Bachelor's Thesis

DEGREE IN COMPUTER SCIENCE AND SOFTWARE ENGINEERING

Faculty of Mathematics and Computer Science
University of Barcelona

WEBFACE: design of an identity verification system in PHP

Sergi Díaz i Planas

Director: Simone Balocco

Submitted to: Department of Mathematics and Computer Science

Barcelona, February 2018

Acknowledgments

I would like to express my sincere gratitude to Simone Balocco for accepting to supervise this thesis. His continuous support has been instrumental in the realization of the project.

I also want to thank my family for their love and comprehension, especially during the last months.

Abstract

As web browsers improve their capabilities and customers get access to faster internet connections, the market is experiencing an increase on the services located in the web. This has opened the gate for the implementation of systems that, not long ago, used to seem unfeasible due to the technical limitations intrinsic to client-server applications. In this context, the project presents a design of an identity verification system integrated on a web application programmed in PHP. The implemented web-server user registration makes use of face detection and optical character recognition techniques to gather the information needed to verify the identity of the user. Such decision is made by using a deep neural network, and requires the user to provide an identity document. In order to simplify, the only document supported is the most recent version of the Spanish national identity document, referred as DNI 3.0.

The scope of this study is use existing libraries and machine learning techniques (including trained classifiers) for the purpose of identity validation, and not not to modify and re-train the classifiers.

Abstracto (Castellano)

A medida que se dispone de navegadores web más capaces y conexiones de Internet más rápidas, el mercado experimenta un aumento en la cantidad de servicios alojados en la red. Esto abre la puerta a la implementación de sistemas que hasta ahora parecían imposibles por las dificultades técnicas propias de las aplicaciones cliente-servidor. En este contexto, el proyecto propone el diseño de un sistema de verificación de identidad integrado en una aplicación web programada en PHP. La implementación hace uso de técnicas de detección facial y reconocimiento óptico de caracteres para verificar la identidad del usuario. Esta decisión se toma mediante el uso de una red neuronal profunda, y requiere que el usuario proporcione un documento de identidad.

Para simplificar, sólo se da soporte a la versión más reciente del documento nacional de identidad español, conocido como DNI 3.0.

Abstracte (Català)

A mesura que es disposa de navegadors web més capaços i connexions d'Internet més ràpides, el mercat experimenta un augment en la quantitat de serveis allotjats a la xarxa. Això obra la porta a la implementació de nous sistemes que fins ara, degut a les limitacions tècniques pròpies de les aplicacions client-servidor, semblaven impossibles. En aquest context, el projecte proposa el disseny d'un sistema de verificació d'identitat integrat en una aplicació web programada en PHP. La implementació utilitza tècniques de detecció facial i reconeixement òptic de caràcters per verificar que la identitat de l'usuari. La decisió es pren mitjançant una xarxa neuronal profunda, i requereix que l'usuari aporti un document d'identitat.

Per simplicitat, tan sols es dóna suport a la versió més recent del document nacional d'identitat espanyol, conegut com a DNI 3.0.

Contents

1. Introduction

1.1. Motivations

From performing bureaucratic procedures to accessing online banking, there are many situations where a provider of web services needs to verify the identity of its customers. Typically, the needs and demands of both service provider and client lie at opposing ends. As service providers, we need to be sure that the identity of the user is legit and belongs to a real person. As customers, though, we want to spend as little time as possible on these identity validation processes. Filling long forms or demanding the customer to provide multiple documents might be a secure way to verify the identity, but are valued negatively by customers because they are slow and unengaging.

In this project, I want to design an identity validation method that uses computer vision to extract the information needed, rather than depending on the user to provide it. From a business view, being able to offer a better user experience should grant an edge against competitors. As a developer, the project gives me the opportunity to engage in some of the most cutting-edge fields of the moment: artificial vision and machine learning. As for the server language, I have decided to use PHP. The decision stems from both this being the most extended language for server programming[1] and the desire to improve my skills on it.

1.2. State of the art

The benefits provided by computer vision tools have incentivated the emergence of multiple business initiatives related to the development of frictionless, user-friendly identity verification systems. Due to the change in market trend, where the usage of web based multi platform applications have gained the upper hand over desktop applications, most of these initiatives are developing their products for web environments.

I have studied some products offered by competitors in the field of face-oriented identity verification systems and most of them appear to work similarly:

1. The user takes a picture of an identity document and uploads it to the application.
2. The application checks the format of the document.
3. Finally, the user is requested to take a picture of his or her face and upload it. The application compares this picture with one found in the previous document to verify that it belongs to the user.

As these are licensed products the information we have about how they work is limited to that which the owner decides to share, making it difficult to compare them. The following table brings together some pros and cons about the products, deducible from the information I could gather.

Product	*Developer*	*Pros*	*Cons*
Onfido	Onfido		Vulnerable to spoofing attacks. The user has to manually select the type of document in use.
Netverify	JUMIO	Uses a liveness detection.	
Cloud_ID	icar	Uses a liveness detection.	

Table 1. List of products that use face recognition methodologies as a mean of validation in user registration. For each product, the table shows its developer and its advantages ("pros") and disadvantages ("cons") over the other.

Now, the table has introduced two terms that might need some clarification:

- A spoofing attack is a souplantation of the identity by using 2D media, typically pictures (both in paper and in a screen) and video. Basically, the attacker tries to cheat the validation system by using an image of the same person that appears in the document.

- Liveness detection tests are regular protection measures against spoofing attacks in face recognition.

1.4. Objectives

The objective of this project is to design a web based client-server application that allows users to make a registration. A server has to use face oriented computer vision to ensure that the user is a real person. Regarding the client’s view, the look & feel and the usability will not be taken into account when evaluating the compliance of the objectives. The same applies to the quality of the communication between client and server. From this, it has to be inferred that the project will focus on the design of the validation system and its integration in the server. The following tables present the requirements of the application, functional and non-functional, sorted by priority.

1.4.1. Functional requirements

1	The server has to be able to recognise human faces in digital images.
2	The server has to be able to discern if two faces belong to a same person.
3	The server has to be able to read text from digital images.
4	the client has to be able to complete a registration.
5	The server has to be able to validate DNI 3.0 documents based on its number and letter.

1.4.2. Non-functional requirements

1	The verification system has to be accurate. That means that the decisions made by the server regarding the authenticity of the user's identity has to be right.
2	The design has to be structured to accommodate change. That means that it has to allow future improvements on the client's side, as well as allowing new features to be included easily.
3	The application has to be fast enough so that the time it takes for the user to complete the whole registration process is within the order of minutes. Then, the time it takes for the server to perform its tasks is to be taken into account when deciding which technologies are to be used.

2. Planification

2.1. Methodology

Three stages have been defined for the project: the investigation stage, the implementation stage, and the results stage.

The i**nvestigation stage** comprehends the analysis of requirements and the technologies available to implement them. The stage proposes four problems that have to be approached to fulfill the objectives, sorted by priority:

1. The **integration problem** englobes the necessity of being able to use computer vision solutions from a web server.
2. The **face verification** problem englobes the necessity of being able to determine whether to face images belong to a same individual or not.
3. The **face detection problem**, or the necessity of being able to find human faces in digital images. It was estimated for this problem to be less complex than face verification, hence the lower priority.
4. The **reading problem**, also referred as the optical character recognition problem. Comprises the need of being able to read text from digital images.
5. Set at a low priority level, the **liveness detection** problem defines the necessity of protecting the verification system against spoofing attacks.

The state of the art regarding each of the problems has been studied to decide how and by using which technologies could they be solved. The decision making has been based on experimental results taken with prototypes of the application, also referred as proofs of concept.

The **implementation stage** comprises both the design of the application and the code writing. The stage has been approached following the agile philosophy, and subsequently organized in multiple, four days long sprints. A functional version of the application has been released at the end of each sprint.

The **results stage** evaluates the fulfillment of the objectives based on a set of tests on the application. The compliance to the planification and the methodology has been also evaluated in this stage. Finally, all the documentation generated in the previous stages has been gathered, revised and compiled into a project written memory.

Regardless the stage, all the advances have been communicated to the project director via email and regular meetings.

2.2. Time expenditures

The project was intended to last 4 months, with the hours of work distributed homogeneously between the four stages. The initial planning and its time expenditures are summarized in the following Gantt chart.

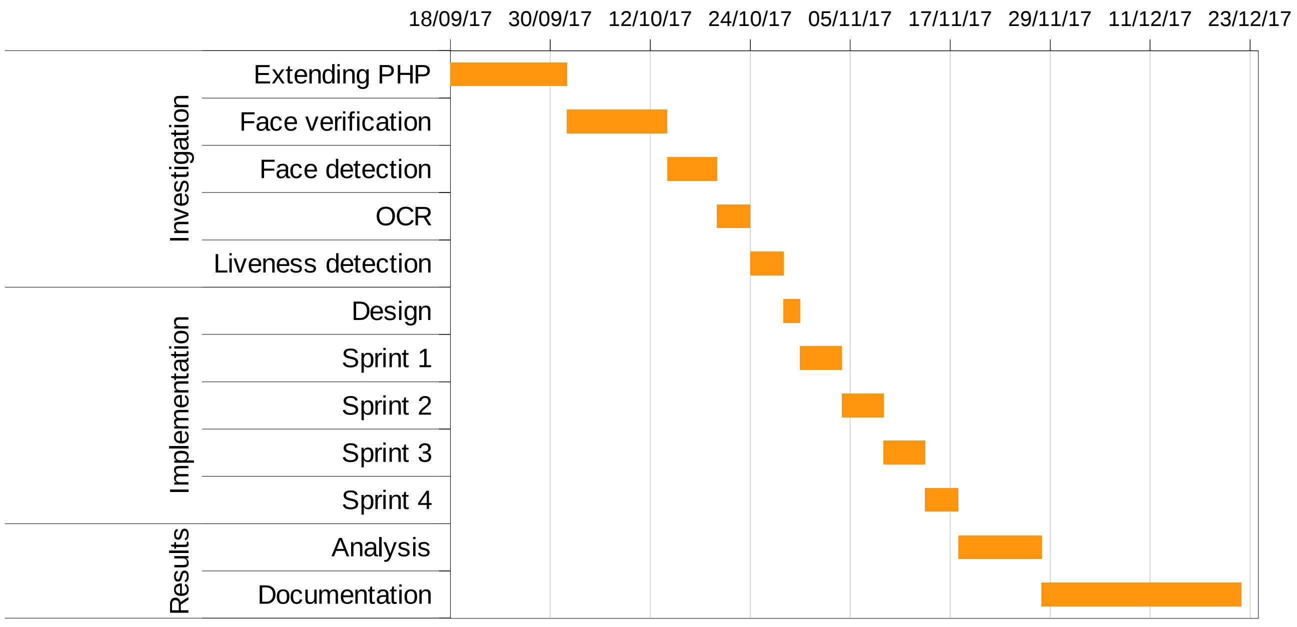


Figure 1. Gantt chart representing the initial estimation of the time expenditures.

Due to the unpredictable nature of the investigation stage, the time dedicated to the stages ended differing from the original. The final time expenditures can be found in the following Gantt charts, classified by stage. As for the causes of such deviation, those are being discussed on Chapter 5 - Conclusions.

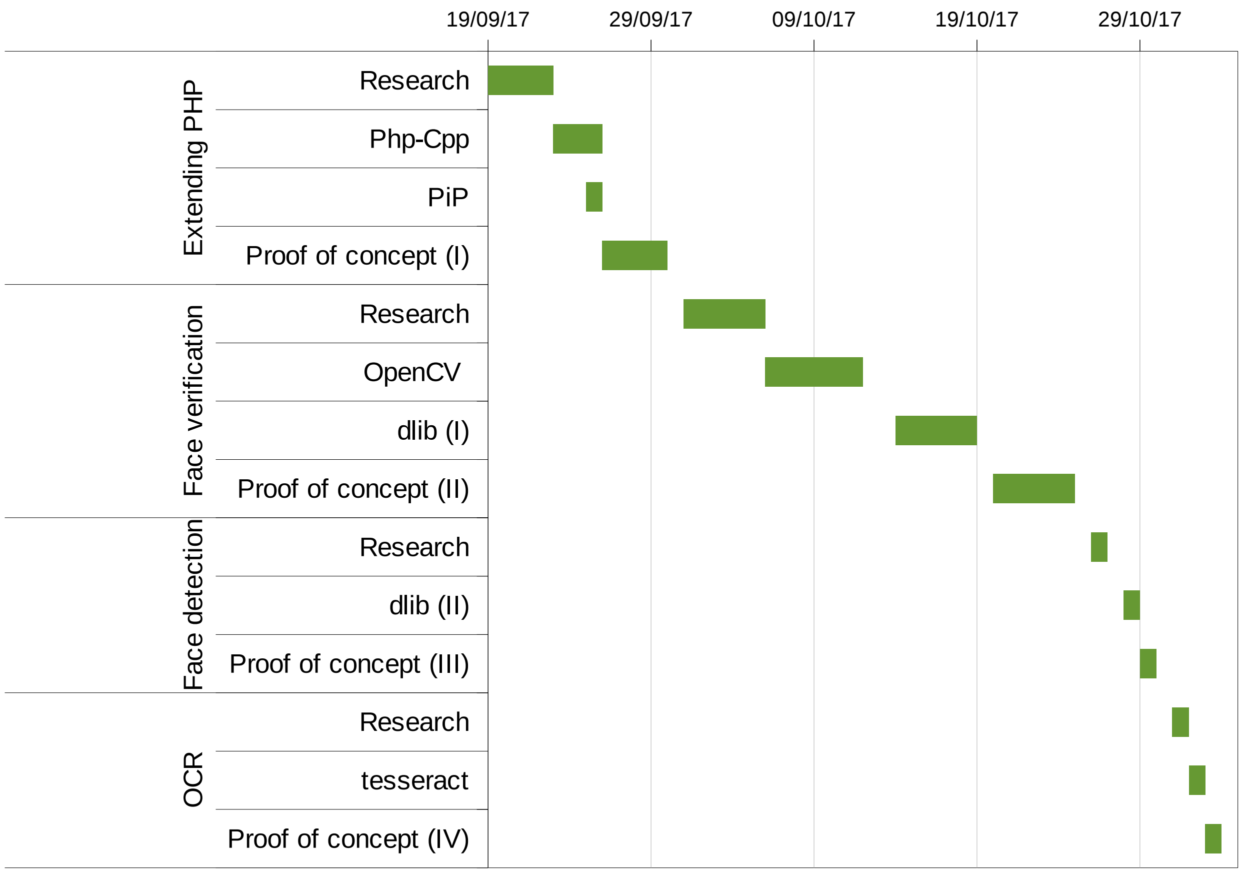


Figure 2. Gantt chart for the time expenditures regarding the investigation phase of the project, disglosed as a series of tasks.

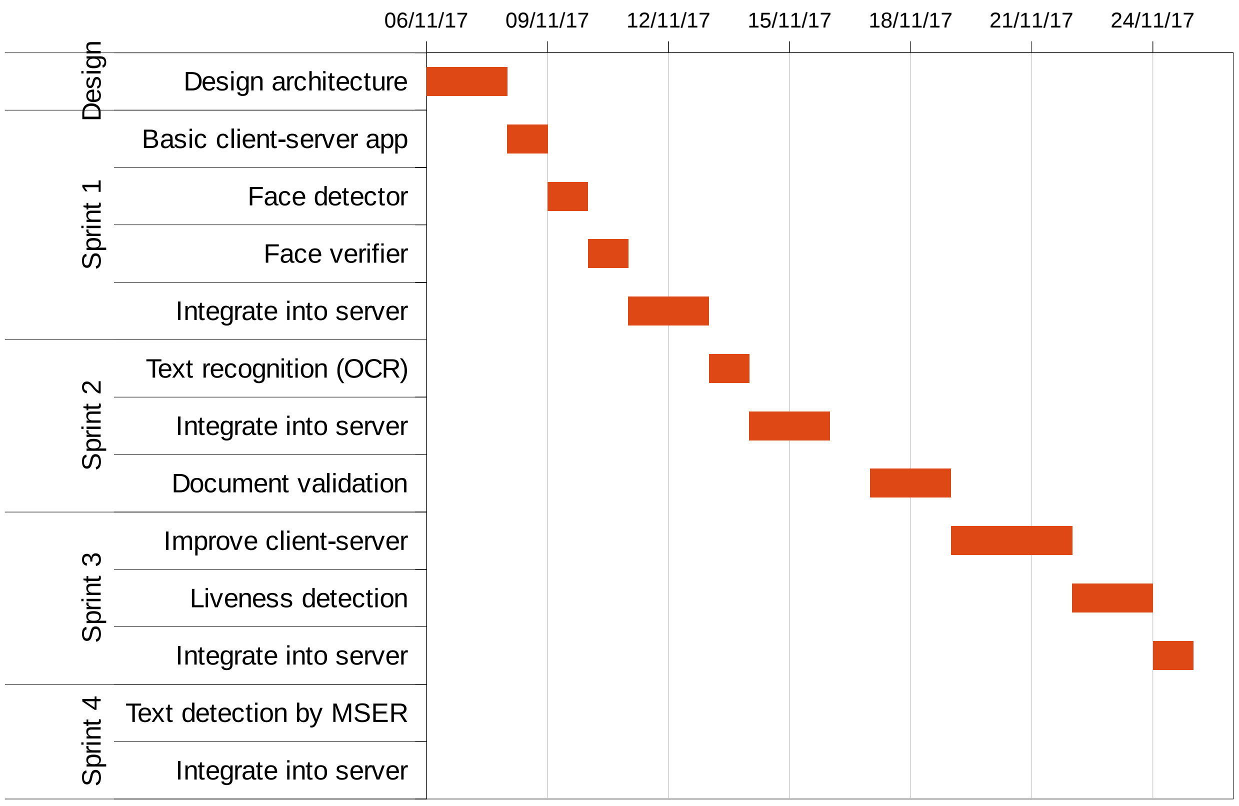


Figure 3. Gantt chart for the time expenditures regarding the implementation phase of the project, disglosed as a series of sprints.

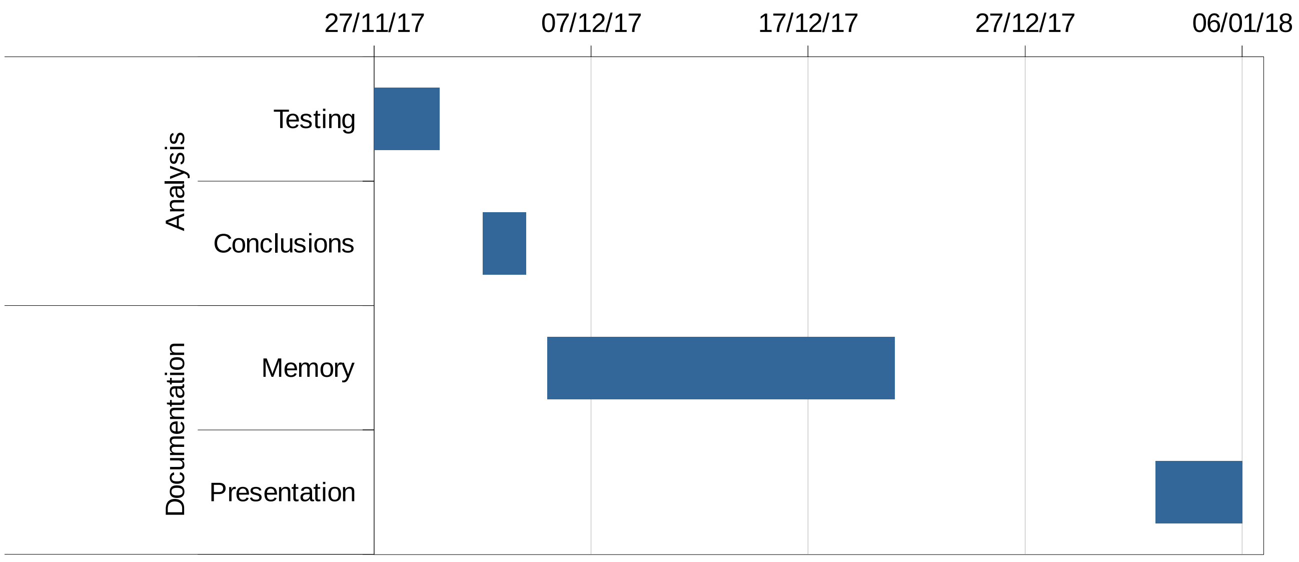


Figure 4. Gantt chart for the time expenditures regarding the analysis and documentation stages.

3. Implementation

3.1. Architecture

The architecture of the application responds to a multilayered design. As developers, this allows us to modify different sections of the architecture without affecting the whole implementation, making for a more scalable design. Four layers have been designed.

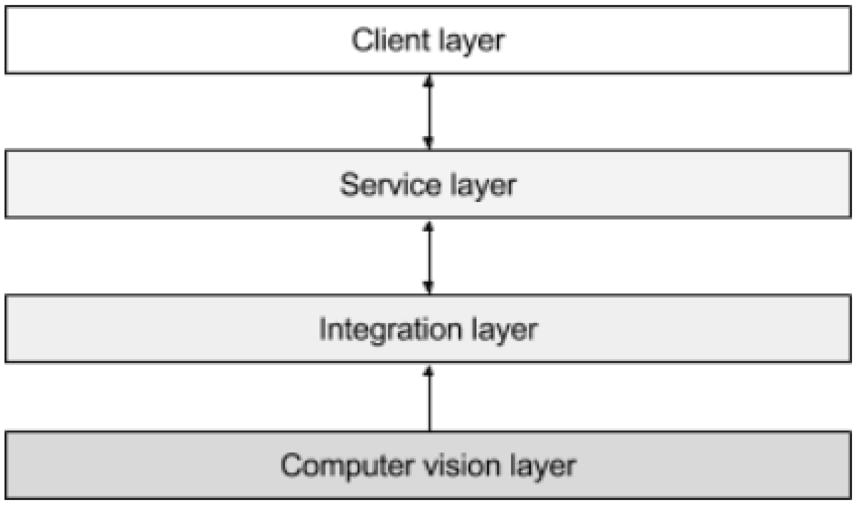


Figure 5. Representation of the multilayered architecture of the application.

Nearest to the client lies the client layer, which includes the view visible for the user and all the logic associated to it, including the communication with the server. The service layer packs the logic responsible for attending client’s requests. The server has to make use of computer vision techniques in order to attend the requests. The integration layer communicates the service layer with these resources, which are contained within the computer vision layer.

The problem of identity validation through artificial vision implies that the system has to be able to find faces in digital images, it has to be able to determine if two image faces belong to the same person and it has to be able to read text from images. From now on, this problems will be referred as face detection, face verification and optical character recognition (OCR). The architecture splits the solution to the aforementioned problems in three components:

- **Detector component**, as a face detection solution.
- **Verifier component**, as a face verification solution.
- **Reader component,** as an OCR solution

Additionally, two data structures have been implemented to allow the components to share information between them and with the server.

- **Image**, as a representation of a digital image understandable inside the application context. In order to support communication between client-server layers, Image objects can be parsed from base 64 encoded strings, and converted back to base 64.
- **Face**, as an extension of Image. Stores the coordinates of both eyes alongside the face image.

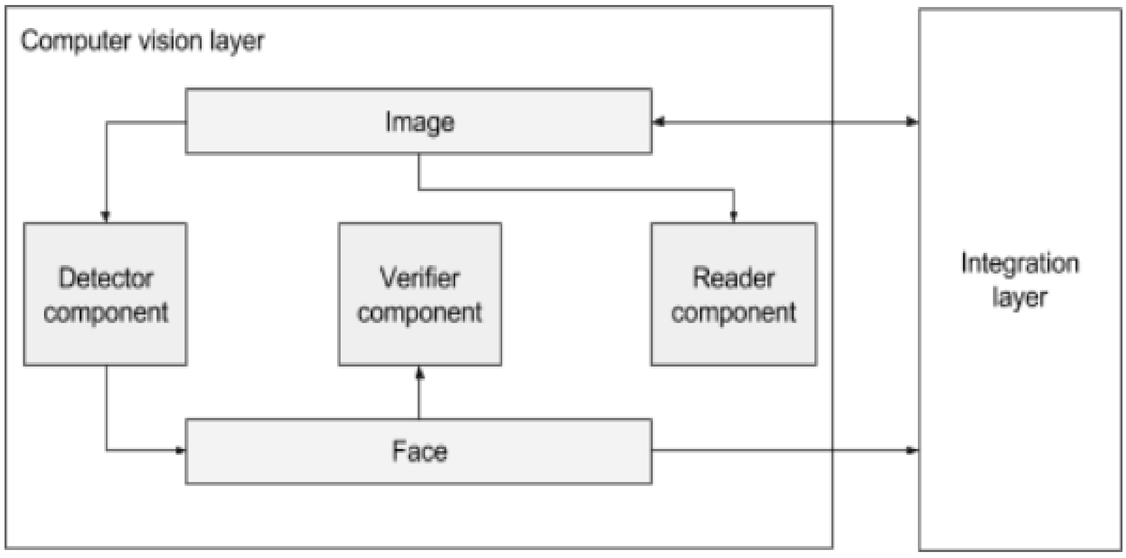


Figure 6. Representation of the internal structure of the computer vision layer.

At a code level, each of the components and data structures in the computer vision layer has been encapsulated into a class.

Before delving into the implementation of the layers and components, there are some particularities about the problems the project faces and the technologies chosen to solve them. The following chapters have been organized thematically by problem, providing a detailed view of the technologies and the implementation of the associated layers and components.

3.2. Client and service layers

The application has been distributed following a client-server structure. As stated in Chapter 1.4, nor the look & feel and the usability, nor the quality of the communication between client and server are being considered as objectives of the project. Taking this into account, the implementation of client and server layers poses two problems:

1. A logic for the server has to be designed to be able to attend client’s requests.
2. The computer vision resources necessary for the verification process have to be accessible and usable for the aforementioned logic. That means, that the resources have to be integrated into the server.

This chapter descrives how the information flows between the two layers, and the tasks performed from the server to attend the client's requests.

3.2.1. Technologies

Regarding the service layer, I have decided to use **PHP** as the programming language for the server. The decision stems from both this being the most extended language for server programming and my desire to improve my skills on it, as stated in chapter 1.1. Regarding the client layer, I have decided to use **HTML5** for the design, **CSS** for the style and **JavaScript** for the behavior. As the design on this layer has a lower priority, I have prioritized development speed, choosing the languages and pseudo-languages in which I have more experience with.

3.2.2. Implementation

The client layer manages the view seen by the user and translates the inputs into requests to the service layer. Between client and service layers, two message exchanges have been identified in the context of a face based identity verification:

1. At each verification attempt, the server will be comparing an image sent by the client with a reference image. Then, before attempting any verification a reference has to be set. Before starting with the verification, the client has to send reference image to the server.
2. Once a reference has been defined, the client sends images for validation. The server searches for faces in these images, and compares them with the reference. If a match happens, the server decides to validate the registration.

Also in this context, there are two types of reference candidates: document photographies and natural images of the user's face. Experimental results show no clear correlation between the type of reference used and the accuracy in the verification, but it is to be noticed that processing a document photography is notoriously slower than processing a natural image. As the verification process might need multiple images per reference in order to produce a positive validation, I have decided to use the document photography as the reference.

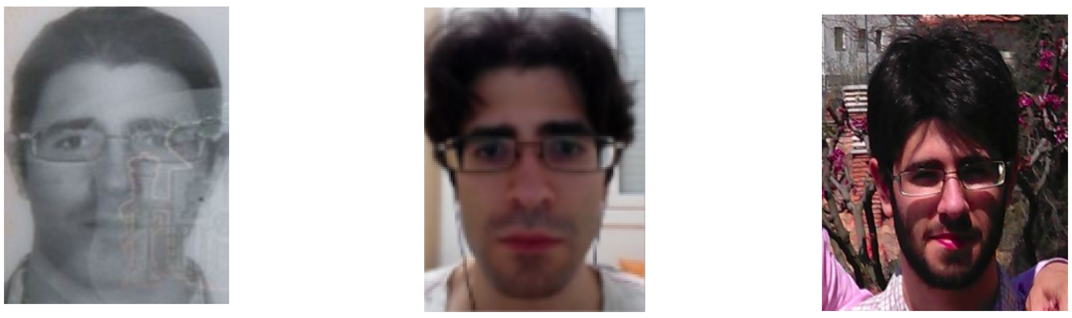
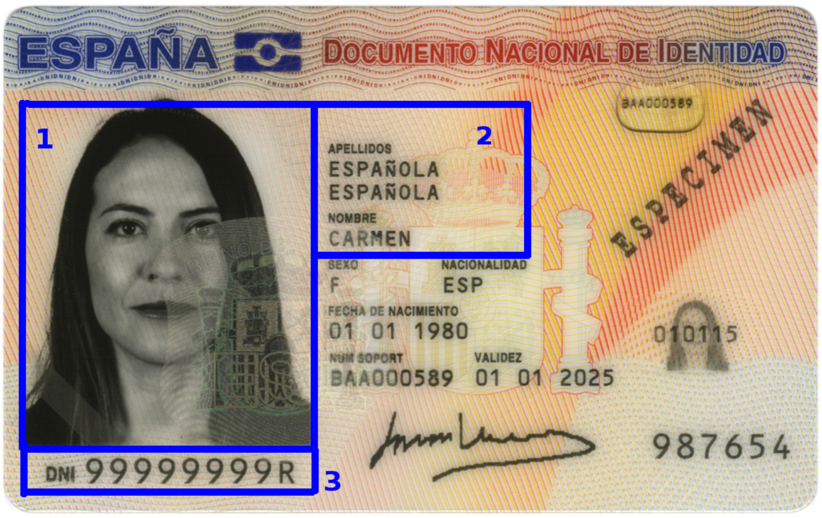


Figure 7. Three examples of reference candidates. From left to right: document photography, natural image and natural image taken under uncontrolled conditions of illumination.

Since it is expected for the user to have easy access to a camera device, the application has been designed to request the user to place the document in front the camera, in a position that allows the system to search the data needed for the validation in known areas of the vision field. This helps to diminish the cost of data extraction. The responsibility of knowing the position and size of the areas belongs to the client layer, as those have to be transparent to the user.


ESPAÑA DOCUMENTO NACIONAL DE IDENTIDAD

BAA000589

APELLIDOS
ESPAÑOLA
ESPAÑOLA
NOMBRE
CARMEN
SEXO F
NACIONALIDAD ESP
FECHA DE NACIMIENTO 01 01 1980
NUM SOPORT BAA000589
VALIDEZ 01 01 2025
DNI 99999999R

ESPECIMEN
010115
987654

Figure 8. Regions of interest in an example DNI 3.0 document. Region 1 contains the document photography, region 2 contains the name of the individual and the number is in region 3.

The dimensions of the document and the regions of interest are communicated to the server when requesting for a reference setting, alongside the image that is to be set as reference. That way, the server is able to crop the image in three sub-images -the photography, the number region and the names region- and process each of them individually. Attempting to read text from an image is an expecting process. Knowing that, the server imposes a series of restrictions so that the face to be found in the document has to be centered inside the photography region, and that the eyes have to be aligned on the x-axis, meaning that the document is not being rotated. The server will reject the request if the restrictions are not satisfied.

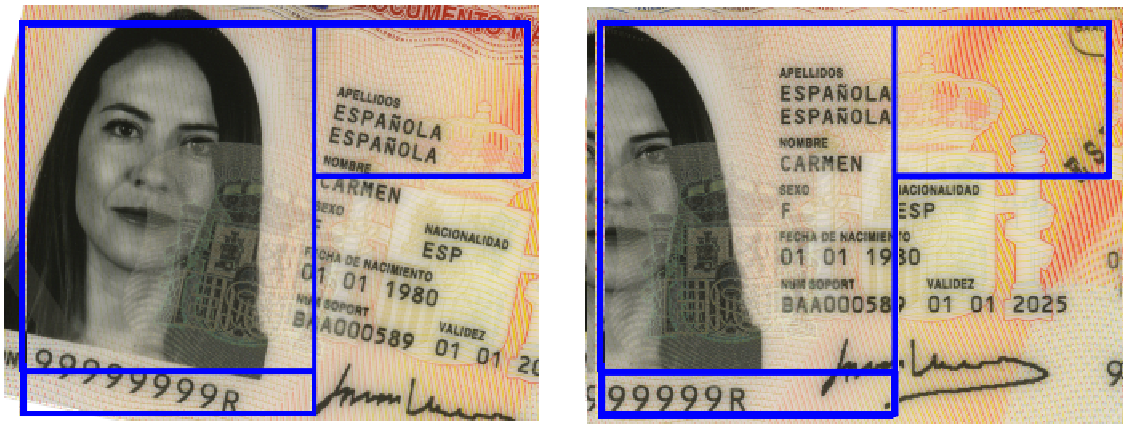


Figure 9. Examples of document positionings failing the restrictions imposed by the server. On the left, the server can infer that the document is rotated because the eyes are not aligned. On the right, displacement is detectable as the face is not centered in the photography region.

The request will also be rejected if the number of the document, although read correctly, is not a valid DNI number. For a number to be valid, it has to contain 8 digits and a letter matching those digits[2]. When requesting a reference setting, the client keeps messaging the server until receiving an answer of valid reference, though the process can also be aborted by the user. The client updates its status when receiving an answer from the user. The view is updated periodically during the process duration according to the current status. Once the server has received a valid reference image, the Face object generated from the document photography is stored in the session. Both the number and the name are answered back to the client, which uses the information to update the view. The following lines define the format of both the reference setting request and the expected answer.

REF	IMG (base64)	PX, PY, PW, PH	DX, DY, DW, DH	NX, NY, NW, NH	SX, SY, SW, SH

TRUE	DNI (00000000x)	Name1	Name2	Name3

FALSE	MSG

Figure 10. First: format of a request message for setting an image as reference. Second: response message when the request has been processed successfuly. Third: response to a failed reference request.

The first line defines the format of the request. The code **REF** is used by the client-server layers as an identificator of the reference setting request. **IMG** holds the image as a Base64 encoded string. The last three parameters contain the origin (X,Y), the width W and the height H of the three rectangles defined by the regions of interest of the document: the photography (P), the DNI number (D) and the names (N).

The second line defines the format of an answer to a valid reference image. **TRUE** is a code indicating that the image sent in the request has been successfully set as reference. The **DNI** field contains the DNI number plus letter read from the sent image. The format supports up to one name (**Name1**) and two surnames (**Name2**, **Name3**) that will be returned to the client.

The last line defines the format of an answer to an invalid reference image. **FALSE** is a code indicating that no reference could be set. Finally, **MSG** is an optional field used by the server to communicate what went wrong in the request.

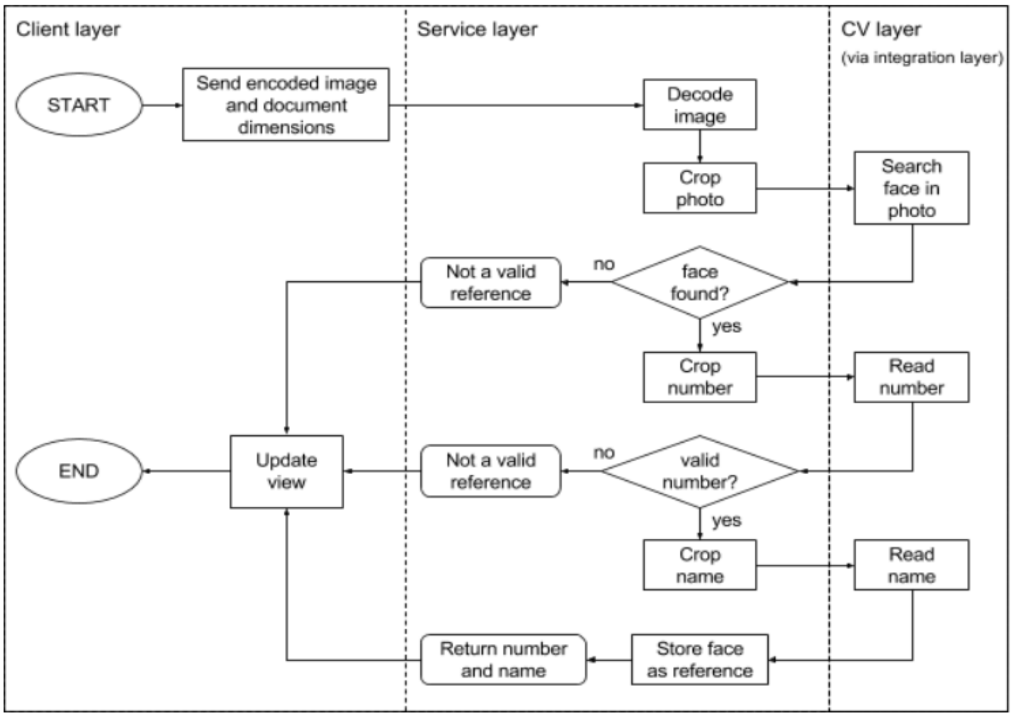


Figure 11. Communication between the client and server layers during a request for reference setting.

A reference has to be set successfully before attempting a face validation. The server will reject validation requests if there is no reference Face stored in the session. Equal to the reference setting, the client keeps messaging the server when requesting a validation until the validation succeeds or the user aborts the process.

VAL	IMG (base64)

TRUE

FALSE	MSG

Figure 12. First: message format for a validation request. Second: format of the answer to a successful validation. Third: format of the answer to a failed validation.

The format of both the validation request and the expected answer is to be found at figure 12. The first line defines the format of the request. The code **VAL** is used by the client-server layers as an identificator of the validation request. Again, **IMG** holds the image as a Base64 encoded string.

The second line defines the format of the answer to a successful validation. The sole content of the message is then the code **TRUE**.

The last line defines the format of an answer to a failed validation. **FALSE** indicates that the validation failed. Again, **MSG** is an optional field containing information about what went wrong with the validation.

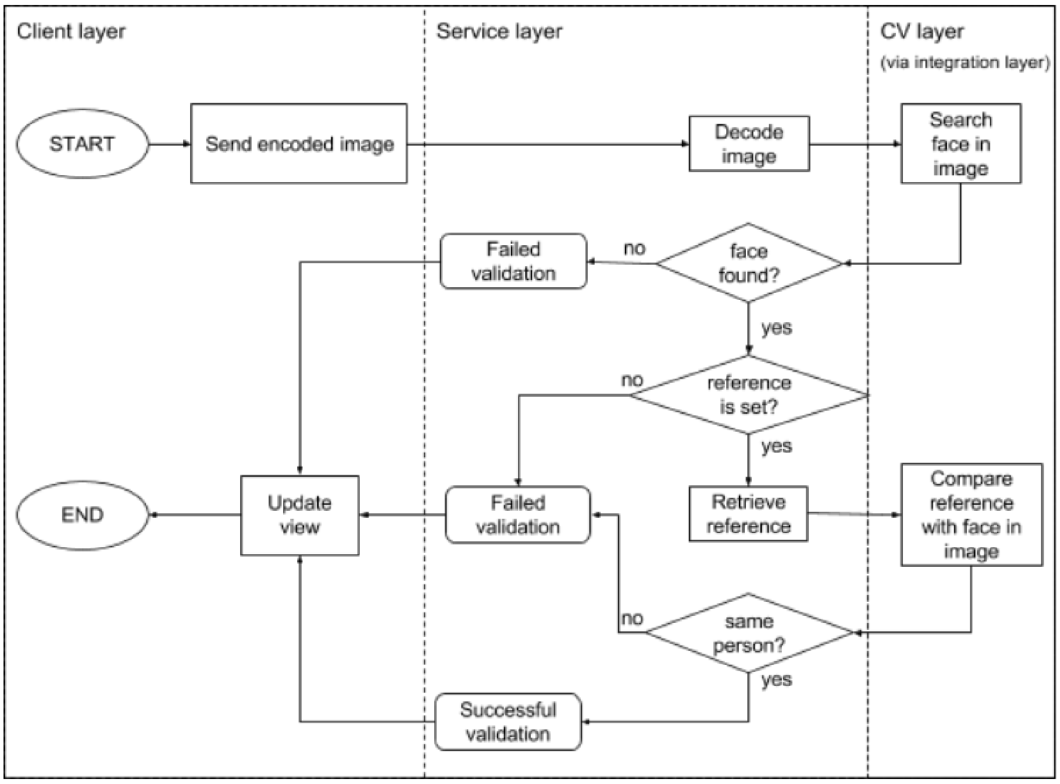


Figure 13. Communication between the client and server layers during a request for reference validation.

3.4. Integration layer

Computer vision algorithms needed to perform the verification tasks cannot be implemented in PHP source bundle. The language has to be extended to allow the integration the computer vision layer.

3.4.1. Technologies

Writing PHP extensions is a challenging task for someone without previous experience, as it is my case. Luckily, some libraries exist that allow developers to minimise the cost of writing extensions, or directly circumnavigate it:

- **PHP-CPP** is a C++ library. It offers a collection of well documented and easy-to-use classes that can be used and extended to build native extensions for PHP.
- **PiP** or Python in PHP is a PHP extension that, in short, allows the Python interpreter to be embedded inside of PHP. This allows native Python objects to be instantiated and manipulated from within PHP.

Op.	*Library*	*Language*	*Pros*	*Cons*
1	**PHP-CPP**	C++	Support for PHP5 and PHP7, backed by an active community.	Version for PHP7 is not as well documented as PHP5.
2	**PiP**	Python	As a high level language, Python allows for easy prototyping of the code.	No longer supported, incompatible with PHP5.

Table 2. Description of the technologies available to alleviate the cost of writing PHP extensions.

Among the options displayed in table , PHP-CPP is a clear winner. Though the documentation on the PHP7 version of the library is scarce, the PHP5 version is both well documented and actively supported by a community of developers who can provide help. On the other hand, PiP is a discontinued project. Even though it has some documentation, it lacks the community PHP-CPP has, and there are no expectations that it will support more recent versions of PHP in the near future.

Taking this into account, I have decided to use **PHP-CPP** to integrate the computer vision layer. Specifically, I am going to use PHP-CPP for PHP5 due to it being the best documented version. This also means that the server layer will be implemented on **PHP5**.

3.4.2. Implementation

Since PHP-CPP has been decided as the technology to use in the integration, the components in the computer vision layer are expected to implemented in C++. Then, the types used in the components will not necessarily match the types available in PHP. The architecture interposes an additional layer between the service layer (the PHP server) and the computer vision layer. By using the PHP-CPP library, the layer defines a series of C++ classes that wrap the components found in the computer vision layer so that its resources can be used in PHP.

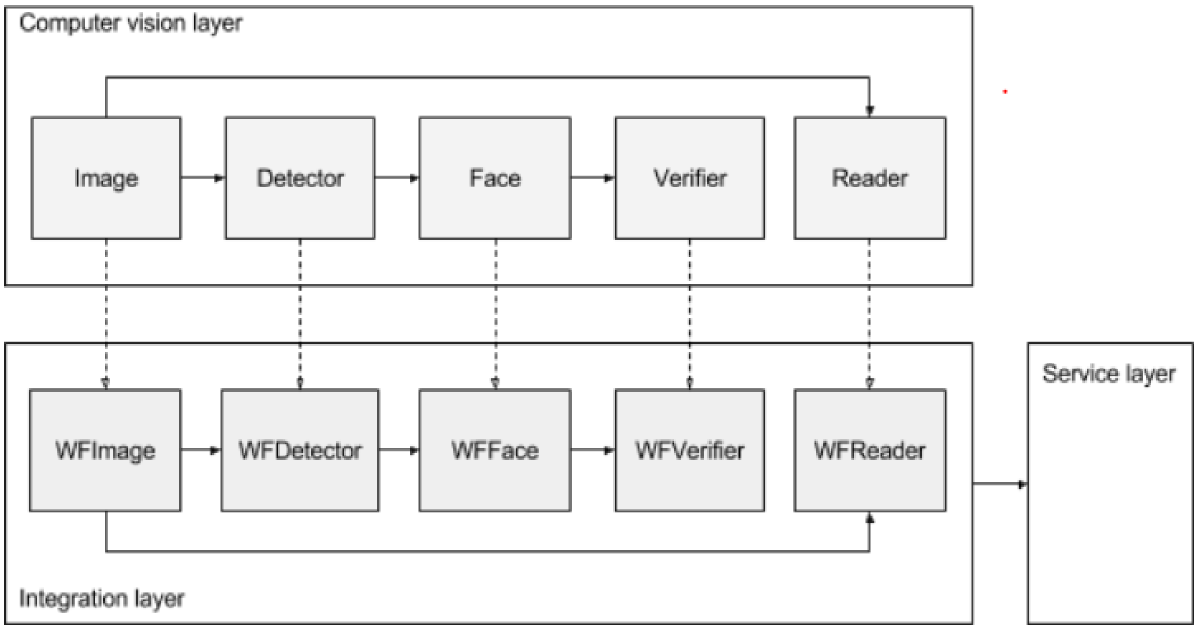


Figure 14. Structure of both the computer vision layer and the integration layer. The components in the integration layer match those found in the computer vision layer.

3.3. Face detection

The problem of identity validation through artificial vision implies that the system has to be able to find human faces in digital images. From now on, this problem will be referred as face detection. The design of the application assumes that the user has easy access to a camera device or webcam. Then, the system can expect all the face images to be frontal face images taken in relatively controlled conditions, alleviating the cost of face detection.

3.3.1. Technologies

There are multiple libraries which can be useful when implementing the face detector. As I have previously decided the language with which extend PHP, the libraries studied limit to those available for that language. The table below summarizes what these libraries have to offer to this project.

OpenCV is a library available for both Python and C++ dedicated to image processing. It provides a Haar Feature-based cascade classifier for object detection[3], proposed by Paul Viola and improved by Rainer Lienhart. The classifier consists on a sequence of one-class classification stages. When applied to a region of interest, the stages are applied subsequently until the candidate is rejected or all the stages are passed. The classification stages are complex themselves and they are built out of basic classifiers consisting of decision trees with at least two leaves. Haar-like features are the input of the basic classifiers.

The library includes multiple training data allowing for the implementation of a variety of of object detectors, including face detection and eye detection.

Another option can be found in **dlib**, a C++ library dedicated to machine learning[4]. Includes a face detector that uses linear classification to discriminate objects belonging to the class in search and other objects. The input images are parameterized by histograms of oriented gradients (HOG). It also includes an implementation of a face landmark predictor[5] proposed by Vahid Kazemi and Josephine Sullivan. Given an input image, the detector uses an ensemble of regression trees to predict where these landmarks are to be found based on the location of these same features in the set of training images.

The library includes the needed training data for both the face detector and the face landmark predictor. Though both libraries provide for the necessary tools to implement the dectector, each of them has is advantages and disadvantages.

Op.	*Library*	*Technologies*	*Pros*	*Cons*
1	**OpenCV**	Cascade Classifier, Haar features	Fast at finding both faces and eyes.	When searching for faces, the classifier is susceptible to false detections (false positives). When searching for eyes, the classifier is susceptible to false omissions (false negatives).
2	**dlib**	Linear classification, Regression trees, HOG	Fast at finding facial landmarks on face images, more accurate.	Slow at finding faces. Lacks documentation when compared with OpenCV.

Table 3. For both OpenCV and dlib, the table contains the technologies offered by each library, its advantages over the other (expressed as "pros") and its disadvantages (expressed as "cons").

When it comes to finding facial features the solution provided by dlib provides superior performances, as it is more precise than the one provided by OpenCV without losing speed. For this reason, I have decided to implement the facial features detection with dlib.

As for facial detection, the difference between options is not so clear. On one hand, dlib is way more precise than OpenCV, but this comes at a cost as finding faces takes up to 30 times more with the dlib implementation. To solve this problem, I propose a design that combines both **OpenCV** and **dlib** to obtain a face detector capable to find faces with the precision of the dlib method at a speed near to that found in OpenCV.

3.3.3. Detector component

The detector component is responsible for finding human faces in digital images. Given an input Image object, the detector extracts one face from it and returns it as a Face object. The face detection process is performed in three subsequent steps:

1. The detector takes an input image and searches in it those regions candidate to be faces. The candidate search is implemented with a Haar features based cascade classifier, provided by the library OpenCV. As the classifier is prone to false positives, it is expected that some candidates do not contain any face.
2. Then the detector searches on each of the candidates for facial features, using a face landmark predictor from the library dlib. The candidate is discarded if it does not contain at least a left eye and a right eye. Due to the accuracy of the predictor, it is expected for all the remaining candidates to contain a face.
3. Finally, the detector rotates the image so that the face ends up straightened taking the eyes coordinates as reference.

Since the classifier makes use of Haar feautures, input images have to be grayscaled. OpenCV proposes incrementing the global contrast of input images by histogram equalization, as a mean to improve the classifier's performace.

Before attempting any face detection, the detector has to be loaded. Loading the detector means to provide it with necessary training data for both the cascade classifier and the face landmark predictor.

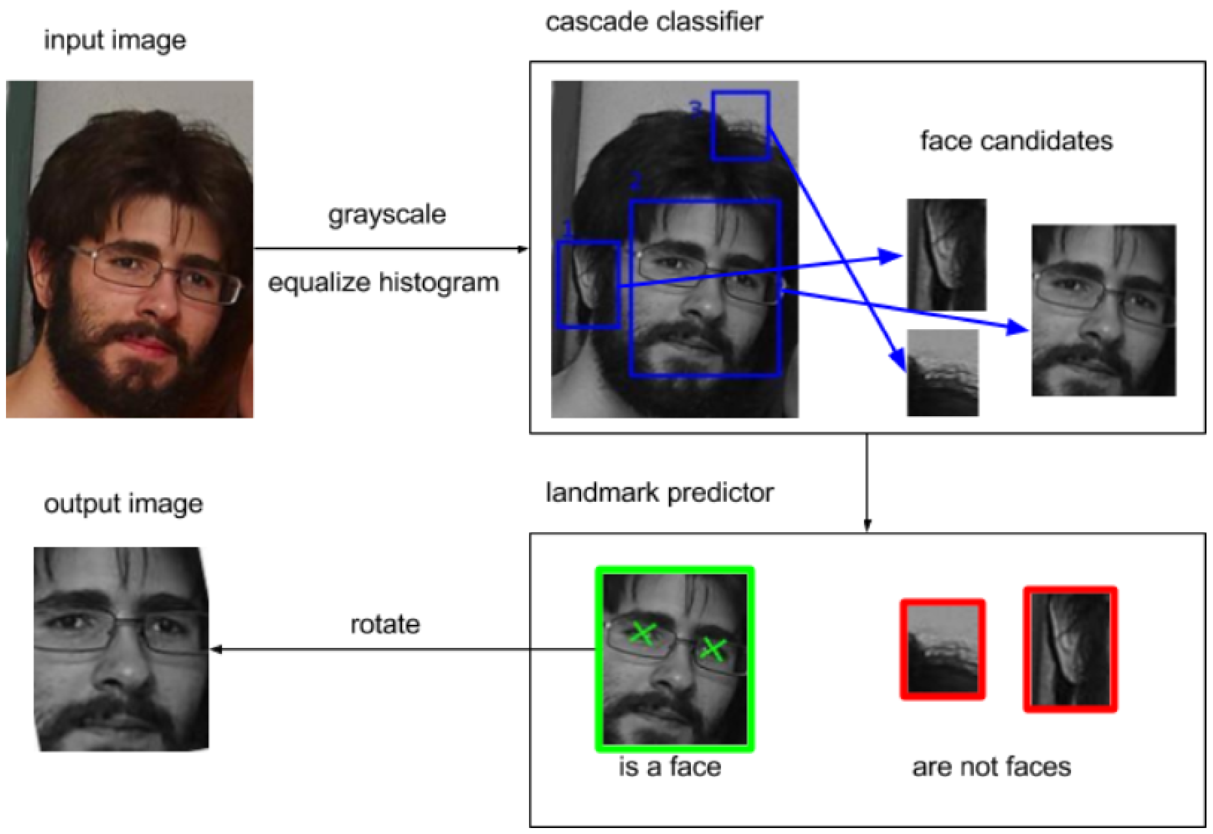


Figure 15. Representation of the detection process performed by the detector component. Note that the false positives in the cascade classifier have been represented to illustrate the behavior of the component, and do not necessarily match real cases.

The component depends on the server layer to provide the training data sets. These are passed into the component as directory paths to the resources. The component itself takes responsibility on the training of both the classifier and the predictor. Once the classifier and the detector have been trained, the state persists for the whole component lifecycle, and therefore there is no need to reload it between detections.

By default, the server creates and loads one detector by session, being the first detector created stored in the user's session. The server can be configured to generate one detector per request. This should not hinder the performance of the application because the cost of loading the detector is minimal when compared to the cost of validating a reference or comparing two faces.

3.4. Face verifier

In the context of a user registry, we have to assume that the user is unknown to the system. That means that all the information needed to perform the verification has to be gathered during the short time it takes for the user to complete the registration. At best, we can only expect to gather some face images through a camera device plus the image found in the identification document provided by the user. Then, the system has to compare an homogeneous set of images with a photography that might have been taken years ago, making the process of face verification a complex one. Two different ways have been identified to approach the problem of face verification.

3.4.1. Technologies

The problem of face validation might be seen as a particular case of face recognition where we are attempting to classificate a given set of face images between those which belong to the user (positive images) and those which don’t (negative images).

Then, the solution to the problem goes through the usage of a one-class classifier, which has to be trained with a set of both positive and negative examples. As the user is unknown, the application can only have one positive example in the reference face image, hindering the performance of the classifier.

OpenCV supports **machine learning** solutions to face recognition in the FaceRecognizer[6] class. This class can be configured to use either PCA or LBPH as a mean to compare the face images:

1. Principal Components Analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. In OpenCV, the procedure can be configured to use either Eigenfaces or Fisherfaces as a mean to

parametrize the images to be compared. Fisherfaces are expected to provide better results when there are changing illumination conditions or facial expressions are involved, making them more appropriate for this application[7].

2. Unlike PCA, where the images are analyzed as a whole set, Local Binary Pattern Histograms (LBPH) analyzes each image independently from the other. Following this method, each image to be compared is being described as an LBP vector.

If trained with the reference image as a positive example and a set of random face images as negative examples, FaceRecognizer can be employed as a face verificator.

Alternatively, face verification might be approached as an independent problem to face recognition. As the user is unknown to the application, we can depend on unattended machine learning to decide if the resemblance between two given face images is enough for them to belong to a same individual.

The **dlib** library provides **deep learning** solutions, allowing for an unattended machine learning approach. The main problem with deep learning, though, is the training cost. The amount of images needed to train a deep neural network is usually several magnitude orders greater than the amount needed to train a classifier. Luckily, dlib provides a pre-trained model of residual network.
A residual network, or **ResNet**, is a deep learning framework developed at Microsoft Research as a solution to the degradation problem on deep networks[8]. This problem describes how the network's accuracy gets saturated and then degrades rapidly at increasing depths. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error. The model provided by dlib has a 29-layer depth and has been trained on a dataset of about 3 million faces and 7485 different identities.

The following table shows the advantages and disadvantages of the aforementioned technologies.

Op.	Library	Technology	Pros	Cons
1	OpenCV	PCA	It is the fastest technology among the investigated.	It is the least accurate. In the context of the application, only one positive sample can be provided for training. The width and height of the input images must match those of the training images[9].
2	OpenCV	LBPH	Faster than ResNet, more accurate than PCA.	Slower than PCA. In the context of the application, only one positive sample can be provided for training. The width and height of the input images must match those of the training images.
3	dlib	ResNet	It is the most accurate technology among the investigated. Insensitive to scaling.	It is the slowest technology.

Table 4. For both OpenCV and dlib, the table contains the technologies offered by each library, its advantages over the other (expressed as "pros") and its disadvantages (expressed as "cons").

Although fast, the solutions provided by OpenCV performed poorly in the testings. They also require for the candidates for validation to be rescaled to match the size of the reference, incurring in an information loss that could hinder the accuracy of the verifier. Concluding that the increment of the accuracy outweighs the increase on the time it takes for the verifier to reach a solution, I have decided to use the ResNet implementation provided by **dlib**.

As stated before, the main problem one faces when employing deep learning solutions is the training cost. As I could have access to the pre-trained model provided by dlib, this has not been considered as an impediment to the election of dlib as the library for face verification.

3.4.2. Verifier component

This component is responsible for deciding whether the face images contained in two given Face objects belong to the same person or not. Compared to the face detector, the verifier is more straightforward. The verifier uses a residual deep neural network to compare the images contained in two given Face objects. The network is being implemented with dlib library, and for each comparison made it returns a distance of dissimilarity between the images. To make the decision, the distance is compared with a threshold value. That way, two face images are considered to belong to a same person if the distance is inferior to the threshold. The component uses a default threshold of 0.6.

Although the network is insensitive to the size of the input faces, it expects them to be 3-channeled images[10]. In chapter 3.3 it is stated that the face detector grayscales the images when performing the detection, and hence we can expect the Face objects passed to the verifier to also contain grayscaled images. Then, the verifier component has to rescale the images back to 3 color channels before attempting the comparison. The verifier does that by simply copying the intensity values of the pixels in the input images in all three RGB channels.

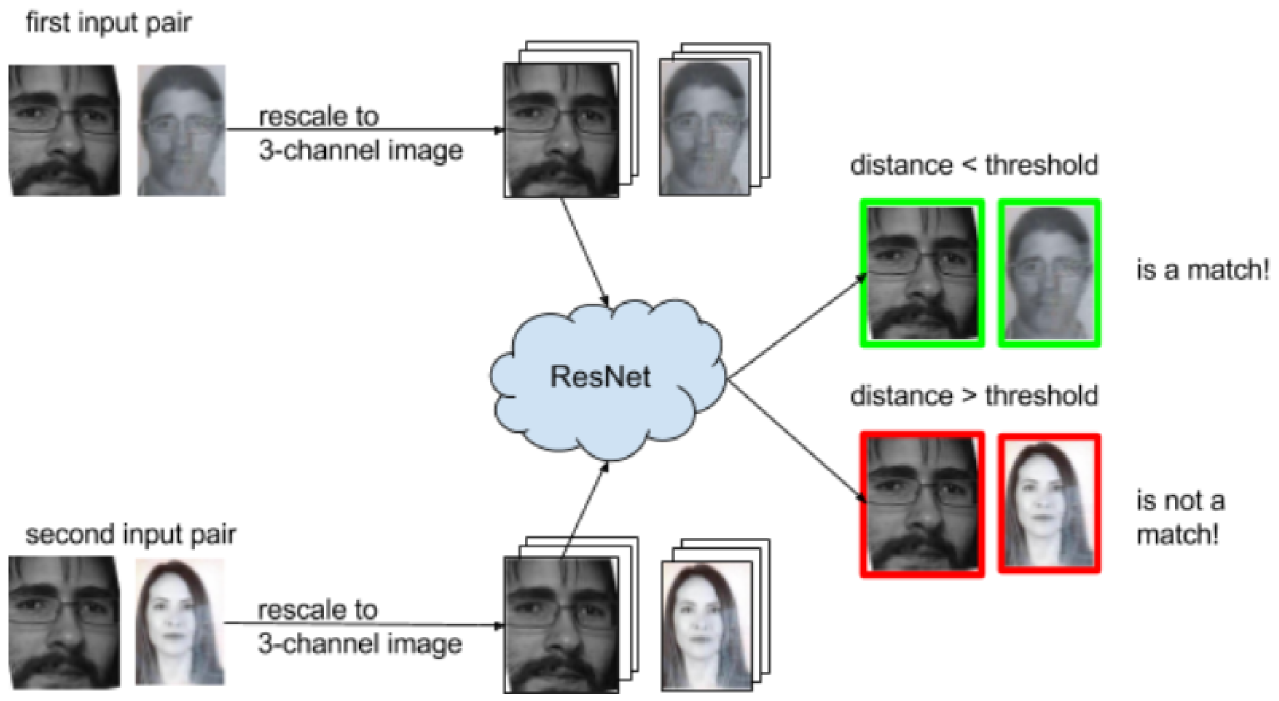


Figure 16. Verification process performed by the verifier component over two pairs of input faces.

3.5. Optical character recognition

The problem of reading text from document images has been approached by using OCR techniques. Optical character recognition is the process of extracting textual information from digital images, allowing the application to be able to validate a DNI document through its number and obtain the name and surnames of its owner.

3.5.1. Technologies

The state of the art of open-source OCR engines is currently dominated by **Tesseract**. The engine began as a PhD research project in HP Labs and first developed between 1984 and 1994. The engine was sent to the 1995 Annual Test of OCR accuracy, where it proved its worth against the commercial engines of the time. The project has been continued since then, with its latest stable version released in June 2017, licensed under the Apache License Version 2.0. Processing follows a traditional step by-step pipeline[11].

1. The first step is a connected component analysis in which outlines of the components are stored. The outlines are gathered together and nested into blobs.
2. Blobs are organized into text lines, each line then analyzed for fixed pitch of proportional text. Text lines are splitted into words differently according to the type of character spacing: fixed pitch text is spitted by character cells, whereas proportional text is splitted into words by using definite spaces and fuzzy spaces.
3. Then, recognition proceeds by attempting to recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. This means that the accuracy of the recognition improves as long as the engine keeps recognizing words. Since the adaptive classifier might have learned something useful too late to make a contribution on the first words, the engine does a second attempt at recognizing the words with which it failed the first time.

4. A final phase resolves fuzzy spaces, and checks alternative hypotheses for the x-height to locate small-cap text.

The performance of the engine is very susceptible to the quality and conditions of the image where the recognition is attempted. Hence, it is recommended that the images undergo an enhancement process or preprocessing before using the engine. Experimental results show that Tesseract is a solid, viable technology when attempting to read from preprocessed images of DNI 3.0. For this reason, I have decided to use **Tesseract** for the implementation of the reader component.

3.5.2. Reader component

The reader component is responsible for extracting textual information from digital images, given as Image objects. The component interprets any text found in the image and returns a string with the information that has been read successfully. In the context of the application, the component behaves as a “dummy” reader. That means that the component does not interpret the information found in the images, as that is the responsibility of the service layer. From the reader component perspective, it doesn't matter if the text to be read is the id number, the name or any other field. Before attempting the reader, the component has to preprocess the input images.

As stated in chapter 3.2, once a document image is received from the client the server layer splits it into multiple, smaller sub-images. The images correspond to the places in the document where the information relevant to the verification process is to be found. Therefore, the images the engine has to deal with are rather simple, as they are expected to include nothing more than the text.

Tesseract engine assumes that the input is a binary image[12]. Then the preprocessing consists in the binarization of the image. The threshold T is computed as the product of the mean of the intensity in the image and a parameter W, where W is an optimization parameter that modifies the threshold value to allow for more or less permissive binarizations. Experimental results by using DNI 3.0 documents set the optimal value of the parameter as W=0.7, but it can be changed through the configuration of the application.

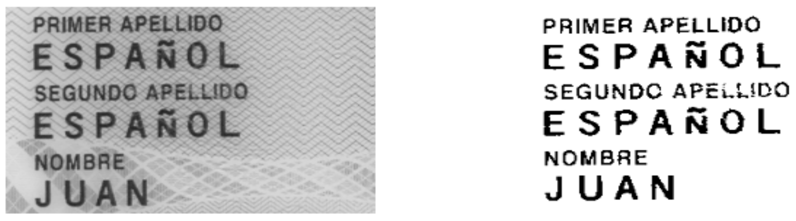


Figure 17. To the left, image of the name and surname fields in a DNI 3.0 example. To the right, the same image after undergoing the binarization with W=0.7.

3.6. Tools used

The tools used can be classified in software tools, integrated development environments and testing tools, and hardware tools. I decided to use the tools which I had some previous, positive experience with. The tables below list the tools used, including the specifications of the machine used to test the performance of the application.

IDEs	Qt Creator Microsoft Visual Code
Testing tools	PHP integrated server Mozilla Firefox 55.0.2
Operative system	Ubuntu 14.04 LTS (x64)
Memory	7,6 GiB
Processor	Intel Core i5-6200U CPU @ 2.30GHz x 4
Graphics	Intel HD Graphics 520 (Skylake GT2)

Table 5. Table containing the tools used during the implementation and testing of this project, as long as the specifications of the machine used.

4. Results

4.1. Detector performance

OpenCV documentation does not provide measures on the cascade classifier performance. Therefore, I have decided to test the detector component to estimate its accuracy and speed.

4.1.1. Metrics used

The detector component behaves as a binary classifier which discriminates sample images between those belonging to a Face class and those which do not belong to it. Taking this into account, I have decided to use **confusion matrix** to display the results of the test on the detector. In statistical analysis, the **F_1 score** has been commonly used as a measure of the accuracy of binary classifiers[13] where a value of 1 represents perfect accuracy and 0 represents the worst. The score computed taking into account both precision and recall.

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

Formulas 1,2,3. Formulas for the precision, recall and the F1 score of a binary classifier where TP is the amount of true positives, TN is the amount of true negatives, FN is the amount of false negatives and FP is the amount of false positives.

Finally, in order to evaluate the speed of the component, I have measured the average **CPU time** it takes for the detector to attempt a classification. As all the tests have been executed in only one computer, the time measured is to be taken as an estimation of the components velocity rather than a reliable metric.

4.1.2. Description of the test

The detector was tested over a dataset of 1541 grayscale images, 1521 of them containing one face and 20 containing no face at all.

Among the faces in the dataset, 23 distinct identities are to be found. Both the identity of the individual and the position of the eyes are known on each of the images. The face images in the dataset have been obtained from the BioID Face Database[14], while the other 20 have been obtained with Google Images.

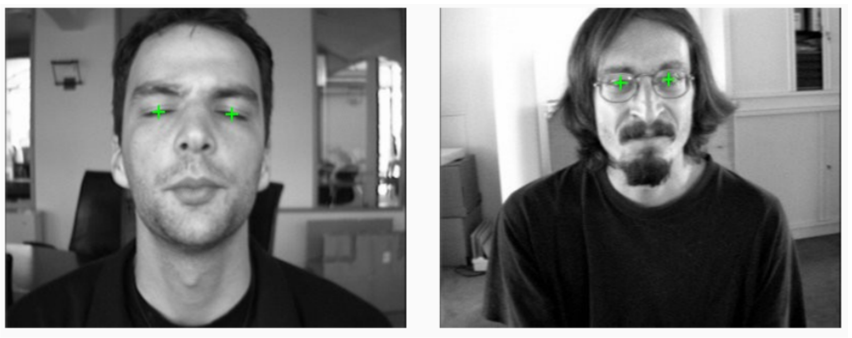


Figure 18. Two sample face images from the BioID Face Database.

The test has been designed so that the detector attempts to find one face with two eyes in each of the images in the database. The detector will predict an image to belong to the Face class if a face with two eyes are found in it.

4.1.3. Detection results

The results obtained for the detector component performance can be found in the table below. The chart contains the confusion matrix for the aforementioned database, the values of precision and recall and the F_1 score as the metric chosen to measure the accuracy. The chart contains also the average CPU time taken for the detector to make a prediction and the distance between the predicted eyes position and their actual position. As a reference, all the images in the dataset have a size of 384x286 pixel.

		Actual class				
		Face	No-Face	Precision	Recall	**F1 score**
Predicted class	Face	1433	0	1	0,94	**0,97**
	No-Face	88	20	L distance	R distance	**CPU**
				1 ± 1 px	11 ± 3 px	**50 ± 5 ms**

Table 6. Results for the test on both accuracy and speed for the detector component. L distance and R distance stand respectively for average distance to left eye and average distance to right eye. The average CPU time is labeled as CPU.

4.2. Verifier performance

According to the dlib documentation, the ResNet model provided and used in the implementation of the verifier component reaches an accuracy of **99.38%**[15] when benchmarked on Labeled Faces in the Wild, a standard database designed for studying the problem of unconstrained face recognition which contains more than 13000 face images collected from the web. As the value provided might be too optimistic, I have decided to execute my own testings on the verifier.

4.2.1. Metrics used

When defining the identity as the class of a face image, the verifier component behaves as a binary classifier which discriminates between faces belonging to the same class as the user in validation and those which do not. Same as with the detector component, I have decided to use use a **confusion matrix** to display the results of the test on the verifier.

For the verifier, I also want to estimate the probability for a user to register by using a document not belonging to him or her. Then, I have decided to use the **fall-out** or false positive rate as a secondary metric for the accuracy of the verifier, the primary metric being the **F_1 score** as in the detector component. The fall-out can be computed from the amount of false positives and true negatives. In this context, it might be understood as the probability that a user not providing a valid document can register, making for a suitable metric for the accuracy of a security system[16]. As for the F_1 score, its description can be found at chapter 4.1.1.

$$fall\ out = \frac{FP}{FP + TN}$$

Formulas 5. Formulas for the fall-out of a binary classifier where TN is the amount of true negatives and FP is the amount of false positives.

Finally, the speed is estimated from the average **CPU time** it takes for the verifier to attempt a classification.

4.2.2. Description of the test

The database used contains 34 facial images from 4 different identities, each of them containing one face. For each of the identities, the database presents variance on the age, hairstyle and complements. The identity of the individual is known at each image. The database contains 4 DNI photographies from 3 different identities.

Figure 19. Five sample face images from the database used for testing the verifier component.

For each of the identities on the dataset, one face image has been randomly selected and compared with the other images in the dataset. That means that the results on the dataset have been cross-validated four times. The test has been performed using the default verifier's threshold of **0.6**.

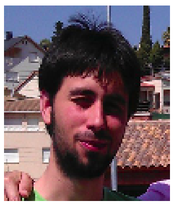
4.2.3. Verification results

The results obtained for the verifier component performance can be found in the chart below. The chart contains the confusion matrix for the aforementioned database, the values of precision and recall and both the F_1 score and the fall-out as the metrics chosen to measure the accuracy.

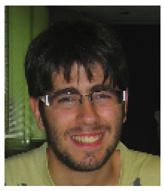
		Actual class				
		User	No-User	Precision	Recall	**F1 score**
Predicted class	User	261	60	0,81	0,99	**0,89**
	No-User	3	798		**CPU**	**Fall-out**
					14 ± 2 s	**0,07**

Table 7. Results for the test on both accuracy and speed for the verifier component and threshold equal to 0.6. The average CPU time is labeled as CPU.

Although the results for the set as a whole are positive, drops on accuracy have been observed when the reference image features strong facial expressions or the illumination is low.



		Actual class			
		User	No-User	**F1 score**	**Fall-out**
Predicted class	User	8	7	**0,69**	**0,28**
	No-User	0	18		



		Actual class			
		User	No-User	**F1 score**	**Fall-out**
Predicted class	User	9	7	**0,72**	**0,29**
	No-User	0	17		

Table 8. Results on accuracy for the references that performed the worst. On the left, the images used as references, the first of them featuring poor light conditions and the second featuring facial expressions.

4.3. OCR performance

Tesseract documentation contains values on the accuracy of the OCR engine. These values are to be accepted as a reference for the component's performance, and therefore, I have decided to not test the reader component on its accuracy at recognizing text from digital images. Regarding the speed, I have decided to do an estimation of the time it takes for the OCR to perform the reading.

4.3.1. Metrics used

Tesseract documentation measures the accuracy of the engine by **percentage of errors** on a series of recognition attempts. The same as for the detector and verifier components, I have decided to use the average **CPU time** it takes for the component to do a reading as an estimation of the speed.

4.3.2. Test description

As described in Tesseract's documentation, the test on the accuracy consists on the reading of four different texts or sets. For each of the sets, the engine attempts to read both words and single characters. The amount of failed attempts are counted in order to compute the percentage of errors. The test on speed has been designed so that the component attempts to read text from two sets: a first set of 40 images containing text and a second set of 20 images containing no text. The accuracy of the reading is irrelevant to the test, whose objective is to measure the CPU time taken for the component to do the reading. All the images on both sets are equally sized.

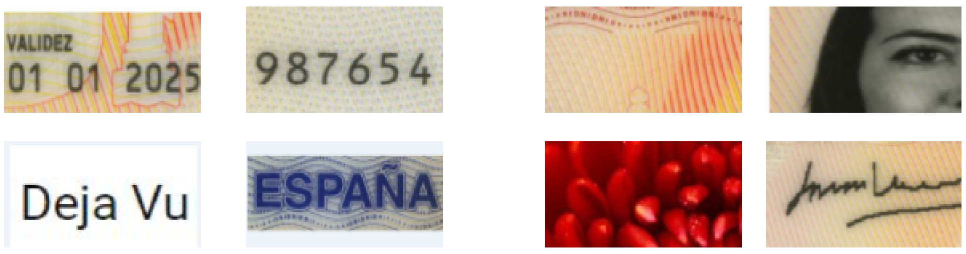


Figure 20. On the left, some samples taken from the text set for the test on the reader component. On the right, some samples taken from the no-text set.

4.3.3. OCR results

The following table shows the results on accuracy provided by the documentation[17].

	Character		*Word*	
Set	Errs	%Err	Errs	%Err
bus	6449	2,02	1295	4,28
doe	29921	2,04	6791	4,95
mag	14814	2,22	3133	4,64
news	7935	1,61	1284	2,62
total	59119		12503	

Table 9. Results on the accuracy of Tesseract OCR engine, classified by either character or word and by text sample, labeled as set. The table features errors per text and the percentage overthe attemps.

As for the speed, results on the aforementioned sets show an important drop on performance when attempting to read from images containing no text:

- Average CPU time for text set: **3 ± 1 s**
- Average CPU time for no-text: **12 ± 4 s**

5. Conclusions

5.1. Compliance with the objectives

The proposed design has been successful at implementing a facial verification system in a web based client-server application that allows users to make a registration. The application designed relies on facial verification in order to validate a registration by comparing the user face with a photography to be found in a DNI 3.0 document. Additionally, the application extracts text information from the document, namely the DNI number and the owner’s name and surnames.

The verifier component has reached an accuracy score of F_1= 0.89, and therefore I conclude that it complies with the objective of implementing an accurate face based validation system. With a fall-out value of 0.07, the verifier component might still be a bit high for a security system. Reducing the threshold in the verifier component allows for the implementation of more impermeable applications, though it would also increase the average time taken for the registration process to conclude as the verifier would concur on more false negative predictions.

As the design assumes that the user has easy access a camera device, the decrease on accuracy observed for both facial expressions and poor illumination should not be something to be concerned for.

Since all the tests on the components speed have been done on the same machine, the results cannot be considered reliable enough for the final application and has to be considered only as an estimation. Then again, taking the results into account we can assume that it takes around 15 seconds for the application to attempt a validation. Empirically, it has been estimated that the face-based validation requires in average a maximum of two attempts. Hence the time required for registration is inferior to 30 seconds.

The performance of the reader component is limited by the necessity of the user to indicate the regions of the image where text is to be found, and can only be estimated. When accepting the values on the accuracy for Tesseract OCR engine, we found that the component is more prone to error when detecting whole words that when detecting series of single characters. I have observed the application to be good enough at reading the DNI number, but with the designed method for text extraction it has some difficulties at reading the name and surnames correctly.

When the document is placed correctly in front of the camera, the cost of attempting a reading of the DNI number has been estimated to be under 5, meaning that the whole process of data extraction and face-based verification is expected to take less than one minute. I conclude that the design complies with the objectives on terms of speed, but I would recommend for the implementation of an alternative, less user dependant method for text extraction.

Last but not least, the design has been structured to accommodate changes. The modular nature of computer vision layer allows for changes on the service layer workflow to be easily applied.

5.2. Future work

The application designed features high accuracy on both face detection and face validation, but there is still room for improvement on matters such as human interaction and security. On the final chapter of this memory, I propose some improvements on the design that could be approached on the future. The improvements proposed stem from features that had been studied at the investigation stage but could not be implemented on time.

5.2.1. Text detection with MSER

Maximally stable extremal regions (MSER) is a computer vision technique which finds correspondences between image elements in order to detect blobs. The technique can be repurposed for text detection when using several geometric properties featured by text to discriminate between text-blobs and non text-blobs, including the aspect ratio, eccentricity or solidity[18].

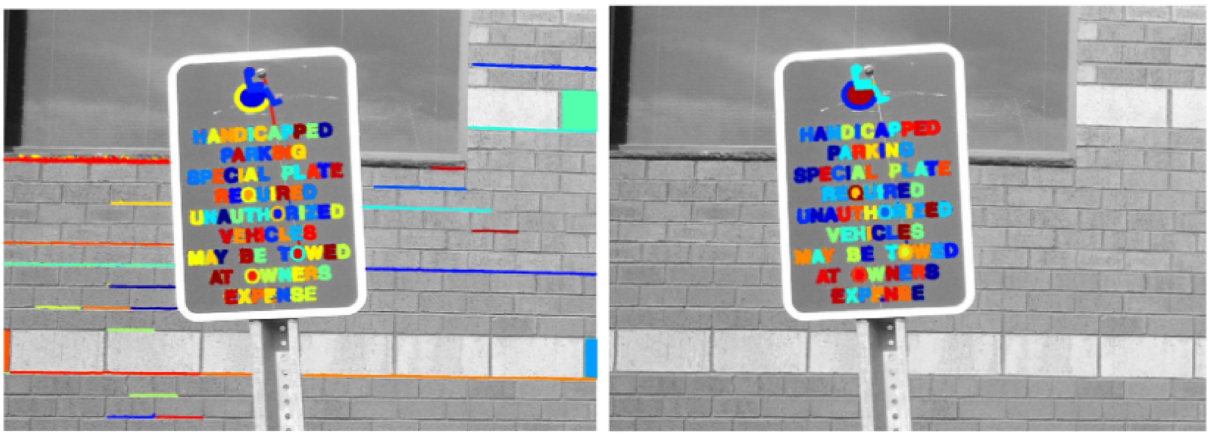


Figure 21. On the left, display of the blobs found with MSER on a sample image. On the right, display after removing the non-text blobs basen on geometric properties.

One of the main problems with current design is that text extraction is too dependant on the user, difficulting the readability of the document fields. Therefore, I propose the improvement of text detection through MSER to automate the task of text detection as a future work. This should improve the quality of data extraction and reduce the time needed to complete the validation process.

5.2.2. Liveness detection

Liveness detection is a recurring concept in the field of security systems based on biometrics. In the context of face based validation, we refer to it as the capability of a system to detect that the source of the face image comes from a real, physical person. As mentioned on chapter 1.2, liveness detection can be used to protect the system from spoofing attacks.

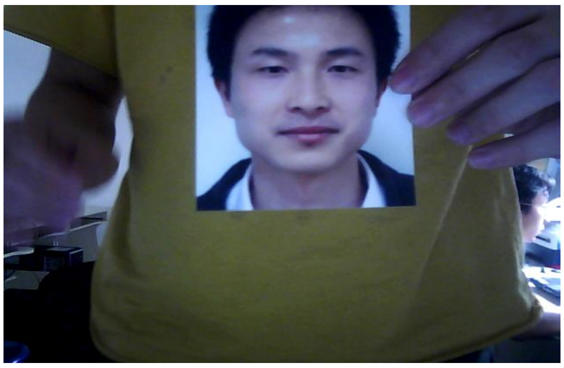


Figure 22. Example of a spoofing attack by using a printed picture.

Against spoofing attacks, a liveness detector can be understood as a classifier that tries to predict whether an input image is legit or not. Such classification is based on the differences featured by images coming from a real person and those coming from a 2D media.

Features	*Observations*	*Pros*	*Cons*
Texture	Light reflects differently on flat surfaces as how it does on volumetric surfaces[19].	Easy to implement. Fast	Dependant on image quality. The training dataset has to be diverse
Color frequency	The pattern of emittance of LCD screens present a shift to the blue when compared with the light reflected on a human face[20].	Easy to implement. Fast	Can only be used against images from LCD screens.
Motion	Eyes position tracked and/or blinking detected.	Independant to texture. Hard to spoof by static images.	Vulnerable to video. Slow

Table 10. The table shows the features which had been considered for the implementation of a liveness detector component during the investigation stage of the project. Advantages and disadvantages of the features are labeled as Pros and Cons.

The proposed design is at a disadvantage against similar products because it lacks the protection provided by liveness detection. For that reason, I propose the implementation of a liveness detection component as a future work to be done.

References

[1] W3Techs Web Technology Surveys. Usage of server-side programming languages for websites (Visited: September 2017).

[2] Government of Spain, Ministry of Interior. Cálculo del Dígito de Control del DNI (Visited: November 2017).

[3] Paul Viola, Michael Jones (2001). Rapid Object Detection using a Boosted Cascade of Simple Features.

[4] Davis E. King (2009). Dlib-ml: A Machine Learning Toolkit

[5] Vahid Kazemi, Josephine Sullivan (2014). One Millisecond Face Alignment with and Ensemble of Regression Trees.

[6] OpenCV 3.0.0-dev documentation. Face Recognition with OpenCV. https://docs.opencv.org/3.0-beta/modules/face/doc/facerec/facerec_tutorial.html

[7] Peter N. Belhumeur, Joao P. Hespanha, David J. Kriegman (1996). Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015). Deep Residual Learning for Image Recognition.

[9] OpenCV 3.0.0-dev Documentation. FaceRecognizer API.

[10] Jun-Cheng Chen, Vishal M. Patel, Rama Chellappa (2016). Unconstrained Face Verification using Deep CNN Features.

[11], **[12]** Ray Smith (2007). An Overview of the Tesseract OCR Engine.

[13] David Powers (2011). Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness & Correlation.

[14] The BioID Face Database (Last modified: 2010).

[15] Davis E. King (2017). High Quality Face Recognition with Deep Metric Learning.

[16] David Powers (2011). Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness & Correlation.

[17] Ray Smith (2007). An Overview of the Tesseract OCR Engine.

[18] Kethineni Venkateswarlu, Sreerama Murthy Velaga (2015). Text Detection on Scene Images Using MSER.

[19] Di Wen, Hu Han, Anil K. Jain (2015). Face Spoof Detection with Image Distortion Analysis.

[20] Chin Lun Lai, Chiu Yuan Tai (2016). A Smart Spoofing Face Detector by Display Features Analysis.