الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة سعد دحلب البليدة
SAAD DAHLAB University of BLIDA
كلية التكنولوجيا
Faculty of technology
قسم الالكترونيك
Electronics department

Master's Thesis

PRESENTED BY

BOUABDALLAH Aimene

&

AMRANE Mohammed Ridha

Specialty: Electronics of Embedded systems

THEME

Face Recognition based attendance system

Proposed by:

Mme Bougherira Hamida

Academic year 2021-2022

Dedication

I dedicate this thesis to my dear parents for their
patience, support and sacrifices since my birth,
during my childhood and even in adulthood.

My dear mother. Thank you for your advice, your
sacrifices, your support and your encouragement,
this is your success before it is mine.

To my sisters and all of my family members.
To all my friends, who supported me in the
accomplishment of this humble work

To all my teachers and to all those who have
engaged in this modest works.
To all of those who helped me from near or far.

Acknowledgements

First of all, Praise ALLAH who has endowed us with the wonderful faculty of reasoning.

I want to express my thanks to my supervisor ***Mme Bougherira Hamida*** *To have supported and trusted me during my project with great patience. With her experience in research and teaching, with her advice, I was able to discover the world of scientific research in the field of image processing and facial biometrics techniques.*

My thanks and my deepest gratitude are addressed to my parents who have shown nothing but support and love, and also to all of my family members.

I also thank my colleagues and students each and every one by name who have always supported my efforts.

In the end we thank all the people who participated from near or far in the realization of this work.

Face Recognition Based Attendance System

ملخص

موضوع هذا الماجستير هو نظام أمان بيومتري يمكن استخدامه للتحكم في حضور الطلاب أو دخول الأشخاص إلى أي بيئة (المنزل أو الجامعة أو المصانع).

في هذا المشروع، نستخدم خوارزميات الذكاء الاصطناعي ورؤية الكمبيوتر لإنشاء التعرف على الوجه كنظام أمان بيومتري ، ونقارن بين نماذج التعلم العميق YOLOv5 و SSD وخوارزمية HOG لاكتشاف الوجه. فيما يتعلق بالتعرف على الوجوه، قمنا بمقارنة نموذج شبكة Siamese بنموذج FaceNet مُدرَّب مسبقًا. أظهرت النتائج أن نموذج YOLOv5 يتمتع بدقة أفضل من طراز SSD، لهما نفس السرعة، ولكن جهاز كشف الوجه HOG أسرع بكثير مع دقة مقبولة. حول نماذج التعرف على الوجوه، يعد نموذج FaceNet المدرّب مسبقًا أفضل من نموذج شبكة Siamese من حيث الدقة والسرعة.

أخيرًا، قمنا بتصميم نظام الحضور باستخدام كل من خوارزمية HOG لاكتشاف الوجه ونموذج FaceNet للتعرف على الوجوه.

Abstract

This master's subject is a biometric security system that can be used to control the student attendance or the entrance of individuals to any environment (home, university or factories).

In this project we use artificial intelligence and computer vision algorithms to create a face identification as a biometric security system, we compare the deep learning models YOLOv5, SSD and the HOG algorithm for face detection. Regarding face recognition, we compare the Siamese network model with a Pre-trained FaceNet model. The results show that YOLOv5 model has better accuracy then the SSD model, having both the same speed, but the HOG face detector is way faster with acceptable accuracy. About the face recognition models, the FaceNet pre-trained model is better than the Siamese network model in both accuracy and speed.

Finally, we design the attendance system using both the HOG algorithm for face detection and the FaceNet model for face recognition.

Résumé

Ce sujet de master porte sur un système de sécurité biométrique qui peut être utilisé pour contrôler la présence des étudiants ou l'entrée des individus dans n'importe quel environnement (maison, université ou usines).

Dans ce projet, nous utilisons des algorithmes d'intelligence artificielle et de vision par ordinateur pour créer une identification de visage comme système de sécurité biométrique, nous comparons les modèles d'apprentissage profond YOLOv5, SSD et l'algorithme HOG pour la détection de visage. En ce qui concerne la reconnaissance des visages, nous comparons le modèle de réseau siamese avec un modèle FaceNet pré-entraîné. Les résultats montrent que le modèle YOLOv5 a une meilleure précision que le modèle SSD, tout en ayant la même vitesse, mais le détecteur de visage HOG est beaucoup plus rapide avec une précision acceptable. En ce qui concerne les modèles de reconnaissance des visages, le modèle FaceNet pré-entraîné est meilleur que le modèle du réseau siamese en termes de précision et de vitesse.

Enfin, nous concevons le système de présence en utilisant à la fois l'algorithme HOG pour la détection des visages et le modèle FaceNet pour la reconnaissance des visages.

Contents table

Table of Contents

Figures table

Abbreviations list

- AI: artificial intelligence
- BBOX: bounding box
- CNN: convolution neural network
- DL: deep learning
- FDC: Fisher Discriminant Criterion
- GPU: graphical processing unit
- GUI: graphical user interface
- HOG: histogram of oriented gradient
- LBP: local binary pattern
- LDA: linear discriminant analysis
- ML: machine learning
- MTCNN: multi task cascaded convolutional neural networks
- PCA: principal component analysis
- RAM: random access memory
- RNN: recurrent neural networks
- SSD: single shot detector
- YOLO: you look only once

General introduction

Students’ attendance can take a long time especially with a big number of students, calling for 100 names may take 15 minutes or more, for this problem we need to work on an automatic system to do the job, there are many biometric identification systems for recognizing people and labelling them like fingerprints, face-id, voice-id, hand geometry, retinal scan and many other systems.

In this paper, we’re going to create a face identification-based attendance system. Face-id is easy to use and it takes no time to detect and recognize faces, the face-id can recognize many people in few seconds.

This system can be helpful in controlled environments or workspaces, it can be used in factories or connected environments like smart homes, all you need is to update the database.

This document contains three chapters, the first chapter is an overview about the classical face-id algorithms like Eigenfaces and LBP and also modern algorithms using a deep learning model for face detection as YOLO and FaceNet for face recognition with a brief introduction to artificial intelligence.

The second chapter is experimental research containing all the work we’ve done and comparing the results of many models that we trained to create the final application.

Finally, in the last chapter we’re going to describe the steps to build the application and the graphical user interface to make it easier for non-programmer.

I. Chapter 1: Overview

Chapter I: Overview

I.1 introduction

Many people are familiar with the face detections and recognitions technologies which are used in social media applications like Facebook or Snapchat face changing filters.

Starting, the face detection and recognition are two different tasks (Figure l-1), the face detection algorithm’s main function is finding faces in pictures or live videos.

Facial recognition is a technology of recognizing or verifying a person's identification by their face. People may be identified in pictures, videos, or in real time using facial recognition.

The first face recognition algorithm was created by Woody Bledsoe, Helen Chan Wolf and Charles Bisson in 1964.

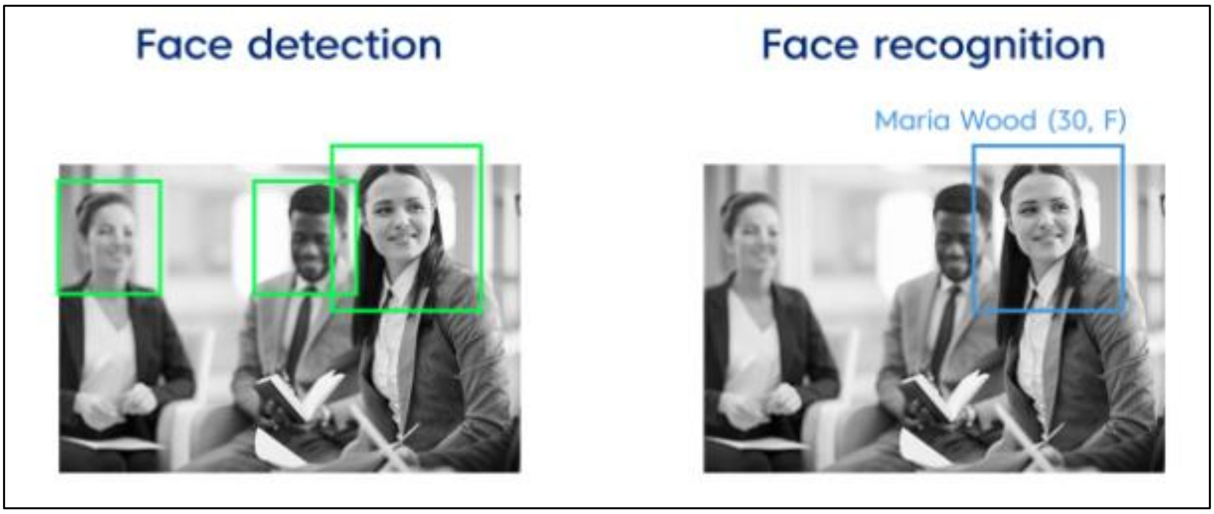


Figure I-1 face detection and recognition

Biometric security includes facial recognition, Voice recognition, fingerprint recognition, and ocular retina identification, those technologies are mostly utilized for security and law enforcement and many other applications.

There are multiple methods for face-id (classical and modern) which pass through different phases as we see on (Figure I-2).

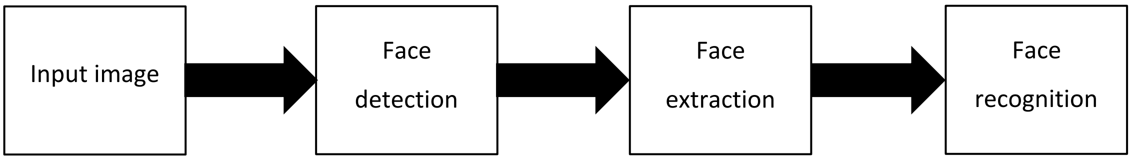


Figure I-2 face recognition steps

I.2 Classical face-id algorithms

I.2.1 Histogram of Oriented Gradients (HOG)

HOG or Histogram of Oriented Gradients is a feature descriptor that allows to extract important information and discard unnecessary information from an image by turning the pixels into a feature vector and it’s used for object detection in computer vision tasks.

I.2.1.1 The HOG follows many steps:

- Preprocessing: resize the image and divide it into blocks.
- Calculating the gradients: calculating the magnitude and the direction of the gradients by converting from cartesian to Polar coordinates:

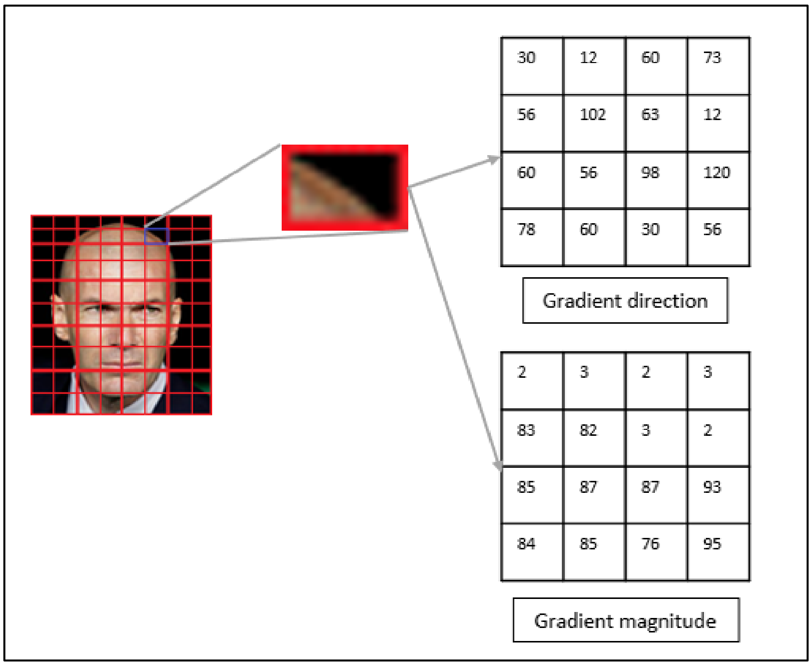
$$g = \sqrt{g_x^2 + g_y^2}$$

Equation I-1 polar coordinates: the length

$$\theta = arc\,tan\frac{g_y}{g_x}$$

Equation I-2 polar coordinates: the angle

- Make a histogram from these gradients: after calculating gradient direction and magnitude like the (Figure I-3) shows, we need to create a histogram for each block.



30	12	60	73
56	102	63	12
60	56	98	120
78	60	30	56

2	3	2	3
83	82	3	2
85	87	87	93
84	85	76	95

Figure I-3 gradient direction and magnitude

- The final step is visualizing the HOG picture.

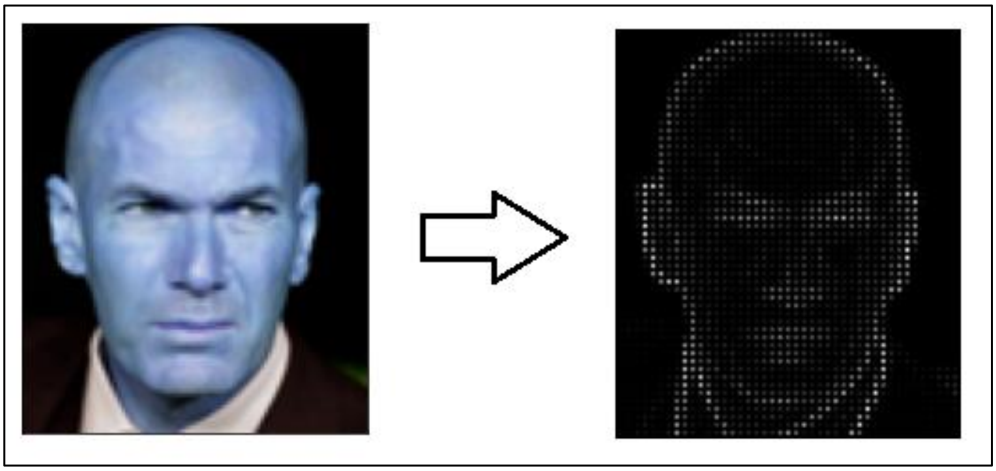


Figure I-4 HOG face

I.2.2 Local Binary Pattern (LBP)

Local Binary Pattern (LBP) is a simple but very efficient texture operator that labels the pixels of an image by evaluating each pixel's neighborhood using a threshold and treating the result as a binary number.

It was first described in 1994 (LBP) and has since proven to be a powerful feature of texture classification. It was also found that when LBP was combined with histograms of oriented gradient (HOG) descriptors, the recognition performance on some datasets improved significantly.

Using LBP with a histogram, we can represent a face image with a simple data vector.

Since LBP is a visual descriptor, it can also be used for face recognition tasks, next we see an LBP example.[1]

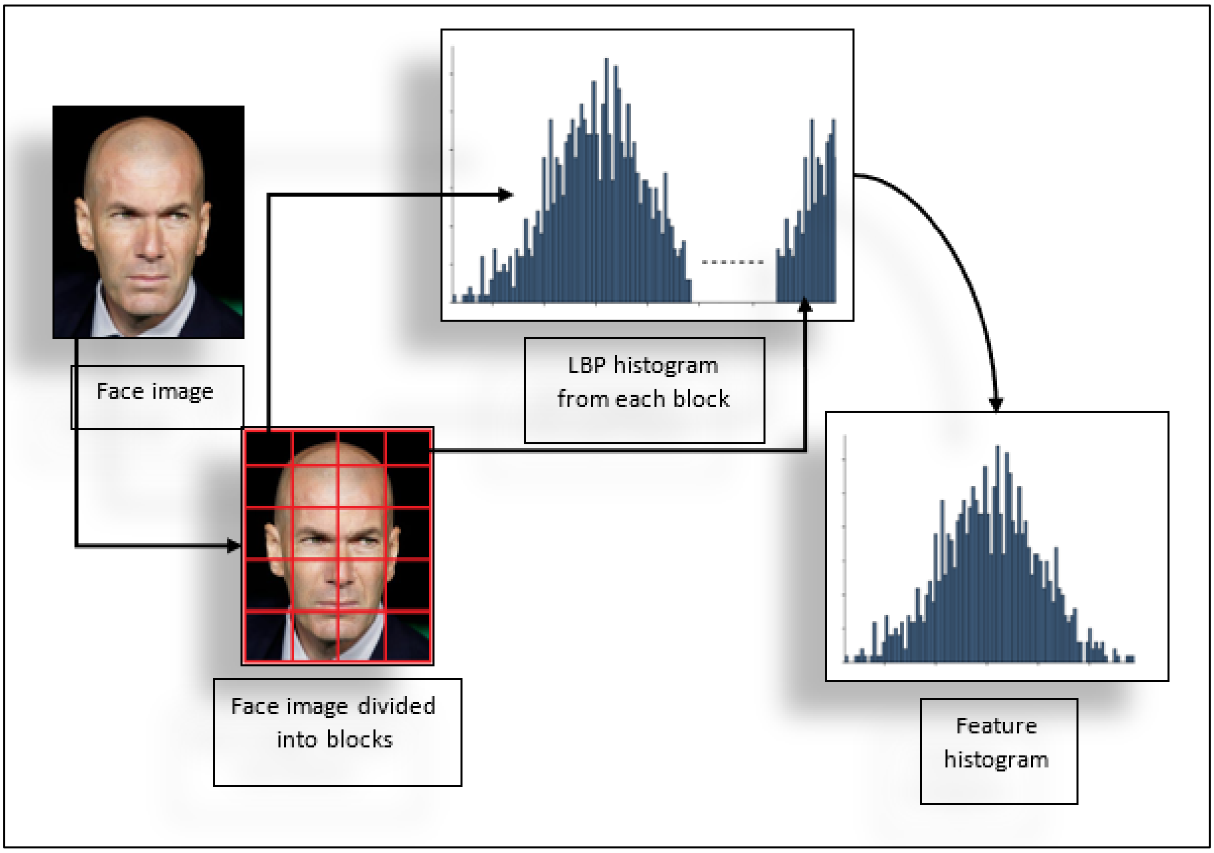


Figure I-5 Local Binary Pattern

I.2.3 Eigenfaces

Eigenfaces are the basic components of a set of feature vectors that are used in face recognition. They were first proposed by **Sirovich** and **Kirby** in 1987 and were then used by **Matthew Turk** and **Alex Pentland** in their face recognition project. An image batch is converted into eigenfaces, which are feature vectors, before the actual training set is made. During face recognition, the system projects the new image to the eigenface subspace and determines its identity by examining the position of the projection points in the subspace as well as the length of the lines projecting out them.

Eigenfaces method of spatial transformation uses Principal Component Analysis (PCA), which takes the covariance matrix of every face in the training set and breaks it down to find the eigenvectors (also known as eigenfaces). This method takes images that are in the same category and clusters them together, and separates images that are in different categories further away. While it is difficult to cut images that are in the original pixel space with simple lines or edges, this method of spatial transformation can separate the images well.[2]

Figure I-6 Eigenfaces

I.2.4 Fisherface algorithm

The PCA method takes the data and projects it in the direction that has the biggest variation. The method is not as good at separating the different classes, but it is good at projecting the data into a direction that has the largest variation.

In 1991, **Cheng et al**. introduced Linear Discriminant Analysis (LDA) method for face recognition.

This method tries to find a linear subspace that maximizes the separation of two pattern classes.

In 1997, **Belhumer** applied the Fisher Discriminant Criterion to face classification and proposed the Fisherface method. The method was based on Linear Discriminant Analysis, LDA attempts to find a line of separation that will best separate two groups, which can be applied to face recognition. [3]

I.2.5 Viola-Jones

Paul Viola and **Michael Jones** are the original names for the algorithm named after. In 2001, they wrote a paper proposing the method, titled "Rapid Object Detection using a Boosted Cascade of Simple Features." Despite being an older method, Viola-Jones is powerful, and is used in real-time face detection. The algorithm is very slow to train, but can detect faces very quickly in real-time.

This algorithm works by examining smaller sections of the original image and looking for specific face attributes in each section. The algorithm has to check many different sizes and positions on the image, because it could contain many different sized faces. Viola and Jones used Haar-like features in this algorithm.

The four main steps of the Viola Jones algorithm are:

1. Selecting Haar-like features.
2. Creating an integral image.
3. Running AdaBoost training.
4. Creating classifier cascades.

I.3 The artificial intelligence

I.3.1 What is artificial intelligence

Computers or robots that can perform tasks normally associated with intelligent beings have been around since the 1940s, when the first digital computer was created. These computers have been shown to be very proficient at carrying out very complicated tasks, such as playing chess or proving mathematical theorems. Despite the fact that computer memory and processing speed have improved significantly since their creation, there are still no programs that can match a human's ability to perform general knowledge tasks or be flexible across a wider range of domains.

Some programs have been able to perform the same tasks that professionals and experts can do, using artificial intelligence. These applications include recognizing handwriting, voice and medical diagnosis.

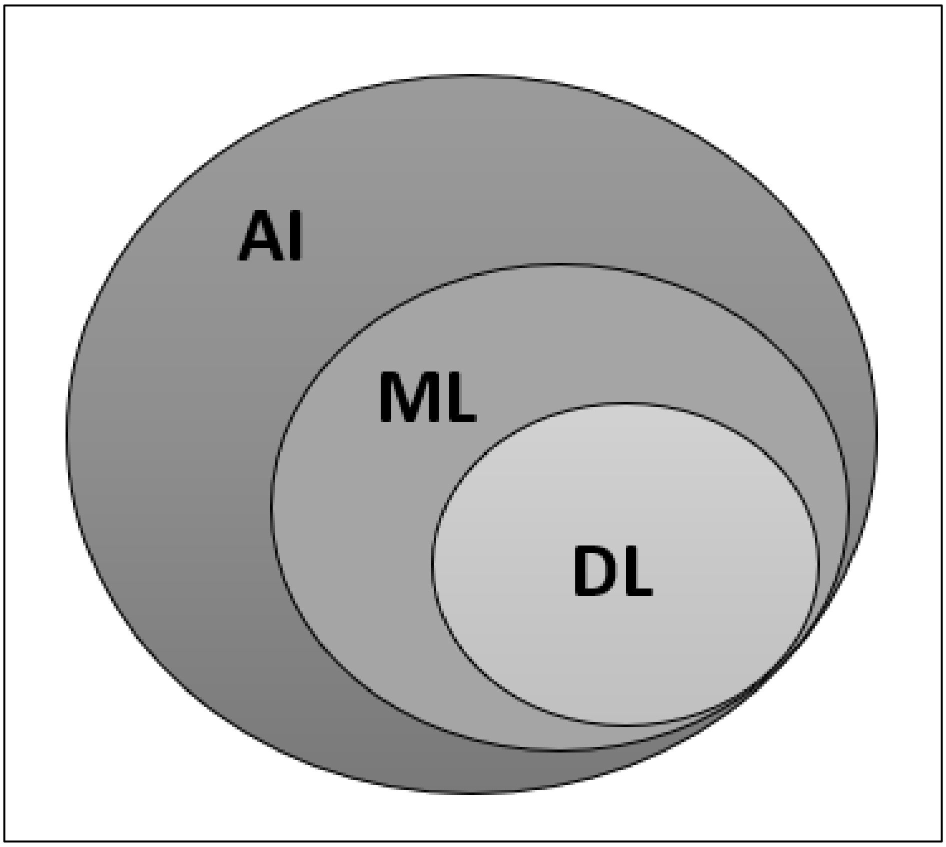


Figure I-7 AI-ML-DL

I.3.2 The evolution of Artificial intelligence

In 1950, Alan Turing suggested that computers could simulate intelligent behaviour and critical thinking. His idea became known as the Turing Test, which John McCarthy later referred to as artificial intelligence. AI is the science and engineering of creating intelligent machines, and has grown over the years to include more complex codes that function similar to a human brain. There are many branches of AI similar to there being different specialties in medicine, including machine learning (ML), deep learning (DL), and computer vision.

In the 1980s computers were able to learn and use their knowledge thanks to two sources: more algorithms, and more funding.

John Hopfield and David Rumelhart made a deep learning technique popular, which allowed computers to learn through experience. Edward Feigenbaum used expert systems to make computers mimic the decision and making process of a human expert.

Expert systems were used in many industries, and could give advice to non-experts based on what they had learned about every possible situation. The program would ask an expert how to respond to a situation, and then once the expert had learned the responses for all situations, the program could give advice to non-experts.

In the 1990s and 2000s, many of the seminal goals of artificial intelligence were achieved. In 1997, chess world champion and grandmaster Gary Kasparov was defeated by IBM's chess computer program Deep Blue.

I.3.3 Machine learning and deep learning

ML is a subset of artificial intelligence that focuses on enabling computers to perform tasks without explicit programming, on the other hand the DL is a subset of machine learning based on artificial neutral networks, the machine learning can’t solve very complex problems as computer vision tasks. Also, Deep Learning supports scalability, supervised and unsupervised learning, More differences in the (Table l-1)

Deep learning	Machine learning
Require big data	train on lesser data
Provide high accuracy	Provide lesser accuracy
Take long time	Take lesser time
The output can be text, sound …	The output is always a numerical value

Table I-1 ML vs DL

Machine Learning is a method of statistical learning where each instance in a dataset is described by a set of features or attributes. In contrast, the term “Deep Learning” is a method of statistical learning that extracts features or attributes from raw data. So, in deep learning before getting to the classification the data has to go through extraction features (hidden layers) (Figure I-8). [4]

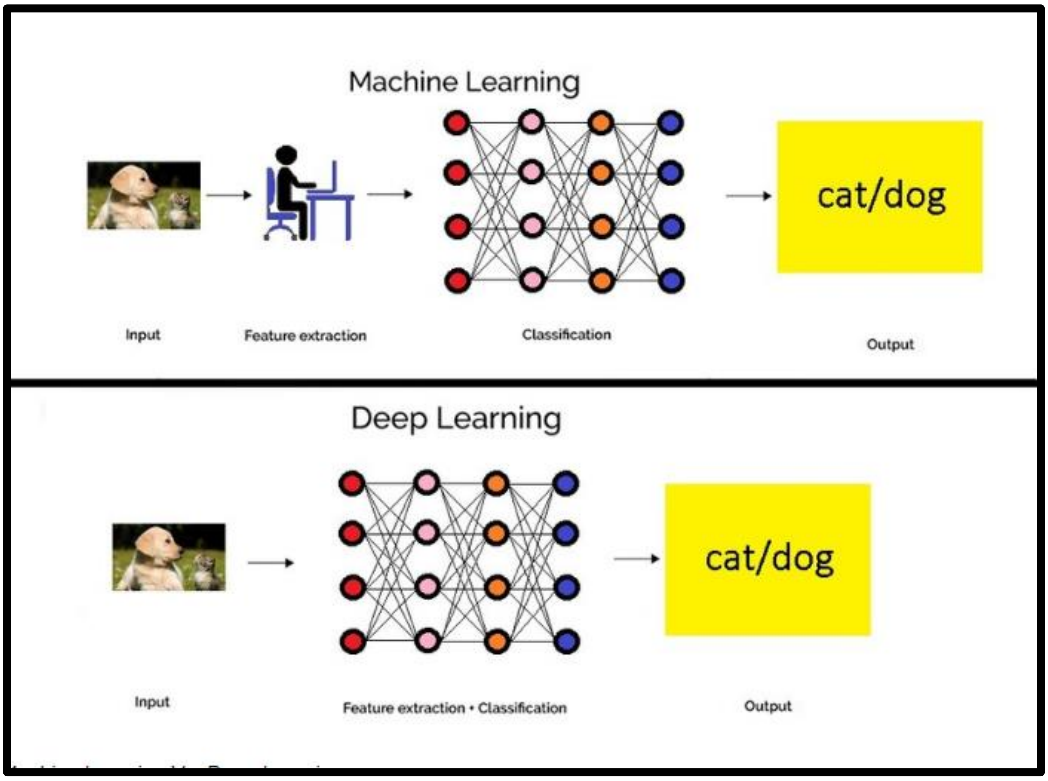


Figure I-8 ML and DL neural networks [4]

I.3.4 The Deep Neural Network

Deep neural nets are neural networks that have multiple hidden layers. There are many types of deep neural net like (CNN, RNN).

A Convolutional Neural Network, or CNN, is a type of deep neural network that is utilized for image and object recognition and classification. CNNs are being used for many different tasks, including video analysis, localizing and separating sections of images, and image processing, there are a very large number of CNN model architectures of different types.

- AlexNet
- ResNet

- GoogleNet
- MobileNetV1
- Wide ResNet
- VGG
- PolyNet

I.4 Modern Face-Id algorithms

I.4.1 YOLO

The object detection networks like YOLO and SSD have become popular and very useful for face detection and many other applications, because they have good performance, and doesn't have a long waiting time or slow speed. Two-stage object detection systems are excellent, but take a while to load, and are not very fast.

YOLO uses a single neural network to detect an object, and does all the necessary steps to detect the object, rather than just classification. YOLO can do real-time detection, and has excellent performance.

The YOLO algorithm was first created by the original author, and then improved by three more versions: YOLOv1, YOLOv2, and YOLOv3. The version YOLOv3 is a big improvement from its previous versions, having both speed and performance increases by utilizing multi-scale features (FPN), a better neural network backbone (Darknet53), and replacing the SoftMax loss with the binary cross-entropy loss in the classification loss. The YOLO algorithm will be further improved over the next five years, developing into 5 more versions with many more improvements from the object detection community.

In 2020, after the original YOLO authors withdrawn from the research field, YOLOv4 was released by a different research team. The team explored a lot of options in almost all aspects of the YOLOv3 algorithm.

Another research team came out with the YOLOv5 one month after YOLOv4. This version of the algorithm does not have many new innovations, and the team who created it didn't write a paper about it. YOLOv5 is a version of the YOLO model that is significantly smaller in size, faster, and performs similar to YOLOv4. There have been some controversies around calling it version 5, but it is welcome in the object detection community due to being completely implemented in Python (Pytorch). [5]

In 2022 two other versions were released, v6 and v7 by another research groups.

I.4.2 MTCNN

Multi-Task Cascaded Convolutional Neural Networks In 2016, **Zhang et al** published a neural network that detects faces and facial landmarks on images.

MTCNN is a Cascaded Network of three CNNs:

The first stage takes as input an image pyramid made up of differently scaled copies of the input image.

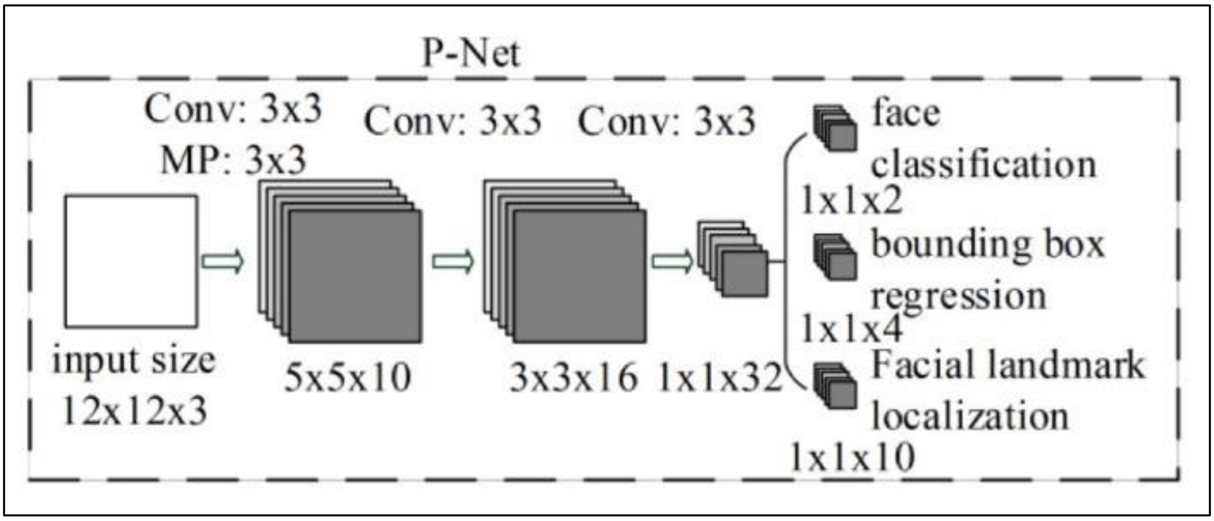


Figure I-9 stage 1 in MTCNN

The second stage is a CNN Refine Network (R-Net). This network takes the boxes that were separated in the first stage and combines candidates that were overlapping, using non-maximum suppression.

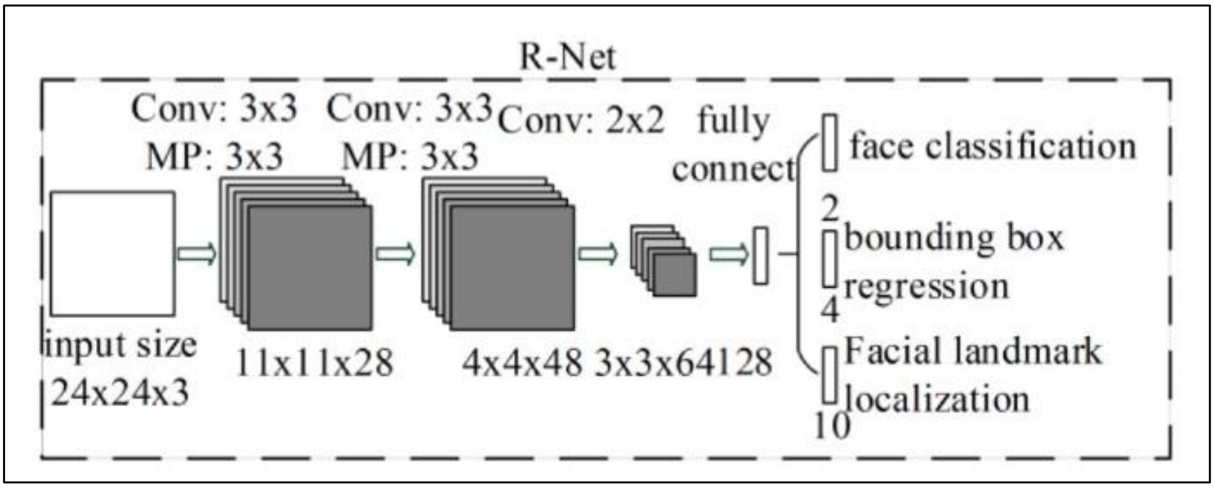


Figure I-10 stage 2 in MTCNN

In the third stage, the Output Network does the same things as R-Net, and adds a 5-point landmark of eyes, nose, and mouth in the final box containing the detected face.[6]

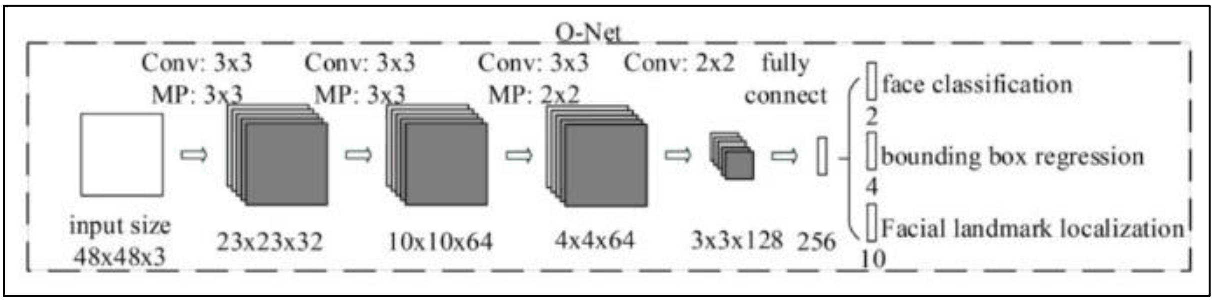


Figure I-11 stage 3 in MTCNN

I.4.3 FaceNet

In 2015, Google researchers **Schroff et al** used a deep neural network called FaceNet that takes a picture of peoples' faces and extract features from them.

FaceNet takes an image of a face as input and outputs a vector of 128 numbers representing key features of the face. In machine learning, this vector is called an embedding. The key information from the image is contained in this vector. The faces of people that have similar pictures are also assumed to be similar themselves.

Mapping high-dimensional data (like images) into low-dimensional representations (embeddings).

To determine if a face is on an unseen image, one possibility would be to calculate the images embedding, and then compare it to the embeddings of known people. If the images face is close enough to the face of person A, we would say that the image is of person A. [7]

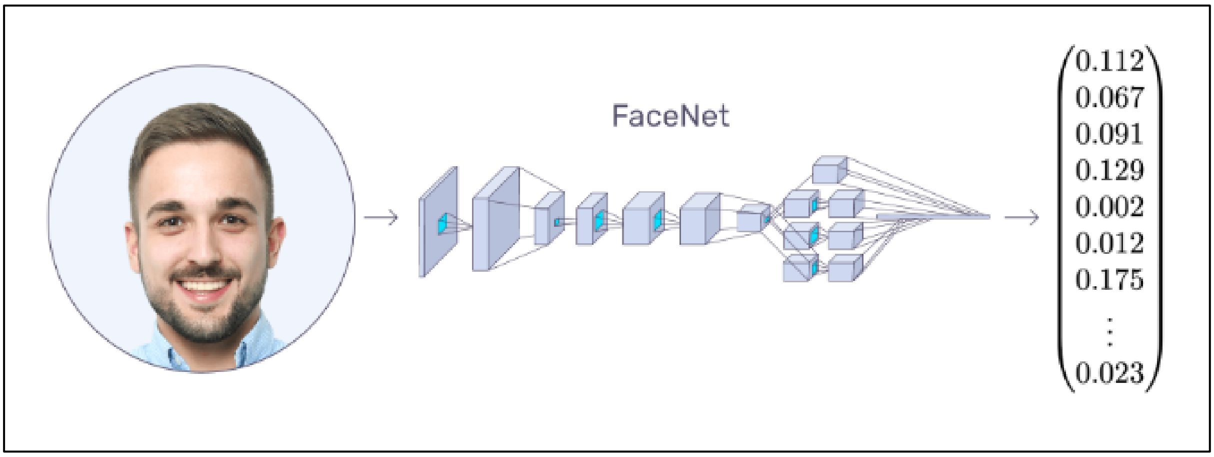


Figure I-12 FaceNet

I.5 Conclusion

In this chapter we had an overview about Face detection and recognition technologies and their improvement in the last 50 years, from classical algorithms like LBP and Eigenfaces to complicated deep learning models as YOLO, MTCNN and FaceNet.

These applications are in a remarkable development due to the revolution of artificial intelligence these days, especially in the domain of computer vision, in the next chapter we're going to expose our steps and methods of creating deep learning models for face identification, and implement it to serve our main case which is the face recognition based attendance system.

II. Chapter 02: Experimentations

Chapter II: Experimentations

II.1 Introduction

In this chapter we’re going to present step by step the experimentations we’ve done which are the face detection and recognition models, we’re going to show the training and the testing of all the models we did create and choose the best methods for our final application.

we have seen many algorithms used for face detection and recognition in the previous chapter, in this chapter we’re exposing the models we trained and compare them with each other.

about the algorithms proposed in this chapter are:

- SSD and YOLOv5 for face detection.
- Siamese network and FaceNet models for face recognition.

II.2 Work environment

II.2.1 Setup configuration

- Laptop DELL Latitude E5440.
- Processor Intel Core i7 4600U @ 2.60 GHZ.
- Graphical Card GT-720M.
- Installed memory (RAM) 6.00 GB.
- Windows 10 pro 64 bit.
- Jupyter Notebook.
- Python 3.7.4.

II.2.2 Python

II.2.2.1 What is python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its built-in high-level data structures combined with dynamic typing and dynamic binding make it very attractive for rapid application development and for use as a scripting or glue language to bring together existing components. Python's easy-to-learn syntax emphasizes readability, thus reducing program maintenance costs. Python supports modules and packages, which promote modularity of programs and code reuse. The Python interpreter and extensive standard library are freely available on all major platforms in source or binary format and can be redistributed free of charge.

II.2.2.2 Why python

AI projects are different from basic software projects, the skills required for AI-based projects, to achieve your AI ambitions, we must use a programming language that is stable, flexible, and provides tools. with an enormous number of libraries Python provides all of this, which is why we can find too many AI-Based project using Python today.

The advantages that make Python the best choice for machine learning and AI it's the access to great AI and machine learning (ML) libraries and frameworks as TensorFlow, Keras, NumPy and OpenCV.

II.2.3 Google Colab

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.[8]

- GPU Tesla T4.
- RAM 13.6 GB.
- Python 3.7.13.

II.2.4 Jupyter notebook

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience.

Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.[9]

II.3 The libraries used

II.3.1 TensorFlow

Google's open-source TensorFlow framework is used to build and run various kinds of machine learning and deep learning applications.

TensorFlow provides a collection of workflows to develop and train models using Python or JavaScript, and to easily deploy in the cloud, on-prem, in the browser, or on-device no matter what language you use.

TensorFlow offers multiple levels of abstraction so we can choose the right one for our needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

Since we need more flexibility, eager execution allows for immediate iteration and intuitive debugging. For large ML training tasks, use the Distribution Strategy API for distributed training on different hardware configurations without changing the model definition. [10]

II.3.2 OpenCV (cv2)

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. [11]

II.3.3 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easier and hard things possible.

II.3.4 OS

This module provides a portable way of using operating system dependent functionality, as creating folders, files, opening, copying, cutting, pasting …

II.3.5 LabelMe

LabelMe is an open-source annotation. It was written in Python to support manual image polygonal annotation for object detection, classification, and segmentation. LabelMe lets you create various shapes, including polygons, circles, rectangles, lines, line strips, and points. You can save your labels as JSON files directly from the app. The LabelMe repository offers a Python script to help you convert annotations to PASCAL VOL. Other formats, such as YOLO and COCO, are not supported.[12]

II.3.6 NumPy

NumPy is the basic scientific computing package in Python. It is a Python library that provides multidimensional array objects, various derived objects such as masked arrays and matrices, and a set of routines for performing quick operations on arrays, including math, logic, shape manipulation, sorting, selection, I /O, discrete Fourier transform, basic linear algebra, basic statistical operations, stochastic simulation, and more.[13]

II.3.7 Albumentations

Albumentations is a fast and flexible image extension library. This library is widely used in the industry, deep learning research, machine learning competitions, and open-source projects. Albumentations is written in Python and licensed under the MIT license.[14]

II.4 Face Detection

in this segment we're showing the training steps and the testing of the SSD (Single shot detector) model and the YOLOv5(You Only Look Once) model.

II.4.1 SSD for face detection

The creation of the custom SSD model goes through many steps (Flowchart 1).

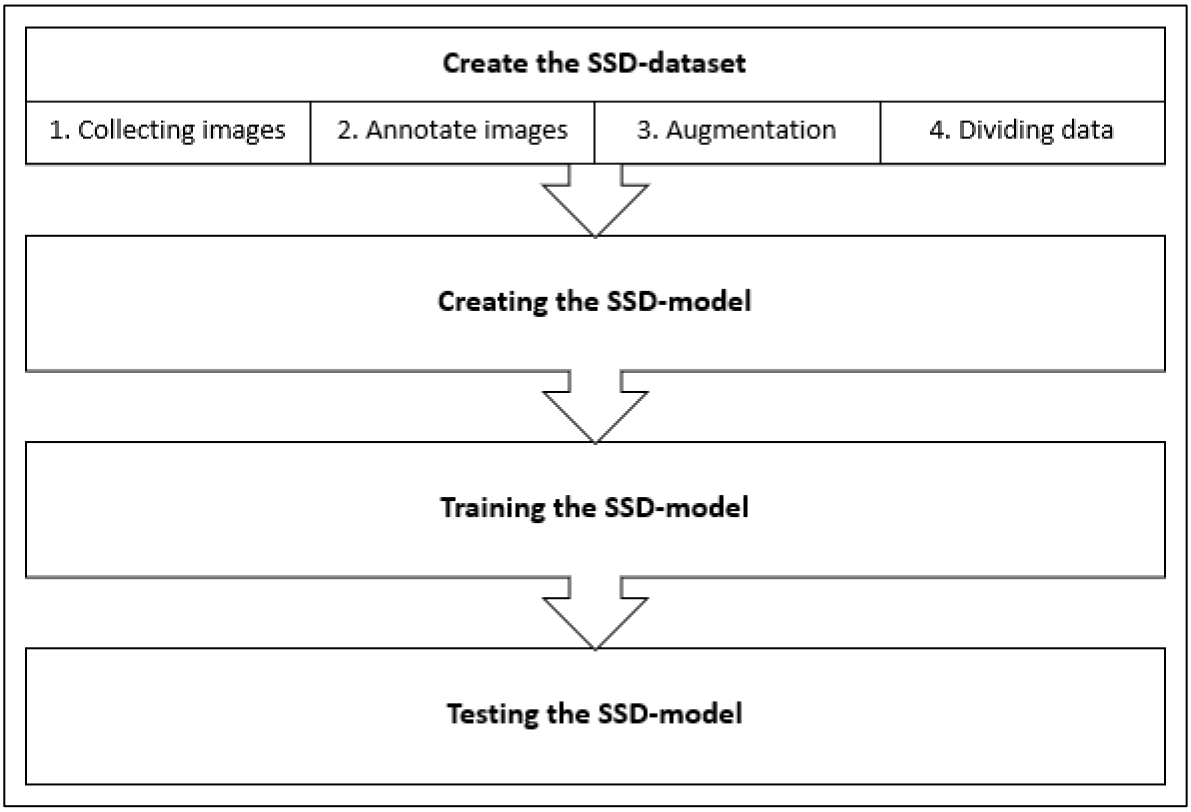


Figure II-1 Steps to create the SSD model

II.4.1.1Creating the SSD-Dataset

II.4.1.1.1 Collecting images

Collecting data is the first step of our actual work, we can get a very large dataset from the internet, or we can just take pictures using a smart phone camera or computer webcam.

To collect data, we’re going to take many pictures using a simple code with the help of the OpenCV library and name those pictures and save them.

if we had to collect images data from different sources it’s very important that we have the same images size so we need to resize them.

II.4.1.1.2 Images collector program

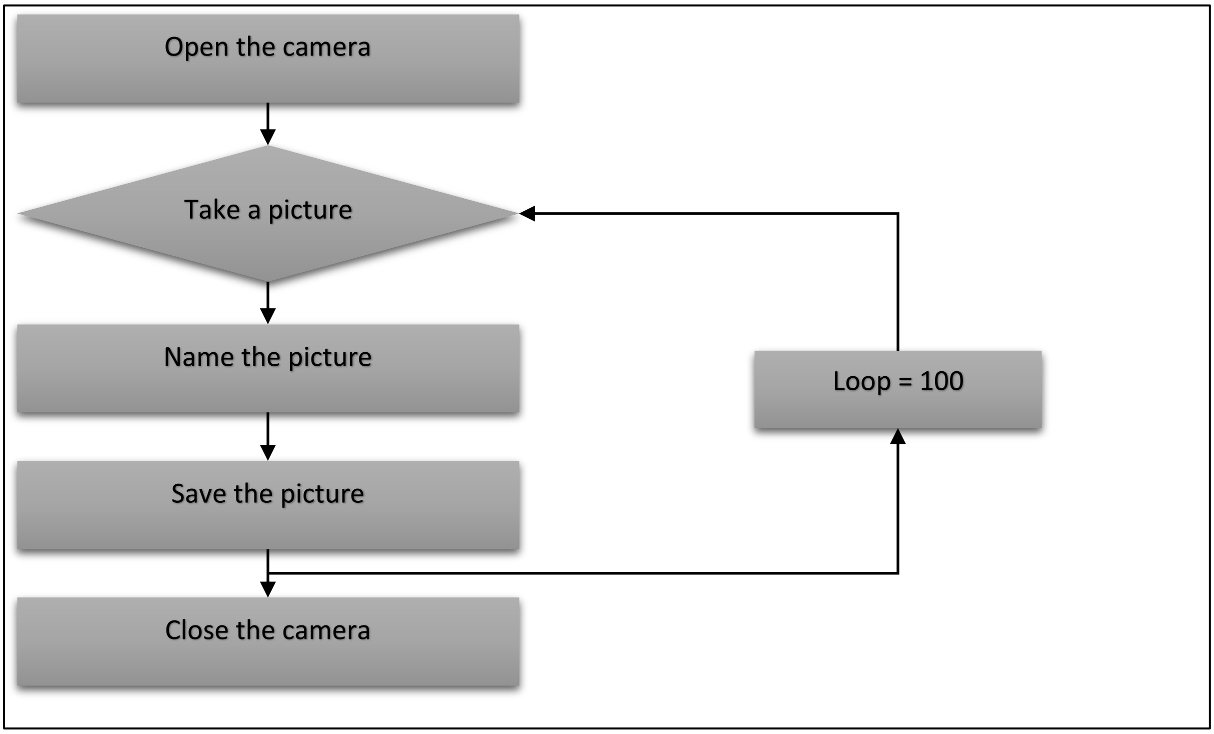
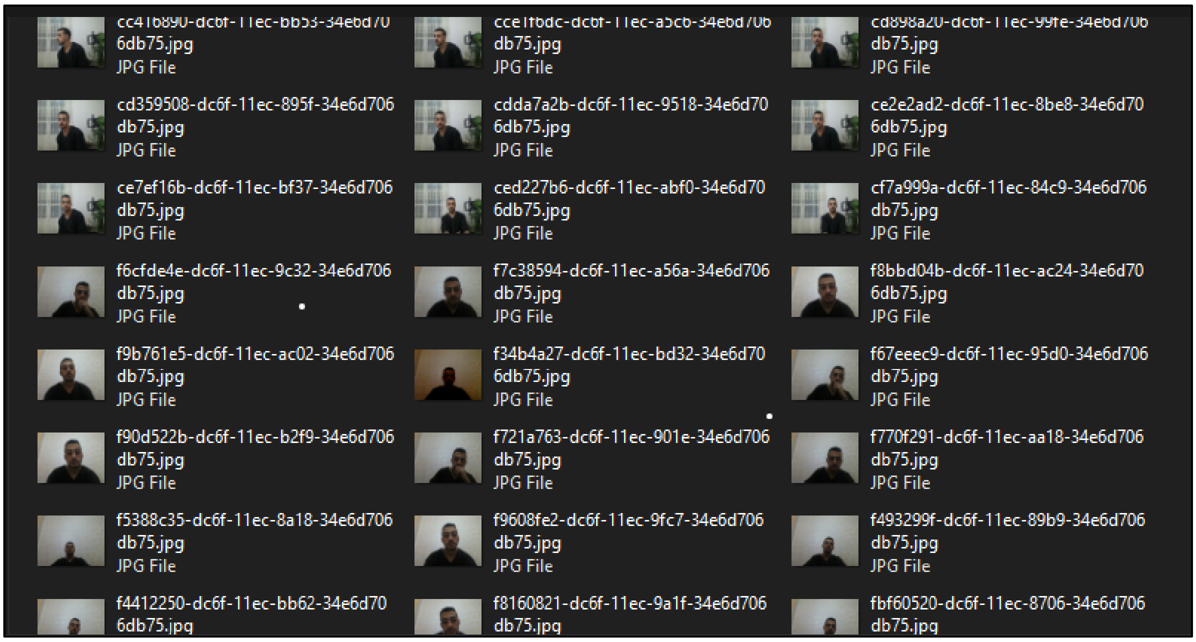


Figure II-2 images collector flowchart

After using the program (Figure II-1) we’ll get 100 pictures of ours. We can add more random human images from other sources, but as we said the images need to have the same size, so using OpenCV or TensorFlow we can resize the images and resave them with the others (Figure ll-3).



cc416890-dc6f-11ec-bb53-34e6d70 6db75.jpg
JPG File

cce1f6dc-dc6f-11ec-a5c6-34e6d706 db75.jpg
JPG File

cd898a20-dc6f-11ec-99fe-34e6d706 db75.jpg
JPG File

cd359508-dc6f-11ec-895f-34e6d706 db75.jpg
JPG File

cdda7a2b-dc6f-11ec-9518-34e6d70 6db75.jpg
JPG File

ce2e2ad2-dc6f-11ec-8be8-34e6d70 6db75.jpg
JPG File

ce7ef16b-dc6f-11ec-bf37-34e6d706 db75.jpg
JPG File

ced227b6-dc6f-11ec-abf0-34e6d70 6db75.jpg
JPG File

cf7a999a-dc6f-11ec-84c9-34e6d706 db75.jpg
JPG File

f6cfde4e-dc6f-11ec-9c32-34e6d706 db75.jpg
JPG File

f7c38594-dc6f-11ec-a56a-34e6d706 db75.jpg
JPG File

f8bbd04b-dc6f-11ec-ac24-34e6d70 6db75.jpg
JPG File

f9b761e5-dc6f-11ec-ac02-34e6d706 db75.jpg
JPG File

f34b4a27-dc6f-11ec-bd32-34e6d70 6db75.jpg
JPG File

f67eeec9-dc6f-11ec-95d0-34e6d706 db75.jpg
JPG File

f90d522b-dc6f-11ec-b2f9-34e6d706 db75.jpg
JPG File

f721a763-dc6f-11ec-901e-34e6d706 db75.jpg
JPG File

f770f291-dc6f-11ec-aa18-34e6d706 db75.jpg
JPG File

f5388c35-dc6f-11ec-8a18-34e6d706 db75.jpg
JPG File

f9608fe2-dc6f-11ec-9fc7-34e6d706 db75.jpg
JPG File

f493299f-dc6f-11ec-89b9-34e6d706 db75.jpg
JPG File

f4412250-dc6f-11ec-bb62-34e6d70 6db75.jpg

f8160821-dc6f-11ec-9a1f-34e6d706 db75.jpg

fbf60520-dc6f-11ec-8706-34e6d706 db75.jpg

Figure II-3 collected images

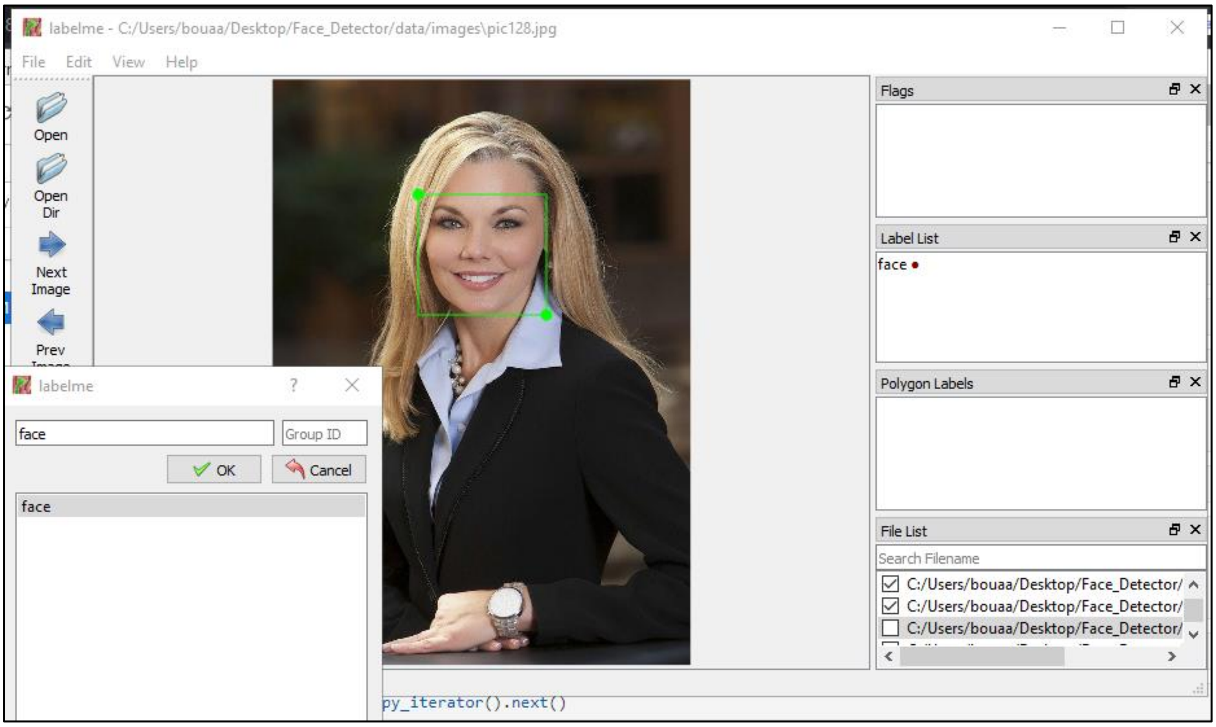
II.4.1.1.3 The annotation

The annotation is a programming way to define the location of the face in the image for the program by selecting the face within a geometric shape (a rectangle) and save its coordinates, to do that we use LabelMe application.

II.4.1.1.4 The annotation steps

After installing LabelMe and running it from the python editor or CMD, following the steps below to create a label file for every image with the same image name and that will happen automatically by LabelMe.

1. Create a folder to save the labels inside it.
2. Open LabelMe by calling it.
3. Click to Open Dir.
4. Select the images folder.
5. After the image displayed on the screen click on File.
6. Then click to change Output Dir and select the labels folder.
7. Then save automatically.
8. Click right in the mouse and choose square.
9. Finally, select the face and click D to move to the next picture (Figure II-4).


labelme - C:/Users/bouaa/Desktop/Face_Detector/data/images\pic128.jpg

face

Label List: face ●

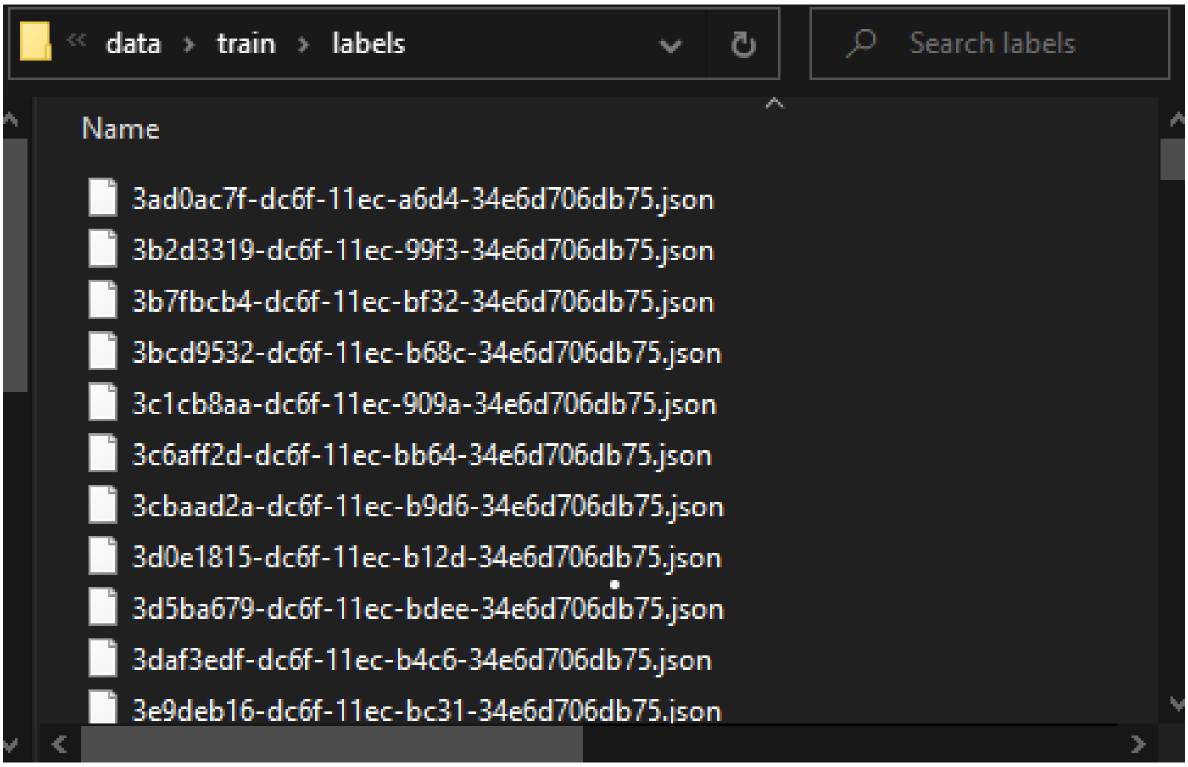
☑ C:/Users/bouaa/Desktop/Face_Detector/
☑ C:/Users/bouaa/Desktop/Face_Detector/
□ C:/Users/bouaa/Desktop/Face_Detector/

py_iterator().next()

Figure II-4 the annotation process

II.4.1.1.5 The annotation results

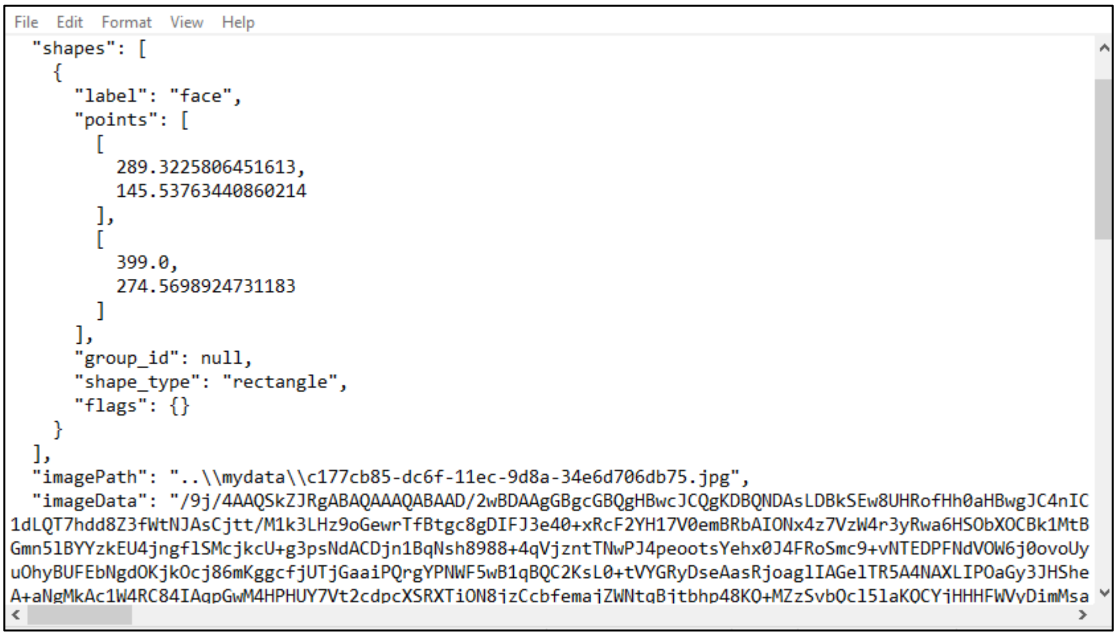
After annotating all the pictures, we can see that the labels files (Figure Il-5) saved at the extension (json) so we need to call the (json) library to open those files in the python notebook.



```
3ad0ac7f-dc6f-11ec-a6d4-34e6d706db75.json
3b2d3319-dc6f-11ec-99f3-34e6d706db75.json
3b7fbcb4-dc6f-11ec-bf32-34e6d706db75.json
3bcd9532-dc6f-11ec-b68c-34e6d706db75.json
3c1cb8aa-dc6f-11ec-909a-34e6d706db75.json
3c6aff2d-dc6f-11ec-bb64-34e6d706db75.json
3cbaad2a-dc6f-11ec-b9d6-34e6d706db75.json
3d0e1815-dc6f-11ec-b12d-34e6d706db75.json
3d5ba679-dc6f-11ec-bdee-34e6d706db75.json
3daf3edf-dc6f-11ec-b4c6-34e6d706db75.json
3e9deb16-dc6f-11ec-bc31-34e6d706db75.json
```

Figure II-5 labels files

The label file defines the label name (face), its location in the picture and the image data as we have in the example (Figure II-6), but we don't need all of those data in the label file, so we're going to remove all unnecessary information (Figure II-7).



```
  "shapes": [
    {
      "label": "face",
      "points": [
        [
          289.3225806451613,
          145.53763440860214
        ],
        [
          399.0,
          274.5698924731183
        ]
      ],
      "group_id": null,
      "shape_type": "rectangle",
      "flags": {}
    }
  ],
  "imagePath": "..\\mydata\\c177cb85-dc6f-11ec-9d8a-34e6d706db75.jpg",
  "imageData": "/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRofHh0aHBwgJC4nIC
1dLQT7hdd8Z3fWtNJAsCjtt/M1k3LHz9oGewrTfBtgc8gDIFJ3e40+xRcF2YH17V0emBRbAIONx4z7VzW4r3yRwa6HSObXOCBk1MtB
Gmn5lBYYzkEU4jngflSMcjkcU+g3psNdACDjn1BqNsh8988+4qVjzntTNwPJ4peotsYehx0J4FRoSmc9+vNTEDPFNdVOW6j0ovoUy
uOhyBUFEbNgdOKjkOcj86mKggcfjUTjGaaiPQrgYPNWF5wB1qBQC2KsL0+tVYGRyDseAasRjoaglIAGelIPOaGy3JHShe
A+aNgMkAc1W4RC84IAqpGwM4HPHUY7Vt2cdpcXSRXTiON8jzCcbfemajZWNtqBjtbhp48lD48OCYjHHHFWVyDimMsa
```

Figure II-6 the annotation result

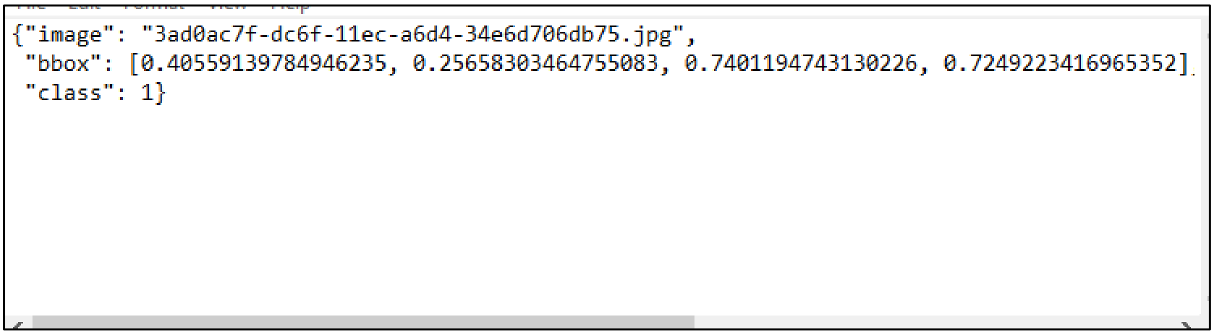
II.4.1.1.6 Data augmentation

From the collecting program we got 100 pictures and for training data it not enough to get good results, in addition to that, annotating thousands of images will take a really long time, for that problem Albumentations library offer a solution to augment our dataset.

Image augmentation is the process of creating new training samples from existing training samples. To create a new pattern, slightly modify the original image. For example, we can make the new image a little brighter; we can cut out a piece of the original image; we can create a new image by mirroring the original image… etc. and finally we got a dataset 50 times larger (5000 image).

And for labels we create a tuple for every label with just few important data

[“image name”,” box localization”, “the class (face or no face)”]



```
{"image": "3ad0ac7f-dc6f-11ec-a6d4-34e6d706db75.jpg",
 "bbox": [0.40559139784946235, 0.25658303464755083, 0.7401194743130226, 0.7249223416965352],
 "class": 1}
```

Figure II-7 the final labels files

II.4.1.1.7 Dividing data

The last step of dataset creation is division of the data and its moving to these new folders (Figure II-8)

- Moving the largest part of the data to the train data folder (65% to 70%).
- Then moving (15% or 25%) to the testing data folder.
- Finally putting the rest in the validation data folder.

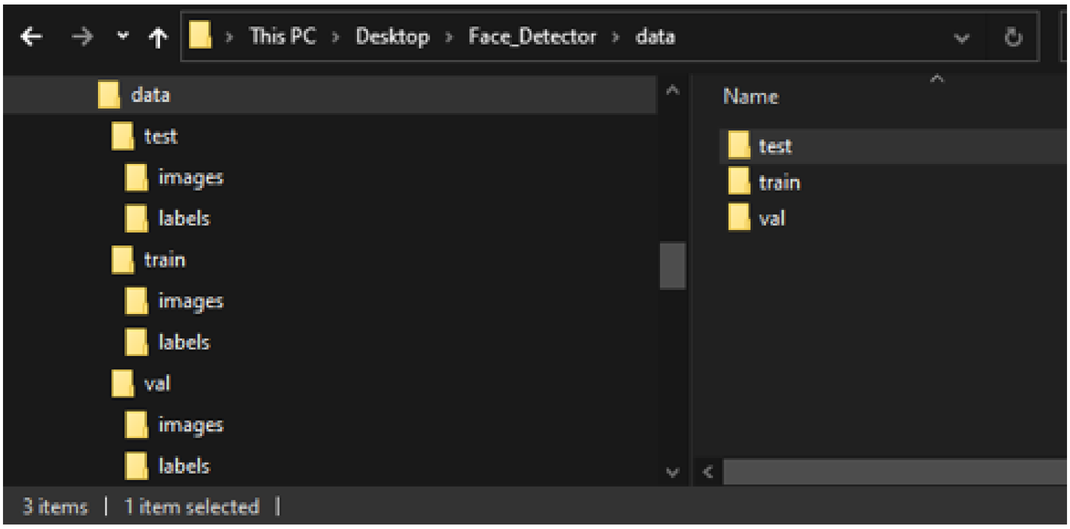


Figure II-8 the data folders 01

II.4.1.2 Creating the SSD-model

II.4.1.2.1 Single shot Detector (SSD)

SSD is designed for real-time object detection. the SSD model detects objects in a single shot, which means works faster and saves a lot of time, SSDs speed up the process by eliminating the need for zone-advised networks. To compensate for the drop in accuracy, SSD has some improvements which make the SSD model have a high accuracy in its detection even on input images of low resolutions.

The SSD model is made up of 2 parts (Backbone and head), The Backbone model is the feature map extractor of the input image. The head is a couple of convolution layers that detect the object and give us the bounding box (Figure II-9).

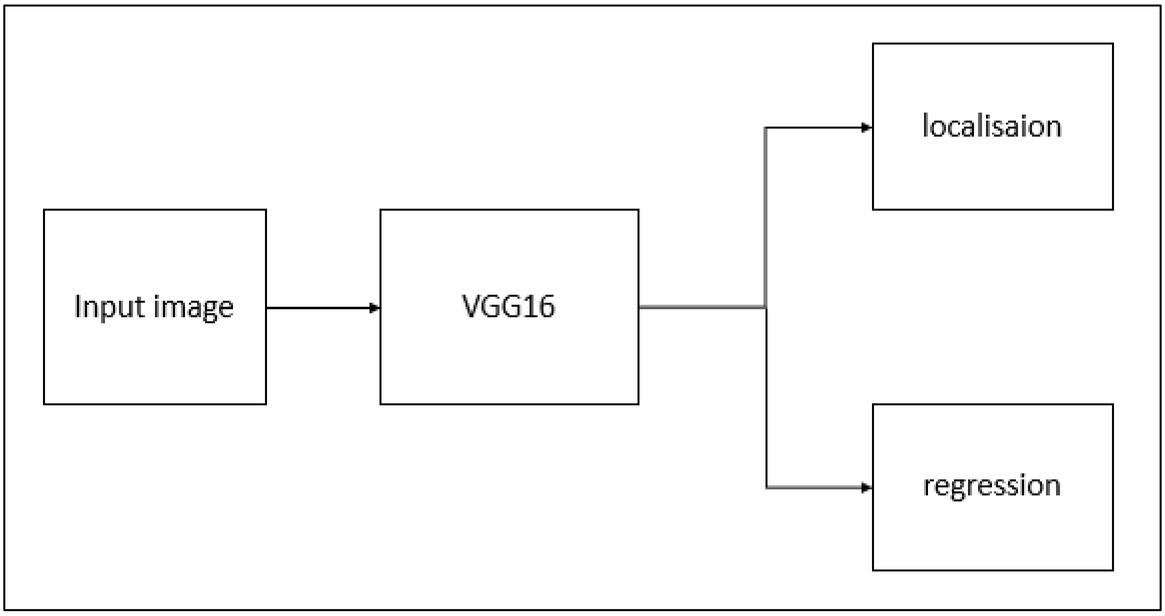


Figure II-9 SSD Architecture

Localization: localization is four outputs for the bounding box.

Regression: regression is one output for classifying (face, no-face).

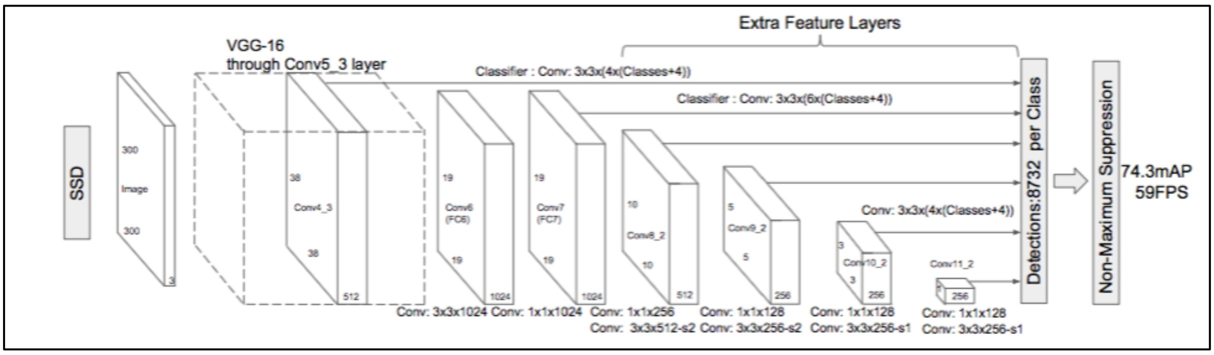


Figure II-10 SSD multi box architecture

II.4.1.2.2 The VGG16

VGG16 is the CNN (Convolutional Neural Network) architecture that won the 2014 ILSVR (ImageNet) competition. It is still considered one of the preeminent Vision model architectures today. The most unique thing about VGG16 is that instead of using a lot of hyperparameters, they focus on convolutional layers of 3x3 filters in step 1, and always use the same padding and Maxpool layers of 2x2 filters in step 2. It follows this order of convolution and max pooling layers throughout the architecture. Finally, it has 2 FCs (fully connected layers) followed by a soft max of the output. The 16 in VGG16 means that it has 16 layers of weights. This network is a fairly large network with about 138 million (approximately) parameters (Figure Il-11). [15]

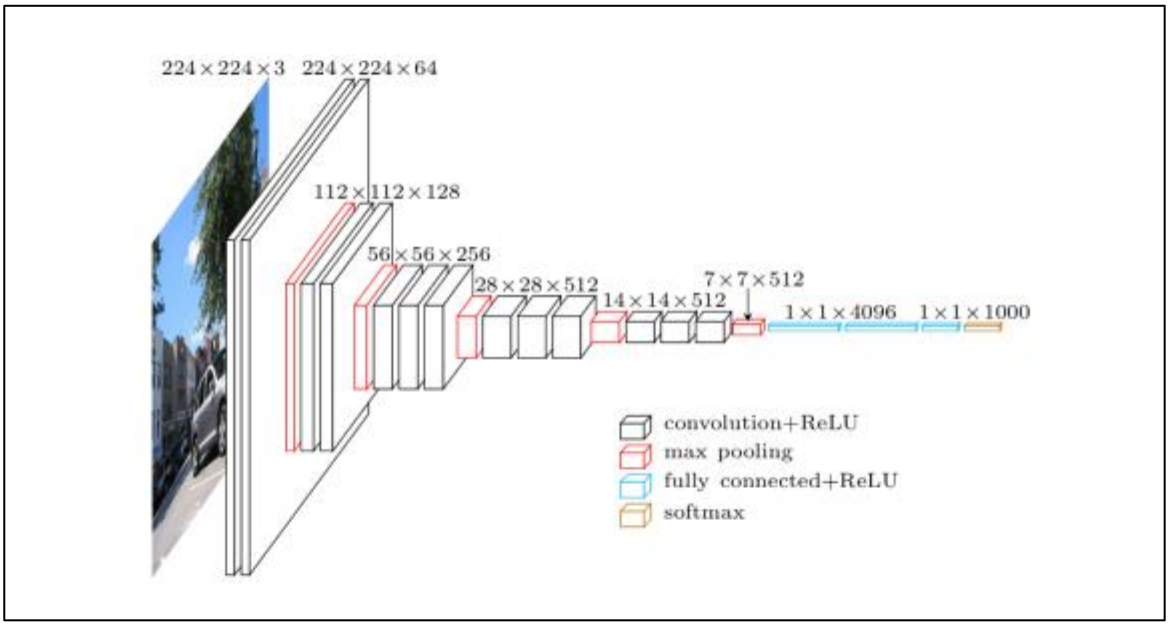


Figure II-11 The VGG16 architecture

II.4.1.2.3 Loading data to TensorFlow dataset

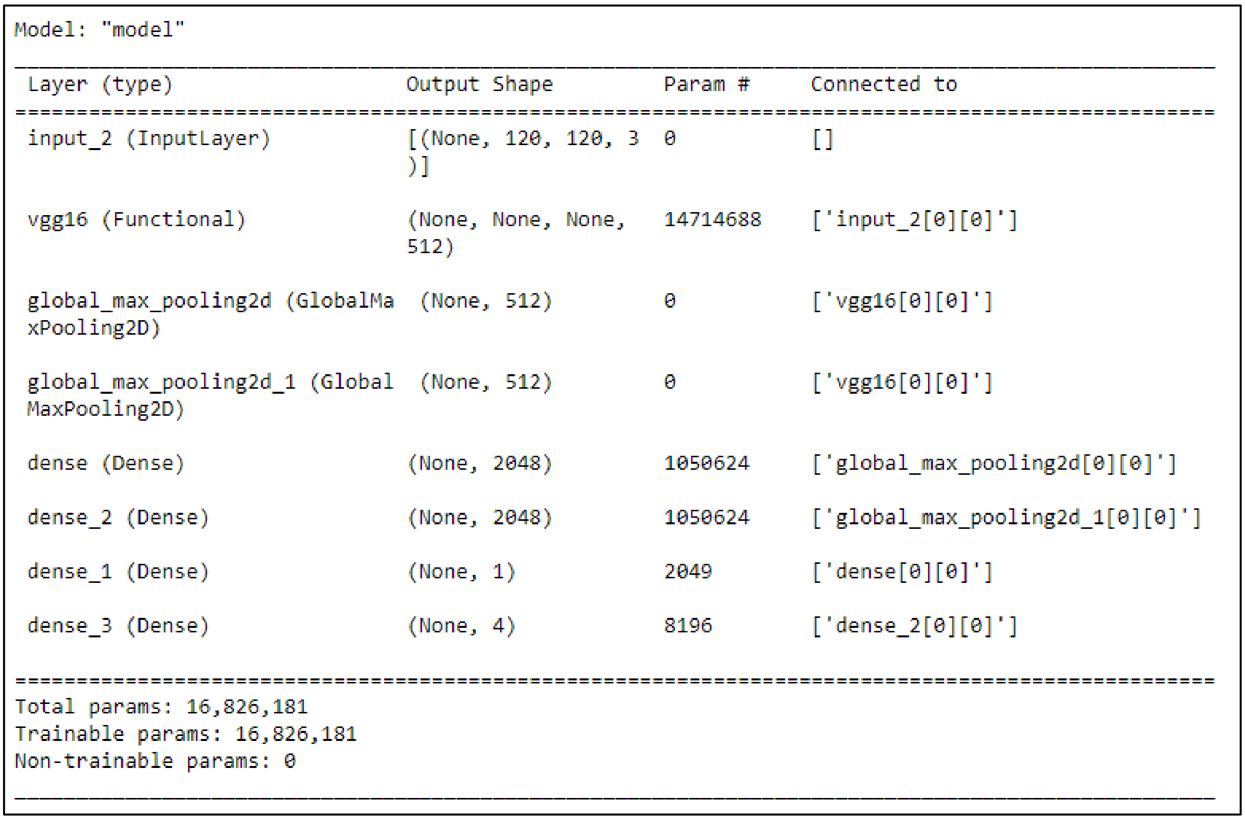
In this step we're going to preprocess the images and load the dataset using some functions below:

1. to load the data to a TensorFlow dataset with the function **tf.data.dataset("data_path")**.
2. than we need to map the data by **map()**.
3. resize all the images with **tf.image.resize()**.
4. scale them by dividing all images to 255.
5. Finally, we need to create the training and testing batches **batch("length of the batch").**

II.4.1.2.4 Building the SSD model using TensorFlow.Keras

we need to import the VGG16 and other functions from TensorFlow.Keras, we import the module **Model** from **tensorflow.keras.models,** CNN **layers** from **tensorflow.keras.layers** and the **VGG16** from **tensorflow.keras.applications**.

this VGG16 is pretrained network, we use it to get a better result. It is important that we remove the head top of the VGG16 network to put our classification layers as we see in the image (Figure II-12):



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 120, 120, 3  0           []
                                )]

 vgg16 (Functional)             (None, None, None,   14714688    ['input_2[0][0]']
                                512)

 global_max_pooling2d (GlobalMa  (None, 512)         0           ['vgg16[0][0]']
 xPooling2D)

 global_max_pooling2d_1 (Global  (None, 512)         0           ['vgg16[0][0]']
 MaxPooling2D)

 dense (Dense)                  (None, 2048)         1050624     ['global_max_pooling2d[0][0]']

 dense_2 (Dense)                (None, 2048)         1050624     ['global_max_pooling2d_1[0][0]']

 dense_1 (Dense)                (None, 1)            2049        ['dense[0][0]']

 dense_3 (Dense)                (None, 4)            8196        ['dense_2[0][0]']

==================================================================================================
Total params: 16,826,181
Trainable params: 16,826,181
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure II-12 the final model architecture

II.4.1.3Training the SSD model

II.4.1.3.1 Training the model

For training we need to set the Loss function and the optimizer

II.4.1.3.1.1 Loss functions

The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by the machine learning model. From the loss function, we can derive the gradients which are used to update the weights. The average over all losses constitutes the cost.

A deep learning model such as a neural network attempts to learn the probability distribution underlying the given data observations. In machine learning, we commonly use the statistical framework of maximum likelihood estimation as a basis for model construction. This basically means we try to find a set of parameters and a prior probability distribution such as the normal distribution to construct the model that represents the distribution over our data.

II.4.1.3.1.2 Cross entropy

In classification scenarios, loss functions that measure cross entropy are commonly used. Cross entropy is a metric that indicates the difference between two likelihood distributions. In machine learning, we use maximum likelihood estimation to determine the difference between the expected outcome distribution (data generating process) and the distribution produced by our model of the process.

The difference between the prediction and reality is called the loss. As the prediction gets farther from the truth, the loss gets bigger faster.

To minimize loss, the model should produce a probability estimate that is as close to 1 as possible if the actual outcome was 1.

If the actual outcome is 0, the model should produce a probability estimate that is as close as possible to 0 (Figure ll-13).

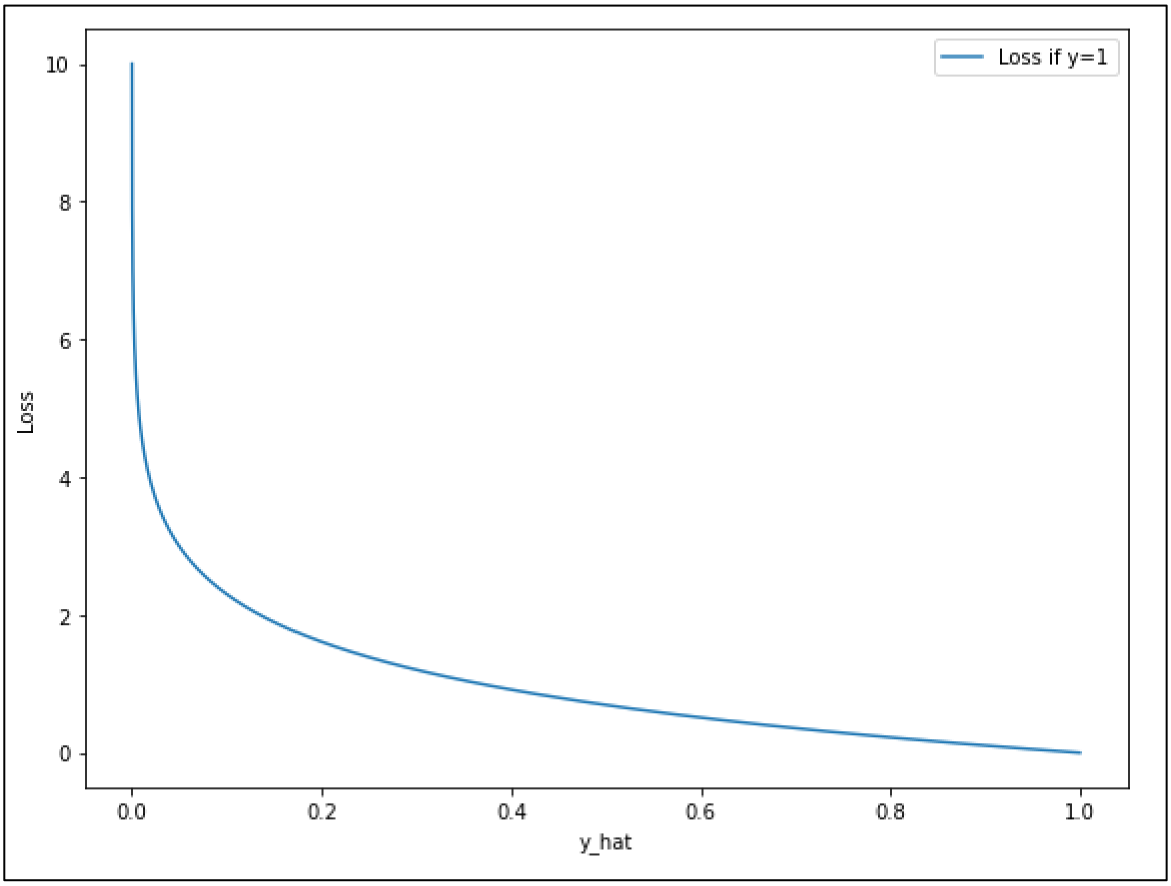


Figure II-13 loss function

II.4.1.3.1.3 Binary Cross-Entropy

The binary cross-entropy loss is used in binary classification settings when there are two possible outcomes. The formula for calculating the loss is as follows: **Y_actual** is the expected outcome, **Y_pred** the output of the model (Equation II-1).

$$Loss = Abs(Y\_pred - Y\_actual)$$

Equation II-1 the loss calculation

After calculating the correct probabilities (Loss), we need to calculate the logloss (the logarithmic loss **L**) (Equation II-2).

$$L = -\left(yilog\left(y^{i}\right) + (1 - yi)log\left(1 - y^{i}\right)\right)$$

Equation II-2 the binary cross-entropy loss equation

Actually, we won’t see any of this happening all we do is calling the binary cross entropy function from the TensorFlow libraries to calculate the loss.

II.4.1.3.1.4 The optimizer

A neural network can have millions of parameters, so selecting the right weights for the model is a big job. An optimizer is a function or algorithm that changes the weights and learning rate of the neural network, helping to lower the loss and improve the accuracy of the model. While training the model, we have to modify each epochs weights, using an optimizer, along with minimizing the loss function.

we can use different optimizers to make changes in the weights and learning rate and for that we’re using Adam optimizer.

II.4.1.3.1.5 Adam Optimizer

Adam is named after the adaptive moment estimation algorithm, which is a variation of stochastic gradient descent optimization used to change neural network weights during learning. The Adam algorithm updates the learning rate for each individual weight in the network, rather than maintaining a single learning rate throughout the entire learning process. Adam was created by people who were familiar with the AdaGrad and RMSProp algorithms, which are also variations of stochastic gradient descent. Adam optimizers combine the features of both Adagrad and RMS Prop algorithms. Unlike RMS Prop, which uses the first moment (mean) of the gradients to calculate learning rates, Adam also considers the second moment of the gradients, the uncentred variance.

Adam is a deep learning algorithm that is used widely, because of its many benefits. It is considered a good benchmark for papers on deep learning, used as the default optimization method, and easy to implement. Adam also requires less tuning than other optimization algorithms, and runs faster and uses less memory.

$$m_t = \beta_1 m_{t-\Lambda} + \left(1 - \beta_1\right) \left[\frac{\delta L}{\delta \omega_t}\right] v_t = \beta_2 v_{t-1} + \left(1 - \beta_2\right) \left[\frac{\delta L}{\delta_{\omega t}}\right]^2$$

Equation II-3 the Adam optimizer equation

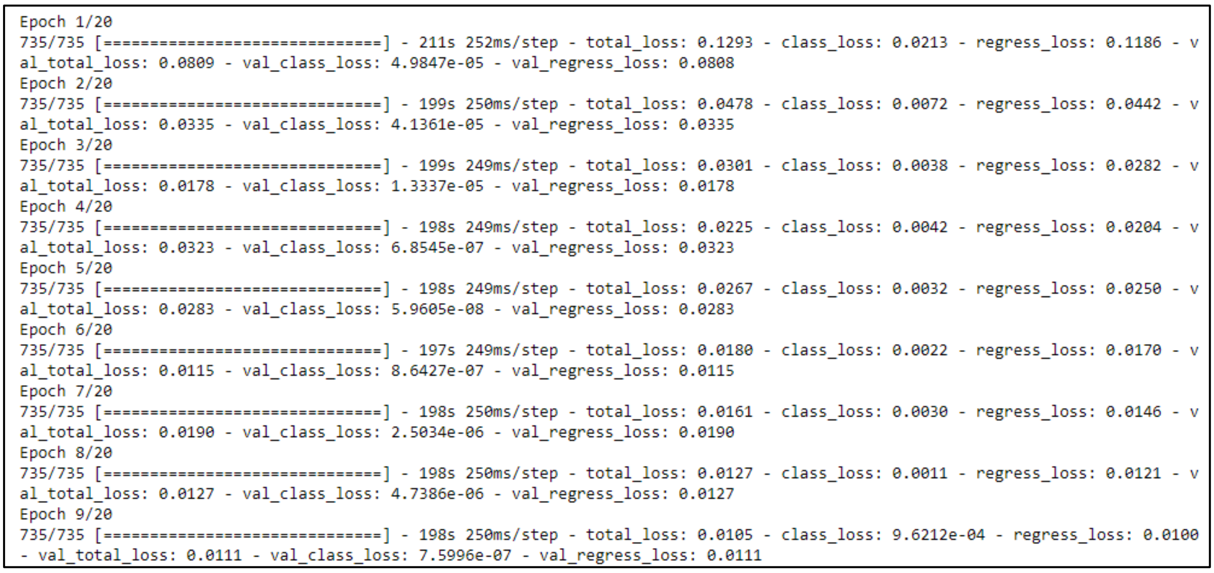
The formula (Equation ll-3) represents the working of Adam optimizer. Here β 1 and β 2 represent the decay rate of the average of the gradients, again all we do is call it from TensorFlow library.

II.4.1.3.1.6 Training

Training a deep learning model need a high-performance computer, and for that we're using Google Colab.

to train the model we need to create a class model and define the classification loss, regression loss (Binary Cross Entropy) and the optimizer (Adam Optimizer).

Next phase is printing the losses (Figure II-14) to get the curves after the training is finished (Figure II-15, Figure II-16).



```
Epoch 1/20
735/735 [==============================] - 211s 252ms/step - total_loss: 0.1293 - class_loss: 0.0213 - regress_loss: 0.1186 - v
al_total_loss: 0.0809 - val_class_loss: 4.9847e-05 - val_regress_loss: 0.0808
Epoch 2/20
735/735 [==============================] - 199s 250ms/step - total_loss: 0.0478 - class_loss: 0.0072 - regress_loss: 0.0442 - v
al_total_loss: 0.0335 - val_class_loss: 4.1361e-05 - val_regress_loss: 0.0335
Epoch 3/20
735/735 [==============================] - 199s 249ms/step - total_loss: 0.0301 - class_loss: 0.0038 - regress_loss: 0.0282 - v
al_total_loss: 0.0178 - val_class_loss: 1.3337e-05 - val_regress_loss: 0.0178
Epoch 4/20
735/735 [==============================] - 198s 249ms/step - total_loss: 0.0225 - class_loss: 0.0042 - regress_loss: 0.0204 - v
al_total_loss: 0.0323 - val_class_loss: 6.8545e-07 - val_regress_loss: 0.0323
Epoch 5/20
735/735 [==============================] - 198s 249ms/step - total_loss: 0.0267 - class_loss: 0.0032 - regress_loss: 0.0250 - v
al_total_loss: 0.0283 - val_class_loss: 5.9605e-08 - val_regress_loss: 0.0283
Epoch 6/20
735/735 [==============================] - 197s 249ms/step - total_loss: 0.0180 - class_loss: 0.0022 - regress_loss: 0.0170 - v
al_total_loss: 0.0115 - val_class_loss: 8.6427e-07 - val_regress_loss: 0.0115
Epoch 7/20
735/735 [==============================] - 198s 250ms/step - total_loss: 0.0161 - class_loss: 0.0030 - regress_loss: 0.0146 - v
al_total_loss: 0.0190 - val_class_loss: 2.5034e-06 - val_regress_loss: 0.0190
Epoch 8/20
735/735 [==============================] - 198s 250ms/step - total_loss: 0.0127 - class_loss: 0.0011 - regress_loss: 0.0121 - v
al_total_loss: 0.0127 - val_class_loss: 4.7386e-06 - val_regress_loss: 0.0127
Epoch 9/20
735/735 [==============================] - 198s 250ms/step - total_loss: 0.0105 - class_loss: 9.6212e-04 - regress_loss: 0.0100
- val_total_loss: 0.0111 - val_class_loss: 7.5996e-07 - val_regress_loss: 0.0111
```

Figure II-14 the training for 20 epochs

II.4.1.4Testing and results

II.4.1.4.1 Results

lastly, after waiting for training process for a long time (2h 30min) we got the learning rate and classification and regression loss and plot them to confirm the training results.

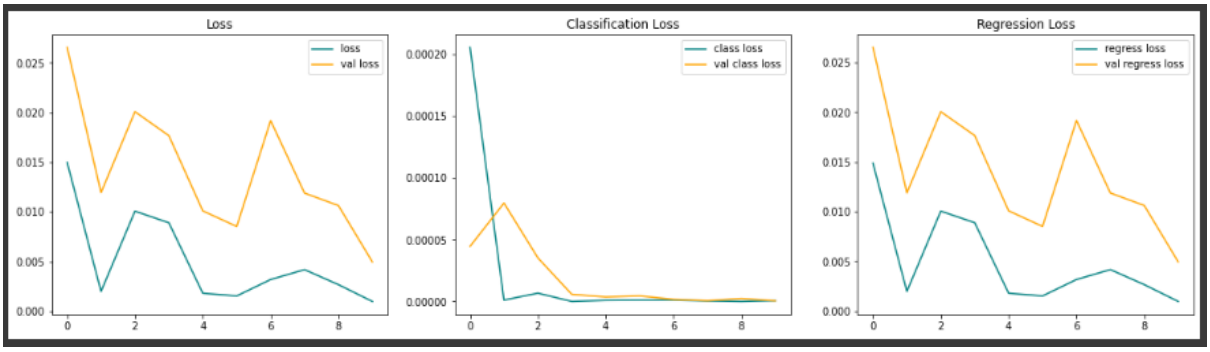


Figure II-15 10 epochs training curves

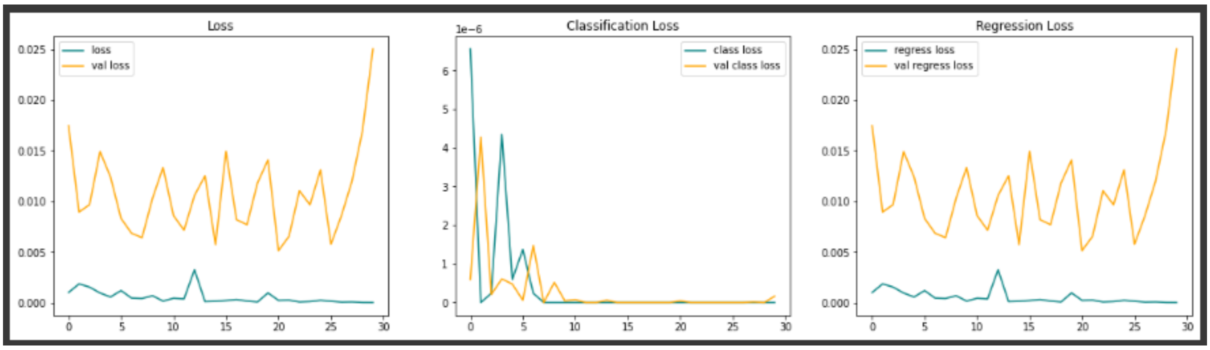


Figure II-16 30 epochs training curves

From the (Figure ll-15) the training and regression loss stabilizes at 10 epochs but the training loss of (Figure ll-16) get higher at 10 epochs than stabilizes at 15 these fluctuations happen because of the dataset augmentation, some pictures in the dataset are lower quality than others, that’s the point of the training. To get a good model you have to train on a diverse dataset (different brightness, different positions …).

II.4.1.4.2 Test

Now we use the batch test to make predictions and test the model before saving it, all we have to do is calling the test batch and make the predictions using the function **predict()**, then plot the result (Figure ll-17).

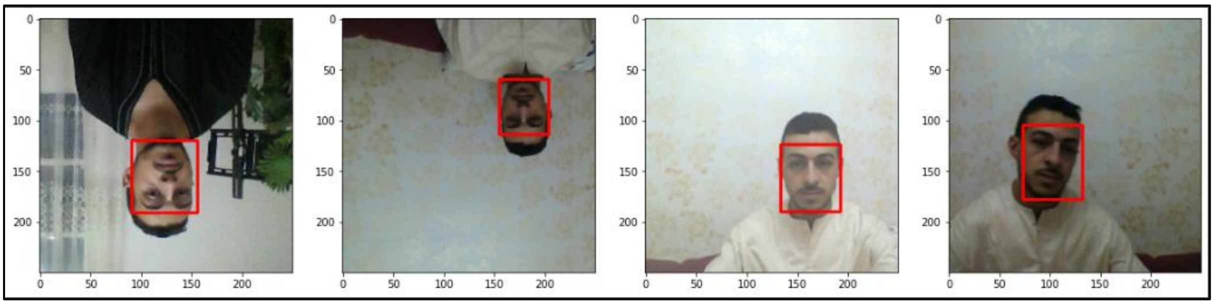


Figure II-17 test batch results

II.4.2 YOLOv5 for face detection

To create the custom YOLOv5 model we follow multiple steps (Figure ll-18).

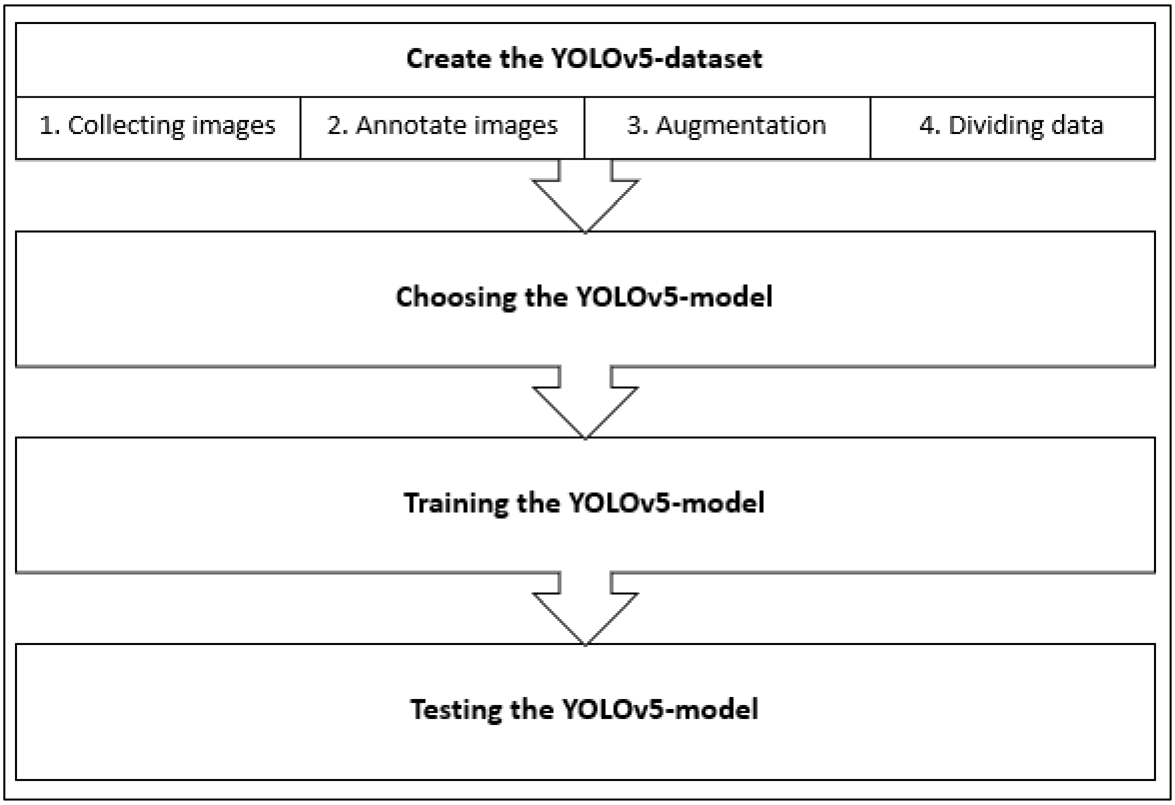


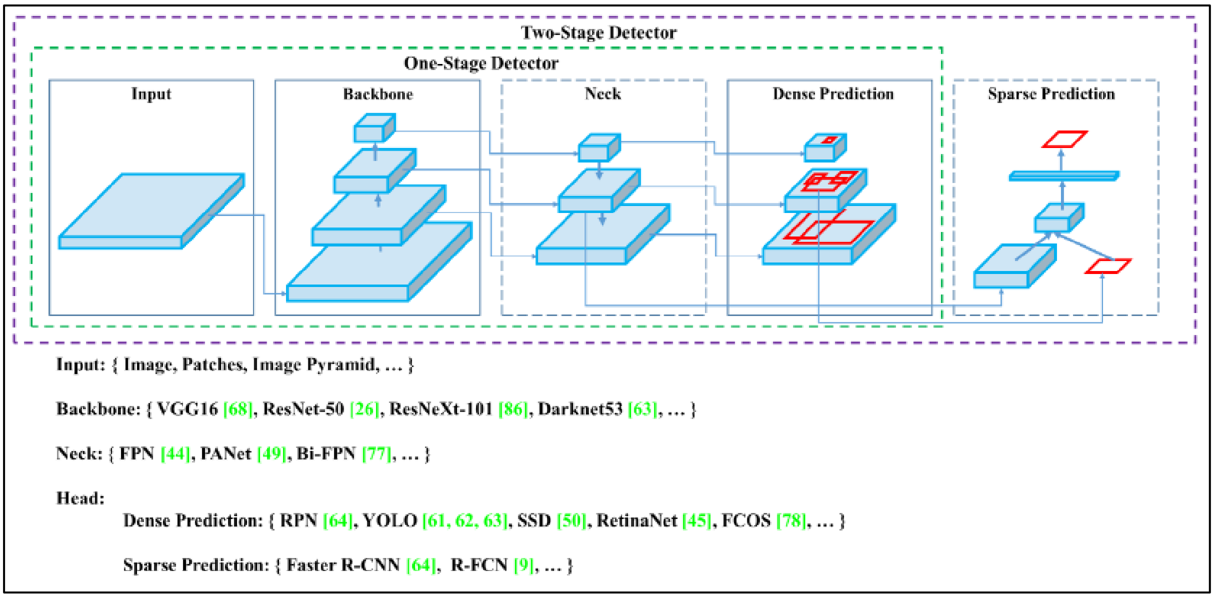
Figure II-18 Steps to create the YOLOv5 model

II.4.2.1 The YOLOv5-model

II.4.2.1.1 You Only Look Once (YOLO)

YOLO (**Y**ou **O**nly **L**ook **O**nce) models are used for Object detection with high performance. YOLO divides an image into a grid system, and each grid detects objects within itself. They can be used for real-time object detection based on the data streams. They require very few computational resources.

To understand how Yolov5 improved the performance and its architecture, let us go through the following high-level Object detection architecture (Figure II-19):



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure II-19 high-level Object detection architecture

General Object Detector will have a **backbone** for pre-training it and a **head** to predict classes and bounding boxes. The ***Backbones*** can be running on GPU or CPU platforms. The ***Head*** can be either **one-stage** (e.g., YOLO, SSD, RetinaNet) for **Dense prediction** or **two-stage** (e.g., Faster R-CNN) for **the Sparse prediction** object detector. Recent Object detectors have some layers (***Neck***) to collect feature maps, and it is between the backbone and the Head.

The network architecture of Yolov5. It consists of three parts, first **CSPDarknet** is used as a backbone and **SPP** block for increasing the receptive field, which separates the significant features, and there is no reduction of the network operation speed. Second part is the neck: **PANnet** is used for parameter aggregation from different backbone levels. And third we have the YOLO layer which is the head, The data are first input to **CSPDarknet** for feature extraction, and then fed to **PANet** for feature fusion. Finally, Yolo Layer outputs detection results (class, score, location, size).[21]

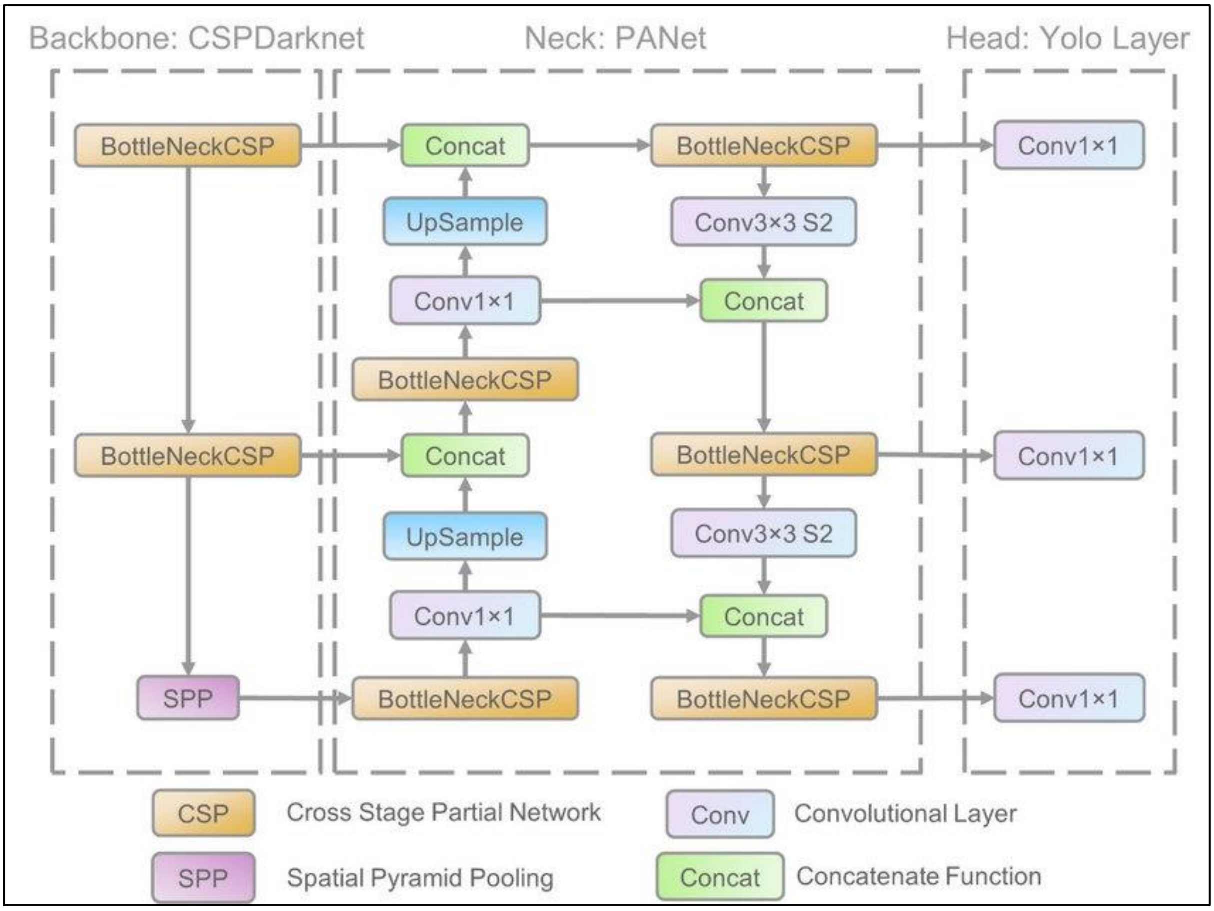


Figure II-20 YOLOv5 architecture

II.4.2.1.2 Working of the YOLO algorithm:

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)

Residual blocks

First, the image is divided into various grids. Each grid has a dimension of S x S. The following image shows how an input image is divided into grids. (Figure ll-21) [24]

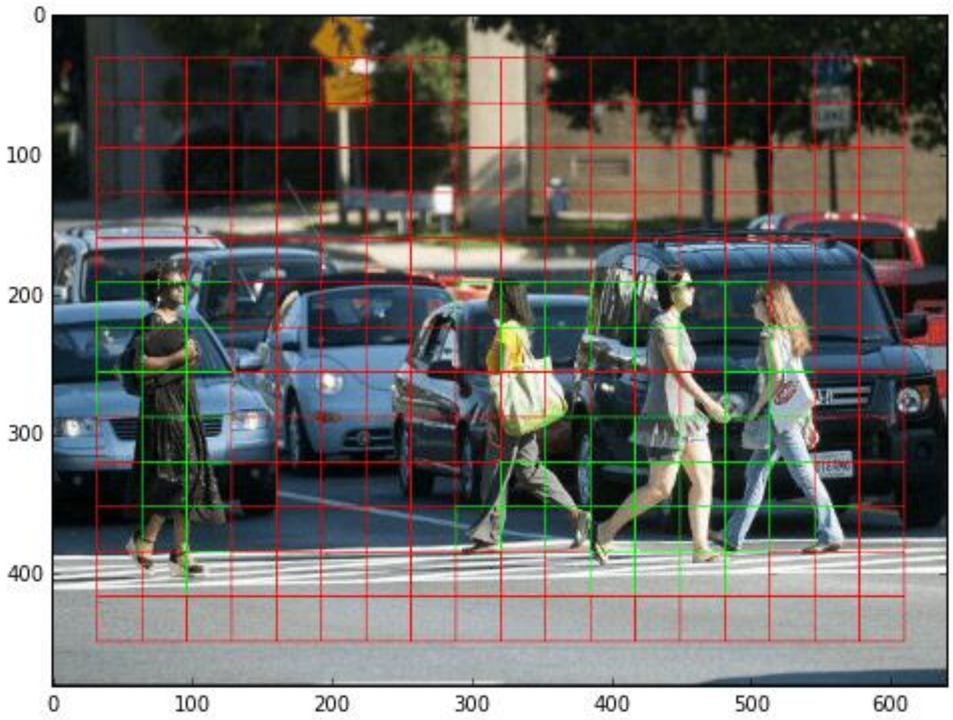


Figure II-21 Residual blocks [24]

In the image above, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.[24]

Bounding box regression

A bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of the following attributes:

- Width (bw)
- Height (bh)
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter c.
- Bounding box center (bx,by)

The following image shows an example of a bounding box. The bounding box has been represented by a yellow outline. (Figure ll-22) [24]

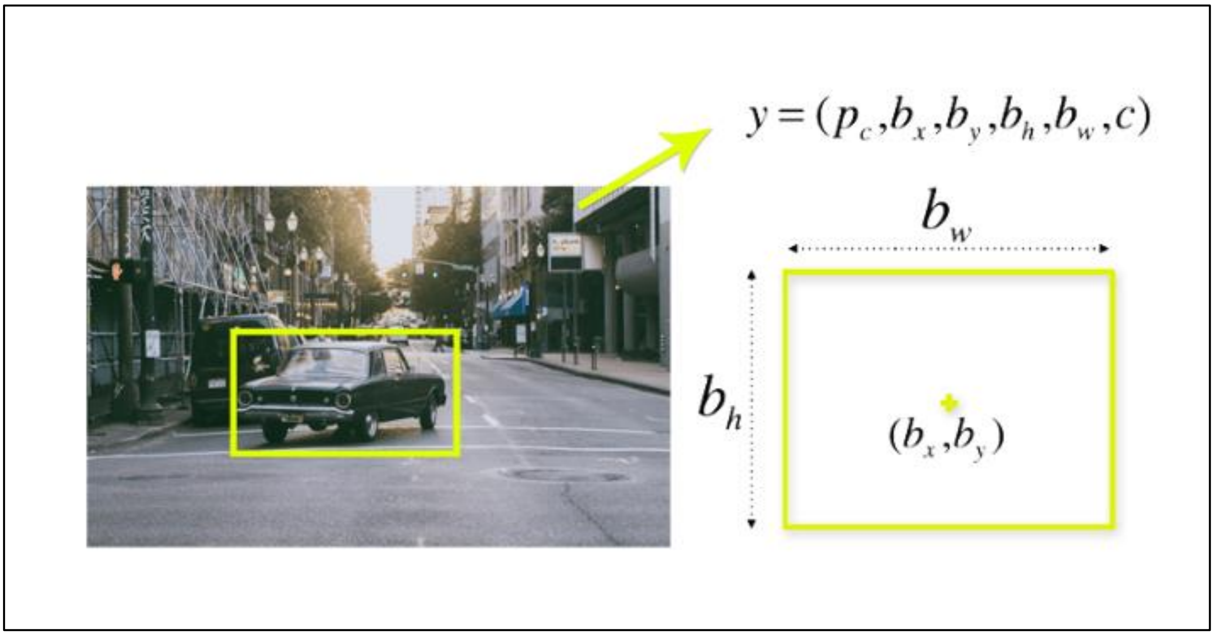


Figure II-22 bounding box[24]

YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. In the image above, represents the probability of an object appearing in the bounding box.

Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box. [24]

The following image provides a simple example of how IOU works. (Figure ll-23) [24]

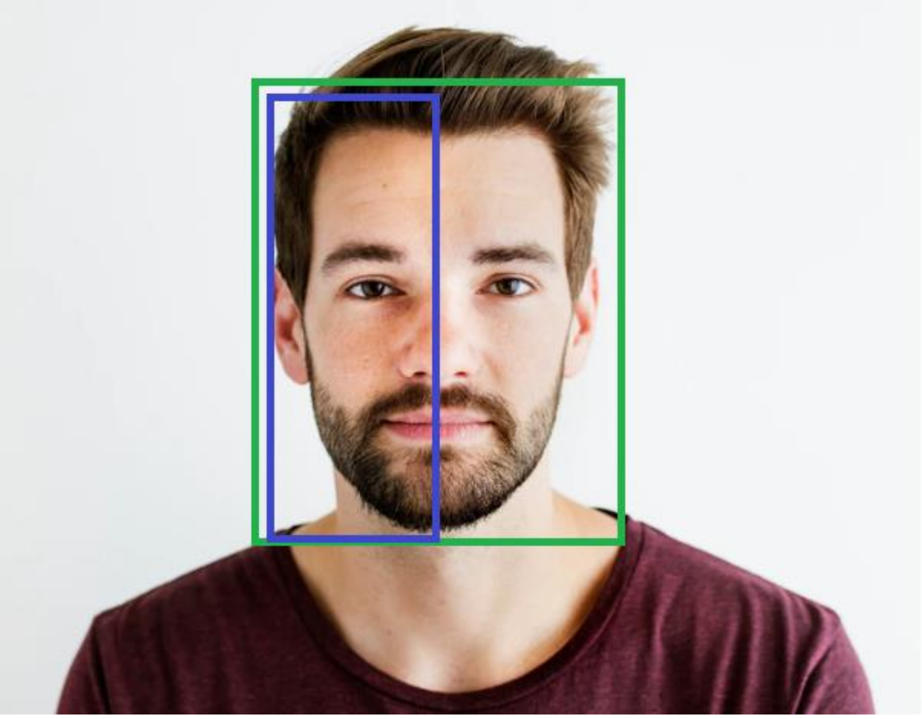


Figure II-23 Intersection over union

In the image above, there are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal. [24]

Combination of the three techniques

The following image shows how the three techniques are applied to produce the final detection results. (Figure II-24)

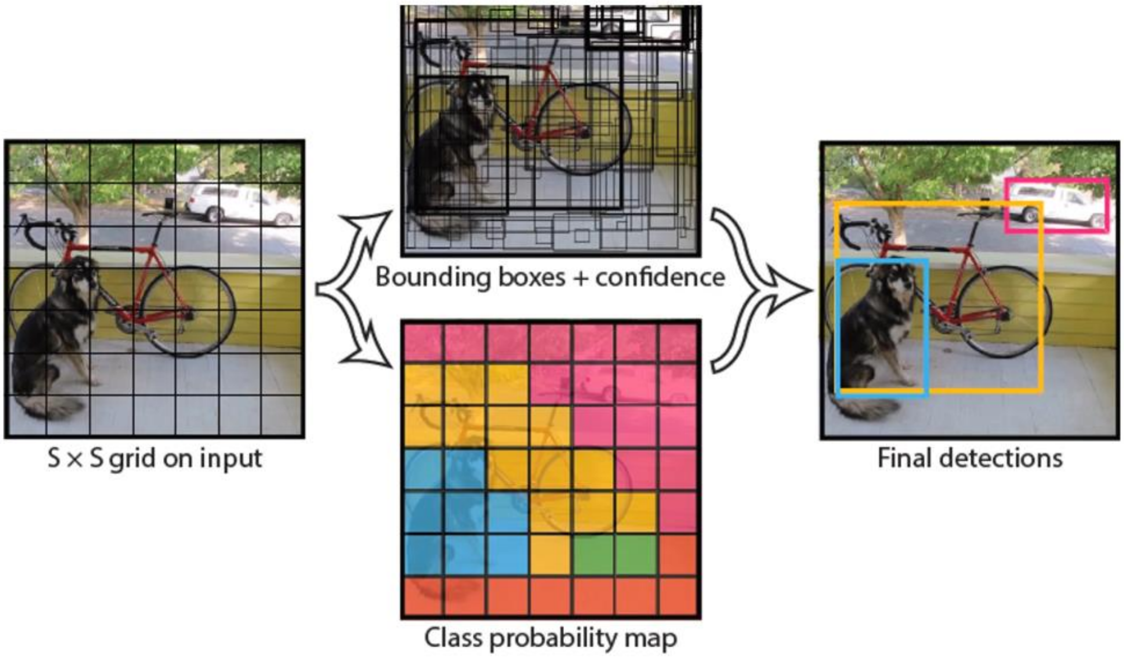
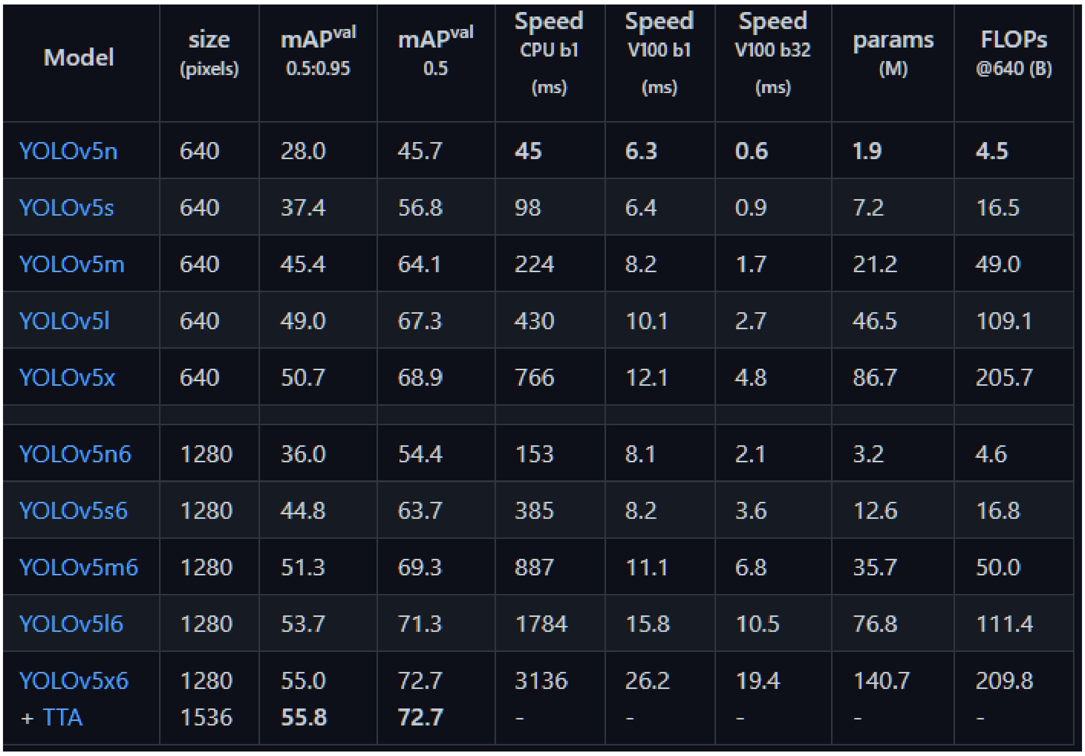


Figure II-24 Combination of the three techniques

II.4.2.1.3 Choosing the Yolov5 model

When choosing the model, we take in consideration the accuracy and speed and image size, because our image size is 416*416, the P5 models are suitable for training, the P5 models are: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), YOLOv5x (extra-large), we chose the YOLOv5s which is the small version, it's great for real-time usage with very small lose in accuracy compared to the bigger models.



Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Figure II-25 yolov5 models [22]

II.4.2.2 Creating the YOLOv5-Dataset

II.4.2.2.1 Collecting the images

The dataset used in the yolov5 model for face detection is obtained from the internet, a popular website used for sharing datasets and pre-trained models. [16]

The images in this dataset have different configurations that we'll explain below.

II.4.2.2.2 Dataset configuration

to achieve better training results different configurations are applied, shown down below:

II.4.2.2.2.1 Pre-processing

Decrease training time and increase performance by applying image transformations to all images in this dataset.

- Auto-Orient: Discard EXIF rotations and standardize pixel ordering.
- Resize: Downsize images for smaller file sizes and faster training.

II.4.2.2.2.2 Augmentation

Create new training examples for your model to learn from by generating augmented versions of each image in your training set (Figure II-26).

- Flip: Add horizontal or vertical flips to help your model be insensitive to subject orientation.
- 90° Rotate: Add 90-degree rotations to help your model be insensitive to camera orientation: clockwise, counter-clockwise, upside down.
- Saturation: Randomly adjust the vibrancy of the colors in the images. Between -10% and +10%
- Brightness: Add variability to image brightness to help your model be more resilient to lighting and camera setting changes. Between -10% and +10%
- Exposure: Add variability to image brightness to help your model be more resilient to lighting and camera setting changes. Between -10% and +10%
- Blur: Add random Gaussian blur to help your model be more resilient to camera focus. Up to 1px.

After augmenting the images in the training set, we have three times (x3) the amount of the original number, going from 737 images to 2211 images in the training set.

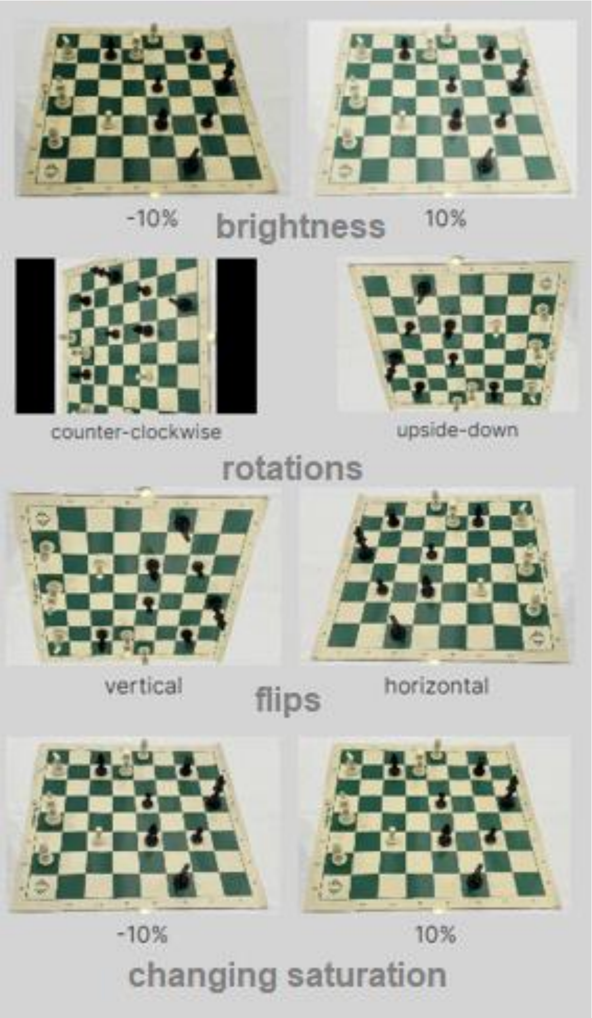


Figure II-26 augmentation examples

II.4.2.2.2.3 Training/validation split

The dataset is split into two categories: training and validation.

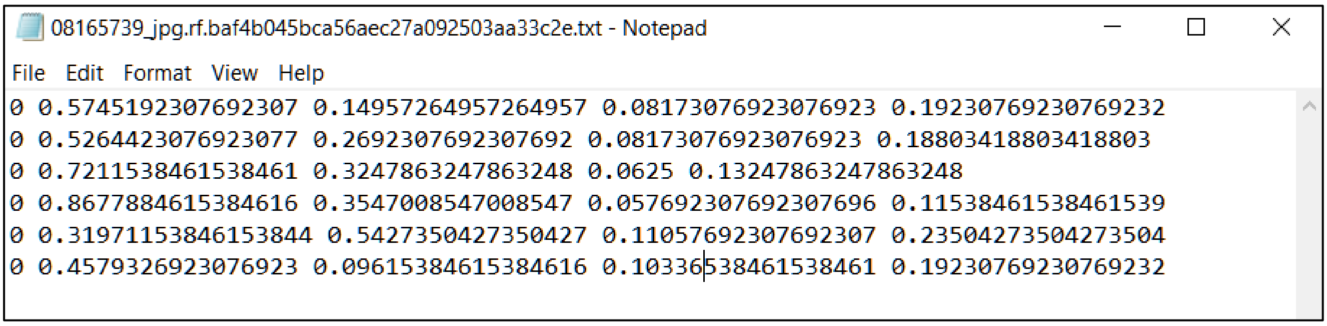
The dataset contains 2420 images in total, 2211 training images and 209 validation images (88% in training / 8% in validation). Both training and validation images are annotated (labeled). We only have 1 class for the annotations which is: face.

II.4.2.2.2.4 YOLOv5 labelling format

The roboflow platform support export to YOLOv5 labeling format, providing one annotations text file per image. Each text file contains one bounding-box (BBox) annotation for each of the faces in the image, the name of the text file is the same as the image. The annotations are normalized to the image size, and lie within the range of 0 to 1. They are represented in the following format:

< object-class-ID> <X center> <Y center> <Box width> <Box height>

If there are 6 faces in the image, the content of the YOLO annotations text file will look like the (Figure II-27)


08165739_jpg.rf.baf4b045bca56aec27a092503aa33c2e.txt - Notepad
File
Edit
Format
View
Help

```
0 0.5745192307692307 0.14957264957264957 0.08173076923076923 0.19230769230769232
0 0.5264423076923077 0.2692307692307692 0.08173076923076923 0.18803418803418803
0 0.7211538461538461 0.3247863247863248 0.0625 0.13247863247863248
0 0.8677884615384616 0.3547008547008547 0.057692307692307696 0.11538461538461539
0 0.31971153846153844 0.5427350427350427 0.11057692307692307 0.23504273504273504
0 0.4579326923076923 0.09615384615384616 0.10336538461538461 0.19230769230769232
```

Figure II-27 YOLOv5 label file

II.4.2.2.2.5 Data directories structure

This is the data structure we followed (Figure ll-28)

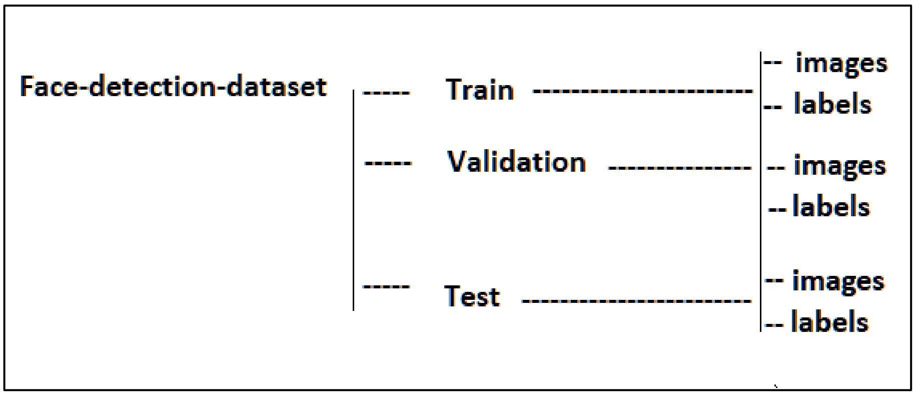

Face-detection-dataset
----- Train -----------------------
-- images
-- labels
----- Validation ---------------
-- images
-- labels
----- Test ------------------------
-- images
-- labels

Figure II-28 YOLOv5 Data directories structures

the images include faces with and without masks, we gave the class ‘face’ to images showing faces and faces with masks, doing so will increase the accuracy of detecting faces even with something blocking the face in the image like with a mask on. Some images have multiple faces in a single image and some images have just 1 face in a single image and also, we included 150 images marked as Null, those are images with no faces, this helps with feature extraction and reducing false detections.

The annotation ‘face’ contains the image name, its path, and the size of the image (Figure ll-30). Notably, this annotation does not contain any bounding box information. It is a **null annotation** (Figure ll-31).

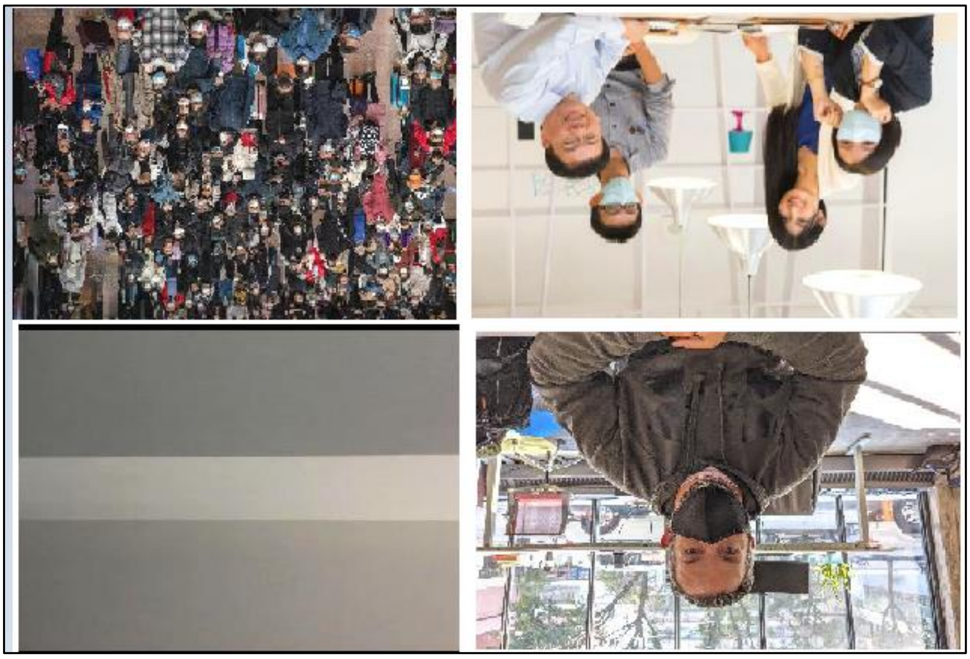
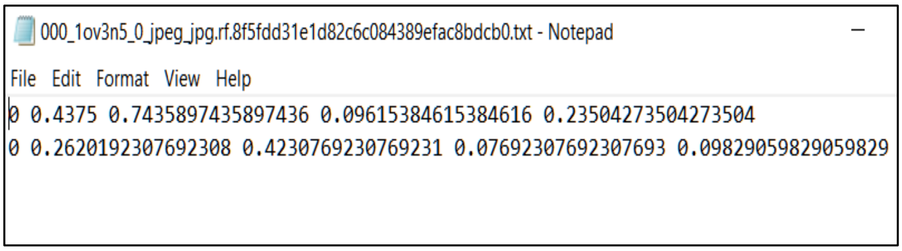


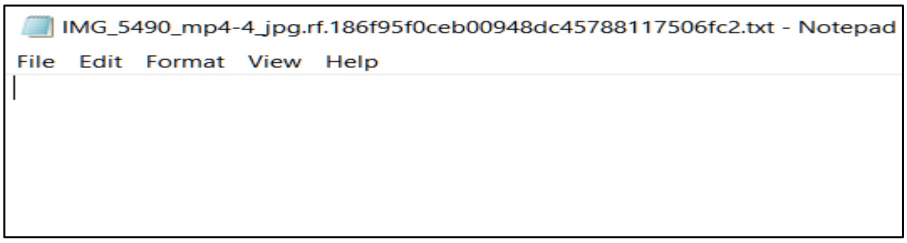
Figure II-29 example of images from the dataset



000_1ov3n5_0_jpeg_jpg.rf.8f5fdd31e1d82c6c084389efac8bdcb0.txt - Notepad
File Edit Format View Help

```
0 0.4375 0.7435897435897436 0.09615384615384616 0.23504273504273504
0 0.2620192307692308 0.4230769230769231 0.07692307692307693 0.09829059829059829
```

Figure II-30 label file for face annotation



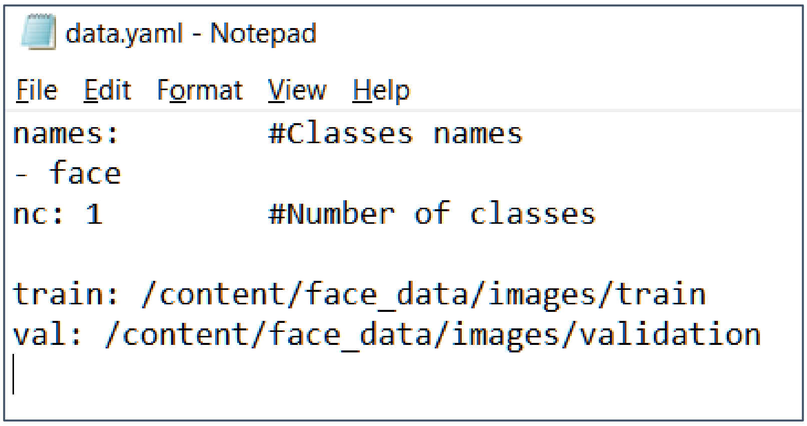
IMG_5490_mp4-4_jpg.rf.186f95f0ceb00948dc45788117506fc2.txt - Notepad
File Edit Format View Help

Figure II-31 label file for Null image not containing a face

II.4.2.2.3 Configuration files

The configuration files for the training are divided to three YAML files, which are provided with the GitHub repository itself. We will customize these files depending on the task, to fit our desired needs. We will only be changing the data configurations file, using the default parameters for the other 2 configurations files is recommended.

1-The data-configurations file: describes the dataset parameters. Since we are training on our custom face-detection dataset, we will edit this file and provide: the paths to the train, validation and test datasets (the test dataset path is optional when training the model); the number of classes (nc); and the names of the classes in the same order as their index. we only have one class, named ‘face’. We named our custom data configurations file as ‘data.yaml’ and placed it under the ‘data’ directory. The content of this YAML file is as follow (Figure ll-32).



```
names:          #Classes names
- face
nc: 1           #Number of classes

train: /content/face_data/images/train
val: /content/face_data/images/validation
```

Figure II-32 Data configurations file

2. The model-configurations file: dictates the model architecture. Ultralytics (the company who released YOLOv5) supports several YOLOv5 architectures, named P5 models, which varies mainly by their parameters size: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), YOLOv5x (extra-large). These architectures are suitable for training with image size of 640*640 pixels, image size of 416*416 also works great. Additional series, that is optimized for training with larger image size of 1280*1280, called P6 (YOLOv5n6, YOLOv5s6, YOLOv5m6, YOLOv5l6, YOLOv5x6). P6 models include an extra output layer for detection of larger objects. They benefit the most from training at higher resolution, and produce better results.

Ultralytics provides built-in, model-configuration files for each of the above architectures, placed under the ‘models’ directory. If we were to train from scratch, we would choose the model-configurations YAML file with the desired architecture, and just edit the number of classes (nc) parameter to the correct number of classes in our custom data.

But since our training is initialized from pre-trained weights, no need to edit the model-configurations file since the model will be extracted with the pretrained weights.

3. The hyperparameters-configurations files: defines the hyperparameters for the training, including the learning rate, momentum, losses, augmentations etc. Ultralytics provides a default hyperparameters files under the ‘data/hyps/’ directory. We will be using the default hyperparameters to establish a performance baseline.

II.4.2.3 Training the YOLOv5 model

II.4.2.3.1 Precision and Recall metrics:

Many object detection algorithms, such as Faster R-CNN, MobileNet, SSD, and YOLO, use **mAP** to evaluate their models. We need to calculate Recall and Precision metrics first.

Precision measures the percentage of the correct predictions, It measures how many of the predictions that our model made were actually correct.[23]

the calculation of the precision is as follow (Equation II-4):

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

Equation II-4 Precision Formula

TP = True Positives (The model predicted a label and matches correctly as per ground truth.)

FP = False Positives (The model predicted a label, but it is not a part of the ground truth)

Object detection systems make predictions in terms of a bounding box and a class label. For each bounding box, we measure an overlap between the predicted bounding box and the ground truth bounding box. This is measured by IoU (intersection over union)[23].

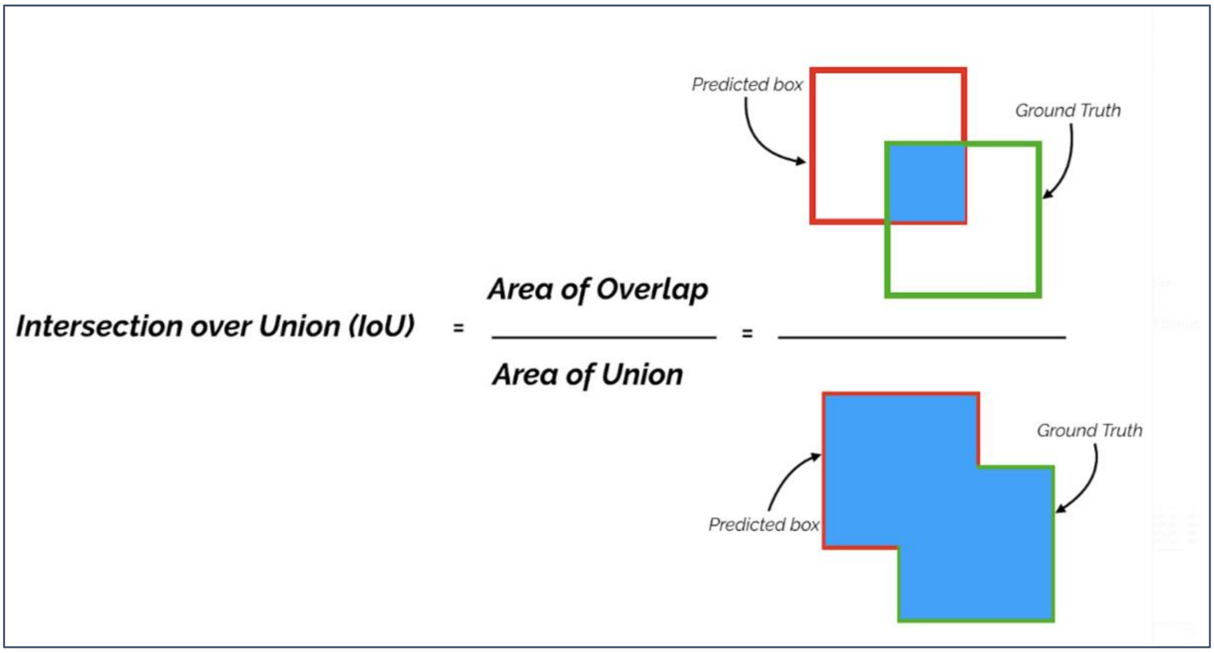


Figure II-33 Intersection over Union

For object detection tasks, we calculate Precision and Recall using IoU value for a given IoU threshold.

For example, if **IoU threshold is 0.5**, and the IoU value for a prediction is 0.7, then we classify the prediction as True Positive (TP). On the other hand, if IoU is 0.3, we classify it as False Positive (FP) (Figure ll-34).

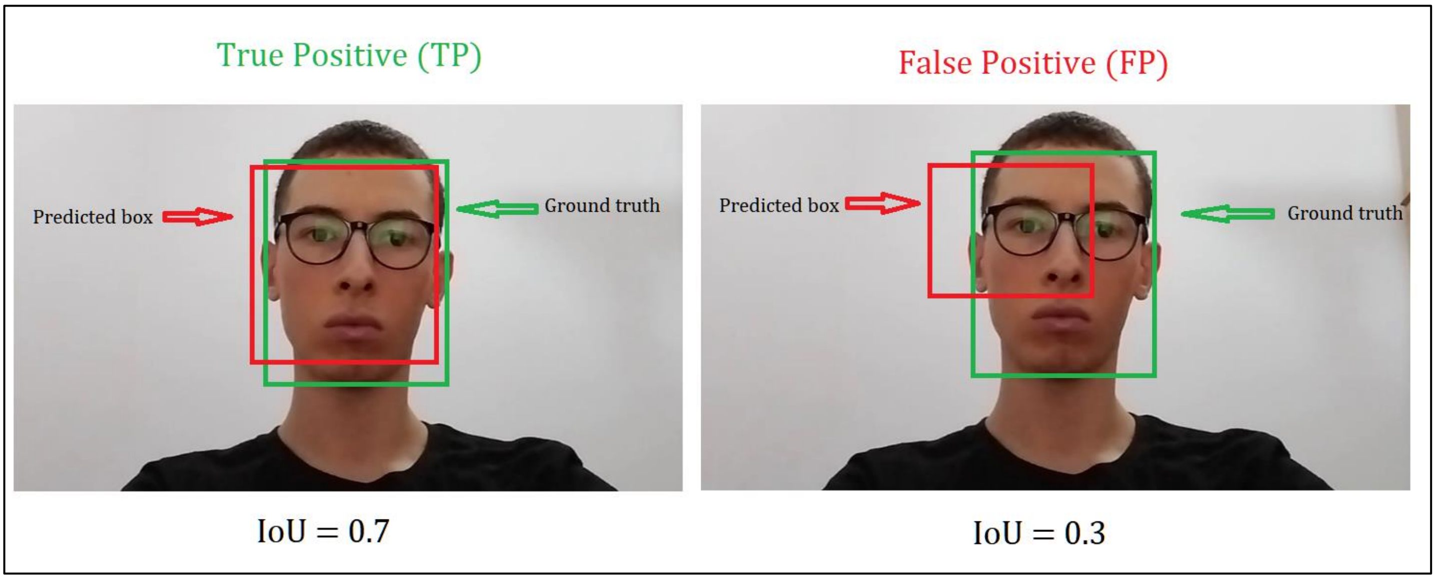


Figure II-34 TP and FP example

That also means that for a prediction, we may get different binary TRUE or FALSE positives, by changing the IoU threshold.

Recall measures how well we find all the positives, the calculation of the precision is as follows:

$$Recall = \frac{TP}{TP+FN}$$ Equation II-5 the Recall formula

FN = False Negatives (The model does not predict a label, but it is part of the ground truth)

The general definition for the Average Precision (AP) is finding the area under the precision-recall curve.

After we calculate the precision and recall metrics, we plot the precision-Recall graph and calculate the area under the precision-recall curve. (Figure II-35)

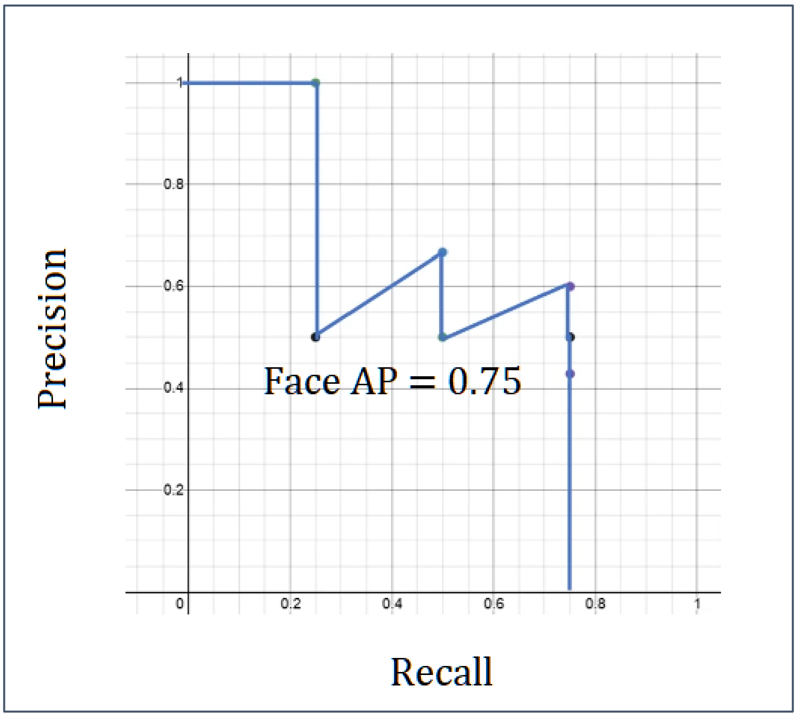


Figure II-35 Precession-recall graph example

The mAP is calculated by finding Average Precision (AP) for each class and then average over a number of classes.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} APi$$

Equation II-6 Mean Average Precision Formula

Because we only have 1 class, the mAP would be equal to Face AP.

II.4.2.3.2 Training

When it comes to training with YOLOv5, two different training approaches are to be considered:

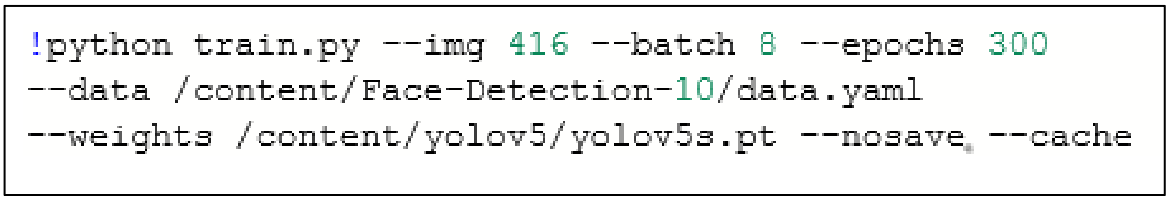
1-Training from scratch:

This approach is good when having a large dataset, the model will benefit most by training from scratch but it takes a lot of time to train.

2- Transfer learning:

Transfer learning speeds up the learning process and reduces the training time, Ultralytic's default model was pre-trained over the COCO dataset, COCO is an object detection dataset with images from everyday scenes containing 80 classes, our model will be initialized with weights from a pre-trained COCO model, this will produce better and faster results than training from scratch.

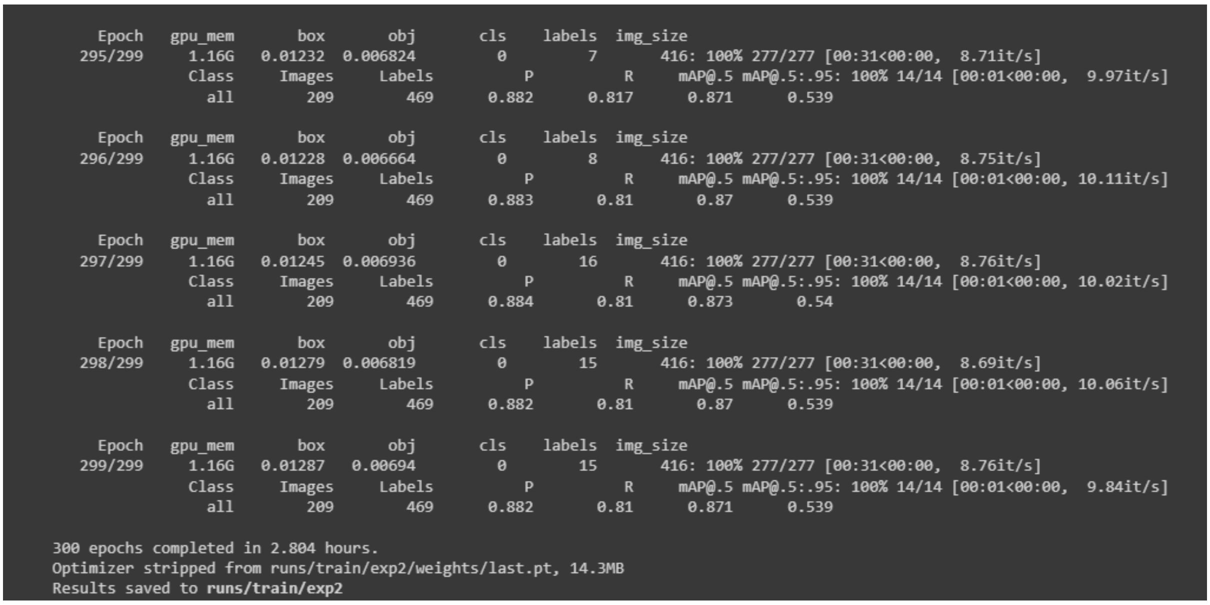
Now that we decided which approach to take, we can start the training. We will be using Google colaboratory to speed up the training process. After uploading all the necessary files including our dataset, the yolov5 repository, we import the required libraries and set our training, validation and testing path, and change directory to the yolov5 path. The training is induced by the following command (Figure ll-36):



```
!python train.py --img 416 --batch 8 --epochs 300
--data /content/Face-Detection-10/data.yaml
--weights /content/yolov5/yolov5s.pt --nosave --cache
```

Figure II-36 training command

- batch — batch size (-1 for auto batch size). Use the largest batch size that your hardware allows for.
- epochs — number of epochs.
- data — path to the data-configurations file.
- cfg — path to the model-configurations file.
- weights — path to initial weights.
- cache — cache images for faster training.
- img — image size in pixels (default — 640).



```
     Epoch   gpu_mem       box       obj       cls    labels  img_size
   295/299     1.16G   0.01232  0.006824         0         7       416: 100% 277/277 [00:31<00:00,  8.71it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 14/14 [00:01<00:00,  9.97it/s]
                 all        209        469      0.882      0.817      0.871      0.539

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   296/299     1.16G   0.01228  0.006664         0         8       416: 100% 277/277 [00:31<00:00,  8.75it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 14/14 [00:01<00:00, 10.11it/s]
                 all        209        469      0.883       0.81       0.87      0.539

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   297/299     1.16G   0.01245  0.006936         0        16       416: 100% 277/277 [00:31<00:00,  8.76it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 14/14 [00:01<00:00, 10.02it/s]
                 all        209        469      0.884       0.81      0.873       0.54

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   298/299     1.16G   0.01279  0.006819         0        15       416: 100% 277/277 [00:31<00:00,  8.69it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 14/14 [00:01<00:00, 10.06it/s]
                 all        209        469      0.882       0.81       0.87      0.539

     Epoch   gpu_mem       box       obj       cls    labels  img_size
   299/299     1.16G   0.01287   0.00694         0        15       416: 100% 277/277 [00:31<00:00,  8.76it/s]
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 14/14 [00:01<00:00,  9.84it/s]
                 all        209        469      0.882       0.81      0.871      0.539

300 epochs completed in 2.804 hours.
Optimizer stripped from runs/train/exp2/weights/last.pt, 14.3MB
Results saved to runs/train/exp2
```

Figure II-37 Model training in google colab

After the training is finished the results are automatically saved under ‘runs/train’ in the YOLOv5 directory (Figure ll-35), the training took a total of 2hours and 48 minutes.

To get a better understanding, we need to take a look at the losses and metrics of our Yolov5 model.(Figure II-38)

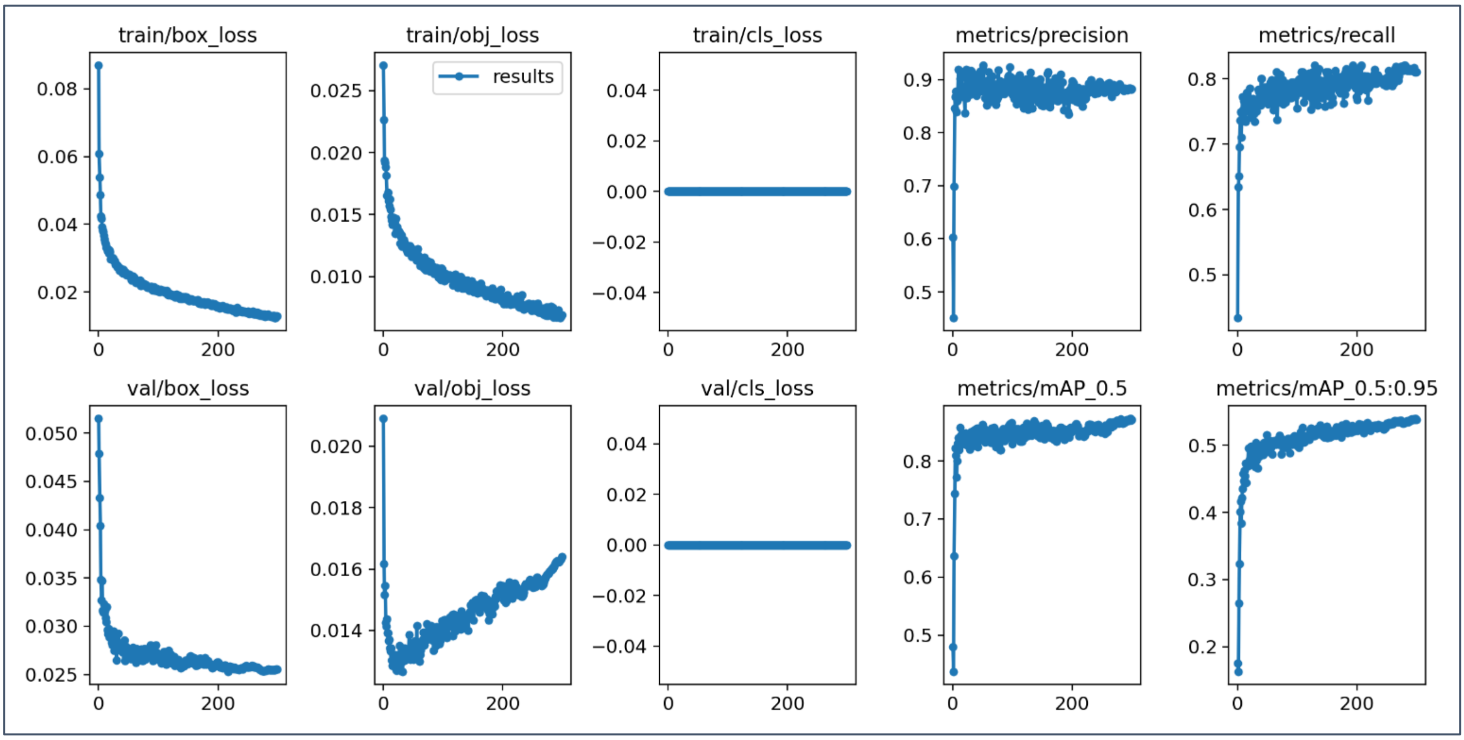


Figure II-38 Losses and Metrics

From the Figure we see that the **precision** and **recall** metrics converge at around 250 epochs with an average value of 0.87 for the **precision** and 0.81 for the **recall**.

The **mAP** is the “mean average precision”, it’s the most important metric to evaluate the model, for the **mAP_0.5** we get an average value of 0.87 and for **the mAP_0.5:0.95** we average a value of 0.52 which is pretty good for 300 epochs.

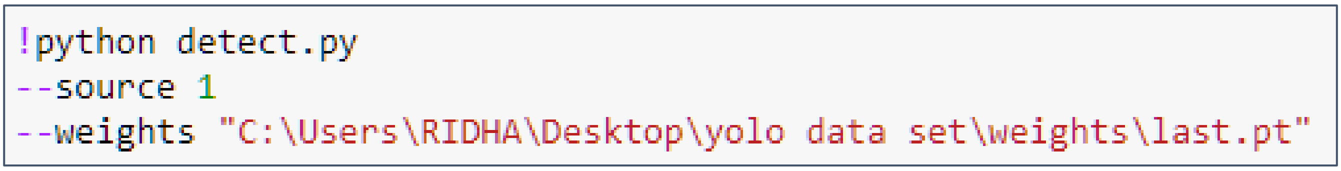
The **mAP metric value** is still increasing at 300 epochs, the value would increase more if trained for more epochs which leads to more precision at detecting the face.

The **box_loss** is the bounding box regression loss (using Mean Squared Error loss function). **Obj_loss** is the confidence of object presence (using Binary Cross Entropy loss function)**.** They both decrease over the number of epochs at a steady rate.

cls_loss is the classification loss (using Cross Entropy function), because we have only 1 class the classification error is constantly zero

II.4.2.4Testing and results:

After training our YOLOv5 model for face detection, it’s finally time to test it, we will be using the Ultralytics detection command, The input data for testing can be an image, a video, a directory, a webcam, a stream or even a YouTube link.



```
!python detect.py
--source 1
--weights "C:\Users\RIDHA\Desktop\yolo data set\weights\last.pt"
```

Figure II-39 Detection command

- source — input path (file, folder path or camera id to use the webcam)
- weights — our trained weights path
- conf — confidence threshold (‘—conf ‘, we are using the default value)

We can see that our model detects faces at pretty high accuracy, drawing bounding boxes around the detected faces, showing a confidence score of about 0.93+ when facing the camera. We can see the different test images at Figure (ll-40).

The model is insensitive to camera and subject orientation, more resilient to lighting, doing surprisingly well at very low light situations.

The model is detecting the face when rotated horizontally at 90 degrees and also detecting when the top and bottom of the face is covered.

The YOLOv5 model is running on CPU, the inference speed on webcam is about **350ms** or about **3 frames per second**. The speed increases about x10 if used on a powerful GPU.

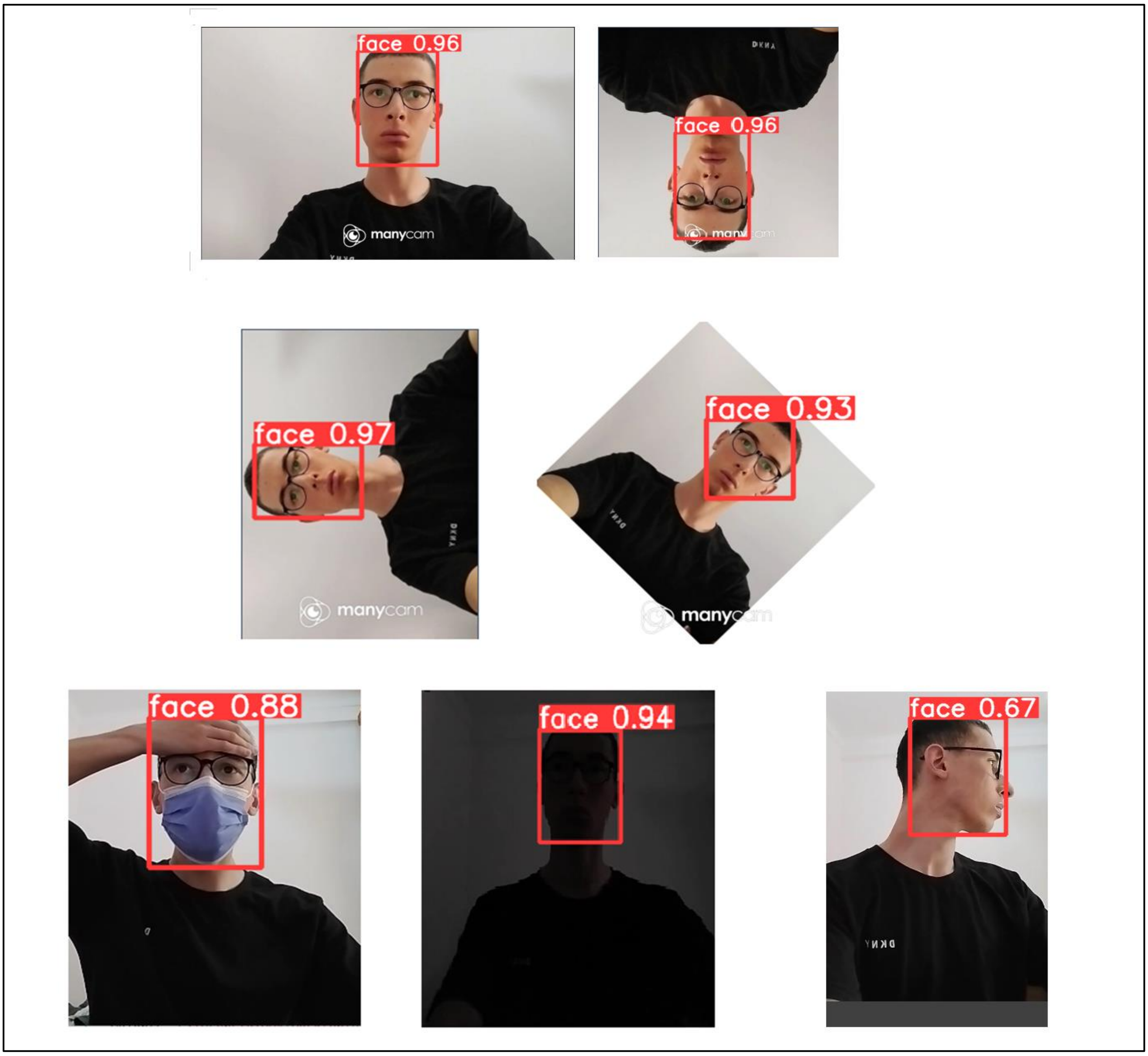
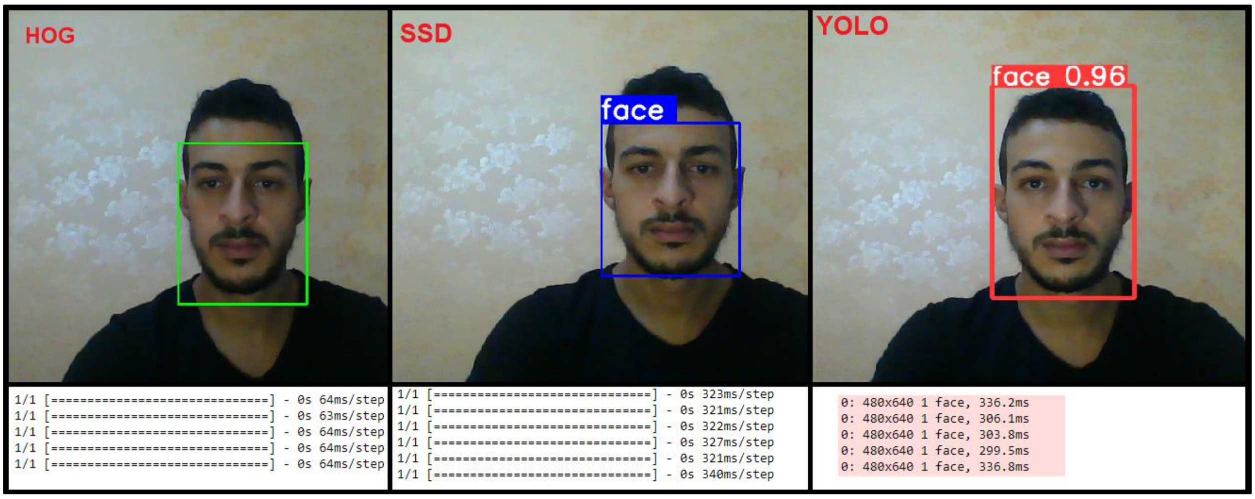


Figure II-40 YOLOv5 face detection Batch test

II.4.3 Face detection models comparison

We’re comparing between the SSD, YOLOv5 face detection models with a HOG-Based algorithm for face detection (Figure ll-41).



```
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 64ms/step
```

```
1/1 [==============================] - 0s 323ms/step
1/1 [==============================] - 0s 321ms/step
1/1 [==============================] - 0s 322ms/step
1/1 [==============================] - 0s 327ms/step
1/1 [==============================] - 0s 321ms/step
1/1 [==============================] - 0s 340ms/step
```

```
0: 480x640 1 face, 336.2ms
0: 480x640 1 face, 306.1ms
0: 480x640 1 face, 303.8ms
0: 480x640 1 face, 299.5ms
0: 480x640 1 face, 336.8ms
```

Figure II-41 real-time test

One can clearly see the differences between the models in the picture above (Figure ll-39), there are a little bit of differences in the BBox shapes and also the HOG-Based algorithm is way faster than the others.

II.5 Face recognition

II.5.1 One-shot learning

One-shot learning is a classification activity in which one or a few instances are used to classify a large number of future examples.

This describes tasks in the domain of face recognition, such as face identification and face verification, in which persons must be correctly identified using various facial expressions, lighting circumstances, accessories, and haircuts based on one or a few template photographs.

Modern face recognition systems solve the difficulty of one-shot learning by learning a rich low-dimensional feature representation as a face embedding, which can be quickly calculated and compared for verification and identification tasks.

Originally, embeddings were learned using a Siamese network for one-shot learning problems. Siamese networks that were trained with comparative loss functions performed better.

II.5.2 Siamese Network

The Siamese network is a network that has been popularized as a result of its usage in one-shot learning.

A Siamese network is a network architecture that combines the outputs of two concurrent neural networks, each of which takes a different input (Figure ll-42).

Input image: the image from the camera.

Validation image: image from the database.

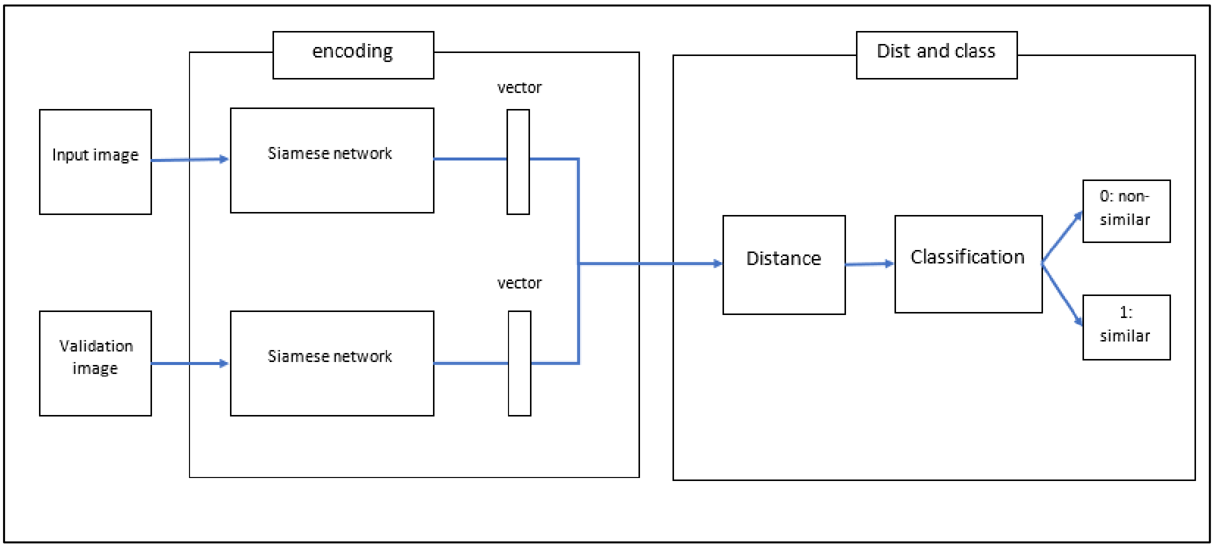


Figure II-42 the Siamese model

II.5.3 Distance calculation

The images go through the Siamese network then a flatten function to get a vector for each image.

Once we have our vectorized data, we can compare the two vectors using a distance function, depending on the input data, setting this threshold might be complex or time consuming.

$$D = vector1 - vector2$$

Equation II-7 distance calculation

II.5.4 Creating Dataset

To create a dataset, we need to get faces images we can use the program in the (Figure II-1) and get some random faces pictures, the most important is dividing the data to three folders:

- Anchor: images for our face.
- Positive: images for our face.
- Negative: random faces.

The point is training the model to recognize the similarity between the anchor and the positive images (Figure II-43).

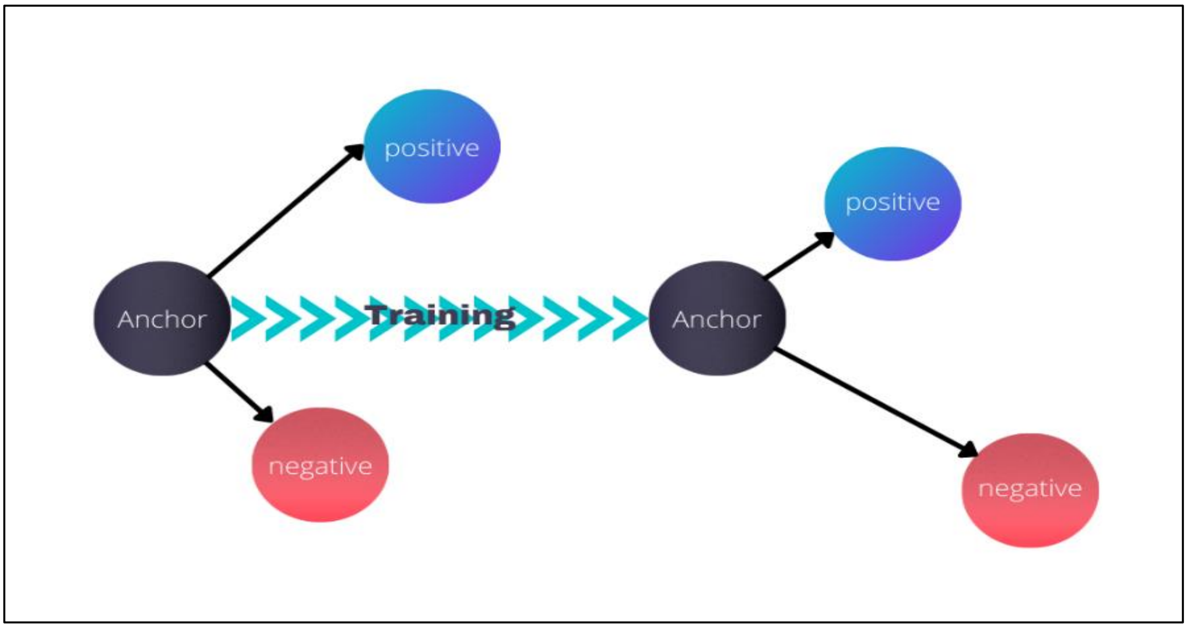
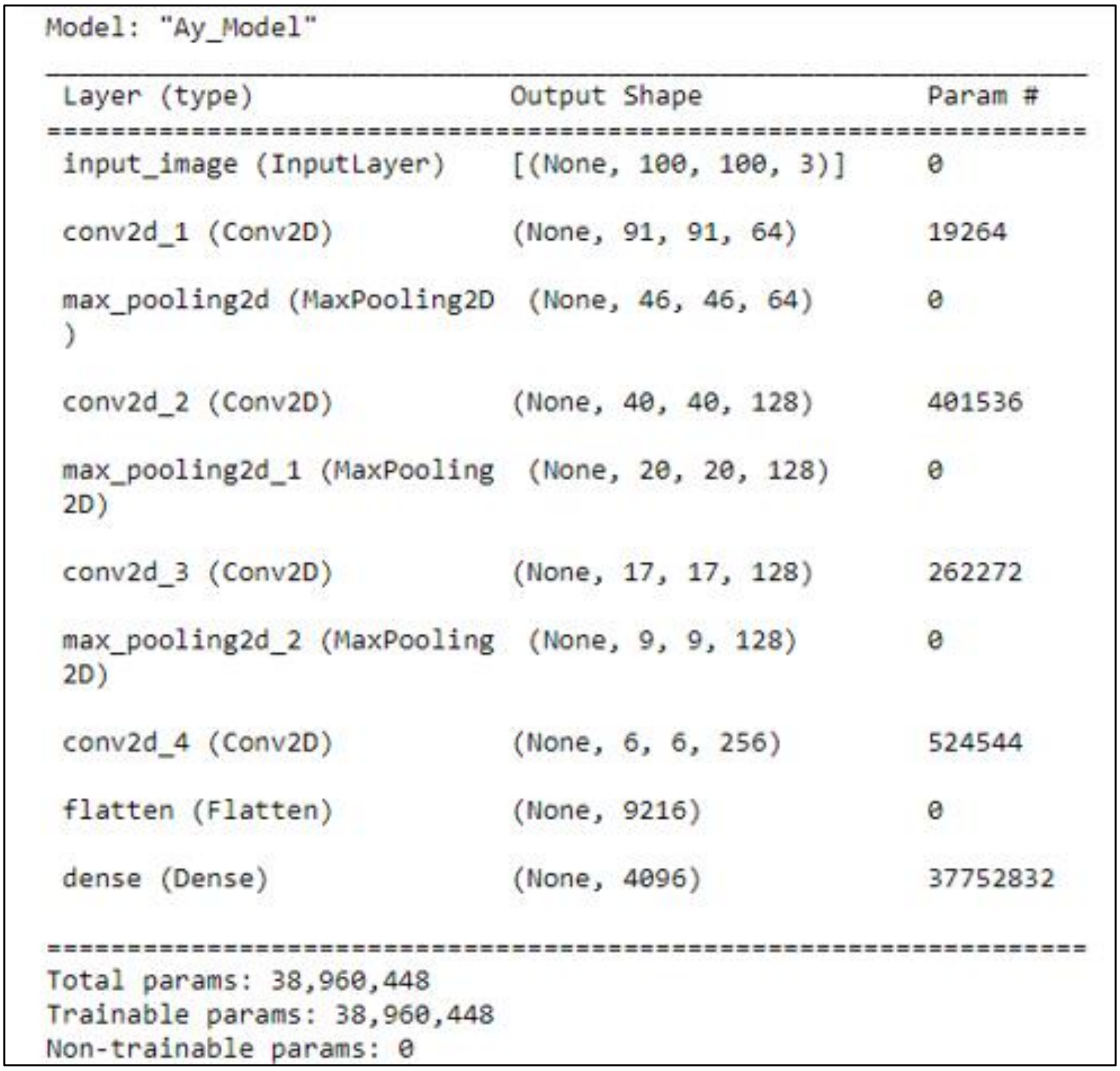


Figure II-43 training the Siamese Network

II.5.5 Building the model

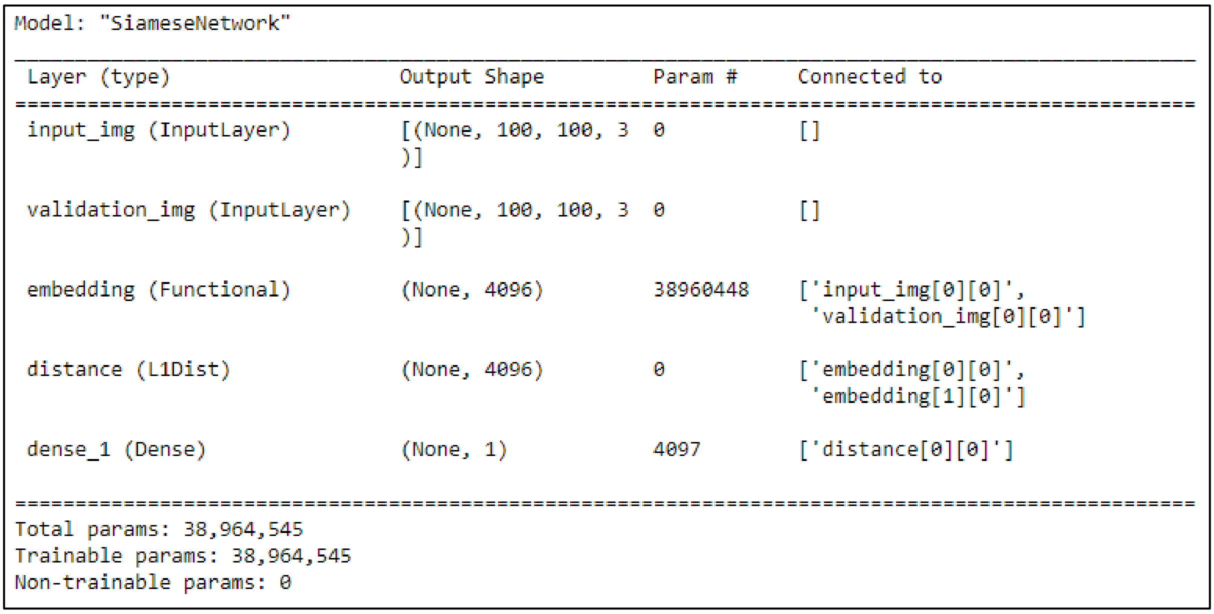
For building this model we use TensorFlow Keras-layers

First thing we create the Siamese network (Figure II-44) and the similarity function then combine them together (Figure II-45).



```
Model: "Ay_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_image (InputLayer)    [(None, 100, 100, 3)]     0

 conv2d_1 (Conv2D)           (None, 91, 91, 64)        19264

 max_pooling2d (MaxPooling2D  (None, 46, 46, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 40, 40, 128)       401536

 max_pooling2d_1 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 17, 17, 128)       262272

 max_pooling2d_2 (MaxPooling  (None, 9, 9, 128)        0
 2D)

 conv2d_4 (Conv2D)           (None, 6, 6, 256)         524544

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832

=================================================================
Total params: 38,960,448
Trainable params: 38,960,448
Non-trainable params: 0
```

Figure II-44 creating the Siamese Network



```
Model: "SiameseNetwork"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_img (InputLayer)         [(None, 100, 100, 3  0           []
                                )]

 validation_img (InputLayer)    [(None, 100, 100, 3  0           []
                                )]

 embedding (Functional)         (None, 4096)         38960448    ['input_img[0][0]',
                                                                  'validation_img[0][0]']

 distance (L1Dist)              (None, 4096)         0           ['embedding[0][0]',
                                                                  'embedding[1][0]']

 dense_1 (Dense)                (None, 1)            4097        ['distance[0][0]']

==================================================================================================
Total params: 38,964,545
Trainable params: 38,964,545
Non-trainable params: 0
```

Figure II-45 creating the whole model

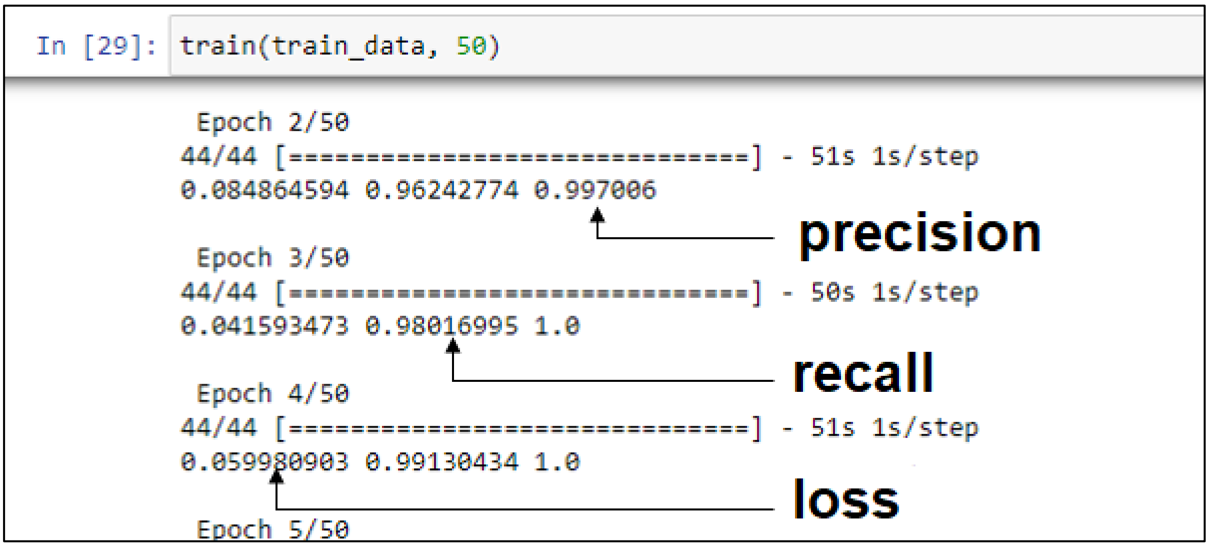
II.5.6 Training the model

For the training we're doing the same work as the SSD detection model. We have to create the training function and define the loss and the optimizer functions.

We choose the binary cross entropy for a loss function, and Adam optimizer as an optimizer function.

Finally, we upload the dataset to Google Colaboratory and train the models, the training takes 1h:30min.

We can read the precision and the loss when the model is training (Figure II-46)



```
In [29]: train(train_data, 50)

 Epoch 2/50
44/44 [==============================] - 51s 1s/step
0.084864594 0.96242774 0.997006

 Epoch 3/50
44/44 [==============================] - 50s 1s/step
0.041593473 0.98016995 1.0

 Epoch 4/50
44/44 [==============================] - 51s 1s/step
0.059980903 0.99130434 1.0

 Epoch 5/50
```

Figure II-46 Siamese model training

II.5.7 The recall and the precision

For testing we use those two functions to observe the progress of the training process.

II.5.7.1The recall

This metric creates two local variables, true positives and false negatives, that are used to compute the recall. This value is ultimately returned as recall, an idempotent operation that simply divides true positives by the sum of true positives and false negatives.

If sample weight is None, weights default to 1. Use sample weight of 0 to mask values.

If top_k is set, recall will be computed as how often on average a class among the labels of a batch entry is in the top-k predictions.

II.5.7.2The precision

The metric creates two local variables, true_positives and false_positives that are used to compute the precision. This value is ultimately returned as precision, an idempotent operation that simply divides true_positives by the sum of true_positives and false_positives.

If sample_weight is None, weights default to 1. Use sample_weight of 0 to mask values.

top_k is set, we'll calculate precision as how often on average a class among the top-k classes with the highest predicted values of a batch entry is correct and can be found in the label for that entry. [17]

II.5.8 Test the model.

For testing the model in real time, we must create small program using Open-CV to open camera and take a picture to verify if the model is working.

Before that we create a folder for validation images (images of ourselves) and input images (images from the camera).

Every time we launch the program it takes a picture using CV2 and save it in the input images path, then load the images to the model.

The model compares the image in the input folder with every image in the validation folder, when the similarity is higher than the verification threshold the output is true otherwise is false.

We divide the validation data to many folders (folder for each person), and the model loop through them all, to apply the model for many people (Figure Il-47).

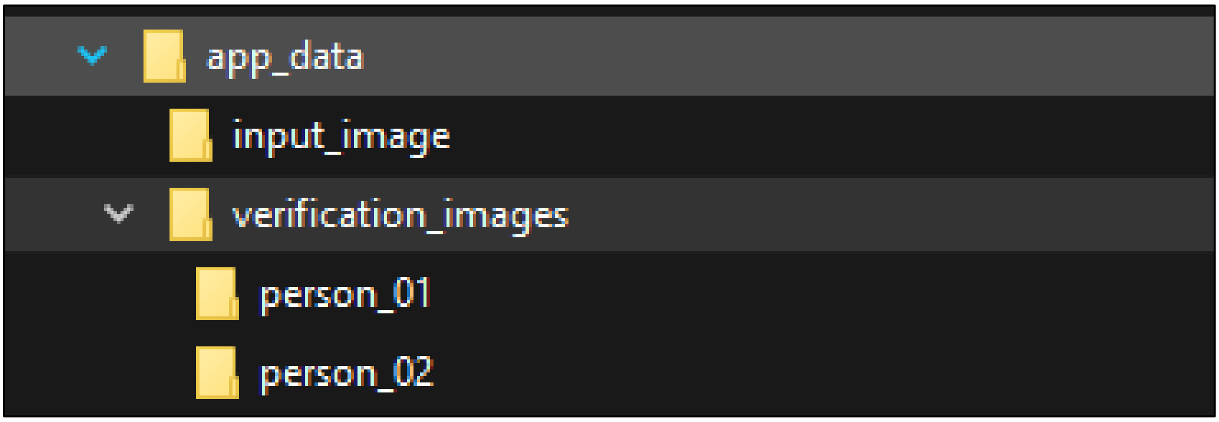

app_data
input_image
verification_images
person_01
person_02

Figure II-47 dividing Siamese application data

II.5.9 FaceNet for face recognition:

FaceNet provides a unified embedding for face recognition, verification and clustering tasks. It maps each face image into a euclidean space such that the distances in that space correspond to face similarity, i.e. an image of person A will be placed closer to all the other images of person A as compared to images of any other person present in the dataset.

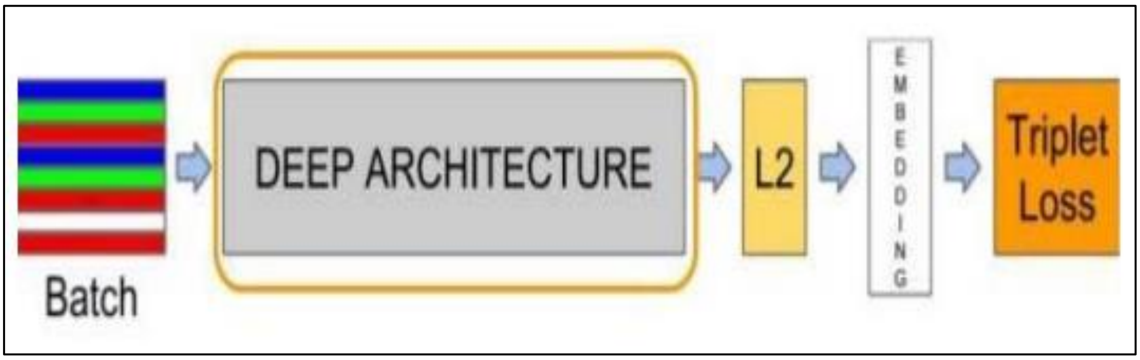


Figure II-48 model structure of facenet

(Figure ll-46) shows the structural model used in FaceNet. Facenet Model structure consists of a batch input layer and a deep CNN followed by L2 normalization, which results in the face embedding. This is followed by the triplet loss during training. In the training process the triplet loss minimizes the distance between an anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity. (Figure ll-49)

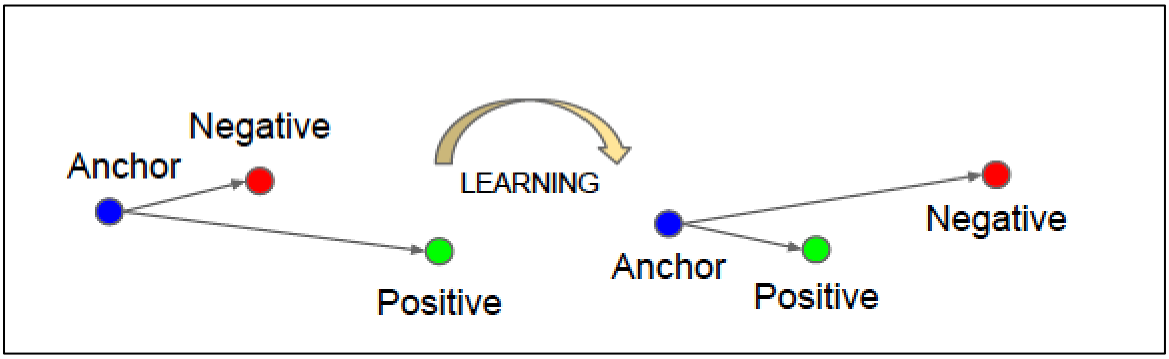


Figure II-49 Triplet Loss

We will use a pre-trained FaceNet model, It was trained on MS-Celeb-1M dataset, the dataset contains 1 million images of celebrities.

then use the Euclidean distance and a predefined threshold to determine the difference between the target face and that of a person in the database.

The face whose difference is minimal is considered to be the person belonging to the face database. The minimum difference must be smaller than that of the threshold to consider him as a similar person. When the difference is greater, the person is considered unknown. (Figure ll-50)

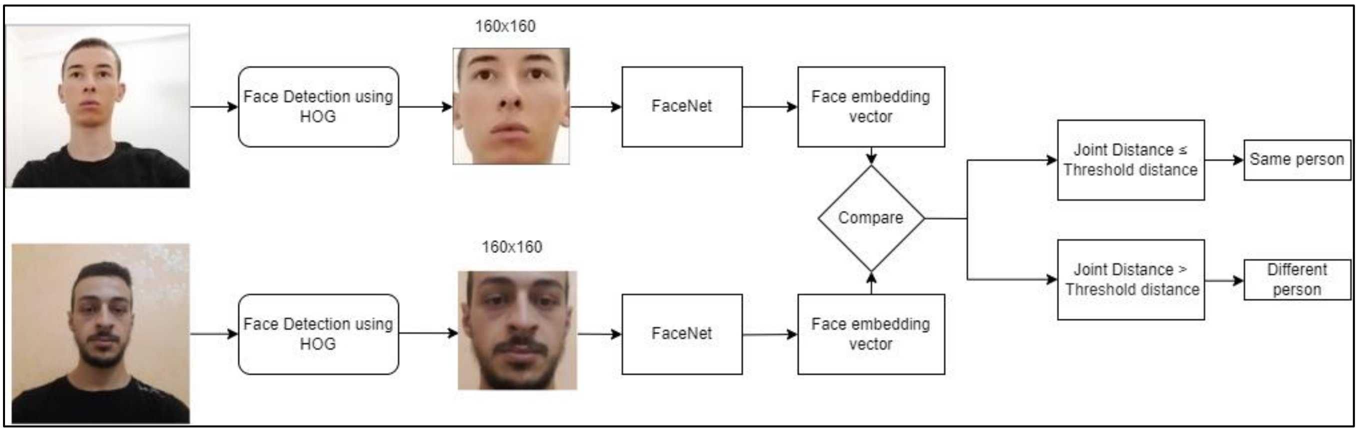


Figure II-50 Classification process with FaceNet and HOG.

II.5.9.1 Testing:

We’re going to use the one shot-learning approach, where we’ll be using just 1 picture to recognize a face, the test image is named after the name of the person, the image could have a random shape, size, lighting and background.

To generate the face embeddings, we need to obtain the face only and pass it as input threw the FaceNet model, to do that we need a face detection algorithm, we used Dlib's HOG face detector which is pretty accurate and computationally efficient which works great for real time usage.

To classify non-registered faces we have two methods, we can use a predefined threshold to classify them as unknown, the face is considered as unknown if the distance is greater than the threshold. (Figure ll-51)

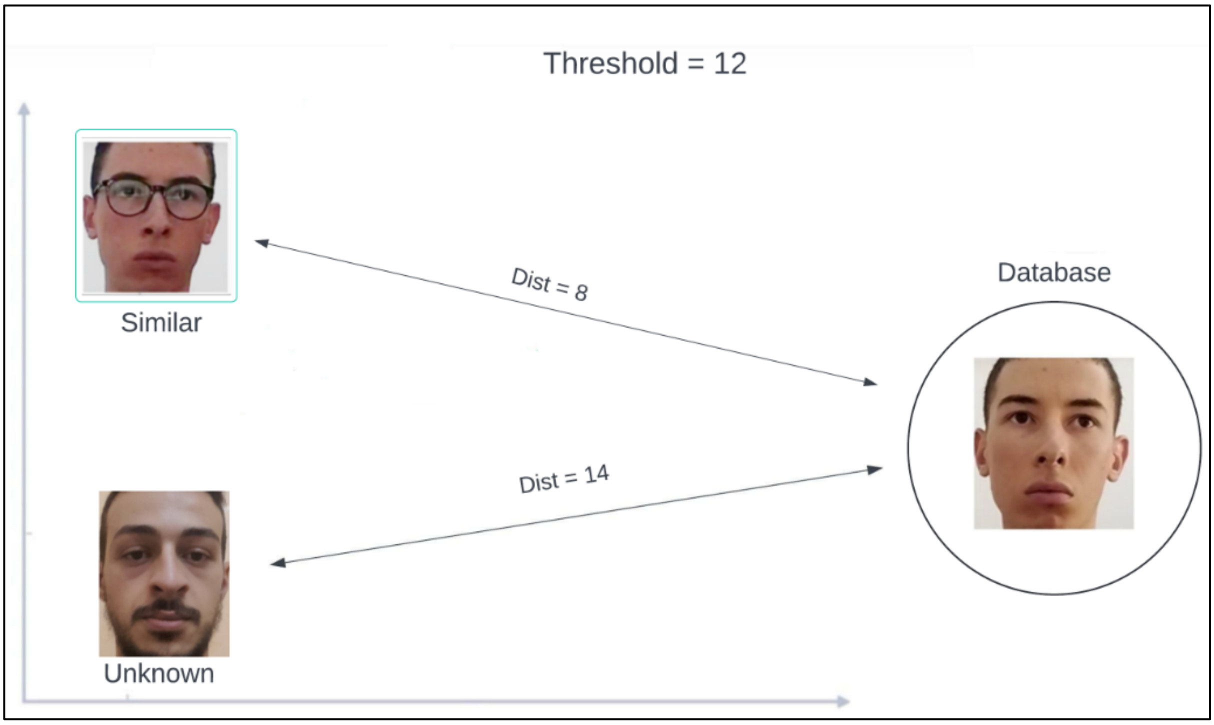


Figure II-51 Classifying with threshold

The second method is adding an image that does not contain a face and name it "unknown". after we add that image to the database, the distance of non-registered faces would be closer to the vector of the unknown image, it's for classifying unknown faces.

Our database consists of 2 images so far, my image and a non-face image for unknown faces. (Figure ll-52)

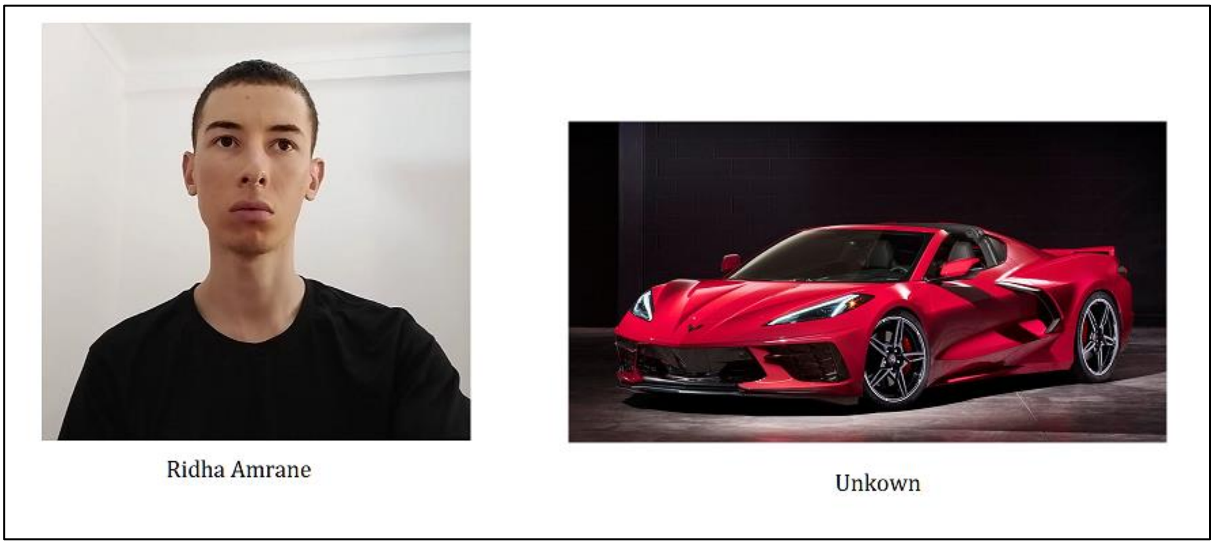
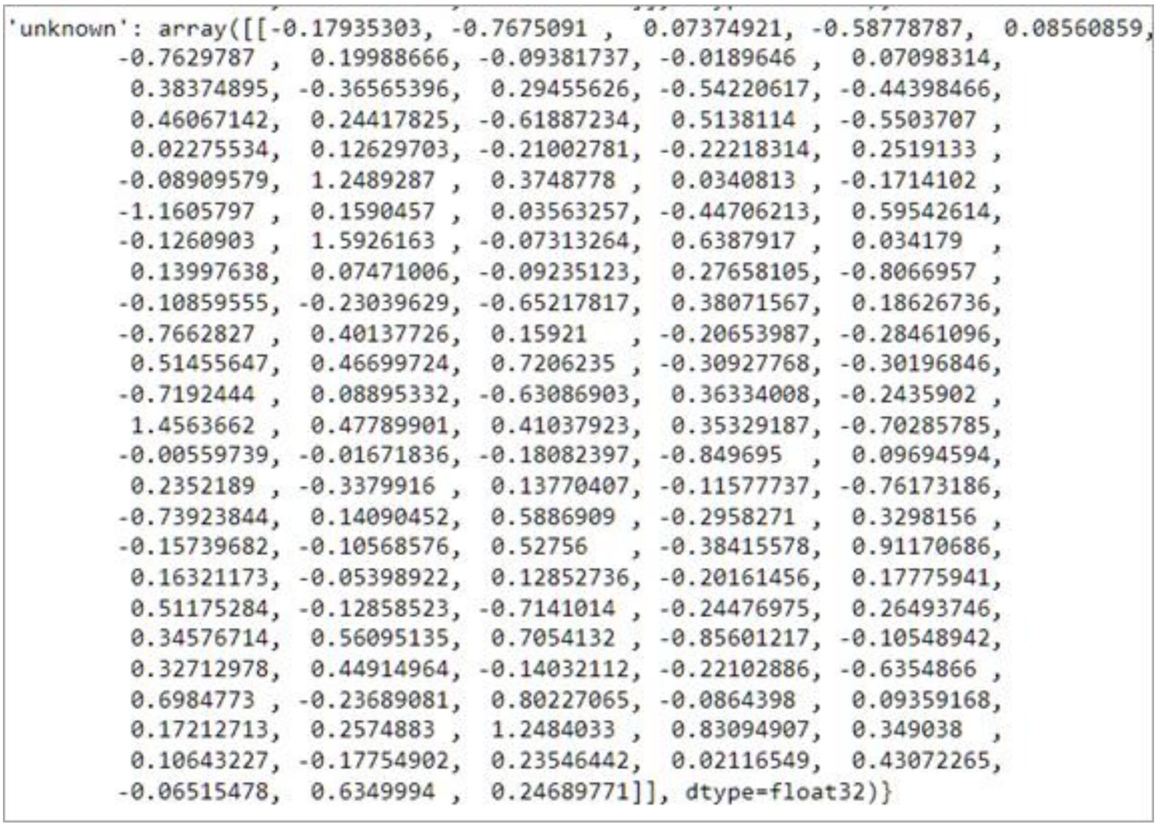


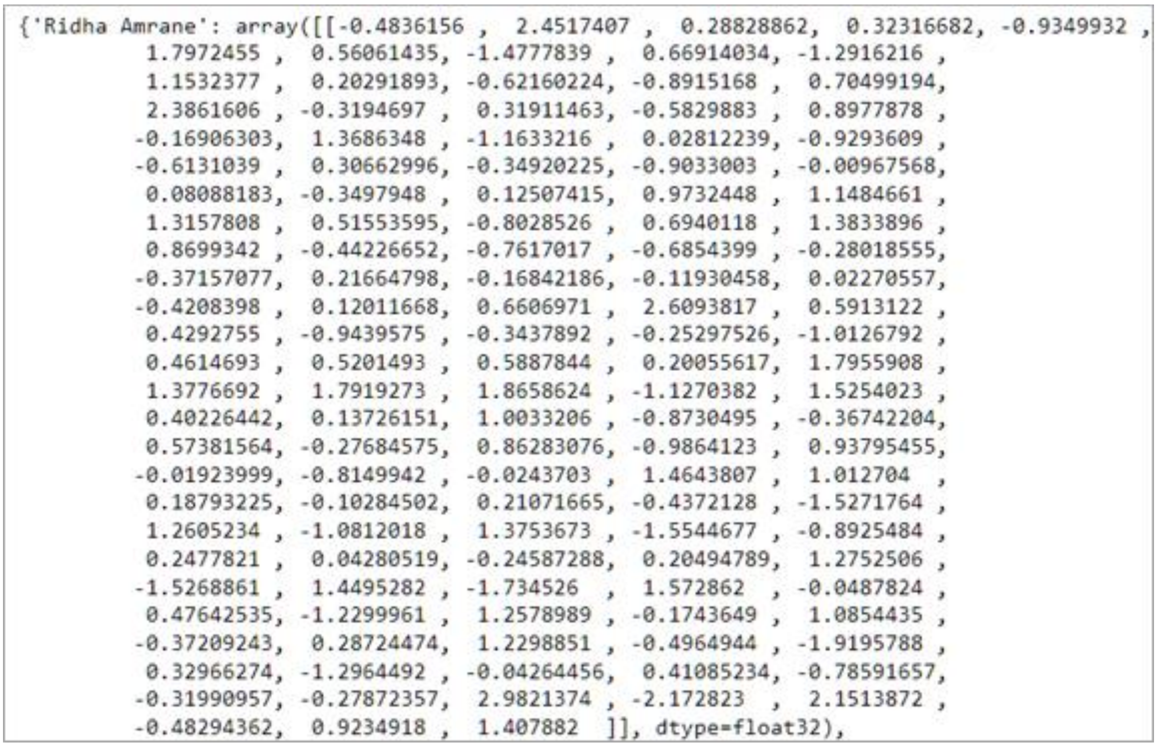
Figure II-52 Database

After adding images to database, we will create the face embeddings or also called face signatures. First, we detect the face only using HOG face detector, resize it to 160 by 160, in the case of the car there is no face, so we resize the whole image to 160 by 160. After doing so we pass the resized images to FaceNet which will give us a vector of 128 numbers for each image (embeddings). (Figure Il-53)



```
'unknown': array([[-0.17935303, -0.7675091 ,  0.07374921, -0.58778787,  0.08560859,
       -0.7629787 ,  0.19988666, -0.09381737, -0.0189646 ,  0.07098314,
        0.38374895, -0.36565396,  0.29455626, -0.54220617, -0.44398466,
        0.46067142,  0.24417825, -0.61887234,  0.5138114 , -0.5503707 ,
        0.02275534,  0.12629703, -0.21002781, -0.22218314,  0.2519133 ,
       -0.08909579,  1.2489287 ,  0.3748778 ,  0.0340813 , -0.1714102 ,
       -1.1605797 ,  0.1590457 ,  0.03563257, -0.44706213,  0.59542614,
       -0.1260903 ,  1.5926163 , -0.07313264,  0.6387917 ,  0.034179  ,
        0.13997638,  0.07471006, -0.09235123,  0.27658105, -0.8066957 ,
       -0.10859555, -0.23039629, -0.65217817,  0.38071567,  0.18626736,
       -0.7662827 ,  0.40137726,  0.15921    , -0.20653987, -0.28461096,
        0.51455647,  0.46699724,  0.7206235 , -0.30927768, -0.30196846,
       -0.7192444 ,  0.08895332, -0.63086903,  0.36334008, -0.2435902 ,
        1.4563662 ,  0.47789901,  0.41037923,  0.35329187, -0.70285785,
       -0.00559739, -0.01671836, -0.18082397, -0.849695  ,  0.09694594,
        0.2352189 , -0.3379916 ,  0.13770407, -0.11577737, -0.76173186,
       -0.73923844,  0.14090452,  0.5886909 , -0.2958271 ,  0.3298156 ,
       -0.15739682, -0.10568576,  0.52756   , -0.38415578,  0.91170686,
        0.16321173, -0.05398922,  0.12852736, -0.20161456,  0.17775941,
        0.51175284, -0.12858523, -0.7141014 , -0.24476975,  0.26493746,
        0.34576714,  0.56095135,  0.7054132 , -0.85601217, -0.10548942,
        0.32712978,  0.44914964, -0.14032112, -0.22102886, -0.6354866 ,
        0.6984773 , -0.23689081,  0.80227065, -0.0864398 ,  0.09359168,
        0.17212713,  0.2574883 ,  1.2484033 ,  0.83094907,  0.349038  ,
        0.10643227, -0.17754902,  0.23546442,  0.02116549,  0.43072265,
       -0.06515478,  0.6349994 ,  0.24689771]], dtype=float32)}
```

Figure II-53 Unknown embedding



```
{'Ridha Amrane': array([[-0.4836156 ,  2.4517407 ,  0.28828862,  0.32316682, -0.9349932 ,
          1.7972455 ,  0.56061435, -1.4777839 ,  0.66914034, -1.2916216 ,
          1.1532377 ,  0.20291893, -0.62160224, -0.8915168 ,  0.70499194,
          2.3861606 , -0.3194697 ,  0.31911463, -0.5829883 ,  0.8977878 ,
         -0.16906303,  1.3686348 , -1.1633216 ,  0.02812239, -0.9293609 ,
         -0.6131039 ,  0.30662996, -0.34920225, -0.9033003 , -0.00967568,
          0.08088183, -0.3497948 ,  0.12507415,  0.9732448 ,  1.1484661 ,
          1.3157808 ,  0.51553595, -0.8028526 ,  0.6940118 ,  1.3833896 ,
          0.8699342 , -0.44226652, -0.7617017 , -0.6854399 , -0.28018555,
         -0.37157077,  0.21664798, -0.16842186, -0.11930458,  0.02270557,
         -0.4208398 ,  0.12011668,  0.6606971 ,  2.6093817 ,  0.5913122 ,
          0.4292755 , -0.9439575 , -0.3437892 , -0.25297526, -1.0126792 ,
          0.4614693 ,  0.5201493 ,  0.5887844 ,  0.20055617,  1.7955908 ,
          1.3776692 ,  1.7919273 ,  1.8658624 , -1.1270382 ,  1.5254023 ,
          0.40226442,  0.13726151,  1.0033206 , -0.8730495 , -0.36742204,
          0.57381564, -0.27684575,  0.86283076, -0.9864123 ,  0.93795455,
         -0.01923999, -0.8149942 , -0.0243703 ,  1.4643807 ,  1.012704  ,
          0.18793225, -0.10284502,  0.21071665, -0.4372128 , -1.5271764 ,
          1.2605234 , -1.0812018 ,  1.3753673 , -1.5544677 , -0.8925484 ,
          0.2477821 ,  0.04280519, -0.24587288,  0.20494789,  1.2752506 ,
         -1.5268861 ,  1.4495282 , -1.734526  ,  1.572862  , -0.0487824 ,
          0.47642535, -1.2299961 ,  1.2578989 , -0.1743649 ,  1.0854435 ,
         -0.37209243,  0.28724474,  1.2298851 , -0.4964944 , -1.9195788 ,
          0.32966274, -1.2964492 , -0.04264456,  0.41085234, -0.78591657,
         -0.31990957, -0.27872357,  2.9821374 , -2.172823  ,  2.1513872 ,
         -0.48294362,  0.9234918 ,  1.407882  ]], dtype=float32),
```

Figure II-54 known face embedding

Now that we have our Embeddings, we can start recognizing, to do that we get a test image (Figure ll-55) and extract the face embedding the same way we did to our database, by using HOG face detector, resizing to 160 by 160 and passing it threw FaceNet.

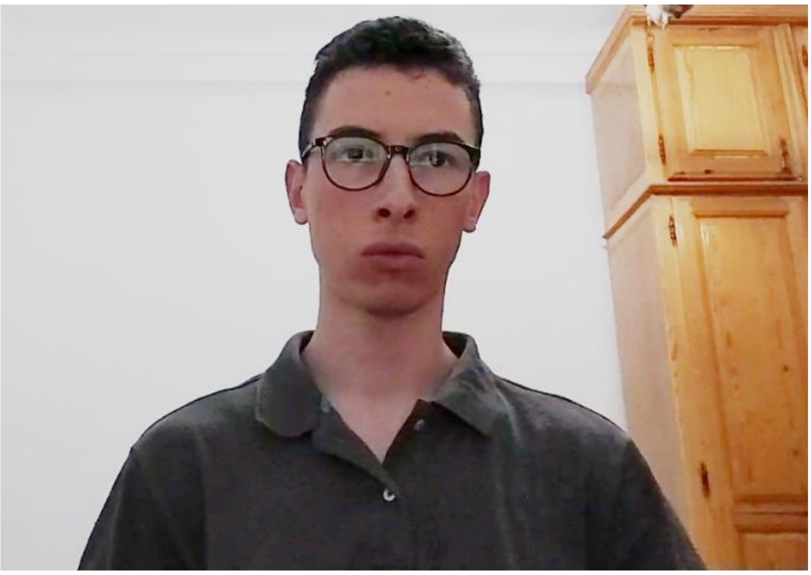


Figure II-55 Test image

Now we calculate the distances of the embeddings in our database with the embedding of the test image, the closest embedding distance in the database will be identified as similar with the test image. (Figure ll-56)

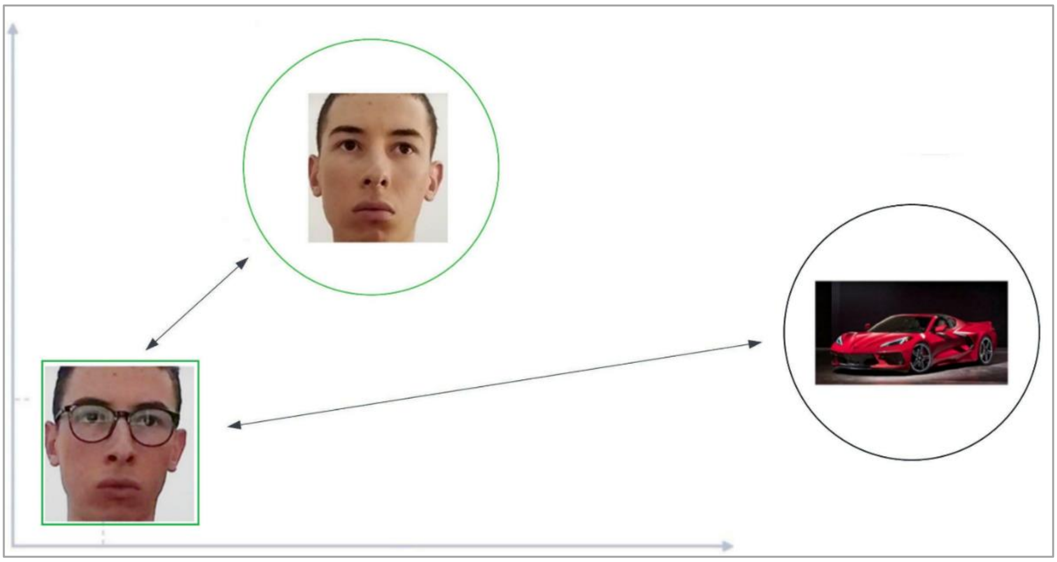


Figure II-56 calculating embedding distances.

Now let’s see the output on our python code when we compare the two images, the python code draws a bounding box around the face and the name of the closest embedding distance. (Figure II-57)

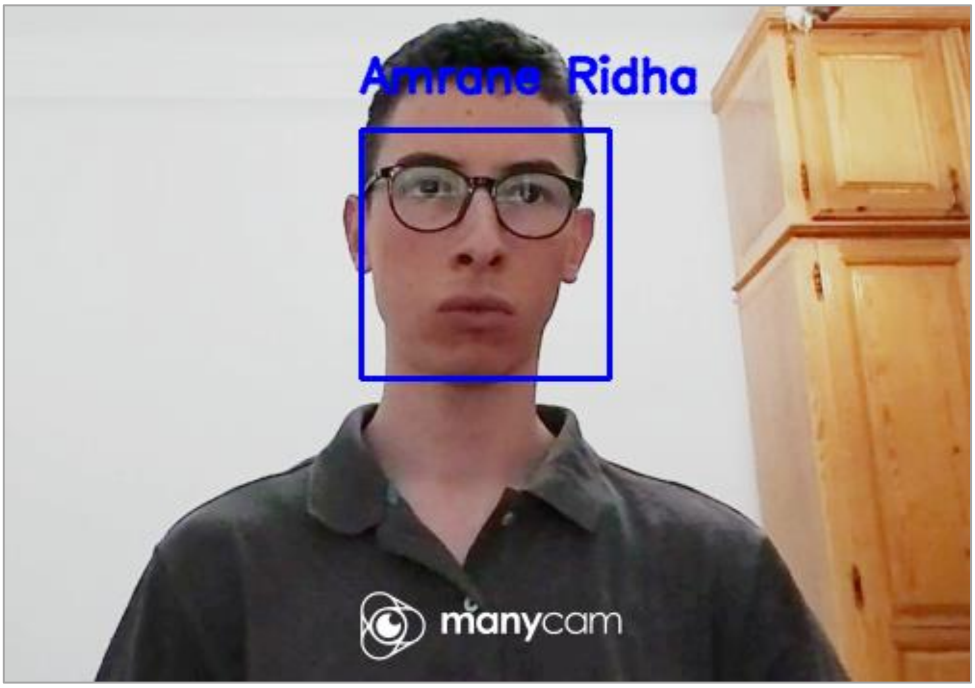


Figure II-57 similar face

Now let’s do the same for another test image, which doesn’t have a similar image in the dataset (Figure ll-58).

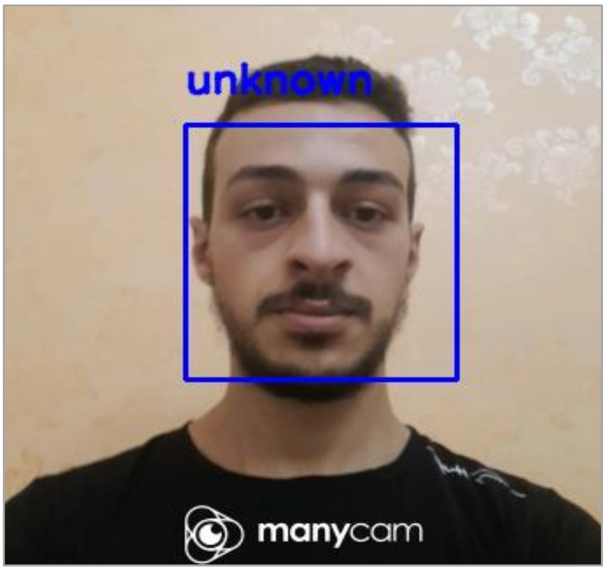


Figure II-58 Unknown face

In the case of my partners test image (Fig II-58), the closest embedding distance was the embedding of the unknown image. Now to test the accuracy of the model even more, we're going to add more images to the database of random celebrities (Figure II-59), and do the test in different type of lighting.

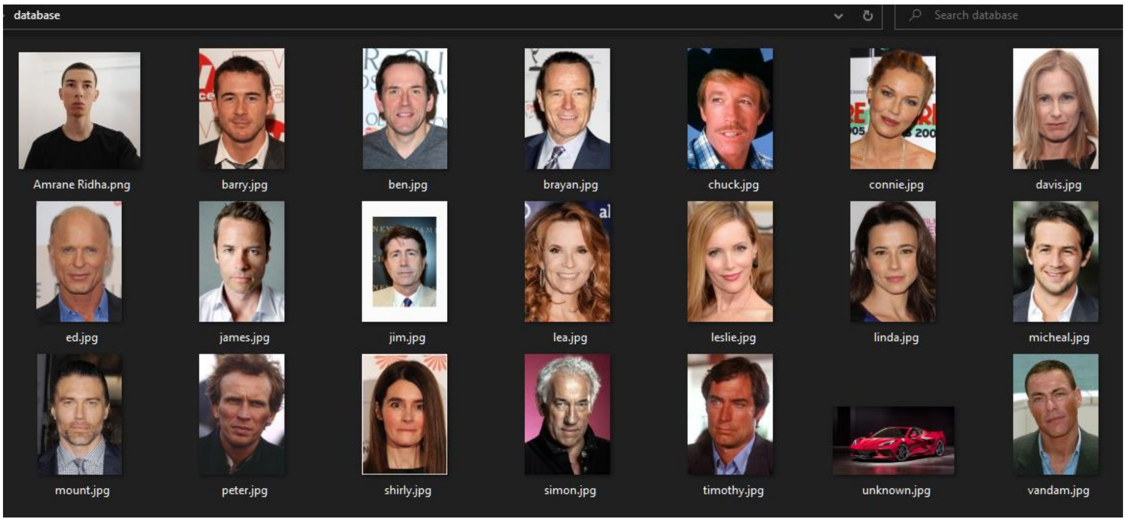


Figure II-59 New Database

Now let's see the output when I compare myself to the database. (Figure II-60)

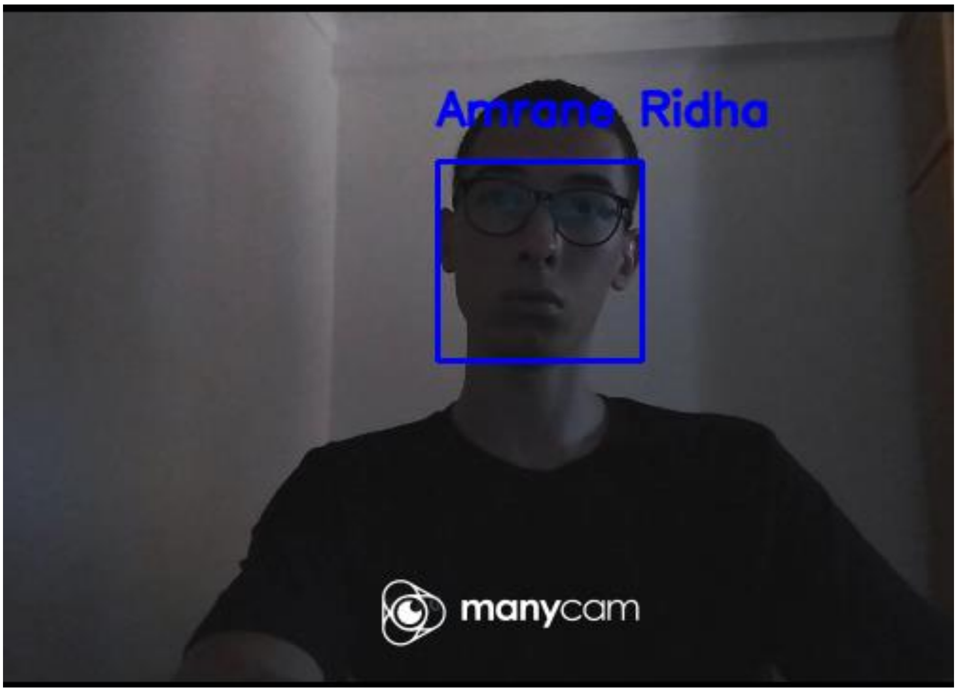
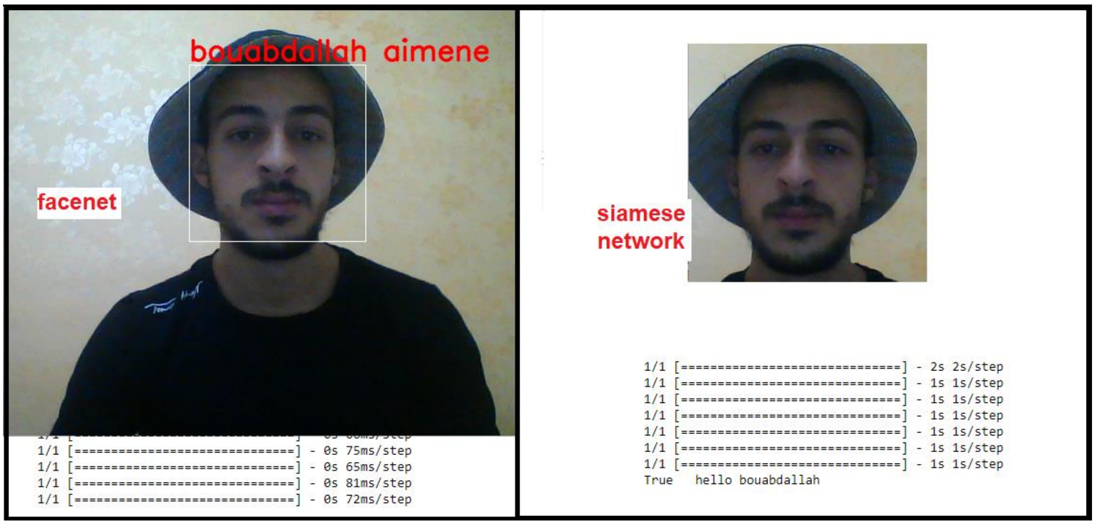


Figure II-60 Testing in low light

Even in low light situation with a heavy database, the model recognizes me correctly.

II.6 Face recognition models comparison



```
1/1 [=============================] - 0s 75ms/step
1/1 [=============================] - 0s 65ms/step
1/1 [=============================] - 0s 81ms/step
1/1 [=============================] - 0s 72ms/step
```

```
1/1 [=============================] - 2s 2s/step
1/1 [=============================] - 1s 1s/step
1/1 [=============================] - 1s 1s/step
1/1 [=============================] - 1s 1s/step
1/1 [=============================] - 1s 1s/step
1/1 [=============================] - 1s 1s/step
1/1 [=============================] - 1s 1s/step
True   hello bouabdallah
```

Figure II-61 Facenet model vs Siamese model

As we see in the (Figure ll-61) the FaceNet model is faster than the Siamese network model. The Siamese network spent 1 sec with a single image and to get a good accuracy u need more than a picture for one person (many pictures with different face positions), unlike the FaceNet one picture is enough and it is way faster.

II.7 Conclusion

the SSD model is able to detect faces correctly with acceptable accuracy, but it is taking a long time, up to 500ms per step. the same thing for the YOLOv5 and that will be a big problem when we combine it with the recognition model.

Extracting the bounding box coordinates using the face detection model to get the face image then load the face image to the Siamese network to recognize it will make the system really slow.

those models are great but creating a face identification system from scratch can be hard a little bit because you need a massive dataset and high-performance setup to get a fast model with high accuracy, and that’s why we’re using a pretrained FaceNet model combined with the HOG algorithm for face detection and apply them for our attendance system.

III. Chapter 3: Application

Chapter III: Application

III.1 Introduction

In this chapter we expose the steps to create the attendance system using the Dlib's HOG and FaceNet model.

From the results of the comparisons in the previous chapter, we observed that HOG algorithm is really fast in detecting faces than the other models and in the matter of the face recognition we’re choosing the FaceNet model for the same reasons.

Finally, for the user application we’re creating a graphical user interface to simplify the access to the attendance lists and for adding new students in the database.

III.2 work environment

- Laptop DELL Latitude E5440.
- Processor Intel Core i7 4600U @ 2.60 GHZ.
- Graphical Card GT-720M.
- Installed memory (RAM) 6.00 GB.
- Windows 10 pro 64 bit.
- Jupyter notebook
- Python 3.7.4

III.3 Libraries

III.3.1 Face_Recognition

Detect, recognize and manipulate faces from Python using dlib’s face detection algorithm HOG and state of the art face recognition models.

III.3.2 Tkinter

Tk is the only cross-platform (Windows, Mac, Unix) graphical user interface toolkit designed exclusively for high-level dynamic languages, like Python, Tcl, Ruby, Perl, and many others. Whatever language you use, this site brings you the current, high-quality essential information you need to get the most out of Tk. [18]

III.3.3 PIL

The Python Imaging Library adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.

The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool. [19]

III.3.4 pickle

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.[20]

III.3.5 datetime

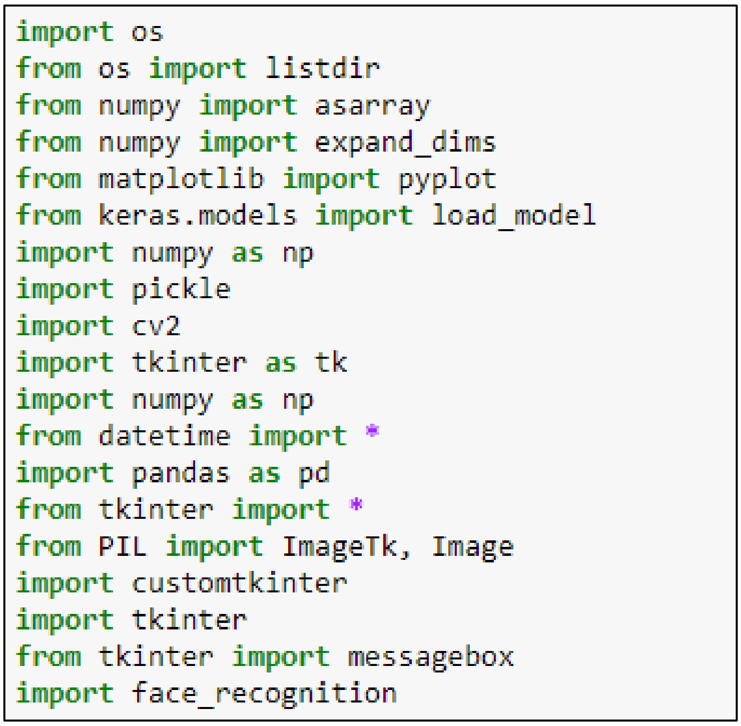
The datetime module supplies classes for manipulating dates and times, the most of the libraries are described in the second chapter.

III.4 Collecting students’ images

First of all, we need to create a database which contain images, each image is named after the name of the student.

III.5 Collecting Face Signatures

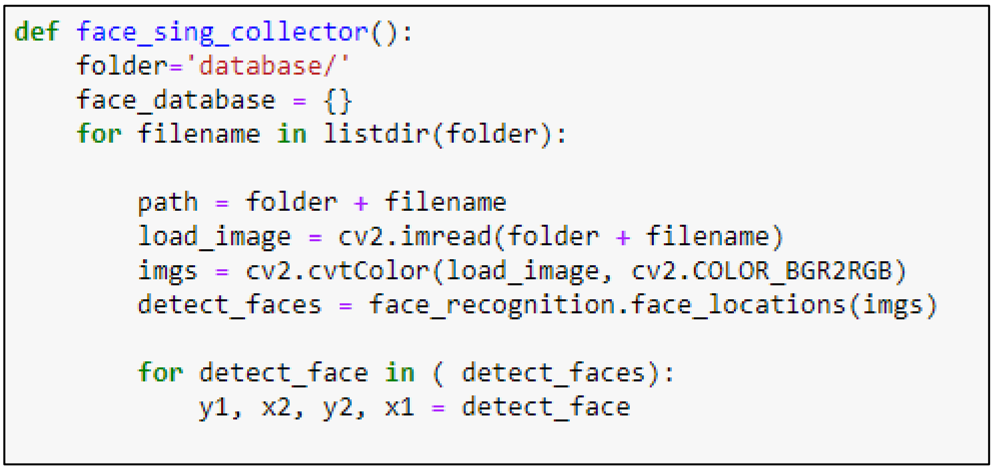
After creating the database, in the face signatures program we have to call some important libraries: OS, PIL, matplotlib, TensorFlow, tensorflow.keras.models, numpy, face_recognition, pickle and OpenCV (Figure III-1).



```python
import os
from os import listdir
from numpy import asarray
from numpy import expand_dims
from matplotlib import pyplot
from keras.models import load_model
import numpy as np
import pickle
import cv2
import tkinter as tk
import numpy as np
from datetime import *
import pandas as pd
from tkinter import *
from PIL import ImageTk, Image
import customtkinter
import tkinter
from tkinter import messagebox
import face_recognition
```

Figure III-1 import all the libraries

The program loop through the database to detect and extract faces from those images using the HOG face detector, so we don't have to get the picture of the face of every student, the face extraction for the data base happens automatically(Figure lll-2).

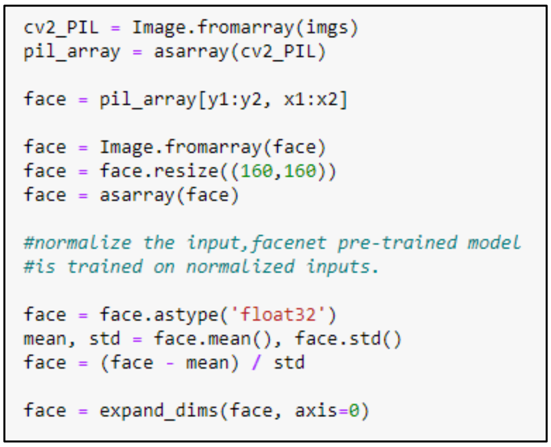


```python
def face_sing_collector():
    folder='database/'
    face_database = {}
    for filename in listdir(folder):

        path = folder + filename
        load_image = cv2.imread(folder + filename)
        imgs = cv2.cvtColor(load_image, cv2.COLOR_BGR2RGB)
        detect_faces = face_recognition.face_locations(imgs)

        for detect_face in ( detect_faces):
            y1, x2, y2, x1 = detect_face
```

Figure III-2 Face extraction

Then the face image went through a little bit of pre-processing steps like resize and type changing (Figure III-3):

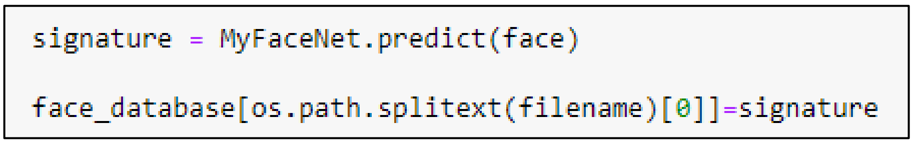
- Resize image to 160*160 pixel.
- Normalize the face.
- Convert image type to float32.
- Convert the image from 3 dimensions shape (160 x 160 x 3) to 4 dimensions shape (1 x 160 x 160 x 3), 160x160 for the height and width, 3 for RGB format and 1 for number of faces in the image could be 2, 3 or more.



```python
cv2_PIL = Image.fromarray(imgs)
pil_array = asarray(cv2_PIL)

face = pil_array[y1:y2, x1:x2]

face = Image.fromarray(face)
face = face.resize((160,160))
face = asarray(face)

#normalize the input,facenet pre-trained model
#is trained on normalized inputs.

face = face.astype('float32')
mean, std = face.mean(), face.std()
face = (face - mean) / std

face = expand_dims(face, axis=0)
```

Figure III-3 images preprocessing

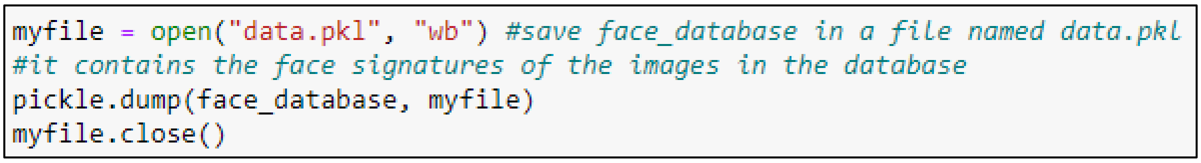
The next step is loading the normalized face as input into the face recognition model (Facenet) and get the face embedding as output which is a vector of 128 number, these numbers represent the most important features in the face (Figure III-4).



```python
signature = MyFaceNet.predict(face)

face_database[os.path.splitext(filename)[0]]=signature
```

Figure III-4 signature extraction

Finally, save the embedding of every face from the database attached with the student's name in a pickle file (data.pkl) like the picture shows (Figure III-5).



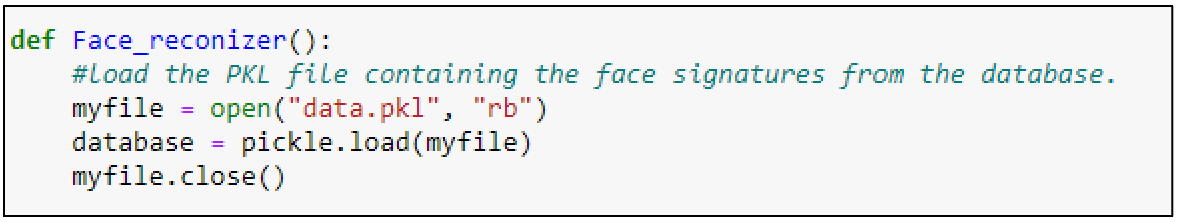
```python
myfile = open("data.pkl", "wb") #save face_database in a file named data.pkl
#it contains the face signatures of the images in the database
pickle.dump(face_database, myfile)
myfile.close()
```

Figure III-5 creating the face embedding database

III.6 Face recognition

Starting, we call some the libraries: PIL, keras.models, numpy, pickle, OpenCV, datetime and face_recognition, same as the face signatures collecting program (Figure III-1).

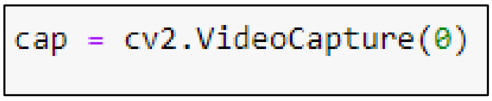
We load the facenet model again, then open the face embeddings file (data.pkl) (Figure III-6).



```
def Face_reconizer():
    #load the PKL file containing the face signatures from the database.
    myfile = open("data.pkl", "rb")
    database = pickle.load(myfile)
    myfile.close()
```

Figure III-6 load the models and open the data files

Next step is moving to the real-time face detection and recognition by opening the camera using OpenCV library(Figure lll-7).

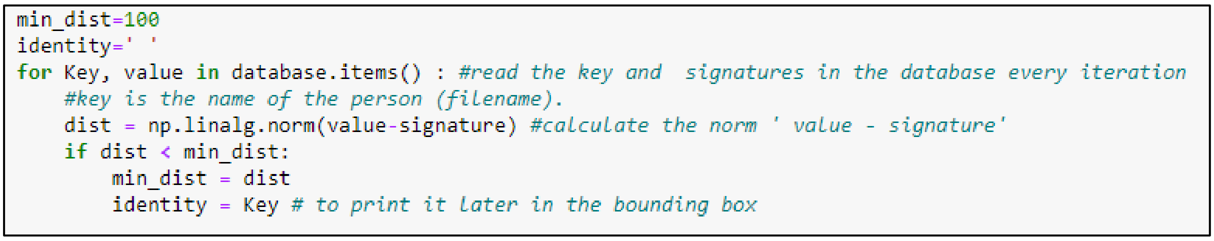


```
cap = cv2.VideoCapture(0)
```

Figure III-7 open the camera

The HOG face detector detects the faces frame by frame from the camera and load it to the FaceNet model, the FaceNet model creates the face embeddings after going through the same pre-processing steps and the conversions we did to the database of the faces images in the collecting signatures part (Figure ll-3).

Finally, we compare the face embeddings from the database with the face embeddings taken from the real-time camera (Figure III-8) if the distance of the face embeddings is lower than the min_dist we assume the faces are similar.

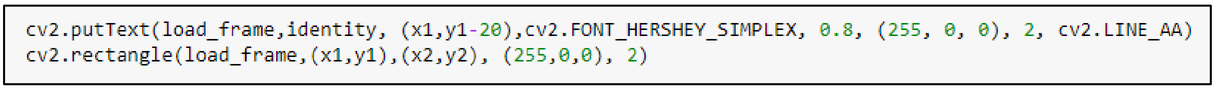


```
min_dist=100
identity=' '
for Key, value in database.items() : #read the key and  signatures in the database every iteration
    #key is the name of the person (filename).
    dist = np.linalg.norm(value-signature) #calculate the norm ' value - signature'
    if dist < min_dist:
        min_dist = dist
        identity = Key # to print it later in the bounding box
```

Figure III-8 similarity calculation

III.7 Creating bounding box

After getting the face coordinates from the HOG face detection model, we use them to draw the bounding box and print the identity name above the BBox (Figure III-9).

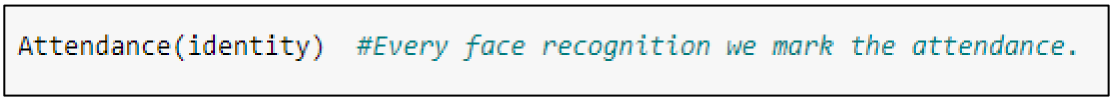


```
cv2.putText(load_frame,identity, (x1,y1-20),cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2, cv2.LINE_AA)
cv2.rectangle(load_frame,(x1,y1),(x2,y2), (255,0,0), 2)
```

Figure III-9 create the BBox and print the id

III.8 The attendance

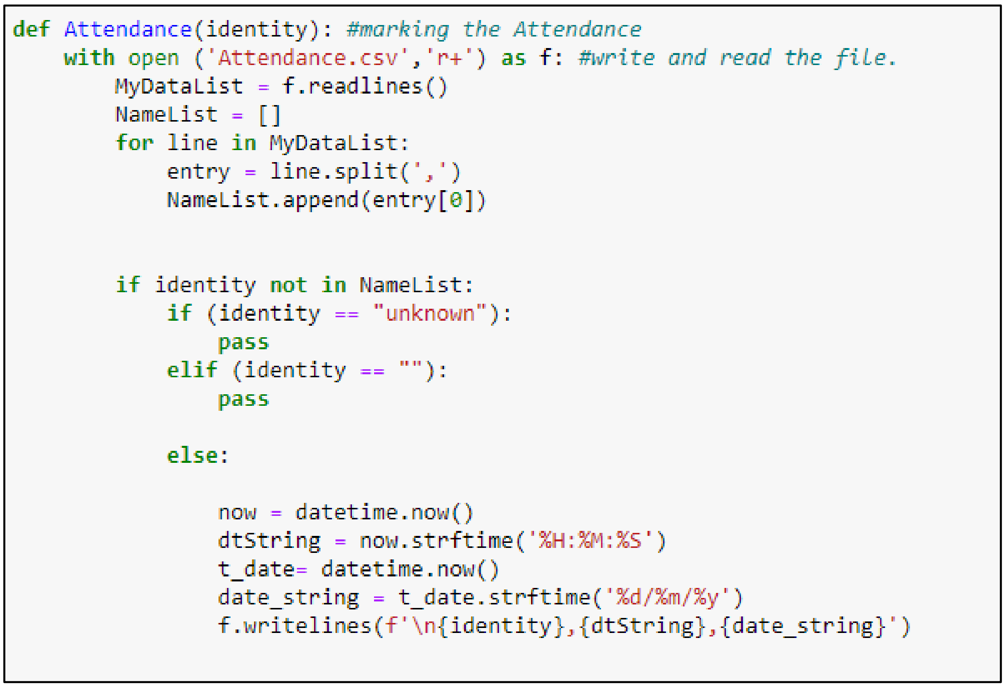
Every time the program recognizes someone it puts his name in Excel file (.CSV) with the date and time(Figure lll-10).



```
Attendance(identity)  #Every face recognition we mark the attendance.
```

Figure III-10 mark the attendance

The attendance function: (Figure III-11).



```
def Attendance(identity): #marking the Attendance
    with open ('Attendance.csv','r+') as f: #write and read the file.
        MyDataList = f.readlines()
        NameList = []
        for line in MyDataList:
            entry = line.split(',')
            NameList.append(entry[0])


        if identity not in NameList:
            if (identity == "unknown"):
                pass
            elif (identity == ""):
                pass

            else:

                now = datetime.now()
                dtString = now.strftime('%H:%M:%S')
                t_date= datetime.now()
                date_string = t_date.strftime('%d/%m/%y')
                f.writelines(f'\n{identity},{dtString},{date_string}')
```

Figure III-11 the attendance function

III.9 The graphical user interface (GUI)

The graphical user interface is put together using Tkinter and CustomTkinter libraries for python, the application is composed of 3 interfaces, the first interface is the home page, this is the main interface in which you can see the main features available (Figure lll-12).

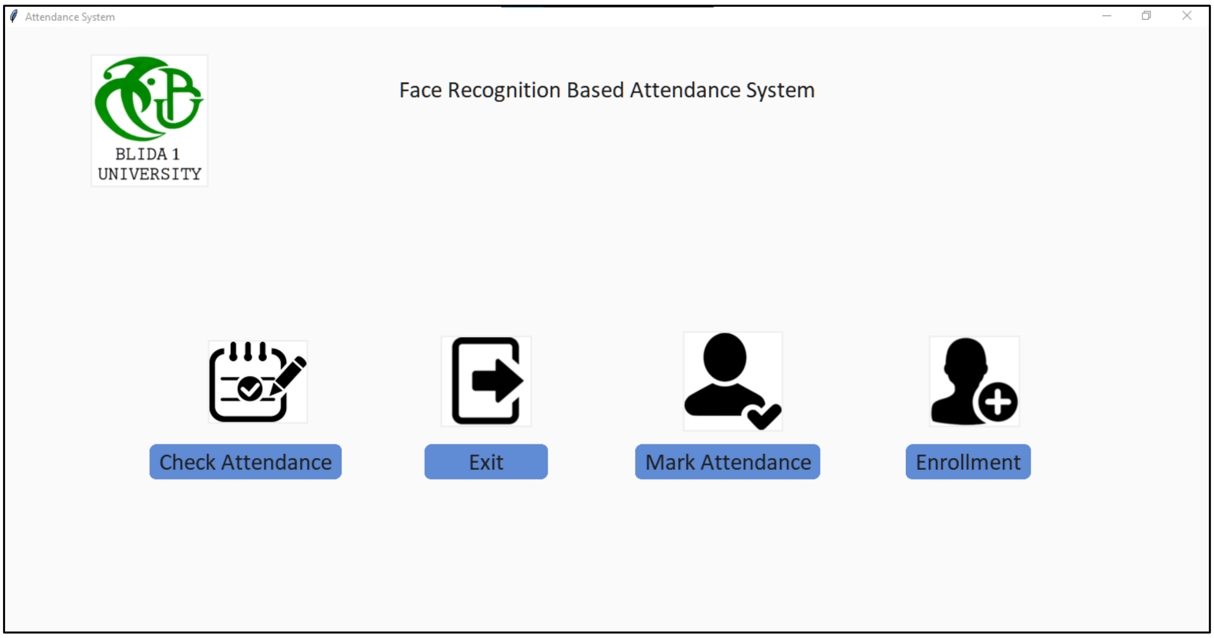


Figure III-12 the graphical user interface home page

1. Enrolment: add a student to the database
2. Mark attendance: mark live attendance
3. Exit: close the application
4. Check attendance: check the attendance

The second interface is the enrolment interface which is accessible by clicking on the Enrolment button (Figure lll-13), used to add new student into the database, map the face features and generate face embeddings.

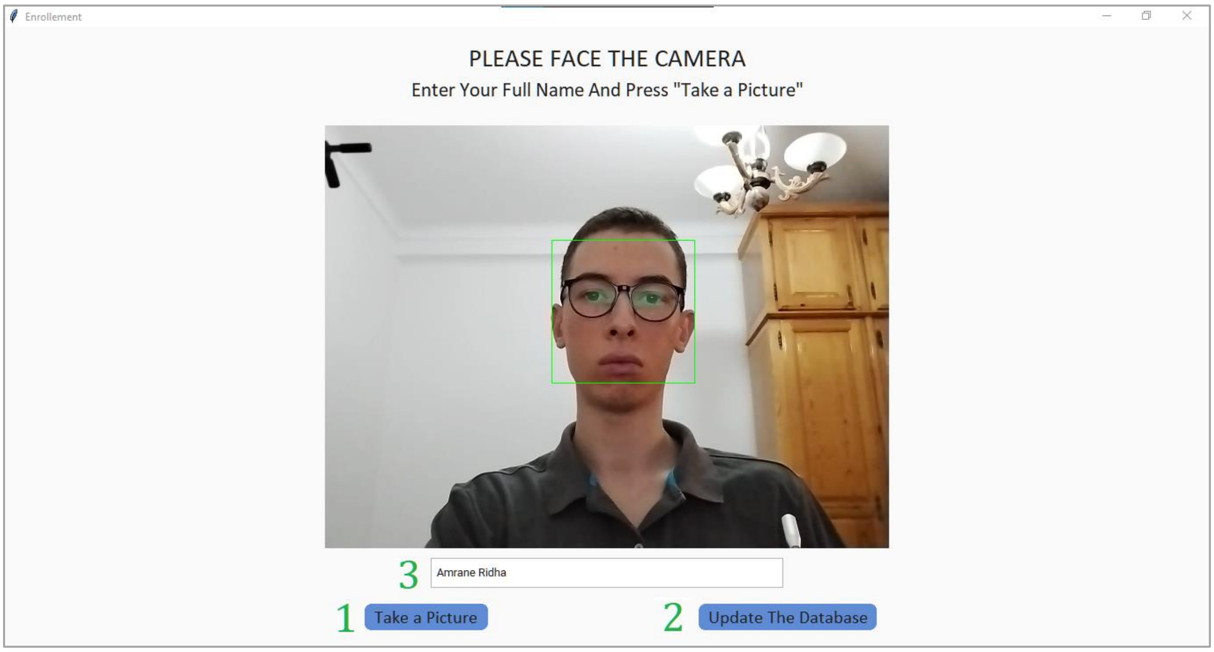


Figure III-13 enrolment interface

1. Take a Picture: Takes a picture of the whole frame and saves it in the database folder.
2. Update the Database: Updates the face embeddings into the pickle file ('data.pkl').
3. User input: the full name of the student provided by the user.

The third interface is used to mark the attendance (figure lll-14), a window will pop up and open the camera, detecting and recognizing faces in real time, also printing the identity above the detected face BBox, it also records the presence of recognized students. This interface is usable by clicking on the "Mark attendance" button on the home page (Figure lll-12).

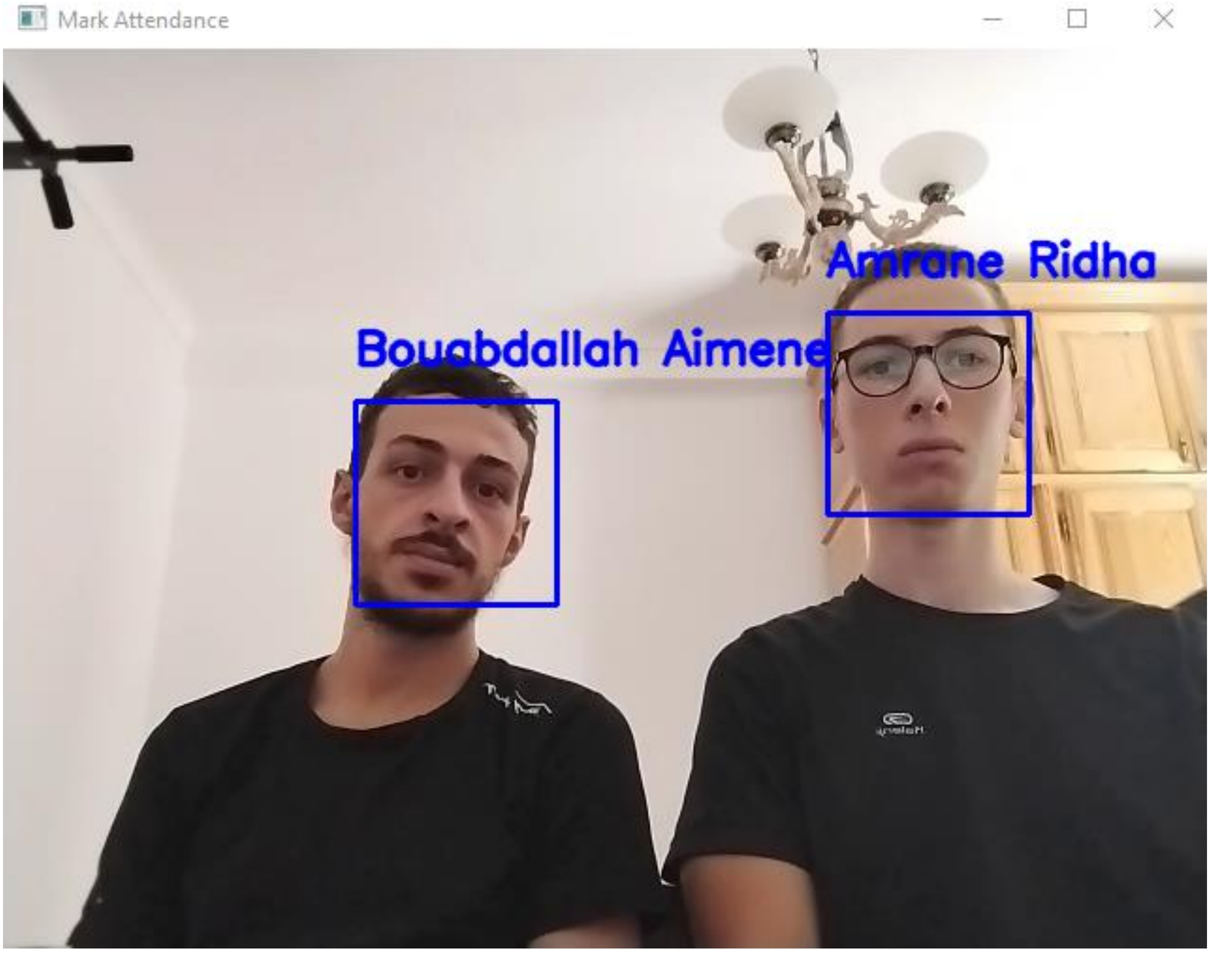
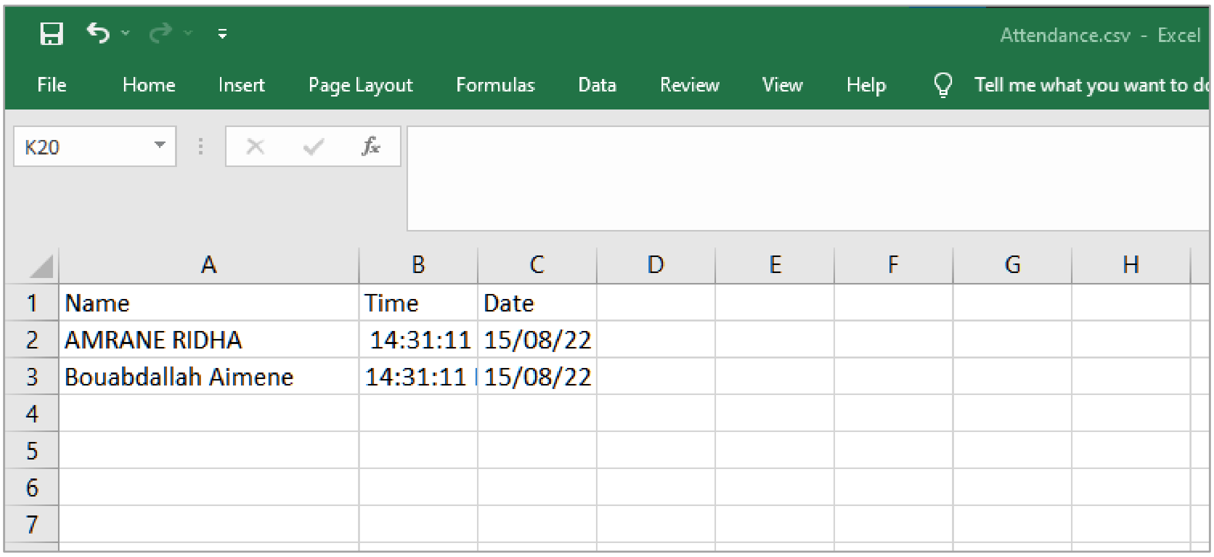


Figure III-14 Marking Attendance

We can check the attendance sheet by clicking on Check attendance button on the home page. showing the identity, the date and time (Figure lll-15)



	A	B	C	D	E	F	G	H
1	Name	Time	Date					
2	AMRANE RIDHA	14:31:11	15/08/22					
3	Bouabdallah Aimene	14:31:11	15/08/22					
4								
5								
6								
7								

Figure III-15 the students Attendance sheet

III.10 Conclusion

In this chapter we’ve introduced the creation steps of the face recognition based attendance system with its different functionalities.

The implementation of the HOG face detection algorithm improved the detection better than our custom models in terms of speed, and the accuracy of the FaceNet recognition model is pretty high.

The graphical user interface simplifies the access to the attendance lists and using the system in general, all it takes is clicking some buttons.

General conclusion

The objective of this master’s thesis is creating a real-time face detection and recognition system to do the attendance for the students, and can be used to control the access to any other environments.

We trained many deep learning models as SSD and YOLOv5 for detection and compare them to other classical algorithms like HOG, FaceNet and Siamese network to recognize the faces. Finally, after a long discussion about the results we chose the HOG algorithm for face detection and the Facenet model for face recognition.

We also created a graphical user interface for our application which contains many functionalities like marking the attendance, enrolling new members and checking the attendance lists.

This master thesis has allowed us to discover a new world of knowledge, we’ve learned many things about the artificial intelligence and the deep learning, we’ve discovered more computer vision techniques for image processing and a many more.

The time we spent in research and development we explored very powerful AI and computer vision tools.

We can’t forget that if you want to achieve any goal you have to walk across many obstacles, and we had our share of difficulties:

- For training a good deep learning model you need a powerful setup with high performance GPUs.
- And about the dataset you need a large dataset to get a high accuracy model, like FaceNet which was trained on one million images.

References

[1] Guoying Zhao, Matti Pietikainen from University of Oulu << Face Analysis Using Local Binary Patterns >> October 2008

Retrieved July 25, 2022, from https://www.researchgate.net/publication/267425681_Face_Analysis_Using_Local_Binary_Patterns

[2] Nev Acar <<Eigenfaces: Recovering Humans from Ghosts>> Aug 2018

Retrieved July 20, 2022, from

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

[3] Belhumeur PN, Hespanha JP, Kriegman DJ << Eigenfaces vs Fisherfaces: Regcognition Using Class Specific Linear Projection>> 1997

Retrieved July 6, 2022, from

https://vision.ucsd.edu/kriegman-grp/papers/pami97.Pdf

[4] Rohit Halder, Rajdeep Chatterjee << Machine Learning Vs. Deep Learning>> September 2019

Retrieved June 25, 2022, from

https://www.researchgate.net/figure/Machine-Learning-Vs-Deep-%20Learning_fig2_336348593

[5] Delong Qi, Weijun Tan, Qi Yao, Jingfeng Liu <<Shenzhen Deepcam Information Technologies>> 2020

Retrieved June 25, 2022, from

https://www.arxiv-vanity.com/papers/2105.12931/

[6] Abhishek Kumar << identifying faces with mtcnn and vggface >> Jan 2021

Retrieved July 20, 2022, from

https://medium.com/swlh/identifying-faces-with-mtcnn-and-vggface-9d0d4927cccf

[7] Luka Dulčić << face recognition with-facenet and mtcnn >> July 2020

Retrieved July 20, 2022, from

https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/

[8] << Colaboratory>>

Retrieved July 27, 2022, from

https://research.google.com/colaboratory/faq.html

[9] <<Jupyter>>

Retrieved July 27, 2022, from

https://jupyter.org

[10] Serdar Yegulalp << What is TensorFlow? The machine learning library explained>> June 2022

Retrieved June 21, 2022, from

https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

[11] << Top 5 Most Popular Computer Vision Tools >> July 2021

Retrieved June 21, 2022, from

https://www.optisolbusiness.com/insight/top-5-most-popular-computer-vision-tools-in-2021

[12] <<Guide: Labelme>> October 2020

Retrieved June 21, 2022, from

https://datagen.tech/guides/image-annotation/labelme

[13] eshan1285 <<NumPy Array Operations>> 2020

Retrieved June 21, 2022, from

https://jovian.ai/eshan1285/basics-numpy

[14] <<Albumentations is a Python library for image augmentation>> Feb 2022

Retrieved June 21, 2022, from

https://curatedpython.com/p/albumentations-is-albumentations-team-albumentations/index.html

[15] Rohini G <<Everything you need to know about VGG16>> Sep 2021

Retrieved June 21, 2022, from

https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[16] Mohamed Traore, Justin Brady <<Face Detection Image Dataset>> July 2022

Retrieved July 23, 2022, from

https://universe.roboflow.com/mohamed-traore-2ekkp/face-detection-mik1i/dataset/10

[17] << Module: tf.keras.metrics >> July 2022

Retrieved July 28, 2022, from

https://www.tensorflow.org/api_docs/python/tf/keras/metrics

[18] Mark <<Modern Tk Best Practices>> Nov 2021

Retrieved august 25, 2022, from

http://tkdocs.com/

[19] Alex Clark <<Pillow>> March 2022

Retrieved august 25, 2022, from

https://pillow.readthedocs.io/en/stable

[20] << pickle — Python object serialization >> Sep 2021

Retrieved august 25, 2022, from

https://docs.python.org/3/library/pickle.html

[21] Surya Gutta << Object Detection Algorithm — YOLO v5 Architecture >> Aug 2021

Retrieved august 27, 2022, from

https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef

[22] Glenn Jocher << ultralytics yolov5>> June 2020

Retrieved august 27, 2022, from

https://github.com/ultralytics/yolov5?fbclid=IwAR0mnFzfq1SHFBIdabgFQs39dUscU3Fy4STvTaQ7m_UoRcBuSg9Itq1m5R8

[23] Shivy Yohanandan << mAP (mean Average Precision) might confuse you!>> June 2020

Retrieved august 27, 2022, from

https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2

24 Grace Karimi << Introduction to YOLO Algorithm for Object Detection >> April 2021

Retrieved august 30, 2022, from

https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection