UNIVERSITÀ DEGLI STUDI
DEL SANNIO Benevento

Dipartimento di Ingegneria

Corso di Laurea in Ingegneria Informatica

FACIAL EXPRESSION RECOGNITION USING CONVOLUTIONAL NEURAL NETWORKS

Relatori

Prof. Massimiliano Di Penta

Dott.sa Fiorella Zampetti

Candidato

Stefano De Lorenzo

Matr. 863/757

Anno Accademico 2018/2019

Index

Introduction

People communicate through speech, gesture and emotions. When one of the previously mentioned means cannot be used, for example between two people who do not speak the same language, the other two become fundamental for interacting with others. Systems that can recognize the same are in great demand in many fields. With respect to artificial intelligence, a computer that can recognize the different ways of how a human can communicate will interact naturally with people. It would also help during counselling and other health care related fields. The use of this technology, however, is not limited to some areas, in fact it can be applied to almost all the work sectors.

For real-life purposes, facial expression recognition has a number of applications. It could be used in conjunction with other systems to provide a form of safety. For instance, ATMs could be set up such that they won't dispense money when the user is scared or it could be used to recognize the features that only people that is infected with a particular virus show, to help understand if the affected person has a chance of being infected. In the gaming industry, emotion-aware games can be developed which could vary the difficulty of a level depending on the player's emotions. It also has uses in video game testing. At present, players usually give some form of verbal or written feedback. Using facial expression recognition is easier to understand the emotions that the game cause in people from different countries without knowing all their languages. By judging their expressions during different points of the game, a general understanding of the game's strong and weak points can be discerned. Emotions can also be gauged while a viewer watches ads to see how they react to them. This is especially

helpful since ads do not usually have feedback mechanisms apart from tracking whether the ad was watched and whether there was any user interaction. Another field where this technology could be applied is cinematography as during the projection of films the emotions of the spectators could be traced to define how people reacts to different kind of behaviours inside the film. It could be, also, possible to develop software that takes images only when the subject smiles.

However not all emotions can be inferred just by looking at someone Even today, researchers aim to identify emotions with reliable accuracy. Emotions can be inferred from a person's actions, speech, writing, facial expressions and body posture. In terms of facial expression recognition, one major challenge lies in the data collected. Most datasets contain labelled images which are generally posed. This generally involves photos taken in a stable environment such as a laboratory. While it is much easier to accurately predict the emotion in such scenarios, these systems tend to be unreliable in predicting emotions in uncontrolled environments. Another issue is that most datasets are from these controlled environments and it is relatively harder to obtain labelled datasets of emotions in the wild. Furthermore, most datasets have relatively lesser training data for emotions such as fear and disgust when compared to emotions such as happiness. Another factor to consider is a person's pose. It is significantly harder to determine the emotion of a person when only half of their face is visible, for this reason we need that the person face is clearly visible. In addition, lighting plays a major role in facial emotion recognition. Systems may fail to identify an emotion that it normally would identify if the lighting conditions are poor. Finally, one must remember that a user's emotional state is a combination of many factors, a smile does not always means that

a person is genuinely happy, in fact this technology can only show the facial expression that not all the time represent the emotive state of the person.

In this document we are going to define the basics of AI, in the first chapters, from the history of artificial intelligence to the neural networks to be able understand better the fundamentals before going into the heart of the development of the project and then in the last chapters we will draw our conclusions.

1. Background notions

1.1 Development of artificial intelligence[1]

The concept of intelligent beings has been around for a long time. The ancient Greeks, in fact, had myths about robots as the Chinese and Egyptian engineers built automatons. However, the beginnings of modern AI has been traced back to the time where classical philosophers attempted to describe human thinking as a symbolic system. Between the 1940s and 50s, a handful of scientists from various fields discussed the possibility of creating an artificial brain. This led to the rise of the field of AI research – which was founded as an academic discipline in 1956 – at a conference at Dartmouth College, in Hanover, New Hampshire. The word was coined by John McCarthy, who is now considered as father of Artificial Intelligence.

Despite a well-funded global effort over numerous decades, scientists found it extremely difficult to create intelligence in machines. Between the mid-1970s and 1990s, scientists had to deal with an acute shortage of funding for AI research. These years came to be known as the 'AI Winters'. However, by the late 1990, American corporations once again were interested in AI. Furthermore, the Japanese government too, came up with plans to develop a fifth-generation computer for the advancement of AI. Finally, in 1997, IBM's Deep Blue defeated became the first computer to beat a world Chess champion, Garry Kasparov.

As AI and its technology continued to march – largely due to improvements in computer hardware – corporations and governments too began to successfully use its methods in other narrow domains. The last 15 years,

[1] https://www.mygreatlearning.com/blog/what-is-artificial-intelligence/

Amazon, Google, Baidu, and many others, have managed to leverage AI technology to a huge commercial advantage. AI, today, is embedded in many of the online services we use. As a result, the technology has managed to not only play a role in every sector, but also drive a large part of the stock market too.

Building an AI system is a careful process of reverse-engineering human traits and capabilities in a machine and using its computational prowess to surpass what we are capable of.

The purpose of Artificial Intelligence is to aid human capabilities and help us make advanced decisions with far-reaching consequences. That's the answer from a technical standpoint. From a philosophical perspective, Artificial Intelligence has the potential to help humans live more meaningful lives devoid of hard labour, and help manage the complex web of interconnected individuals, companies, states and nations to function in a manner that's beneficial to all of humanity.

Currently, the purpose of Artificial Intelligence is shared by all the different tools and techniques that we've invented over the past thousand years – to simplify human effort, and to help us make better decisions. Artificial Intelligence has also been touted as our Final Invention, a creation that would invent ground-breaking tools and services that would exponentially change how we lead our lives, by hopefully removing strife, inequality and human suffering.

That's all in the far future though – we're still a long way from those kinds of outcomes. Currently, Artificial Intelligence is being used mostly by companies to improve their process efficiencies, automate resource-heavy tasks, and to make business predictions based on hard data rather than gut feelings. As all technology that has come before this, the research and

development costs need to be subsidised by corporations and government agencies before it becomes accessible to everyday laymen.

AI can be seen as a set composed of multiple subset, such as Machine learning and Deep learning.

Machine learning is a subset of artificial intelligence (AI) which defines one of the core tenets of Artificial Intelligence – the ability to learn from experience, rather than just instructions.

Machine Learning algorithms automatically learn and improve by learning from their output. They do not need explicit instructions to produce the desired output. They learn by observing their accessible data sets and compares it with examples of the final output. The examine the final output for any recognisable patterns and would try to reverse-engineer the facets to produce an output.

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. Deep Learning concepts are used to teach machines what comes naturally to us humans. Using Deep Learning, a computer model can be taught to run classification acts taking image, text, or sound as an input. Deep Learning is becoming popular as the models are capable of achieving state of the art accuracy. Large labelled data sets are used to train these models along with the neural network architectures.

Simply put, Deep Learning is using brain simulations hoping to make learning algorithms efficient and simpler to use. Let us now see what the difference between Deep Learning and Machine Learning is.To better understand the relationships among Artificial Intelligence, Machine Learning and Deep Learning, let’s visualize them as concentric circles:

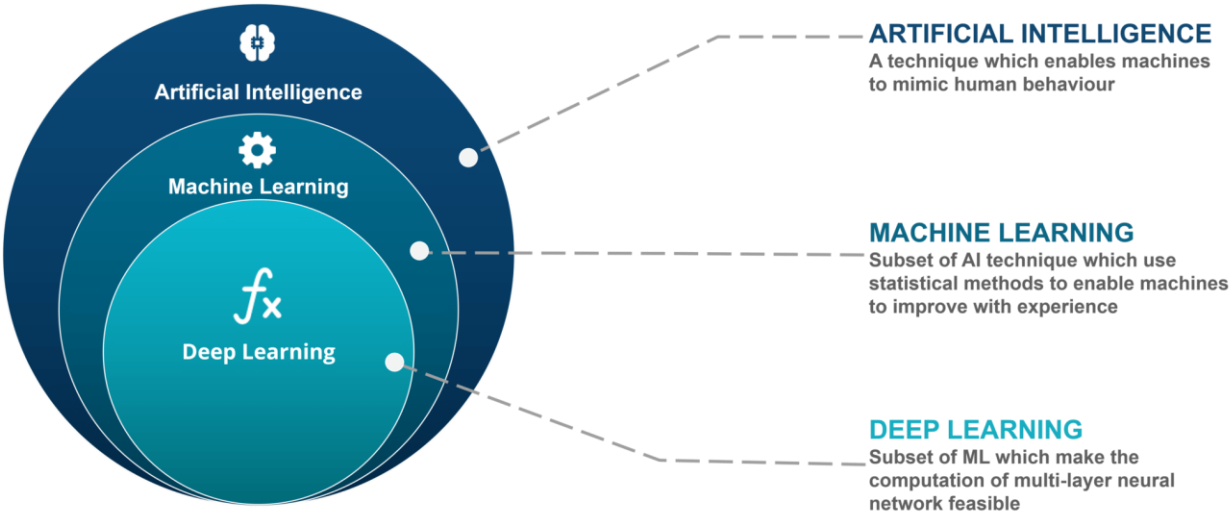

Figure 1. Scheme of relationship between AI, ML, DL (https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/)

1.2 Emotion recognition

Facial expression recognition is a process that consists in interpreting low-level signals from the subject's face. Each expression has different features that help us to understand the differences between each emotion. Every part of the human body can be interpreted to show a different kind of emotion as for example the heart rate or the position of the body. The main part of the human body that shows emotions is the face. Until few years ago the only way to be able to recognize facial expressions was the sight of man, but the development of artificial intelligence in the latest years eventually led computers to be able to recognize facial expressions.

In the early days of AI, machines were able to solve problems that were difficult for humans to solve, but had problems in solving most of the tasks that are intuitive for us, in fact AI was useful in solving well-defined, logical problems, but it was almost incapable of solving complex problems such as object detection and language translation. Newer approaches to artificial

intelligence such as Deep Learning were developed to solve these kinds of problems.

Deep learning requires complex mathematical operations to be performed on high numbers of parameters. Existing CPUs take a long time to perform these kinds of operations. A new kind of hardware called graphics processing unit (GPU) has completed these huge mathematical operations orders of magnitude faster.

In the earlier days, people needed to have expertise in C++ and CUDA to implement DL algorithms. Now, instead, thanks to the organizations open sourcing their deep learning frameworks, it is possible to develop DL algorithms with Python knowledge.

2. Neural Networks

Neural networks are multi-layer networks of neurons that we use to classify things and to make predictions. In this chapter we are going to understand how they work starting from the fundamentals and arriving to the Convolutional Neural Networks.

2.1 Fundamentals

A neural network consists of one or multiple layers of neurons (nodes), named after the biological neurons in human brains. I am going to demonstrate the mechanics of a single neuron implementing a perceptron. In a perceptron, a single unit performs all the computations and it can have multiple inputs. On these inputs, the unit performs some computation and output a single value. The computations performed by the unit are a simple matrix multiplication of the input and the weights. The resulting values are summed up and a bias is added.

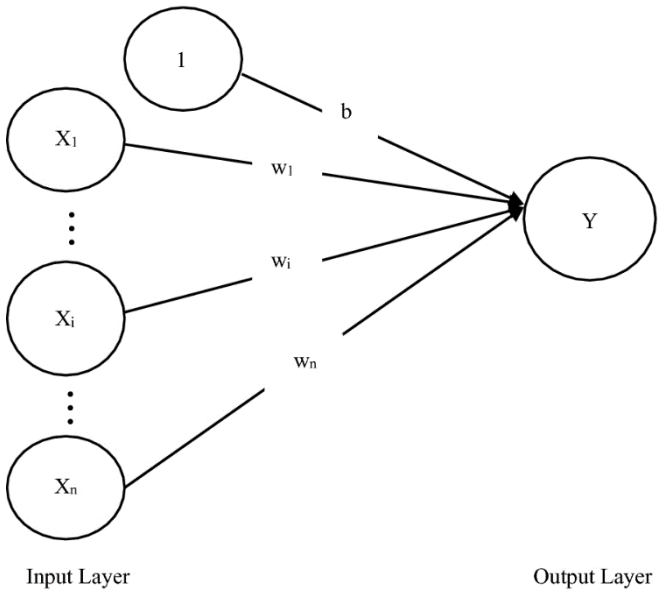

Figure 2. Perceptron (https://link.springer.com/chapter/10.1007/978-981-13-7430-2_13)

Each connection has a weight and a bias. A weight is the strength of the connection. The greater the weight, the greater the impact it will have on the final output. A bias is a minimum threshold which the sum of all weighted input must cross. Neural networks are used for classification tasks and consist of three layers – input, output and hidden layers[1]. Hidden layers are sets of features based on the previous layer. They are intermediate layers in the network. A neural network recognizes objects based on the concept of learning[2]. Learning consists of six steps. Initially weights are initialized and a batch of data is fetched. This data is known as training data. A forward propagation is done on the data by passing through the network. A metric of difference between expected output and actual output is computed by the use of activation functions which perform computations on the data based on standard mathematical distributions. This is known as cost. The goal is to minimize or reduce the cost. For this purpose, gradients of cost and weight are backpropagated to know how to adjust the weights to reduce the cost. Backpropagation refers to a backward pass of the network[3][4][5]. Later, the weights are updated and the whole process is repeated. Feed-forward networks are also termed as multi-layer perceptron.

2.2 Convolutional neural networks

Convolutional neural networks, or ConvNets[6], are designed to process data that come in the form of multiple arrays, for example a colour image composed of three 2D arrays containing pixel intensities in the three colour channels. Many data modalities are in the form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images. There are four key ideas behind ConvNets that take advantage of the properties of natural

signals: local connections, shared weights, pooling and the use of many layers.

The architecture of a typical ConvNet (Figure 3) is structured as a series of stages. The first few stages are composed of two types of layers: convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linearity such as a ReLU. All units in a feature map share the same filter bank. Different feature maps in a layer use different filter banks. The reason for this architecture is twofold. First, in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected. Second, the local statistics of images and other signals are invariant to location. In other words, if a motif can appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array. Mathematically, the filtering operation performed by a feature map is a discrete convolution, hence the name.

Although the role of the convolutional layer is to detect local conjunctions of features from the previous layer, the role of the pooling layer is to merge semantically similar features into one. Because the relative positions of the features forming a motif can vary somewhat, reliably detecting the motif can be done by coarse-graining the position of each feature. A typical pooling unit computes the maximum of a local patch of units in one feature map (or in a few feature maps). Neighbouring pooling units take input from patches that are shifted by more than one row or column, thereby reducing

the dimension of the representation and creating an invariance to small shifts and distortions. Two or three stages of convolution, non-linearity and pooling are stacked, followed by more convolutional and fully-connected layers. Backpropagating gradients through a ConvNet is as simple as through a regular deep network, allowing all the weights in all the filter banks to be trained.

Deep neural networks exploit the property that many natural signals are compositional hierarchies, in which higher-level features are obtained by composing lower-level ones. In images, local combinations of edges form motifs, motifs assemble into parts, and parts form objects. Similar hierarchies exist in speech and text from sounds to phones, phonemes, syllables, words and sentences. The pooling allows representations to vary very little when elements in the previous layer vary in position and appearance.

The convolutional and pooling layers in ConvNets are directly inspired by the classic notions of simple cells and complex cells in visual neuroscience[7], and the overall architecture is reminiscent of the LGN–V1–V2–V4–IT hierarchy in the visual cortex ventral pathway[8]. When ConvNet models and monkeys are shown the same picture, the activations of high-level units in the ConvNet explains half of the variance of random sets of 160 neurons in the monkey's inferotemporal cortex[9]. ConvNets have their roots in the neocognitron[10], the architecture of which was somewhat similar, but did not have an end-to-end supervised-learning algorithm such as backpropagation. A primitive 1D ConvNet called a time-delay neural net was used for the recognition of phonemes and simple words[11][12].

There have been numerous applications of convolutional networks going back to the early 1990s, starting with time-delay neural networks for speech recognition[11] and document reading[13]. The document reading system used a ConvNet trained jointly with a probabilistic model that implemented language constraints. By the late 1990s this system was reading over 10% of all the cheques in the United States. A number of ConvNet-based optical character recognition and handwriting recognition systems were later deployed by Microsoft[14]. ConvNets were also experimented with in the early 1990s for object detection in natural images, including faces and hands[15][16] , and for face recognition[17].

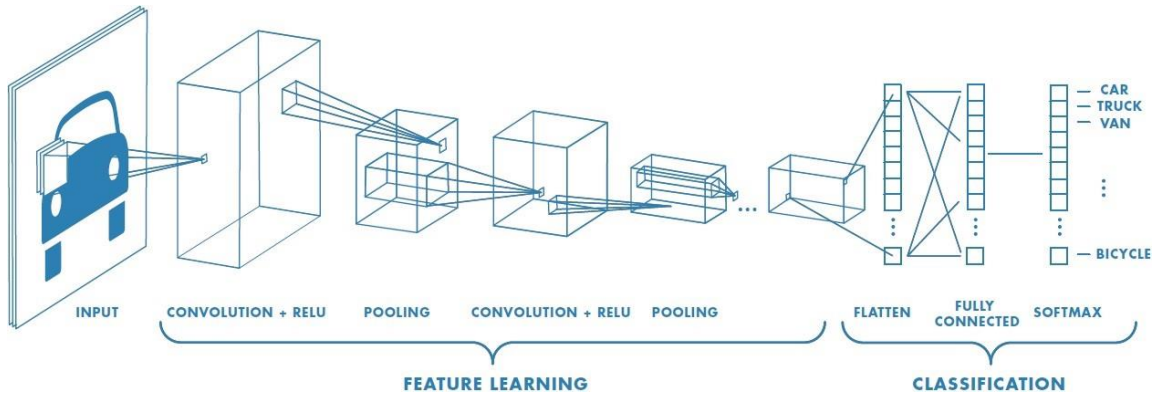

Figure 3.Example of CNN architecture (https://medium.com/@rohanthomas.me/convolutional-networks-for-everyone-1d0699de1a9d)

3. Tools

To be able to create a neural network I needed a set of tools that was useful to the development of this project. I used TensorFlow as backend in order to create a deep neural network and Keras as API to write less lines of code. CUDA was implemented to use the power of the GPU to execute parallel computing. Anaconda was used to build a virtual environment and eventually, in order to operate functions on tensors, I had also to use NumPy and Scikit-learn.

3.1 TensorFlow

TensorFlow[2] is a software framework for numerical computation based on dataflow graphs. It is designed primarily as an interface for expressing and implementing machine learning algorithms, chief among them deep neural networks. Data such as images enters this network as input and flows through the network as it adapts itself at training time or predicts outputs in a deployed system. Tensors are the standard way of representing data in deep learning. They are just multidimensional array, for example black-and-white images are represented as “tables” of pixel values.

In TensorFlow, computation is approached as a dataflow graph and in this graph, nodes represent operations and edges represent data flowing around the system. TensorFlow was designed with portability and flexibility in mind, allowing researchers to express model with relative ease, in fact it is sometimes revealing to think of modern deep learning research and practice as playing with “LEGO-like” bricks [18].

[2] https://www.tensorflow.org

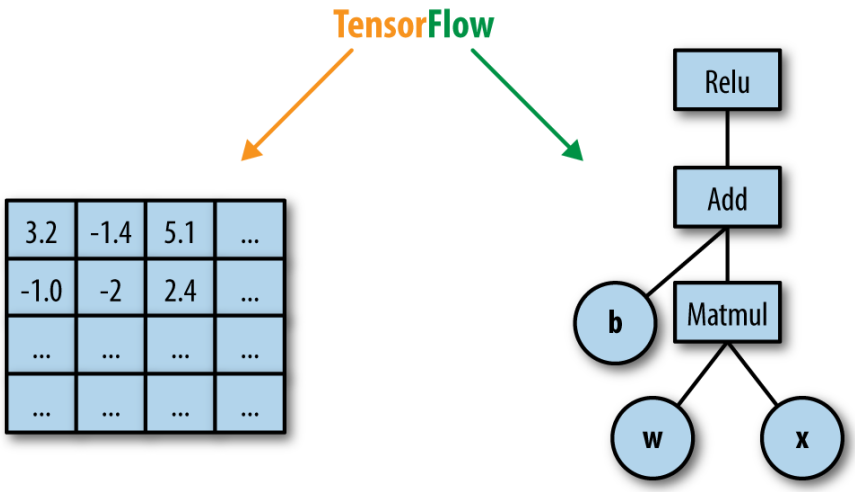

Figure 4. Dataflow graph [18]

3.2 Keras

Keras[3] is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It was developed with a focus on enabling fast experimentation.

Guiding principles

- User friendliness: Keras is an API designed for human beings, not machines. It puts user experience front and centre.
- Modularity: a model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible.
- Easy extensibility: new modules are simple to add, and existing modules provide ample examples.
- Work with Python: models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

[3] https://www.keras.io/

3.3 CUDA

CUDA[4] is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing[19] – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming. CUDA-powered GPUs also support programming frameworks such as OpenACC and OpenCL; and HIP by compiling such code to CUDA. When CUDA was first introduced by Nvidia, the name was an acronym for Compute Unified Device Architecture, but Nvidia subsequently dropped the common use of the acronym.

3.4 Anaconda

Anaconda[5] is an open-source distribution of the Python programming language for scientific computing, that aims to simplify package management and deployment. Conda analyses the current environment including everything currently installed, works out how to install a compatible set of dependencies and shows warning if this cannot be done.

[4] https://developer.nvidia.com/cuda-gpus
[5] https://www.anaconda.com/

Anaconda distribution comes with 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages[citation needed. It will install a package and any of its dependencies regardless of the state of the existing installation[citation needed. Because of this, a user with a working installation of, for example, Google TensorFlow, can find that it stops working having used pip to install a different package that requires a different version of the dependent NumPy library than the one used by TensorFlow. In some cases, the package may appear to work but produce different results in detail.

In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g. the user may wish to have TensorFlow version 2,0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the `conda install` command. Anaconda Inc compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit.

Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed.

Custom packages can be made using the `conda build` command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

3.5 NumPy

NumPy[6] is a library for the Python programming language, adding support for large, multi-dimensional array and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

The Python programming language was not initially designed for numerical computing but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van

[6] https://www.numpy.org/

Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become *Numeric*, also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called *Numarray* was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.

In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006. This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0.

3.6 Scikit-learn

Scikit-learn[7] (formerly scikits.learn and also known as sclera) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately developed and distributed third-party extension to SciPy. The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012. Scikit-learn is one of the most popular machine learning libraries on GitHub.

[7] https://www.scikit-learn.org/

4. Project development

In this chapter I am going to explain the approach that I used to create a Neural Network that is capable of recognize emotions. Different steps were needed to develop this project: the importation of the libraries, the preprocessing[20] of the data, the creation of the model architecture, the training of the model and eventually the tuning of the hyperparameters.

4.1 Approach

Imported libraries:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.layers.normalization import BatchNormalization
from keras.callbacks import EarlyStopping
```

Variables declaration:

```
num_classes = 7
epochs = 15
batch_size = 256
```

The dataset consisted of a number of images represented as strings of 2304 space separated numbers which were then converted to a 48*48 matrix. Each number represented a pixel value. The original data of 35,887 images was split into a training set of 28,709 images and 3,589 images for the testing and validation sets - an 80-10-10 split. Generally, when it comes to

deep learning, data is the biggest factor. The bigger the training set, the better the output. If there is less training data, there is a lot more variance in the final outputs due to a smaller set to train on.

```
with open(r"D:\Users\Stefano\Desktop\fer2013.csv") as file:
    content = file.readlines()


x_train, y_train, x_test, y_test, x_val, y_val = [], [], [], [], [], []

# assigning the training, test and validation images to them variables
for i in range(1, num_instances):
    try:
        emotion, img, usage = lines[i].split(",")

        val = img.split(" ")

        pixels = np.array(val, 'float32')

        emotion = to_categorical(emotion, num_classes)

        if 'Training' in usage:
            y_train.append(emotion)
            x_train.append(pixels)
        elif 'PublicTest' in usage:
            y_test.append(emotion)
            x_test.append(pixels)
        elif 'PrivateTest' in usage:
            y_val.append(emotion)
            x_val.append(pixels)
    except:
        print("", end="")
```

Normalization of the data:

```
# data transformation for train, test and validation sets
x_train = np.array(x_train, 'float32')
y_train = np.array(y_train, 'float32')
x_test = np.array(x_test, 'float32')
y_test = np.array(y_test, 'float32')
x_val = np.array(x_val, 'float32')
y_val = np.array(y_val, 'float32')

# normalization
x_train /= 255
x_test /= 255
x_val /= 255

# reshaping images as 48x48 matrix
x_train = x_train.reshape(x_train.shape[0], 48, 48, 1)
x_train = x_train.astype('float32')
x_test = x_test.reshape(x_test.shape[0], 48, 48, 1)
```

```
x_test = x_test.astype('float32')
x_val = x_val.reshape(x_val.shape[0], 48, 48, 1)
x_val = x_val.astype('float32')
```

A callback is a set of functions to be applied at given stages of the training procedure. It can be used to get a view on internal states and statistics of the model during training. It is possible to pass a list of callbacks to the .fit() method of the model.

In this case was implemented an EarlyStopping callback that stops the training when a monitored quantity has stopped improving after a certain number of epochs.

```
#defining a callback to prevent overfitting
callbacks = [EarlyStopping(monitor='val_loss', patience=5, verbose=1)]
```

The next step is the most important one, it involves the designing of the model architecture through which will be passed the features to train the model and eventually test it using the test features. To build this CNN model was used a combination of different functions that are going to be defined below:

- Sequential is just a linear stuck of layers which is putting layers on top of each other as the architecture progress from the input layer to the output layer.

- Conv2D is a two-dimensional Convolutional layer which performs the convolution operation. Here was used a 3x3 kernel size and Rectified Linear Unit as our activation function. This function squashes any input that is negative to zero and leaves positive number as they are. We can visualize the function as follow:

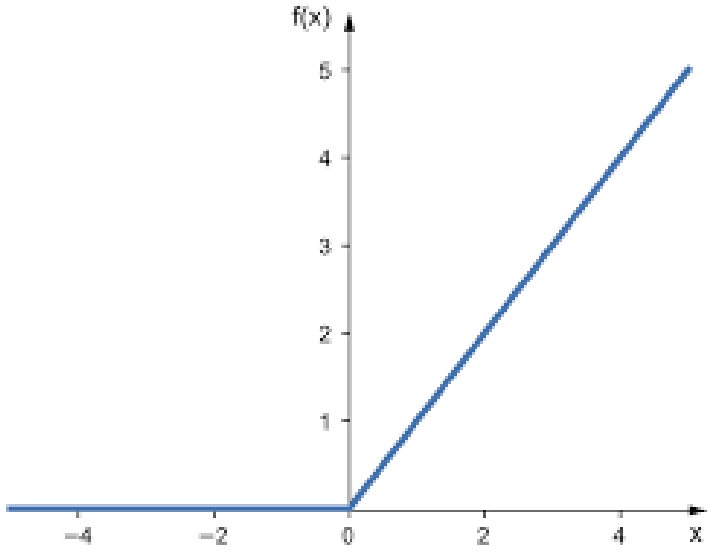

Figure 5. RElu function (https://www.researchgate.net/figure/The-plot-of-the-ReLU-function_fig5_335540811)

  It makes the convergence of stochastic gradient descend faster and it is computationally inexpensive, as we are just thresholding.
- BatchNormalization normalizes the inputs of the current batch before feeding it to the next layer. It behaves differently during training and validation or testing. During training, the mean and the variance is calculated for the data in the batch. For validation and testing, the global values are used.
- MaxPooling2D is a popular optimization technique for CNNs, as it reduces the size of feature maps and the outcome of convolution layers. It offers two different features: one is reducing the size of data

to process, and the other is forcing the algorithm to not focus on small changes in the position of an image.

The following image is a demonstration of how this function works:

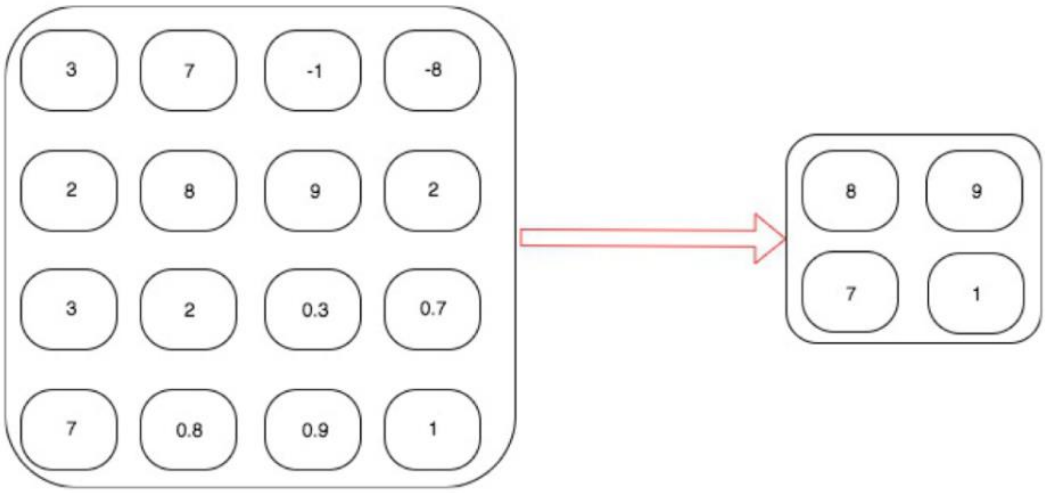

Figure 6. MaxPooling2D function [21]

- Dropout is a method for regularization. It forces a neural network to learn multiple independent representations by randomly removing connections between neurons in the learning phase. We can see an example below when using a dropout of 0.2:

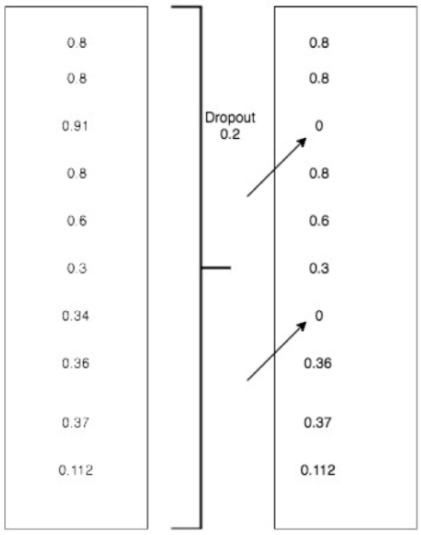

Figure 7. Dropout function [21]

- Flatten is a function that flattens the input from the two-dimension to one-dimension and does not affect the batch size

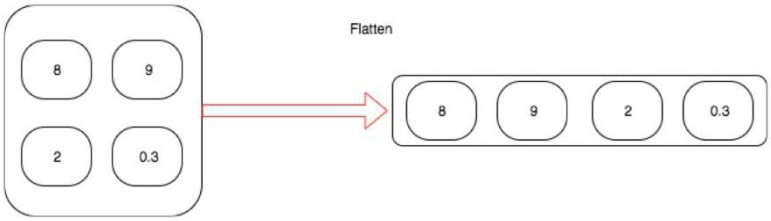

Figure 8. Flatten function [21]

- Dense implements the operation: output = activation(dot(input, kernel)) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer. In simple words, it uses the features learned using the layers and maps it to the label. During testing, this layer is responsible for creating the final label for the image being processed.[4]

```
# construct CNN structure
model = Sequential()

# 1st layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1),
padding='same'))
model.add(BatchNormalization())

# 2nd layer
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

model.add(Dropout(0.25))

# 3rd layer
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

model.add(Dropout(0.25))

# 4th layer
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
```

```
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

model.add(Dropout(0.25))

# classification
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Using the command model.summary() I obtained the architecture:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 32)        320
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 46, 46, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 23, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 23, 23, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 64)        36928
_________________________________________________________________
batch_normalization_4 (Batch (None, 23, 23, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
```

```
dropout_2 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 128)       147584
_________________________________________________________________
batch_normalization_6 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               2359808
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 903
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 903
=================================================================
Total params: 2,716,647
Trainable params: 2,714,727
Non-trainable params: 1,920
_________________________________________________________________
None
```

Figure 9. Architecture of the model

In the next chunk of code the model was compiled with:

- Categorical crossentropy: a loss function that is used for single label categorization. This is when only one category is applicable for each data point.
- Adam optimizer helps us in adjusting random weight created initially to help the model calculate target values more accurately.

Accuracy was used as metric for validation.

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

As loss function I used “Categorical cross-entropy”, Also called Softmax Loss. It is a Softmax activation plus a Cross-Entropy loss. If we use this loss, we will train a CNN to output a probability over the C classes for each image. It is used for multi-class classification.

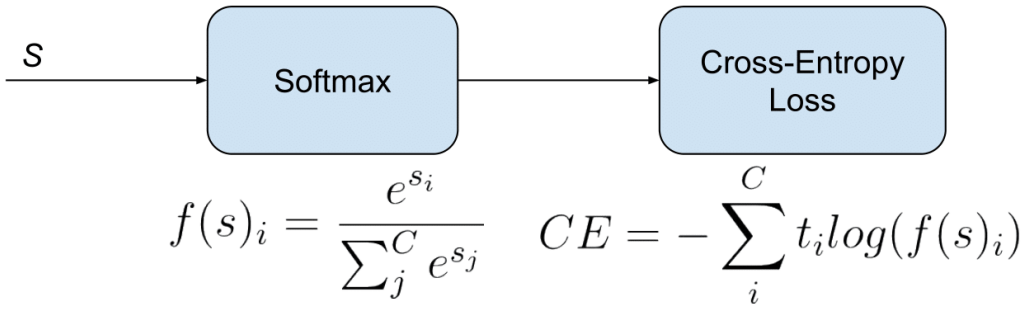

Figure 10. Categorical cross-entropy loss function. https://gombru.github.io/2018/05/23/cross_entropy_loss/

The model was trained.

```
model.fit(x_train,
          y_train,
          batch_sizes=batch_size,
          epochs=epochs,
          validation_data=(x_val, y_val),
          callbacks=callbacks)
```

4.2 Calibration of the hyperparameters

Deep learning optimization often entails fine tuning elements that live outside the model but that can heavily influence its behaviour. Deep learning often refers to those hidden elements as hyperparameters as they are one of the most critical components of any machine learning application. Hyperparameters are settings that can be tuned to control the behaviour of a machine learning algorithm. Conceptually, hyperparameters can be considered orthogonal to the learning model itself in the sense that, although they live outside the model, there is a direct relationship between them.

The criteria of what defines a hyperparameter is incredibly abstract and flexible. Sure, there are well established hyperparameters such as the number of hidden units or the learning rate of a model but there are also an arbitrarily number of settings that can play the role of hyperparameters for specific models. Sometimes, a setting is modelled as a hyperparameter because is not appropriate to learn it from the training set. A classic example are settings that control the capacity of a model (the spectrum of functions that the model can represent). If a deep learning algorithm learns those settings directly from the training set, then it is likely to try to optimize for that dataset which will cause the model to overfit (poor generalization).

To get the best performance from this model I did some tuning on the following hyperparameters.

- The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

  Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model.

```
        batch_size = 256
```

In this case I found that a batch size of 256 slowed down the training but at the same time gave a bigger stability of the model.

- The number of epochs is a hyperparameter that defines the number 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. An epoch is comprised of one or more batches.

  You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples.

  I tried to experiment with different number of epochs and I found that a number of epochs equal to 25 was the maximum before getting to overfitting[22].

```
epochs = 25
```

5 Evaluation

The dataset used in this project is from Kaggle[8], it allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

Usually neural networks tend to perform better with larger amount of training dataset. For this reason, the Facial Expression dataset (FER-2013) was chosen.

The set is composed of 35887 images and it is divided in 80% for the training data and 10% for the testing data and 10% for the validation data.

The images are divided into 7 emotions: angry, disgust, fear, happy, sad, surprise and neutral.

It is possible to see the distribution of the images in classes from the figure below:

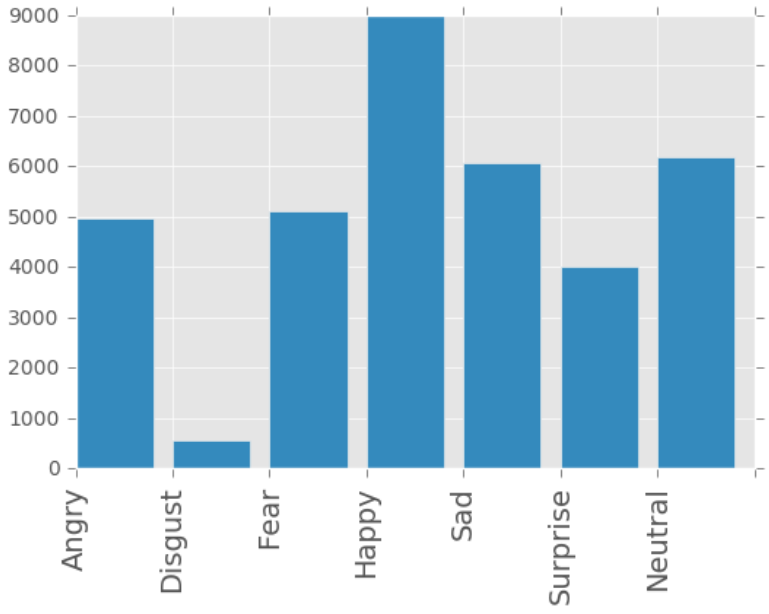

Figure 11. Training data distribution

[8] https://www.kaggle.com/

As we set the variables before the data will pass through the model 25 times and in batches of 256 images. These were the values of the variables because after tuning was found that a batch size of 256 slowed down the training but led the model to be more stable and 25 for the epochs as increasing this number will led to overfitting.

After executing the code the following results were obtained:

Epoch 1/25

1/256 [.....] - ETA: 30:32 - loss: 4.0953 - accuracy: 0.0898

2/256 [.....] - ETA: 22:16 - loss: 4.0062 - accuracy: 0.0723

3/256 [.....] - ETA: 19:25 - loss: 3.8450 - accuracy: 0.0846

.

.

.

256/256 [==========] - 869s 3s/step - loss: 0.1898 - accuracy: 0.9326 - val_loss: 1.6191 - val_accuracy: 0.6481

Test loss: 1.6873480854475522

Test accuracy: 64.11256790161133

That means that I obtained the following results on the two sets:

Train loss: 0.1898

Train accuracy: 0.9326

Test loss: 1.6873

Test accuracy: 64.1125

The accuracy formula is detailed below. It simply counts the number of samples where the model correctly predicted the emotion and divides it by the total number of samples in the testing set.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{All Samples}}$$

Figure 12. Equation of accuracy (https://lawtomated.com/accuracy-precision-recall-and-f1-scores-for-lawyers/)

A confusion_matrix is useful for quickly calculating precision and recall given the predicted labels from a model. The actual values form the columns, and the predicted values (labels) form the rows. The intersection of the rows and columns show one of the outcomes.

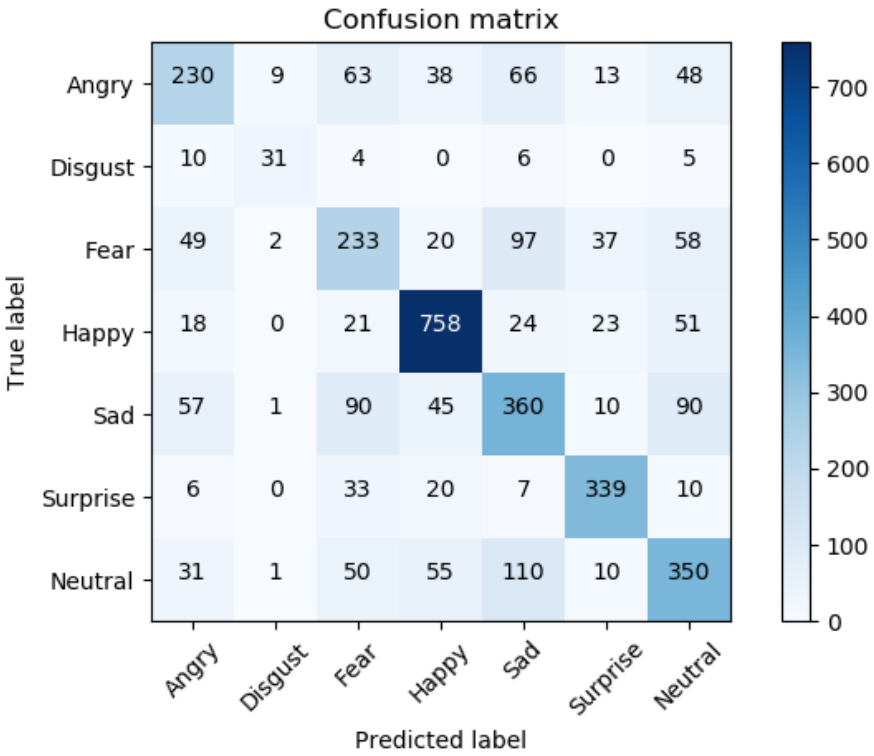

Figure 13. Confusion matrix of the model

As we can see some emotions like “angry” and “disgust” were confused with each other as they are very similar negative emotions.

To better evaluate the project I printed different metrics, some of the most important are precision and recall.

Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. False positives are cases the model incorrectly labels as positive that are actually negative, or in our example, individuals the model classifies as terrorists that are not. While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant.

Recall expresses the ability of a model to find all the relevant cases within a dataset. The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives. True positives are data point classified as positive by the model that actually are positive (meaning they are correct), and false negatives are data points the model identifies as negative that actually are positive (incorrect).

	precision	recall	f1-score	support
angry	0.5736	0.4925	0.5300	467
disgust	0.7045	0.5536	0.6200	56
fear	0.4717	0.4698	0.4707	496
happy	0.8098	0.8469	0.8280	895
sad	0.5373	0.5513	0.5442	653
surprise	0.7847	0.8169	0.8005	415
neutral	0.5719	0.5766	0.5742	607
accuracy			0.6411	3589
macro avg	0.6362	0.6154	0.6239	3589
weighted avg	0.6380	0.6411	0.6389	3589

Figure 14. Metrics of the model

I wrote a function that prints some images from the dataset included in an interval and their predictions to see how the model operated, obtaining the following results:

```
def check_img_dataset(first, second):

    predictions = model.predict(x_test)

    index = 0
    for i in predictions:
        if first < index < second:

            test_img = np.array(x_test[index], 'float32')
            test_img = test_img.reshape([48, 48]);

            plt.gray()
            plt.imshow(test_img)
            plt.show()

            emotion_analysis(i)

        index = index + 1
```

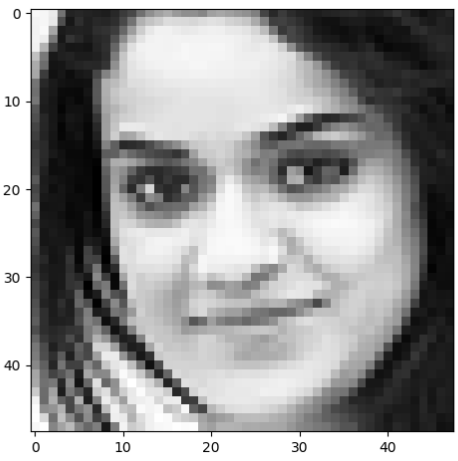

Figure 16. Analysed image 1

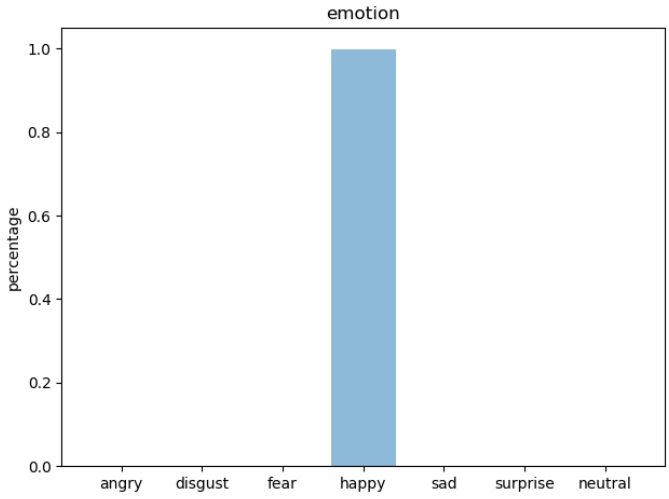

Figure 15. Activation function value for the output layer of analysed image 1

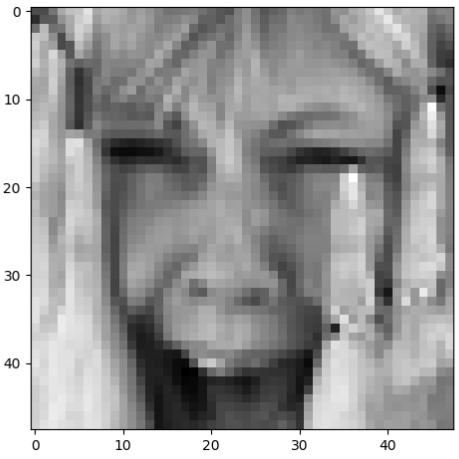

Figure 18. Analysed image 2

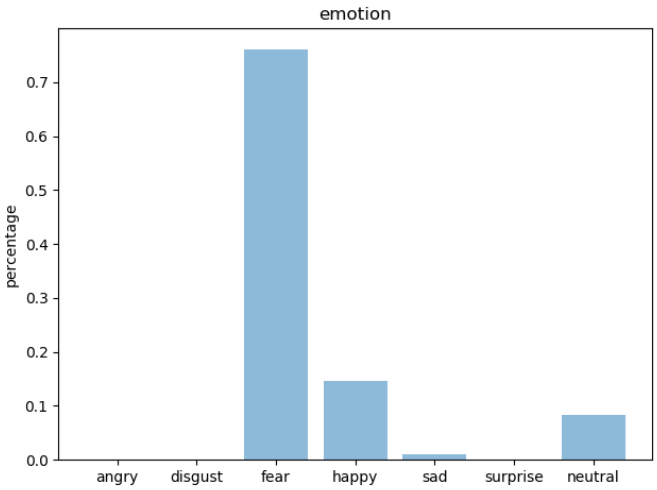

Figure 17. Figure 13. Activation function value for the output layer of analysed image 2

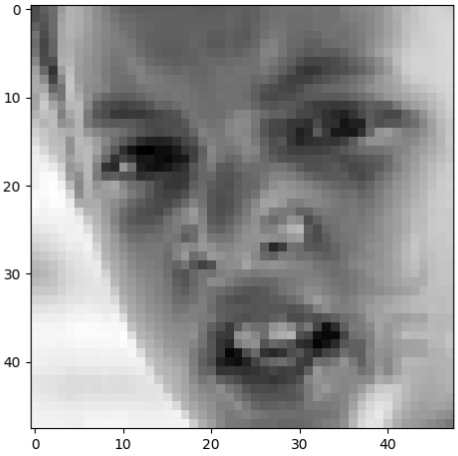

Figure 20. Analysed image 3

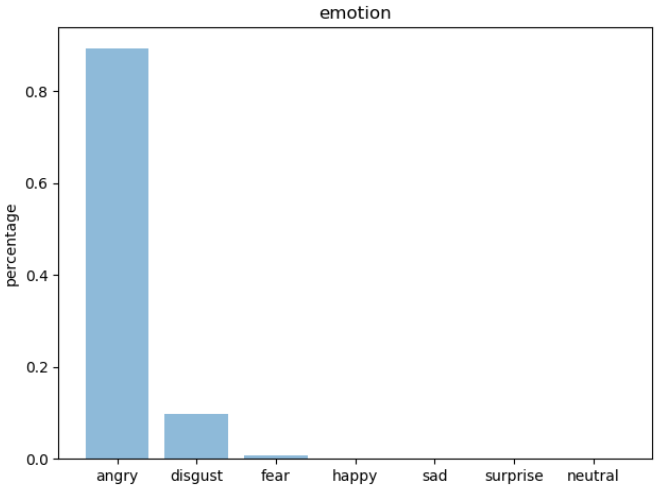

Figure 19. Activation function value for the output layer of analysed image 3

It is evident from the last image that the first two emotions are easy to confuse, as they are similar negative emotions.

I decided also to test the model with custom images, that I have took with my smartphone and the results are shown below.

```
def check_img(img_path):
    img = load_img(img_path, color_mode='grayscale', target_size=(48, 48))

    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)

    x /= 255

    custom = model.predict(x)
    emo_analyzer(custom[0])

    x = np.array(x, 'float32')
    x = x.reshape([48, 48])

    plt.gray()
    plt.imshow(x)
    plt.show()
```

Figure 21. Normal expression photo

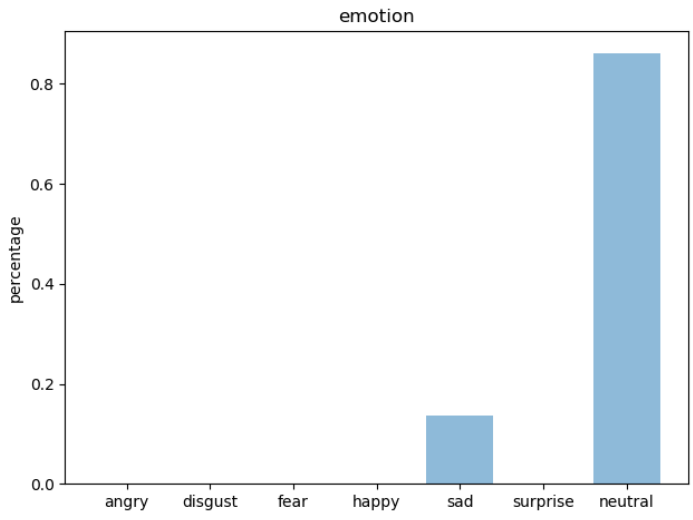

Figure 22. Activation function value for the output layer of normal expression photo

Figure 23. Surprised expression photo

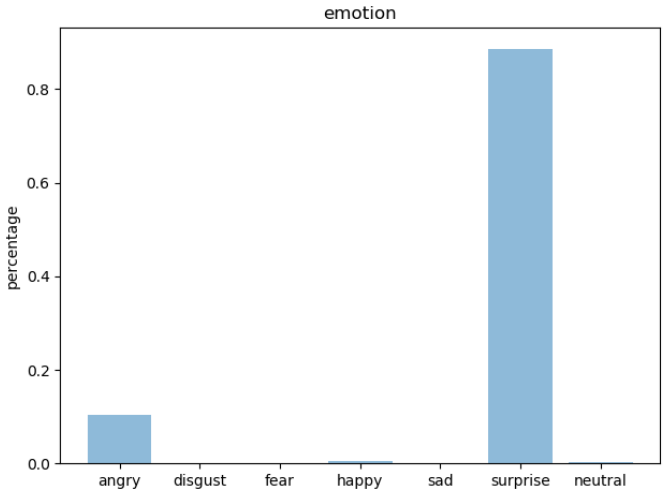

Figure 24. Activation function value for the output layer of surprised expression photo

Figure 26. Happy expression photo

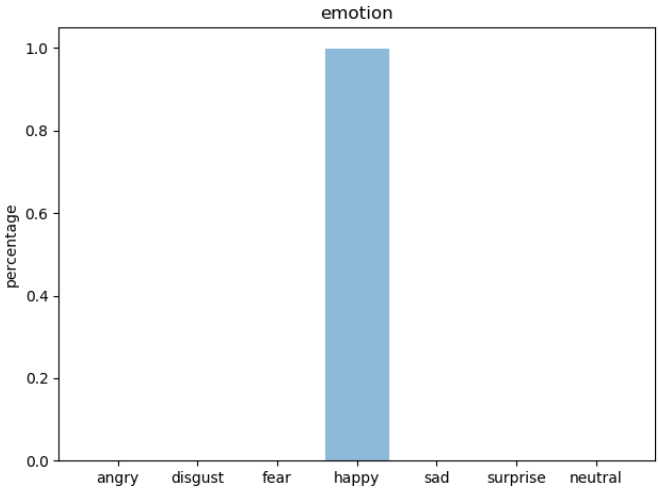

Figure 25. Activation function value for the output layer of happy expression photo

It is possible to see that the model is more accurate with the emotions that have the bigger number of instances inside the dataset. However, the accuracy of the model is acceptable as it recognizes each facial expression.

6. Conclusions

The aim of this project was to classify facial expressions into one of seven emotions by using the FER-2013 dataset. To increase the accuracy of the model the hyperparameters were tuned. The final accuracy of 64% was achieved using the Adam optimizer.

It should also be noted that a nearly state-of-the-art accuracy was achieved with the use of a single dataset as opposed to a combination of many datasets.

Given that only the FER-2013 dataset was used in this case without the use of other datasets, an accuracy of 64% is admirable as it demonstrates the efficiency of the model. In other words, the model demonstrated has used significantly less data for training and a deep but simple architecture.

The relatively lower amount of data for emotions such as “disgust” make the model have difficulty predicting it. This however does illuminate a path for future work. If provided with more training data while still retaining the same network structure, the efficiency of the proposed system will be enhanced considerably. Thus, augmenting the existing data to enlarge the dataset might also prove to be a worthwhile avenue to explore.

In the future, an in-depth analysis of the top 2 predicted emotions may lead to a much more accurate and reliable system. Further training samples for the more difficult to predict emotion of disgust will definitely be required in order to perfect such a system.

The quick training and accuracy allow the model to be adapted and used in nearly any use-case. This also implies that with some work, the model could

very well be deployed into real-life applications for effective utilization in domains such as in safety, healthcare, gaming and entertainment.

7. Index of Figures

8. Bibliography

[1] Indra den Bakker, Python Deep Learning Cookbook: Over 75 Practical Recipes on Neural Network modelling, Reinforcement Learning, and Transfer Learning Using Python, October 2017: First Edition

[2] Sutskever, I. Vinyals, O. & Le. Q. V. Sequence to sequence learning with neural networks. In *Proc. Advances in Neural Information Processing Systems 27* 3104–3112 (2014).

[3] Bengio, Y., Lamblin, P., Popovici, D. & Larochelle, H. Greedy layer-wise training of deep networks. In Proc. Advances in Neural Information Processing Systems 19 153–160 (2006).

[4] Ranzato, M., Poultney, C., Chopra, S. & LeCun, Y. Efficient learning of sparse representations with an energy-based model. In Proc. Advances in Neural Information Processing Systems 19 1137–1144 (2006).

[5] Hinton, G. E. & Salakhutdinov, R. Reducing the dimensionality of data with neural networks. Science 313, 504–507 (2006).

[6] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539

[7] Hubel, D. H. & Wiesel, T. N. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. J. Physiol. 160, 106–154 (1962).

[8] Felleman, D. J. & Essen, D. C. V. Distributed hierarchical processing in the primate cerebral cortex. Cereb. Cortex 1, 1–47 (1991).

[9] Cadieu, C. F. et al. Deep neural networks rival the representation of primate it cortex for core visual object recognition. PLoS Comp. Biol. 10, e1003963 (2014).

[10] Fukushima, K. & Miyake, S. Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shifts in position. Pattern Recognition 15, 455–469 (1982).

[11] Waibel, A., Hanazawa, T., Hinton, G. E., Shikano, K. & Lang, K. Phoneme recognition using time-delay neural networks. IEEE Trans. Acoustics Speech Signal Process. 37, 328–339 (1989).

[12] Bottou, L., Fogelman-Soulié, F., Blanchet, P. & Lienard, J. Experiments with time delay networks and dynamic time warping for speaker independent isolated digit recognition. In Proc. EuroSpeech 89 537–540 (1989).

[13] LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. Proc. IEEE 86, 2278–2324 (1998).

[14] Simard, D., Steinkraus, P. Y. & Platt, J. C. Best practices for convolutional neural networks. In Proc. Document Analysis and Recognition 958–963 (2003).

[15] Vaillant, R., Monrocq, C. & LeCun, Y. Original approach for the localisation of objects in images. In Proc. Vision, Image, and Signal Processing 141, 245–250 (1994).

[16] Nowlan, S. & Platt, J. in Neural Information Processing Systems 901–908 (1995).

[17] Lawrence, S., Giles, C. L., Tsoi, A. C. & Back, A. D. Face recognition: a convolutional neural-network approach. IEEE Trans. Neural Networks 8, 98–113 (1997).

[18] Tom Hope, Yehezkel S. Resheff & Italy Lieder, Learning TensorFlow, A Guide to Building Deep Learning Systems, September 2017: Second Edition

[19] Raina, R., Madhavan, A. & Ng, A. Y. Large-scale deep unsupervised learning using graphics processors. In Proc. 26th Annual International Conference on Machine Learning 873–880 (2009).

[20] Chris Albon, Python Machine Learning Cookbook, Practical Solutions From Preprocessing to Deep Learning, July 2017: First Edition

[21] Manas Agarwal, Deep Learning with PyTorch, A practical approach to building neural network models using PyTorch, February 2018: First Edition

[22] Bengio, Y., Courville, A. & Vincent, P. Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Machine Intell. 35, 1798–1828 (2013).

9. Thanks

I would like to thank the prof. Massimiliano Di Penta and the doctor Fiorella Zampetti for constantly following me, despite the adversities that we are facing in this historic moment, in this project showing their professionalism.

Finally, I would particularly like to thank my family without whom I could not have achieved this title, thanks to their closeness and their love I was able to overcome the toughest moments of this path.