Department of Information Science and Technology

Open Source Face Recognition API

Diogo Neto Coxinho Mourisco da Conceição

Thesis in partial fulfillment of the requirements for the degree of
Master in Open Source Software

Supervisor
PhD, João Pedro Afonso Oliveira da Silva, Assistant Professor, ISCTE-IUL

Supervisor
PhD, Maurício Breternitz, Principal Integrated Reseacher, ISCTE-IUL

October, 2019

Abstract

Face recognition applications are widely used today for a variety of tasks, whether personal or professional. When looking for a service that provides face detection and classification, it is easy to find several solutions. In this project another way is described so that it is possible to perform this task according to the desired needs without the need to use proprietary software. With the emergence of the Django Rest Frame Work, web application development has become easier. This work describes development of stable foundation and features that offer an administration panel, relational database management, and support for a Restful Application Programming Interface (API). This takes advantage of the exclusive use of Open Source technologies thus the application code can be modified and distributed free of charge. For the development of an API that could perform detection and facial recognition, applying an Open Source philosophy, in addition to Django Rest Framework technologies such as Python, C++, MySql and JSON were used. The prototype is initially capable of recognizing the number of faces per image, assessing eyes, smile, age and gender. Flexibility is designed to increase application capabilities with new algorithms implemented in various programing languages.

Key Words: Application, Algorithms, C++, Django, Framework, Image, Interface, JSON, Language, MySQL, Python, Programming, Recognition.

Resumo

Atualmente, as aplicações de reconhecimento de facial são amplamente utilizadas para uma variedade de tarefas, pessoais ou profissionais. Ao procurarmos um serviço que forneça deteção e classificação de rosto, é fácil encontrar várias soluções. Neste projeto, é descrita outra maneira para que seja possível executar esta tarefa de acordo com as necessidades desejadas, sem a necessidade de usar software proprietário. Com o surgimento do Django Rest Framework, o desenvolvimento de aplicações web ficou mais fácil. Este trabalho descreve o desenvolvimento de bases e recursos estáveis que oferecem um painel de administração, gestão de uma base de dados relacional e o suporte para uma API (Application Programming Interface) Restful. Ao tirar proveito do uso exclusivo de tecnologias Open Source, é permitido que o código possa ser modificado e distribuído gratuitamente. Para o desenvolvimento de uma API que pudesse realizar a deteção e o reconhecimento facial, aplicando uma filosofia Open Source, para além da tecnologia Django Rest Framework foram utilizadas tecnologias como Python, C ++, MySql e JSON. O protótipo é inicialmente capaz de reconhecer o número de rostos por imagem, e avaliar olhos, sorriso, idade e sexo. Mas para além disso, foi projetada flexibilidade para aumentar os recursos através da implementação de novos algoritmos em várias linguagens de programação.

Palavras-chave: Aplicação, Algoritmos, C ++, Django, Framework, Imagem, Interface, JSON, Linguagem, MySQL, Python, Programação, Reconhecimento.

Acknowlegement

To my thesis supervisors, family and friends, thank you for your support.

Index

Table Index

Figure Index

List of Abbreviations and Acronyms

BM - Boltzmann Machines

CPU - Central Processing Unit

CRUD – Create Remove Update Delete

DBMS – Database Management System

DBN - Deep Belief Network

DSP - Digital Signal Processor

DRY - Don’t Repeat Yourself

DBMS - Database management systems

FPGA - Programmable Field Arrays

FSF - Free Software Foundation

GNU - Gnu is not Unix

GPU - Graphic Processor Unit

HTML - Hypertext Markup Language

JSON - JavaScript Object Notation

LPB – Local Binary Pattern

MAX - Model Asset Exchange

MIT - Massachusets Institute of Technology

MVC - Model-View-Controller

MVT - Model-View-Template

OSI - Open Source Initiative

ORM - Object Relational Mapping

ROI - Region of Interest

RBM - Boltzmann Restricted Machine

REST - Representational State Transfer

SQL - Structured Query Language

URI - Uniform Resource Identifier

XML - Extensible Markup Language

1. Introduction

Facial recognition algorithms and open source software are two themes that merge in this research. This thesis makes an analysis of the potential that exists in face recognition and classification, and observes where open source software is used and how it can be beneficial for creating an API that can recognize and classify faces using Open Source technologies exclusively, taking advantage of it.

Open Source Application Programming Interfaces (APIs) allow communication between services through available features that allow creation of new applications using existing tools. This makes it possible to add different standards, routines and languages without the need for complex implementation processes. With different algorithms of different complexities, we can develop processes that allow classification of faces by various characteristics.

1.1. Motivation and relevance of the theme

Facial recognition and its classification is a task which humans perform in a natural way and with virtually no effort. However, for a machine, this process is not that easy. Currently the increase in computational power has generated a great interest in image processing, applying it to several areas. This thesis aims to describe the various phases for the creation of an API that, through algorithms in the field of computer vision and pattern recognition, can classify faces. Using only Open Source technologies, such as, Python, Django and MySQL, the process for its construction is described, as well as the different types of algorithms that can be used. This application includes a scalable architecture where new capabilities may be added. At the end some future works are suggested that can be used as a starting point for the continuation of this research or even new research related to this theme.

1.2. Open Source Software

Open source software is free software where its use and alteration is concerned. It is usually created by developer communities that ensure its development and maintenance without the usual restrictions of proprietary software. This software development paradigm has large implications, either technologically or at a culturally level.

Some people are still a little confused with this philosophy, because free is not associated with free software monetarily, but yes with freedom. According to FSF, the term free software refers to the freedom to execute, copy, distribute and improve the program. *"The term free is used in the sense of free speech, not of free of charge."* (Stallman, 2013) [45].

1.2.1. Open Source Definition

Open source software is software in which the source code is visible to that can be analyzed, changed and improved. Given this, many programmers inspect the code for bugs, which makes corrections faster and improve stability levels. The distribution should be the same terms as the original license. However, in some cases modified programs have different names and version numbers than original versions. The programmer has to take into account the various types of existing licenses ('Open Source Initiative', n.d.) [36]:

- Copyleft Licenses - A of copyright protection that places barriers to the use, dissemination and modification of a creative work due to the classical application of the intellectual property.
- Permissive Licenses - Few restrictions imposed on those who obtain the product. These licenses should be used when someone wants the project to reach more people for broader dissemination. A major case of success in this case is the Apache web server of the Apache Software Foundation.
- Proprietary Licenses - In this type of license, all copying, redistribution are strictly prohibited, and violations can lead to legal proceedings. In order to circumvent the previously mentioned restrictions, one should contact the developer for permission to do so, or to acquire a license for each one of the cases described above.

In order to better understand the concept, it must be mentioned that there are two main international organizations which are responsible for the protection and open source software, the Free Software Foundation (FSF) and the Open Source Initiative (OSI).

For the FSF, software can only be considered free when it considers the four types of freedoms for users. These are:

- Run the program for any purpose
- Study how the program works, and adapt it to customer needs
- Distribute copies of the program so that to can help the next user
- Change the program and distribute the changes so that the entire community benefits from them

As we can see, the issue of freedom is associated with a of a movement of ethical and political character, this means that any type of person or legal entity can use the software on as many computers as wanted, in any type of computer system, for any type of work or activity, without any restriction imposed by the supplier.

According to OSI, the availability of the source code is not a sufficient condition for it to be considered open source. It is necessary to meet ten criteria:

1.1. The license cannot restrict anyone, prohibiting the sale or donation of software to third parties;
1.2. The program must include source code and distribution of both the source code and the program already compiled;
1.3. The license must allow for changes and works that may be redistributed under the same terms as the original license;
1.4. The license can restrict source-code from being distributed in changed form only if the license grants the distribution of "patch files" with the source code for the intention of changing the program at construct time. The license shall explicitly allow distribution of software built from mutated source code. The license may demand derived works to take a different name or version number from the original software;

1.5. The license cannot discriminate against persons or groups;
1.6. The license cannot restrict users from using the program in an specific area;
1.7. The rights associated with the program through the license are automatically transferred to all persons to whom the program is redistributed without the need to define or accept a new license;
1.8. The rights associated with a program do not depend on which distribution a particular program is inserted;
1.9. The license may not place restrictions on other programs that are distributed along with the software in question;
1.10. No license requirement may be specific to a particular technology or interface style.

It is possible to observe that there is no great disagreement between the two institutions. The difference focuses on the discourse in relation to the target audience. User freedom features referred to by the OSI include some restrictions to corporate and business models developed around the software. However, the licenses approved by both foundations are almost the same and therefore, we can consider that the free software movement and the Open Source initiative (‘Open Source Initiative’, n.d.) [36] are concerned with similar software categories.

1.2.2. Open source history

With the emergence of the first commercially available computers, which it is from the 1950s, the first software was also created for them. Usually hardware sales presupposed the respective software because programs were strongly associated with the architecture of the machines on which they were executed. At that time, the focus of companies was the sale of the hardware, and there were not many restrictions placed on the use of software. It was possible to adapt it as they wished, in order to better use the available hardware. Then in the 1970s the situation began at some companies, such as Microsoft, which were not satisfied with the way their programs were redistributed without the company receiving royalties for copies. Thus, on February 3, 1976, Bill Gates wrote "*Open Letter to Hobbyists*" (Gates, 1976) [18], which was published in the Homebrew Computer Club newsletter. In this letter, Bill Gates states that the total royalties received by Altair BASIC were equivalent to just two dollars per hour spent on its development and documentation. He further

claims that the practice of sharing software is not fair and states that such practice prevents well-written software from being written. So began a changing posture in the industry, which now prohibits software from being copied or modified. Then came closed source software, characterized by restrictions which are made to the way it is used. In response to this new situation, initiatives to regain the freedom to improve and to share software were started. One of these initiatives was the GNU Project created by Richard Stallman.

In September 1983, being a programmer of the Laboratory of Artificial Intelligence MIT (Massachusets Institute of Technology), Richard Stallman posted a message on the net.unix wizards and net.usoft with the subject "new Unix implementation" (Stallman, 1983) [44]. In this message, he informs that groups are starting a system with UNIX called GNU (a recursive acronym for Gnu is not Unix) to be shared with all interested people. It also refers to some of the components that would be included, such as an operating system core, C compiler and text editor, and some articles to improve on existing UNIX systems at the time to leave also in its message of search or edition of GNU by its principles. "*To take it is motivated due to the development of software already not be one. People are in compliance with the new companies, but yes, software, such as neither access nor modification of the source code*." It explains that who like a program need to share with others people who like it. "*To be possible to continue to use computers without violations, to create a sufficient set of free software so that can be possible without using any software".*

In 1985 Richard Stallman published the "GNU Manifesto" (Stallman, n.d.) [46] to describe the project and to explain the importance of free software. In 1986 a definition of "Free Software" was published and created a Free Software Foundation (FSF), and in 1989 was published the first edition of the "GNU General Public License" (*General Public License—GPL*, 1989) [19] for an external site (General Public License - GPL).

1.3. Django Framework

The Django Rest Framework is an Open Source Framework that makes building APIs much easier, using serializers and views to allow developing applications very quickly. It is developed through a high level and easy to implement language. It is scalable, thus allowing the application growth, as well as the use of different databases using the Object Relation Mapping Layer.

Operating through the widely used HTTP transfer protocol, Django can be adopted by virtually any client which supports HTTP, with the flexibility to represent resources in different formats. Django applies the principle of DRY (Don't Repeat Yourself), in other words, take advantage of code already made, avoiding its repetition. It is an open source project published with the BSD license, which is considered public domain and may be modified without restriction.

1.4. Structure and organization of work

This dissertation describes the entire investigation process of development of a Django REST API for image classification. In addition to this chapter (Introduction), this report contains five more chapters:

In Chapter 2 (State of Art / Literature revision) the contextual framing is done through a brief characterization of the state of the art which the functionalities available in APIs, as well as the algorithms used.

In Chapter 3 (Problem Formulation) this project is discussed in more depth along with, the concepts and technologies that are the necessary knowledge base for its development.

In Chapter 4 (Proposed solution) some good practices for the development of a Web API are described, the standard of the implemented architecture, the stages of creation of an API and the specification of an operation mode.

In Chapter 5 (Implementation and Deployment) are presented technologies used in the development of API, the methodology applied, the various functionalities implemented, technologies used in API development and a description of how the API was implemented.

Finally, in chapter 6 (Conclusion) presents a conclusion about the project status in the final phase and it gives suggestions for future work.

2. State of Art / Literature Revision

Computer vision technology provides the ability of recognizing different objects through the identification and processing of images, similarly to human vision. Through artificial intelligence algorithms the images may be interpreted through an appropriate analysis. In this chapter we will look at the capabilities of the most well-known Face Recognitions APIs, architecture features and the different types of algorithms used.

2.1. Face Analisys in Images

Face detection and classification is one of the most popular challenges in computer vision today (*Computer Vision and Image Understanding*, 2015) [11]. There are several solutions available, including many made available in proprietary code. The features, which these applications usually offer are: face detection; extraction of facial features such as eyes, nose and mouth coordinates; face comparison; and finding the face of a particular person in a group of people.

For companies like Facebook or Google this is one of the most significant topics in the field of artificial intelligence systems. Companies in the sectors of Banking, health, security and travel are also investing heavily in this area. In Tables 1 and 2 we can observe some of the features provided by some APIs in this area.

The above tables illustrate common features of such APIs, green represent an included feature and red a non-included feature:

Table 1 – Historical Context (part 1)

Name	Face Detections	Land Marks	Recognition	Age	Gender	Emotions	Position	Mustache	Beard
BetaFace API		123							
Face ++		106							
Amazon AWS Recognitions		23							
Microsoft Cognitive Face API		27							
Google Cloud Platform		-							
IBM Watson Face		-							
Animetrics Face API		24							
Kairos API		49							
Meerkat FR		68							

Table 2 –Historical Context (part 2)

Name	Beard	Glasses	Sun Glasses	Blured	State of eyes	Ethnicity	Beauty	State of Mouth	Fixation of the look	Skin	Hat
BetaFace API											
Face ++											
Amazon AWS Recognitions											
Microsoft Cognitive Face API											
Google Cloud Platform											
IBM Watson Face											
Animetrics Face API											
Kairos API											
Meerkat FR											

We can notice that face detection is a feature common to all APIs, and they differ in features related to classification. Regarding the most frequent classifications, it can be verified that they are age, gender, position and emotions. The number of Landmarks also differ, with Betaface API having the most (123) and Animetric Face API having the least (24). Following is a description of each of the analyzed APIs.

Betaface API

Betaface API is a face detection and recognition web service (‘BetaFace API’, n.d.) [6]. It is possible scan image files or uploaded image URLs, identify faces and analyze them. It also provides verification (face comparison) and identification (face search) services, and maintains multiple user-defined recognition databases (namespaces).

Face ++

Detects and locates human faces in an image and returns high precision bounding boxes. It also allows the storage of metadata for each detected face for future use (‘Face ++’, n.d.) [16].

It checks the probability of two faces belonging to the same person, and finds faces that appear to be a new face from a particular collection of faces. Face ++'s quick and accurate search returns a collection of similar faces, along with confidence scores and thresholds for assessing similarity.

Amazon AWS Rekognition

Amazon Rekognition makes it easy to add image and video analytics to applications (‘Amazon AWS Rekognitions’, n.d.) [15]. Simply provide an image or video to the Rekognition API and the service will be able to identify objects, people, text, scenes and activities, as well as detect any inappropriate content. In addition, it offers highly accurate facial analysis and recognition for images and videos. It is available to detect, analyze, and compare faces for a wide variety of user verification, people counting, and public safety use cases.

Microsoft Cognitive Face API

It detects one or more human faces in an image and obtains facial rectangles at the points where the faces are in the image (‘Microsoft Cognitive Face API’, n.d.) [33], along with facial attributes that contain machine learning-based predictions of facial features. The features of facial attributes available are: Age, Emotion, Gender, Position, Smile, and Facial Hair, along with 27 references to each face in the image.

Google Cloud Platform

The Google Cloud Vision API offers advanced machine learning models pre-trained through REST APIs (‘Google Cloud Platform’, n.d.) [20]. It can label images and quickly sort them into millions of predefined categories. It detects objects and faces, reads printed and handwritten texts and creates valuable metadata in the image catalog.

IBM Watson Face

The IBM Watson Visual Recognition service uses deep learning algorithms to identify scenes and objects in images (‘IBM Watson Face’, n.d.) [25]. It allows one to create and train a custom classifier to identify specific subjects.

Animetrics Face Recognition

The Animetrics Face Recognition API can be used to find human faces, detect feature points, correct off-angle photographs, and ultimately perform face recognition (‘Animetrics Face API’, n.d.) [3]. Information on facial features including ears, nose, eyebrows, lips, chin is returned as coordinates in the image. The Animetrics Face Recognition API will also detect and return the gender and orientation of faces along 3 axes.

Kairos

Kairos is a face recognition platform that enables one to quickly and easily integrate human identity resources into products and services (‘Kairos API’, n.d.) [27]. Deep learning algorithms analyze the faces found, then the API returns useful data about the faces found. It can be used to search, match and compare faces or measure features like age and gender.

Meerkat FR

No system installation required, it uses simple HTTP calls and, the on-premise version has a web interface that allows to setup in less than 5 minutes with an assertiveness of over 98%, providing fast responses and a high availability server (‘Meerkat FR’, n.d.) [32].

2.2. REST Architecture

Representational State Transfer (REST) is a World Wide Web style of software architecture designed to build Web applications. REST offers a set of guidelines to develop a cohesive, scalable, high performance service. This style of architecture is based on the following basic principles:

Client - Server

A REST application must separate architecture and responsibilities into two environments (client and server), becoming standalone and consequently more scalable. The service consumer is concerned only with the interface while the service provider (server) is responsible for returning a response to the customer by executing a request sent by the customer.

Stateless

The server does not store any client status information. This information is stored on the client. A customer can place multiple requests to the server. Each request to the server is made independently and standardized, passing only the necessary information to the server so that it can process it properly and correctly.

Cacheable

In order to avoid unnecessary processing and significantly increase performance, when a client makes a request to the server, the response is temporarily copied and stored. Thus, if multiple clients make the same request to the server, the server returns what is cached without having to process it again, improving efficiency, scalability and user performance. However, cached information when overused can decrease the reliability of the data making it obsolete, so it is important to implement a gateway (or reverse proxy) cache, that is, a network server (independent layer) that caches responses as they are returned, reusing them for future requests, decreasing thus the number of direct server interactions.

Uniformized Interface

Communication between clients and server follows simple rules that follow certain guideline, which when well defined, make communication more uniform and

normalized. The following is a set of guidelines based on the defined architectural pattern:

- Each resource is identified through a Uniform Resource Identifier (URI) specific and cohesive;
- A URI represents resources, not actions;
- Actions are represented by HTTP verbs (‘HTTP Methods’, n.d.) [23] (or methods);
- The format in which the resource can be returned to the customer is chosen according to the specific needs of the service provider, the most commonly used formats being JavaScript Object Notation (JSON) and eXtensible Markup Language (XML);
- The format of client / server communication must be defined in Content Type;
- The client receives all the necessary information in the response so that he can navigate and have access to all application resources;
- In requests and responses, metadata information must be passed (for example, HTTP code, Content-Type, Host, among others).

Layered System

The REST application should consist of layers that can be easily changed, added and / or removed. Each layer communicates or processes information between client and server either specifically or individually. Thus, as a rule, the client should not invoke the application server directly without first going through middleware, for example a load balancer or a machine that interfaces with the server. Middleware is responsible for distributing requests to the server, ensuring that each layer performs specific functions leading to a much more flexible structure for changing, providing better performance, scalability, simplicity, flexibility, visibility, portability, reliability and security.

2.3. Application Programming Interfaces

Application Programming Interfaces (APIs) (‘Red Hat: What is an API?’, n.d.) [43] allow one to communicate with other services, to use their existing functionality without knowing how they were implemented, or to develop and provide functionality for use by others. With this we receive flexibility to develop new applications using existing tools, saving time and simplifying development. This connection between different applications/services, enables aggregation of different patterns, routines and languages without the need for complex implementation processes.

For the creation of face recognition APIs, it is possible to use various algorithms from the traditional face detection algorithms to those using artificial neural networks such as Machine Learning or Deep Learning (Chi-Fa, Yu-Shan, & Chia-Yen, 2013) [10]. These kinds of APIs are widely used in everyday life in various areas such as security implementation, face modeling and multimedia data management techniques, computer entertainment and improvements in digital communication tools.

2.4. Algorithms for Facial Recognition

Facial recognition is the procedure for automatically locating people's faces in images or videos. (‘FaceRecognition Using Evolutionary Pursuit’, n.d.) [17] Although this technology has been around for decades and its use has become more noticeable and affordable in recently years. It is now used in innovative solutions such as applications of personal photo recognition and classification, investigation tool, means of surveillance and authentication for mobile devices. When a face is detected it can be analyzed for several characteristics, such as position, size, orientation, eyes, nose and mouth, through reference points that follow the shape of a facial feature. By this means it is feasible to classify a particular facial feature if it is present at detection. As this technology is implemented by the use of Artificial Intelligence algorithms, it makes it possible to learn with the evolution of the records becoming more accurate and with greater detection capacity.

2.4.1. Traditional Algorithms

Traditional Algorithms are based on handcrafted resources such as edges and texture descriptors mixed with Machine Learning techniques such as principal component analysis, linear discriminant analysis or support vector analysis. Researchers focus on specialized methods for each type of variation such as age (U. Park, & A. K. Jain, 2010) [51], pose (C. Ding & D. Tao, 2016) [8] or lighting (Liu, Lam, & Shen, 2005) [29].

These traditional methods can be categorized into two different groups: holistic resources and local resource approaches. The holistic group can also be separated into the linear and nonlinear projection methods. Many applications demonstrate good results from the methods used in linear projection, such as principal component analysis (PCA) (Turk, Pentland, & J. Cogn, 1991) [50], independent component analysis (ICA) (Bartlett, Movellan, & Sejnowski, 2002) [4], linear discrimination analysis (LDA) (Belhumeur, Hespanha, & Kriegman, 1997) [5] and linear regression classifier (LRC) (Naseem, Togneri, & Bennamoun, 2010) [34]. However, due to changes in lighting, facial expression, and other aspects, these methods cannot classify faces as correctly as possible. The main cause is complex variability with non-convex, nonlinearly exposed face patterns. To use these cases, applications may be non linear such as kernel PCA, kernel LDA (KLDA) (Lu, Plataniotis, & Venetsanopoulos, 2003) [30] or linear local embedding (LLE) (He, Yan, Hu, Niyogi, & Zhang, 2005) [22]. These nonlinear methods use kernel techniques to map face images into a larger space without any simplified linear face variety, what makes traditional linear methods usable. But while there is a strong theoretical basis for kernel methods, their application in practice does not yield a significant improvement in comparison to linear methods. That said, the methods of nonlinear projection, inherited the simplicity of linear methods and the ability to handle complex data from nonlinear methods. Among which it is worth highlighting LLE and LLP (Xiaofei & Partha, 2003) [52].

The second group, with local resources, has certain advantages over holistic resources. They are better prepared methods for altering the image, such as expression, desalination and illumination. The common method is identifying Local Binary Patterns (LBPs) (Ahonen, Hadid, & Pietikäinen, 2006) [1]. Neighbouring changes around the centre pixel are made a simple but effective way described by LBP. It is a constant transformation of intensity that supports minor lighting changes. LBP variants have been suggested for the creation of an LBP original, such as the Gabor Phase Pattern Histogram (Zhang, Shan, Chen, & Gao, 2007) [55] and the local Gabor Binary Pattern

Histogram Sequence (Yang & Chen, 2013) [53], with a view to the model or close relationship in the spatial, frequency and orientation domains (Zhang, Gao, Zhao, & Liu, 2010) [54].

These traditional methods that provided cutting edge efficacy for a few years have been lately replaced by Deep Learning methods based on CNNs and RNNs due to the significant improvement in achieved accuracy.

2.4.2. OpenCV

OpenCV ('OpenCV', n.d.) [38] is one of the most popular open source Machine Vision and Machine Learning software libraries. It is licensed by BSD, making it easier for companies to use and modify their code. Its purpose is to provide a common infrastructure for Computer Vision applications and to accelerate their process. It contains over 2500 optimized algorithms, including an extensive set of algorithms in Computer Vision and Machine Learning. In these algorithms there is the ability to identify faces, facial expressions, to find similar faces and to remove red eyes, through small patterns and features that must be matched. The algorithms divide the task of identifying the face into a large number of smaller and simpler tasks to facilitate this resolution.

2.4.3. Artificial Neural Networks

An ANN can be defined as a network of several parallel and distributed information processing systems composed of artificial neurons, with a simple processing, and of highly degree of interconnection. This processing structure, which can be implemented in electronic devices, is made up of a number of interconnected units, called artificial neurons. Each unit exhibits specific input/output behavior, determined by its transfer function, by interconnections with other units within a vicinity radius, and possibly external inputs. These simple and highly interconnected functional processing units may generate complex behaviors.

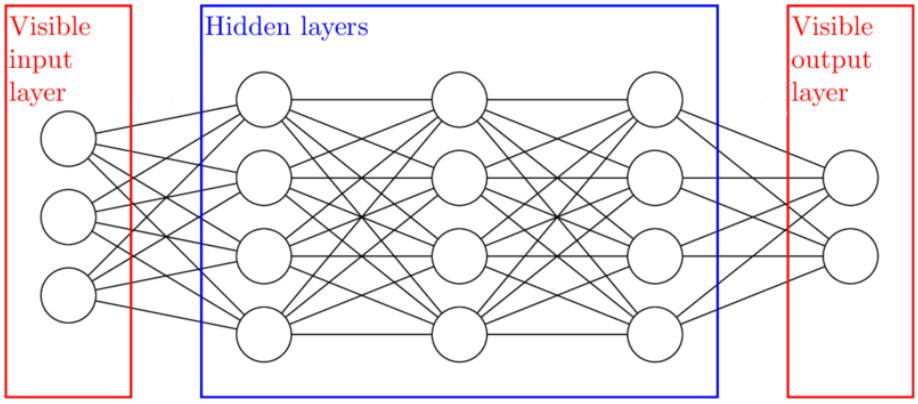

Figure 1 – Artificial Neurons

Neurons are organized in layers, where each layer handles information received in a different way. The signal enters the input layer and passes through the hidden layers until it reaches the output layer. Each connection between neurons can transmit signals to another neuron. The receiving neuron processes the input signal, applies a mathematical function and it sends the result to the next neuron to which it is connected. Each connection, in turn, has an associated weight, which corresponds to the information stored by the neuron, which can increase or decrease the signal strength that is transmitted. By executing this process multiple times with various examples, learning is provided to the network through a gain of experience that will allow it to recognize patterns.

2.4.4. Deep Learning

Dealing is an important part of Artificial Intelligence, and a subcategory of Machine Learning, Deep Learning has a specific approach of building and shaping neural networks that can support and also work with big data, as this type of architectures tends to operate better with more input data, and operate autonomously by overlapping nonlinear data processing layers. Based on the way the human brain processes information and learns, it is capable to learn unsupervised from unstructured, unlabeled data to take on a variety of computational tasks such as speech recognition, vision, and natural language processing. A simple Neural Network has up to 5 layers, a Deep Network has more than 5 layers. (‘Deep Learning: The Confluence of Big Data, Big Models, Big Compute’, n.d.) [14]

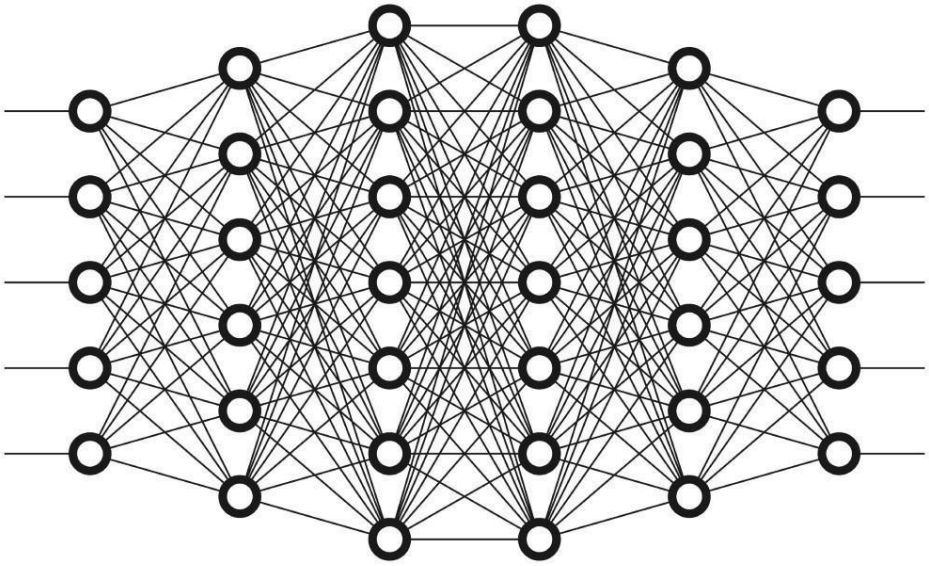

Figure 2 – Deep Learning Architecture

This requires a higher processing capacity to train deep networks, depending on the network architecture used, leading to the need for advancement of parallel and GPU processing techniques. The leading companies focused on developing hardware, which is dedicated for deep learning and also graphics processing, are *Google TPU* (‘Google TPU, n.d.) [21], *Intel Processor Graphics* (‘Intel Porcessing Graphics’, n.d.) [26] and *Nvidia AI Computing* (‘Nvidia AI Computing’, n.d.) [35].

It is possible to train images in a deep learning network in two different ways:

- **Supervised Training**

Supervised Training is the most usual form of Machine Learning. With Supervised Learning, a set of samples are sent as input to the system during the training phase. Each input is labeled with a desired output value, so the system knows what the output should be when the input is received. The training is accomplished by minimizing a particular cost function which represents the input connection and the desired output. Figure 3 illustrates the Supervised Training process.

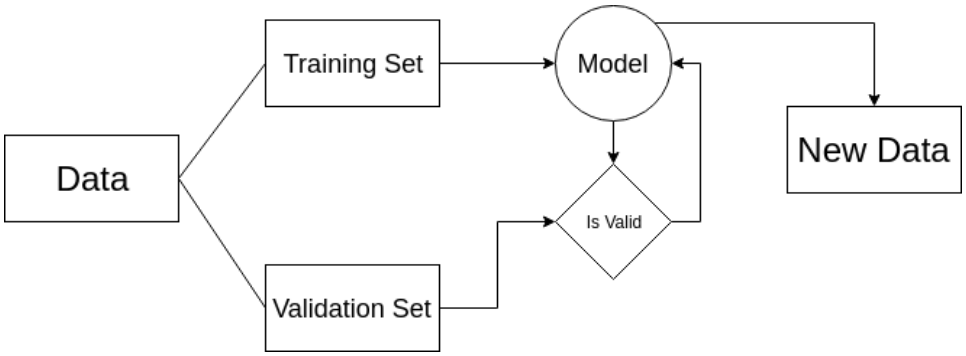

Figure 3 – Supervised Training Model

- **Unsupervised Training**

Unsupervised Training works with an unlabeled sample set. Thus, the system develops and organizes data by seeking common characteristics between them based on inferred knowledge. Figure 4 illustrates the Unsupervised Training process.

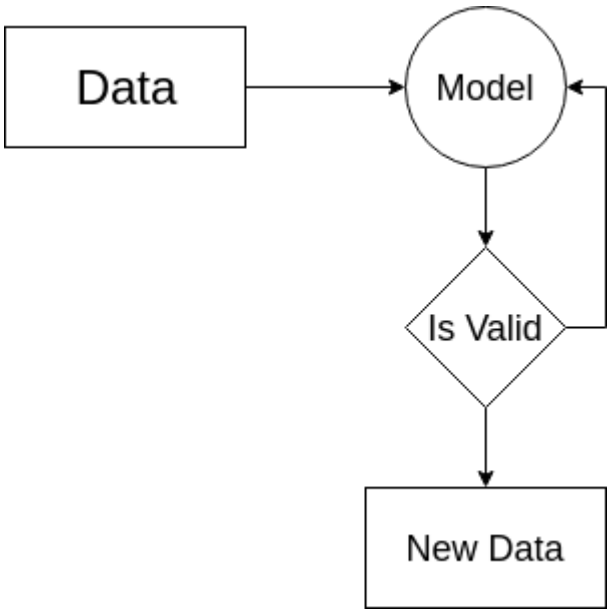

Figure 4 – Unsupervised Training Model

3. Problem Formulation

In preparing this project a solution was developed that differs from the ones observed in the previous chapter. Based on a free software philosophy, using exclusively opensource technologies, we propose an API architecture that allows the user to observe, create and modify algorithms for face recognition and classification. This will allow not only the use of the original features, but also the freedom to improve and implement algorithms in different languages.

3.1. Architecture

The proposed architectural approach to create this API is fundamentally supported by five elements, which are: '*Storage'*, '*Algorithms'*, '*Data Stream'* and '*Users' and 'Data Management'*. The available features allow private, public or restricted access to API access, which can be changed or expanded later. Figure 5 illustrates the proposed architecture.

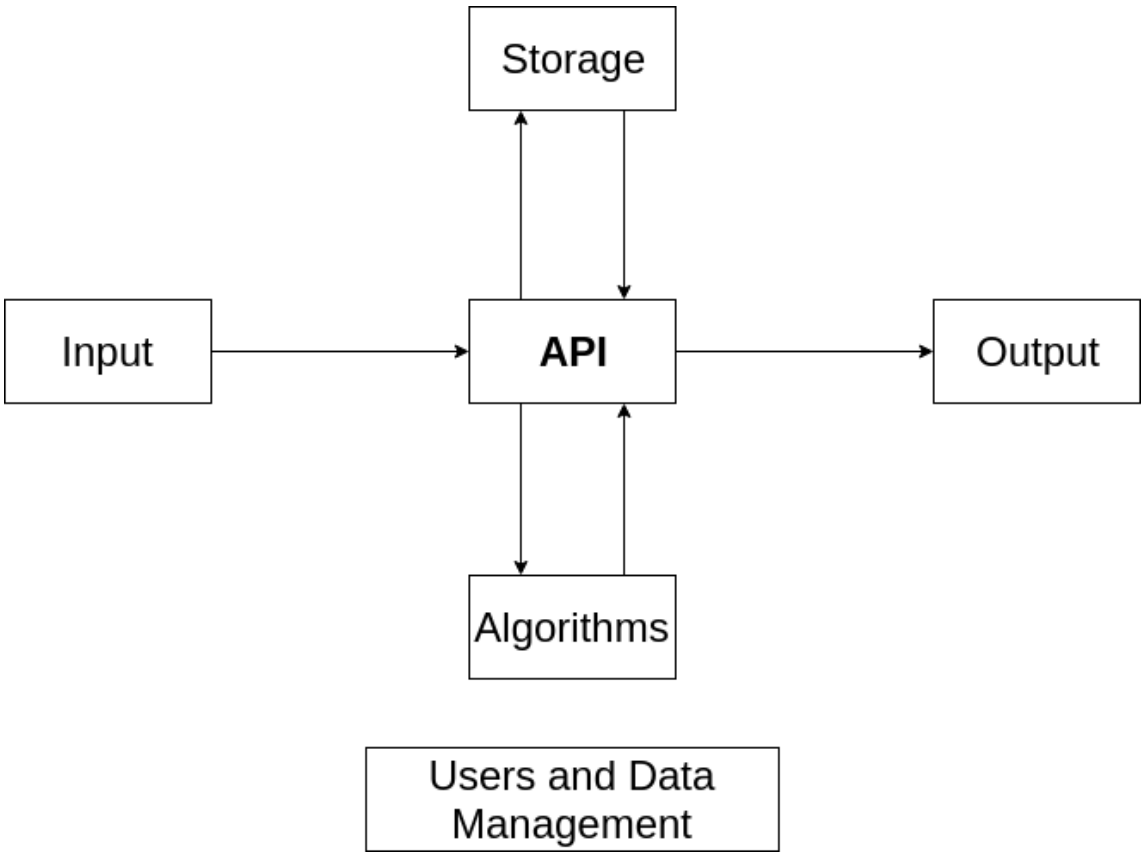

Figure 5 – API Architecture

3.1.1. Input

The input is provided through a URL and Path of an image (or a directory which contains multiple images, if it is for the purpose of classifying multiple images at once). The image name is associated with the identifier with which it will be associated in the database.

3.1.2. Storage

The stored databases are required for the proper functioning of an application. For information to be easily accessible, used and understood, Database Management Systems (DBMS) are used. DBMS can provide information stored in the database as well as statistics of changes and information in a database.

DBMS perform the same basic task, which is to allow users to create, edit, and access information in databases, and to update it. For this reason, it is critical to choose the database that works with large-scale information. Often the first options are relational databases, which are best known among programmers, but they are not always the most appropriate solutions. When comparing different popular databases, we should consider how much the DBMS is friendly and scalable, as well as being able to be integrated into other products as well. The differences between relational and nonrelational databases have to do with how information is inserted and organized. A relational database provides greater consistency and reliability, but it requires the relationship between multiple tables for access. With the help of the SQL (Structured Query Language), information is stored in rows and columns. However, in order to insert data into the tables, the programmer must design each structure, offering greater consistency for each operation. DBMS have functions such as validation, verification and integrity assurance, concurrency control and fault recovery, as well as transaction management and query optimization. Non-relational DBMS has a greater advantage, with information grouped and stored in the same registry. It has better performance and high scalability, ensures more efficient management with the NoSQL language. It does not require design of its implementation, because all information is grouped into a single record. Focusing on cross-variables, dimensional modeling allows for making miscellaneous analytic combinations of information stored in a non-relational database. Each type of NoSQL database has its own schema for query. The key-value category, which is

simpler, can be accomplished by means of a key. To perform a search, one may simply access the hash of the key to retrieve the information that is being sought.

Table 3 – Storage Analysis

Non-relacional	**Relacional**
Stores data in JSON documents, key/value pairs, wide column stores, or graphs	Stores data in a table
Offers flexibility as not every record needs to store the same properties	Great for solutions where every record has the same properties
New properties can be added on the fly	Adding a new property may require altering schemas or backfilling data
Relationships are often captured by denormalizing data and presenting all data for an object in a single record	Relationships are often captured in normalized model using joins to resolve references across tables
Good for semi-structured, complex or nested data	Good for structured data
Dynamic or flexible schemas	Strict schema
Database is schema-agnostic an the schema is dictated by the application. This allows for agility and highly iterative development	Schema must be maintained and kept in sync between application and database
ACID transaction support varies per solution	Supports ACID transactions
Eventual to strong consistency supported, depending solution	Strong consistency enforced
Consistency, availability and performance can be traded to meet the needs of the application	Consistency is prioritized over availability and performance
Performance can be maximized by using scaling up available resources and using in-memory structures.	Performance can be maximized optimizing queries
Information are stored on a flexible format	Information about an entity may be spread across many tables or rows, requiring many joins to complete an update or a query
Supports horizontally scaling	Scaling is typically achieved vertically with more server resources

In the context of the developed application we opted for the use of a relational database such as MySQL, because it allows a more systematic and clear view of the data and it is an available, widely used DBMS. It is free and also can be used by anyone, under the GNU General Public License and the source code is available under the domain.

3.1.3. Algorithms

For the operation of this API we can use various types of *API – Software* comunications mechanims, in this section we will characterize and understand the differences between them.

- **Embedded**

To allow an API to perform an image classification, we use algorithms that makes this task possible. To do this, we may simply use of algorithms embedded in the API code itself or apply external algorithms, where we can adapt, remove or update the rules as needed. The use of algorithms is already an indicator of a quick task that provides an API, such as the Facial Age Estimator case available in the *IBM Model Asset Exchange (MAX)* (‘IBM Model Asset Exchange (Max)’, n.d.) [24].

- **OpenCV library**

With regard to the development of algorithms one can use an OpenCV library (‘OpenCV’, n.d.) [38] that already contains many pre-trained classifiers of eyes, smiles, facial expressions and so on. To make it work, first we must use the XML classifiers and then load an input image and convert it into shades of gray, as shown in the following piece of code:

```
import numpy as np
import cv2 as cv
face_cascade = cv.CascadeClassifier ('haarcascade_frontalface_default.xml')
eye_cascade = cv.CascadeClassifier ('haarcascade_eye.xml')
img = cv.imread ('image.jpg')
```

```
gray = cv.cvtColor (img, cv.COLOR_BGR2GRAY)
```

The main faces in the image are provided, as markings with a rectangle likes Rect (x, y, w, h). Now that we find the faces in the picture, we can look for aspects in the face. If we want to find the position of the eyes, we will have to create a Region of Interest (ROI) within the face and apply a detection of eyes in this ROI. The following piece of code illustrates this actions:

```
faces = face_cascade.detectMultiScale (gray, 1.3, 5)
for (x, y, w, h) in faces:
    cv.rectangle (img, (x, y), (x + w, y + h), (255,0,0), 2)
    roi_gray = gray [y: y + h, x: x + w]
    roi_color = img [y: y + h, x: x + w]
    eyes = eye_cascade.detectMultiScale (roi_gray)
    for (ex, ey, ew, eh) in eyes:
        cv.rectangle (roi_color, (ex, ey), (ex + ew, ey + eh), (0,255,0), 2)
cv.imshow ('img', img)
cv.waitKey (0)
cv.destroyAllWindows ()
```

- **Deep Learning**

It is also possible to use Deep Learning techniques to classify an image. Models of Deep Learning, involve knowledge of Mathematics, Statistics, Programming, Computer Vision, Image Preprocessing, among other areas, for the recognition of Images with Neural Convolutional Networks. There are many frameworks that can be used like TensorFlow, Keras, PyTorch, Caffe and Deeplearning4J.

These tools help to quickly implement an appropriate framework to optimize performance, parallelize processes, facilitate code comprehension, and have good community support. In the tables below it's possible to see some specifications about this Open Source Frameworks:

Table 4 – Deep Learning Framework Features (part 1)

Framework	**Creator**	**Initial Release**	**Software license**	**Written in**	**Interface**
TensorFlow	Google Brain	2015	Apache 2.0	C++, Python	C, C++, Go, Java, JavaScript, Julia, Python, R, Swift
Keras	François Chollet	2015	MIT License	Python	Python, R
PyTorch	Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan (Facebook)	2016	BSD	C, C++, Python	Python, C++
Caffe	Berkeley Vision and Learning Center	2013	BSD	C++	Python, MATLAB, C++
Deeplearning 4J	Skymind engineering team; Deeplearning 4j community; originally Adam Gibson	2014	Apache 2.0	C++, Java	Clojure, Java, Kotlin, Scala, Python

As can be seen in the table above all Frameworks were mostly built with the C ++ and Python languages. All use an Open Source license, giving them the flexibility to modify the software and use it privately. For the implementation of algorithms several languages can be used with Python being the most usual.

Table 5 – Deep Learning Framework Features (part 2)

Framework	OpenMP	OpenCL	CUDA	RBM	DBN
TensorFlow	No	Yes	Yes	Yes	Yes
Keras	Only if using Theano as backend	Yes	Yes	No	No
PyTorch	Yes	Yes	Yes	No	No
Caffe	Yes	No	Yes	No	No
Deeplearning 4J	Yes	No	Yes	Yes	Yes

Table 5 shows some technologies which can improve algorithms to get better results in terms of evaluation precision and velocity. It is possible to verify that all of them use CUDA to accelerate frameworks performance, but others techniques are also used. Next is we provide more details about each one:

OpenMP

OpenMP ('OpenMP', n.d.) [39] is an API for multi-platform shared memory multi-processing. It allows one to add concurrency to the programs written in various languages based on the fork-join execution model. Its portability and scalability gives to programmers a simple and flexible interface for the development of applications built with a parallel programming model.

OpenCL

OpenCL('OpenCL', n.d.) [37] is the first open, royalty-free unified programming standard to accelerate algorithms in heterogeneous systems. OpenCL allows the use of a C-based programming language to develop code on different platforms, such as central processing units (CPUs), Graphic Processor Unit (GPUs), Digital Signal Processors (DSPs), and Programmable Field Arrays (FPGAs).

CUDA

CUDA ('CUDA', n.d.) [12] is a platform that uses the GPU for parallel computing using a

programming model that makes its use it as simple and elegant. The use of multiple computing cores in a graphics processor to perform general-purpose mathematical calculations, achieving considerable acceleration in computing performance. It is proprietary to Nvidia.

Boltzmann Machines

The *Boltzmann Machines* (BM) (‘Bolzmann Machine’, n.d.) [7] were one of the first neural networks capable of learning internal representations, which are capable of representing and solving difficult combinatorial problems. They played a major role in the resurgence of neural networks, allowing efficient deep learning training.

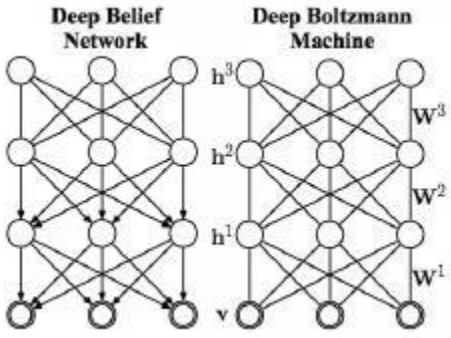

Figure 6 – Boltzmann Machines

Boltzmann Restricted Machine

A *Boltzmann Restricted Machine* (RBM) (‘RBM’, n.d.) [42] is a BM in which each visible node is connected to each hidden node. There are usually no other connections. This transforms the RBM into a bipartite graph, which means that the values of the nodes can be inferred in blocks, first inferring all visible nodes and then inferring all hidden nodes. Because of this, RBM is considered a neural network of two layers, with a visible and a hidden layer.

Deep Belief Network

A *Deep Belief Network* (DBN) (‘DBN’, n.d.) [13] is formed by training RBMs one at a time and then stacking them one on top of the other to infer successive hidden layers. After the RBMs are stacked, it changes to the previous distribution over the hidden values of the lower RBM in the

stack (the former is now determined by the upper layer rather than by the hidden deviations of the lower layer). While the RBMs have very simple inference rules, the inference in a DBM is approximate and depends on several passages through the hidden layers. To infer the state of a hidden layer, it is necessary to know about the state of the layer above and below it, so it is necessary to do some iterative sampling to find the hidden states.

TensorFlow

TensorFlow ('Tensorflow', n.d.) [48] was developed by engineers of the Google Brain team, being the software library most used in the field of Deep Learning. It works well on a sequence-based data images and concepts such as calculus.

Keras

Keras ('Keras', n.d.) [28] is a very solid framework and the one to start the study in Deep Learning, for those who are familiar with Python. With the focus on getting results, it makes it possible to get a working model quickly. Keras is also integrated with TensorFlow. By creating a *tf.keras* template it makes TensorFlow easier to use without sacrificing flexibility and performance.

PyTorch

PyTorch ('Pytorch', n.d.) [41] is a more intuitive Framework than TensorFlow, without the need for much mathematical knowledge, Machine Learning experience and easier to understand models. Although it does not contain a preview tool such as *TensorBoard*, it allows the use of graphic tools such as matplotlib ('Tensorflow', n.d.) [48].

Caffe

Caffe ('Caffe', n.d.) [9] is mainly used to build Deep Learning models for mobile phones and other computationally restricted platforms. It is quite functional for modeling in image data, but stays a bit behind the other frameworks in recurrent neural networks and language models.

Deeplearning4J

Deeplearning4J (‘Deeplearning4J’, n.d.) [15] is a framework for Java programmers. It provides massive support for some different neural networks like CNNs, RNNs and LSTMs. It can process a huge amount of data without sacrificing speed. Through a composition approach, neural networks such as restricted Boltzmann machines, convolutional networks, self-encoders and recurrent networks can be interconnected to create deep networks of varied types.

3.1.4. Output

Once the algorithms have performed the image classification and information tasks have been stored in the database, it is necessary to transmit this information to anyone consuming the generator API information. This requires that these data be serialized in a given format.

This serialization is used to convert an object to a canonical data storage format. This format can be text, such as JSON and XML, or binary as Protobuf (‘Protobuf’, n.d.) [40]. This format change facilitates data storage (either on the computer or in a memory buffer) or transmission over a network connection. Upon receipt, the object can be recreated in the same state as the original.

Table 6 – Serialization Analyse

JSON	XML	Protobuf
Human readable/editable	Human readable/editable	Very dense data (small output)
Can be parsed without knowing schema in advance	Can be parsed without knowing schema in advance	Hard to robustly decode without knowing the schema (data format is internally ambiguous, and needs schema to clarify)
Excellent browser support	Good tooling support (xsd, xslt, sax, dom, etc)	Very fast processing
Less verbose than XML	Pretty verbose	Not intended for human eyes (dense binary)

In this project the option for integration with the RESTful API fell on JSON with the comparison that we can verify in the *Table 2* considering the following aspects:

- **Less verbose**: JSON is more compact and generally it is a more readable style. JSON's lightweight approach can make significant improvements in performance.
- **Fast**: JSON uses less data overall, reducing the cost and increasing the speed of analysis.
- **Readable**: Easier mapping for domain objects.
- **Structure Matches data**: Key / value pairs may at some point limit what is possible to do, but it is a predictable, easy-to-understand data model.
- **Alignment of objects in code**: a JSON structure is intuitive, making it easier to read and map directly to domain objects.
- **Alignment of objects in code**: JSON objects and code objects match which is beneficial when quickly creating domain objects in dynamic languages.

4. Proposed solution (architecture)

As seen in the previous chapter the architecture for developing this API is made up of five elements. In this chapter we will cover the best practices for their construction, the design patterns used, the design phases and specify their mode of operation for different types of users.

4.1. Good Practices Web API Opensource

In the development of this project we must take into account several aspects that allow us to apply good development practices. A well-structured Web API Opensource must comply with certain rules:

Should support all platforms. Any client should be able to call the API, regardless of how the API is implemented internally. This requires the use of standard protocols, as well as a mechanism from which the client and the Web service can agree on the format of the data to be exchanged.

The Web API must be able to evolve and add functionality independently from client applications, in order to support evolution of service. As the API evolves, existing client applications must continue to function without modification. All features should be Detectable, so that client applications can fully utilize it.

The version numbering system used should be, for software systems, based on natural numbers separated by a dot. When a new version is released, it should be the largest number of all preceding versions.

The HTTP protocol should be used on a correct way in their methods (GET, POST, PUT, DELETE, OPTIONS, PATCH, HEAD).

The order headers should be properly defined as well as the correct coding of the responses, taking advantage of the standard meanings of different response codes.

Resource names, URIs, must be correctly defined. All operations on the same resource must use the same name and must be identified by the URL.

Data must be in JSON and XML formats. By default, the first one must be used. If the costs of implementing two serializers are not increased, the possibility of representation in the XML format should also be offered, increasing interoperability with other systems.

Provide all CRUD (Create Remove Update Delete) operations, starting from the same for the construction of more elaborate resources, adding other ones as a way of reducing the amount of communications.

In order to maintain compatibility with existing clients, changes to the service should lead to the creation of new versions, allowing customers to continue to use the services until they upgrade to the new version.

Develop and maintain of service documentation, preferably with examples of use in various programming languages, so that customer service learning is as simple as possible.

Define the type of license that should be applied to the project.

Create of a README.txt file with the project's actual information.

4.2. Architecture

In planning for the development of the project it was opted to use the Django REST Framework to carry it out it, because it is an open, mature, well-supported library, which points to the creation of sophisticated web APIs. Through flexible and complete tools, with a modular and customizable architecture, it allows both the development of simple and ready-to-use APIs as well as complicated REST constructs. Django's REST structure contains a large set of ready-made features, but the main view class is very simple and the overall structure is easy to use.

4.2.1. Architectural Patterns

The architectural pattern used by Django REST Framework technology are the Model-View-Template (MVT) and the Object Relational Mapping (ORM). Through these patterns it was possible to structure the REST API architecture developed.

4.2.2. MVT Pattern

To begin implementing the REST API, it is necessary to understand the architecture of the Django framework. According to the *Django Book* (*The Model-View-Controller Design Pattern*, n.d.) [49], it is considered a Model-View-Controller (MVC) framework, however it follows its own architecture pattern, Model-View-Template (MVT), in which considers the Controller as View and View as " Template ".

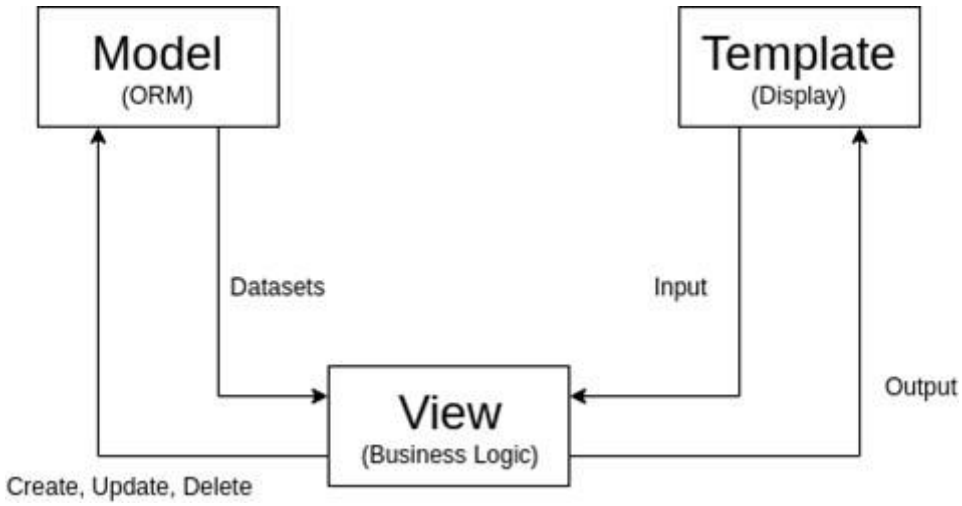

Figure 7 – Model-View-Template Architecture

Model

The Model layer (database access layer) consists of several classes associated with the tables in the database. This layer does not contain the data but rather an interface for the data, which includes all information that is related to the database including its structure, access, validation and behavior of the data and relations between them. The Django framework makes it easy to interface with the database allowing the user not to worry about the connection between the domain classes and the database. For this, it uses the technique called ORM.

View

The View layer defines what data is presented to the client being responsible for the logical processing of the data. It is the View layer that communicates with the Model and Template. Each "View" is a return function for a specific URL. This function returns to the client the information in the defined format (XML, HTML, JSON, among others) or the errors found. Comparing with the MVC model, this layer corresponds to the Controller. In this architectural pattern, the controller is responsible for controlling the information exchange between the model and the view, decide what information to obtain in the database through the model and which information will be sent to the visualization.

Template

The Template layer describes the visual form in which the data is presented to the user. This layer is composed of HTML, CSS, JavaScript, among others. In the MVC model, this layer corresponds to the View layer.

4.2.3. Default ORM

ORM is a technique that allows one to manipulate and to query data from a database through an object-oriented perspective adapted to the programming language being used instead of using SQL statements. It serves as a bridge between the relational model (database) and the object-oriented world (classes and methods) by mapping the tables that constitute the relational database for the data structures. It is in the Model entity that this mapping is done. It is important to note that the mapping is done only for the database, not for the records. The records of each table are represented by instances of corresponding classes. Thanks to ORM, there is an independence of the models from the database, a direct access to the different components in the database (related objects), an easy and flexible implementation of the CRUD operations and a validation of the fields. Through ORM it is possible to manipulate and to understand the data, in an easy and intuitive way. However, the ORM is a procedure that can potentially reduce performance.

4.3. API Creation Phases

The development structure of this API proceeded in eight phases: Know Customer Needs, Define Project Requirements, Choosing Technologies, Assess Project Viability, Document All Procedures, Choosing the Development Methodology, Test Created Features and Choose License.

1.1. Before starting the application framework, it was necessary to know the needs for project implementation. An initial meeting, general discussions were held about what was needed, what needed to be done and what it would be like.

1.2. A list of priorities was made after the requirements analysis, which defined what the software needed to have, what might limit each function and how long it could take in theory.

1.3. In the first stage of development, an application for classifying images was created. The programming language used was PHP's interface to the MySQL database due to its open source feature, working well with each other and availability to run on Linux and Windows servers. Then a Rest API was made with the Django Framework that can use various algorithms for face recognition.

1.4. Then we analyzed the types of algorithms that could be used and the way of storing the images and their classifications.

1.5. This was followed by the creation of a Django Rest Framework API with the administrative functionality provided by it, the classification of images through some implemented algorithms and its storage in a MySQL database.

1.6. Throughout the project, all procedures were documented. This action is necessary due to incremental changes that may exist. A list of what should be developed and what should be done was made. Also, how the process will have to happen.

1.7. After the work was completed some tests were developed and performed.

1.8. Finally, the license was chosen to make the software available for its intended purpose.

4.4. Specification Of Operation Mode

To specify how the API operates, we have to distinguish between two types of users with different functionality: the regular user and the Administrator.

4.4.1. Regular User

A regular user can upload photos and rate photos by different criteria, as well as observe how photos are being evaluated. It allows to sort images in isolation by entering the URL of the image or multiple images contained in a directory, indicating the path to that directory. The past address will consist of the function we want to apply and the image or images we want to evaluate. The output of this classification is transmitted in JSON format. In *Figure 8* we can see the representation of this whole procedure.

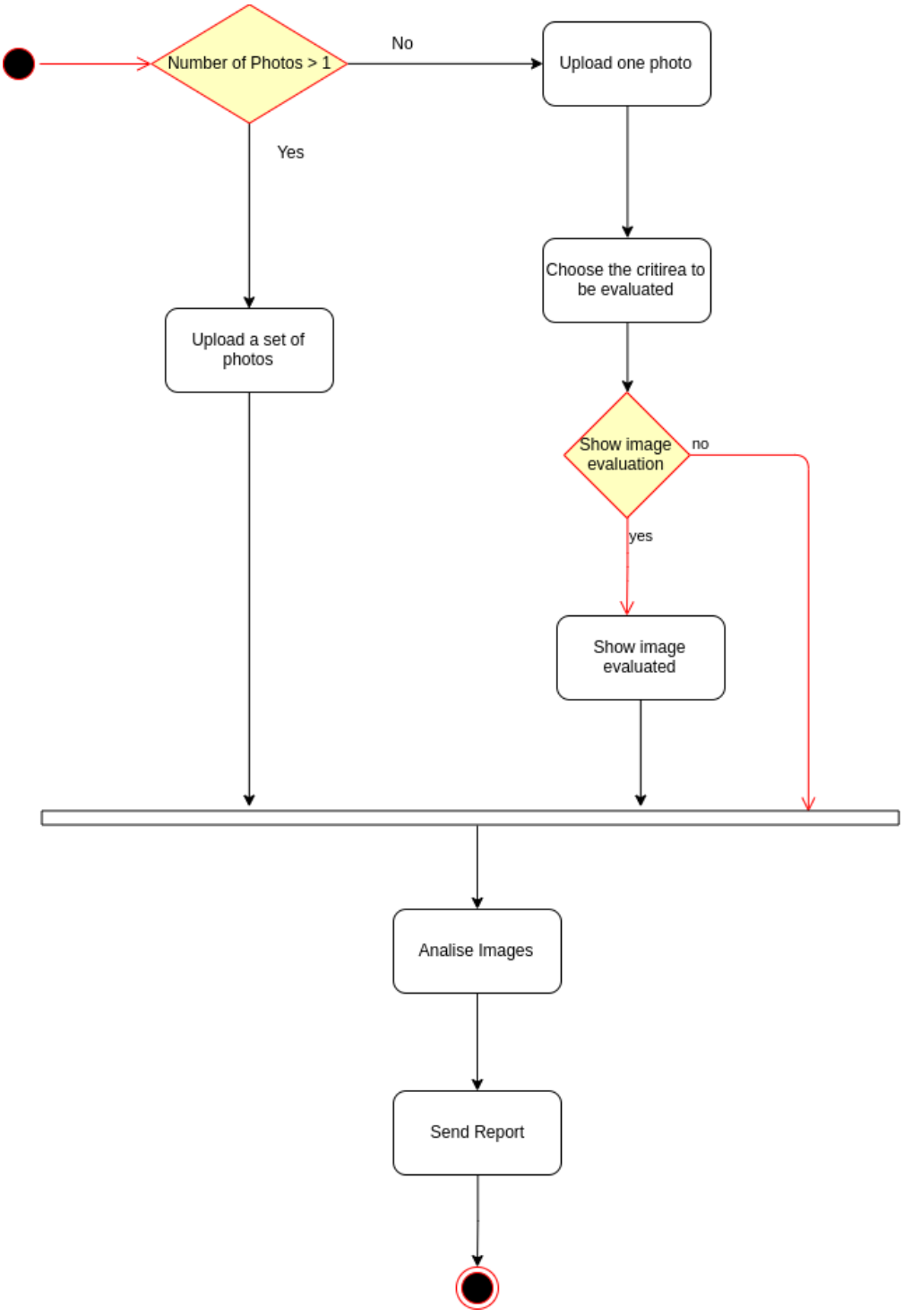

Figure 8 – Regular User Use Case Diagram

4.4.2. Administrator User

To use this API as an administrator role, is needed to authenticate first, provide username and password. After that, its available to perform three types of actions: manually inserting a photo by specifying its URL and entering its rating; change a rating, select a photo to correct the rating, and delete a photo if it is not necessary anymore in the database. After completing the desired actions, the administrator user should log out. In *Figure 9* we can represent the flow of all these tasks.

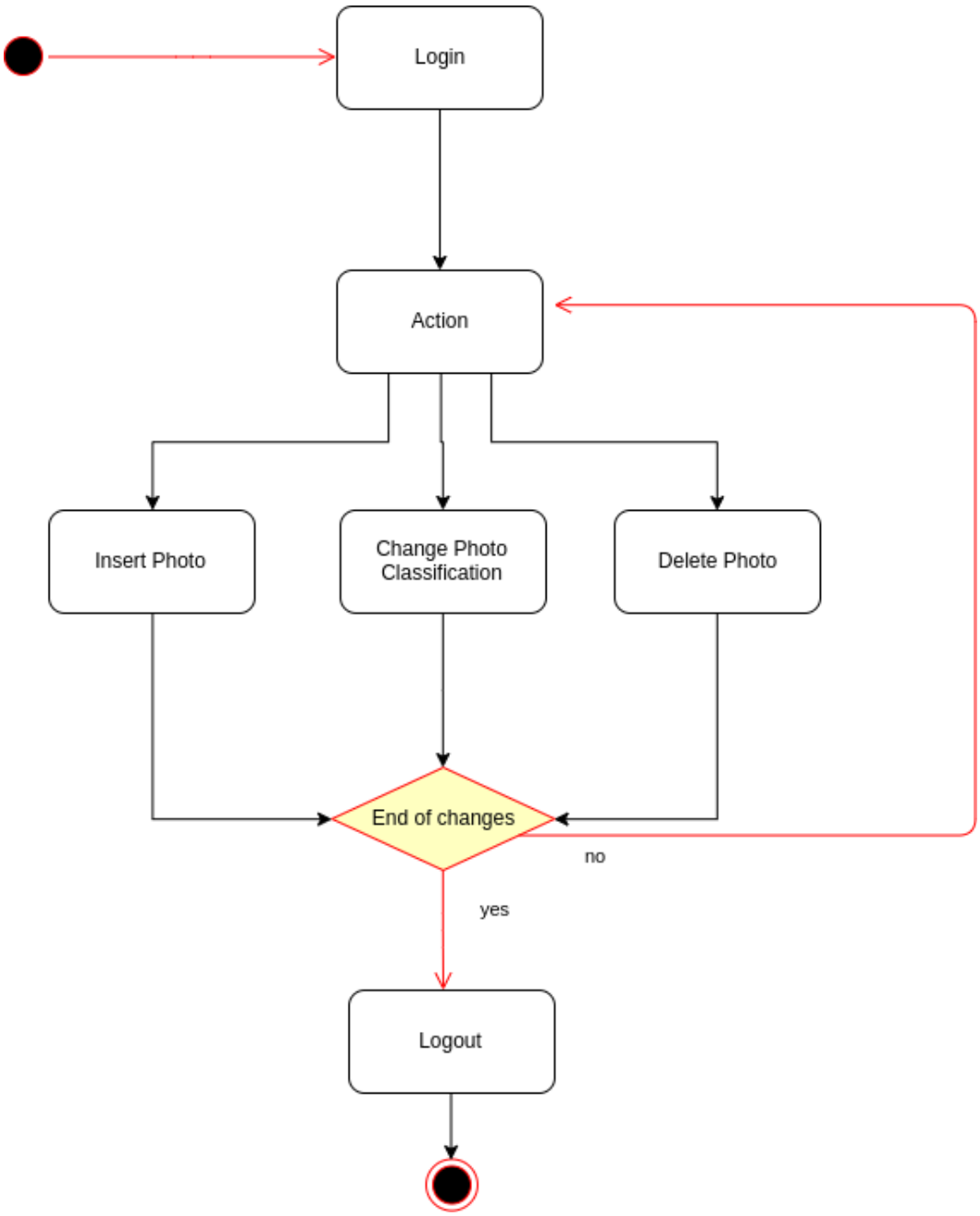

Figure 9 – Activity Administrator User Diagram

4.4.3. Choose the License

The last part about development is the choice of license to use. A software license is a legal instrument, usually by way of contract law, with or without printed material governing the use or redistribution of software.

Given the purpose of this project, the MIT license was chosen because it has no restrictions on the level of code handling. It is allowed to copy, modify, distribute and sell copies of the software, provided that only have the copyright notice and one copy of the license on all copies of the software.

5. Implementation and Deployment

To demonstrate the proposed architecture an API in Django Rest Framework was implemented, which allows to recognize and classify faces through algorithms and developed in different languages. To demonstrate this, it used languages like *Python* and *C++*, using *OpenCV* face recognition libraries. These languages are treated differently as regards their use. However, because *Python* is *Django's* implementation language, it is feasible to use and compile its algorithms directly in the application, but in the case of *C++* it is necessary to use a subprocess module that allows one to execute additional processes ('Subprocess Python', n.d.) [47].

After performing a photo classification process, it is stored in the photo database with its classification. This produces as output a response in JSON format and/or its image where the rating is indicated. If the user has an administrator profile, and it can change a photo rating, remove the photo, and manually insert a photo and its respective rating.

5.1. Django Rest Framework Implementation

The framework used to create this solution enables a fast development of Web APIs. An MVT (Model View Template) pattern is used which interacts by requesting a particular task from the user, which is mapped via a URL.

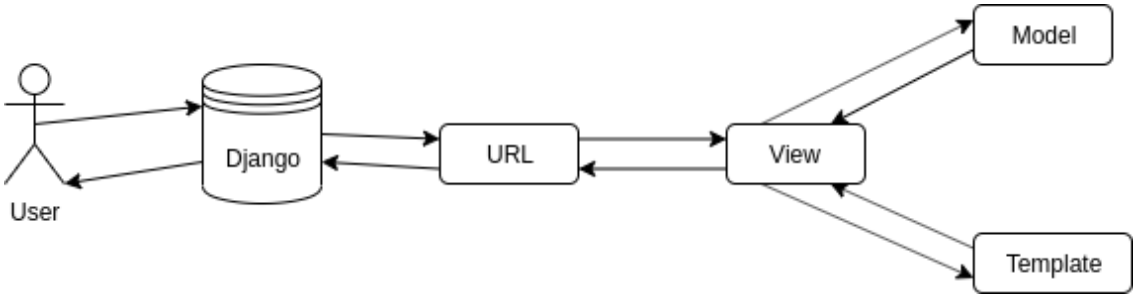

Figure 10 – API Implementation

This API incorporates functions that allow recognition of faces and classify their characteristics with respect to eyes, mouth, age and gender. These are stored in a database, for this purpose a model was built that defines an object that saves its classification and indicates where the image will be saved:

```
class Image(models.Model):
    descriptions = models.CharField(max_length=50)
    image = models.ImageField(upload_to='./pictures/%Y/%m/%d/',
    max_length=255, blank=True)
```

Once the template is created, it is necessary to implement a *ModelSerializer* to serialize and deserialize the data. For this a file *'serializers.py'* has been created a class *ImageSerializer*, where are the fields passed which will be performed to generate the objects:

```
class ImageSerializer(serializers.ModelSerializer):
    class Meta:
        model = Image
        fields = ('id', 'descriptions','image')
```

Now that we have the Model and the Serializer created, it is time to create the Views. In a file called views.py we place the functions that receive web requests and return web responses. There we have a *get* method for all images, a *post* method to insert new images and where we can implement algorithms to classify images:

```
#Number of faces in the image
def getFacesUrl(request, path="/home"):
    queryset = Image.objects.all()
    serializers_class = ImageSerializer
    path = '/' + path.replace('-', '/')
    image = face_recognition.load_image_file(path)
    face_locations = face_recognition.face_locations(image)
```

```
face_landmarks_list = face_recognition.face_landmarks(image)
list_of_face_encodings = face_recognition.face_encodings(image)
responseData = {
    'Faces': len(face_locations),
 }
return JsonResponse(responseData)
```

Here it is possible, due to open source features and the Python language capability, to add new algorithms from different languages to add new classifications to images. Through subprocesses we can generate new processes and invoke external algorithms to get or get their return code.

```
import subprocess

from subprocess import Popen, PIPE

proc = subprocess.Popen("/home/diogo/django/drf/01/a.out", shell=True)
```

Finally, it is necessary to create the URLs to access the views, in order to classify the images and to manage the API.

```
router = routers.DefaultRouter()
router.register(r'images',ImageViewSet)
urlpatterns = [
    path('', include(router.urls)),
    path('admin/', admin.site.urls),
    path('getfacesurl/<path>/',views.getFacesUrl),
    path('getfacesurlfolder/<path>/', views.getFacesUrlFolder),
     ...
```

This way we can have access to the admin interface provided by Dango REST Framework and to the image classification algorithms with just a few lines of code.

The following session presents screen shots of the developed prototype API.

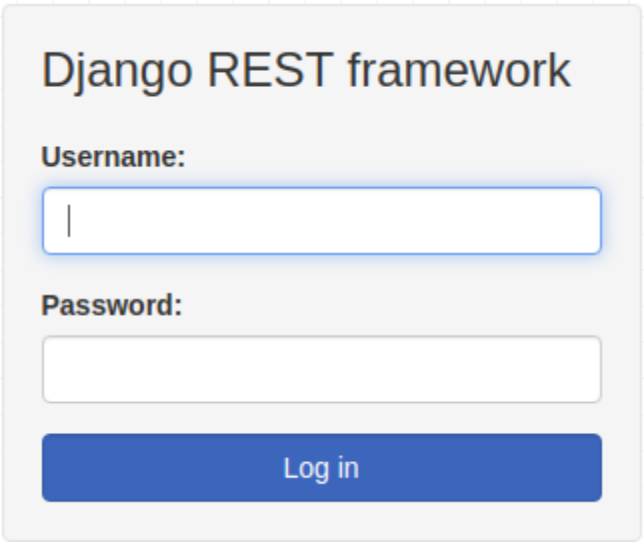

Figure 11 – Administrator Login

Figure 11 shows the Administrator login interface.

After of the administration login we can watch the list of images and manipulate them, in Figure 12.

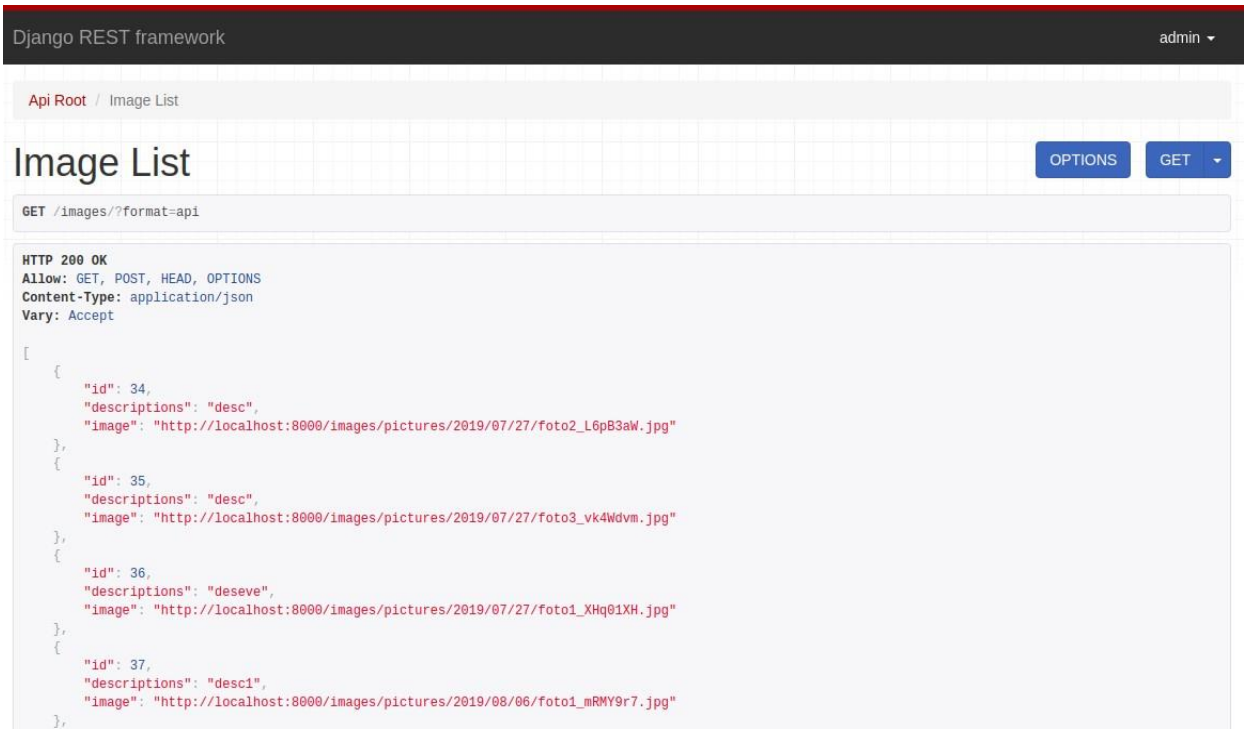

Figure 12 – Image List

With the Django-admin panel it is possible to manipulate the data easily.

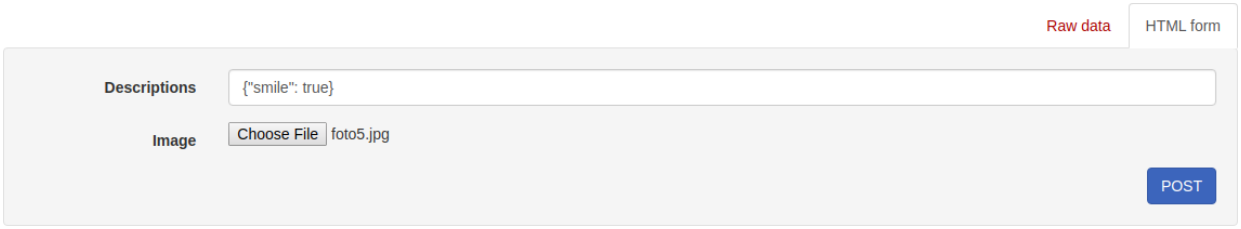

Figure 13 – Image Insert

The Regular User can recognize faces and classify some features, shown in Figure 14.

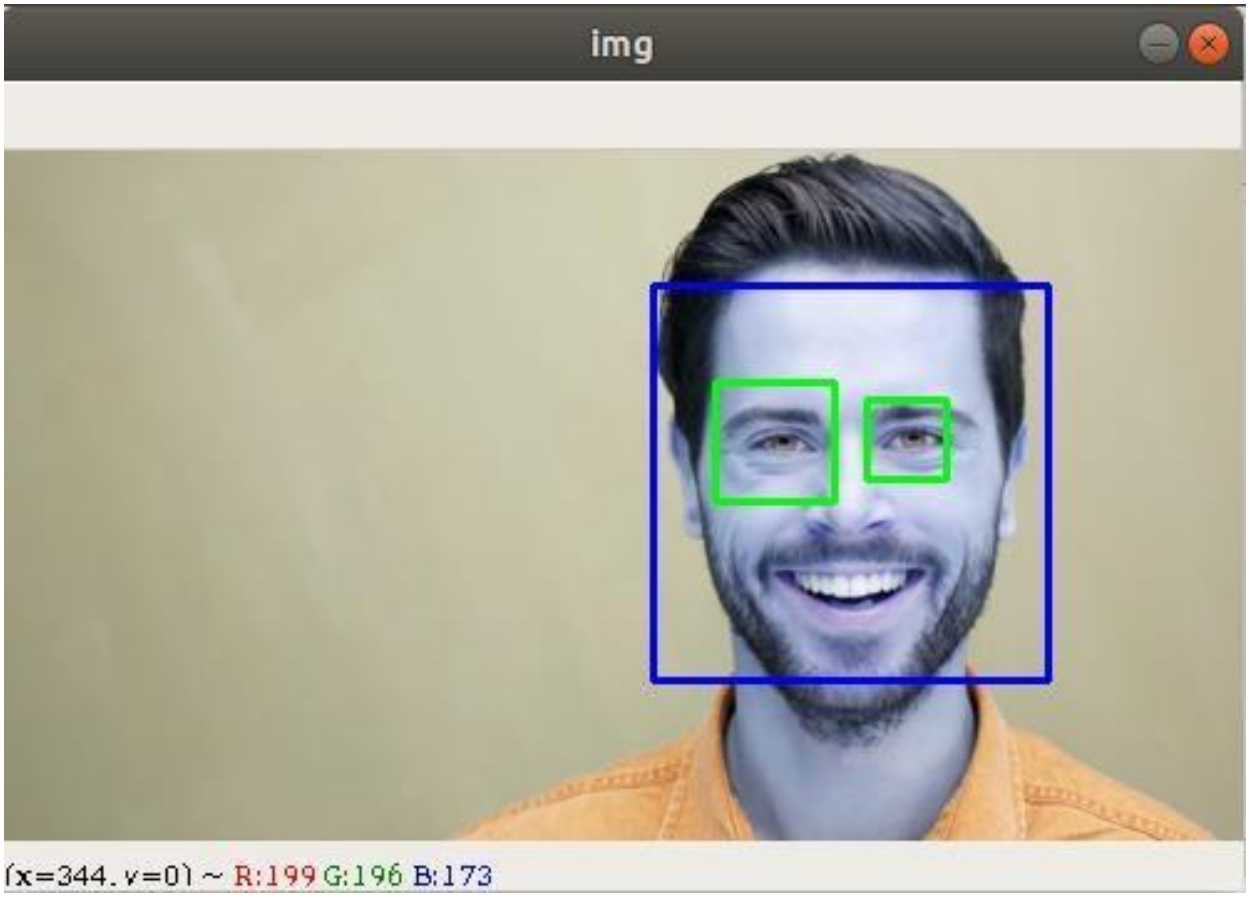

Figure 14 – Recognise Face and Eyes

The response will be given in JSON format.

{"Faces": 1, "Landmarks": [{"chin": [[348, 147], [348, 167], [351, 188], [355, 210], [363, 230], [374, 248], [388, 266], [404, 280], [425, 284], [446, 280], [462, 266], [476, 248], [488, 229], [495, 208], [498, 186], [499, 164], [500, 142]], "left_eyebrow": [[359, 140], [368, 130], [381, 127], [396, 127], [410, 131]], "right_eyebrow": [[433, 130], [448, 125], [462, 125], [476, 128], [484, 137]], "nose_bridge": [[422, 144], [422, 155], [422, 168], [423, 180]], "nose_tip": [[406, 189], [414, 192], [423, 195], [431, 192], [440, 189]], "left_eye": [[378, 147], [386, 143], [395, 143], [403, 148], [395, 149], [386, 149]], "right_eye": [[442, 147], [450, 141], [460, 141], [468, 146], [460, 148], [451, 148]], "top_lip": [[388, 211], [400, 208], [414, 208], [423, 210], [434, 208], [448, 208], [460, 211], [456, 212], [434, 212], [423, 213], [414, 212], [392, 213]], "bottom_lip": [[460, 211], [448, 229], [435, 237], [424, 238], [413, 237], [399, 228], [388, 211], [392, 213], [413, 228], [423, 230], [434, 228], [456, 212]]}]}

Figure 15 – Landmarks Response

Smile is one of the features which its able to analyze. In this API it is possible to preview the analysis (in Figure 16) before the JSON response or ask just for the JSON response.

Figure 16 – Smile Analisys

The generated response is: {“smile”: true}

Figure 17 shows other classification of the smile feature.

Figure 17 – Smile Analysis(2)

The generated response is: {“smile”: false}

5.2. Technologies Used

Django

It is a free framework and open source written in Python and provides a ready-to-use management dashboard. It allows quick development uses the DRY principle and defines an ORM data modeling, with it allows *spread sheet* to be generated in the database without manipulating them through SQL. Published in 2005 with a BSD license, it allows its unrestricted use and incorporation into other products, whether they are open source or proprietary code.

Python

It is a high-level / object-oriented language, with simple and easily readable syntax. Uses a dynamic data type and resource for multiple library calls. In this project, more emphasis was placed on libraries that cover classes that allow data processing for facial analysis such as OpenCV (Maureira, n.d.) [31].

C++

It is a programming language that adds object-oriented features to its predecessor C. In the context of this project I was used to demonstrate the API's ability to work with different language algorithms from Django's native language.

JSON

It is a light format, simple and easy to read. Based on the JavaScript language, it is used for exchanging information between systems.

MySQL

It is the world's most popular Open Source relational database management system. Due to its greater realability and ease of use when compared to Sqlite which is the native django system, it was chosen to implement it in this project.

To enable the use of a MySQL Database in a Django application is one to uses the following settings in the Database dictionary:

```
DataBases = {
        'default' : {
                'ENGINE' : 'django.db.backends.mysql',
                'NAME' : 'DB_NAME',
                'USER': 'DB_USER',
                'PASSWORD': 'DB_PASSWORD',
                'HOST': 'localhost',
                'PORT': '3306',
        }
}
```

Similarly, it is also necessary to create the file *'/path/to/my.cnf"* with the following settings:

```
[client]
database = DB_NAME
host = localhost
user = DB_USER
password = DB_PASSWORD
default-character-set = utf8
```

5.3. Functionalities

To show the features available to users of this API, the following two figures to illustrate two common users: regular user and an Administrator.

A regular user using this API has the ability to perform diverse analysis regarding facial recognition and classification. Each of the existing features that can be performed by an individual image or by a set of images. Tasks that are performed will be saved in the Database.

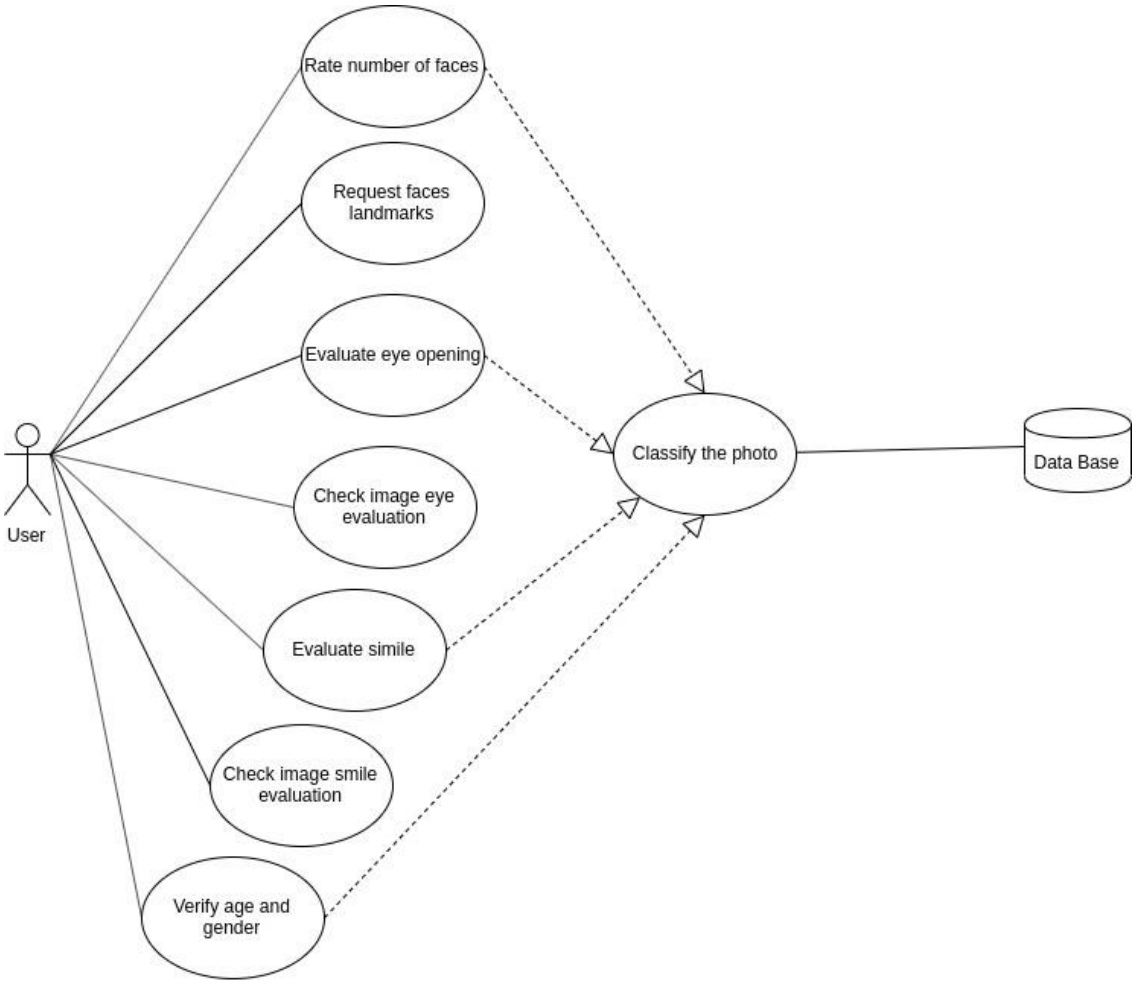

Figure 18 – Regular User Use Case Diagram

In case of a user that has the administrator profile to access the management interface, a login is required. After the account has been validated a permission is granted to perform administrative tasks and the ability to change data generated by previously performed tasks.

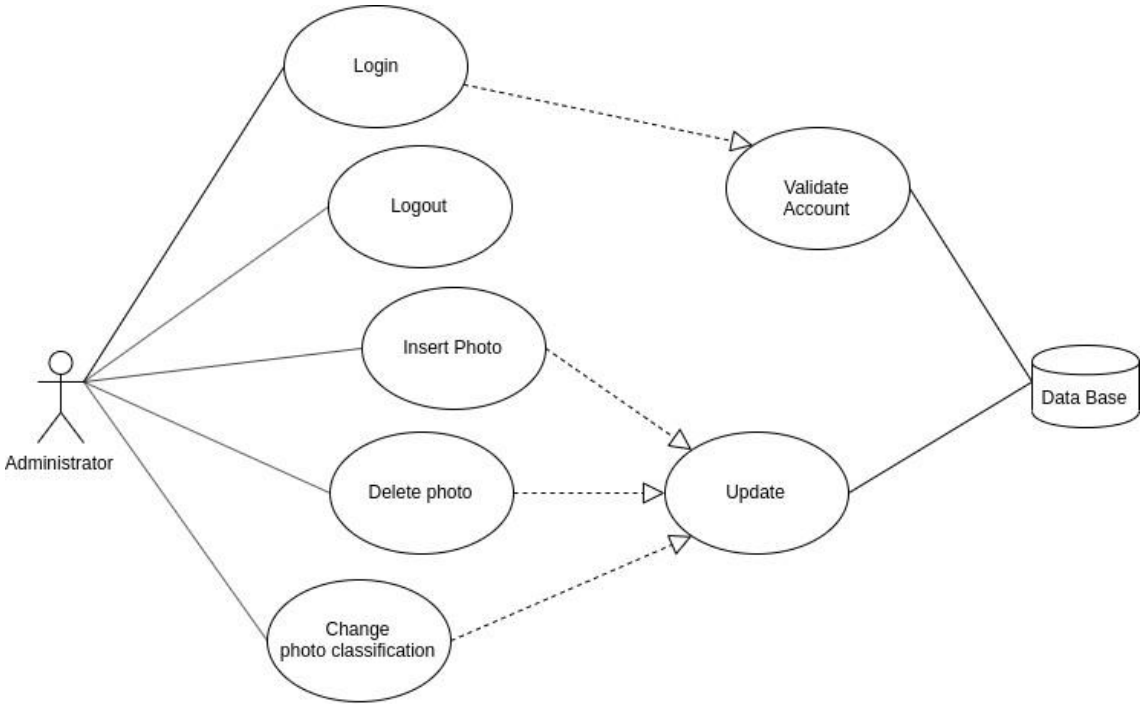

Figure 19 – Administrator Use Case Diagram

6. Conclusion

The solution presented for the elaboration of this work was carried out in parallel with a Web API that was being built, aiming at the study and development of an Open Source API, which incorporates several algorithms, enabling recognition and classification of faces. It uses the Django Rest Framework Simplified Development to have the REST architecture implemented natively. Python is the implementation language, having a simple and object oriented syntax, which provides an easy readability, and also contains the ability to use algorithms compiled in other programming languages via sub processes, thus enabling the API with this flexibility.

Regarding the difficulties encountered during the preparation to decide the best way to do realize this task, we relate the learning of technologies for this realization. Another conditioning factor was the requirements having undergone some changes, due to the development options made throughout the project.

For future work it is suggested to use this API in other systems or applications, the development of new features and the optimization of existing ones, to correspond to user's needs. This API can be useful for professional, academic or personal issues. It allows to be integrated in Open Source or Proprietary Applications to enrich their functionalities. To consult and use this API one may access the repository at *https://github.com/DiogoNeto/ImageClassificationAPI.git*.

References

[1] Ahonen, T., Hadid, A., & Pietikäinen, M. (2006). Face description with local binary patterns: Application to face recognition. IEEE Trans. Pattern Anal. Mach. Intell, 28(12), 2037– 2041.

[2] Amazon AWS Rekognitions. (n.d.). Retrieved 10 July 2019, from https://aws.amazon.com/pt/rekognition/

[3] Animetrics Face API. (n.d.). Retrieved 10 July 2019, from http://animetrics.com/

[4] Bartlett, M. S., Movellan, J. R., & Sejnowski, T. J. (2002). Face recognition by independent component analysis. IEEE Trans. Neural Netw, 13(6), 1450–1464.

[5] Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. 19(7), 711–720.

[6] BetaFace API. (n.d.). Retrieved from https://www.betafaceapi.com/wpa/

[7] Bolzmann Machine. (n.d.). Retrieved 11 May 2019, from http://proceedings.mlr.press/v5/salakhutdinov09a/salakhutdinov09a.pdf

[8] C. Ding, & D. Tao. (2016). A comprehensive survey on pose-invariant face recognition. ACM Transactions on Intelligent Systems and Technology (TIST), 7(3), 37.

[9] Caffe. (n.d.). Retrieved 28 April 2019, from https://caffe.berkeleyvision.org/

[10] Chi-Fa, C., Yu-Shan, T., & Chia-Yen, C. (2013). Combination of PCA and Wavelet Transforms for Face Recognition on 2.5D Images. Department of Electrical Engineering, I-Shou Univeristy, Kaohsiung, Taiwan cfchen@isu.edu.tw 2CITR, Tamaki Campus, The University of Auckland, Auckland, New Zealand.

[11] Computer Vision and Image Understanding (Statistics Department, University of Nebraska–Lincoln, NE 68583-0712, USA). (2015). Retrieved from https://www.sciencedirect.com/science/article/pii/S1077314205001761

[12] CUDA. (n.d.). Retrieved 3 April 2019, from https://developer.nvidia.com/cuda-zone

[13] DBN. (n.d.). Retrieved 3 April 2019, from http://www.cs.utoronto.ca/~gdahl/papers/dbnPhoneRec.pdf

[14] Deep Learning: The Confluence of Big Data, Big Models, Big Compute. (n.d.). Retrieved 27 August 2019, from https://www.datanami.com/2019/01/10/deep-learning-the-confluence-of-big-data-big-models- big-compute

[15] Deeplearning4J. (n.d.). Retrieved 9 April 2019, from https://deeplearning4j.org/

[16] Face ++. (n.d.). Retrieved 10 July 2019, from https://www.faceplusplus.com/

[17] Face Recognition Using Evolutionary Pursuit. (n.d.). Retrieved 7 August 2019, from http://www.face rec.org/algorithms/EP/liu98face.pdf

[18] Gates, B. (1976). An Open Letter To Hobbyists. Retrieved from Homebrew Computer Club Newsletter. (Mountain View, CA)

[19] General Public License—GPL. (1989). Retrieved from https://www.gnu.org/licenses/old-licenses/gpl-1.0.html

[20] Google Cloud Platform. (n.d.). Retrieved 10 July 2019, from https://cloud.google.com/vision/

[21] Google TPU - https://cloud.google.com/tpu/. (n.d.). Retrieved from https://cloud.google.com/tpu/

[22] He, X., Yan, S., Hu, Y., Niyogi, P., & Zhang, H. -J. (2005). Face recognition using Laplacian faces. IEEE Trans. Pattern Anal. Mach. Intell., 27(3), 328–340.

[23] HTTP Methods. (n.d.). Retrieved 29 June 2019, from https://restfulapi.net/http-methods/

[24] IBM Model Asset Exchange (Max). (n.d.). Retrieved 21 April 2019, from https://developer.ibm.com/exchanges/models/all/max- image-segmenter/

[25] IBM Watson Face. (n.d.). Retrieved from https://www.ibm.com/cloud/watson-visual-recognition

[26] Intel Porcessing Graphics. (n.d.). Retrieved 8 July 2019, from https://software.intel.com/en-us/articles/intel-graphics- developers-guides

[27] Kairos API. (n.d.). Retrieved from https://www.kairos.com/docs/

[28] Keras. (n.d.). Retrieved 9 July 2019, from https://keras.io/

[29] Liu, D. H., Lam, K. M., & Shen, L. S. (2005). Illumination invariant face recognition. Pattern Recognition, 38(10), 1705–1716.

[30] Lu, J., Plataniotis, K. N., & Venetsanopoulos, A. N. (2003). Face recognition using kernel direct discriminant analysis algorithms. , IEEE Trans. Neural Netw, 14(1), 117–126.

[31] Maureira, C. (n.d.). Euro Python Talk 2019. Retrieved from https://github.com/cmaureir/unleash_cpp - Euro Python Talk 2019

[32] Meerkat FR. (n.d.). Retrieved 10 July 2019, from https://www.meerkat.com/

[33] Microsoft Cognitive Face API. (n.d.). Retrieved 10 July 2019, from Microsoft Cognitive Face API - https://azure.microsoft.com/pt-pt/services/cognitive- services/face/

[34] Naseem, I., Togneri, R., & Bennamoun, M. (2010). Linear regression for face recognition. IEEE Trans. Pattern Anal. Mach. Intell, 32(11), 2106–2112.

[35] Nvidia AI Computing. (n.d.). Retrieved 13 March 2019, from https://www.nvidia.com/en-us/about-nvidia/ai-computing/

[36] Open Source Initiative. (n.d.). Retrieved 6 August 2019, from https://opensource.org/licenses

[37] OpenCL. (n.d.). Retrieved 23 July 2019, from https://opencv.org/opencl/

[38] OpenCV. (n.d.). Retrieved 20 December 2018, from https://opencv.org/about/

[39] OpenMP. (n.d.). Retrieved 25 June 2019, from https://www.openmp.org/

[40] Protobuf. (n.d.). Retrieved 27 September 2019, from https://github.com/protocolbuffers/protobuf

[41] Pytorch. (n.d.). Retrieved 9 May 2019, from https://pytorch.org/

[42] RBM. (n.d.). Retrieved 3 April 2019, from http://proceedings.mlr.press/v15/courville11a/courville11a.pdf

[43] Red Hat: What is an API? (n.d.). Retrieved 19 August 2019, from https://www.redhat.com/en/topics/api/what-are-application- programming-interfaces

[44] Stallman, R. (1983). New UNIX implementation. net.usoft. (Newsgroups: net.unix- wizards).

[45] Stallman, R. (2013, June 18). What is free software? Retrieved from Gnu.org

[46] Stallman, R. (n.d.). GNU Manifesto. Retrieved from https://www.gnu.org/gnu/manifesto.html

[47] Subprocess Python. (n.d.). Retrieved 17 July 2019, from https://docs.python.org/3/library/subprocess.html

[48] Tensorflow. (n.d.). Retrieved 9 April 2019, from https://www.tensorflow.org/

[49] The Model-View-Controller Design Pattern. (n.d.). Retrieved from https://djangobook.com/mdj2-django-structure/

[50] Turk, M., Pentland, A., & J. Cogn, N. (1991). Eigenfaces for recognition. pp. 71– 86.

[51] U. Park, Y. T., & A. K. Jain. (2010). Age-invariant face recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(5), 947–954.

[52] Xiaofei, H., & Partha, N. (2003). Locality preserving projections. Int. Conf. on Advances in Neural Information Processing Systems (NIPS'03), 153–161.

[53] Yang, B., & Chen, S. (2013). A comparative study on local binary pattern (LBP) based face recognition: LBP histogram versus LBP image. Neurocomputing, 22, 620– 627.

[54] Zhang, B., Gao, Y., Zhao, S., & Liu, J. (2010). Local derivative pattern versus local binary pattern: Face recognition with high-order local pattern descriptor. IEEE Trans. Image Process, 19(2), 533–544.

[55] Zhang, B., Shan, S., Chen, X., & Gao, W. (2007). Histogram of Gabor phase patterns (HGPP): A novel object representation approach for face recognition. IEEE Trans. Image Process, 16(1), 57–68.