propietary applications, to start using it in our applications we just have to run *pip install PyQt5* and import the libraries.

Figure A.4: PyQt allows the Qt library (C++) to be embedded in Python applications

To start using PyQt5 in our application we can import the libraries as follows:

Listing A.1: Example of imports for PyQt5 in order to be used for the AIWake project *(PickerPlotter.py)*

```python
from PyQt5.QtGui import QPen
import variables as config
from PyQt5.QtCore import Qt, QThread, QPoint
from PyQt5.QtChart import QChart, QChartView, QPieSeries, QPieSlice, QLineSeries, QBarSeries, QBarSet, QValueAxis
```

To give an example on how to generate an application lets take a look at the main window code for AIWake UI:

Listing A.2: Example initialization of the AIWake UI *(AIWakeUI.py)*

```python
1  class AIWake_UI(QMainWindow):
2      def __init__(self):
3          super(AIWake_UI, self).__init__()
4          uic.loadUi("AIWake_app.ui",self)
5          self.setWindowTitle("AIWake")
6
7          self.testing = True
8          self.thread = {}
9          self.currentThread = [True, False, False, False]
10
11         self.initUI()
12
13     def initUI(self):
14
15         self.playBt.clicked.connect(self.playBtClick)
16         self.showBoxesCb.stateChanged.
17             connect(self.previewChange_showHB)
18         self.showDetailedCb.stateChanged.
19             connect(self.previewChange_showDE)
20         self.tillFrameCb.stateChanged.connect(self.tillFrame)
21         self.faceDataOnlyCb.stateChanged.connect(self.faceDataOnly)
22         self.pauseBt.clicked.connect(self.pauseBtClick_step1)
23         ...
24     ...
25 ...
```

As we can see in the example the line *1* shows the class declaration which inherits form the QMainWindow class, this class includes all the necessary methods to use the class as a main window for our application, in the line *4* the program loads a .ui file, we will see what this .ui extension means.

In the line *11* a function initUI is declared, all declarations under this functions will be executed at the UI initialization and will establish the mapping of the UI components behaviour through different functions for example in the line *15* we map the *PlayBt* component and its *clicked* event to the function *playBtClick* which is declared as follows in the *AIWake* class code:

Listing A.3: playBtClick function*(AIWakeUI.py)*

```
    def playBtClick(self):
        if self.currentThread[0]:
            if self.thread[1].isRunning():
                self.thread[1].pause = False
            else:
                self.thread[1].start()
            ...
        ...
    ...
```

As simple as it looks, the playBt component will start a thread in our application when clicked.

This example shows how to use PyQt in our application, but i used a little trick in order to achieve a faster development for my application, i'm talking about *Qt Creator*, an IDE created to develop Qt applications through a drag and drop mechanic (similar to the widely used *Visual Studio IDE*) and *QML* statements

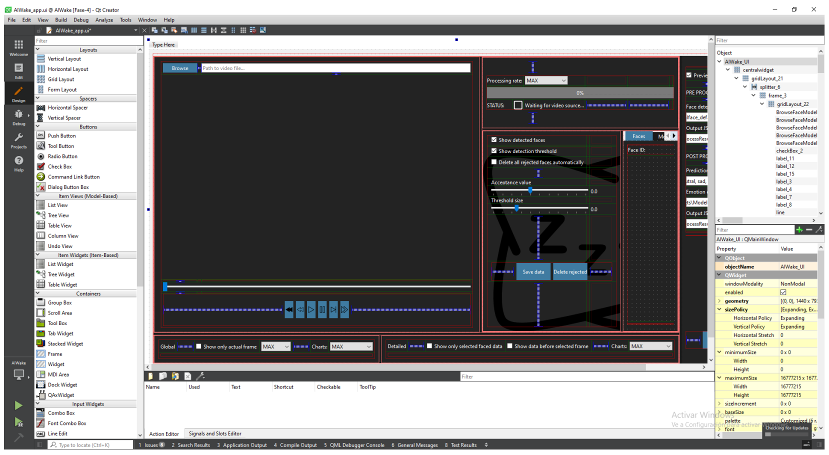
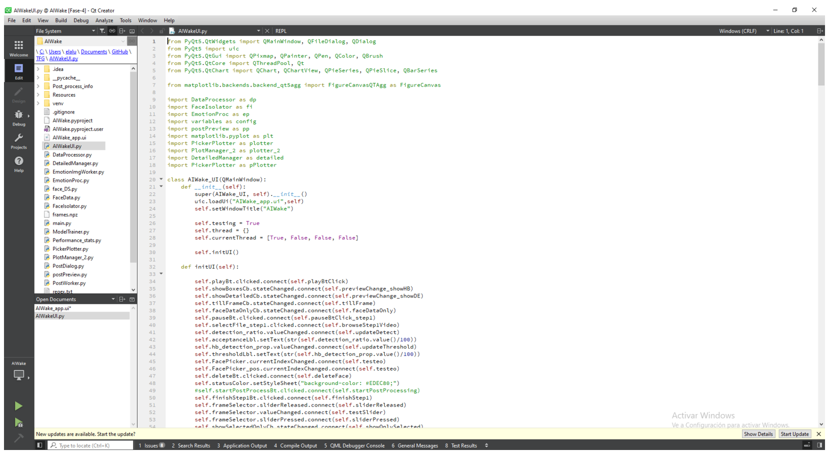


Figure A.5: Qt Creator interface with AIWake being developed, a great IDE for UI development using Qt

Qt Creator as said above is an IDE for development of Qt applications, it uses *QML* which is a language based on *JavaScript* created for the design of user interface focused applications *QML* stands for *Qt Meta Language*, Qt creator is programmed in C++ which makes a need for our application to include the PyQt5 libraries into our Qt Creator instance, it was created by Trolltech also creators of Qt as mentioned above and it offers support for *Windows, MAC OS and GNU/Linux*.

Qt Creator offers an advanced code editor but as the AIWake project will be developed using other libraries the use of another IDE *PyCharm* was required so an IDE for the UI *Qt Creator* that provides powerful tools for UI development and an IDE for the source code *PyCharm* which provides powerful debugging tools were used in the development of this project.



```python
from PyQt5.QtWidgets import QMainWindow, QFileDialog, QDialog
from PyQt5 import uic
from PyQt5.QtGui import QPixmap, QPainter, QPen, QColor, QBrush
from PyQt5.QtCore import QThreadPool, Qt
from PyQt5.QtChart import QChart, QChartView, QPieSeries, QPieSlice, QBarSeries

from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas

import DataProcessor as dp
import FaceIsolator as fi
import EmotionProc as ep
import variables as config
import postPreview as pp
import matplotlib.pyplot as plt
import PickerPlotter as plotter
import PlotManager_2 as plotter_2
import DetailedManager as detailed
import PickerPlotter as pPlotter

class AIWake_UI(QMainWindow):
    def __init__(self):
        super(AIWake_UI, self).__init__()
        uic.loadUi("AIWake_app.ui",self)
        self.setWindowTitle("AIWake")

        self.testing = True
        self.thread = {}
        self.currentThread = [True, False, False, False]

        self.initUI()

    def initUI(self):

        self.playBt.clicked.connect(self.playBtClick)
        self.showBoxesCb.stateChanged.connect(self.previewChange_showHB)
        self.showDetailedCb.stateChanged.connect(self.previewChange_showDE)
        self.tillFrameCb.stateChanged.connect(self.tillFrame)
        self.faceDataOnlyCb.stateChanged.connect(self.faceDataOnly)
        self.pauseBt.clicked.connect(self.pauseBtClick_step1)
        self.selectFile_step1.clicked.connect(self.browseStep1Video)
        self.detection_ratio.valueChanged.connect(self.updateDetect)
        self.acceptanceLbl.setText(str(self.detection_ratio.value()/100))
        self.hb_detection_prop.valueChanged.connect(self.updateThreshold)
        self.thresholdLbl.setText(str(self.hb_detection_prop.value()/100))
        self.FacePicker.currentIndexChanged.connect(self.testeo)
        self.FacePicker_pos.currentIndexChanged.connect(self.testeo)
        self.deleteBt.clicked.connect(self.deleteFace)
        self.statusColor.setStyleSheet("background-color: #EDEC80;")
        #self.startPostProcessBt.clicked.connect(self.startPostProcessing)
        self.finishStep1Bt.clicked.connect(self.finishStep1)
        self.frameSelector.sliderReleased.connect(self.sliderReleased)
        self.frameSelector.valueChanged.connect(self.testSlider)
        self.frameSelector.sliderPressed.connect(self.sliderPressed)
        self.showSelectedOnlyCb.stateChanged.connect(self.showOnlySelected)
```

Figure A.6: Qt Creator advanced code editor with AIWake being developed

A.2.0.0.2. Qt Charts vs Matplotlib i cant end the Qt section without briefly talking about one of the best modules it could have, while developing AIWake i started using *Matplotlib* in order to represent the data obtained during processing of facial emotions.

matplotlib

Figure A.7: Matplotlib,one of the most popular python libraries for data representation

The use of Matplotlib came with a big problem for me while developing *AIWake*, the library wasn't able to support multiple plots at the same time and also wasn't capable of redrawing them efficiently, it is a very simple and easy to use library but it came with a great disadvantage, the poor performance for achieving one big approach of *AIWake*: the real time data representation while processing

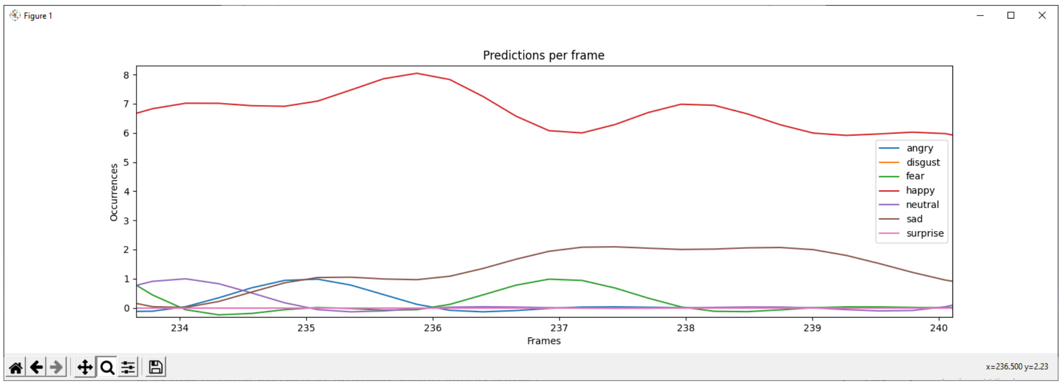
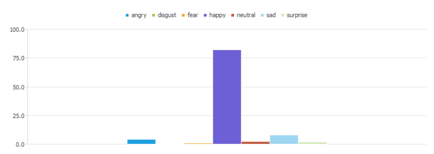
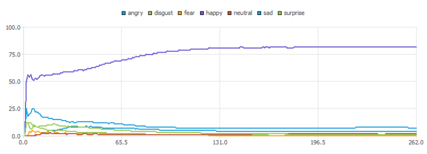


Figure A.8: First tests running matplotlib to represent the processed data by *AIWake*

This lack of performance caused the need of finding a new library for data representation and while looking for such library i found that *Qt* came with a library for data representation, the only disadvantage was the fact that in terms of coding it was far more complex than *Matplotlib* which causes this library to be widely hated by most *Qt* developers, but it was worth a try and it came with great results.



(a) Bar plot using QtCharts



(b) Line plot using QtCharts

Figure A.9: Different data plots using QtCharts in *AIWake*

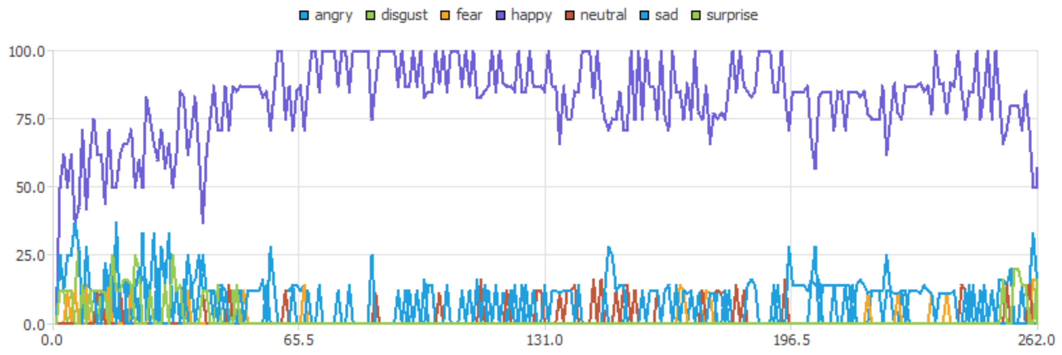


Figure A.10: Detailed line plot of the processed data frame by frame in real time by *AIWake*

So at the end *QtChart* was chosen to be the data plotting engine of *AIWake* due to its low impact in performance and capacity to run along with the rest of processes of the main application (even in different threads Matplotlib impact on performance was significant due to the limited resources a home pc can have).

A.3. PYCHARM

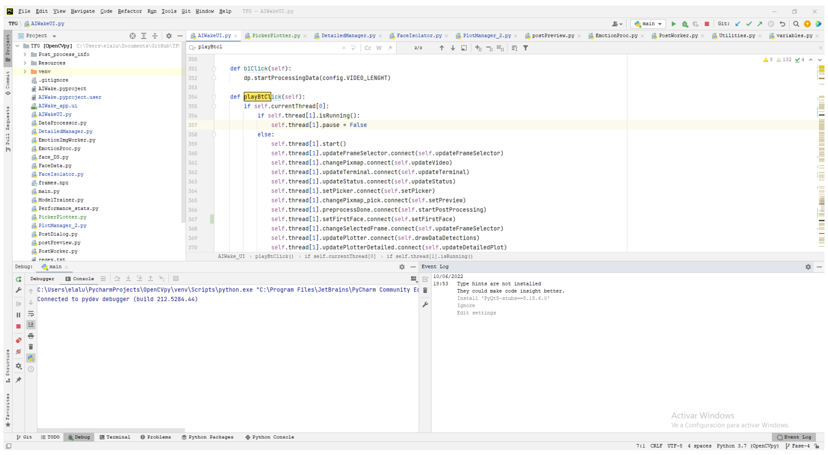
PyCharm is a multi-platform IDE developed by the czech company *JetBrains* for the *Python* programming language that offers a community editions with as its website says *all the python tools just in one place.*

Figure A.11: PyCharm, a powerful tool for python development

PyCharm offers advanced code analysis, a graphic debugger and integration with version control systems like *Git*

It was first released in *2011* and *JetBrains* has been releasing new versions of the IDE since then, at the moment of developing this project the last version of *PyCharm* was *2021.3* and a new version is about to come as 3 versions are released each year usually in a 3-month period.

Its well known that nowadays *PyCharm* in industries is used by most of the professional developers and it has been considered as the best IDE for *Python* so the list of actual applications that have been developed using this IDE is so long that it wouldn't fit this document.



```
    def b1Click(self):
        dp.startProcessingData(config.VIDEO_LENGHT)

    def playBtClick(self):
        if self.currentThread[0]:
            if self.thread[1].isRunning():
                self.thread[1].pause = False
            else:
                self.thread[1].start()
                self.thread[1].updateFrameSelector.connect(self.updateFrameSelector)
                self.thread[1].changePixmap.connect(self.updateVideo)
                self.thread[1].updateTerminal.connect(self.updateTerminal)
                self.thread[1].updateStatus.connect(self.updateStatus)
                self.thread[1].setPicker.connect(self.setPicker)
                self.thread[1].changePixmap_pick.connect(self.setPreview)
                self.thread[1].preprocessDone.connect(self.startPostProcessing)
                self.thread[1].setFirstFace.connect(self.setFirstFace)
                self.thread[1].changeSelectedFrame.connect(self.updateFrameSelector)
                self.thread[1].updatePlotter.connect(self.drawDataDetections)
                self.thread[1].updatePlotterDetailed.connect(self.updateDetailedPlot)
```

```
C:\Users\elalu\PycharmProjects\OpenCVpy\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Ed
Connected to pydev debugger (build 212.5284.44)
```

```
10/06/2022
18:53   Type hints are not installed
        They could make code insight better.
        Install 'PyQt5-stubs==5.15.6.0'
        Ignore
        Edit settings
```

Figure A.12: PyCharm UI during the execution of AIWake

PyCharm is the most popular tool for *Python* development with (according to the *2018 python developers survey by the Python software foundation together with JetBrains)* a 35% combined share for *PyCharm Professional and Community editions.*

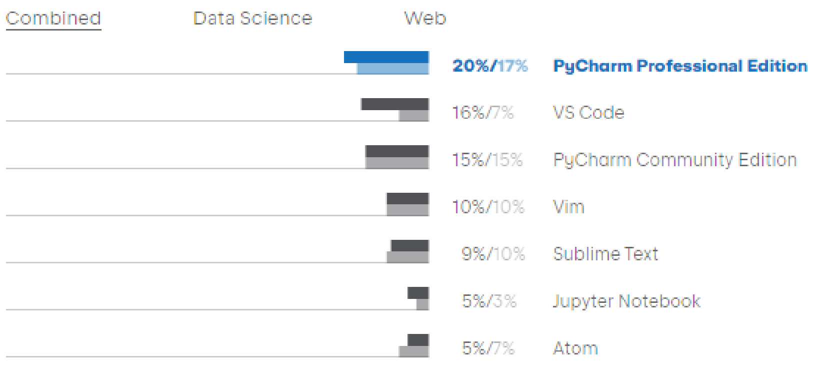


Figure A.13: *2018 python developers survey* results

The popularity of this IDE has gifted it with an immense amount of guidelines, tutorials, plugins and facilities in general that has made the choice of this IDE for the *AIWake* project development an easy one.

A.4. OPENCV

OpenCV is an open source computer vision and machine learning software library built to be one common infrastructure for computer vision applications.

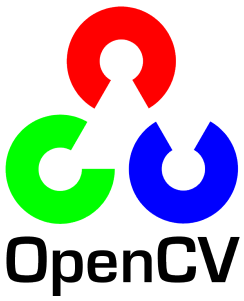


Figure A.14: OpenCV, the library used for the computer vision component of AIWake

Many companies make an extensive use of *OpenCV* such as *Google, Yahoo, Microsoft, Intel, IBM, Sony* and many more along with many startups that use *OpenCV* for developing surveillance software, robots navigation and interactive art applications in Spain, these are just a few examples of the *OpenCV* nowadays applications.

The use of *OpenCV* in this project is needed due to the fact that a lot of images will be processed through the data model and in order to prepare these images an *OpenCV* integration is needed.

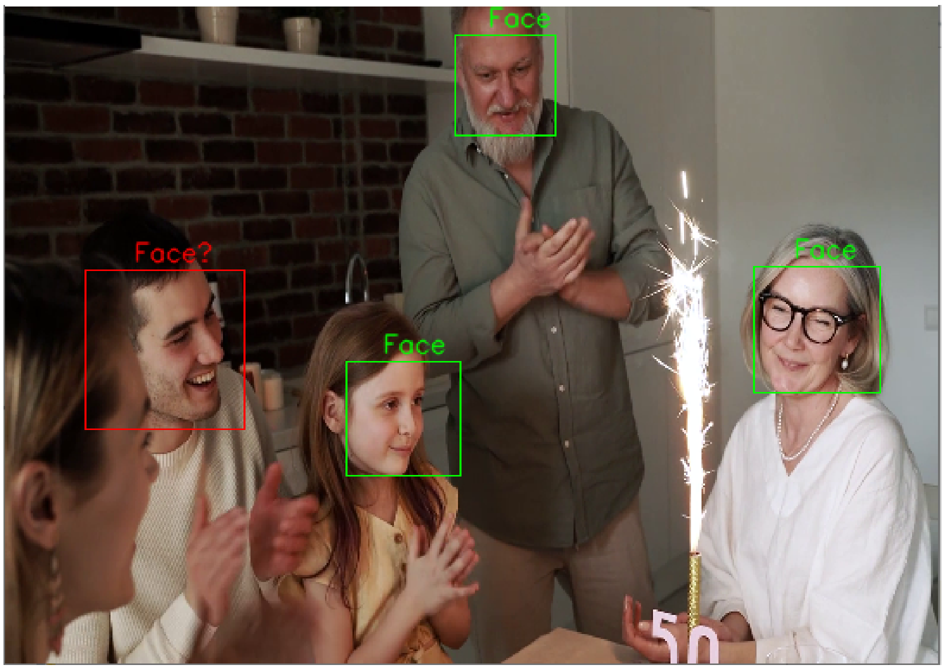


Figure A.15: An image generated by OpenCV in AIWake first runs

Python-OpenCV provides a set of bindings in order to use the *OpenCV* library along with *Python* in A.15 we can see one of the first test runs in *AIWake* that along with the use of a model provided by *OpenCV (haarcascade)* for face detection allowed the first impletations of the *AIWake* project.

The model provided by *OpenCV* is part of a set of data models that *OpenCV* contains due to its computer vision oriented characteristics, this model was used for the final version of *AIWake* as it was already a powerful model for face detection.

Python-OpenCV makes use of **numpy**, a highly optimized library for numerical operations, all the OpenCV array structures are converted to and from **numpy** which makes it easier to integrate *Python* libraries that make use of **numpy**.

Also mention that compared to other languages like *C/C++*, *Python* is slower (with *OpenCV*) so the *Python-OpenCV* bindings are wrappers for *Python* that use the *C/C++* code meaning that we have a code that is as fast as it is in *C/C++* and we have an easier language to code which is *Python*.

A.5. TENSORFLOW

Tensorflow is an open source end to end platform for *machine learning* that counts with an integral and flexible ecosystem of tools, libraries and resources coming from the contributions of its community that allows developers and researchers to innovate through the creation of applications using *machine learning*.

TensorFlow

Figure A.16: *Tensorflow*, an open source platform for machine learning

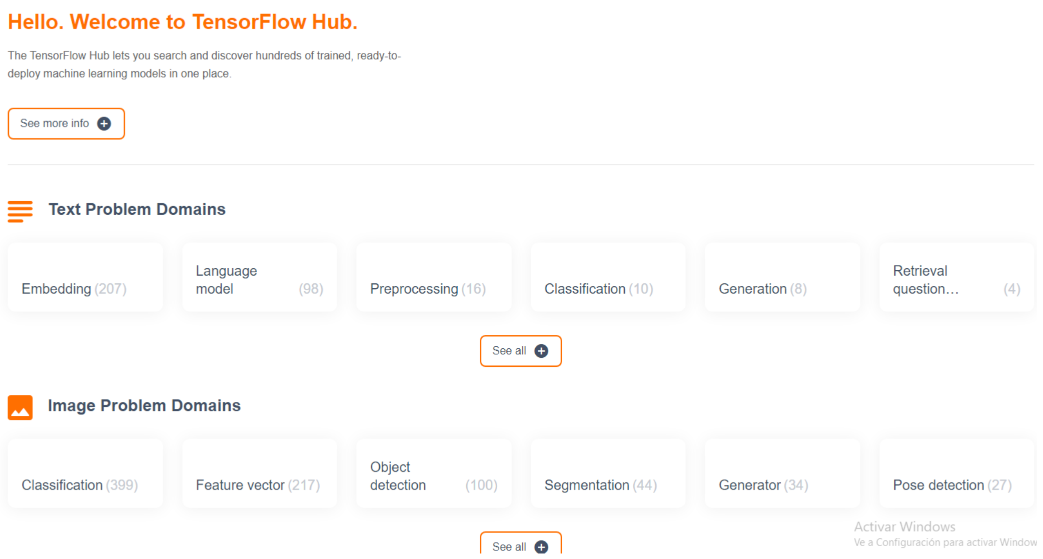
It was developed by *Google* to fulfill their need of creating and training neural networks to detect and decipher patterns and correlations like the human reasoning does, nowadays *Tensorflow* is used in research and in some products.

Originally developed by the *Google Brain* team in order to replace the closed source platform (also developed by *Google Brain*) *DistBelief* as a company private platform till *2015* when it was published under an *Apache* open source license.

Tensorflow is available in *Windows, Linux, macOS* and mobile platforms like *Android* and *iOS* and can run over multiple CPUs and GPUs, for *NVIDIA* GPUs it also can run along using *CUDA* which we will talk about in future sections if this document.

The name *Tensorflow* comes from the multidimensional arrays used by the neural networks operations which are called "tensors" it is important to mention that in *2016* after the *2015* license change of *Tensorflow* about 1500 *GitHub* repositories where mentioning *Tensorflow* and only 5 repositories where *Google's*.

Tensorflow website offers a section called *Tensorflow hub* where the community can post already trained models for different problem domains, in the development of *AIWake* none of these were used but this option already exists as a powerful set of tools for machine learning research and development.



Hello. Welcome to TensorFlow Hub.

The TensorFlow Hub lets you search and discover hundreds of trained, ready-to-deploy machine learning models in one place.

See more info

Text Problem Domains

Embedding (207) | Language model (98) | Preprocessing (16) | Classification (10) | Generation (8) | Retrieval question… (4)

See all

Image Problem Domains

Classification (399) | Feature vector (217) | Object detection (100) | Segmentation (44) | Generator (34) | Pose detection (27)

See all

Activar Windows
Ve a Configuración para activar Window

Figure A.17: *Tensorflow hub*, a repository with an immense amount of pretrained models.

In *2019* the alfa version of *Tensorflow 2.0* was announced which was focused on simplicity and the *eager* execution mode, it also aims to consolidates the use of high level APIs based on *Keras* and the flexible deployment of data model at any platform.

The *AIWake* project uses the *Tensorflow 2.7* version along with the *Keras* API and the *CUDA* library for pararell computing, this combinations allowed the generation of the model for our classification problem and the results will be shown in future sections.

A.6. KERAS

Keras is an API (Application Programming Interface) designed as it website claims *for human beings, not machines*, that follows best practices in order to reduce cognitive load minimizing the number of user actions for common use cases, it also comes with an extensive documentation and user guides.

Figure A.18: *Keras*, an API designed for human beings, not machines

Among the 5-top winning teams on *Kaggle* (we will discuss *Kaggle*) in the next section) *Keras* was the most used deep learning framework due to the fact that its design makes easy to run new experiments empowering the developers to try more ideas against the competitors, since *2017, Keras* counts with more than 200.000 users and was the tenth most quoted tool in the *KD Nuggets 2018* software survey with an average use of 22%.

Figure A.19: *KDNuggets*, a leading platform for AI developers with more than 500.000 monthly users

It is built on top of *Tensorflow 2* which makes it an industry-strength framework scalable to large clusters of GPUs with a vast ecosystem that covers every step of the machine learning workflow from data management to deploy of new solutions.

Originally developed as part of the research efforts of the project *ONEIROS* by the *Google* engineer *Francois Chollet* and supported by the *Tensorflow* team since *2017*, also, *Microsoft* offered a *CNTK* backend to *Keras* which is available since *CNTK 2.0*.

Keras has many impletations of building blocks for neural networks like the layers, goal functions, activation functions and mathematical optimizers, it also offers support for convolutional networks and recurrent ones (*AIWake* uses a convolutional network for solving the classification problem it has to face) and has the option of generating deep learning models for Android and iOS devices.

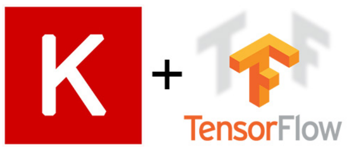


Figure A.20: *Keras + Tensorflow*, the best combo generating data models

Used in *CERN, NASA, NIH* and more scientific organizations, its ease of use makes *Keras* the deep learning solution for most universities courses as it is the most recommended way to learn deep learning.

A.7. NVIDIA CUDA

CUDA is a pararell computing platform and programming model developed by *NVIDIA* for computing on *NVIDIA* GPUs, *CUDA* allows to speed up drastically very demanding computing applications using the power of GPUs, *CUDA* makes the compute intensive portions of programs to be executed in parallel using the thousands of GPU cores available.

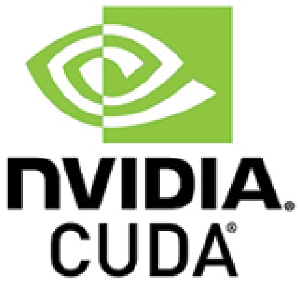


Figure A.21: *NVIDIA CUDA*, a library for systems using NVIDIA GPUs optimized for parallel programming

Some companies like *Adobe, Ansys, Autodesk* or *Microsoft* deploy their applications to GPUs embedded systems, workstations, data-centers and in the cloud, *CUDA* encompasses many domains in terms of highly computing power demanding applications such as computational chemistry, **machine learning**, data science, bio-informatics and fluid dynamics.

As the system where *AIWake* will be developed its running a *NVIDIA 2060 RTX* i decided to integrate the *CUDA* platform into the development of the project which has a great disadvantage: the systems that doesn't run on a *NVIDIA GPU* will experience performance losses as they will have to run all the processing of *AIWake* limited by their CPU processing power, an interesting future approach for the applications would be to run it in the cloud with a powerful system but this approach considerably exceeds the time limitations of the development.

CUDA first **SDK** was released on *2007* for *Windows and Linux* and didn't arrive to *macOS* till its *2.0* version, using *CUDA* can cause bottlenecks because of its aim of taking advantage of the great parallelism level and high bandwidth of GPUs memories instead of accessing multiple times main memory of the system, the structure of *CUDA* model is defined by a grid that is conformed by blocks of threads that have *1024* distinct threads at max, it uses kernels that specify which instructions are going to be executed by each individual thread and can launch all threads in a single block, a block for each thread or launch multiple threads in each block.

A *multiprocessor* contains eight scalar processors, two units specialized in transcendental functions, one multi-thread unit and a shared memory, the *multiprocessor* creates the threads with a planning that allows the synchronizations of barriers and a very light thread creation allowing *CUDA* to be use in very low granularity problems even assigning each thread to a single element in our case of a image (a *pixel*).

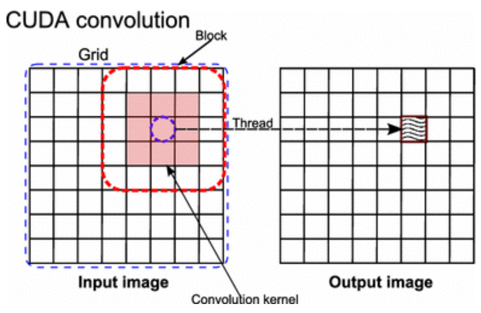


Figure A.22: *CUDA* is able to assing every pixel of a image to a single GPU thread.

A.7.0.0.1. CUDA Deep Neural Network (cuDNN) *cuDNN* is a GPU-accelerated library of primitives for **deep neural networks** that provides highly tuned implementations for standard routines such as forward and backward **convolution**, **pooling**, normalization and activation layers.

Figure A.23: *cuDNN*, useful when accelerating deep learning frameworks

Integrating *cuDNN* with a project allows developers to focus more on training neural networks and developing software rather than spending that time on low-level GPU performance tuning as *cuDNN* accelerates most used deep learning frameworks such as *Keras* and *Tensorflow*.

AIWake makes use of *cuDNN 8.1* in order to accelerate the video processing and take advantage of the *NVIDIA RTX 2060* GPU running in the machine the project is being developed, the last version of *cuDNN* is *8.3* and it brings improvements for A100 GPUs which are GPUs that are built with the A100 tensor core architecture (latest *NVIDIA* GPUs such as rtx 3000 series) in our case RTX 2060 is built with the U100 tensor core architecture so we will use *cuDNN 8.1*.

A.8. KAGGLE

Kaggle is an online community composed by data scientist and machine learning experts that allows users to find and publish data sets and generate models along these data sets, *Kaggle* also offers working along with other data scientists and engineers and participate in competitions of solving data science challenges against other teams since *2010*.

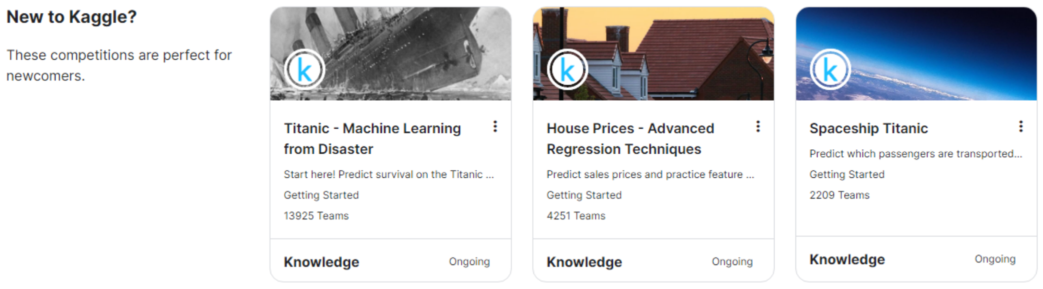
Figure A.24: *Kaggle*, a community of data scientist and machine learning experts

Nowadays the data sets offered by *Kaggle* and the models generated in those competitions are publicly available for anyone to use them as a cloud workbench for data science and Artificial Intelligence education, it was acquired by *Google* as a subsidiary in *2017*.

The number of users or as *Kaggle* refers to them, "Kagglers", is more than a million as said by *Kaggle* website from the rookiest ones to the most experts they come from 194 different countries, the competitions run by *Kaggle* usually attract more than a thousand teams that use the multiple data sets and code fragments, *Kaggle kernels*, available to compete.

Competitions run by *Kaggle* are as follows:

1. A host prepares the data and the problem description.
2. The competitors experiment with the data using different techniques and try to obtain the best model possible model, this work is shared publicly in *Kaggle* in order to allow the come of new ideas, the models are evaluated almost immediately and these qualifications are shown live in a table publicly on *Kaggle*.
3. When the due date is over, the host gives the price to the winners in exchange of a global license, perpetual and irrevocable to use the winning project.


New to Kaggle?

These competitions are perfect for newcomers.

Titanic - Machine Learning from Disaster
Start here! Predict survival on the Titanic ...
Getting Started
13925 Teams
Knowledge Ongoing

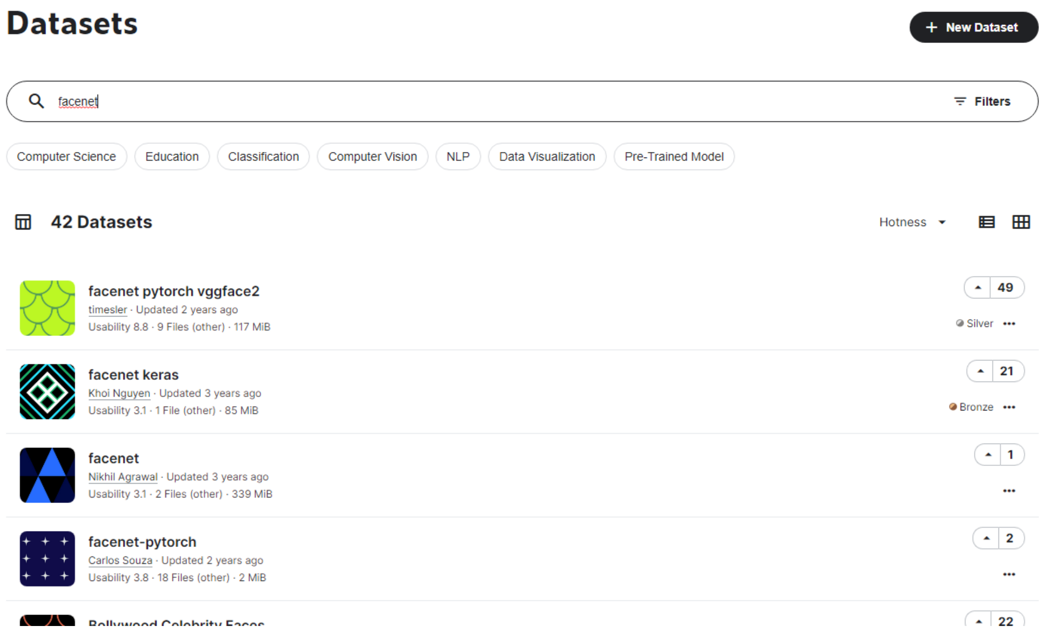
House Prices - Advanced Regression Techniques
Predict sales prices and practice feature ...
Getting Started
4251 Teams
Knowledge Ongoing

Spaceship Titanic
Predict which passengers are transported...
Getting Started
2209 Teams
Knowledge Ongoing

Figure A.25: Some competitions available in *Kaggle* website, 12/06/2022.

The impact of *Kaggle* competitions encompass from improving the pose recognition for *Microsoft Kinect* to improving the search of the *Higgs boson* in *CERN* and some of this competitions have come with great success in for example improving the search for a VIH cure or the rising of new techniques for Artificial Intelligence experiments such as *XGBoost*.

Now lets focus on *AIWake*, for this project a data set was required so a little search on *Kaggle* was made to find out about *FaceNet*, a deep learning model by *Google* that is proven to be the best one in terms of face recognition so the aim of the search in *Kaggle* was to find data sets prepared for the *Google FaceNet* architecture.


Datasets

+ New Dataset

facenet

Filters

Computer Science · Education · Classification · Computer Vision · NLP · Data Visualization · Pre-Trained Model

42 Datasets

Hotness

facenet pytorch vggface2
timesler · Updated 2 years ago
Usability 8.8 · 9 Files (other) · 117 MiB
49 · Silver

facenet keras
Khoi Nguyen · Updated 3 years ago
Usability 3.1 · 1 File (other) · 85 MiB
21 · Bronze

facenet
Nikhil Agrawal · Updated 3 years ago
Usability 3.1 · 2 Files (other) · 339 MiB
1

facenet-pytorch
Carlos Souza · Updated 2 years ago
Usability 3.8 · 18 Files (other) · 2 MiB
2

Bollywood Celebrity Faces
22

Figure A.26: Listing of models published in Kaggle using the FaceNet architecture

In Figure A.26 we can appreciate some projects created by *Kaggle* users, these projects can be filtered by hotness that means we can pick a project that is being watched very closely by other users which may mean that the project is interesting and useful or we can filter by votes which gives us the all-time best projects, also when searching for projects we can specify *Kaggle* the type of input data we want for our model and the file size of the data this model has.

The fact that we are using a home computer for developing *AIWake* made *FaceNet* power requirements a disadvantages so the final decision about architecture was to use *FaceNet* data sets found in *Kaggle* with a less power hungry architecture as *MobileNetV2* with allowed the generation of a model for our classification problem in less time by using *Keras*.

APPENDIX B

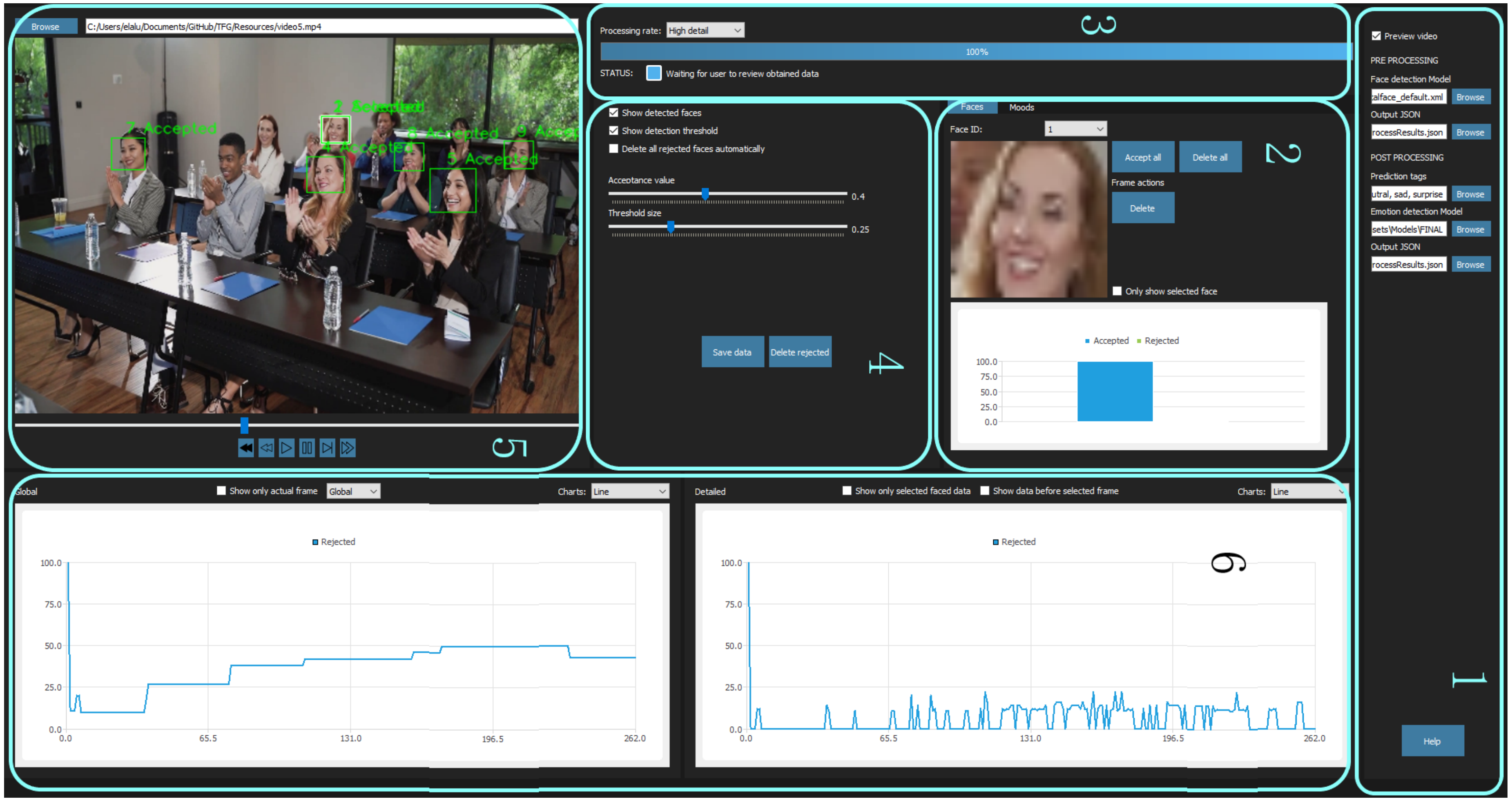
AIWake reference guide

AIWAKE REFERENCE GUIDE

For users and system understanding

AIWake
Platform for the analysis of the level of attention in lectures using computer vision and deep learning

Enrique Valverde Soriano



C:/Users/elalu/Documents/GitHub/TFG/Resources/video5.mp4

Processing rate: High detail

100%

STATUS: Waiting for user to review obtained data

☑ Show detected faces
☑ Show detection threshold
☐ Delete all rejected faces automatically

Acceptance value 0.4

Threshold size 0.25

Face ID: 1

☐ Only show selected face

☑ Preview video

PRE PROCESSING

Face detection Model: alface_default.xml

Output JSON: rocessResults.json

POST PROCESSING

Prediction tags: utral, sad, surprise

Emotion detection Model: sets\Models\FINAL

Output JSON: rocessResults.json

☐ Show only actual frame Global

Charts: Line

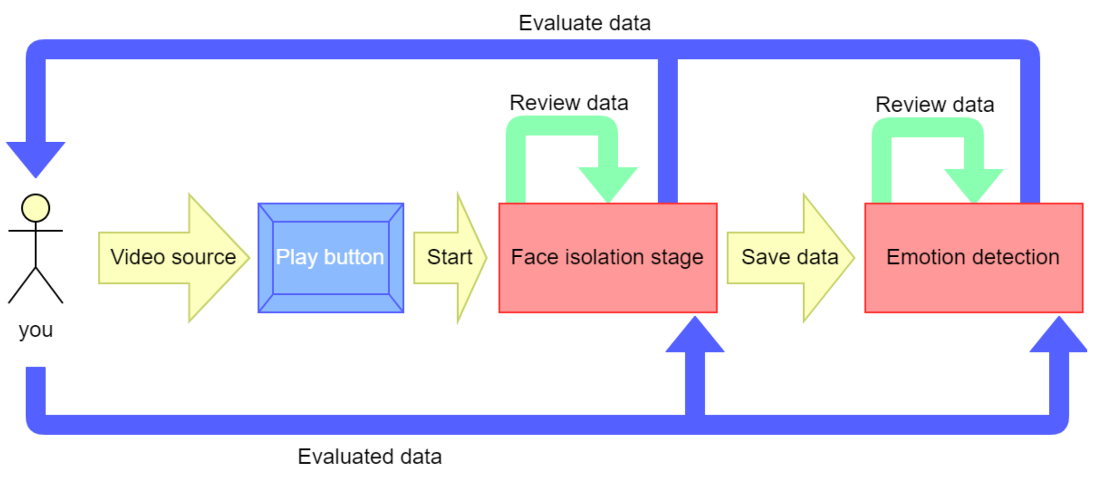
☐ Show only selected faced data ☐ Show data before selected frame

Charts: Line

1. **General options section:** Here you can modify the options regarding the overall configuration of the system, the available options are:
   - **Face detection model:** if you want to use a different model face detection in the face isolation stage, you can browse from your computer the model you want to use, remember that the architecture of the model must be similar in terms of input images processing!.
   - **Output JSON:** you can modify the output json file path in order to store the data processed by the face isolation stage to any location of your computer.
   - **Prediction tags:** if you want to use a custom emotion detection model that uses different labels for emotions, you can specify these labels here.
   - **Emotion detection model:** Opens the possibility of using your own classification models remember that the input parameters must match the input parameters used by AIWake.
   - **Output JSON:** Allows you to chose a new location for the json files produced by the emotion detection stage.
2. **Face picker section:** Here you can manually manage the data produced by the processing stages of the system, there is a tab for each stage, in the image we can see the face isolation face picker tab, in this tab you can delete face frame occurrences, delete all the face occurrences or mark a face as accepted (if you consider a face occurrence as a valid face but the system rejected it), for the case of the emotion detection stage, you can perform the same actions but with the added option of modifying the mood detected by the system, all depends on your criteria.
3. **Feedback section:** Here you can obtain feedback from the system, status text regarding the internal actions that are being executed, the progress bar that gives you the amount of data that has been processed and a selector were you can tell the system to skip some frames in order to obtain a performance improvement with the cost of sacrificing some accuracy, it is highly recommended to use this option only once at the start of the execution in order to obtain a consistent output.
4. **Video player options:** This options mostly belong to the face isolation stage, here you can modify the size of the collision squares (depending on the amount of faces there could be errors if big size squares are used for isolation), the acceptance rate in order to establish a criteria for the system to determine when to consider a detected face as valid, the show detected faces option that draws the areas of the video regarding faces, the show detection threshold option were you make the system to stop showing the collision squares, the delete all rejected automatically option that automatically deletes all the rejected faces at the end of the face isolation stage and the buttons:
   - **Save data:** when the review stage of face isolation is over, this button must be used in order to produce the json output and tell the system to continue with the next stage.
   - **Delete rejected:** if you forgot to select the delete all rejected option or if you want to review all the detected faces that were isolated, don't worry, you can always use this button to tell the system to delete all the rejected faces.
5. **Video player section:** works as a common video player, the controls are the usual play, pause, forward and backward buttons the leftmost and rightmost buttons are meant to chose the next frame that was processed in the case the option of skipping frames was chosen, in the upper section the browser for choosing the video file to process is located.
6. **Data representation section:** This section provides you with real time information regarding the processed data of each stage, the left chart represents global data which provides you with a cumulative representation of the data being processed, in the case of face

isolation it plots the value of the amount of faces that were rejected over time, for emotion detection it plots the labels and its values, the right chart is similar but with the detailed representation (not cumulative, each processed frame is plotted), you can select the type of chart to display between bar chart, pie chart and line chart.

To end the reference guide the action flow that you should follow when using AIWake should look like this:


Evaluate data
Review data
Review data
you
Video source
Play button
Start
Face isolation stage
Save data
Emotion detection
Evaluated data