Politecnico di Torino

Master of Science in
Computer Engineering

FACE RECOGNITION SYSTEM FOR SERVICE ROBOTICS APPLICATIONS WITH DEEP LEARNING AT THE EDGE

Supervisors:
prof. Marcello Chiaberge
Co-supervisors:
Vittorio Mazzia
Francesco Salvetti

Student:
Musaab Ahmed Elsheikh Awouda

December 2021

Acknowledgments

First of all, I would like to thank my beloved family, for always being caring, understanding, and supporting me throughout my life.

I am deeply thankful and grateful to my supervisor Prof. Marcello Chiaberge, Eng. Vittorio Mazzia, Eng. Francesco Salvetti, and all members of the PIC4SeR research group for their help and guidance throughout this work.

I am also thankful and appreciative for Politecnico di Torino for giving me the chance to push my career to the next level, and for EDISU Piemonte, the Regional Agency for the Right to University Education of Piemonte, for financially supporting me during my study.

Finally, a thank you is dedicated to all old and new friends for the laughs, the words of comfort, and all the shared moments.

THANK YOU ALL!

GRAZIE A TUTTI!

Abstract

In this thesis, we introduce the problem of facial recognition, which is an active research field in computer vision; many real-life applications like identification, access control, and human-machine interactions require an accurate, fast, and stable face recognition algorithm. With the rise of Deep convolutional neural networks (CNNs), and the massive advancement in relevant technologies (e.g.,smartphones, digital cameras, GPU,...), this task has gained a significant performance improvement.

The first stage in the face recognition pipeline is to identify a model capable of detecting and locating a face, if present, in an image or video frame, and then locate the face landmarks (left eye, right eye, nose, upper lip, and lower lip). One of the best tools for face detection is Dlib, an open-source library that provides the best environment for developing software based on machine learning in C++. After face detection, we crop the face, normalize the image, and then feed it to the second stage.

In the second stage, we extract features from the detected face in the form of a vector called the embedded vector, which contains enough information to distinguish between different identities. After studying the state-of-the-art models and algorithms, we decided to adopt ArcFace (Additive Angular Margin Loss for Deep Face Recognition). The key feature in this model is the design of appropriate loss functions that enhance discriminative power for the extracted features. This model is trained using CASIA, VGGFace2, and MS1MV2 datasets and achieved a verification performance of 99.83% when tested on the LFW dataset.

In the last stage, we use the embedded vectors extracted from all the images in our local dataset to carry out the classification and identification. When a new image is to be identified, we extract its embedded vector and calculate its distance from the other vectors in the dataset. Based on the distance value, the system can identify it. Moreover, to distinguish between a real face and an image of a face in front of the camera, we utilize the depth map information extracted from the Intel RealSense depth camera.

Finally, a user-friendly GUI (Graphical User Interface) is developed to allow users to add and remove IDs from the Dataset, and to launch the application to do real-time face recognition.

This thesis is developed in collaboration with PoliTO Interdepartmental Centre for Service Robotics (PIC4SeR). It is a lightweight application that can run locally and without an internet connection. The application is developed and deployed on the Nvidia Jetson AGX Xavier developer kit, and can be beneficial for many service robotics applications.

Contents

1 Introduction **7**

1.1 Problem identification and motivation 7

1.2 Face Recognition History 8

1.3 Automated Face Recognition Systems Pipeline 8

2 Face recognition technology: Fundamental rights and ethical issues **10**

3 Machine Learning and Deep Learning **14**

3.1 What is Machine Learning? 14

3.2 Support Vector Machine 15

3.3 K-Nearest Neighbors 17

3.4 Neural networks 17

3.4.1 Neural networks training 19

3.5 Convolutional Neural Networks 20

3.5.1 CNN architecture building blocks 21

3.6 AlexNet 23

3.7 VGG 23

3.8 GoogleNet 24

3.9 ResNet 25

4 Face Detection **28**

4.1 Overview 28

4.2 State-Of-The-Art Algorithms 29

4.2.1 Viola-Jones Face Detector 29

4.2.2 MTCNN 30

4.2.3 Dlib CNN 31

5 Feature Extraction **32**

5.1 Overview 32

5.2 Available Datasets for Face Recognition 33

5.2.1 FRGC Dataset 33

5.2.2 LFW Dataset 34

5.2.3 CASIA-WebFace Dataset 34

5.2.4 MegaFace Dataset 34

5.2.5 Ms-Celeb-M1 Dataset 34

5.2.6 VGGFACE2 Dataset . . . 34
5.2.7 LFR Dataset . . . 35
5.3 State-Of-The-Art Algorithms . . . 35
5.3.1 FaceNet . . . 35
5.3.2 Center Loss . . . 36
5.3.3 SphereFace . . . 38
5.3.4 CosFace . . . 39
5.3.5 ArcFace . . . 41

6 Methodology **44**
6.1 Hardware Specifications . . . 44
6.1.1 ROScube-X . . . 44
6.1.2 Intel Realsense Depth Camera D435i . . . 45
6.2 Sofware . . . 46
6.3 System Development . . . 46
6.3.1 Camera Input . . . 46
6.3.2 Dlib face detection . . . 48
6.3.3 Feature Extraction using ArcFace . . . 48
6.3.4 Selecting Threshold Value . . . 50
6.3.5 Head Pose Estimation . . . 52
6.3.6 Graphical User Interface . . . 54
6.3.7 Depth Information . . . 56

7 Results and Discussion **57**
7.1 Evaluation Results . . . 57
7.1.1 Results on local dataset . . . 57
7.2 Discussion . . . 60

Bibliography **62**

List of Figures

1.1 Automated Face Recognition System Pipline 9

2.1 A survey results about how comfortable people are you with using their face images when crossing the border 12

3.1 Machine Learning Vs Conventional Programming 14
3.2 Support Vector Machine Algorithm 16
3.3 Support Vector Machine kernel Trick 16
3.4 K-Nearest Neighbors 17
3.5 Neural network Layers 18
3.6 Neuron Unit in Neural networks 18
3.7 Some of the most used activation functions 19
3.8 CNN Convolutional Layers 21
3.9 CNN Pooling Layers 22
3.10 CNN Fully Connected Layer 22
3.11 AlexNet Architecture 23
3.12 VGG Architecture 24
3.13 GoogleNet Architecture Building Block 25
3.14 inception v3 Architecture 25
3.15 Training and Testing errors 26
3.16 ResNet Residual Block 26
3.17 ResNet Architecture 27

4.1 Face Detection 28
4.2 Haar-like Features 29
4.3 Cascade Classifier 30
4.4 MTCNN Stages Architecture 31

5.1 Inter-Class and Intra-Class Distances 32
5.2 FRGC Dataset samples: controlled stills, uncontrolled stills, and 3D shape. 33
5.3 Example Images from the VGGFACE2 dataset. 35
5.4 FaceNet Triplet Loss Function 36
5.5 Softmax Loss Function and the distance between classes 37
5.6 Center Loss 37
5.7 SpereFace 3D features projection and angle distribution 39
5.8 Softmax Loss Vs LMCL with different m value 40
5.9 ArcFace Loss 42

5.10 ArcFace Vs Softmax 42
5.11 Comparesion between Softmax, SphereFace, CosFace, and ArcFace 43

6.1 ROScube-x Controller 45
6.2 Intel Realsense Depth Camera D435i 45
6.3 Intel realsense-viewer showing both RGB and Depth frames 47
6.4 Dlib Facial Landmarks 48
6.5 Max accuracy and the corresponding threshold 51
6.6 Coordinates system in head pose estimation 53
6.7 Automated Face Recognition GUI 54
6.8 GUI, Add personnel 55

7.1 model Accuracy 58
7.2 model Accuracy 59
7.3 F1 score 60

Chapter 1

Introduction

1.1 Problem identification and motivation

Face recognition is a task that humans perform routinely and effortlessly on a daily basis. The wide availability of robust and low-cost desktop and embedded computing systems has created a massive interest and tremendous attention in the automatic processing of digital images in various applications, including biometric authentication, surveillance, human-machine interface, and multimedia management. Research and development in automatic face recognition follow naturally.

Face recognition is a biometric procedure that uses automated systems to verify or identify a living person based on his/her physiological features. In general, a biometric identification system makes use of either physiological features (such as a fingerprint, iris pattern, or face) or behavior patterns (such as hand-writing, voice, or key-stroke pattern) to identify a person. Compered to other biometric approaches, Face recognition has several advantages over them, it is more natural and non-intrusive, and the most essential advantage that is capturing a face can be done from a distance and in a covert manner.

A face recognition system, as a biometric system, can function in one of two modes:

1. Face verification or authentication: it is a one-to-one model that is validating a claimed identity through an image of a face, and either accepting or rejecting the identity claim.

2. Face identification: it is a one-to-many model that compares a query face image to multiple faces stored in a local dataset and outputs the identity of the query image if it belongs to one in the dataset.

Considering constrained environments, hand-crafted features such as Local Binary Patterns (LBP) [1] and Local Phase Quantisation (LPQ) [2, 4] have achieved acceptable face recognition performance. However, the performance dropped dramatically in unconstrained environments where the face image can be affected by several factors such as illumination, facial pose, expression, age span, hair, facial wear.

It remains an open problem to find a robust face recognition system in unconstrained environments, however, in the last years, convolutional neural networks (CNN) have achieved outstanding performance results.

According to a report by the analytical company Mordor-Intelligence [8], the face recognition market was estimated at 4.4 billion dollars worldwide in 2019, and it is expected to exceed 10.9 billion in 2025.

1.2 Face Recognition History

This section reviews the history of face recognition algorithms and reports the most critical stages that have contributed to the state-of-the-art performance of today's algorithms.

- 1964: A semi-automatic system was developed by the American researchers Bledsoe et al.[1] where they input to the system 20 values such as the size of the mouth or the eyes.
- 1977: The system was enhanced by adding 21 further inputs (e.g., the width of the lips, the color of the hair).
- 1988: Artificial intelligence and linear algebra was used to translate images and manipulate them in simple way and independently of the human features.
- 1991: Eigenfaces [2] was introduced by Alex Pentland and Matthew Turk of MIT, using the statistical Principal component analysis method. It is considered the first acceptable face recognition algorithm.
- 1998: the Defense Advanced Research Projects Agency introduced face recognition technology (FERET) [3] trained on a dataset of 2400 images of 850 identities.
- 2005: The Face Recognition Grand Challenge competition (FRGC) [4] was launched to encourage developing face recognition algorithms.
- 2011: With the coming of deep learning, face recognition got a massive boost in terms of results and performance. Many deep learning algorithms achieved accuracy close to the human eye.

Today face recognition algorithms play a crucial part in multiple investments in different domains such as commercial, industrial, legal, and governmental applications.

1.3 Automated Face Recognition Systems Pipeline

The problem of automated face recognition system can be divided into three main stages, as shown in figure(1.1):

1. Face detection: To determine whether or not an image includes a face or multiple faces and return the location in the image of each detected face.

2. Feature extraction: To extract from the detected face a feature vector that contains enough information to represent the face and distinguish between different faces of different identities.

3. Classification: To identify or verify the faces using appropriate classification algorithms.

Figure 1.1: Automated Face Recognition System Pipline

Chapter 2

Face recognition technology: Fundamental rights and ethical issues

Face recognition technology automatically allows the recognition and identification of individuals by processing their facial images. It is extensively used in private sectors in different domains such as advertisement and marketing, here are some examples of the usage of face recognition technology in the private sector:

- In marketing, customers preferences can be predicted by studying their facial expressions [25].
- Social media companies like Faceboob, are using face recognition technology in their tagging images systems [28].
- Face recognition technology is used by some companies to analyze the facial expression of candidates during job interviews [27].
- Some football clubs use face recognition technology in their stadium during matches to track people who are banned from attending due to violation of the club's rules [26].

Although there are many implementations for face recognition technology in the private sector, only recently, with the evolution of deep neural networks and their tremendous impact on face recognition performance accuracy, it has managed to attract public administration attention. However, this has led to a serious debate on its possible impact on fundamental rights. one example is the intense use of facial recognition technology with surveillance cameras in China which led to many concerns about potential human rights violations.

EU General Data Protection Regulation (GDPR) calls for increased attention to compliance among both governmental and private organizations when using this technology. According to GDPR, the data collected for any face recognition system is classified as **biometric data**, which is a special category of personal data because it makes it possible to uniquely identify a person and it is particularly connected to the protection of individual privacy.

Generally, the processing of biometric data are prohibited, and can be processed only under the following conditions:

- An explicit legal consent must be granted by the data subjects before the process of personal data.
- The data processing has to be considered as necessary for reasons of significant public interest.

Case Study:

In 2019, a Swedish school decided to run a pilot program that uses face recognition to automatically carry out the attendance of 22 students to save the numerous hours that teachers spent on attendance reporting. Local media reports drew the attention of Swedish authorities and pressure them to take action, at the end the Swedish Data Protection Authority (DPA) decided to impose a $20,700 fine on the school although consent from the student's parents was obtained.

According to Swedish Data Protection Authority the main issues are:

- **satisfaction of the principles of data protection**. Swedish Data Protection Authority claimed that principles governing the processing of personal data were violated and personal data was processed more extensively than was needed, using automated face recognition system was deemed disproportionate for tracking attendance and less intrusive alternative approaches were available.
- **The nature of consent**. As mentioned before, the collected data for the case of the Swedish school is classified as a special category of personal data and it is prohibited to process such data without consent, which was given in this case, however, the Data Protection Authority claimed that due to the imbalance of power dynamics between the school (the controller) and its students (study subjects) and the one sided nature of the collecting of attendance data, the consent could not be acknowledged as freely given in the way meant by GDPR, and thus this considered as a non-valid exception to the prohibition of processing personal biometric data.
- **Data protection impact assessment**. The school claims that they have done a risk analysis, and they consider it enough and it is not necessary to carry out a specific analysis on personal data, however, local DPA stated that an actual data protection analysis had to be done, and the risk assessment carried out by the school did not consider the risks to the data subject's rights and freedoms.

- **Prior consultation**. According to Article 36 of the GDPA a prior consultation had to be requested, and a prior consultation can only be requested after the impact assessment has been carried out.

All these points and others must be taken into consideration when developing, testing, and deploying an automated face recognition system regardless of its scale to stay GDPR compliant.

However, some human rights activists argue that when it comes to face recognition systems, the problem is not only limited to being a GDPR compliant or not, the use of this technology is also related to the right to human dignity. Human dignity is the foundation of all fundamental rights guaranteed by the EU Charter of Fundamental Rights [29]. Article 1 of the Charter states that human dignity is inviolable and that it must be respected and protected. The Court of Justice of the EU (CJEU) has confirmed in its case law that the fundamental right to dignity is part of EU law [30].

In 2015 a survey was conducted including 1,227 third-country nationals at seven border crossing points in Europe where a face recognition technology is used, see figure (2.1), 12% of all participants replied that they felt very uncomfortable when their face image was used for crossing the border, 18% considered providing face images at a border very intrusive to their privacy, and 26% said that doing so was humiliating. This study presents how face recognition is perceived and the fundamental rights implications of such technology.

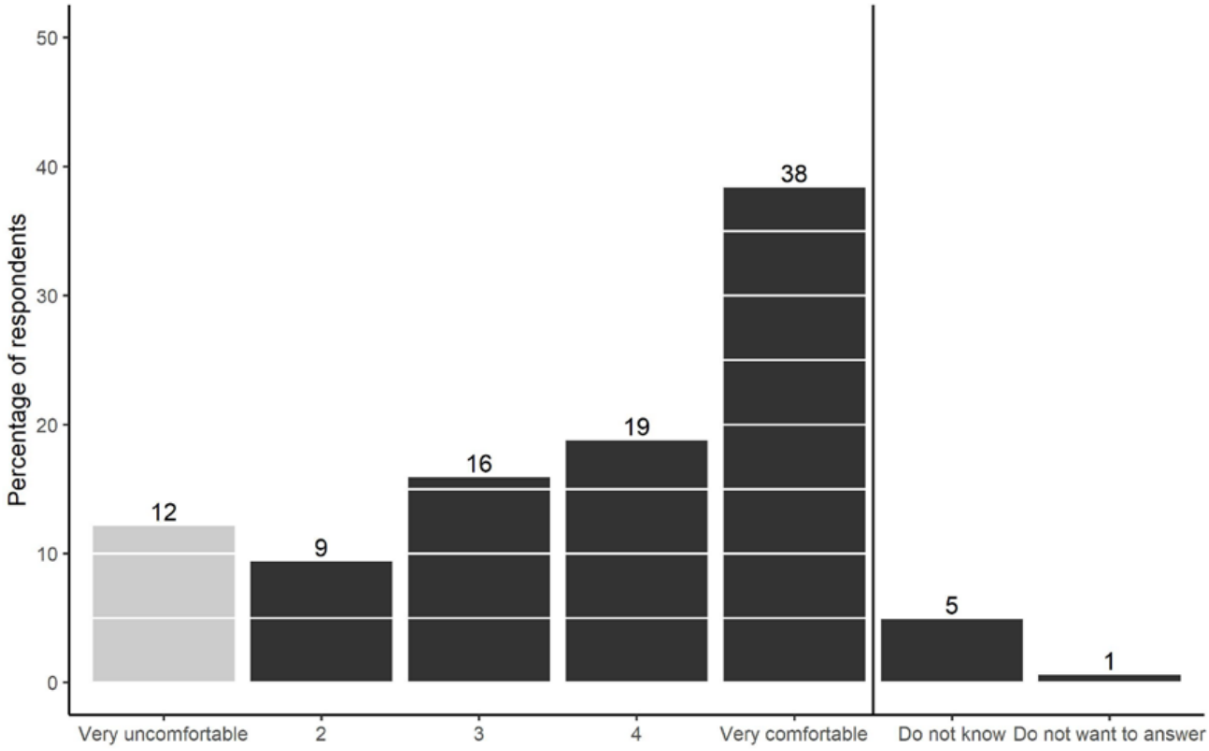


Figure 2.1: A survey results about how comfortable people are you with using their face images when crossing the border

To summarize, face recognition technology has introduced a range of exciting possibilities to enhance human lives, such as preventing crimes, increasing safety and security, reducing avoidable human interaction and labor, and even in some cases, it can help in supporting medical efforts. Nevertheless, the development of any face recognition system has to be compliant with all the rules and regulations regarding collecting such private data and must be done with grace and respect to people feeling to preserve their dignity, for human dignity is the foundation of all fundamental rights.

Chapter 3

Machine Learning and Deep Learning

3.1 What is Machine Learning?

According to Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, Machine Learning is: "the field of study that gives computers the ability to learn without being explicitly programmed." This can be considered as an old, informal definition.

Tom Mitchell, on the other hand, provided a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Machine Learning is a subset of Artificial Intelligence. Machine Learning is the study of making machines more human-like in their behavior and decisions by giving them the ability to learn and develop their own programs. This is done with minimum human intervention, i.e., no explicit programming. And that is the different between machine learning and conventional programming, as can be seen in figure 3.1, in conventional programming, we would feed the input data and a well written and tested program into a machine to generate results. while with machine learning, input data along with the results is fed into the machine during the learning phase, and it works out a program for itself.

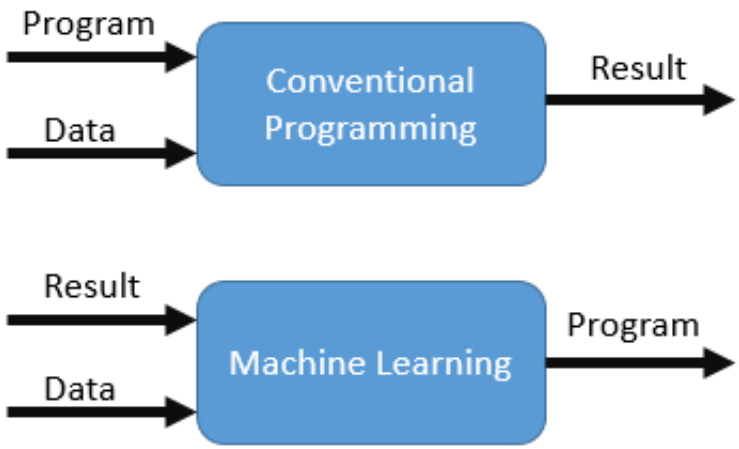


Figure 3.1: Machine Learning Vs Conventional Programming

The performance of the machine learning algorithms depends heavily on the amount and quality of the input data; a simple ML algorithm can achieve outstanding results if the model is fed with enough data. However, gathering enough data is not always that easy, this process can be expansive, costly, and even sometimes have legal issues related.

Machine learning algorithms, depending to the task to be performed, can be broadly devided into two main catogeries:

- Supervised learning: In supervised learning, we have prior knowledge of what the output values for our samples should be. Therefore, supervised learning aims to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data. The training process continues until the model achieves the desired level of accuracy on the training data. Examples of supervised learning: SVM, KNN, Neural Networks, Logistic Regression etc.

- Unsupervised learning: Is to learn a useful structure without labeled data. In this algorithm, we do not have any target variable to estimate means here we do not have any label associated with data points. This type of algorithms is used for organizing the data into groups of clusters to describe its structure, i.e., cluster the data to reveal meaningful partitions and hierarchies. It makes data look simple and organized for analysis. Examples of unsupervised learning: K-means, Fuzzy clustering, Hierarchical clustering.

In this thesis, and considering our underlying problem of face recognition, we will focus on the supervised algorithms, and in the following sections, we will briefly describe some of those algorithms.

3.2 Support Vector Machine

SVM or Support Vector Machine is a linear model used for classification and regression problems, but mostly for classification. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

According to the SVM algorithm, we find the closest point (in each class) to the line that separates the classes. These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin. The goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.

This algorithm is very powerful and effective because it only uses the support vector points and not all points, which reduces the amount of computation needed.

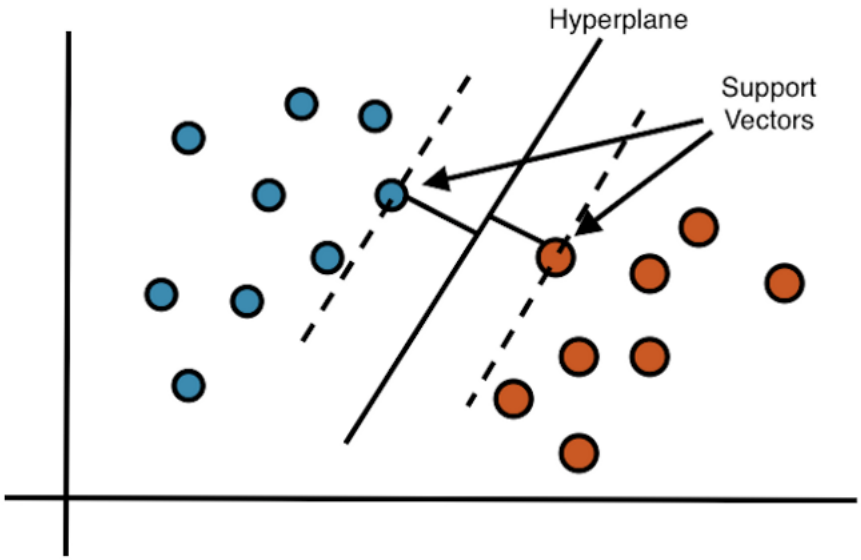


Figure 3.2: Support Vector Machine Algorithm

However, if the complete separation of the classes is not possible, an approach called soft margin classification can be used. using soft margin allows some outliers to be wrongly classified (stay on the wrong side of the separation line), and the algorithm tries to minimize this number of outliers.

Furthermore, if We cannot draw a straight line that can classify this data (not linearly separable), we can use another trick called the kernel trick.

The idea behind the kernel trick is that if the data is not linearly separable, we can project this data into a higher dimension where it is linearly separable and obtain a hyperplane that can classify the data.

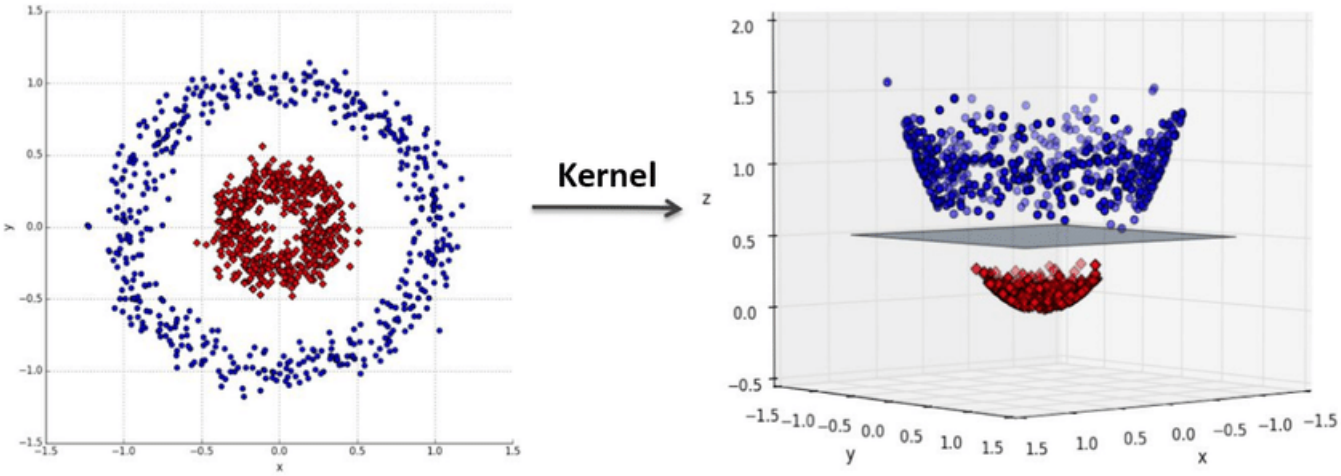


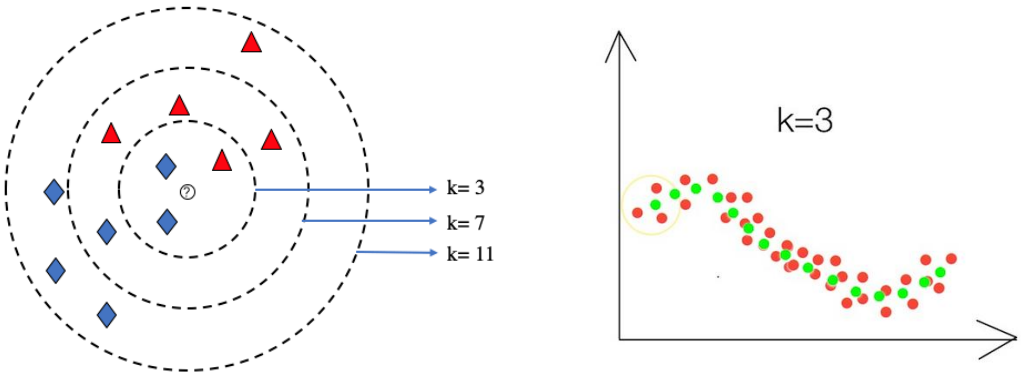
Figure 3.3: Support Vector Machine kernel Trick

3.3 K-Nearest Neighbors

KNN or K-Nearest Neighbors is a non-parametric, lazy learning algorithm. Non-parametric means the algorithm does not make any assumptions on the data being studied, i.e., the model is distributed from the data. And lazy learning term comes from the fact that it does not learn from the training set immediately; instead, it stores the dataset, and at the time of classification, it performs an action on the dataset.

KNN is a supervised learning algorithm that can be used for both classification and regression.

In k-NN classification, the output is a class membership. An object is classified by a majority class of its neighbors, meaning the object being assigned to the class most common among its k nearest neighbors (k is a positive integer number, input to the algorithm).



(a) Classification (b) Regression

Figure 3.4: K-Nearest Neighbors

As we can see in figure(2.4), the new object will be classified differently based on the value of K. If K is three or eleven, the new object will be classified as Blue, while if k is seven, it will be classified as Red.

In k-NN regression, the output value is the average of the values of the k nearest neighbors.

3.4 Neural networks

The neural networks is an algorithm modeled relatively after the human brain developed to recognize patterns. They interpret distinct data through a kind of machine perception, labeling, or clustering raw input. The recognized patterns are numerical, included in vectors. The input data can be of different kinds, images, sound, text, or time series.

The neural networks consist of multiple layers, the input layer, which contain the input data, the output layer, and one or more hidden layer.

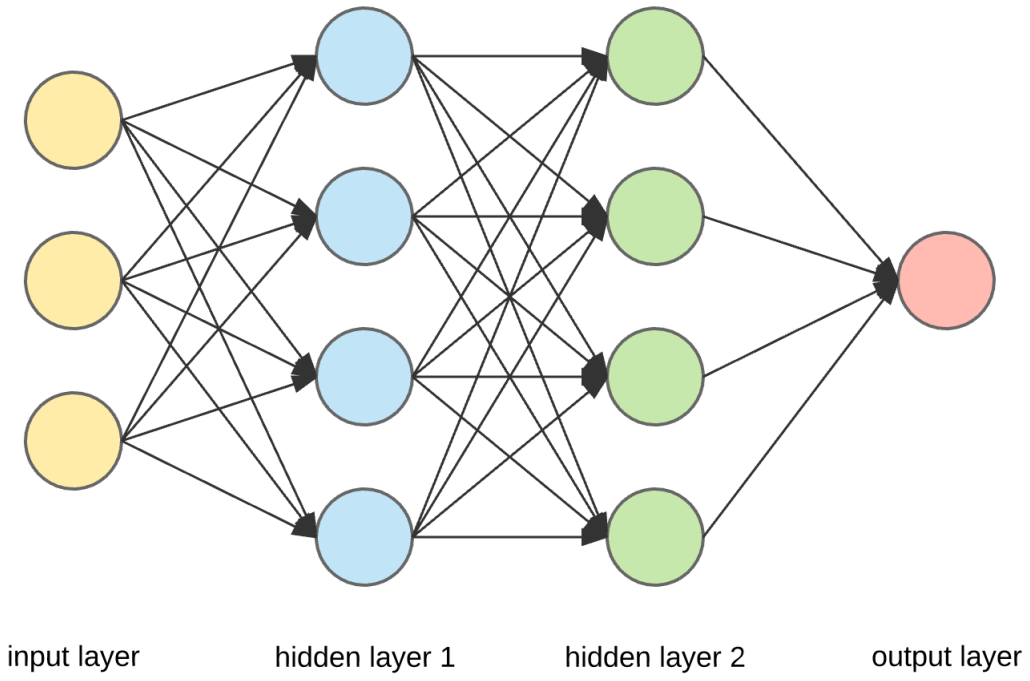


Figure 3.5: Neural network Layers

Each layer consist of multiple nodes. A node is just a place where computation is carried out, roughly trying to mimic the brain's neurons, which are activated when it receives enough stimuli. A node multiply inputs with a set of coefficients or weights that magnify or dampen that input value, thereby assigning a weight to the inputs concerning the task the algorithm is trying to learn and deciding which input is the most significant to do the task without errors.

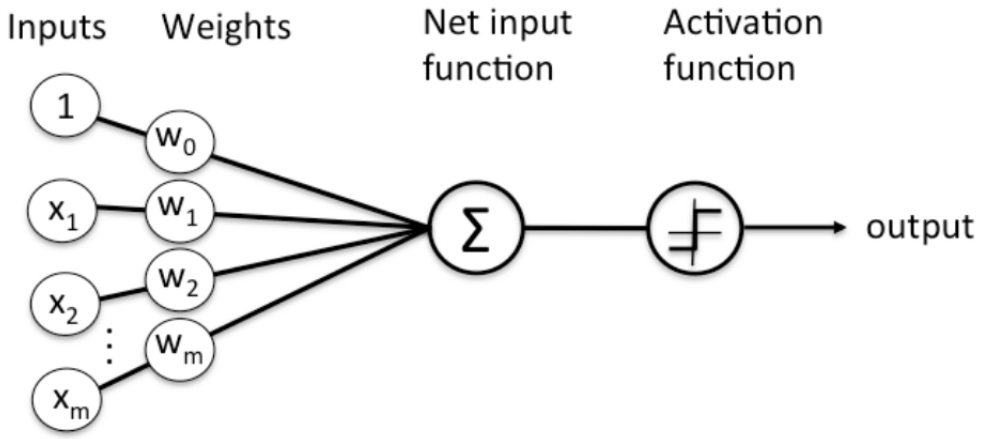


Figure 3.6: Neuron Unit in Neural networks

The inputs-weights product values are summed, and then the final value is passed through a node called the activation function to determine whether and to what extent that signal should progress further through the network to affect the eventual output. The output of the activation function is rescaled between 0 and 1, or between -1 and 1. Computational efficiency is fundamentally essential, as activation functions must be run for each neuron; for this reason, many different functions have been proposed to obtain the best trade-off between performance and simplicity. Some examples can be found in figure(2.7).

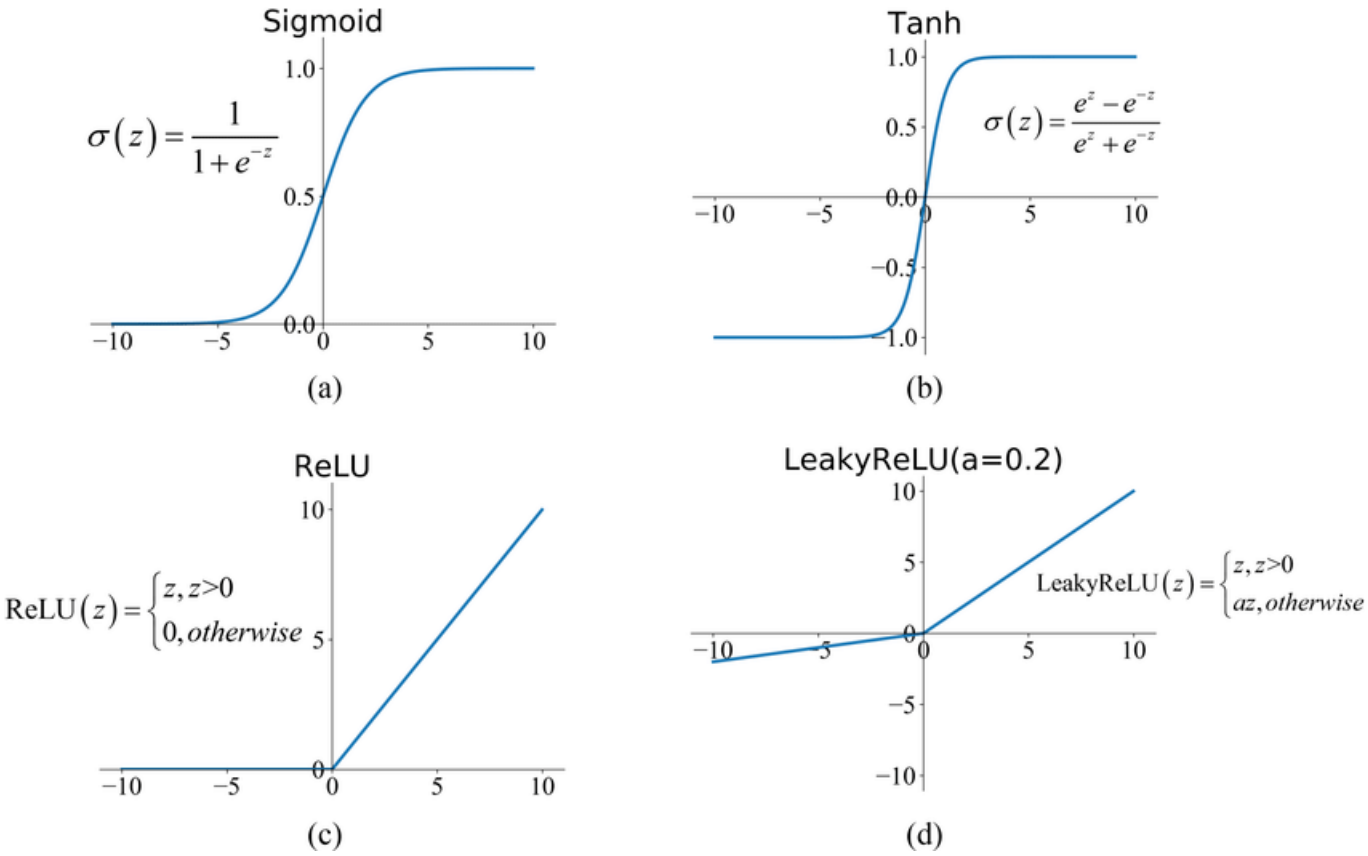


Figure 3.7: Some of the most used activation functions

3.4.1 Neural networks training

The neutral networks training is a relatively long and complicated process; however, in this section, we are going to have a general overview of this process and how the neural networks actually learn from the input data.

The training process consists of two main parts, the forward pass and the backward pass.

The forward pass: is to move forward through the network, starting from the input layer, calculating the output of each layer as we mentioned before by multiplying the

input value by the value of the weights and then through the activation function.

$$a^{(l)} = \sigma(\omega * a^{(l-1)} + b) \tag{3.1}$$

Moving forward through the layers until we reach the output layer and get an output. Then using the cost function, we calculate how accurate our output is.

There are many representations of the cost function; the most common one is the mean square error (MSE), which is the mean square of the difference between the predicted output and the actual output (the ground truth).

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{3.2}$$

The backward pass: after getting the final output of the network (which surely will not be accurate after the first forward pass) we go back and adjust the weights and biases to optimize the cost function.

To update weights and biases, we calculate so-called gradients, which is small nudges (updates) to individual weights and biases in each layer.

$$\frac{\partial C}{\partial \omega^{(l)}} = \frac{\partial C}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial \omega^{(l)}} \tag{3.3}$$

We get the updated weight value by subtracting the value of the learning rate (a hyperparameter that can be tunned) times the cost of a particular weight from the original value that particular weight initially had.

$$\omega^{(l)} = \omega^{(l)} - LearningRate * \frac{\partial C}{\partial \omega^{(l)}} \tag{3.4}$$

We keep doing the forward pass and the backward pass and adjusting the weights until we reach optimum values for the weights and the biases.

3.5 Convolutional Neural Networks

A convolutional neural networks (CNN) is a neural network that has one or more convolutional layers in its structure. CNNs are mainly used in image processing, classification, and segmentation, but we will focus on image processing application type here in this thesis.

In image processing, the input is usually is RGB image (three color channels matrix Red, Green, and Blue). However, here, for simplicity, we consider grayscale images (two-dimensional matrix, one color channel).

In 2012, AlexNet network 13 achieved fantastic results and won the ILSVRC-2012

image classification competition; since then, many researches have focused on improving the architecture of CNN and increasing network depth and width. VGG, GoogLeNet, and ResNets are some famous CNN architecture with fantastic results and performance.

3.5.1 CNN architecture building blocks

The main building blocks of CNN are the convolution layer, pooling layers, and fully connected layers.

Convolutional layers:

In the convolutional layer, we have a matrix of integers known as filter or convolution kernel. The filter is placed over a subset of the input pixels matrix; each pixel value is multiplied by the corresponding value in the filter matrix, then the results are summed up in a single value that represents the grid cell value in the output feature map.

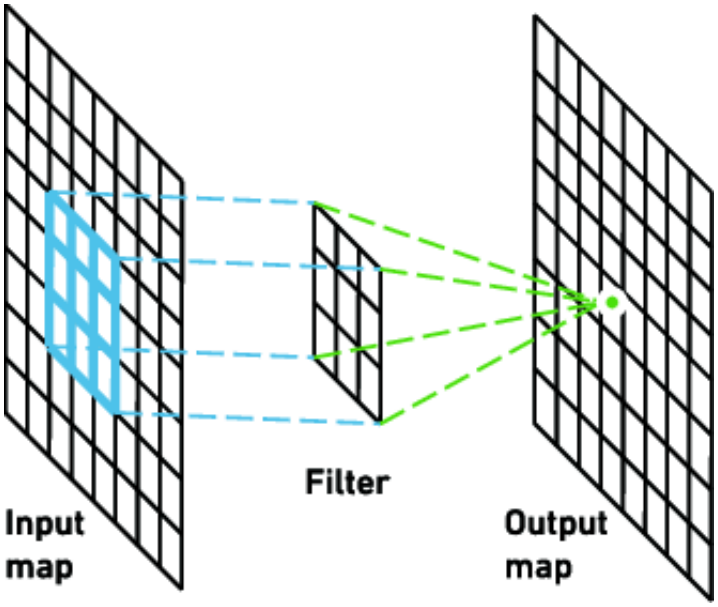


Figure 3.8: CNN Convolutional Layers

Pooling layers:

The pooling layer is usually placed between the convolution layers. Its task is to gradually reduce the size of the input data and, therefore, reduce the number of parameters in the network. The most used pooling layers are average pooling and max pooling.

We select the size of the filter, and then in the case of average pooling, we calculate the average value of the input, while in max-pooling, we select the max value.

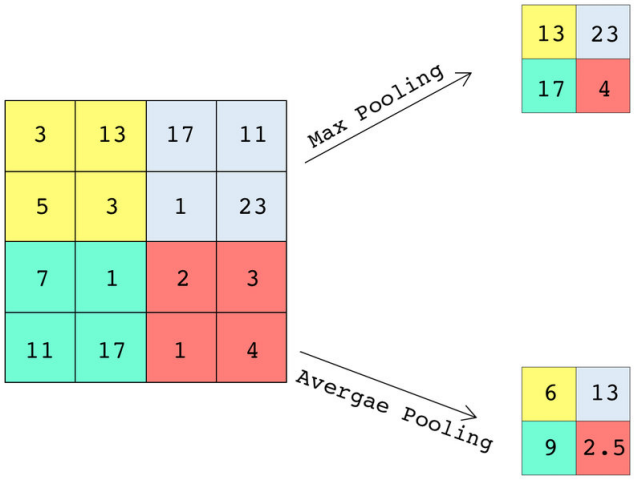


Figure 3.9: CNN Pooling Layers

Fully Connected Layer:

The feature map of the input image is extracted using a series of convolution and pooling layers; after that, the feature map can be are transformed into a fully connected layer.

In fully connected layers (FC), we connect each neuron in one layer to each neuron in the next layer. FC creates 2 to the power of n connections, where n is the number of neurons of the layer.

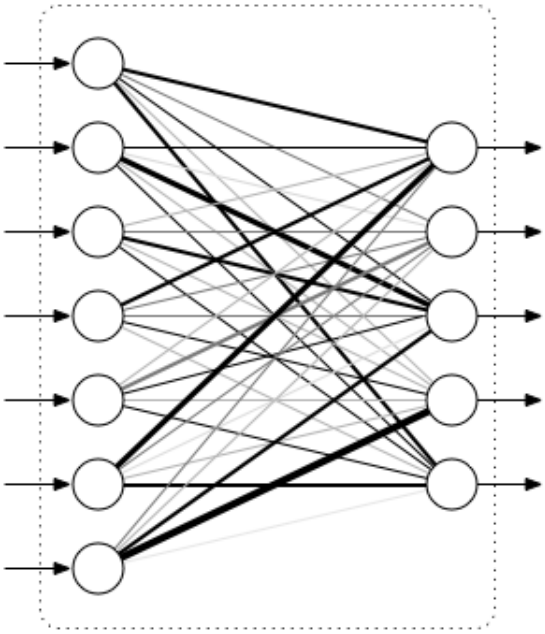


Figure 3.10: CNN Fully Connected Layer

3.6 AlexNet

Alex Krizhevsky introduced AlexNet in 2012 in the ImageNet Large Scale Visual Recognition Challenge (ILSVRV). AlexNet architecture consists of five convolution layers; the first and second convolution layers are connected to max-pooling layers to extract a maximum number of features. The remaining convolution layers are connected directly to three fully connected layers (FC). The output of both convolution layers and fully connected layers are connected ReLu activation function. The final layer is then connected to a softmax layer that produces a distribution of 1000 classes.

The input image size is $(256 \times 256 \times 3)$, that is an image in RGB of (256×256) pixel size. The architecture contains 60 million parameters and over 650,000 neurons.

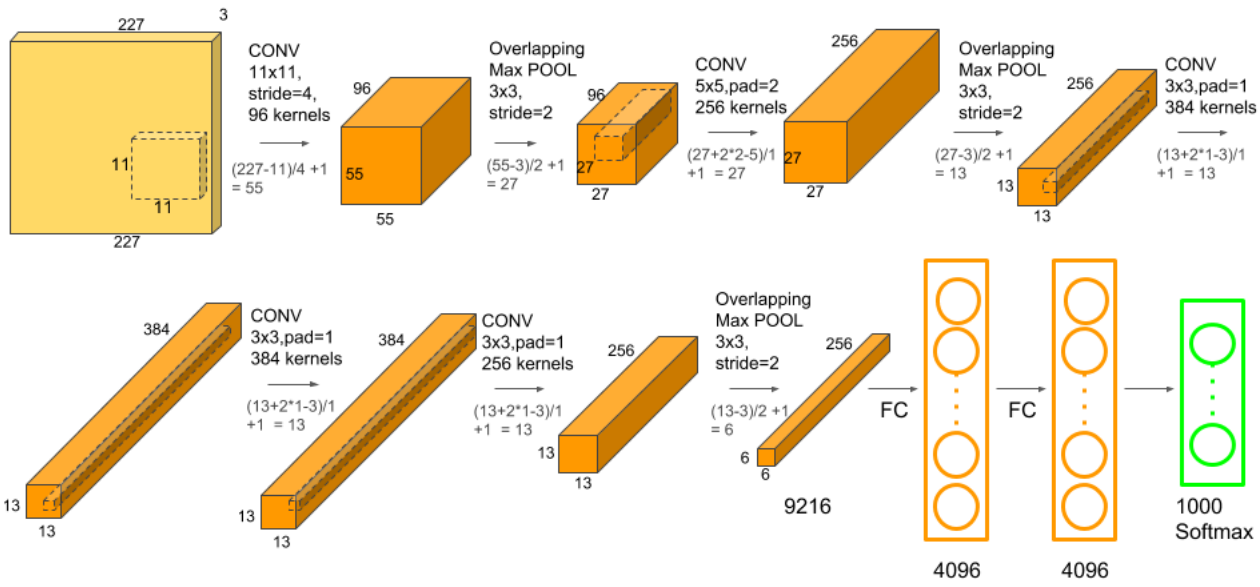


Figure 3.11: AlexNet Architecture

3.7 VGG

Karen Simonyan and Andrew Zisserman from the University of Oxford introduced the VGG in 2014. Unlike AlexNet, VGG uses large filter sizes (11 x 11 and 5 x 5 in the first and second convolutional layers, respectively).

The input image is passed to a stack of convolutional layers, the stride operation of the convolution layers is fixed to 1, pooling is imposed using five max-pooling layers of

size (2 x 2) with a stride of 2. Finally, there are three fully connected layers.

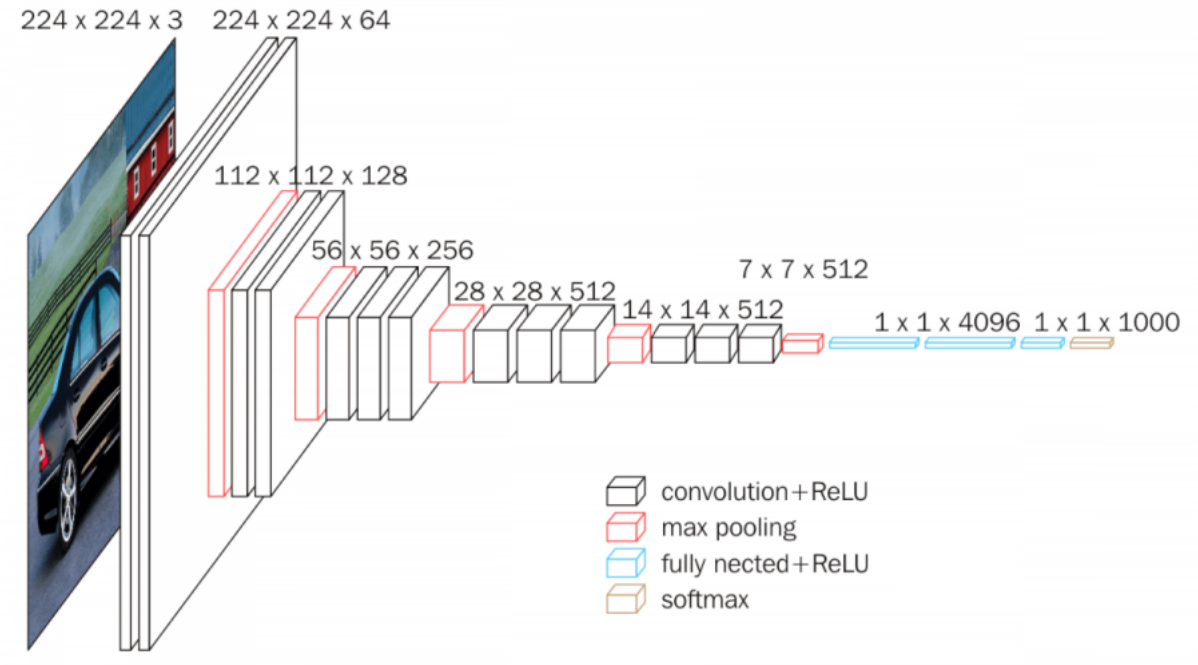


Figure 3.12: VGG Architecture

3.8 GoogleNet

The GoogleNet, or as famously known as the Inception network, is considered a significant breakthrough in Neural Networks and CNN. GoogleNet, as the name implies, was developed by a team at Google.

The main idea behind this architecture is to have filters with multiple sizes that can operate on the same level to solve the problem of overfitting when having many deep layers.

The building blocks of GoogleNet consist of a set of filters of different sizes: (1×1), (3×3), and (5×5) and max-pooling of size (3 x 3).

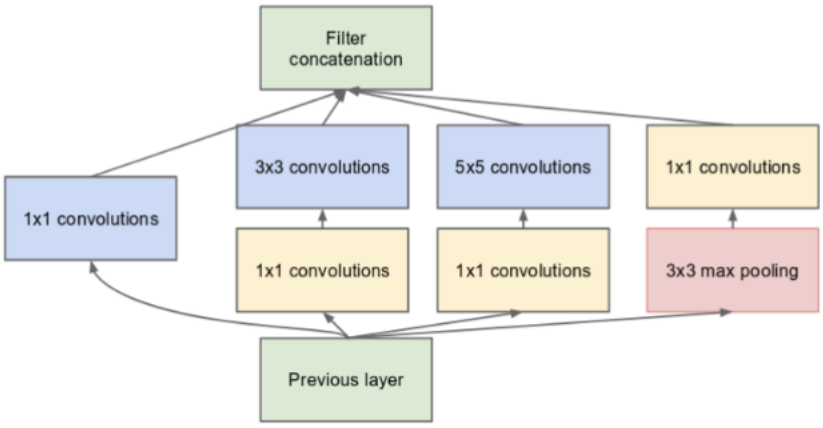


Figure 3.13: GoogleNet Architecture Building Block

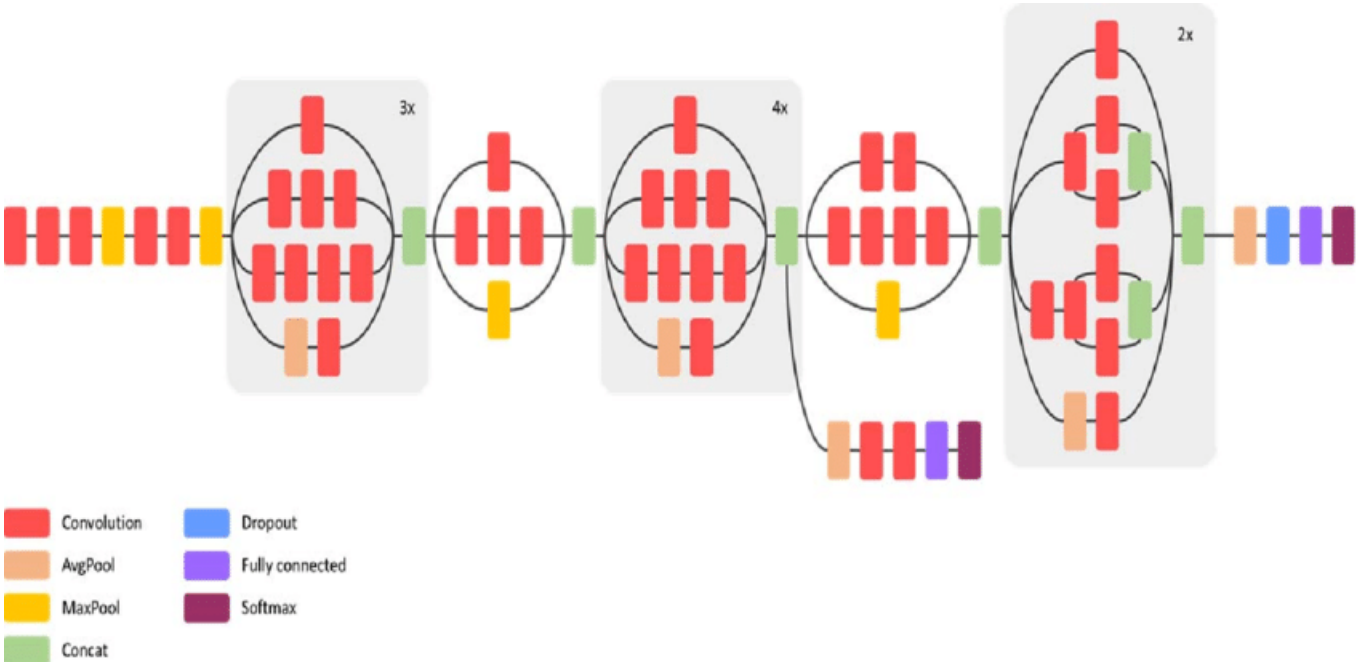


Figure 3.14: inception v3 Architecture

3.9 ResNet

Residual Network (ResNet) was first introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. The ResNet was remarkably successful and won the ILSVRC 2015 classification competition with a top-5 error rate of 3.57

As a result of the residual block structure, the main building block of the residual network, network architecture can go very deep.

The need for adding more layers and going deeper into the network architecture comes from the fact that these layers can progressively learn more complex features. But it has been found that going deeper after a certain threshold with the traditional Convolutional neural network model can lead to many problems that will affect the model accuracy.

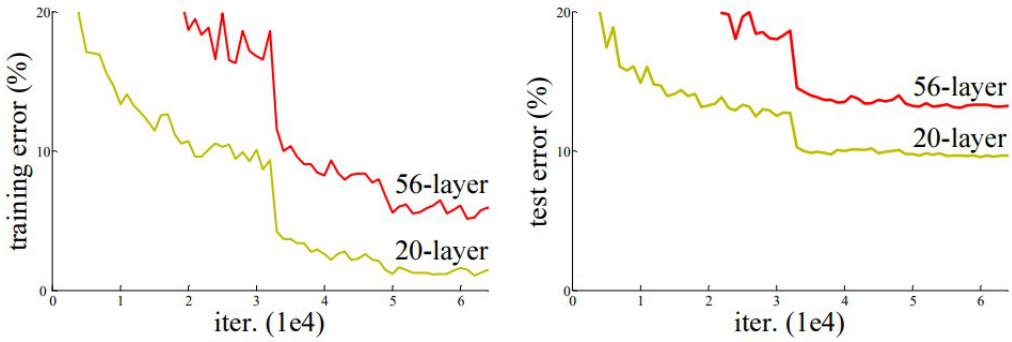


Figure 3.15: Training and Testing errors

We can observe that in both the training and testing, the error percentage is higher for the 56-layer than the 20-layer.

ResNet was able to solve the problem of training a very deep network by using the Residual Blocks.

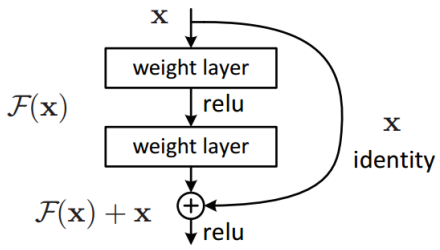


Figure 3.16: ResNet Residual Block

As we can see in figure(2.16) there is a direct connection that skips some layers; this connection is called the skip connection. The skip connections solve the vanishing gradient problem in deep neural networks by allowing this shortcut path for the gradient to flow through. Moreover, the skip connection enables the model to learn the identity functions ensuring that the higher layer will perform at least as well as the lower layer.

ResNet architecture uses a 34-layer plain network architecture inspired by VGG-19, then adds the shortcut connection. These added connections convert the architecture into the residual network.

There are other ResNet implementations with varying depths of 50, 101, and 152 layers.

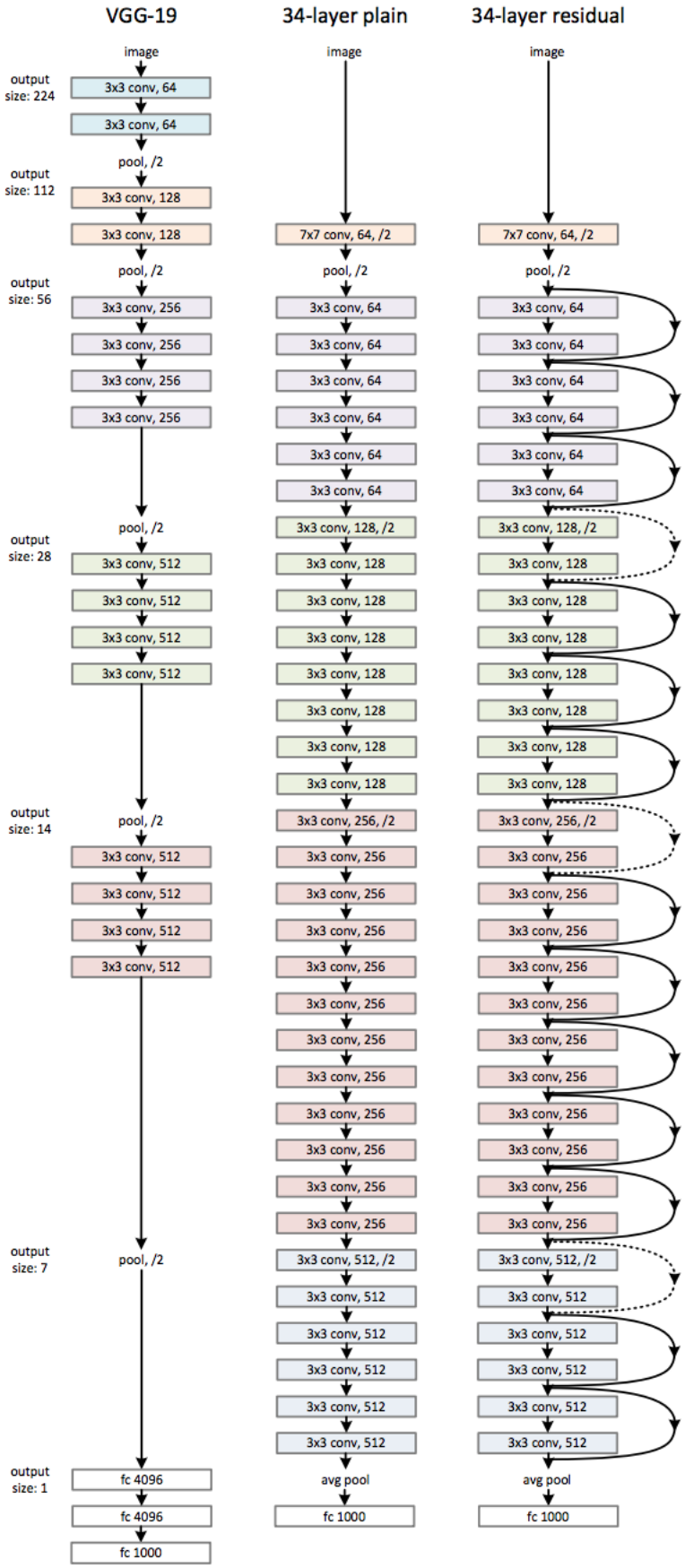


Figure 3.17: ResNet Architecture

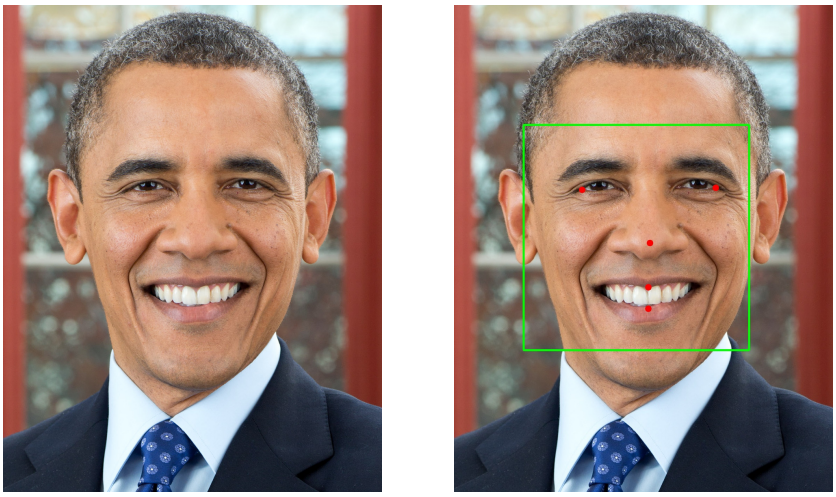
Chapter 4

Face Detection

4.1 Overview

As described previously, face detection is the first stage in our automated face recognition system, and it is considered an essential part of any facial analysis system.

Face detection is a computer vision technology that aims to detect the frontal faces of the human from digital images or videos frames and localize the detected face and some facial landmarks. As we can see in figure 4.1, the input is an image, and the output is the image with a bounding box around the face and some of the face landmarks (left eye, right eye, nose, upper lib, and lower lib).



(a) Input Image (b) Output Image

Figure 4.1: Face Detection

Due to the expanding interest in face recognition systems, face detection algorithms are used in many real-life applications. To date, many face detection algorithms have been evolved and studied for years to get better performance.

Detecting faces in images is simply solved by humans. However, it has historically been challenging for machines to do, given the dynamic nature of faces. For example, faces must be detected regardless of orientation or angle they are facing, light levels, facial clothing and accessories, hair color, facial hair, makeup, age, and so on.

Face detection models and algorithms performance have progressed from rudimentary computer vision techniques to advanced machine learning (ML) to sophisticated deep neu-

ral networks and related technologies. Nowadays, there are a lot of models and algorithms that have achieved tremendous performance for the face detection problem.

In the next section, we will review some of the state-of-the-art algorithms for face detection.

4.2 State-Of-The-Art Algorithms

4.2.1 Viola-Jones Face Detector

In 2001, Paul Viola and Michael Jones presented the Viola-Jones Object detector [9], a fast and accurate object detector model that can perform well with human faces. The Viola-Jones model combines four main concepts: Haar-like Features, Integral Images, the AdaBoost Algorithm, and the Cascade Classifier to develop a model for object detection.

Haar-like Features:

A Haar-like feature is a set of dark cells and light cells. By summing the pixel values of the light cells and subtracting the values of the dark cells, the output is a single value that can be used to know some useful information from the input image such as edges, straight lines, and diagonal lines.

There are many different haar-like features; however, Viola-Jones adopted only the four features shown in figure(3.2).

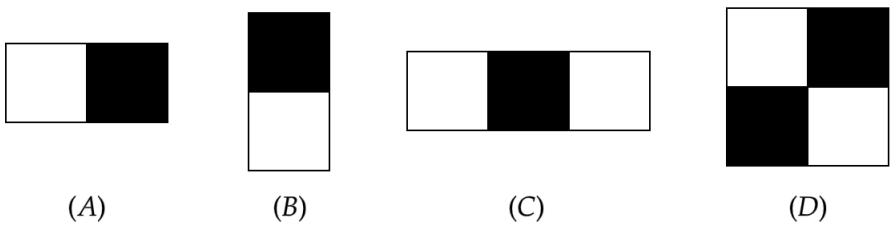


Figure 4.2: Haar-like Features

Integral Images: For each computation in Haar-feature, we may need to obtain every single pixel in the features areas. This step can be bypassed by applying integral images that the value of each pixel is equal to the summation of gray values above and left in the image. Therefore, it only calculates the pixel value for four pixels lookups from the integral image.

AdaBoost Algorithm:

The AdaBoost algorithm is a machine learning algorithm that, given a set of features, it selects the best subset of those available features.

The algorithm output is a classifier called Strong Classifier; the strong classifier is a combination of Weak classifiers. In each iteration, the algorithm calculates the error rate

of the features and then chooses the feature with the lowest error value.

A Cascade Classifier is a multi-stage classifier that allows for the development of a classifier that can be used to reject most negative inputs (not face images) quickly and focus more on positive inputs (face images).

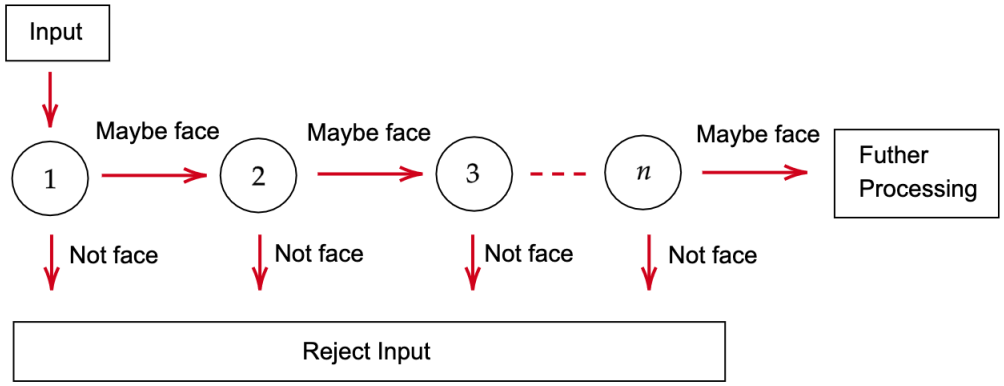


Figure 4.3: Cascade Classifier

4.2.2 MTCNN

MTCNN stands for Multitask Convolutional Neural Network. It is based on a cascade framework. Kaipeng Zhang et al. proposed this algorithm in their paper 'Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks' [10].

The overall pipeline of this approach is shown in Fig. 1. Given an input image, the model initially resize it to different scales to build an image pyramid, which is the input of the following three-stage cascaded framework.

1. **P-Net:** The Proposal Network (P-Net), is a fully convolutional network, aims to produce candidates for face windows and their bounding box regression vectors. Then, based on the predicted bounding box regression vector values, the candidates are calibrated. After that, a layer of non-maximum suppression is implemented to suppress the overlapped candidates.

2. **R-Net:** The Refine Network (R-Net) receives the output of the P-Net stage as input, performs additional refine and reject more false candidate predictions, and finally performs a non-maximum suppression.

3. **O-Net:** The Output Network (O-Net) is similar to the R-Net stage. However, this stage aims to identify face regions with more supervision. Mainly, the network will output the bounding box plus five facial landmarks' locations.

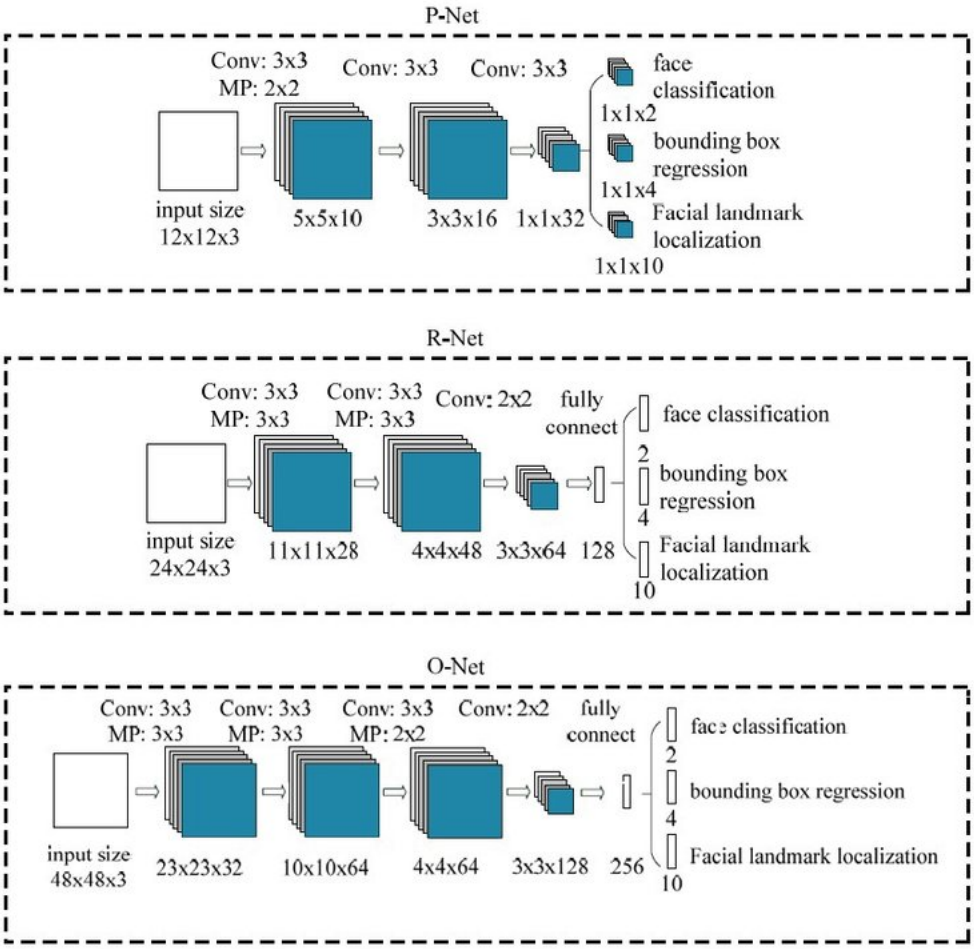


Figure 4.4: MTCNN Stages Architecture

4.2.3 Dlib CNN

Dlib is an open-source library that provides a stable environment for developing software based on machine learning and deep learning in C++ to solve complex problems. The algorithm uses a Convolution Neural network that allows it to detect faces, locate some facial landmarks, and overcome several limitations of other face detection algorithms as the CNN features give the algorithm an edge.

Dlib library is based on linear algebra with Basic Linear Algebra Subprograms (BLAS). It uses the Bayesian networks and Kernel-based algorithms for classification, clustering, detection, and regression. The machine learning toolkit is used to provide a high and straightforward modular architecture for kernel-based algorithms.

Dlib face detection algorithm is easy to implement and provides high accuracy results for frontal faces in various orientations and angles.

Chapter 5

Feature Extraction

5.1 Overview

The feature extraction stage is the most critical in the automated face recognition system; it extracts a feature vector we call the embedded vector from the detected face. The embedded vector must contain information that is enough to represent each face and must have a discriminative feature to distinguish between different faces of different identities.

In many face recognition models, image database massive scale led the feature to be spread, and that caused the boundary between the classes to be blended at the edges.

The goal is to design appropriate loss functions that enhance discrimination power. In the Ideal case, as we can see in 5.1, the extracted face feature should have the maximal intra-class distance smaller than the minimal inter-class distance under a suitably chosen metric space.

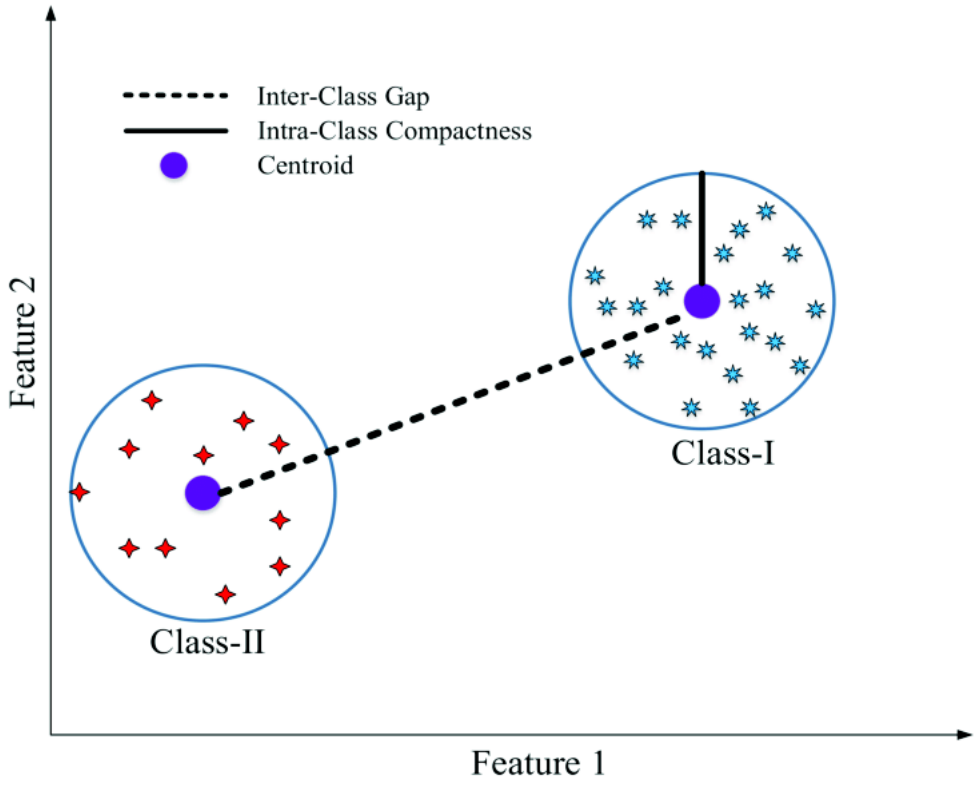


Figure 5.1: Inter-Class and Intra-Class Distances

With the rise of deep learning in recent years, we have witnessed great success in face recognition systems. Thanks to advanced network architectures [12],[13],[16],[11] and discriminative learning approaches [15],[14],[17], deep CNNs have promoted face recognition performance to an unprecedented level.

In table 5.1 we mention some of the state-of-the-art algorithms based on deep learning and their accuracy when tested on the Labeled Face in the Wild (LFW) dataset.

Method	**Authors**	**Year**	**Architecture**	**Verif. Metric**	**Accuracy** (%)
DeepFace	Taigman et al.	2014	CNN-9	Softmax	97.35 ± 0.25
DeepID3	Sun et al.	2015	VGGNet	Contrastive Softmax	99.53 ± 0.10
FaceNet	Taigman et al.	2015	GoogleNet	Triplet Loss	99.63 ± 0.09
Center Loss	Wen et al.	2016	LeNet	Center Loss	99.28
SphereFace	Liu et al.	2018	ResNet-64	A-Softmax	99.42
Cosface	Wang et al.	2018	ResNet-64	LMCL	99.73
ArcFace	Deng et al.	2019	ResNet-100	ArcFace	99.83

Table 5.1: Stae-Of-The-Art Face Recognition Algorithms Performance

5.2 Available Datasets for Face Recognition

For training and testing the performance of object recognition systems in general and face recognition systems in particular, image benchmark datasets must be available and accessible to the public. In this section, we will mention some of the datasets for testing the performance of face recognition systems.

5.2.1 FRGC Dataset

The Face Recognition Grand Challenge (FRGC) [1] was produced at the University of Notre Dame from 2004 to 2006 to improve the performance of the face recognition system. The dataset contains 50,000 images divided into a training set and validation set. A subject session contains four controlled still images, two uncontrolled images, and a 3D image, as can be seen in figure 5.2.

Figure 5.2: FRGC Dataset samples: controlled stills, uncontrolled stills, and 3D shape.

5.2.2 LFW Dataset

Huang et al. created the Labeled Faces in the Wild dataset (LFW) [2] in 2007 to study the problem of unconstrained facial recognition, that is the variation in posture, expression, skin color, lighting, facial clothing, hair color and style, image quality, and other parameters. The dataset contains 13,233 images of 5749 different identities. each image is (250x250).

5.2.3 CASIA-WebFace Dataset

Yi et al. created this dataset at the Institute of Automation, Chinese Academy of Sciences (CASIA) [3]. It is a large-scale dataset containing 494,414 face images of 10,575 different identities collected from the IMDB website.

5.2.4 MegaFace Dataset

In 2016, Shlizerman et al. created the MegaFace database [4]. This dataset contains 1,027,060 face images of 690,572 different identities. MegaFace challenge uses a gallery to test face recognition systems (identification and verification) performance.

5.2.5 Ms-Celeb-M1 Dataset

Ms-Celeb-M1 [5] dataset was developed by Microsoft in 2016, it contains around 10 million facial images of 100,000 celebrities collected from the web with the intention of improving the face recognition systems.

5.2.6 VGGFACE2 Dataset

In 2016, the Visual Geometry Group (VGG) created the VGGFACE [6] dataset at the University of Oxford, which contains around 2.6 million face images of 2.6 thousand identities. Then, in 2017, Cao et al. created VGGFACE2 dataset [7], a large-scale face dataset. It was collected from Google Images with a wide range of poses, ages, and ethnicity. It contains 3.31 million facial images of 9131 different identities. VGGFACE2 has two categories: a training set containing 8631 classes and a testing set containing 500 classes. VGGFACE2 has two template annotations that are used to allow evaluation over pose and age:

- Pose template: including five faces per representing a consistent pose (frontal, profile, or three-quarter view) for 9 K facial images of 1.8 K templates.
- age template: 400 templates (five faces per template with either an apparent age below34, 34, or above) with 2 k facial images.

Figure 5.3: Example Images from the VGGFACE2 dataset.

5.2.7 LFR Dataset

Left-Front-Right (LFR) dataset [8] was developed by Elharrouss et al. from Qatar University in 2020. this dataset was developed to deal with the problem of pose variation. A CNN model for pose estimation is created and trained with a local dataset assembled from three standard datasets: LFW, CFP, and CASIA-WebFace. LFR dataset contains 542 identities; each identity have images of left (1-100 images), front (50-260 images), and right (1-100 images).

5.3 State-Of-The-Art Algorithms

5.3.1 FaceNet

FaceNet [14] is a unified system for face verification, recognition, and clustering. it wa introduced by F. Schroff, D. Kalenichenko, and J. Philbin In 2015. This method aims to extract an embedding vector for each input image using a trained CNN network. The network is trained so that the square L2 distance of all embedding vectors in the embedding space simulates the similarity between the inputs; that is, faces of the same identity have a small distance while faces of different identities have a large distance.

To achieve this kind of discriminative feature, this approach employs the triplet loss function. The term Triplet in triplet loss function comes from the fact that the network is trained using three sets of images, an anchor image, a positive image, and a negative image.

The Positive image is an image that has the same identity as the anchor image, while the negative image is an image of a different identity. The Triplet Loss, as can be seen in figure 5.4 aims to minimize the distance between the anchor image and a positive image and maximizes the distance between the anchor image and a negative image.

$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ \tag{5.1}$$

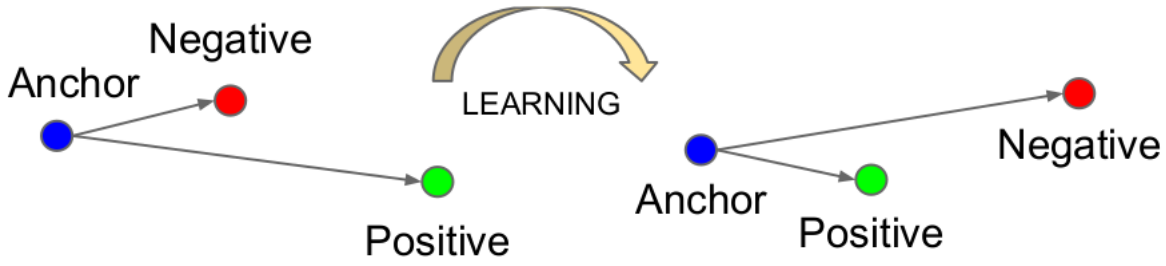


Figure 5.4: FaceNet Triplet Loss Function

The downside with this model is that when selecting the three image set for training, there is a combinational explosion in the number of face triples that cause a significant increase in the number of iterations in each step.

Moreover, we want to select the three images so that the positive and negative distances are similar, that is, the maximum distance between positive and anchor close to the minimum distance between the negative and the anchor. This process is quite a tricky problem for effective model training.

The FaceNet model is trained using 200M training faces of 8M different identities, and when tested on the Labeled Faces in the Wild (LFW) dataset, it achieved a performance accuracy of 99.63±0.09 %.

5.3.2 Center Loss

In 2014, Facebook introduced its model DeepFace [18]. DeepFace model was based on the softmax loss function.

Softmax based models calculate the distance between what the distribution of the output should be (ground truth) and what the original distribution really are. and then normalize a vector of logits (output of last FC layer) to be a probability distribution.

$$L_{softmax} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{W_{yi}^{T}x_i+b_{yi}}}{\sum_{j=1}^{N}e^{W_{yj}^{T}x_i+b_{yj}}} \tag{5.2}$$

The problem with these models is that they do not enforce separation between classes, as we can see in figure 5.5 the classes are closely clustered.

To solve this issue, Wen et al. introduced a discriminative feature learning approach for deep face recognition in 2016 [19], this approach was based on a new loss function based on softmax loss function called the center loss.

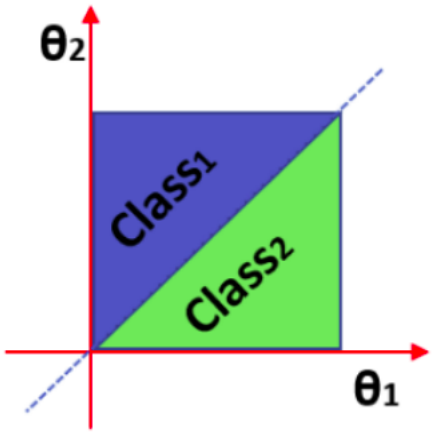


Figure 5.5: Softmax Loss Function and the distance between classes

The center loss aims to calculate the center of each class and penalizes the distance between the feature and its corresponding class center. This can achieve intra-class compactness and inter-class disparity.

$$L_c = \frac{1}{2}\sum_{i=1}^{m}\|x_i - C_{yi}\|_2^2 \tag{5.3}$$

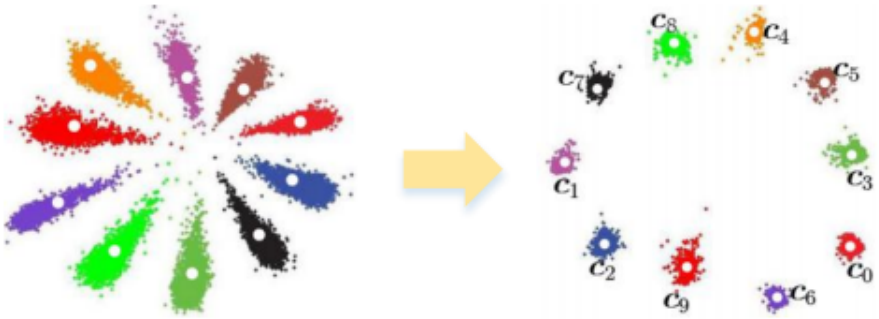


Figure 5.6: Center Loss

However, measuring the center point for each class is computationally expensive because we need to calculate the distance between all features to find the center. Furthermore, it is not possible to carry out this computation ahead of time; we need to do all that during the training and recalculate and redefine for each batch.

5.3.3 SphereFace

In 2018, Liu et al. introduced SphereFace [20], a deep Hypersphere Embedding for Face Recognition; this model is based on the softmax loss function with some modifications.

The author of SphereFace claims that features learned by softmax loss have an intrinsic angular distribution. SphereFace utilizes this angular distribution by imposing a discriminative constrain in a hypersphere manifold, allowing the intra-class and inter-class feature to be controlled by a parameter m. This method is called Angular Softmax "A-softmax".
Starting from the sofmax equation (equation 5.14) We can say:

$$W_i^T x + b_i = \|W_i^T\| \|x\| \cos(\theta_i) + b_i \tag{5.4}$$

By normalize the weights and zero the biases:

$$\|W_i\| = 1 \quad , \quad b_i = 0 \tag{5.5}$$

The modified loss function can be represented as:

$$L_{modified} = \frac{1}{N} \sum_i -\log \frac{e^{\|x_i\| \cos(\theta_{yi})}}{\sum_j e^{\|x_i\| \cos(\theta_{yj})}} \tag{5.6}$$

The decision boundaries can significantly affect the feature distribution, so the basic idea is to manipulate decision boundaries to produce an angular margin.

In the case of binary classification, assume a learned feature x from class 1 is given and (i) is the angle between x and W_i, it is known that the modified softmax loss requires cos(1) > cos(2) to correctly classify x.

But what if we instead require cos(m*1) > cos(2) where (m is an integer 2) in order to correctly classify x, this will essentially make the decision boundary more stringent than previously.
By directly formulating this idea into the modified softmax loss Equation, we have:

$$L_{ang} = \frac{1}{N} \sum_i -\log \frac{e^{\|x_i\| \cos(m * \theta_{yi})}}{e^{\|x_i\| \cos(m * \theta_{yi})} + \sum_{j \neq i} e^{\|x_i\| \cos(\theta_{yj})}} \tag{5.7}$$

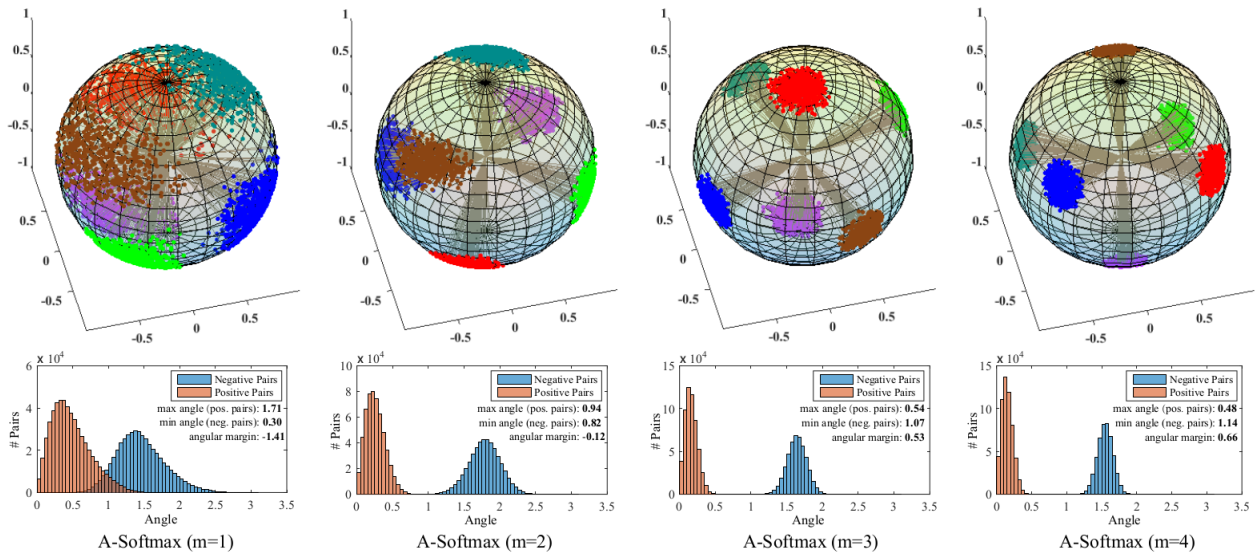


Figure 5.7: SpereFace 3D features projection and angle distribution

Figure 5.7 illustrates the learned feature and its 3D projection in the unit sphere for different values of m. we can see that as the value of m goes higher, the discriminative power of the leaned features gets better. The figure also shows the angle distribution for different values of m.

5.3.4 CosFace

CosFase [21] was introduced in 2018, and like SphereFace, CosFace adopted the idea of utilizing softmax natural angular distribution but introduced another angular margin technique called Large Margin Cosine Loss "LMCL" that can better maximize inter-class variance and minimize intra-class variance.
If we consider sofmax loss function (equation 5.14) we can say that:

$$W_i^T x + b_i = \|W_i^T\| \, \|x\| \cos(\theta_i) + b_i \tag{5.8}$$

By L2 normalization and zeroing the bias term:

$$\|W_i\| = 1 \quad , \quad \|x\| = 1 \quad , \quad b_i = 0 \tag{5.9}$$

The modified loss function can be represented as:

$$L_{ns} = \frac{1}{N} \sum_i -\log \frac{e^{s \cos(\theta_{yi})}}{\sum_j e^{s \cos(\theta_{yj})}} \tag{5.10}$$

This model only emphasizes correct classification but does not enforce discriminative features. To introduce the margin, CosFace implements the same idea of manipulating decision boundaries to produce angular margin.

$$C1 : \cos(\theta_1) \geq \cos(\theta_2) + m \tag{5.11}$$

$$C2 : \cos(\theta_2) \geq \cos(\theta_1) + m \tag{5.12}$$

By directly formulating this idea into the LMCL loss Equation, we have:

$$L_{lmc} = \frac{1}{N} \sum_{i} -\log \frac{e^{s(\cos(\theta_{yi}) - m)}}{e^{s(\cos(\theta_{yi}) - m)} + \sum_{j \neq i} e^{s\cos(\theta_{yj})}} \tag{5.13}$$

Figure 5.8 illustrates the extracted features of 8 identities using the softmax loss and Large Margin Cosine Los (LMCL) using different values of m (m = 0.0, m = 0.1, m = 0.2), the features are represented in the euclidean space and the angular space.

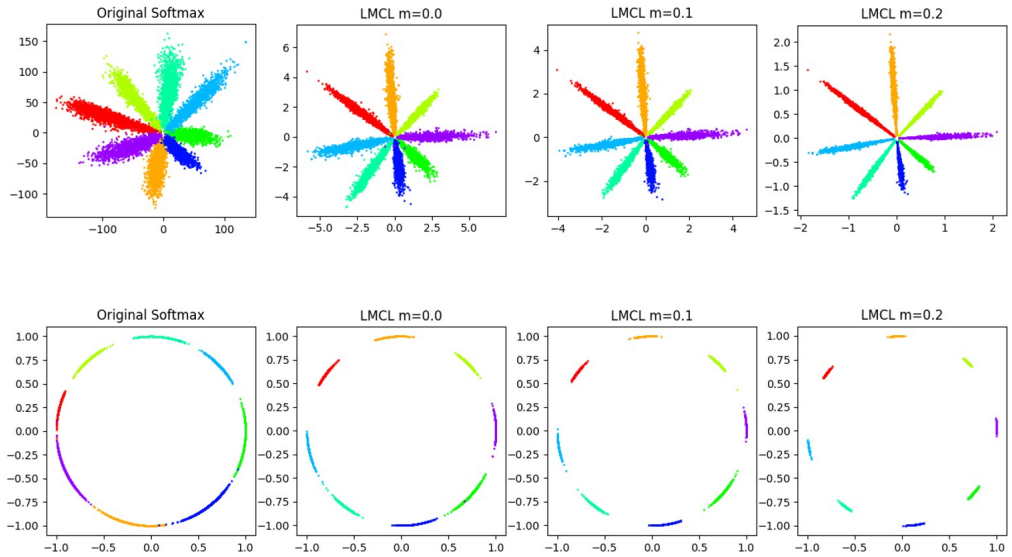


Figure 5.8: Softmax Loss Vs LMCL with different m value

5.3.5 ArcFace

In 2019, Deng et al. Introduced ArcFace [22], an Additive Angular Margin Loss for Deep Face Recognition. Similar to SphereFace and CosFase, ArcFace utilizes softmax natural angular distribution but introduces another angular margin technique. The authors of ArcFace proposed an Additive Angular Margin Loss function further to improve the face recognition model's discriminative power and stabilize the training process.

Authors claims that ArcFace consistently outperforms the state-of-the-art models and can be easily implemented with negligible computational overhead.

Same As CosFace and SphereFace, ArcFace starts from the softmax loss function:

$$L_1 = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{W_{yi}^T x_i + b_{yi}}}{\sum_{j=1}^{N} e^{W_{yj}^T x_i + b_j}} \tag{5.14}$$

For simplicity:

$$b_j = 0 \tag{5.15}$$

$$W_j^T x_i = \|W_j^T\| \, \|x_i\| \cos(\theta_j) \tag{5.16}$$

Where θ_j is the angle between the weight W_j and the feature x_i . then the individual weights are fixed by L2 normalization.

$$\|W_j\| = 1 \tag{5.17}$$

Also embedding feature is fixed by L2 normalization and rescaled to s.

$$\|x_i\| = s \tag{5.18}$$

Now the prediction depends only on the angle between the weight and the feature. Hence, the learned embedding features are distributed on a hypersphere of a radius s.

$$L_2 = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cos(\theta yi)}}{e^{s\cos(\theta yi)} + \sum_{j=1, j\neq yi}^{n} e^{s\cos(\theta j)}} \tag{5.19}$$

Now, to intensify the intra-class compactness and inter-class disparity, an angular margin penalty 'm' is added between x_i and W_{yi} Since the proposed additive angular margin penalty is equal to the geodesic distance margin penalty in the normalized hypersphere, this method is called ArcFace [22].

$$L_3 = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cos(\theta yi + m)}}{e^{s\cos(\theta yi + m)} + \sum_{j=1, j\neq yi}^{n} e^{s\cos(\theta j)}} \tag{5.20}$$

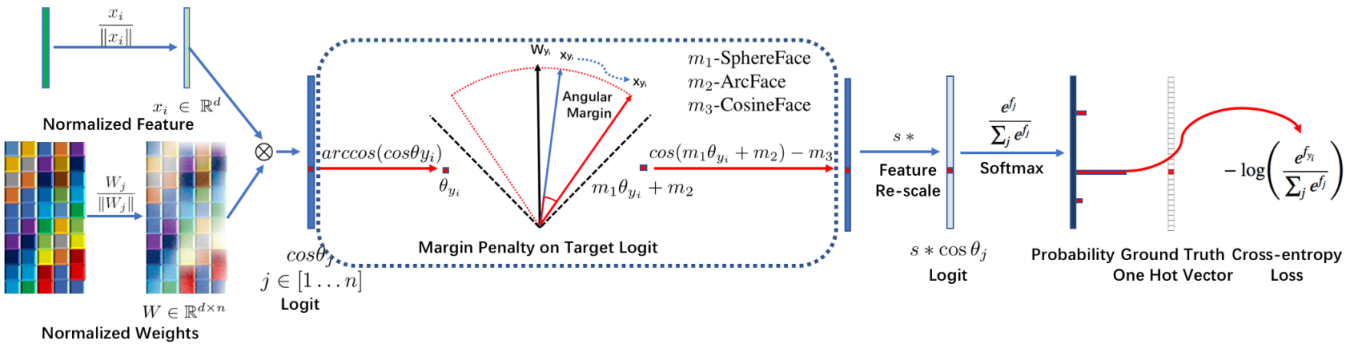
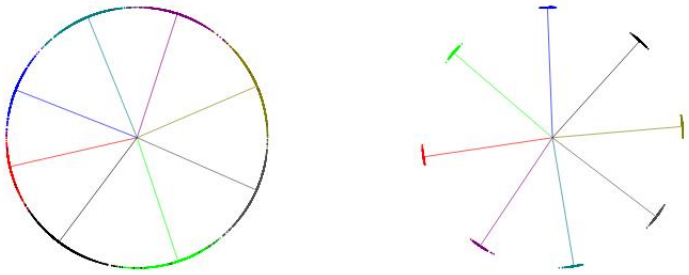


Figure 5.9: ArcFace Loss

Figure 5.9 illustrate the ArcFace Loss. Starting from the normalized feature and the normalized weights, the angle between the feature x_i and the ground truth weight W_{yi} is calculated using the arccos function, then the angular margin m is added, and $cos(\theta_{yi}+m)$ is calculated and multiplied by the feature scale s. Finally, the logits then go through the softmax function and contribute to the cross-entropy loss.

In figure5.10 we represent the classification of 8 different identities containing enough information (around 1500 images per class) trained on two different networks with softmax and ArcFace loss.

We can see the in case of softmax loss, the boundries between classes are blended at the edges, while in the case of ArcFace the classes are compact.



(a) Softmax (b) ArcFace

Figure 5.10: ArcFace Vs Softmax

The advantages of the proposed ArcFace can be summarised as follows:

- Effective: ArcFace achieves state-of-the-art performance on ten face recognition benchmarks, including large-scale image and video datasets.
- Easy: ArcFace only needs several lines of code and is extremely easy to implement in all frameworks, like MxNet, Pytorch, and Tensorflow . Furthermore, ArcFace

does not need to be combined with other loss functions to have stable performance and can easily converge on any training datasets.

- Efficient: ArcFace only adds negligible computational complexity during training. Current GPUs can easily support millions of identities for training, and the model parallel strategy can easily support many more identities.

In SoherFace, CosFace, and ArcFace, three different kinds of margin penalties are proposed. Multiplicative angular margin m1 , additive angular margin m2 , and additive cosine margin m3, all enforce the intra-class compactness and inter-class diversity by penalising the target logit.

Comparing the decision boundaries under the binary classification case. The proposed ArcFace has a constant linear angular margin throughout the whole interval. By contrast, SphereFace and CosFace only have nonlinear angular margins as in figure 5.11.

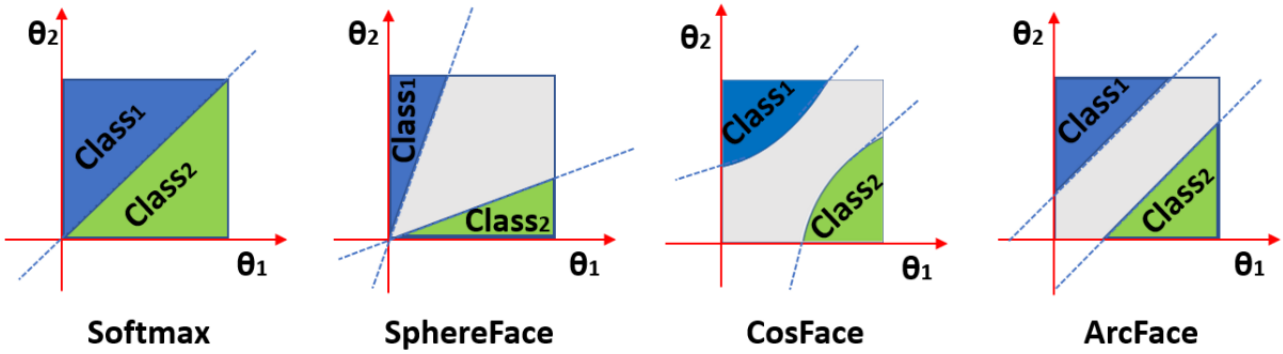


Figure 5.11: Comparesion between Softmax, SphereFace, CosFace, and ArcFace

Chapter 6

Methodology

As mentioned before, the automated face recognition system pipeline consists of three main stages: face detection, feature extraction, and classification. In this chapter, we will go through these stages and our full implementation from receiving the video frame from the camera feed until we output the final label of the person in front of the camera, describing along the way the important functions and the deep neural networks used and its architectures.

However, before that, we will have an overview of the entire development environment, the hardware used and its specifications, and the software tools and packages.

6.1 Hardware Specifications

6.1.1 ROScube-X

As main developing kit, we used ROScube-x RQX-580, it is an Embedded Robotic Controller Powered by NVIDIA Jetson AGX Xavier, the board is running Ubuntu 18.04 LTS, all the system was developed, tested, and deployed on this board.

ADLINK's ROScube-X, a ROS2-enabled robotic controller, powered by the NVIDIA Jetson AGX Xavier module, features an integrated NVIDIA Volta GPU and dual deep learning accelerators, with a wide variety of interfaces including GMSL2 camera connectors for advanced robotic system integration. ROScube-X supports the full complement of resources developed with the NVIDIA JetPack SDK and ADLINK's Neuron SDK, and is specifically suited for robotic applications demanding high-AI computing with minimal power consumption [23].

ROScube-x main features:

- Powerful AI computing for intelligent robotics development
- Excellent performance per watt with power consumption as low as 20 W
- Comprehensive I/O for connecting a wide range of devices
- Ruggedized, secure connectivity with locking USB ports

Figure 6.1: ROScube-x Controller

6.1.2 Intel Realsense Depth Camera D435i

The Intel RealSense D435i is a stereo vision depth camera system. It contains, as can bee seen if figure (6.2) two stereo depth module, RGB module, and Infrared projector.it also have a vision processor with USB 2.0/USB 3.1 Gen 1 or MIPI connection to host processor [24].

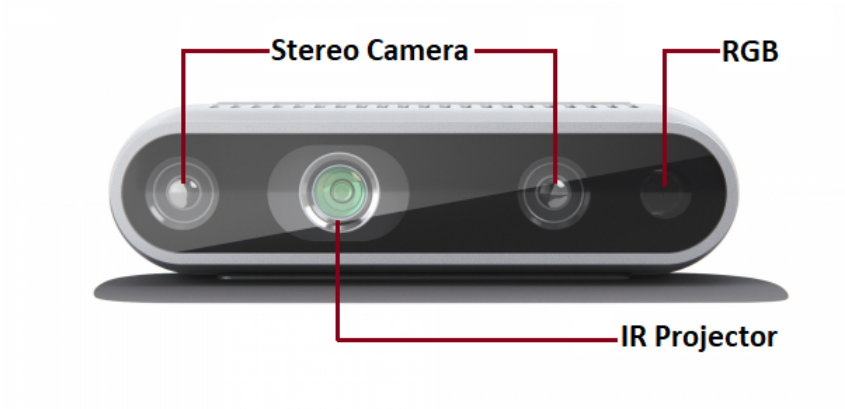


Figure 6.2: Intel Realsense Depth Camera D435i

6.2 Sofware

All the system development was done on the ROScube-X board. The board is running Ubuntu 18.04LTS as an operating system. The main platform used for development is TensorFlow, it is an end-to-end open-source platform for machine learning. The software programming language is Python.

In order to manage Python packages probably, we used a virtual environment and then installed all Python needed packages inside the virtual environment. Below is a list of some of the main Python packages used in the development of our automated face recognition system.

- Tensorflow
- dlib
- opencv-python
- numpy
- os-sys
- scipy
- sklearn
- tkinter
- threaded
- pyrealsense2

6.3 System Development

6.3.1 Camera Input

As we mentioned before, we are using the Intel Realsense Depth Camera D435i module; this module as can be seen in figure (6.2) contains both RGB frame and depth frame. We will use the RGB fame to capture the face image and pass it through our two face detection and feature extraction networks. While the depth frame will be used to calculate the depth distance of the face landmarks detected by the face detection network, based on those distance values, the system will be able to distinguish between a live person or an image placed in front of the camera. We can access most camera functionality by the realsense-viewer, as we can see in figure(6.3) we can access both the RGB and the Depth frames.

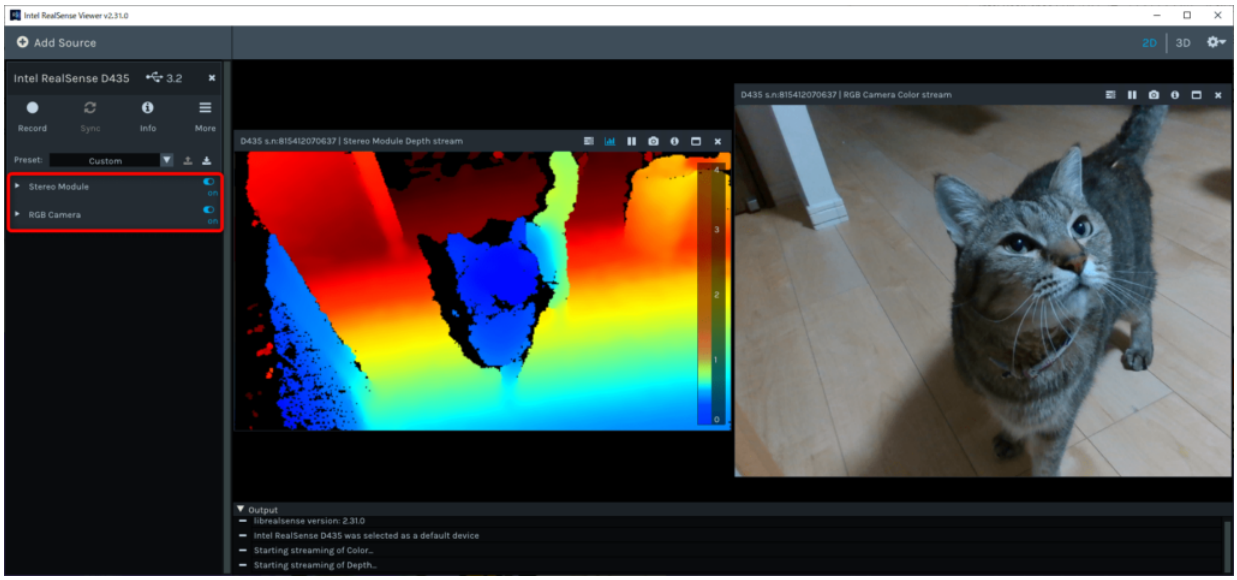


Figure 6.3: Intel realsense-viewer showing both RGB and Depth frames

However, to read these frames into our python code we need the pyrealsense2, it is a python wrapper for Intel RealSense SDK 2.0 provides the Python binding required to access the SDK. The pyrealsense2 was build from the source code using CMAKE and installed in the project virtual environment. After installing the pyrealsense2 package, to read the RGB and Depth frames, we need to configure the pipeline and config parameters, enable the stream, and wait for the frames to be ready to read it in our code, below a piece of code describing this procedure.

```
import pyrealsense2 as rs

pipeline = rs.pipeline()
config = rs.config()

config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

pipeline.start(config)
frames = pipeline.wait_for_frames()

depth_frame = frames.get_depth_frame()
color_frame = frames.get_color_frame()

depth_image = np.asanyarray(depth_frame.get_data())
color_image = np.asanyarray(color_frame.get_data())
```

Now both our RGB frame and Depth frames are ready to be processed.

6.3.2 Dlib face detection

Our choice for the face detection module is Dlib. Dlib contains a wide range of machine learning algorithms designed to be highly modular, quick to execute, and simple to use via a clean and modern C++ API. It is used in a wide range of applications, including robotics, embedded devices, mobile phones, and large high-performance computing environments.

Dlib face detection module is trained to detect faces in images and return the location of the bounding box containing the face and 68 landmarks in the face, such as nose, eyes, upper lip, lower lip, chin, and other lam]ndmarks. Figure(6.4) illustrates the facial landmarks that the Dlib face detection module uses.

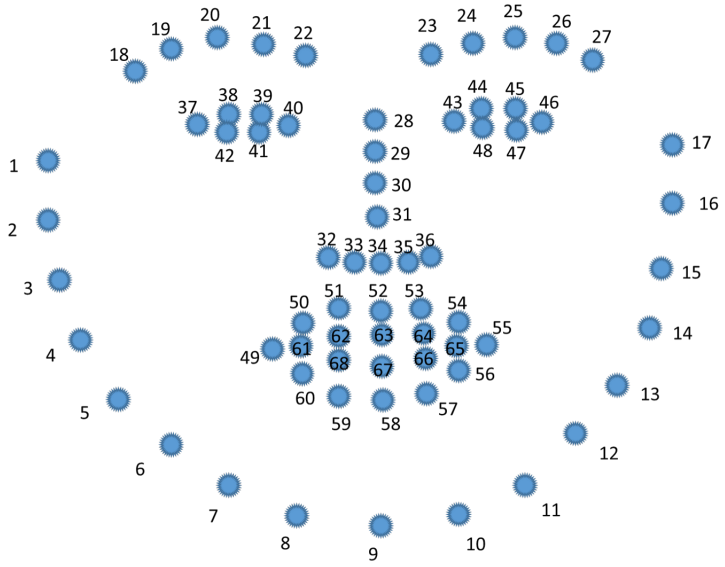


Figure 6.4: Dlib Facial Landmarks

To get the bounding box and the landmarks, three function are used:

- **getLargestFaceBoundingBox:** Receiving the image or video frame containing the face image as input, returns a rectangle object (the bounding box) described by two points, the (top,left) corner and the (bottom,right) corner.
- **getAllFaceBoundingBoxes:** Receiving the image or video frame containing the face image as input, returns an array of rectangle objects.
- **findLandmarks:** Receiving the image or video frame containing the face image and the bounding box as inputs and returns a list of 68 points.

6.3.3 Feature Extraction using ArcFace

As mentioned before, ArcFace achieved a state-of-the-art performance and can be easily implemented with negligible computational overhead. Our choice for the ArcFace

network architecture is ResNet-34. ResNet-34 architecture can be seen in figure(3.17) After the implementation of ResNet architecture, pre-trained weights are loaded into the model. we can see that the number of parameters in this architecture is:
Total params: 34,165,184
Trainable params: 34,139,328
Non-trainable params: 25,856

The ArcFace model requires the input image to have a size of (112x112x3), so the input image is first cropped using the bounding box points and then resized into the target size. Moreover, sincethe image alignment significantly affects the model performance, the image is aligned using some of the facial landmarks, specifically, the left eye, the right eye, and nose locations is used to to rotate the image left or right so the two points (left eye and the right eye) should be at the same horizontal line before feeding it to the ArcFace model.

After alignment, cropping and resizing the input image, we need to normalize the image by dividing each pixel value by 255 so all the pixel values will be in the 0-1 range. All these steps are implemented in a function called creat_emp that receives the image, the Dlib model, and the ArcFace model, and returns the embedded vector which has a size of 512 representing the face in that image.

```
def creat_emp(image, alignment, model):
    target_size = (112, 112)
    if not (image is None):
        bb = alignment.getLargestFaceBoundingBox(image)
        if not (bb is None):
            left = bb.left()
            top = bb.top()
            right = bb.right()
            bottom = bb.bottom()
            land_marks = alignment.findLandmarks(image, bb)
            left_eye = land_marks[36]
            right_eye = land_marks[45]
            nose = land_marks[30]

            image = image[top:bottom, left:right] #Cropping
            image = alignment_procedure(image, left_eye, right_eye, ...
               nose) #Alignment
            image = cv2.resize(image, target_size) #Resizing
            image = (image/255).astype(np.float32) #Normalizing
        if not (image is None):
            embeddings = model.predict(np.expand_dims(image, axis=0))[0]
        return embeddings
```

6.3.4 Selecting Threshold Value

Until now, the system is able to receive the input image from the camera feed, detect if a face is present in the input image, return the bounding box of the detected face, and extract features in the form of an embedded vector of size 512.

As we mentioned before, the extracted features represent the detected face and have a discriminative power to distinguish between different identities based on the distance between the embedded vectors. If this distance is more than a threshold, these will be considered different identities; otherwise, they will be considered the same identity.

To implement this concept, we need to define a metric system for calculating the distance between two vectors and find a way to select the optimal threshold value. For the metric system, we can use either the euclidian distance or the cosine distance.

The euclidian distance can be measured as:

$$d_{euclidian} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{6.1}$$

While the The cosine distance can be measured as:

$$d_{cosine} = \frac{p_i.q_i}{\|p_i\|\|q_i\|} \tag{6.2}$$

Where $p_i.q_i$ is the dot product between the two vectors. Since we are using the ArcFace model, and the ArcFace loss function is based on the Softmax loss function, which has an intrinsic angular distribution. So we decided to adopt the cosine distance as our metric system for calculating the distance between vectors. Below is our implementation of a function called findCosineDistance.

```
import numpy as np

def findCosineDistance(source_representation, test_representation):
    a = np.matmul(np.transpose(source_representation), test_representation)
    b = np.sum(np.multiply(source_representation, source_representation))
    c = np.sum(np.multiply(test_representation, test_representation))
    return 1 - (a / (np.sqrt(b) * np.sqrt(c)))
```

After implementing the findCosineDistance function, we prepared a subset of the LFW dataset containing 1021 images. Then we created two numpy array distances and identical, going through all the images in the dataset we calculate the cosine distance and append this value to the distances array, we also check if the two images we calculated the distance between them are of the same person we append the value 1 to the identical array, and 0 otherwise. Now, we select a range for the threshold (from 0.4 to 1 with a step of 0.01)

and we calculate the accuracy based on the values of distances and identical arrays for each threshold in the chosen range. then we select the max value of the accuracy and the corresponding threshold as our optimal value.

```
from sklearn.metrics import accuracy_score

distances = [] # cosine distance between pairs
identical = [] # 1 if same identity, 0 otherwise

for i in range(len(metadata) - 1):
    for j in range(i + 1, len(metadata)):
        distances.append(findCosineDistance(embeddings[i], embeddings[j]))
        identical.append(1 if metadata[i].name == metadata[j].name else 0)

distances = np.array(distances)
identical = np.array(identical)

thresholds = np.arange(0.4, 1.0, 0.01)

acc_scores = [accuracy_score(identical, distances < t) for t in thresholds]
opt_idx = np.argmax(acc_scores)

threshold_opt = thresholds[opt_idx]
```

In figure(6.5) we plotted the accuracy for all thresholds in the range (0.4 to 1 with a step of 0.01)

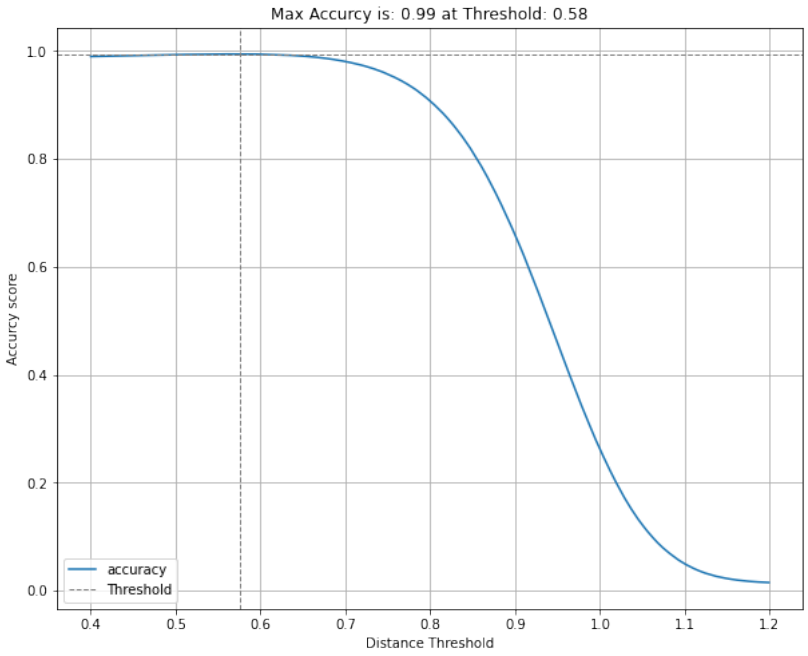


Figure 6.5: Max accuracy and the corresponding threshold

6.3.5 Head Pose Estimation

At this point, we collect images of faces of different identities and store them in a local folder on the board, inside the local dataset folder each identity has a folder of its own with the name of the person and inside it there are the person images. the system will extract the embedded vector for each image in the local dataset and combine all of them in a matrix, we call it the embedded matrix.

However, we want the automated face recognition system to have the ability to collect images of new identities and add them to the local dataset. Although it might be enough to collect a few images of the new face, it is very important that these images are different from each other, that is the pose of the face with respect to the camera is different in each image. The goal here is to have images in different poses, such as looking forward, left, right, up, and down. To be able to do that, the system must understand the head pose in the image, which is known as the head pose estimation problem.

In computer vision, the pose of an object refers to its relative orientation and position with respect to a camera. This problem is often referred to as the Perspective-n-Point (PNP). To calculate the pose of a head in an image we need the following information:

- **2D coordinates of a few points:** We need the 2D coordinates (x,y) of some of the landmark points in the face, in our implementation, we decided to select six points: tip of the nose, chin, left corner of the left eye, right corner of the right eye, left corner of the mouth, and right corner of the mouth. All these point can be taken from the landmark points list returned by our Dlib face detection model as can be seen in figure(6.4).

- **3D coordinates of the same points:** We also need to know the 3D coordinates (x,y,z) of the six 2D points mentioned before. to get these 3D coordinates, ideally, we need a 3D model of the person's face, however, practically a generic 3D model will be sufficient. So we just need to know the 3D locations of our six points in some arbitrary reference frame, in our implementation these points are:

  1. Tip of the nose : (0.0, 0.0, 0.0)
  2. Chin : (0.0, -330.0, -65.0)
  3. Left corner of the left eye : (-225.0f, 170.0f, -135.0)
  4. Right corner of the right eye : (225.0, 170.0, -135.0)
  5. Left corner of the mouth : (-150.0, -150.0, -125.0)
  6. Right corner of the mouth : (150.0, -150.0, -125.0)

  these points are in some arbitrary reference frame. This is called the World Coordinates (The Model Coordinates as per the OpenCV documentation).

- **Camera intrinsic parameters:** here we need to know the focal length of the camera, the optical center in the image, and the radial distortion parameters.

We can transform the 3D points in world coordinates to 3D points in camera coordinates. The 3D points in camera coordinates can be projected onto the image plane (i.e. image coordinate system) using the intrinsic parameters of the camera (focal length, optical center, and radial distortion).

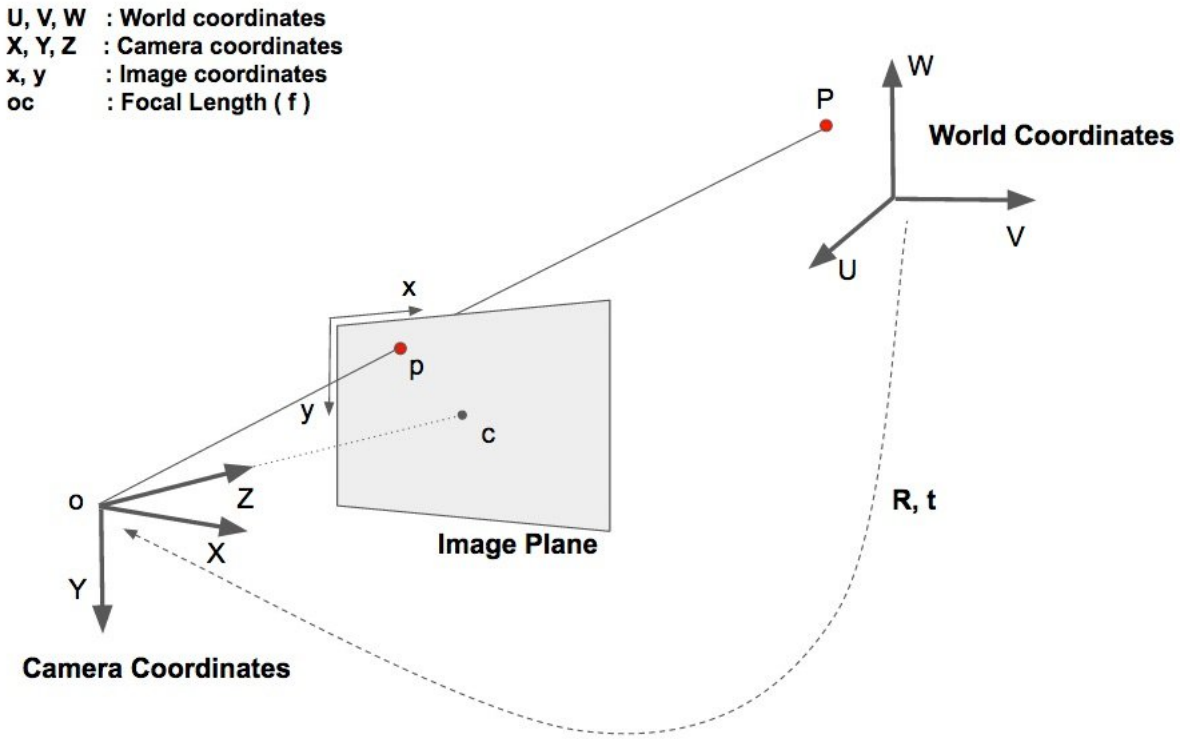


Figure 6.6: Coordinates system in head pose estimation

In Open-CV there is an implementation for the perspective-n-Point problem called solvePnP, it takes as input the 3D points coordinates, the 2D points coordinates, and the camera matrix (a matrix containing the te focal length and the center coordinates), and it returns a success flag, the rotation vector, and the translation vector. Then we will use these vectors along with the camera parameters to project a point that represents the location on the screen the person is looking at.

Then we will connect this point to the tip of the nose point, and we will calculate the length of this line and the angle it makes with the horizontal line, based on the length and the angle we can estimate the head pose.

- Regardless of the angle, if the length is small (less than the distance between the nose and the chin) the pose will be forward.
- If the angle is around zero (0<angle<30 or 330<angle<360) the pose will be right.

- If the angle is around 90 (60<angle<120) the pose will be up.
- If the angle is around 180 (150<angle<210) the pose will be left.
- If the angle is around 270 (240<angle<300) the pose will be down.

This approach is implemented in our automated face recognition system and used when collecting images for new identities, the collected images will be in all five different poses forward, left, right, up, and down.

6.3.6 Graphical User Interface

The purpose of creating a Graphical User Interface (GUI) is to create a friendly user experience with our automated face recognition system and to give the user the ability to easily lunch the camera to detect the faces and check the identity and be able to manage the local dataset and add new personnel.

The implementation of the GUI was done using a Python package library dedicated to GUI development called Tkinter. There are two main functionalities in the GUI, The Recognize, and the Add personnel.

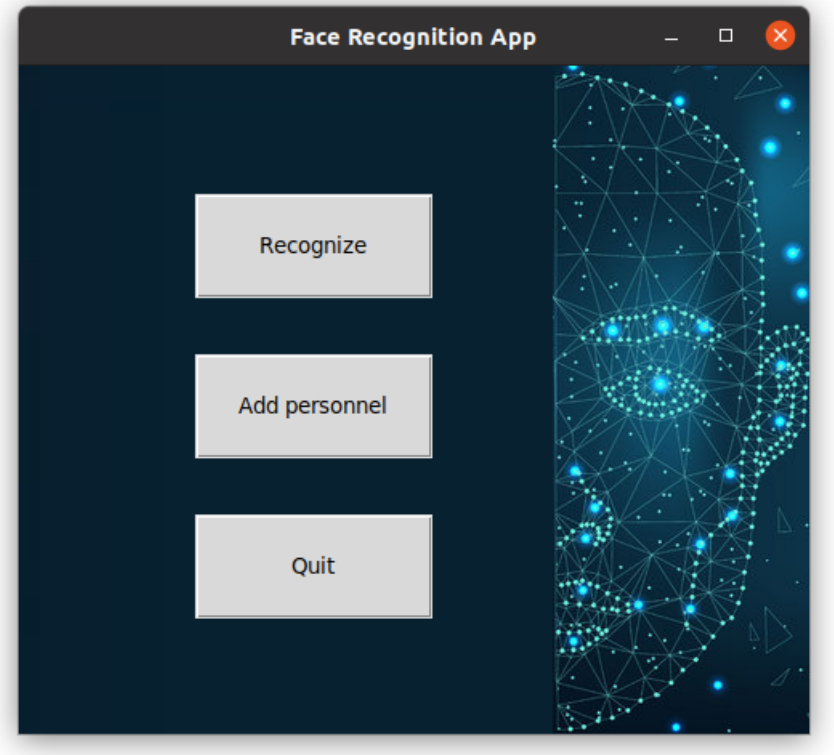


Figure 6.7: Automated Face Recognition GUI

Pressing the Recognize button will lunch the camera, each frame will go through the Dlib face detection, which will return the bounding box and the facial landmarks, then

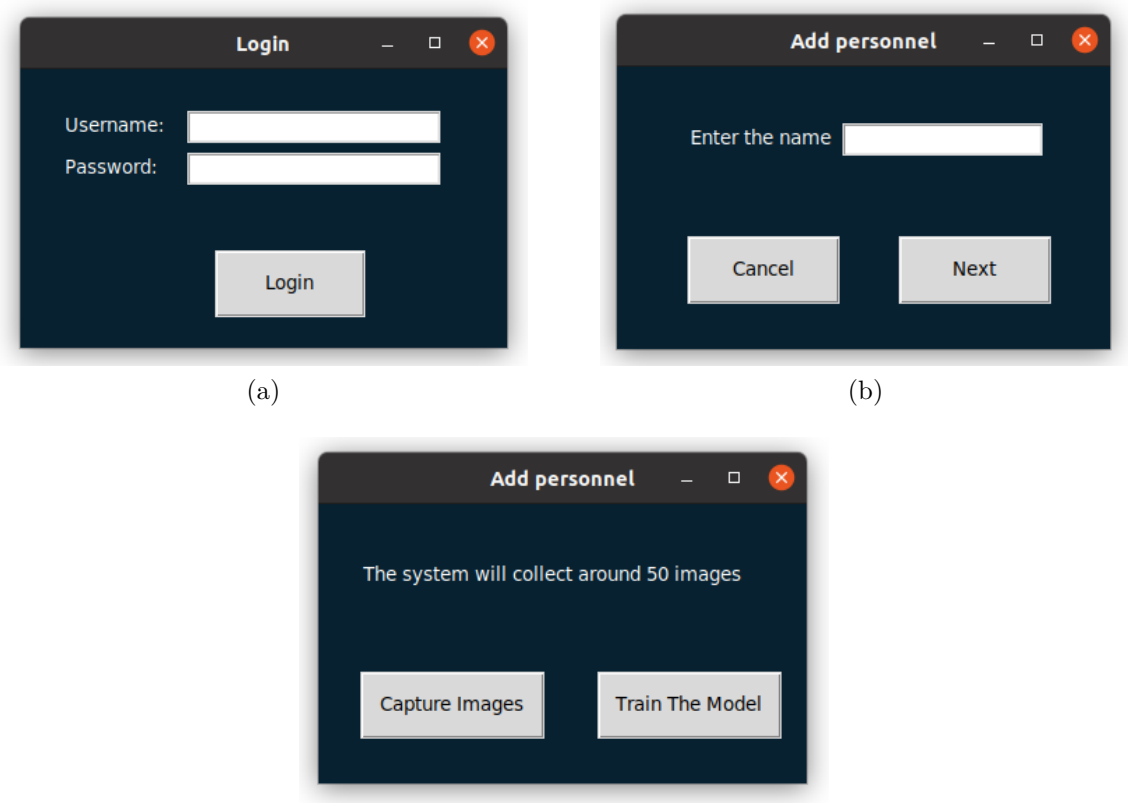
the detected face will go through the feature extraction and output the embedded vector, the cosine distance between this vector and all the vectors stored in the embedded matrix will be calculated, and if the minimum distance is less than the optimal threshold, the corresponding label will be returned and displayed on the bottom of the screen, otherwise, the identity will be considered **unknown**.

On the other hand, when pressing the Add personnel button, a pop-up screen will ask for login credentials, the login screen figure(6.8 (a)) was developed to guarantee that only an authorized person can carry out the task of adding new personnel.

After entering the correct credentials, another pop-up screen will ask the user to enter the name of the new person to be added figure(6.8 (b)). At this point, a folder under the new person's name will be created inside the local dataset folder.

Then a pop-up screen figure(6.8 (c)) will notify the user that the system is going to collect images of their face. if the user presses the collect image button the system will lunch the camera and using the pose estimation function we mentioned before, the system will collect images in five different poses (forward, left, right, up, and down) and store these images in the newly created folder.

Finally, when the user presses the Train the model button, the system will create embedded vectors for all the newly collected images and append these vectors to the embedded matrix.



(a) (b)

(c)

Figure 6.8: GUI, Add personnel

6.3.7 Depth Information

The Intel RealSense depth frame contains the depth distance of each pixel. we want to use this information to make the system able to distinguish between a real person's face and an image of the person's face. To do that we extracted from the depth frame the distance value of our 68 facial landmarks and store them into an array (land_marks_dis). Then we calculated two values, the difference between the maximum distance and the minimum distance, and the variance of all the distance values. Then we collected some data of real faces and others for still images, and a simple SVM model is used to find values for the variance and the maximum and minimum difference to act as separators between the two classes.

When the system captures a new frame to be recognized, first it calculates the array land_marks_dis and these two values. If these values belong to the class of the still image, the system will not continue and it will return that this is not a live person in front of the camera.

Chapter 7

Results and Discussion

7.1 Evaluation Results

First we will represent the evaluation results of our pre-trained feature extraction model (ArcFace) when tested on LFW, YTF, CALFW, and CPLFW datasets.

LFW and YTF are the most used benchmark data sets for the unconstrained face images. ArcFace achieved a 99.83% when tested on the LFW dataset, while on the YTF dataset the performance was 98.02%.

We also present the performance of ArcFace on the recent CALFW and CPLFW dataset, which are datasets with higher pose and age variations for the same set of identities of LFW dataset.

Dataset	Accuracy (%)
LFW	99.83
YTF	98.02
CALFW	95.45
CPLFW	92.08

Table 7.1: ArcFace performance on LFW, YTF, CALFW, and CPLFW datasets.

7.1.1 Results on local dataset

For further testing, a local dataset is created, it consists of 743 images of 15 members from PIC4SeR research group. All images were collected using the GUI of our automated face recognition system and as a result of our pose estimation model, the collected images are in different poses (looking forward, left, right, up, and down).

only five images (one in each pose) were used to train the model, while the remaining images were used for testing the model.

During the testing stage, the recorded measures are essentially focused on the difference between the actual output (ground truth of identity labels) and the model output (predicted labels) on the test set, and the following measures are recorded:

- True positive (TP): The cases in which the model's predicted label is the same label in the ground truth.

- True negative (TN): The cases in which the model's predicted label is unknown, and the identity is actually not in the training data.
- False positive (FP): The cases in which the model's predicted label is different from the label in the ground truth.
- False negative (FN): The cases in which the model's predicted label is unknown, and the identity is actually present in the training data.

Then the **Accuracy** can be measured as the percentage or number of samples classified correctly out of the total samples as shown in equation (7.4).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

then we calculate the accuracy fro a range of threshold values range from 0.3 to 1.2 with step of 0.01, we can see from figure(7.3), the maximum accuracy is **0.9656**, and this value is at threshold of **0.58**

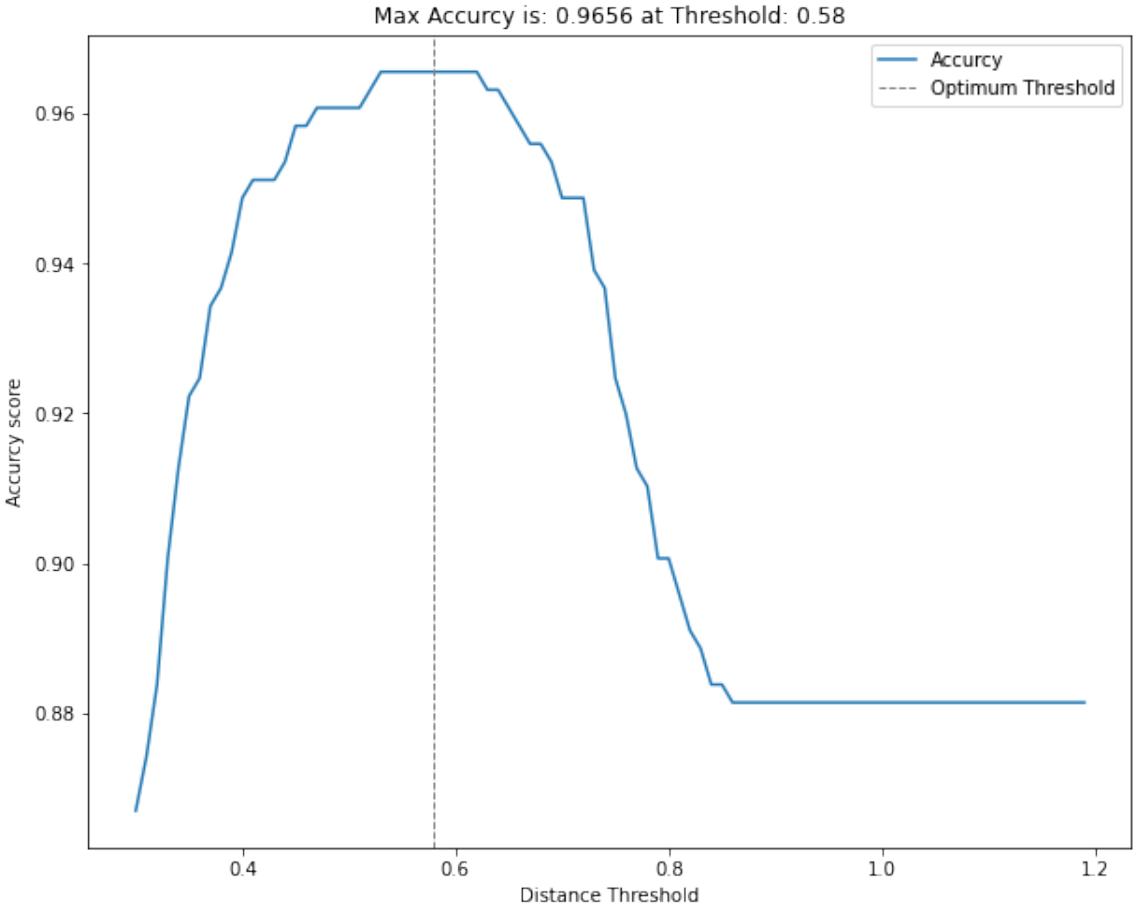


Figure 7.1: model Accuracy

The **Precision** is the ability of the classifier not to label positive sample as negative, and can be calculated as:

$$Precision = \frac{TP}{TP + FP} \tag{7.2}$$

The **Recall**, also called sensitivity, is a measurement of the ability of the classifier to find all the positive samples.

$$Recall = \frac{TP}{TP + FN} \tag{7.3}$$

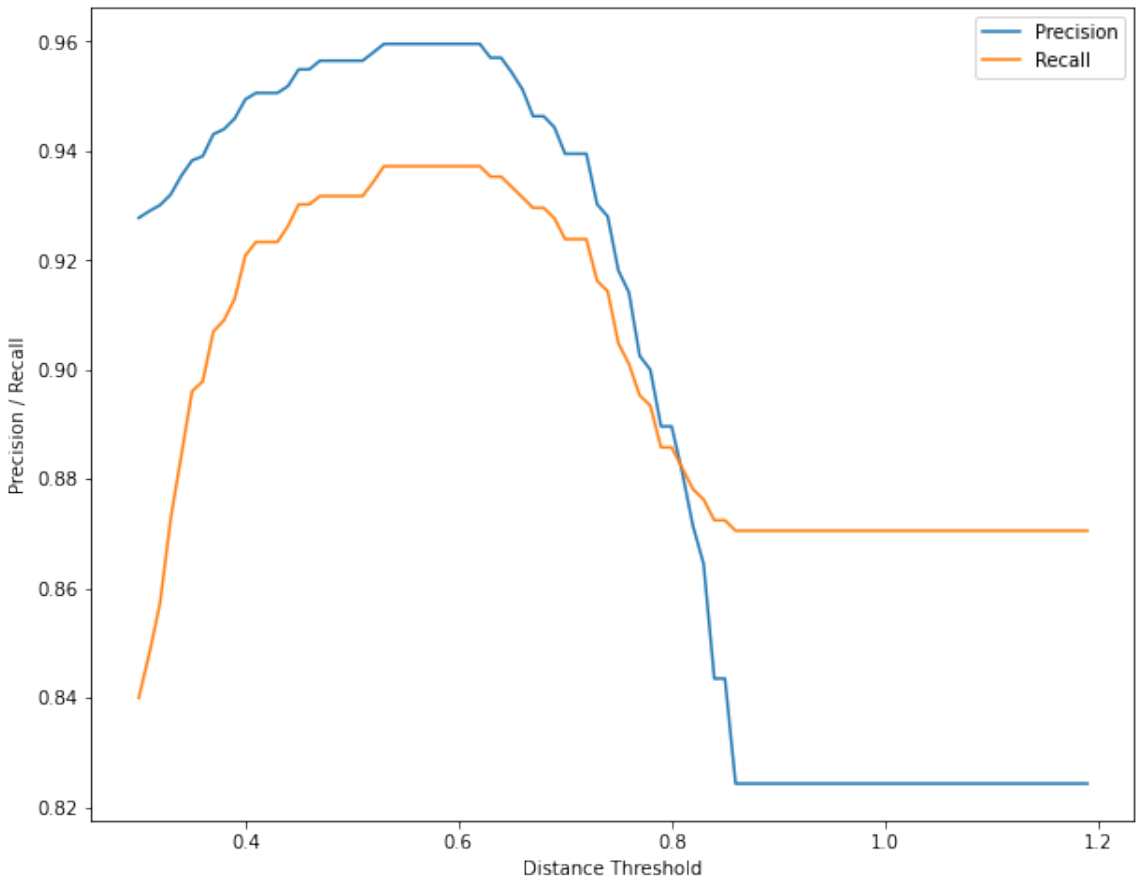


Figure 7.2: model Accuracy

The **F1 score** can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$F1score = 2 * \frac{precision * recall}{precision + recall} \tag{7.4}$$

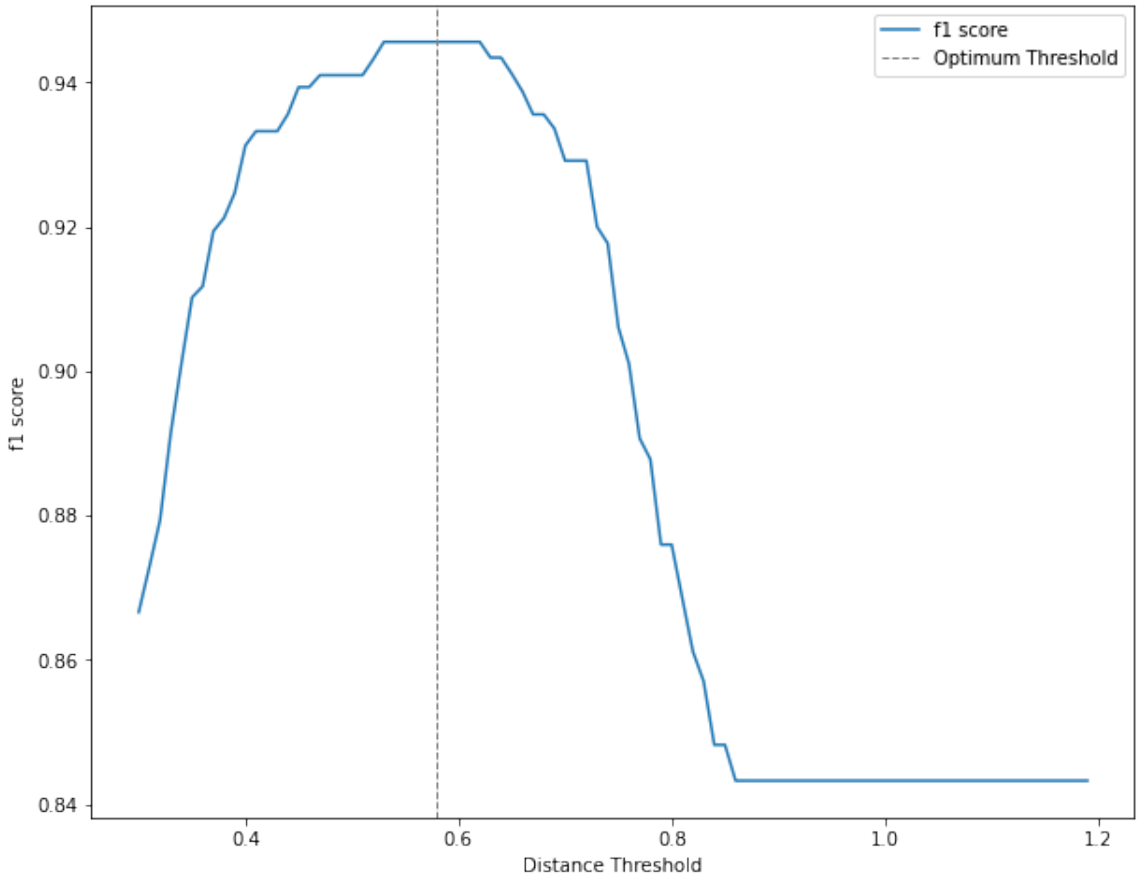


Figure 7.3: F1 score

We can see that, under different measures, the model is better to perform when the threshold is around its optimum value (0.58).

7.2 Discussion

In this thesis, we represent an end-to-end face recognition system. the system is able to collect face images from the camera feed, detect faces, extract an embedded vector, and based on the extracted embedded vector the system can identify the person in front of the camera.

The system is based on ArcFace model, which utilizes the Additive Angular Margin Loss function that improved the face recognition model's discriminative power. When ArcFace was tested on the LFW and YTF datasets it achieved an accuracy of 99.83% and 98.02% respectively. However, when tested on the CALFW and CPLFW datasets, there was a drop in the accuracy because these datasets have higher pose and age variations.

The Local dataset does not have an age variation since all the images were collected in the same time frame, but it contains a considerable pose variation. the calculated accuracy was 96.55% which is less than the LFW and YTF datasets accuracy and higher than the CALFW and CPLFW datasets accuracy.

This system is a lightweight application that runs locally on the ROS-cube board without any need for an internet connection. The application and can be beneficial for many service robotics applications.

As future work, the system can be expanded to not only predict the identity of the person but also to detect the age of the person and the facial expressions.

Bibliography

[1] Phillips, P.J.; Flynn, P.J.; Scruggs, T.; Bowyer, K.W.; Chang, J.; Homan, K.; Marques, J.; Min, J.;Worek,W. Overview of the face recognition grand challenge. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–26 June 2005; pp. 947–954. 5.2.1

[2] Huang et al. created the Labeled Faces in the Wild dataset (LFW) in 2007 to study the problem of unconstrained facial recognition, that is the variation in posture, expression, skin color, lighting, facial clothing, hair color and style, image quality, and other parameters. The dataset contains 13,233 images of 5749 different identities. each image is (250x250). 5.2.2

[3] CASIA Web Face. Available online: http://www.cbsr.ia.ac.cn/english/CASIA-WebFace-Database.html (accessed on 21 July 2019). 5.2.3

[4] Shlizerman, I.K.; Seitz, S.M.; Miller, D.; Brossard, E. The MegaFace benchmark: 1 million faces for recognition at scale. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 4873–4882. 5.2.4

[5] Guo, Y.; Zhang, L.; Hu, Y.; He, X.; Gao, J. Ms-Celeb-1m: A dataset and benchmark for large-scale face recognition. In Proceedings of the 14th European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 8–16 October 2016. 5.2.5

[6] Parkhi, O.M.; Vedaldi, A.; Zisserman, A. Deep Face Recognition. In Proceedings of the 2015 British Machine Vision Conference, Swansea, UK, 7–10 September 2015; pp. 41.1–41.12. 5.2.6

[7] Cao, Q.; Shen, L.; Xie, W.; Parkhi, O.M.; Zisserman, A. VGGFace2: A dataset for recognizing faces across pose and age. In Proceedings of the 2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG), Xi'an, China, 15–19 May 2018; pp. 67–74. 5.2.6

[8] Elharrouss, O.; Almaadeed, N.; Al-Maadeed, S. LFR face dataset: Left-Front-Right dataset for pose-invariant face recognition in the wild. In Proceedings of the 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2–5 February 2020; pp. 124–130. 5.2.7

[9] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In Proc. of CVPR, 2001. 4.2.1

[10] K. Zhang, Z. Zhang, Z. Li and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," in IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499-1503, Oct. 2016, doi: 10.1109/LSP.2016.2603342. 4.2.2

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016. 5.1

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012. 5.1

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint:1409.1556, 2014. 5.1

[14] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In CVPR, 2015. 5.1, 5.3.1

[15] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In NIPS, 2014. 5.1

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015. 5.1

[17] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In ECCV, 2016. 5.1

[18] Taigman, Yaniv and Yang, Ming and Ranzato, Marc'Aurelio and Wolf, Lior. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014. 5.3.2

[19] Wen, Yandong and Zhang, Kaipeng and Li, Zhifeng and Qiao, Yu. A Discriminative Feature Learning Approach for Deep Face Recognition. In Computer Vision – ECCV 2016 5.3.2

[20] Liu, Weiyang and Wen, Yandong and Yu, Zhiding and Li, Ming and Raj, Bhiksha and Song, Le. SphereFace: Deep Hypersphere Embedding for Face Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018 5.3.3

[21] Wang, Hao and Wang, Yitong and Zhou, Zheng and Ji, Xing and Gong, Dihong and Zhou, Jingchao and Li, Zhifeng and Liu, Wei. CosFace: Large Margin Cosine Loss for Deep Face Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018 5.3.4

[22] Deng, Jiankang and Guo, Jia and Xue, Niannan and Zafeiriou, Stefanos. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019 5.3.5, 5.3.5

[23] ROScube-x Hardware datasheet available at: https://www.module-store.de/media/pdf/c1/71/a2/ROScube-X$_{s}eries_{v}2.pdf$ 6.1.1

[24] Intel RealSense Product Family D400 Series Datasheet available at: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf 6.1.2

[25] Italy, Garante per la protezione dei dati personali, Installazione di apparati promozionali del tipo "digital signage" (definiti anche Totem) presso una stazione ferroviaria, 21 December 2017. 2

[26] EDRi, "Danish DPA approves Automated Facial Recognition", 19 June 2019. 2

[27] The Telegraph, "AI used for first time in job interviews in UK to find best applicants", 27 September 2019. 2

[28] Wired, "Facebook can now find your face, even when it's not tagged", 19 December 2017. 2

[29] Barak, A. (2019), 'Human dignity as a framework right (mother-right)', in Barak, A., Human Dignity: The Constitutional Value and the Constitutional Right, Cambridge, Cambridge University Press, 2015, Chapter 9 (pp. 156-169). 2

[30] CJEU, C-377/98, Netherlands v. European Parliament and Council, 9 October 2001, paras. 70-77. 2