Tunisian republic

Ministry of Higher Education

and Scientific Research

Carthage University

Higher School of Communication of Tunis-SUP'COM

Second year engineering Internship report

Presented to validate the internship

National Engineering Diploma

Speciality : Telecommunications

By

Hadil BEN AMOR

Development Of a New Dynamic Approach For Facial Recognition and Emotions Detection

Professional Supervisor: **Mrs Afef BOHLI** Co-founder at Digi2S

Academic Supervisor: **Mr Ridha BOUALLEGUE** Professor at SUP'COM

Internship realized at INNOV'COM

Academic Year 2021 - 2022

Tunisian republic

Ministry of Higher Education

and Scientific Research

Carthage University

Higher School of Communication of Tunis-SUP'COM

Second year engineering Internship report

Presented to validate the internship

Speciality : Telecommunications

By

Hadil BEN AMOR

Development Of a New Dynamic Approach For Facial Recognition and Emotions Detection

Professional Supervisor: **Mrs Afef BOHLI** Co-founder at Digi2S

Academic Supervisor: **Mr Ridha BOUALLEGUE** Professor at SUP'COM

Realized at INNOV'COM

Academic Year 2021 - 2022

I authorize the student to submit her internship report.

Professional Supervisor, **Mrs Afef BOHLI**

I authorize the student to submit her internship report.

Academic Supervisor, **Mr Ridha BOUALLEGUE**

Dedications

To my parents for their sacrifices,

their encouragement throughout my studies. their patience and for the good values they teached me.

Only one year left, and I will become that little engineer, who makes you proud, like I always do.

Thank you for everything you have done for me.

To my best friend Hamdene who believed in me,

supported me, encouraged me through many times

of stress, excitement, frustration and celebration.

I thank him for his support and I wish him success in his graduation project.

To my colleagues at INNOV'COM,

it was an honor meeting you and share together lots of unforgettable moments.

Hadil BEN AMOR

Gratitude

First of all, I would like to express my deepest gratitude to everyone who supported me and gave me the opportunity to complete this project, I am eternally grateful to every one of you.

My sincere thanks go to Mr Ridha BOUALEGUE, who provided me this chance to join his team as an intern.

I would like to express my greatest gratitude to my professional supervisor, Mrs. Afef BOHLI, for her continuous support, her patience, her motivation and her immense knowledge. Her advices helped me throughout the realization of this project and learning the basics of a professional research.

Thank you for believing in my potential and always pushing me beyond expectations.

Contents

List of Figures

List of Tables

Acronyms

— **AI** = Artificial Intelligence

— **CNN** = Convolutional Neural Network

— **DL** = Deep Learning

— **ML** = Machine Learning

— **NN** = Neural networks

— **lr** = Learning rate

Introduction

Artificial Intelligence refers to systems or machines that mimic human intelligence to perform tasks and can iteratively improve themselves based on the information they collect. AI is gradually penetrating all areas and it is now an integral part of the majority of companies, to offer the greatest of products and services in an intelligent way, and high quality level services by decreasing the percentage of error.

Nowadays, facial recognition and emotions detection are commonly needed in different contexts such as security, survey, at smart homes, for buildings security, systems authentication, authorization and much more. For that there are already many solutions developed by Microsoft, Google and many companies. However, for clients, it's a black box which has an input and an output without the ability to modify or interpret the accuracy. Hence, our project is a detailed code for face recognition and emotions detection by keeping the track of hyper-parameters impact on the resulted metrics. The main contribution is to understand deeply how those mechanisms work and analyse on what our AI model depends to predict more accurate result.

This report aims to give a clear idea of the work done during this internship. It is structured around four chapters:

• The first chapter introduces the problem, the objective and the development environment more clearly.

• The second chapter presents the "Facial detection" part.

• The third chapter deals with"Emotions detection" part.

• The last chapter treats "Facial recognition".

We will end with a conclusion, which will summarize our work and set out future prospects.

Chapter 1

Context

Plan

Introduction

In this chapter we will set the topic in its general context. So we will start by introducing the research laboratory. Then, we are going to present the problem deeply and our objective. We will finish by preparing the environment.

1.1 Research Laboratory

INNOV'COM is a research laboratory which offers a space for research activities putting innovation first as an objective to provide the necessary foundations for the study, design and production of mobile, cooperating and communicating systems for future information exchange and remote control systems. It targets academic research, development and innovation activities in the field of cooperative networks and radiocommunications. This expertise is strong in the field of connected objects and engineering studies related to telecommunications networks.

1.2 Problem

No one can deny the importance of recognizing people's faces and emotions. Those may be used when we want to:

- restrict access to a resource to one person, called face authentication.
- confirm that the person matches their ID, called face verification.
- assign a name to a face, called face identification.

Generally, we refer to this as the problem of automatic "face recognition". It is often described as a process that first involves four steps:

1. Face Detection: locate one or more faces in the image and mark with a bounding box.

2. Face Alignment: normalize the face to be consistent with the database, such as geometry and photometrics.

3. Feature Extraction: extract features from the face that can be used for the recognition task (name, emotions..)

4. Face Recognition(resp. Emotions detection): perform predicting the name(resp. emotion) of the person.

1.3 Existant Technologies

There are many solutions, APIs, defines how two applications communicate with each other using requests and responses, that solve this kind of problems.

• MediaPipe Face Detection [1]: developed by Google. It is an ultra-fast solution that includes 6 landmarks and multi-face support. It is based on Blaze Face, a lightweight and high-performance face detector designed for mobile GPU inference.

• Microsoft Face API [2]: service developed by Microsoft. It provides AI algorithms that detect, recognize, and analyze human faces in an image and return the rectangular coordinates of their locations. It can recognize age, gender and emotions. Face comparison and verification included.

• Face Reader [3]: developed by Noldus. It is used in the academic sphere, and based on machine learning, tapping into a database of 10,000 facial expression images. The API uses 500 key facial points to analyze 6 basic facial expressions as well as neutral and contempt. Face Reader also detects gaze direction and head orientation.

APIs make it possible to communicate with other products and services without knowing the details of their implementation. They simplify application development, saving you time and money. However in this project, we built the models step by step, follow the evolution of the result, adjust the parameters and analyze the effect of each, and why not, be the core of a new API.

1.4 Objective

Being able to develop an AI model that helps you to solve problems is an essential thing to learn for an AI enthusiast. In this project we will use prebuilt classifier, build new artificial intelligence models step by step, improve the accuracy while varying the hyper-parameters and explore ready architecture by adapting them to our problem and discover how they perform.

Our solution is divided into three parts: facial detection, emotions detection and facial recognition. It will be a software as a Service (SaaS)[4] that can be implemented and deployed depending on the client's need.

Figure (1.1) represents the process related to our service.

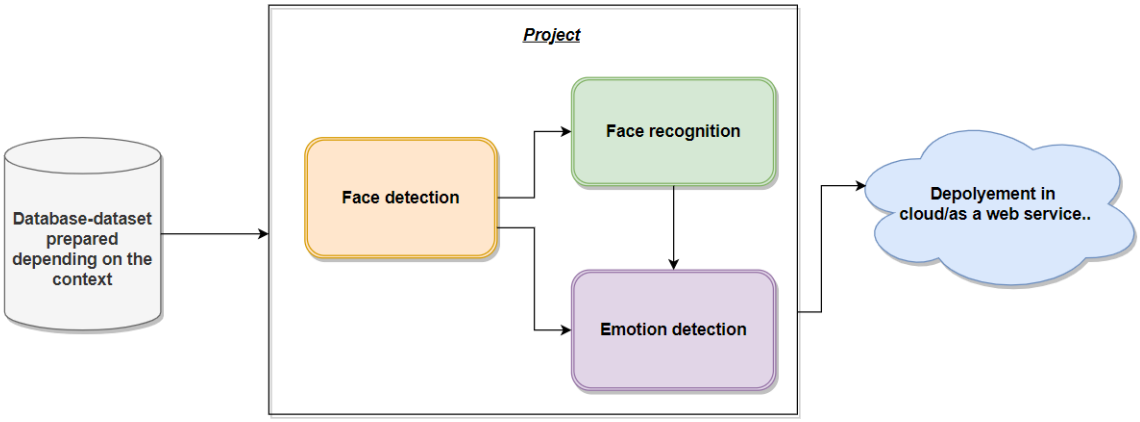


Figure 1.1: Overview

1.5 Environment

1.5.1 Algorithms

• Deep Learning: neural networks trained using the stochastic gradient descent algorithm. The most popular DL algorithms are: Convolutional Neural Networks (CNN), Long Short Term Memory Networks (LSTM), Recurrent Neural Networks (RNN), Generative Adversarial Networks (GAN), Function Networks Radial Base (RBFN) and auto-encoders. In this project, we will focus on CNNs.

• Convolutional Neural Networks: is a type of artificial neural network used in image recognition and processing and specially designed for pixel analysis. The CNN compares images fragment by fragment. The fragments it looks for are called features. By finding approximate features that look roughly alike in two different images, CNN is much better at detecting similarities than by an entire image-to-image comparison.

Figure (1.2) represents a basic CNN architecture.

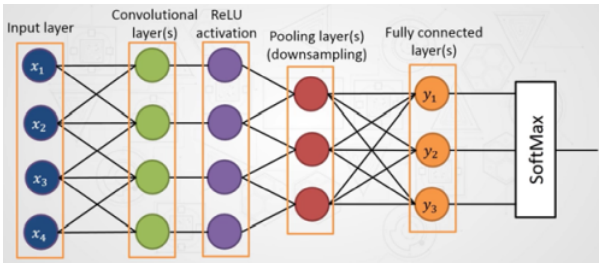


Figure 1.2: CNN Architecture

1.5.2 IDE

• Google Colaboratory: or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. Google have released Colaboratory web IDE for python, to enable Machine Learning with storage on the cloud — this internal tool had a pretty quiet public release in late 2017, and is set to make a huge difference in the world of machine learning, artificial intelligence and data science work.

Figure 1.3: Google Colab Logo

• Python: is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together.

Figure 1.4: Python Logo

1.5.3 Libraries and Framework

Those python libraries and framework will be used in the three parts of the project.

• Numpy: is the fundamental package for scientific computing in Python. It is a python library that provides a multidimensional array object, various derived objects and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation,basic linear algebra, basic statistical operations, random simulation and much more

• TensorFlow: is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications. We will use its 2.8.2 version.

• OS: provides functions for interacting with the operating system. It comes under python's standard utility modules.

• PIL.Image: is an imaging library which provides the python interpreter with image editing capabilities.

• Pandas: is an open source python package that is most widely used for data science, data analysis and machine learning tasks.

• Matplotlib: is a comprehensive library for creating static, animated, and interactive visualizations in python. Matplotlib makes hard things easy and possible. It also creates publication quality plots.

• Joblib: set of tools to provide lightweight pipelining in python. In particular: transparent disk-caching of functions and lazy re-evaluation easy simple parallel computing.

• Pickle: used in serializing and deserializing[5] a python object structure. It's the process of converting a python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network

1.5.4 Installation

The datasets used in this project are imported from Kaggle. So, to avoid downloading the datasets into the local disk and upload them again into Colab, we will use Kaggle API command.

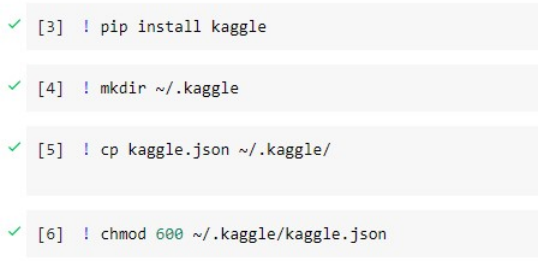


Figure 1.5: Installation

1.5.5 Hyper-parameters

The hyperparameters to be tuned are the number of neurons, epochs, batch size, optimizer, learning rate and activation function.

• Epochs: an hyper-parameter that defines the number of times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

• Batch size: an hyper-parameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.

• Optimizer: an algorithm or a method used to change the attributes of the neural network

such as weights and learning rate to reduce the loss.

• Learning rate: controls how much to change the model in response to the estimated error each time the model weights are updated.

• Activation function: defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.

1.5.6 Accuracy and Loss

• Accuracy: is the number of correctly predicted data points out of all the data points. An accuracy metric is used to measure the algorithm's performance in an interpretable way. The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage. This accuracy is represented by the matrix called Confusion Matrix.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Figure 1.6: Accuracy function formula

• Loss: a value implies how poorly or well a model behaves after each iteration of optimization. It is calculated by the loss function. In our project, we used "Sparse Categorical cross-entropy" loss function since our classes are mutually exclusive, that means when each sample belongs exactly to one class.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

Figure 1.7: Loss function formula

Conclusion

This chapter allowed us to represent the existent technologies related to facial recognition and emotions detection, to introduce our project's objective, and the work environment. The "Facial detection" part will be detailed in the next chapter.

Chapter 2

Face Detection

Plan

Introduction

In this chapter, we will present what face detection is, and our realization of the AI algorithms. This part constitutes a major input for the other two parts of the project.

2.1 Definition

Face detection is an artificial intelligence (AI) based computer technology used to find and identify human faces in digital images. Face detection applications try to find human faces within images, which often incorporate other non-face objects such as landscapes, buildings and other human body parts like feet or hands. Face detection algorithms typically start by searching for human eyes, one of the easiest features to detect. The algorithm might then attempt to detect eyebrows, the mouth, nose, nostrils and the iris. Once the algorithm concludes that it has found a facial region, it applies additional tests and features extraction to confirm that it detected a face.

2.2 Libraries

• OpenCV: Open Source Computer Vision library is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. We will use 4.6.0 cv2 version

• sys: is a module in python with various functions and variables that are used to manipulate different parts of the python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter.

• Darknet: is an open-source NN framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. Darknet is used as the framework for training meaning it sets the architecture of the network.

2.3 Face detection using Haarcascade

A HaarCascade is basically a classifier used to detect the object it was trained for (eyes, face..). It is an algorithm used to identify faces in an image or in a video. The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper "Rapid Object Detection using a Boosted Cascade of Simple Features" published in 2001[6]

We will use HaarCascade and rely on the webcam with Google Collab to detect faces from an uploaded image or captured from a real-time video thanks to a JavaScript code.

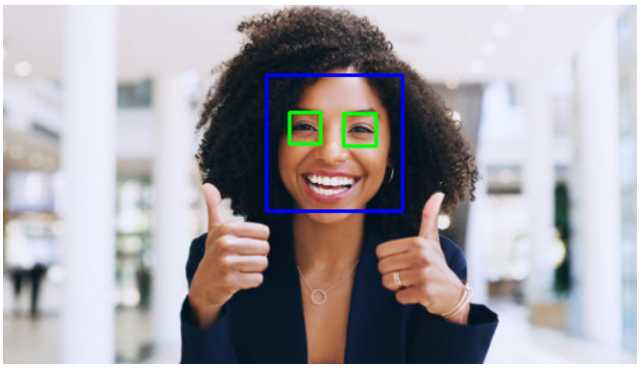


Figure 2.1: Uploaded Image

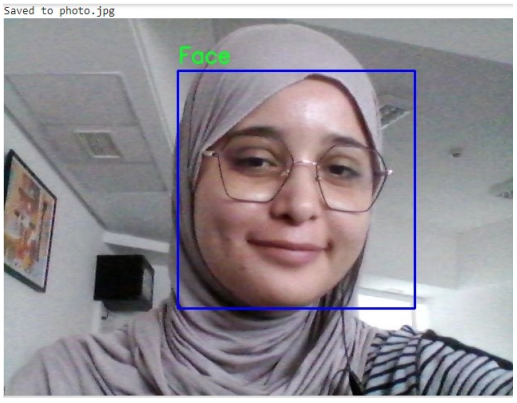


Figure 2.2: Image captured from real time video

2.4 Face detection using Yolov4

2.4.1 Definition

YOLO (You Only Look Once) is an algorithm that detects and recognizes various objects in a real-time picture. Object detection in YOLO is done as a regression problem and provides various class probabilities and bounding boxes simultaneously. YOLO algorithm employs CNN to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects. This means that prediction in the entire image is done in a single algorithm run.

YOLOv4 is a model published in April 2020 and works by breaking the object detection task into two pieces, regression to identify object positioning via bounding boxes and classification to determine the object's class. This implementation of Yolov4 uses the Darknet framework. By

using YOLOv4, we are implementing many of the past research contributions in the YOLO family (Yolo,Yolov2,Yolov3) along with a series of new contributions unique to YOLOv4 including new features: WRC, CSP, CmBN, SAT, Mish activation, Mosaic data augmentation, CmBN, DropBlock regularization, and CIoU loss. So with YOLOv4, we are using a better object detection network architecture and new data augmentation techniques.

Although we have new advanced versions of Yolo (Yolov5, Yolov6, Yolov7), we chose to use Yolov4 because it perfectly responds to our need for a single face detection. Figure 2.3 represents the architecture.

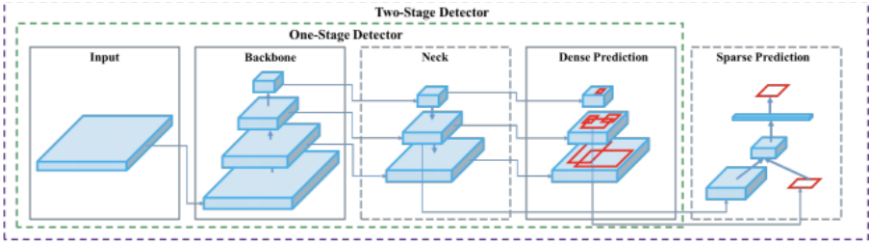


Figure 2.3: Yolov4 Architecture

2.4.2 Results

Yolov4 predicts correctly the type of objects detected in the real-time video and provides their exact positions with a bounding box. The highest accuracy given by Yolov4 in this example of person detection is 94.82%. Figures 2.4, 2.6 represent how Yolov4 works with an uploaded image and through real-time video.

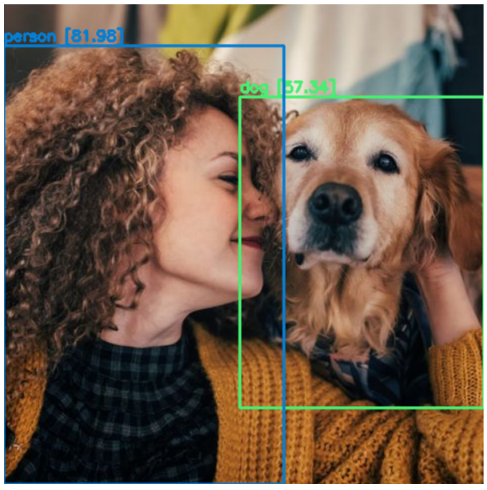


Figure 2.4: Uploaded Image (Yolov4)

Figure 2.5: Screenshots from a real time video (Yolov4)

2.4.3 Limits of Yolov4

Despite the high performance of Yolov4 and the high accuracy, it has some limitations we remarked through experimentations. The accuracy of detecting a person become lower when we turn the face with an angle. Then for a cup with a yellow color, it's detected as a Banana not as cup since they both have the same color. It's about the extracted features and depending on which one, Yolov4 took the decision. The color of the object or the direction of the face haven't to be so determinate. Figures 2.6, 2.7 illustrate the above remarks.

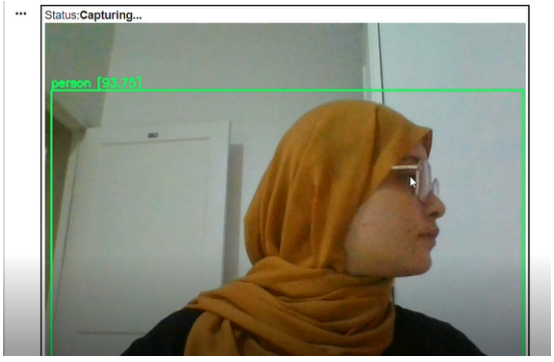


Figure 2.6: Case of face rotation

Figure 2.7: Case of color resemblance

Conclusion

Through this chapter, we have studied the detection of faces in real time through the camera. Thus, we have positioned ourselves in the context of our project, this will allow us to begin the next step which consists in presenting the sentiment analysis.

Chapter 3

Emotions Detection

Plan

Introduction

Emotions detection is a big challenge today. Emotional artificial intelligence is a subset of AI that allows a computer to predict, recognize, interpret the human's emotions.

3.1 Dataset

We will use Face expression dataset from Kaggle which:

- contains 35887 images, each of size (48,48,3)
- is divided into seven classes: 'angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise'
- is splitted into 0.8 train and 0.2 test

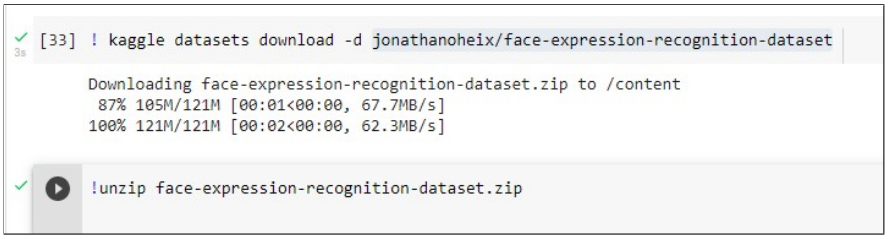


Figure 3.1: Import the dataset

3.2 Model Construction

For our model, we will use the CNN. The layers used here are:

- Convolution2D: a convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training. The size of the filters is usually smaller than the actual image. Each filter convolves with the image and creates an activation map.

- MaxPooling2D: is similar to convolution layer, but instead of doing convolution operation, we are selecting the max values in the receptive fields of the input, saving the indices and then producing a summarized output volume. The implementation of the forward pass is pretty simple.

- ZeroPadding2D: can add rows and columns of zeros at the top, bottom, left and right side of an image tensor.

- Flatten: is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the image.

• Dense: is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to every neuron of its preceding layer. This layer is the most commonly used layer in artificial neural network networks.

The figures 2.3, 2.4, represent our model

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 zero_padding2d (ZeroPadding  (None, 50, 50, 3)        0
 2D)

 conv2d_36 (Conv2D)          (None, 48, 48, 32)        896

 max_pooling2d_3 (MaxPooling  (None, 24, 24, 32)       0
 2D)

 conv2d_37 (Conv2D)          (None, 22, 22, 64)        18496

 max_pooling2d_4 (MaxPooling  (None, 11, 11, 64)       0
 2D)

 conv2d_38 (Conv2D)          (None, 9, 9, 128)         73856

 flatten_10 (Flatten)        (None, 10368)             0

 dense_19 (Dense)            (None, 64)                663616

 dense_20 (Dense)            (None, 7)                 455

=================================================================
Total params: 757,319
Trainable params: 757,319
Non-trainable params: 0
```

Figure 3.2: Model's architecture

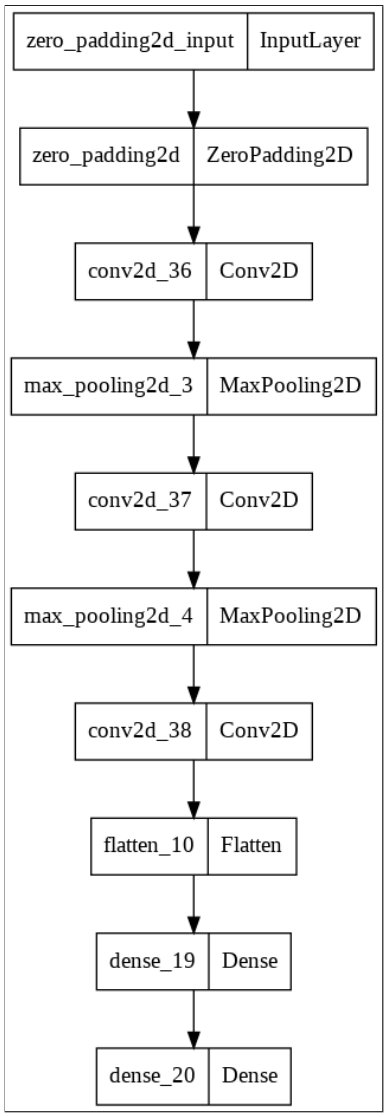


Figure 3.3: Layers'Input/Output

Avoid Overfitting

In order to prevent overfitting [7], we included EarlyStopping.

• EarlyStopping: is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration. Up to a point, this improves the learner's performance on data outside of the training set.

3.3 Observations

For this part we will study the hyperparameters by modifying one of them each time. A follow-up of the loss and accuracy values will be conducted.

3.3.1 Epochs and Batch Size tunning

We fixed the Optimizer= "Adam(lr=0.001)", Loss= "Sparse Categorical", Early-stopping=True and we will modify the number of epochs.

Epochs	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
5	46.51	1.3725	44.83	1.4255
7	70.45	0.7863	47.68	1.6892
15	89.87	0.2896	47.43	3.9174
30	94.75	0.1754	47.38	6.2116

Table 3.1: Epochs as a variable

Analysis : We remarked that we got the highest accuracy thanks to the higher number of epochs we used. The more the number of epochs increases, the more number of times the weights are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve.

3.3.2 Optimization and learning rate tunning

We fixed the number of epochs=5, Loss= "Sparse Categorical", Early-stopping=True and we will modify the optimizer and its learning rate.

Adam	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
lr=0.001	55.97	1.1609	48.15	1.4714
lr=0.01	24.86	1.8123	25.83	1.8098
lr=0.1	24.26	1.8212	25.83	1.8278
lr=0.5	21.52	1.8584	25.83	1.8326

Table 3.2: Adam optimizer

SGD	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
lr=0.01, m=0(default values)	21.66	1.8569	25.83	1.8406
lr= 0.01, m = 1	21.32	1.8551	25.8	1.9
lr=0.1, m=0	22.23	1.8543	17.21	1.8275
lr=0.1, m=1	21.43	1.8555	25.83	1.8782

Table 3.3: SGD optimizer

Analysis : We got the best result when using the Adam optimizer. Observing the learning rate change, the best lr is 0.001. We remarked that if we increase the learning rate of Adam optimizer, the accuracy decreases and the loss increases. The learning rate must be discovered via trial and error.

3.4 Final results

After training the model, changing the hyperparameters, and analyzing the results, we attend a final result with a 94% accuracy on the training data and 51% on the test data. Those results could be improved in the future.

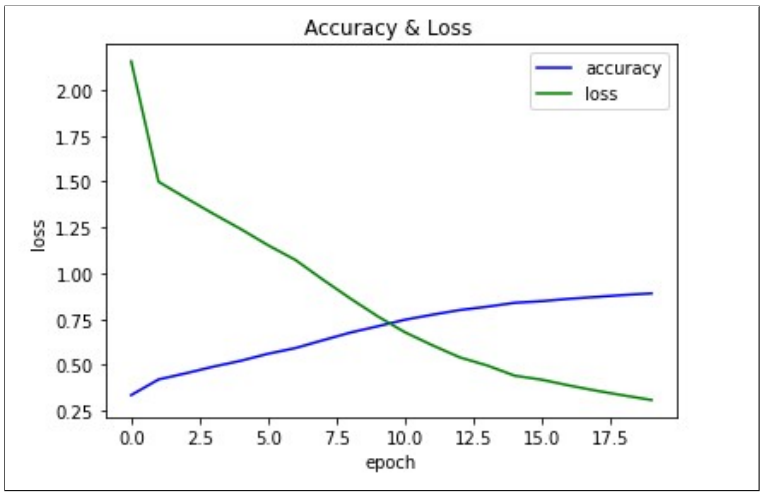


Figure 3.4: Accuracy and Loss

The model gives the results as an array of probabilities for each image. The shape of array is (7066,7) with 7066 is the number of images in the test data and 7 is the number of our predefined classes. To resolve this issue:

• use numpy.argmax() function that returns the index of the max element of the array in a particular axis.

• define a python function "index-to-class" that switches the integer [0..6] to its respective classe.

predicted_class	Class
5	sad
0	anger
0	anger
5	sad
6	surprise
...	...
3	happy
2	fear
6	surprise

Figure 3.5: Predictions

The only thing left is to save our model Emotion.joblib, as a pretrained model, for further uses.

3.4.1 Prediction using a real time image

The image detected from the camera has a shape of (480,640) different from the image size (48,48) with which our model is trained on. Figure 3.6 shows the low resolution of the image when predicting with a captured image's real shape.

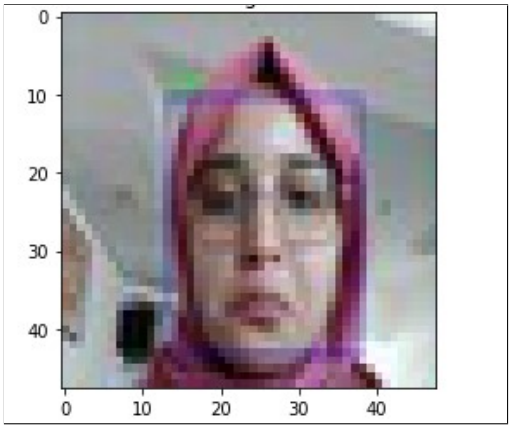


Figure 3.6: Prediction with image's real shape

To solve this issue, we tried three different methodes, could be found in the colab notebook, but in the report we are going to detail the method which gave us the best resolution:

• Open the captured image from face detection algorithm

- Resize the image to (48,48)
- Process the image to an array using tf.keras.preprocessing.image.img-to-array
- Convert single image to a batch
- Divide all the values by 255 will convert it to range from 0 to 1.
- Load the saved model, predict the emotion and plot the figure

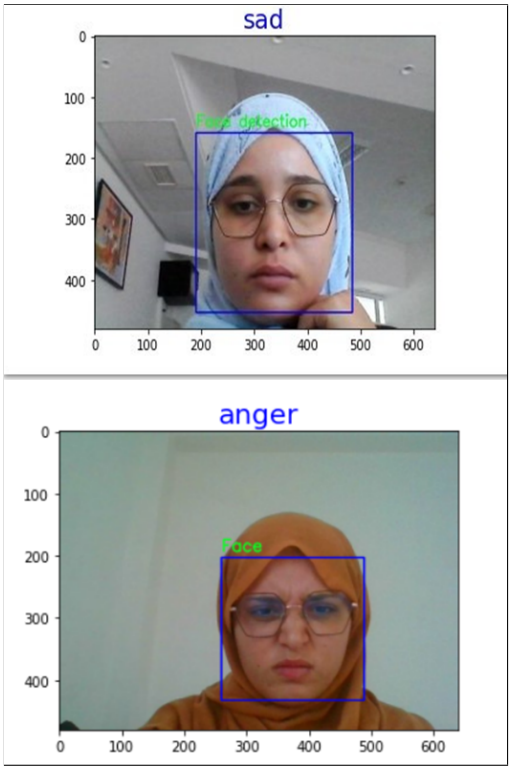


Figure 3.7: Emotions prediction after image processing

Conclusion

This chapter explores our newly built AI model for emotion detection and analyse the effect of changing hyperparameters' values.

Chapter 4

Facial recognition

Plan

Introduction

A facial recognition system is an application aimed, as the name indicates, to automatically recognize a person. In this chapter, we are going to follow two path: construct a model and use a predefined architecture with modifications.

4.1 Dataset

We will use LFW dataset from Kaggle.

• Labeled Faces in the Wild (LFW) is a database of face photographs designed for studying the problem of unconstrained face recognition. This database contains 13,233 images of 5,749 people detected and centered by the Viola Jones face detector[8] and collected from the web. 1,680 of the people pictured have two or more distinct photos in the dataset.

We injected some photos of my family, my friends and myself into the database, imported it into drive, and used it directly in collab. The purpose of this injection is to take a real-time photo of a close person and predict their identity to test our model's performance.

We splitted the data into (train, validation, test)= (0.7, 0.1, 0.2) after installing split-folders.

Figure 4.1: Install split-folder

4.2 Model Construction

For this model, we will use CNN. The layers used here are: Convolution2D, MaxPooling2D, Dense and Flatten. We started by a basic CNN model and continue adding layers and train the model. The architecture is built progressively.

```
model1.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 32)        896

 max_pooling2d (MaxPooling2D  (None, 24, 24, 32)       0
 )

 dropout (Dropout)           (None, 24, 24, 32)        0

 conv2d_1 (Conv2D)           (None, 22, 22, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 11, 11, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 11, 11, 64)        0

 conv2d_2 (Conv2D)           (None, 11, 11, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_2 (Dropout)         (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 5749)              9204149

=================================================================
Total params: 9,260,469
Trainable params: 9,260,469
Non-trainable params: 0
_________________________________________________________________
```

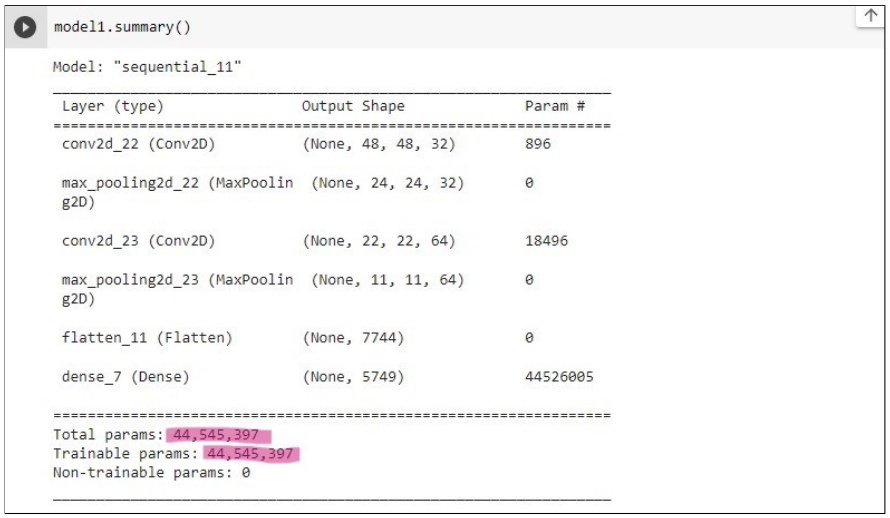
Figure 4.2: Model Summary

Avoid overfitting

In order to prevent overfitting, we included Dropout layers in the very early architecture of our model.

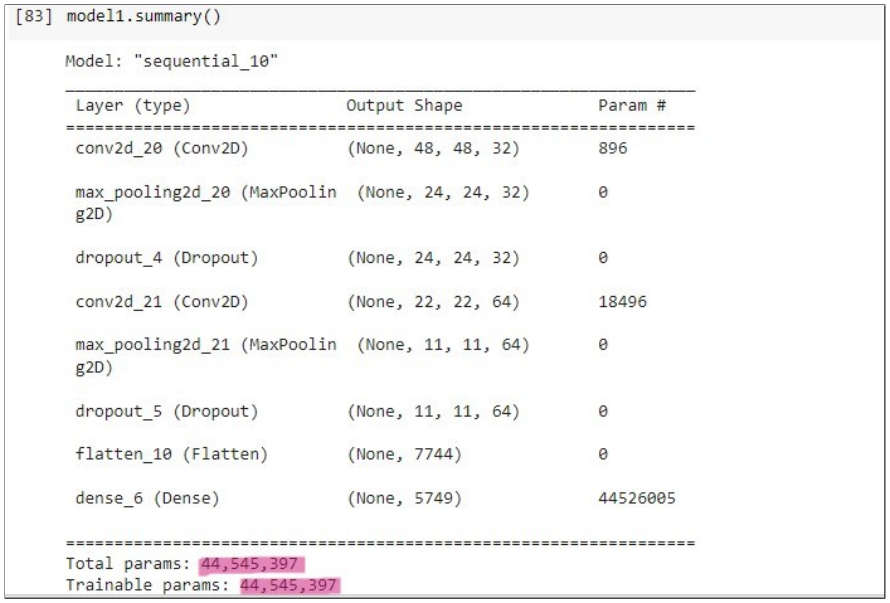
• Dropout: randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

Adding Droput layers has never changed the number of trainable parameters as Dropout does not have any variables/weights that can be frozen during training like it is demonstrated in the figures 3.3, 3.4.



```
model1.summary()

Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_22 (Conv2D)          (None, 48, 48, 32)        896

 max_pooling2d_22 (MaxPoolin  (None, 24, 24, 32)       0
 g2D)

 conv2d_23 (Conv2D)          (None, 22, 22, 64)        18496

 max_pooling2d_23 (MaxPoolin  (None, 11, 11, 64)       0
 g2D)

 flatten_11 (Flatten)        (None, 7744)              0

 dense_7 (Dense)             (None, 5749)              44526005

=================================================================
Total params: 44,545,397
Trainable params: 44,545,397
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.3: Model without dropout layers



```
[83] model1.summary()

Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 48, 48, 32)        896

 max_pooling2d_20 (MaxPoolin  (None, 24, 24, 32)       0
 g2D)

 dropout_4 (Dropout)         (None, 24, 24, 32)        0

 conv2d_21 (Conv2D)          (None, 22, 22, 64)        18496

 max_pooling2d_21 (MaxPoolin  (None, 11, 11, 64)       0
 g2D)

 dropout_5 (Dropout)         (None, 11, 11, 64)        0

 flatten_10 (Flatten)        (None, 7744)              0

 dense_6 (Dense)             (None, 5749)              44526005

=================================================================
Total params: 44,545,397
Trainable params: 44,545,397
```

Figure 4.4: Model with dropout layers

4.2.1 Observations

For this part we will study the hyperparameters by modifying one of them each time. A follow-up of the Loss and Accuracy Matrix will be conducted.

4.2.1.1 Epochs and Batch-Size tunning

We fixed the Optimizer= "Adam(lr=0.01)", Loss= "Sparse Categorical", Early-stopping=True and we will modify the number of epochs with different batch-size values.

Epochs(batch-size=64)	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
10	21.37	4.62	8.49	6.62
15	34.7	3.24	10.08	7.8
25	50.9	2.36	10.88	9.8
35	55.88	1.94	11.67	10.54

Table 4.1: Epochs variation, batch-size=64

Epochs(batch-size=32)	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
10	11.28	4.94	11.94	5.6
15	22.42	4.01	9.55	6.3
25	32.37	3.3	10.34	6.5

Table 4.2: Epochs variation, batch-size=32

Epochs(batch-size=96)	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
10	6.62	6.51	14.06	5.3
15	6.86	6.44	13.53	5.53
25	6.95	6.44	13.53	5.88

Table 4.3: Epochs variation, batch-size=96

Analysis: batch-size=64 gave us the best result concerning the training accuracy and it generalizes well on the validation set. Then followed by batch-size=32 and batch-size=96. Larger batch sizes may (often) converge faster and give better performance. Also, a larger batch size may improve the effectiveness of the optimization steps resulting in more rapid convergence of the model parameters.

4.2.1.2 Optimization and learning rate tunning

We fixed the number of epochs=5, Loss= "Sparse Categorical", Early-stopping=True, batch-size=64 and we will modify the optimizer and its learning rate.

Adam	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
lr=0.001	63.3	1.55	9.8	11.17
lr=0.05	6.48	6.57	14.06	5.08
lr=0.1	6.48	6.66	14.06	5.1
lr=0.5	5.97	7.52	6.1	5.4

Table 4.4: Adam optimizer

SGD	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
lr=0.01, m=0(default value)	6.48	7.13	14	5.42
lr= 0.01, m = 1	2.13	7.5	6.10	6.4
lr=0.1, m=0	6.48	7.73	14.06	6.21
lr=0.1, m=1	5.66	6.88	14.06	5.06

Table 4.5: SGD optimizer

Analysis: for a fixed number of epochs and batch-size, Adam optimizer with its default learning rate value (0.001) gave us the best result for the training accuracy. Adam implicitly performs coordinate-wise gradient clipping[9] and can, unlike SGD, tackle heavy-tailed noise [10]. We proved that using such coordinate-wise clipping thresholds can be significantly faster than using a single global one. This can explain the superior performance of Adam.

4.2.2 Final results and plot

After changing the hyper-parameters, analyzing results, training the model and retraining it, we attended an accuracy of 76.34% on training set and 14.32% on validation set after epochs=120.

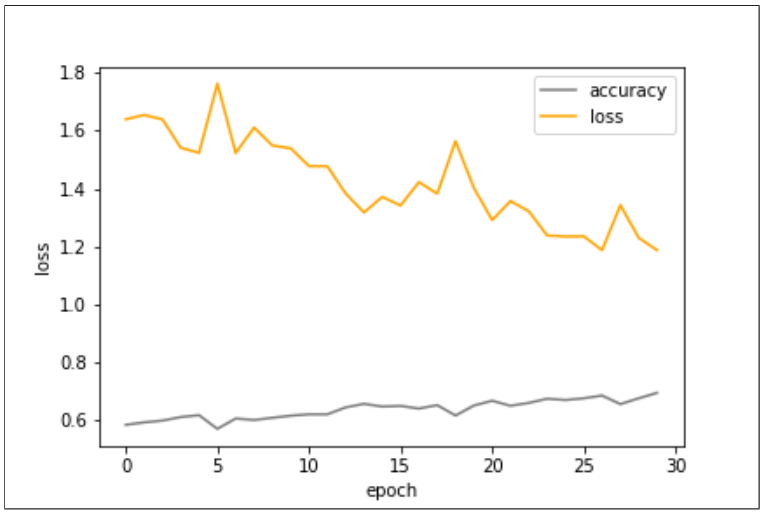


Figure 4.5: My Face recognition architecture (Accuracy and Loss)

The model gives the results as an array of probabilities for each image. The shape of the array is (7129, 5758) with 7129 is number of images in the test data and 5758 is the number of persons (initial dataset+injected). To resolve this issue:

• use numpy. argmax() function that returns indice of the max element of the array in a particular axis.

	predicted_person_ID
0	4012
1	148
2	1892
3	2506
4	1047
...	...
7124	2180
7125	1871
7126	1047
7127	417
7128	1871

7129 rows × 1 columns

Figure 4.6: Predictions of IDs

• use the people.csv file from the same dataset, to write a python function index-to-names.

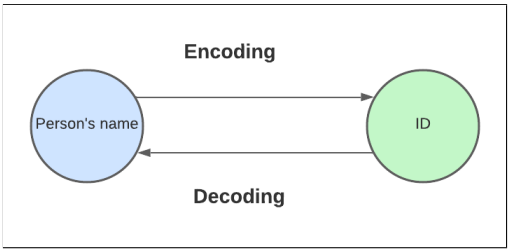


Figure 4.7: IDs Vs Names

predicted_person_ID	Names
4012	Paradorn_Srichaphan
148	Kelsey_Grammer
1892	Richard_Cohen
2506	Leslie_Wiser_Jr
1047	Raul_Gonzalez
...	...
2180	Eli_Broad
1871	Horst_Koehler
1047	Raul_Gonzalez
417	Robert_Korzeniowski
1871	Horst_Koehler

7129 rows × 1 columns

Figure 4.8: Prediction of Names

The final output is represented by the figure 4.9

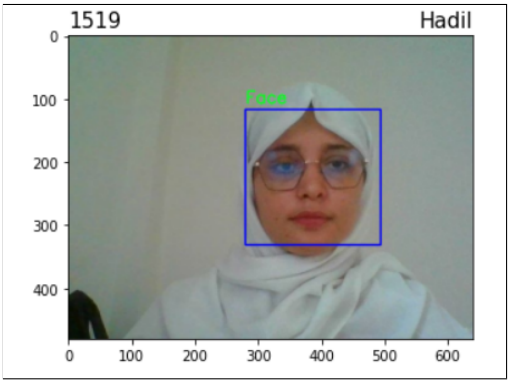


Figure 4.9: Recognition result

4.3 LeNet5 Architecture

4.3.1 Definition

Lenet5 is one of the earliest pre-trained models proposed by Yann LeCun and others in the year 1998, in the research paper Gradient-Based Learning Applied to Document Recognition [11]. LeNet is small and easy to understand yet large enough to provide interesting results. The architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers.

Since this architecture was mainly used for recognizing the handwritten and machine-printed characters, we thought of applying it to see what it gives for another type of recognition and try to adapt it to our face recognition problem.

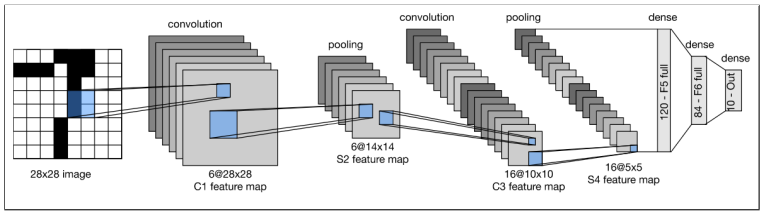


Figure 4.10: LeNet5 architecture

4.3.2 Observations

We import the Lenet5 code into our notebook as it is pretrained for (28*28) images. We retrain it on our dataset.

Initially we have: activation-function=tanh, input-shape=(32,32), kernel-size=(3,3). The following table will explicit the modifications we import to the Lenet5 architecture.

Successive Modifications	TrainAcc(%)	TrainLoss(%)	TestAcc(%)	TestLoss(%)
None (epochs=25)	5	14	1.2	14
activation-function=relu	6.48	15	6.48	15.4
kernel-size=(5,5)	50	3.51	9	77
image size=real-image shape=(250,250)	86	2.321	4	11.3

Table 4.6: Modifications

Analysis: the successive modifications lead to an augmented training accuracy. For the

pre-trained model, we got the lowest accuracy. Since previously, it wasn't trained for a faces' dataset. We changed the activation function to Relu, the best and most advanced activation function right now compared to Tanh because all the drawbacks like Vanishing Gradient problem is completely removed in this activation function. In fact, during back propagation, a neural network learns by updating its weights and biases to reduce the loss function. In a network with vanishing gradient, the weights cannot be updated, so the network cannot learn. The performance of the network will decrease as a result. By increasing the kernel size to (5,5), we effectively increase the total number of parameters. So, it is expected that the model has a higher complexity to address a given problem like face recognition and it should perform better at least for our particular training set. We train the model with small image size then increase it to help the model extract features properly and be finally adapted to images with different shapes.

4.3.3 Final result and plot

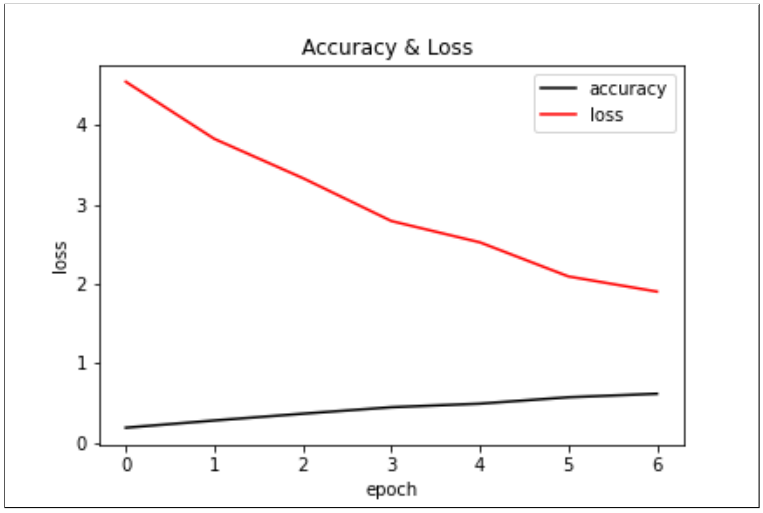


Figure 4.11: Modified LeNet5 architecture (Accuracy and Loss)

Conclusion

Thanks to this chapter, we built a new model for facial recognition and we utilize an earlier architecture, formerly trained for another purpose and adapt, test it to meet our needs.

Conclusion

This project was born after the discovery of APIs associated with facial recognition and the detection of emotions. They were introduced as black boxes, with no knowledge of what is happening inside. As an artificial intelligence engineer with a passion for research, we decide to contribute to a new API in which we can have dynamic parameters.

Within this project, we:

• Detect faces using two approaches: Haarcascade classifiers and Yolo

• Build a new CNN model that predicts the emotions of the detected person

• Recognize the identity of the detected person through two approaches: a pre-trained model with modifications and a ew CNN algorithm.

During this project we learnt how to:

• Pursue a professional research combining research, coding and practical

• Follow the change of the accuracy and Loss metrics

• Optimize the hyperparameters

• Fix and vary the variables' values

• Analyze results

We couldn't conclude an internship project without thinking about its future. For me, it will be a new start for a sucess API, that could be used as an open source. The user will be able to enter the image or start a real time video, but in the mean time he will be able to modify the parameters such as the input shape, the learning rate and observe the fluctuations of the accuracy.

Bibliography

[1] B Thaman, T Cao, and N Caporusso, « Face mask detection using mediapipe facemesh », in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, IEEE, 2022, pp. 378–382.

[2] K. Maheshwari *et al.*, « Facial recognition enabled smart door using microsoft face api », *arXiv preprint arXiv:1706.00498*, 2017.

[3] T. Skiendziel, A. G. Rösch, and O. C. Schultheiss, « Assessing the convergent validity between the automated emotion recognition software noldus facereader 7 and facial action coding system scoring », *PloS one*, vol. 14, no. 10, e0223905, 2019.

[4] W. Tsai, X. Bai, and Y. Huang, « Software-as-a-service (saas): perspectives and challenges », *Science China Information Sciences*, vol. 57, no. 5, pp. 1–15, 2014.

[5] K. Tanaka and T. Saito, « Python deserialization denial of services attacks and their mitigations », in *International Conference on Computational Science/Intelligence & Applied Informatics*, Springer, 2018, pp. 15–25.

[6] P. Viola and M. Jones, « Rapid object detection using a boosted cascade of simple features », in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, Ieee, vol. 1, 2001, pp. I–I.

[7] D. M. Hawkins, « The problem of overfitting », *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[8] Y.-Q. Wang, « An analysis of the viola-jones face detection algorithm », *Image Processing On Line*, vol. 4, pp. 128–148, 2014.

[9] F. Huang, J. Li, and H. Huang, « Super-adam: faster and universal framework of adaptive gradients », *Advances in Neural Information Processing Systems*, vol. 34, pp. 9074–9085, 2021.

[10] M. Barsbey, M. Sefidgaran, M. A. Erdogdu, G. Richard, and U. Simsekli, « Heavy tails in sgd and compressibility of overparametrized neural networks », *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 364–29 378, 2021.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, « Gradient-based learning applied to document recognition », *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.