Report

Photos Recognition on Student ID Cards using Deep Learning.

End-of-First-Year Project

Software Engineering Program

Achieved by:

Adnane MANDILI

Zakaria BAOU

Supervised by:

M. Rachid OULAD LHAJ THAMI

Academic year 2022/2023.

Dedication

We dedicate this effort first and foremost to those who have made sacrifices for our education and well-being: our parents, who have made sacrifices to care for us throughout our education and who are the source of our success; may God keep and protect them.

To our families and beloved friends who have stood by us in the most trying times, aswell as to all those who are dear to us, to you all.

Thank you.

Acknowledgment

We would like to express our gratitude to all the people who have contributed to the realization of this project. First, we would like to thank Mr. Rachid OULAD LHAJ THAMI for his support and guidance throughout this process. His expertise and wise advice were of great help in successfully completing this project.

We extend our thanks to Pr. MOUNIA Abik , Pr. BOUTAINA Hdioud and other jurys for agreeing to evaluate our work.

Finally, we would like to thank the entire faculty and the administrative staff of the National Higher School for Computer Science and Systems Analysis for all the teachings they have imparted to us during this year. Their support and commitment have been a valuable asset in the realization of this project.

Once again, we would like to express our gratitude to all the people who have contributed to the realization of this project. Their help and support were essential to the success of this work.

Summary

This document is the result of our work as part of the project at the end of the first year of the engineering cycle in Software Engineering. The aim of this project was to set up photo recognition on a student card as well as the detection of text in these cards, to then compare the information present on the student card studied with that present in our database, and to determine whether they are compatible or not. This is to reduce student card fraud. Following the great success of convolutional neural networks (CNN) in image classification and recognition, we used a deep learning method to build our model.

First, we carried out a pre-processing of the data by resizing the student cards to reduce the complexity of the model and thus reduce the computing power necessary for its training.

Then, to automatically extract faces from student cards, we used the pre-trained Haar Cascade Frontal Face model. We then processed and resized the face image to prepare it for our facial recognition model.

Finally, we apply our facial recognition model to the image and it returns the name of the person. Next, we deployed our model to a website for ease of use, using Flask as the API and HTML/CSS for the front-end.

Key Words : API, HTML, CSS, CNN, Flask.

Abstract

The present document is the result of our work as part of the end-of-first-year project for the Software Engineering program. The objective of this project was to implement a photo recognition system on student ID cards, as well as to detect the text on these cards, and then compare the information on the studied student ID card with the information in our database to determine if they are compatible or not. This is aimed at reducing fraud related to student ID cards.

Given the great success of Convolutional Neural Networks (CNNs) in image classification and recognition, we used a deep learning method to build our model.

Firstly, we performed data preprocessing by resizing the student ID cards to reduce the complexity of the model and decrease the computing power required for training.

Next, we used pre-trained models to extract the information from the student ID cards. To automatically extract the text, we used keras-ocr. To automatically extract faces from the student ID cards, we used the pre-trained Haar Cascade Frontal Face model. We then processed and resized the face image to prepare it for our facial recognition model.

Finally, we applied our facial recognition model to the image and returned the person's name, which was then compared to the name in our database, which returns "True" if they are compatible and "False" otherwise.

We then deployed our model on a website to facilitate its use, using Flask as an API and HTML/CSS for the frontend.

Key Words : API, HTML, CSS, CNN, Flask.

LIST OF FIGURES

CONTENTS

CHAPTER

1

GENERAL INTRODUCTION

General Introduction

During our first year at ENSIAS, we embarked on a comprehensive exploration of the field of computer science, where we acquired essential knowledge and skills. As part of our educational journey, we were tasked with a project aimed at enhancing our expertise and applying the principles we had learned in a real-world context.

Our primary goal was to develop an advanced identity card verification program utilizing deep learning techniques. By embracing this innovative approach, we aimed to address the prevalent issue of card fraud more effectively than existing solutions. The core foundation of our proposed system lies in its ability to employ artificial neural networks for recognizing faces on identity cards and accurately associating them with corresponding names from a pre-established database. The implementation of this program holds significant potential for a wide range of applications.

Recognizing the importance of creating an efficient verification system, we initiated a study specifically focused on ENSIAS student cards. This involved extracting photographs from the cards to facilitate facial recognition. Subsequently, the program would identify the individual in the photograph by utilizing the available database. Looking ahead, this technology could potentially be expanded to encompass other forms of identification, such as national identity cards, passports, and more.

CHAPTER

2

GENERAL CONTEXT OF THE PROJECT

2.1. Introduction

The manual verification process of photographs on student ID cards poses challenges in terms of accuracy, efficiency, and scalability. Human errors, time-consuming procedures, and limitations in handling large volumes of data , call for an automated and accurate solution to streamline identification and enhance security in educational institutions.

2.2. Problem

How can a deep learning-based pipeline be developed and implemented effectively to detect faces and recognize facial patterns from a student card image, with the ultimate aim of enhancing student card security? What challenges and limitations could potentially arise during the process of developing such a system and how could these be addressed to ensure successful integration of face detection, facial recognition, machine learning models, and front-end technologies?

2.3. Qualification of the need

In this section, we'll go over how to identify all of our system's functionalities for each type of user,starting with a list of functional needs, then understanding the list of requirements translated into non-functional needs, and finally looking at the various use cases that our system has to offer.

- **Who does it serve?** : customer or intended user
- **What does it act upon?**: elements on which the subject acts, the work material
- **For what purpose?** : main need to satisf

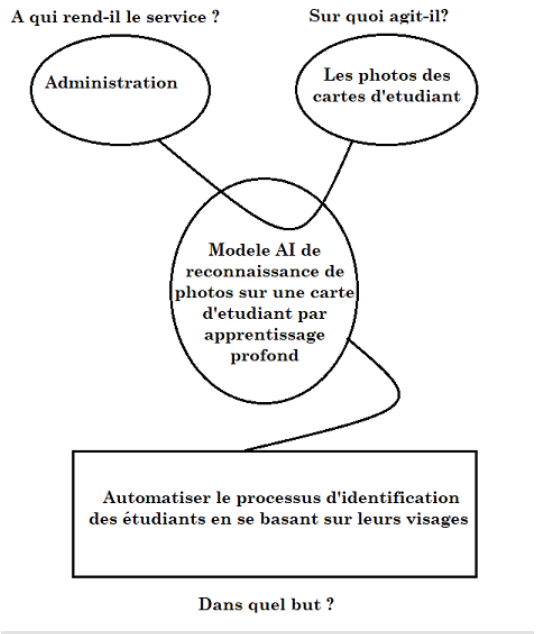

Figure 2.1: Bete a cornes

2.4. Approach :

The main objective of this project is to propose a functional pipeline based on deep learning for face detection and facial recognition from a student card image. The project is divided into four principal parts:

1. Cut the student card from the original image by extracting ORB features with OpenCV.
2. Use a pre-trained detector (Haar Cascade Frontal Face) to automatically extract faces from the student card.
3. Train a neural network for facial recognition of the faces in our database. The specific machine learning model used in this project is the very popular Convolutional Neural Network (CNN).
4. Deploy the model on a Web Site using Flask and HTML/CSS as front-end technologies.

Thus, this project aims to develop a comprehensive solution for verifying the security of student cards using deep learning techniques, face detection, and facial recognition. The resulting mobile application can help prevent identity fraud and improve the security of individuals and organizations.

With that, we can draw a UML diagram to visualize the various steps of our project. UML diagrams provide a clear representation of the structure and behavior of the system being developed. By creating a UML diagram, we can better understand the relationships between the different classes and components of our project and ensure that they are working together seamlessly. The UML diagram shows the various classes and components involved in the project, as well as the relationships between them. The main objective of the project is to propose a functional pipeline for face detection and facial recognition from a student card image, using deep learning techniques. The project is divided into four main parts, represented by the following classes in the diagram:

- **StudentCardImage:** This class represents the student card image and contains methods for extracting ORB features from the image and cropping the image to isolate the student card. In addition, the class includes a method for matching the extracted ORB features with the ORB

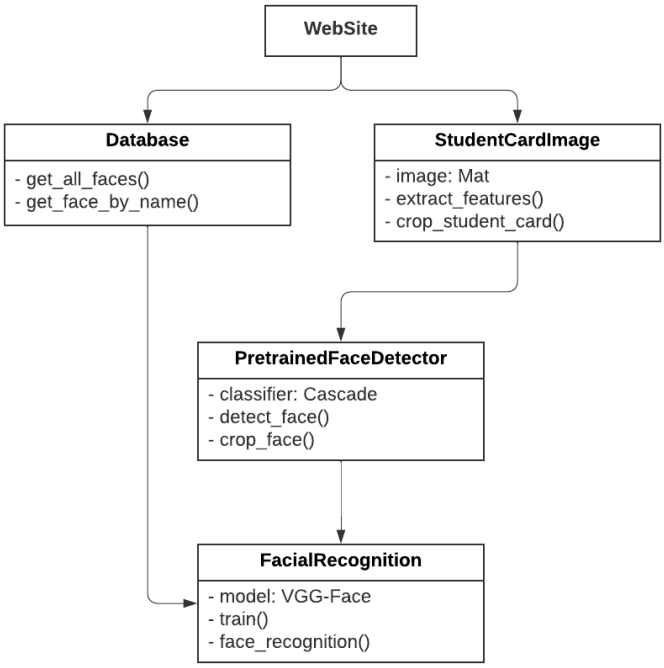

Figure 2.2: Uml class

features of a student card template. This matching process involves comparing the descriptors of the keypoints extracted from the student card image with those of the keypoints extracted from the template. The goal of this process is to verify that the extracted image corresponds to a student card and to extract the relevant information from the card.

- **Database:** This class represents the component responsible for storing and retrieving student information used in the facial recognition system as well as the text used in the data comparaison.
- **PretrainedFaceDetector:** This class represents the Haar Cascade Frontal Face detector used to automatically extract faces from the student card image. Haar Cascade is an Object Detection Algorithm used to identify faces in an image or a real time video.
- **FacialRecognition:** This class represents the Convolutional Neural Network (CNN) used for facial recognition of the extracted faces. The specific machine learning model used in this project is the VGG Face model, which is a deep neural network trained for face recognition tasks. The class includes methods for training the VGG Face model on a database of facial images and for using the trained model to recognize the faces extracted from the student card image. The recognition process involves extracting facial features from the input images and comparing them with the features of the faces in the database using techniques such as Euclidean distance or

cosine similarity. The goal of this process is to identify the person depicted in the input image and match them with the corresponding entry in the database.

The UML diagram provides a clear overview of the project's structure and how the different components work together to achieve the desired functionality.

2.5. Work Organisation

our project's administration is based on a chronological breakdown of tasks and resources, organized into phases. To visualize and manage the project's schedule, we utilize a Gantt chart, which displays a list of activities along with their start and end dates. This chart helps us monitor progress, identify dependencies, and ensure the project stays on track. By leveraging the Gantt chart, we streamline workflow, track deadlines, and optimize resource allocation, leading to improved project efficiency and success. It is presented as follows :

2.6. Conclusion

This chapter served as the foundation for the project's development, as it outlined the overall goal to be achieved, the project's specifications, as well as the methodology of the life cycle methodology employed and the project's planning.

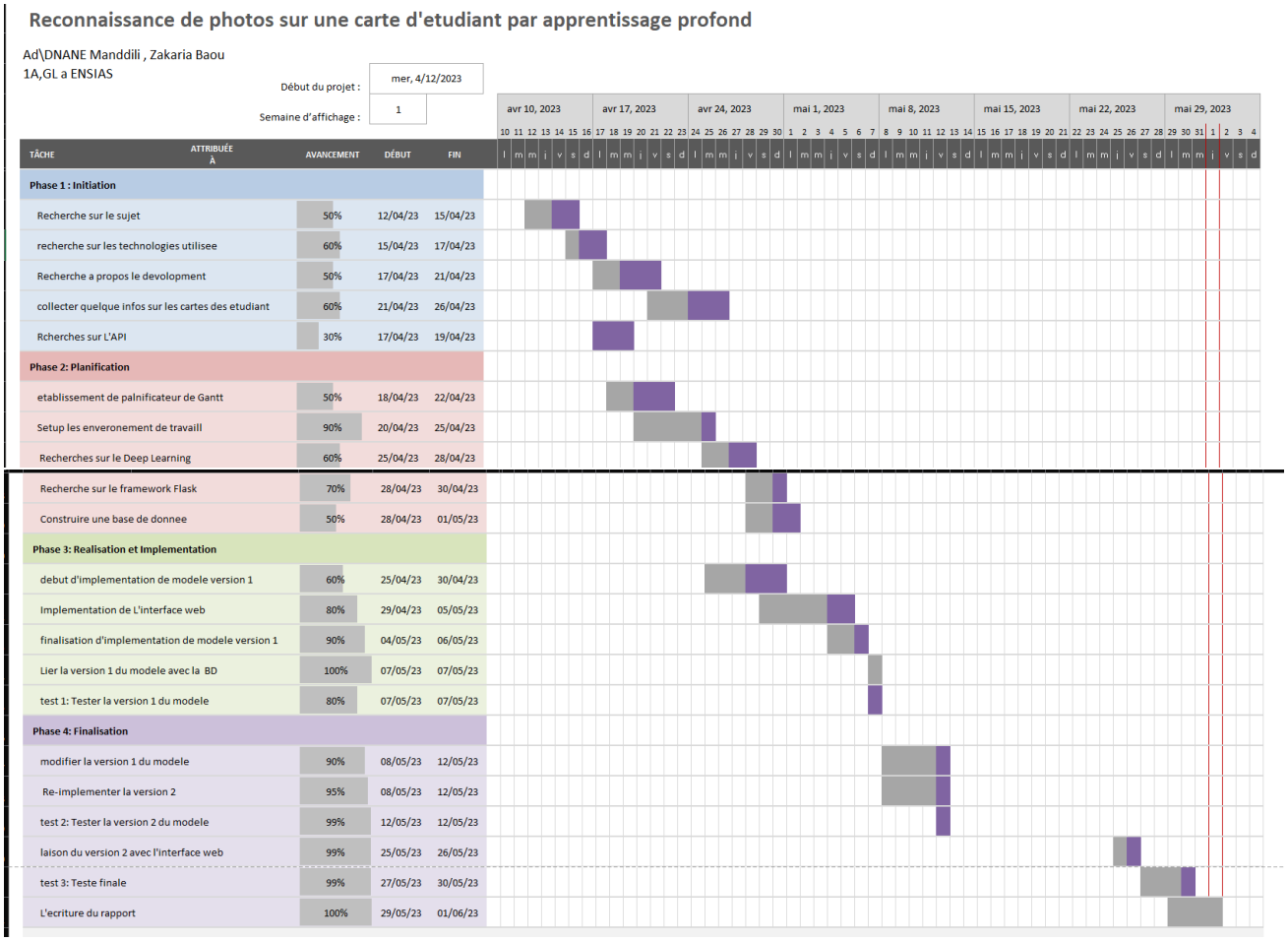

Figure 2.3: Gantt chronological diagram

CHAPTER

3

USED TOOLS

3.1. Oriented FAST and Rotated BRIEF

the initial step in our project involves cropping the student card from the original image. To achieve this, we employed the extraction of ORB (Oriented FAST and Rotated BRIEF) features. ORB was developed at OpenCV Labs in 2011 by Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski.

ORB is an effective and efficient feature extraction method that builds upon two well-established techniques: the FAST (Features from Accelerated Segment Test) keypoint detector and the BRIEF (Binary Robust Independent Elementary Features) descriptor. The combination of these methods allows ORB to provide robust and accurate feature extraction, while maintaining computational effi-

ciency and a low cost.

ORB (Oriented FAST and Rotated BRIEF) is a feature extraction algorithm that combines the strengths of the FAST keypoint detector and the BRIEF descriptor. It is designed to be computationally efficient and robust, making it suitable for real-time applications. The ORB algorithm can be broken down into several steps:

1. **Scale-space pyramid:** To ensure scale invariance, the input image is divided into multiple layers, each representing a different scale. This allows ORB to detect features across various scales.

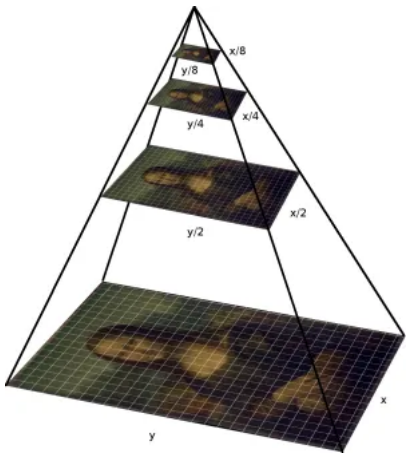

Figure 3.1: Scale-space pyramid

2. **Keypoint detection:**Using the FAST keypoint detector, ORB identifies keypoints (corners) in the image by examining pixel intensity differences within a small circular region. The FAST algorithm's performance is improved by applying a Harris corner measure to retain only the strongest keypoints.

3. **Orientation assignment:** To achieve rotation invariance, ORB assigns an orientation to each

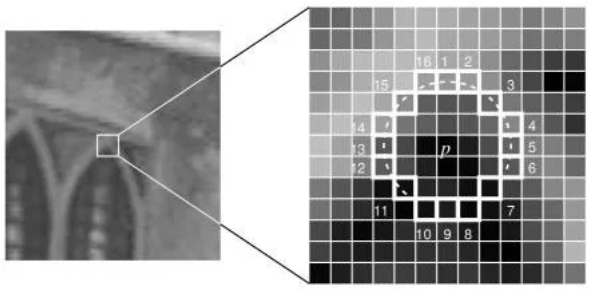

Figure 3.2: Keypoint

keypoint based on the intensity-weighted centroid of the patch surrounding the keypoint. The angle between the centroid and the keypoint determines the orientation

4. **Rotation-aware BRIEF descriptor:** ORB modifies the original BRIEF descriptor by incorporating the orientation information from the previous step. This results in a rotation-invariant descriptor called rBRIEF.

5. **Efficient descriptor matching:** Using the Hamming distance, ORB efficiently matches the binary rBRIEF descriptors between images. The low memory footprint of the binary descriptors and the simplicity of the Hamming distance calculation allow for fast and efficient matching.

In summary, ORB is a feature extraction method that combines the speed and efficiency of the FAST keypoint detector and the BRIEF descriptor, while adding rotation invariance and maintaining scale invariance. This makes ORB a powerful choice for a wide range of computer vision applications.

3.2. Haar Cascade

The Cascade Classifier is a machine learning-based approach for object detection. This methodology was developed by Paul Viola and Michael Jones. The idea behind this strategy is not to process the entire image when detecting faces but rather to identify distinctive features or attributes associated with faces. This results in a significant improvement in detection efficiency and speed.

The classifier is described as a ”cascade” because it encompasses multiple stages, each of which is capable of identifying faces with increasing levels of accuracy. The ”Haar Features” are specific, identifiable patterns or characteristics within an image, named after Haar wavelets.

Training the classifier requires a substantial dataset composed of positive examples (images containing the object, such as faces) and negative examples (images that do not contain the object). The training process essentially involves the application of the algorithm to find the best features in these examples. These features are then used to create a series of classifiers or a ”cascade.”

In the detection phase, this cascade of classifiers is applied to an unknown image to detect the presence of faces. This works by moving a window across the image and using the classifiers to determine whether each window contains a face or not.

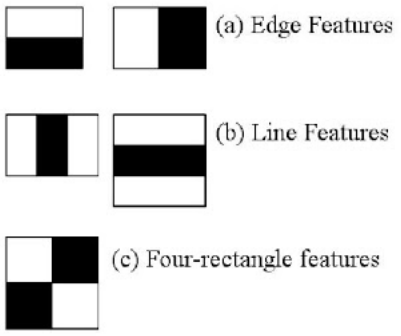

Figure 3.3: Haar features

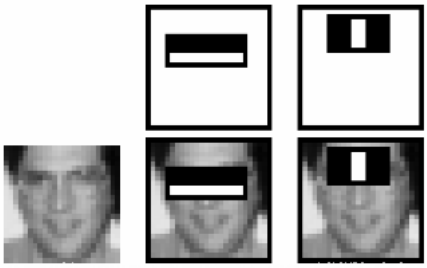

Figure 3.4: haar

3.3. Deep Learning

Deep learning is a set of machine learning methods designed based on deep neural networks, aimed at mimicking the ”depth” of layers in a brain. The human brain is ”deep” in the sense that each action is the result of a long chain of synaptic communications with many layers of processing. Deep learning brings together a class of learning algorithms corresponding to these deep architectures. It is often used for ”end-to-end” learning, which means simultaneous learning of useful data features and the best way to use them.

3.3.1 Artificial neural network

These are mathematical functions with several adjustable parameters. The analogy dates back to the first automata proposed in 1943 by Warren McCulloch and Walter Pitts. Like in the neurons of the brain where connections are created, disappear or strengthen depending on various stimuli and produce an action, artificial (or formal) neural networks adjust parameters (called synaptic weights in reference to the biological functioning of the brain) based on input data in order to provide the best

possible response.

A neuron makes a linear combination of the inputs it receives, to which it adds a value called bias. Then a non-linear activation function (such as hyperbolic tangent, for example) is applied to the output value. This value is then transmitted to the next layer of neurons. Each neuron thus performs a very rudimentary calculation, and it is the succession of layers of neurons that allows for complex networks to be obtained. The feedforward neural network was the first and simplest artificial

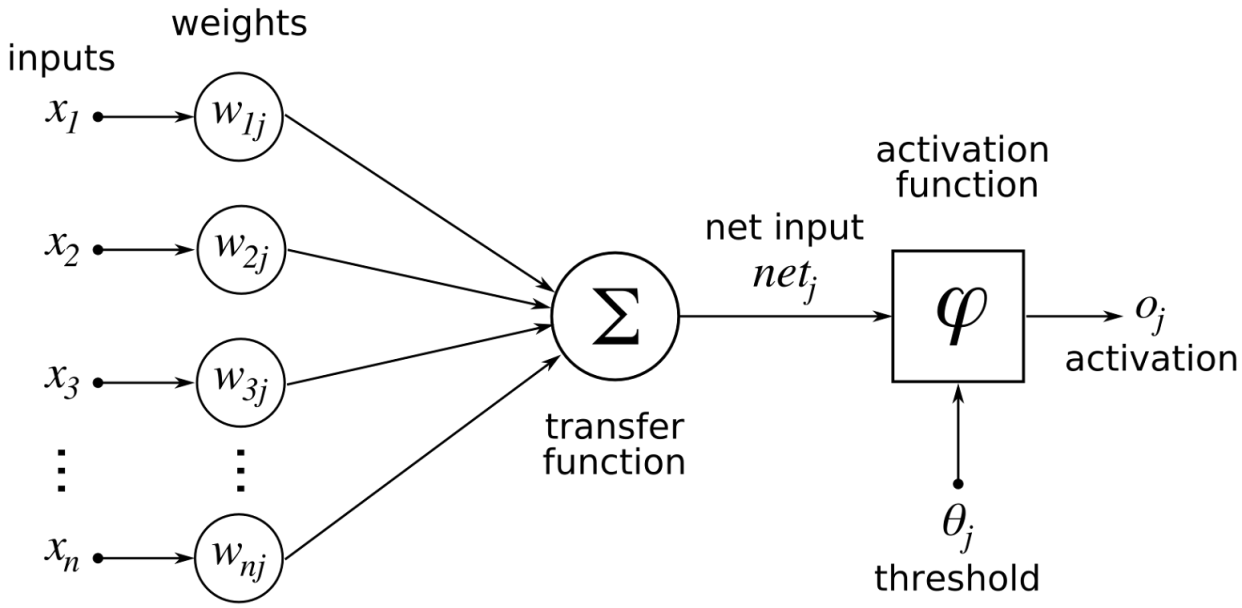

Figure 3.5: Artificial neural network 1

neural network developed. It contains several neurons arranged in layers. The neurons of adjacent layers have connections between them. All these connections have weights associated with them. A feedforward neural network can consist of three types of neurons:

- **Input neurons:** The input neurons provide information from the outside world to the network and together form an ”input layer”. No calculations are performed in the input nodes. They simply transmit information to the hidden nodes.
- **Hidden neurons:** Hidden neurons have no direct connection to the outside world (hence the name ”hidden”). They perform calculations and transfer information from input neurons to output neurons. A set of hidden neurons forms a ”hidden layer”. Although a feedforward network historically had only one input layer and one output layer, it can have zero or several hidden layers, following a principle of generalization.

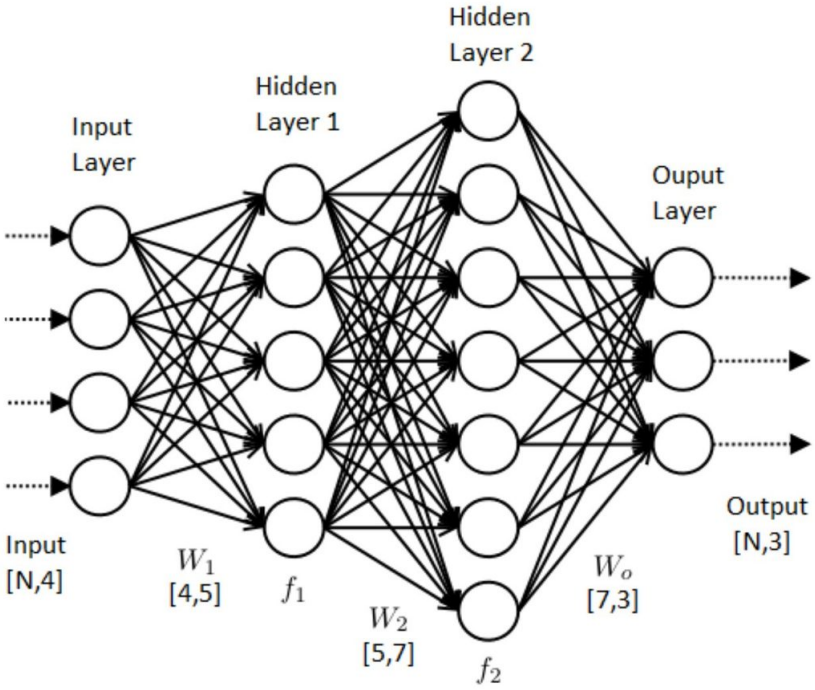

Figure 3.6: Artificial neural network 2

- **Output neurons:** The output neurons form an "output layer" together. They are responsible for calculations and transferring information from the network to the outside world.

3.3.2 Training a neural network

The training phase consists of adjusting the weights of a neural network to obtain the best regression or classification results. The goal of this optimization is to minimize the cost function, which calculates the error between the data in the training set and the data predicted by the neural network.

One of the proposed optimization algorithms is the backpropagation algorithm which is based on gradient descent.

The gradient descent algorithm is used to approach its local minimum by searching for points at which its gradient is zero. In the case of neural networks, this means that, at each iteration, backpropagation calculates the derivative of the cost function with respect to each weight W and subtracts it from that weight.

Among the problems that can be encountered during the learning process is overfitting. This is due to poor parameterization of the gradient descent hyperparameters or problems related to the database. Following overfitting, the neural network becomes capable of capturing information that

is not useful for accomplishing its task or it becomes unable to generalize the characteristics of the data, which limits its ability to recognize new data. To avoid this problem of overfitting, regularization techniques have been proposed. The first technique is Dropout. It involves eliminating certain neurons at each iteration to reduce the number of network parameters and update a reduced number of weights. The second technique is data augmentation. It involves increasing the number of training samples by adding some variety without affecting their semantics. For example, we can add images by introducing rotations, vertical symmetry or even changing their dimensions.

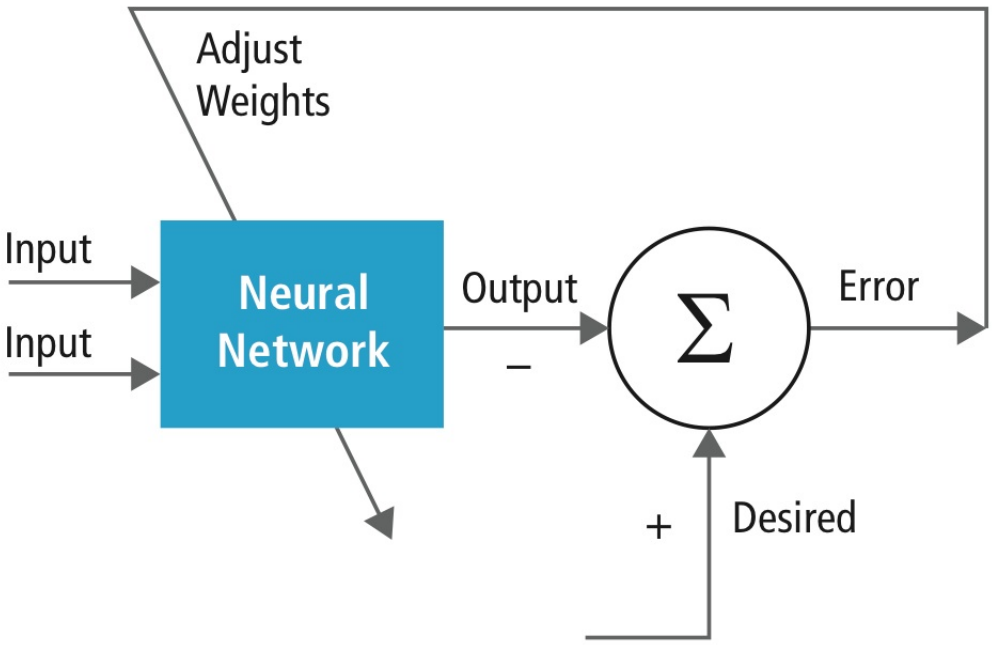

Figure 3.7: training Neural Networks

3.3.3 Convolutional Neural Networks

Convolutional networks, also known as Convolutional Neural Networks (CNNs), are a specific type of multilayer neural network inspired by the structure and functionality of the visual cortex in mammals. They are capable of processing information in a hierarchical manner, from simple to complex, and have gained significant popularity in various domains, particularly in computer vision tasks.

CNNs are composed of multiple layers of neurons, called mathematical functions, with adjustable parameters. These networks preprocess small amounts of information at each layer. The key characteristic of CNNs is their first few layers, typically one to three convolutional layers. The convolutional layers are named after the mathematical operation of convolution, which they employ

to detect patterns in signals or images.

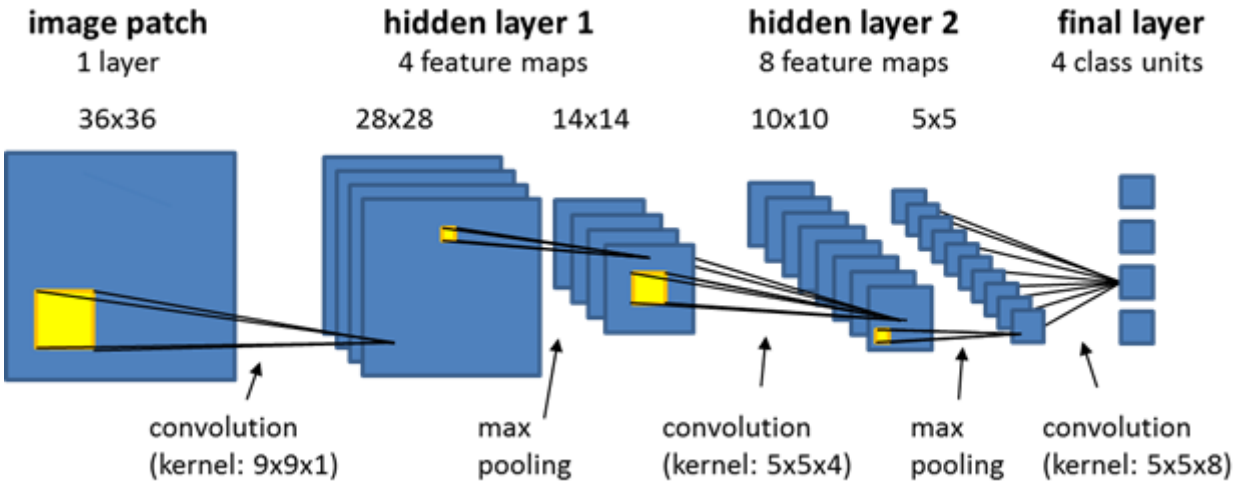

Figure 3.8: Convolutional Neural Networks

In the context of images, the first convolutional layer is designed to identify basic features like object edges, such as a circle. The subsequent convolutional layers combine these edges to form higher-level representations, such as objects like a wheel. Further layers, which may not necessarily be convolutional, utilize this information to distinguish between different objects, for example, identifying whether an input represents a car or a motorcycle. CNNs undergo a learning phase, where the networks are trained on known objects by being exposed to thousands of relevant images, such as images of dogs, cars, or sports. This training helps determine the optimal parameters for the network to make accurate predictions.

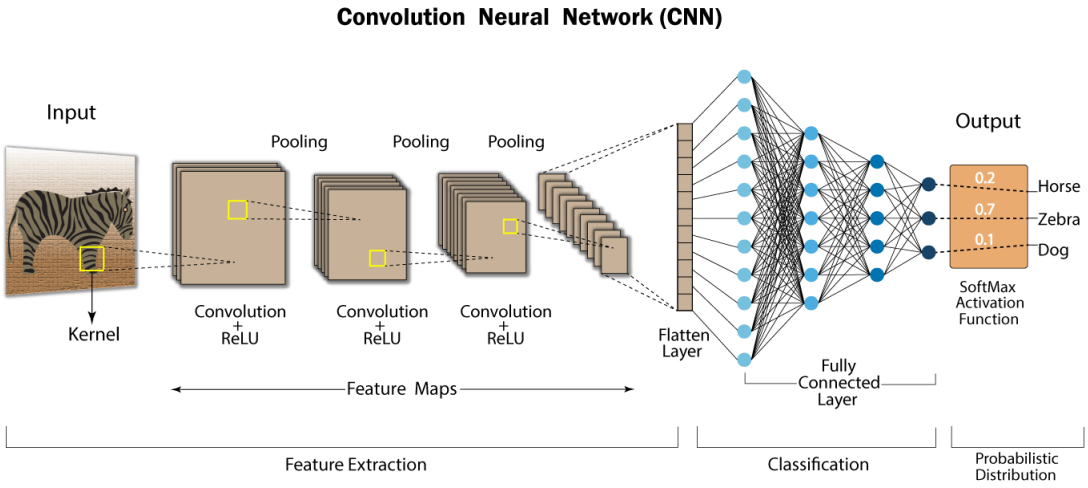

Figure 3.9: Convolutional Neural Networks

To introduce non-linearity and enhance the capabilities of CNNs, several techniques are commonly used:

1. Rectified Linear Unit (ReLU) Function: ReLU is an activation function applied to the output of neurons in the CNN. It is defined as ReLU(x) = max(0, x), where x represents the input to the function. ReLU sets negative values to zero while retaining positive values, thereby introducing non-linearity and enabling the network to learn complex patterns effectively.

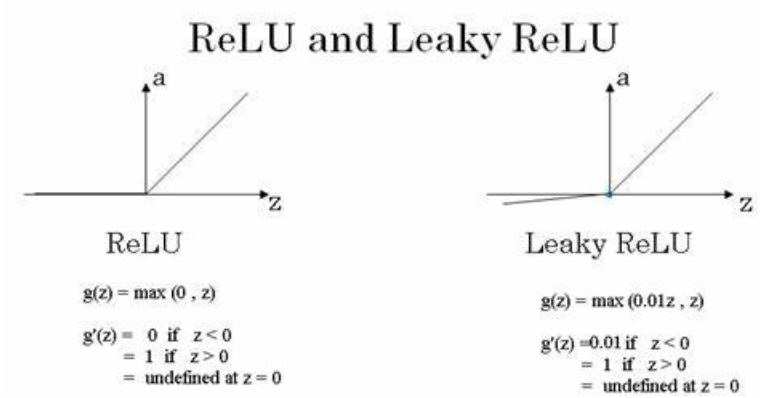

Figure 3.10: relu function

2. Max Pooling: Max pooling is a pooling operation used in CNNs to downsample the spatial dimensions of the feature maps. It involves dividing the feature map into non-overlapping regions and selecting the maximum value within each region. Max pooling helps to reduce the computational complexity, achieve translation invariance, and retain the most salient features of the input data.

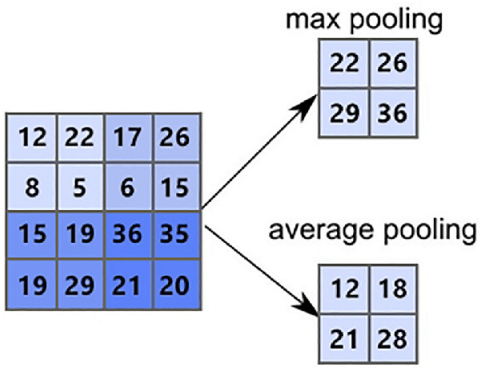

Figure 3.11: max pooling

3. Softmax: Softmax is an activation function commonly used in the output layer of a neural network, including CNNs, for multi-class classification tasks. It converts the network's raw output

scores into a probability distribution over multiple classes. Softmax computes the exponential of each input score and normalizes the results to sum up to 1, assigning higher probabilities to larger scores. It enables the network to make confident predictions by selecting the class with the highest probability.

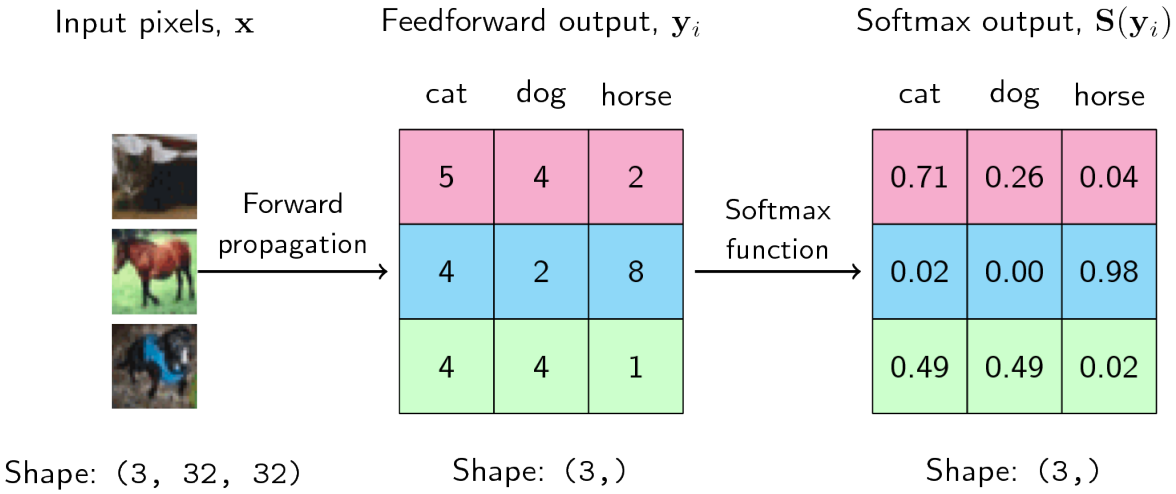

Figure 3.12: Softmax

By combining convolutional layers, ReLU activation, max pooling, and softmax, CNNs can effectively learn hierarchical representations from raw input data, making them powerful tools for image recognition, video analysis, and natural language processing tasks.

The VGG (Visual Geometry Group) architecture is a widely used convolutional neural network (CNN) model in the field of computer vision. It was proposed by Simonyan and Zisserman from the University of Oxford in 2014. The VGG architecture is known for its depth and simplicity.

Here are the main characteristics of the VGG architecture:

1. **Layer Structure:** The VGG model consists of stacked convolutional layers. There are different versions of VGG, including VGG16 and VGG19, which differ in the number of layers.
2. **Convolution:** The convolutional layers consist of small-sized filters (e.g., 3x3) that analyze local features of the image by performing convolution operations.
3. **Pooling:** After every few convolutional layers, a pooling layer is used to reduce the spatial dimensions of the output while preserving essential features. The most commonly used pooling is max pooling.
4. **Activation Functions:** Non-linear activation functions, such as ReLU (Rectified Linear Unit), are used to introduce non-linearities into the model.
5. **Fully Connected Layers:** The output from the convolutional layers is flattened and connected to fully connected layers. These final layers perform classification using activation functions like softmax.
6. **Pre-training:** VGG models are often pre-trained on large datasets, such as ImageNet, to capture general image features. Subsequently, these models can be fine-tuned for specific tasks.

3.4. Tools and used Languages

Python

Figure 3.13: python logo

Python is an incredibly versatile and powerful programming language, renowned for its simplicity and readability, which makes it an excellent choice for both novice and experienced developers. In the context of our deep learning model, Python was selected primarily due to its extensive suite of scientific computing and machine learning libraries, such as TensorFlow, PyTorch, Keras, and NumPy. These libraries simplify the complex mathematical operations and algorithms involved in deep learning, allowing us to develop and iterate our model with greater efficiency. Additionally, Python's broad and active community consistently provides updated resources, problem-solving aid, and cutting-edge developments in the field of AI, ensuring that we are equipped with the latest tools and techniques in the industry.

HTML, CSS

Figure 3.14: HTML logo

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are fundamental technologies used in building web pages and web applications. HTML provides the basic structure of the site, defining elements like headers, paragraphs, and data tables, while CSS dictates the layout and appearance of these elements, such as color, font, and spacing. To deploy our deep learning model,

we used HTML and CSS to create an intuitive and user-friendly interface.

OpenCV

Figure 3.15: OpenCV logo

OpenCV (Open Source Computer Vision Library) is a powerful and comprehensive library aimed at real-time computer vision. It consists of more than 2500 optimized algorithms, which are widely used for various image and video analysis processes such as face recognition, object identification, and many more. In our project, we utilized OpenCV for its Oriented FAST and Rotated BRIEF (ORB) feature detector and extractor.

Tensorflow, Keras

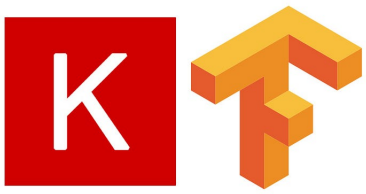

Figure 3.16: Keras logo

TensorFlow is an open-source deep learning library developed by the Google Brain team, widely used for creating and training neural networks. Keras provides an easier mechanism to express neural networks with its high-level building blocks. It abstracts many low-level details of TensorFlow, thereby simplifying the process of designing and implementing neural networks. Keras allows developers to construct and train complex deep learning models with just a few lines of code, accelerating the development process while maintaining a high degree of flexibility and control.

Figure 3.17: Flask logo

Flask

Flask is a lightweight Python web framework that is optimal for small scale projects but also capable of powering complex web applications. It is known for its simplicity and adaptability, allowing developers to use any kind of libraries or tools they prefer. In the context of deploying a deep learning model, Flask is used to develop a web server that communicates with the model. After training the model with the deep learning framework, we wrote a Flask application to accept and process incoming data, run this data through the model, and return the model's predictions.

Jupyter Lab

Figure 3.18: Jupiter logo

JupyterLab is a web-based interactive development environment, offering a powerful and flexible platform for working with Jupyter notebooks, code, and data. It extends the classic Jupyter Notebook by adding a highly customizable interface that integrates text editors, terminals, and other components. JupyterLab is often used in data science and machine learning for its ability to combine narrative text with executable code, facilitating tasks like data cleaning, statistical modeling, and machine learning in a single, interactive document.

Visual Studio Code

Visual Studio Code (VS Code) is a highly versatile and free source-code editor developed by Microsoft. It comes with built-in support for JavaScript, TypeScript, and Node.js, along with a rich ecosystem of extensions for other languages such as Python, C++, Java, and many more. VS Code provides

Figure 3.19: VScode logo

features like debugging, intelligent code completion, snippets, embedded Git control, and GitHub, making it a popular choice for developers across different platforms.

CHAPTER

4

IMPLEMENTATION

4.1. Cropping the student card

As previously discussed, we will implement the ORB (Oriented FAST and Rotated BRIEF) algorithm using the OpenCV library. To do this, we require an ENSIAS student card template. We have created a customized template using Adobe Photoshop, where we removed specific information such as the student's name, picture, QR code, etc. This was done to ensure that the program does not mistakenly consider these elements as keypoints during the feature detection and matching process. This will help improve the accuracy and efficiency of our ORB implementation. Firstly, we'll begin by importing the OpenCV library. Next, we'll import two images: the template student ID card and a photograph of an actual ID card. To verify that the images have been successfully imported, we'll display them using the Matplotlib library. We will first convert the two images to grayscale, as grayscale images are better suited for feature detection and matching due to their reduced computational complexity. Next, we choose 500 as the maximum number of features for the ORB algorithm. This number strikes

Figure 4.1: student ID card exemple template

Figure 4.2: template of card and the original picture

a good balance between performance and complexity. To detect keypoints in the two images (im1 and im2), we use the ORB_create() function to create an ORB object and the orb.detectAndCompute() method to store the keypoints and descriptors in variables. Finally, we display the two images to visualize the detected keypoints. Now, after extracting the keypoints we'll performs feature matching between the two images (im1 and im2) using the detected keypoints and descriptors from the ORB algorithm. The matches are then drawn onto a combined image for visualization. We then extract the matched keypoints coordinates from the two images (im1 and im2) and calculate a homography matrix using the RANSAC algorithm with the OpenCV function cv2.findHomography() and retrieve the height, width, and the number of channels (usually 3 for color images) of the first image (im1) by accessing its shape attribute and use the cv2.warpPerspective() function to apply a perspective transformation on the second image (im2) using the homography matrix h. The transformed image, im2_reg, will have the same dimensions (width and height) as the first image (im1). We display the

Figure 4.3: comparison of the picture with the tempalte

images and we get: Now that we have aligned and cropped the student ID card from the original image, we'll save the cropped student ID card. After saving the cropped student ID card, we can proceed to the next step, where we will detect and crop the face from the card.

4.2. Cropping the face

Now that we have successfully performed a perspective transformation on the second image (im2) to align it with the template student ID card, we'll proceed to detect the face in the cropped student ID card. To accomplish this, we will use the Haar Cascade classifier for frontal face detection, which is a pre-trained classifier available in the OpenCV library.

After detecting the face in the cropped student ID card using the Haar Cascade classifier for frontal face detection, we will proceed to extract the face from the card by cropping the region of interest (ROI) containing the face. After detecting and cropping the face from the student ID card, we save the image and we can proceed to the face recognition step.

Figure 4.4: comparison of the picture with the template

Figure 4.5: comparison of the template with cropped pic

4.3. Face recognition

Several CNN models have been effectively trained for facial recognition tasks. In this study, we utilize the VGG-face model proposed by Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman in 2015. This model was trained on a dataset of 2.6 million faces and achieved an accuracy of 97.4%, surpassing Google's FaceNet model trained on a dataset of 200 million faces with an accuracy of 95.1% and Facebook's DeepFace model trained on 4 million faces with an accuracy of 91.4%. The high accuracy of VGG-face in recognizing faces is one of the reasons we chose to use it in our project.

VGG-Face consists of 22 layers and 37 deep units, 12 convolutional layers and 3 fully connected

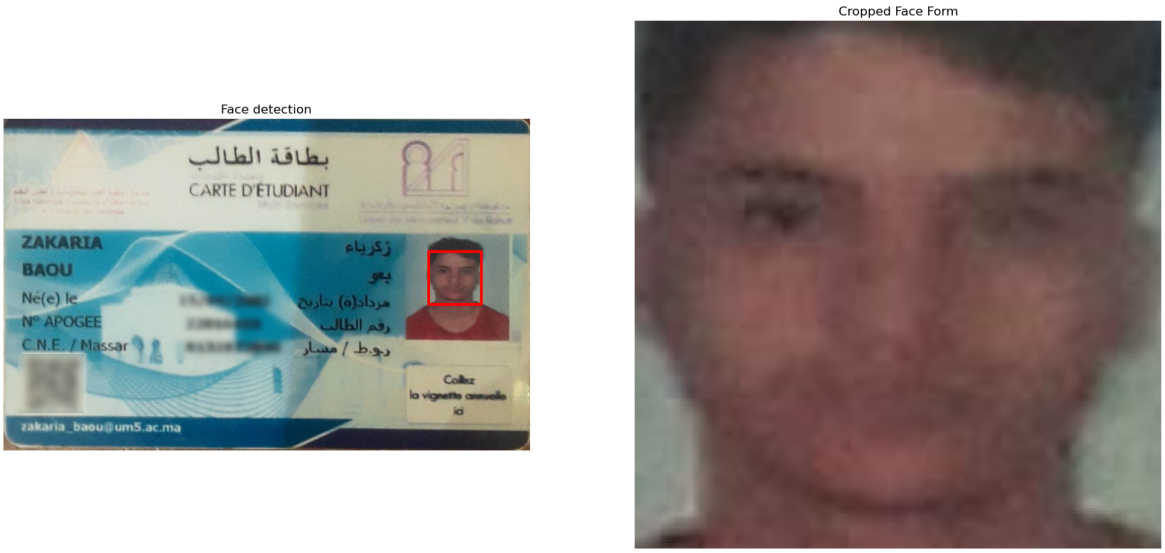

Figure 4.6: cropping the face from the picture

layers. Each convolutional layer is followed by a rectification layer, whereas a max pool layer is operated at the end of each convolutional block. Research paper denotes the layer structure as shown below. The image presented below helps to provide a clear understanding of the VGG-Face

layer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
type	input	conv	relu	conv	relu	mpool	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv
name	–	conv1_1	relu1_1	conv1_2	relu1_2	pool1	conv2_1	relu2_1	conv2_2	relu2_2	pool2	conv3_1	relu3_1	conv3_2	relu3_2	conv3_3	relu3_3	pool3	conv4_1
support	–	3	1	3	1	2	3	1	3	1	2	3	1	3	1	3	1	2	3
filt dim	–	3	–	64	–	–	64	–	128	–	–	128	–	256	–	256	–	–	256
num filts	–	64	–	64	–	–	128	–	128	–	–	256	–	256	–	256	–	–	512
stride	–	1	1	1	1	2	1	1	1	1	2	1	1	1	1	1	1	2	1
pad	–	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1

layer	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
type	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	softmx
name	relu4_1	conv4_2	relu4_2	conv4_3	relu4_3	pool4	conv5_1	relu5_1	conv5_2	relu5_2	conv5_3	relu5_3	pool5	fc6	relu6	fc7	relu7	fc8	prob
support	1	3	1	3	1	2	3	1	3	1	3	1	2	7	1	1	1	1	1
filt dim	–	512	–	512	–	–	512	–	512	–	512	–	–	512	–	4096	–	4096	–
num filts	–	512	–	512	–	–	512	–	512	–	512	–	–	4096	–	4096	–	2622	–
stride	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1
pad	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0

Figure 4.7: layers structure

architecture. For this project, we'll first create our dataset and import it into Python. To gather the data, we shared a Google form with ENSIAS students to collect their photos and names for use in face recognition. As a result, we received about 24 responses, providing us with 26 photos to use for training and testing our model. We then imported this data into Python and stored it in a dictionary, using the student's name as the key and the corresponding photo as the value. To ensure that the dataset was imported correctly, we can check the length of the student_images_dict dictionary. As expected, we have 26 labeled images in our dataset. Now, we will separate the dictionary's keys and values into two separate lists. The first list will contain the names (labels), and the second list

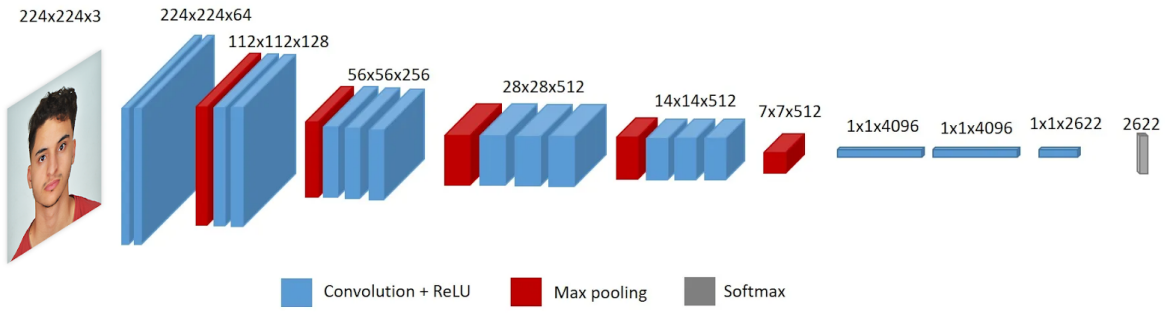

Figure 4.8: VGG face

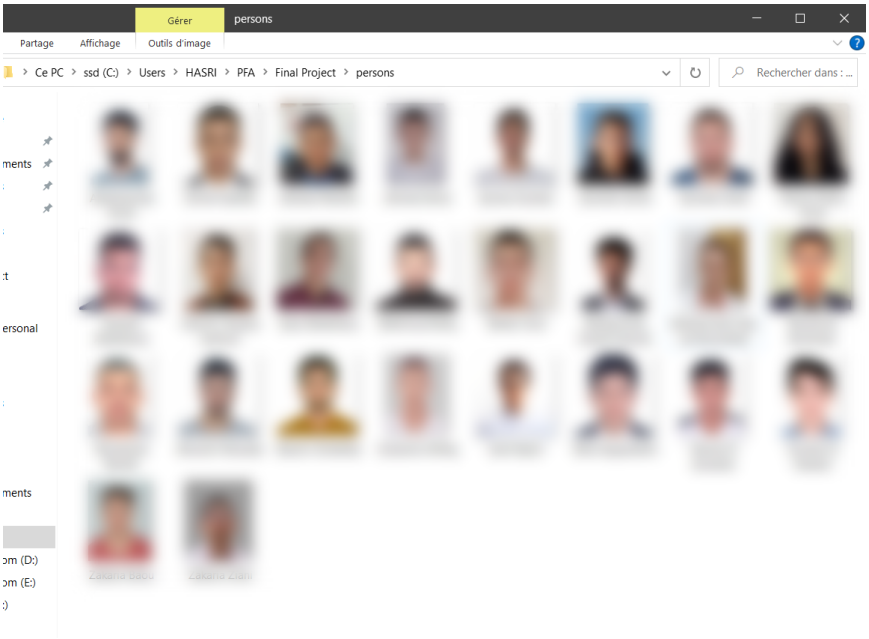

Figure 4.9: pictures used for comparison

will have the corresponding image paths. This separation allows for easier access and manipulation of the names and image paths while preserving their correspondence. After importing our dataset, we can proceed to construct the VGGFace model for face recognition as mentioned earlier. We'll be using the Keras deep learning library to construct our model. Now that we have defined the VGGFace model architecture, we can proceed to import the pre-trained weights available online. Pre-trained weights can help improve the performance of our face recognition model, as they have already been trained on a large dataset of faces. We will create a new model called vgg_face_descriptor by specifying the input and output layers of the original VGGFace model. The new model will be a submodel of the original one, keeping only the parts necessary for generating face embeddings (feature representations). Since the VGGFace model takes input only images of dimensions (224x224), we

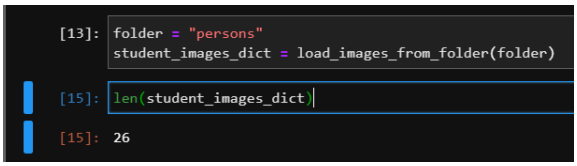

Figure 4.10: caption 1

define the fonction preprocess_image that takes an image path as input, loads the image, resizes it to the required dimensions (224x224), converts it into an array format, adds an extra dimension for the batch, and applies the necessary pre-processing required by the VGGFace model. To evaluate the similarities between two images, we will use Cosine similarity, which is a measure of similarity between two non-zero vectors. Cosine similarity values range from -1 to 1, with 1 signifying that the two vectors are identical and -1 indicating that they are entirely dissimilar. Mathematically, the Cosine similarity can be expressed as follows:

$$\textbf{cosine similarity} = S_C(A,B) = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

In this context, **A** represents the **source_representation** array and **B** represents the **test_representation** array. To facilitate working with these arrays, we will create a function that returns 1 minus the cosine similarity of the two arrays. This way, the resulting value will be smaller when the similarity is higher, making it more intuitive for comparison purposes.

After setting up the necessary functions, we'll begin by importing our target image and applying the required modifications. This will enable our model to predict a 2622-dimensional feature vector that captures the essential characteristics of a face, such as the shape of facial features, their relative positions, and other unique factors. We will then set our similarity threshold, epsilon, to 0.4. This value determines the degree of similarity required for a match to be considered valid. Next, we will create a list called 'verified' where we will store the index of the photo from the paths list that matches our input image. This way, we can efficiently identify and retrieve the matching images based on their similarity to the input image.

We will utilize a for loop to iterate over the paths list, which contains the image paths of our database. For each image in the database, we will import it, apply the necessary modifications, and let our model predict its features. Subsequently, we will use the findCosineSimilarity function to calculate the cosine similarity between the input image and each image in the database.

If the calculated cosine similarity is less than our epsilon threshold (0.4), the images will be considered as belonging to the same person. In that case, we will save the index of the matching image in the 'verified' list. This index will later be used to print the name of the person using the 'names' list.

To visualize how the cosine similarity changes for each image in the database, we will add a print statement that displays the cosine similarity value for each comparison. This provides insight into the degree of similarity between the input image and the images in the database.

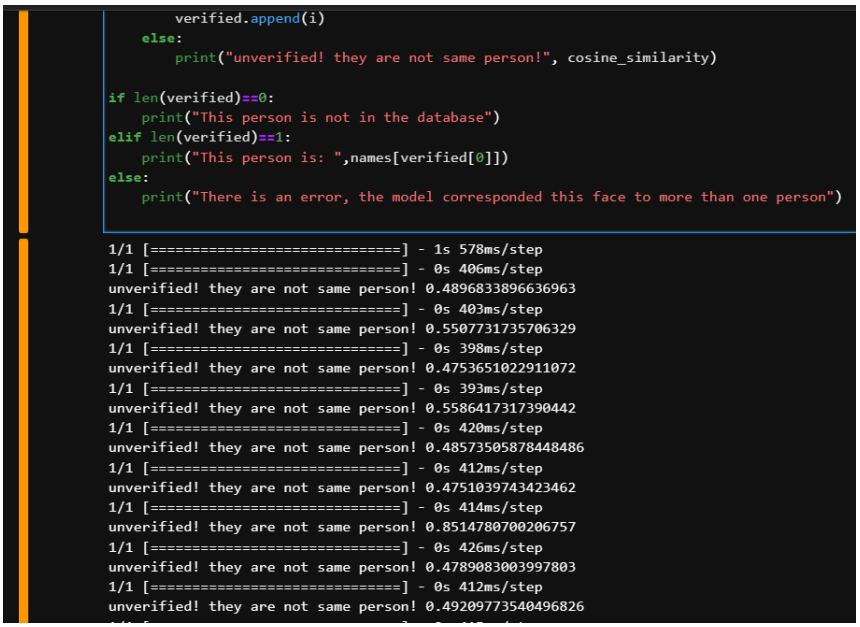

Figure 4.11: Caption 2

In the end, we will obtain the corresponding name of the person in the input image by comparing it to the images in our database and identifying the one with a cosine similarity below the threshold.

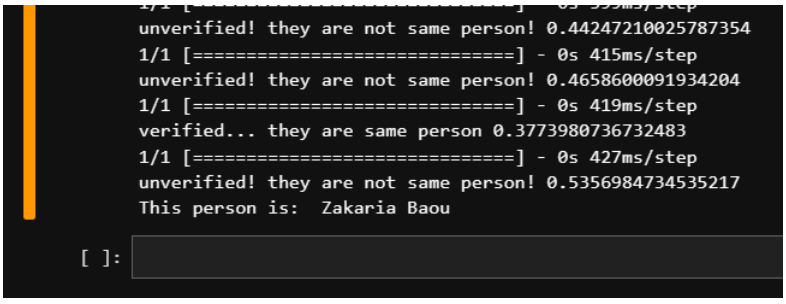

Figure 4.12: Caption 3

4.4. Deploying the model

To deploy our model, we utilized Flask, which is a widely used micro web framework for creating APIs in Python. It is a simple yet powerful web framework that is designed to get started quickly and easily, with the ability to scale up to complex applications. We designed a simple homepage for our application using HTML and CSS to provide a user-friendly interface. This allows users to interact with our Photos recognition on Student ID Cards model, making it more accessible and easy to use.

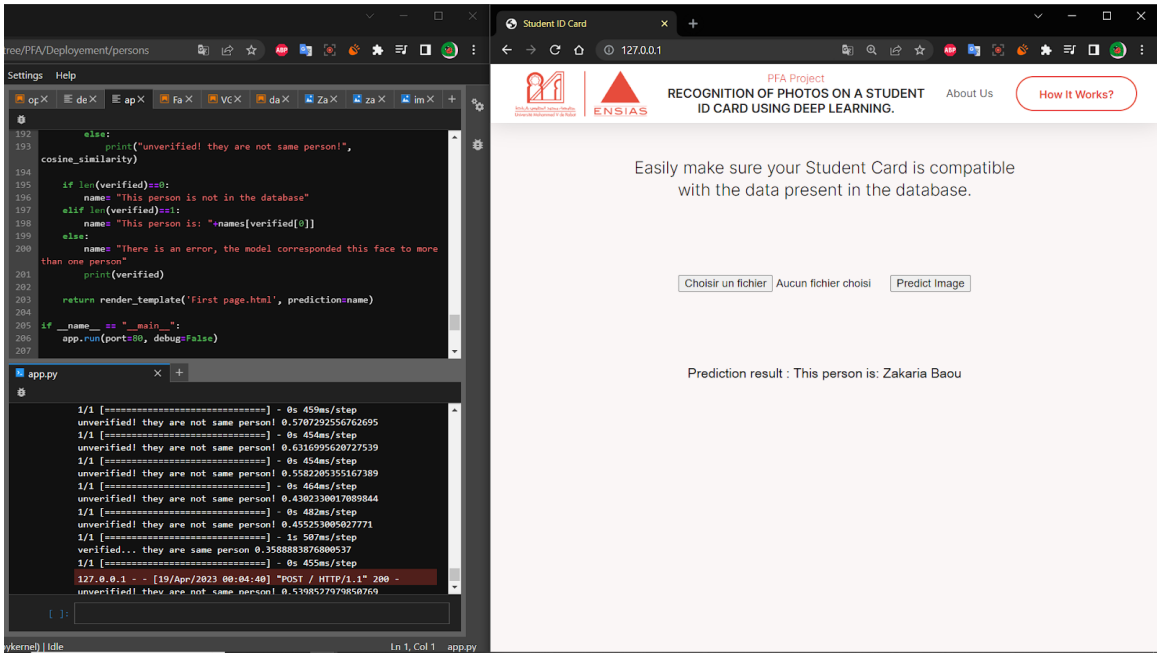

Figure 4.13: plate form

CHAPTER

5

GENERAL CONCLUSION

In summary, this project aimed to develop an AI model for efficient recognition of student ID card pictures. We presented the process from conception to implementation, focusing on a cross-platform application that verifies and extracts information from ID cards.

Throughout the project, we gained valuable insights into image recognition, data processing, and development, enhancing our problem-solving and collaboration skills. Future possibilities include integrating a database, implementing additional security measures, integrating with existing student management systems, and creating a web-based administration panel.

This project provided us with valuable experience in image recognition and development. It emphasized the importance of accurate ID card recognition in educational institutions. We believe further enhancements will enhance its value and usability.

CHAPTER

6

BIBLIOGRAPHY

[1] Deep Face Recognation - Omkar M. Parkhi , Andra Vedaldi, Androw Zisserman - Departement of engeneering sciences , University Of Oxford.

[2] ORB: an efficient alternative to SIFT or SURF - IEEE International Conference on Computer

[3] Introduction to ORB (Oriented FAST and Rotated BRIEF) - https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

[4] OpenCV Documentation - https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[5] Face Detection with Haar Cascade - Girija Shankar Behera - https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

[6] VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION - Karen Simonyan, Andrew Zisserman -Department of Engineering Science, University of Oxford.

[7] Cosine Similarity Cosine Distance - Anjani Kumar - https://medium.datadriveninvestor.com/cosine-similarity-cosine-distance-6571387f9bf8

[8] Material Classification using Convolutional Neural Networks,repo - Anca Sticlaru

[9] Deep learning report - Aravinda Dharmalingam - https://www.researchgate.net/publication/342233219_Deep_learning_report

[10] Object Recognition Using Deep Learning - Rohini Goel, Avinash Sharma, and Rajiv Kapoor - Department of Computer Science and Engineering, Maharishi Markandeshwar - https://www.researchgate.net/publication/337082426_Object_Recognition_Using_Deep_Learning