



UNIVERSITA' DEGLI STUDI DI  
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Magistrale in Ingegneria Informatica



Master's degree thesis in Computer Engineering

# *Multimodal analysis of emotions in multimedia*

Academic year 2019/2020

Supervisors

**Ch.mo prof. Sandra Gómez Canaval**

**Ch.mo prof. Vincenzo Moscato**

Candidate

**Antonio Giordano**

**matr. Unina M63000588**

**matr. Etsisi BR0037**

# Abstract

The goal of this work is to present a combined approach of emotion detection using two subsystems for video and audio. We treat emotion recognition as classification task by using deep learning models to process encoded emotions on different datasets: Affwild [1] and Fer2013 [3] for the video subsystem and TESS and RAVDESS [4] for the audio subsystem. We used three networks for the video subsystem: ResNet50 [10], VGG16 [7] and a custom CNN. A 1-dimensional CNN has been used for the audio part. In all networks we used dropout, flatten and dense layers, only for the custom CNN we added a MaxPooling layer. We analyzed how each neural network performs singulary and also built a postprocessing step that is able to run both subsystems at same time and provide a combined prediction on the video input. The entire project is written in Python, and the deep learning networks are implemented using Tensorflow/Keras [2].

# Contents

<b>Abstract</b>	<b>i</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Preamble . . . . .	3
1.2 Context . . . . .	3
1.3 Motivation and Justification . . . . .	4
1.4 Goal . . . . .	4
1.4.1 Main Goal . . . . .	5
1.4.2 Specific Goals . . . . .	5
1.5 Document Structure . . . . .	6
<b>2 Theoretical Background</b>	<b>7</b>
2.1 Artificial Intelligence . . . . .	7
2.1.1 Types of algorithms . . . . .	9
2.1.2 Machine Learning process . . . . .	11
2.2 Deep Learning . . . . .	14
2.2.1 A "bit" of History . . . . .	15
2.2.2 Fundamentals of Artificial Neural Networks . .	16
2.2.3 Fundamentals of Deep Neural Networks . . . .	18
2.2.4 Types of Deep Neural Networks . . . . .	19

2.2.5	Quality Evaluation of Deep Neural Models . . .	21
2.3	Multimodal Deep Learning . . . . .	22
2.3.1	Multimodal Learning Representations . . . . .	26
2.3.2	Architectures, Techniques, Strategies . . . . .	27
<b>3</b>	<b>State of Art</b>	<b>30</b>
3.1	Introduction . . . . .	30
3.2	Comparison table of ER systems . . . . .	31
<b>4</b>	<b>Proposed Model</b>	<b>34</b>
4.1	Methodology . . . . .	34
4.2	Problem setting . . . . .	35
4.3	Technological framework . . . . .	35
4.4	Proposed Solution . . . . .	36
4.5	Hyperparameters selection and optimization . . . . .	37
4.6	Training model . . . . .	40
4.7	Data preparation and preprocessing . . . . .	42
4.8	Validation model . . . . .	44
4.9	Training and Validation Evaluation . . . . .	46
<b>5</b>	<b>Testing, Experiments and Result Analysis</b>	<b>54</b>
5.1	Testing model and evaluation . . . . .	54
5.2	Set of Experiments . . . . .	54
5.3	Results Analysis . . . . .	59
<b>6</b>	<b>Conclusions and Future Work</b>	<b>60</b>
6.1	Conclusions . . . . .	60
6.2	Future Work . . . . .	61

*To Prof. Sandra Gómez Canaval for her professional and all help she  
gave me during my stay in Madrid, thanks a lot.*

*To Prof. Vincenzo Moscato for his help, despite the distance.*

*To me...*

*"And once the storm is over, you won't remember how you made it  
through, how you managed to survive. You won't even be sure  
whether the storm is really over. But one thing is certain. When you  
come out of the storm, you won't be the same person who walked in.*

*That's what this storm's all about."*

*[Haruki Murakami]*

# Chapter 1

## Introduction

This work presents an emotion recognition system from audio and video. It includes methods for: feature extraction, feature standardization, hyperparameter selection, neural networks, and algorithm parameters optimization. The considered emotional states are represented by seven classes: neutral, anger, disgust, fear, happiness, sadness, and surprise. Developing technologies for automatic emotion recognition is becoming increasingly demanded in the last years. It has many applications in environments where machines need to act and collaborate with human beings. Indeed, it finds applications in: *personnel selection, social integration of children with special needs, didactic, cybertherapy, commercial areas robotics, automobile industry, the entertainment industry, affect-aware learning systems, recommendation systems for tourism, affect-aware smart city, and intelligent conversational system* and the *the marketing industry*.

We could find many others, since a human being is an emotional being who thinks more than a thinking being who feels emotions. Finally, a concept to keep in mind to understand the importance of this work is the **emotional intelligence**: it is the ability of a person to feel, express, recognize, and handle the emotional state of other people, and the latest developments of deep learning in emotion recognition will help people to improve their emotional intelligence.

## 1.1 Preamble

*"While there are commercial tools that predict "emotion" out there, they are often exaggerating their capabilities, as recognizing internal emotions of someone without additional context is almost impossible. While there are "rules" for converting facial expressions to a set of "basic emotions" they are just rough guidelines and not very accurate due to the subjectivity and ambiguity of the task."* (Tadas Baltrusaitis, OpenFace creator)

## 1.2 Context

EAI (Emotional artificial intelligence) is a relatively new field in AI, but it has a great potential to do immense good to people for people. It goes by various names: "affective computing" or "human-centric artificial intelligence" but actually behind all of those names there is

the same research topic: automatic sensing and analysis of human behavior and in particular human facial behavior. We use the face to recognize other members of our species, but also to judge various things such as age, gender, beauty, and sometimes even personality. The human face is the only observable window of our inner selves and of our intentions, attitudes, and moods.

## 1.3 Motivation and Justification

Main motivation for this work is helping doctors in therapy with autistic children. The human face displays 10,000 different facial expressions, 7,000 of which we display on a daily basis. People classify those expressions in maybe 15 or 20 categories, but autistic children do not have this kind of generalization ability, so they see 7,000 different categories. This is the reason autistic children usually do not look at our faces, they are too confusing for that, but this is also the reason they can not understand our emotions. Furthermore, this work aims to take a step, albeit minimal, forward in this direction: to help people, mainly children, to better manage their own and others' emotions.

## 1.4 Goal

The goal of this work is to recognize the emotional state that a human is experiencing analysing a video, using two separate subsystems for



image and audio classification. Regarding the audio part, the aim will be almost always in understanding *how* someone says something more than *what* he say. As regard as the video part we want to build a baseline system that can be used and replaced in the future with another one if needed.

### 1.4.1 Main Goal

The main goal of this work, is to provide the community with an open source repository that allows the user to accomplish a combined emotion detection in audio and video. Therefore we hope that in the future it can be improved in efficiency and accuracy by the community, that way we will feel a productive part of a far-reaching development process that can be of help to future generations.

### 1.4.2 Specific Goals

Specific goals are to provide the community something that is missing at the moment: an easy guide to the installation of all pre-requirements needed and a single place where to find everything that allows any user to try it and edit as he wish.

## 1.5 Document Structure

The report structure is organized as follows. Section 2 is a theoretical background on machine learning and its application about emotion analysis. In Section 3 a state-of-art overview is illustrated. In Section 4, our ML approach for emotion recognition is presented. Section 5 presents the experimental setup and the experimental results. Finally, the conclusion and a brief discussion about the results is given.

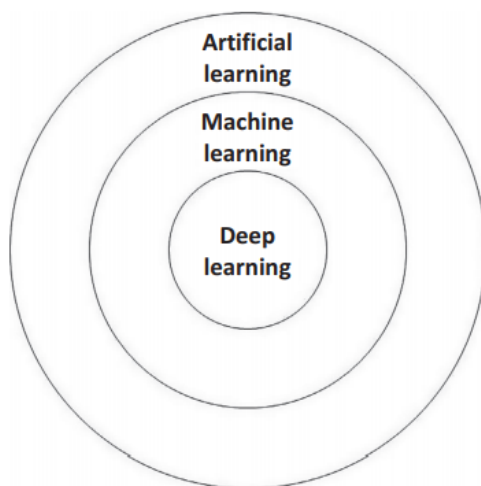
# Chapter 2

## Theoretical Background

### 2.1 Artificial Intelligence

AI is a discipline that studies the design, development, and implementation of systems capable of simulating human skills, reasoning, and behavior. When we talk about AI, we need to understand some key concepts, such as the special **algorithms** needed for these kinds of jobs: here we mean a series of calculations, from the simplest to the most complex. Machine learning uses algorithms to process data and discover its implicit rules to encode them in a “model” useful for processing subsequent data. Machines are always so precise, objective, consistent. However, they often suffer from the same unconscious prejudices as the human beings who made them that we call **bias**. This is why it is essential to avoid human biases accidentally ending up in the algorithm. The fault generally lies with the data that feed the algo-

rithm. One of the most famous algorithms used in AI is the **backpropagation**, indeed they allow a neural network to learn from its errors: proceeding backwards from the final result to the initial forecast, they analyze the margin of error of the various phases to gradually improve subsequent operations. What is very different between classical types of algorithms and AI algorithms is **explainability**, in fact software based on artificial intelligence must not only work well: it must also be explainable. And this also applies to his mistakes. However, this is an open problem for Artificial Neural Network (ANN). Of course, "explainability" is essential so that consumers can trigger the necessary trust in AI but, in this regard, it should be emphasized that the models produced by ANN's neural networks, even if very efficient, cannot be explained in human symbolic language: the results must be accepted "as they are", indeed they need to be interpreted as black boxes.



A large part of AI is machine learning that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access

Figure 2.1: Deep learning concept

data and use it to learn for themselves. **Neural networks** (NN)

is an interesting field for AI. It is a mathematical model composed of artificial neurons inspired by biological neural networks and is used to solve engineering problems of AI related to various technological fields. Inside Machine Learning (ML) we found **Deep Learning**, that is based on particularly stratified and complex neural networks, to emulate the cognitive abilities of the human mind.

### 2.1.1 Types of algorithms

#### *Supervised*

The machine observes a series of cases ("it's winter and it's cold") and their results ("Antonio will turn on the heaters") and learns the rules to be able to predict the results of future similar situations.

An implementation of a supervised algorithm is the classifier. It divides the input objects into classes. When an object is submitted to the classifier, for example, the algorithm returns the class it might belong to. The classification can be linear or nonlinear, with two or more dimensions. The main difference is that the linear ones are simple and fast but suffer from the underfitting problem (they do not fit the data very much), while the nonlinear ones are more precise but suffer from overfitting, that is they adapt too much to the training data and perform poorly on the test data. Among the most famous algorithms

that implement classifiers we include: K-Nearest Neighbor, Support Vector Machine, Decision Tree, Linear Perceptron, MLP.

Another supervised algorithm is the regressor. Regression is used to predict the value of one variable based on the value of the other one. The variable that is used to predict the value of the other variable is called an independent variable. Linear regression corresponds to a straight line or surface that minimizes the difference between the expected and actual output values.

### *Unsupervised*

The machine observes a series of cases and learns to recognize structures within the data, for example "clusters" (sets with similar characteristics). Therefore, they can learn that "it's winter and it's cold" is different from "it's summer and hot".

Some unsupervised algorithms are: Clustering, Anomaly detection, Neural Networks, and approaches for learning latent variable models.

### *Reinforcement Learning*

The machine receives feedback on its operations and learns and improves by trial and error. Example: if a machine suggests Antonio not to go to the beach when it is raining and cold, it could learn to suggest when to do that.

The most used algorithms in reinforcement learning are [11]: Monte Carlo, Q-learning, SARSA, DQN, DDPG, A3C, NAF, TRPO, PPO, TD3, SAC.

## 2.1.2 Machine Learning process

ML process can be divided into 7 steps, listed below.

- Gathering data
- Preparing data
- Choosing a model
- Training
- Evaluation
- Hyperparameter tuning
- Prediction

To develop a generic ML model, the first step is always gathering relevant data that can be used as input to the system. Data collection phase is the foundation of the ML process. In this phase, a common mistake can be a wrong choice of the features or not suitable choice of their types for the model to be developed for. Therefore, a lot of attention is needed because the mistakes made in this phase can be amplified as we approach the final phase.

The next step is to prepare the data. A key goal of this phase is to recognize and minimize any potential data bias. A good tool to use is to randomize the order of the data, so that the results are independent of a predetermined order. Another important component

of data preparation is breaking down the data sets into 2 parts. The larger part (about  $4/5$ ) is usually used for model training, while the smaller part ( $1/5$ ) is used for evaluation purposes. This is important because using the same data for both training and assessment would give a result that is not comparable to what you would get in real life applications. In addition to splitting the data, further steps are performed to refine the data sets, such as the removal of duplicate entries, the elimination of incorrect readings, the removal of redundant features with PCA, and clustering techniques, etc.

Well-prepared data for your model can improve its efficiency, which results in better forecast accuracy. Therefore, you should always review your datasets so that they can be optimized to produce better results.

The third step is model selection. We can choose from various models, since data scientists have developed them for different purposes. Some models are more suitable for dealing with images, video, text, or binary code. So the choice depends a lot on the developer's experience and the context of the problem. It is also conceivable to merge several models together to obtain greater accuracy in the results.

The crucial point is model formation. In the training phase, the so-called "training set" is used to teach our model to classify a certain object or predict a certain value. From a mathematical point of view, the features of the input data are accompanied by coefficients called



weights.

Training requires patience and experimentation. It is also useful to know the field in which the model will be implemented. For example, if a ML model is to be used to rank a student (passed or not) based on their grades, knowledge of how the university sector operates (the distribution of grades) would speed up the learning process as during iterations it is possible to make more plausible assumptions.

Once the model has been trained, we move on to the testing phase to understand how well it works. In this regard, the validation set helps us, which aims to submit to the system of "situations" that are part of the domain used in the learning phase but that were not part of it.

If the validation phase returns the desired results, we proceed to the hyperparameter tuning phase, i.e. we try to improve the results obtained by appropriately modifying the parameters that regulate the trained model. One of these is to revisit the training phase and using the sampling technique, that is "scrambling" the training and validation data several times in order to avoid overfitting. All this is always done for better performance. Another way to do this is to refine the input values supplied to the model. There are other parameters we could "play" with to refine the model, but the process is more intuitive than logical, so there is no real vadevecum for this operation, but a lot is up to the developer's experience.

The final step of the ML is the prediction. This is the stage where the built model is put to the test. Therefore, it is sufficient to simply input new data to the system, often called testing sets, in the same form in which the previous datasets (strings, numeric, or other) were previously submitted. At this stage, it is often realized that the model can perform better than the human ones.

## 2.2 Deep Learning

Deep learning is that research field of machine learning that is based on different levels of representation, corresponding to hierarchies where high-level concepts are defined on the basis of low-level ones. In other words, according to the definition of the Artificial Intelligence Observatory of the Polytechnic of Milan, by deep learning we mean *"a set of techniques based on artificial neural networks organized in different layers, where each layer calculates the values for the next one so that the information is elaborated in an ever more complete way"*.

Deep learning architectures include deep neural networks, convolution of deep neural networks, deep belief networks, and recursive neural networks, which have been applied in computer vision, automatic speech recognition, natural language processing, in audio recognition and bioinformatics, among other fields.

### 2.2.1 A "bit" of History

The first studies on multilayer neural networks were produced by the Japanese scientist Kunihiko Fukushima who, with the model of the cognitron first, and of the neo-cognitron then, introduced the idea of connection area for neurons that developed in convolutional neural networks.

The study of multilayer artificial neural networks developed as early as the 1980s, but only in the last decade is their usefulness in a wide range of sectors and applications being demonstrated. More in detail, the recent success of deep learning is due to the overcoming of some obstacles that in the past prevented the achievement of the expected results, such as the lack of data or adequate computational capacity. Indeed, today the available data has increased, parallel computing systems based on GPUs have been developed, and, above all, the training methods of neural networks have been optimized, which today can find solutions to problems that in the past prevented researchers from achieving satisfactory results.

Today, deep learning systems, among other utilities, allow you to:

- identify objects in images and video;*
- transcribe speech into text;*
- identify and interpret the interests of online users, showing the most relevant results for their research;*

Thanks to these and other solutions, deep learning is experiencing

years of rapid progress, even reaching, in many cases, exceeding the performance of human beings.

### **2.2.2 Fundamentals of Artificial Neural Networks**

Artificial Neural Networks are mathematical models that emulate the structure and functioning of human and other neural networks. Their purpose is, precisely, to imitate the activities considered more "intelligent" of the human brain: recognition of objects, faces, structures, understanding of language, etc. In recent years, applications have also been developed to support complex operations that humans carry out, such as multidimensional scaling or data mining. The fundamental element of an ANN is the artificial neuron: an "object" that is typically represented as a "point" having many arrows in the input but only one output. Each input has a weight which is the strength of the input signal. The neuron turns on according to the weighted sum of the inputs and its activation function, which serves to define an operating threshold.

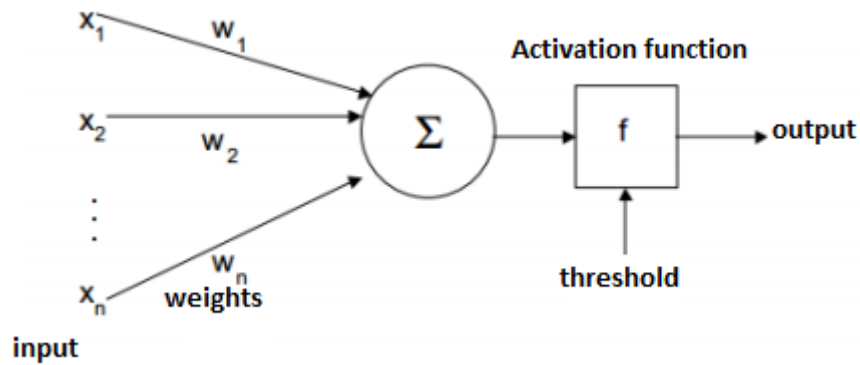


Figure 2.2: Neurons structure

The most famous activation functions are: linear, step, sigmoid, piecewise linear, or Gaussian. They make the input signal pass or not according to the result obtained by the function itself.

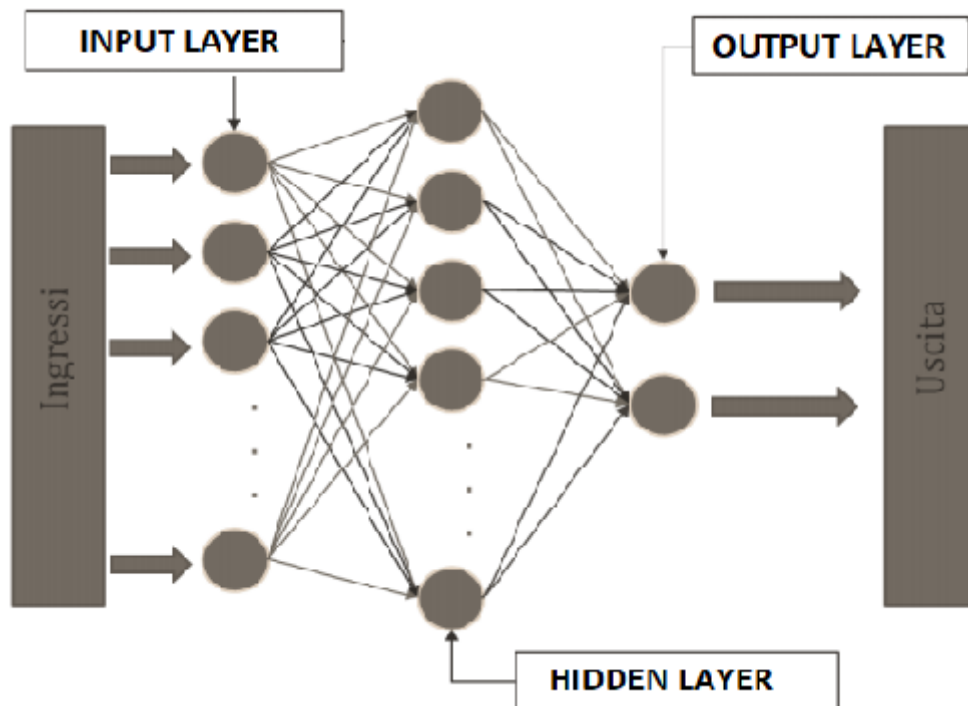


Figure 2.3: Neural Network structure

Mathematically, the simplest ANNs are "trained" with an algorithm that compares the inputs with the outputs: the difference be-

tween the (weighted) input and output values is calculated and the difference (error) is used to recalculate all weights of the input. This procedure is repeated to minimize the error.

### 2.2.3 Fundamentals of Deep Neural Networks

A Deep Neural Network (DNN) [9] is a type of ANN that has many internal levels that stand between the input and output levels. Although there are many types of deep neural networks, they are always composed of the same components: neurons, synapses, weights, biases and functions. The most recognized quality of a DNN is that, unlike simple ANNs, they are able to model complex nonlinear relationships. DNN architectures generate compositional models in which the object is expressed as a layered composition of primitives. Hidden layers allow the composition of elements from lower levels, modeling complex data with fewer units than a surface network with similar performance.

The performance of DNNs is strictly dependent on the datasets to which they are subjected, so a comparison is possible only if the same data is used.

Deep networks are typically feedforward networks in which data flows from the input level to the output level without feedback. Initially, the learning process creates a map of neurons and assigns random numerical values, the so-called "weights". The weights and inputs are multiplied and return an output between 0 and 1.

## 2.2.4 Types of Deep Neural Networks

The types of DNN are:

- Autoencoders
- Convolutional Neural Networks (CNN)
- Recursive Neural Networks (RNN)
- Deep Belief Net (DBN)

Autoencoders [8] are composed by encoders and decoders. The main difference with others types of DNN is the number of nodes, indeed they need to have the same number of nodes as in the input layer. The goal of autoencoders is to produce a useful representation of the input data. Furthermore, they must be able to output a representation as close as possible to the input. data.

CNN's are ANN's in which a less preprocessing phase is required than other networks, as they are able to automatically extract (if properly programmed) the features of the input data. A CNN is typically used for image analysis and processing. This network is able to identify an image using many superimposed layers, which first recognize the edges, then simpler shapes, and then more complex ones, up to representing the entire image at the last layer. This mechanism was inspired by that of living beings. The main operator of this type of network is the kernel (or convolution matrix), which is used to extract the features of the input image.

An RNN is an ANN made up of neurons that are joined in loops, which is why, unlike traditional neural networks, they have input and output layers correlated with each other. Each RNN can be seen as a form of short-term memory: the information collected in the activation vector of the intermediate layer is processed in the next layer and, at the same time, sent back to the origin. This creates an internal state of internal memory to process inputs of variable length over time, and for this reason these networks are often used to process multimedia files (audio and video).

DBNs are ANNs composed of binary latent variables and contain both undirected and directed layers. They are very similar to RBM (Restricted Boltzmann Machines) but, unlike these, the nodes do not communicate laterally within their layer. Moreover, the connections in the lower levels are direct. Greedy algorithms are used to pretrain DBN networks, which are fast and efficient. They start from the bottom layer and move up, tuning the weights. Learning occurs on a layer-by-layer basis, which means that layers of DBNs are trained one at a time. Thus, each level also takes in a different version of the data and each level uses the output of the previous level as input. DBN networks are generally used in audio and video recognition, and in motion acquisition data.



## 2.2.5 Quality Evaluation of Deep Neural Models

The metrics for the quality evaluation of a deep neural model derive from the generic ones of machine learning. We can list:

- Accuracy
- Error rate
- Confusion matrix
- ROC plot
- Sensitivity
- Specificity
- Precision

Accuracy is a statistic that indicates the number of correct predictions that the classification system outputs. Technically, it is calculated as the number of "true positive" and "true negative" divided by the sum of true positive, true negative, false positive and false negative. The error rate is the accuracy dual value and tells us how much the system fails to classify the inputs. The confusion matrix is a table that shows the errors that a machine learning model has in its output. It is composed by the number of correctly classified datapoints (true positive or true negative), incorrectly classified data points (false positive or false negative).

The Receiver Operating Characteristic (ROC) curve is a graph typically used to evaluate the quality of binary classifier outputs. The true positive and false positive are placed along the two axes, so we can affirm that the ROC curve shows the relationship between true alarm (hit rate) and false alarms. Sensitivity, specificity, and precision are statistics measures are defined as follows:

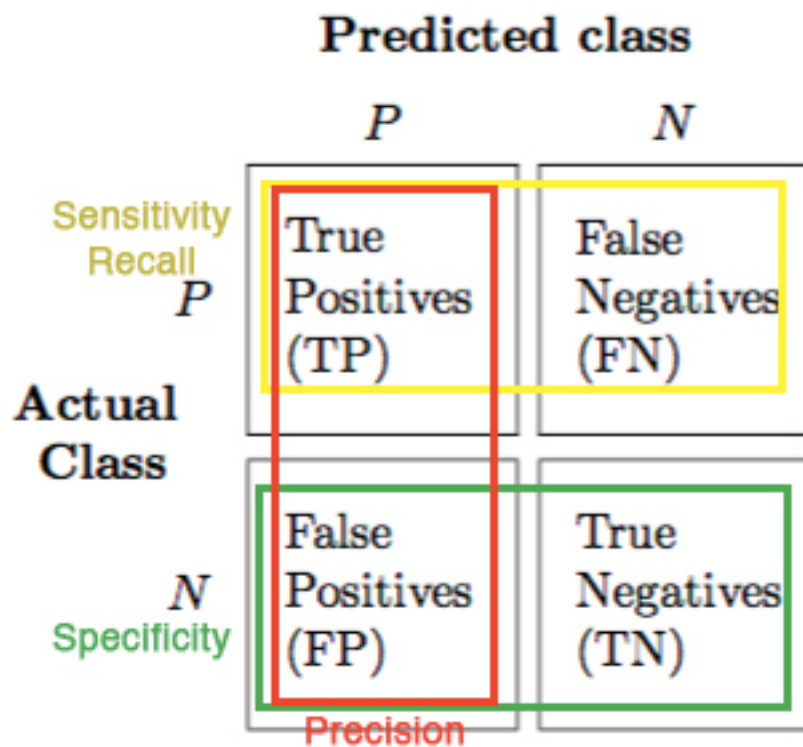


Figure 2.4: Metrics

## 2.3 Multimodal Deep Learning

Deep learning has specialized in learning from a single modality for many years, the future is represented by networks able to learn features from multiple modality inputs. Multimodal learning consists in

obtaining information relating to multiple areas, multiple sources, in fact in this work we will consider the audio, video, and text modes. The need to exploit multiple modalities arises from the ability of human beings to simultaneously consider and analyze signals of various kinds, and the goal of deep networks is precisely to emulate this attitude. These networks can be exploited to create socially aware artificial systems capable of grasping, elaborating, and expressing complex concepts such as, in this case, emotions. To study human communicative behavior, therefore, its low-level constructs must be captured and analyzed. We will see how this phase will be implemented in chapters 4 and 5.

## **Multimodal Classic models**

Among the most famous multimodal models in DL we find the Bayesian networks. A Bayesian network is a probabilistic graphical model that represents a set of stochastic variables. Specifically, a Bayesian network could represent the probabilistic relationship existing between symptoms and diseases. Given the symptoms, the network can be used to calculate the probability of the presence of different diseases. Bayesian networks that model sequences of variables that vary over time are called dynamic Bayesian networks. DBNs were developed by Paul Dagum in the early 1990s at Stanford University's medical informatics section. Dagum developed DBN to unify and extend traditional

linear state space models such as Kalman filters, linear and normal prediction models such as ARMA, and simple dependency models such as Markov models hidden in a general probabilistic representation and mechanism of inference for arbitrary nonlinear and non-normal time-dependent domains.

To better understand multimodal models, we first need to introduce a key concept: discriminative classifiers. For example, talking about animal classification, a discriminative classifier can learn what features in the input are most useful to distinguish between the various possible classes. For instance, if given images of elephants and giraphs, and all the giraphs images have a long neck, the discriminative models will learn that having a long neck means the image is of giraph. Main discriminative models are: linear classifier, logistic regression, conditional random fields (CRFs), decision trees, and many others. However, the model that best suits our need in multimodal learning is Boltzman machine. Deep Boltzmann machines are interesting for several reasons. First, DBMs have the potential of learning internal representations that become increasingly complex, which is considered a promising way of solving object and speech recognition problems. Second, high-level representations can be built from a large supply of unlabeled sensory inputs, and very limited labeled data can then be used to only slightly fine-tune the model for a specific task at hand.

The key concept used by multimodal models is representation learn-

ing. It is a set of tools that can automatically discover the representations for feature detection or classification from raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task. Representation learning is motivated by the fact that machine learning tasks such as classification often require input that is mathematically and computationally convenient to process. However, real-world data like images, video, and text are not easily processable without a feature extraction. An alternative is to discover such features or representations through examination, without relying on explicit algorithms.

Autoencoders are single layer networks which encode an input into an alternate representation at the hidden layer and at the output decode back to a reconstruction of the input. The parameters between the encoder and decoder layers are often tied and are learned through backpropagation of the gradient of the reconstruction error between the output and input. An extension of autoencoders are deep autoencoders which perform the same task through multiple hidden layers / representations. Deep autoencoders are typically initialised through greedy layer-wise training using RBMs where deeper layers are trained on the posterior of the previous layer. This allows that the model is close to a solution when beginning a finetune training stage reducing the training time and avoiding poor solutions typically experienced in training deep networks from random initialisation

### 2.3.1 Multimodal Learning Representations

In this section, we are going to illustrate multimodal learning representations. It is an alternative to a joint multimodal learning, indeed, instead of projecting the modalities together into a joint space, we learn separate representations for each modality but coordinate them through a constraint. Similarity models minimize the distance between modalities in the coordinated space. Neural networks have become a popular way to construct coordinated representations, due to their ability to learn representations. Their advantage consists in the fact that they can jointly learn coordinated representations in an end-to-end manner. Structured coordinated space models enforce additional constraints between the modality representations. The first coordinated approach to multimodal representations is embedding. Embedding is the mapping of discrete (category) variables to continuous digital vectors. In a neural network, the embedding is a low-dimensional discrete variable, a continuous vector representation of learning. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space. Neural network embedding has three main purposes:

- Find the nearest neighbor in the embedding space. These can be used to make suggestions based on user interests or cluster categories.

- The input to the machine learning model of the supervised task.
- Used to visualize the relationship between concepts and categories.

This means that as far as the book project is concerned, using neural network embedding, we can get all 37,000 book articles on Wikipedia and use only 50 numbers in the vector to represent each article.

Another useful multimodal representation tool is vector space arithmetic. A vector is an ordered array of single numbers and is an example of a first-order tensor. A vector is a fragment of an object called a vector space. It can be described as a complete set of all possible vectors of a certain length (or dimension). In deep learning, vectors usually represent feature vectors, and their original components define the correlation of specific features. Such elements may include the intensity of a set of pixels in a two-dimensional image or the relative importance of the historical price values of a cross-section of a financial instrument. The more general entities of tensors encapsulate scalars, vectors, and matrices. In physical science and machine learning, sometimes it is necessary to use tensors with an order of more than two.

### 2.3.2 Architectures, Techniques, Strategies

In this subsection we are going to illustrate some multimodal techniques. One of the most used technique in this field is the kernel func-

tion. It operates in a high-dimensional, implicit feature space without calculating the coordinates of the data in the space, but simply calculating the internal image of the image. This operation is usually computationally cheaper than the explicit calculation of coordinates. This particular method is called "kernel trick". Kernel functions for sequence data, graphics, text, images, and vectors have been introduced as well. Another tool for multimodal analysis are graphical models. Generally, a probabilistic graphical model uses a graph-based representation as the basis for encoding a distribution in a multidimensional space, and a graph is a compact or factored representation of a set of independence maintained in a particular distribution. Two branches of the graphical representation of the distribution are commonly used, namely, Bayesian networks and Markov random fields. Both families include the properties of factorization and independence, but they differ in the codable set of independence and the factorization of the distributions they cause. However, the success of machine learning algorithms usually depends on the data representation [13]. We assume that this is because different representations can entangle or hide or hide different explanatory factors behind the data. Although domain-specific knowledge can be used to help design representations, learning with general priors can also be used. The pursuit of AI is inspiring the design of more powerful representation learning algorithms to achieve such priors. The last two strategies we will see are: machine trans-



lation and co-learning. At the basic level, MT [12] will mechanically replace words in one language with words in another language in one language, but this language alone rarely produces good translation results because it needs to recognize the entire phrase and its target. The closest phrase in the language. Not all words in one language have equivalent words in another language, and many words have more than one meaning. Using corpus statistics and neurotechnology to solve this problem is a rapidly developing field, which has led to better translations, handling differences in language typology, idiom translation, and abnormal isolation. Finally, we have co-learning. Co-learning is a popular semi-supervised learning framework. In addition to a small number of labeled sets, it also uses a large amount of unlabeled data. The collaborative training method uses the predicted labels with the unlabeled data and selects samples according to the prediction confidence to enhance training. However, the sample selection in the existing co-training method is based on a predetermined strategy, which ignores the sampling deviation between the unlabeled subset and the labeled subset, so the data space cannot be explored.

# Chapter 3

## State of Art

### 3.1 Introduction

Emotion recognition (ER) and analysis has been extensively researched in in many scientific subjects. The use of automatic emotion recognition has a great potential in many intelligent systems, such as digital ads, online gaming, customer's care, and healthcare. For example, in an online gaming provided with a ER system, the player can have more excitement, and the gaming display can be adjusted according to the emotion. In an online shopping system, the company can immediately get emotional feedback from the customers, and consequently can present a new deal to them. In healthcare, patients can be monitored, and appropriate medicine or therapy can be prescribed.

The main goal of this chapter is presenting a state-of-the-art overview of studies that have been carried out in the domain of multimedial

based emotion detection. To achieve in-depth and scientifically structured evaluation, we study the underlying structure of several approaches in the literature.

Similar to traditional sentiment analysis, one of the most basic tasks in multimodal sentiment analysis is sentiment classification, which divides different sentiments into positive, negative, or neutral categories. The complexity of analyzing text, audio, and visual features to perform such tasks requires the application of different fusion techniques, such as feature level, decision level, and hybrid fusion. The performance of these fusion technologies and the classification algorithms applied are affected by the types of text, audio, and visual functions used in the analysis.

## 3.2 Comparison table of ER systems

Existing methods will be rigorously assessed against the proposed scoreboard. In particular, the results of the various systems are discussed and compared. Such detailed analysis results highlight the current gaps in the scientific literature in the field of automatic emotion detection. Multimodal sentiment analysis is a new field of traditional sentiment analysis, which goes beyond text analysis and includes other modalities, such as audio and video data.

It has been immediately realized indeed that nowadays there are many emotion recognition systems, and what we are going to do is

focus on the best ones, study them in depth, and extrapolate the best insights to be inserted into our solution, later shown in the final chapters.

NAME	DATASET	ARCHITECTURE	TECHNOLOGIES	RESULTS	METRICS	YEAR
CSDN challenge (based on Kaggle)	Kaggle fer2013 dataset	CNN	Numpy Cv2 tensorflow	70%	accuracy	2019
EEG Classifier	DEAP dataset	Adaboost, SVM Random Forest ANN	Scikit-learn Tensorflow PyEGEG,numpy	83,3%	accuracy	2017
Xception	-CK+ -Kaggle fer2013 dataset	miniXception AlexnNet	Opencv, Dlib Numpy, keras, tensorflow	99,87% CK+ 66% Kaggle	accuracy	2018
EmoTxt	MSR18	SVM	Math, numpy Scikit-learn Scipy, pattern	79%	precision	2019
Emotion recognition (Narendren Saravanan)	Pierre Luc Carrier Aaron Courville dataset	CNN	Tensorflow Keras, Numpy,Sklearn, pandas,opencv	66,36%	accuracy	2018

Figure 3.1: Emotion classifiers comparative table

In the figure above, five emotion recognition systems have been compared. They have been selected randomly among the most referenced ones. As the main selection criterion, it was decided to compare only systems that use Python as a programming language and that have been developed in the last 3 years. The results of these systems are discussed and contrasted below. What immediately is noticed is the use of technology: the one present in every system is Tensorflow, an open source platform developed by Google, that is indispensable for building complex machine learning systems using its simple APIs,

which allow you to simplify network programming to the nth level. As for the other libraries used, we see that the most frequent ones are Numpy and Opencv, which are, respectively, used for numerical matrix calculations and to build the visual infrastructure underlying the neural network. Furthermore, if we exclude the results of EmoTxt on CK + (probably plagued with overfitting), we discover that the average accuracy of the systems considered is 73%.

# Chapter 4

## Proposed Model

### 4.1 Methodology

The used methodology in this work derives from the previous analysis of the state of the art. In fact the proposed model is a syntesis of the most used architectures in the literature. In particular, we have used a convolutional neural network (CNN) with dense layers. For the audio subsystem, we only considered the MFCCs (Mel-frequency cepstral coefficients) as the unique feature to train the model on. The analogy for the video subsystem is the frame pixels. The audio clips have been divided into frames with a fixed window size (3secs), and the Fourier Transformation is applied to them in order to get the features. For each audio file, 40 features have been extracted to create the MFCCs using keras backend. The 40 values represent the numerical form of 2s length. So these files are given as input to the network three convolutional

layer, batch normalization, dropout, and dense layer.

## 4.2 Problem setting

In this work, we are interested in building a multimodal emotion detection system. In particular, we realized that humans often show their emotions in different ways, and *not always their facial expressions fit their words*. So we choose to build two different subsystems for audio and video emotion detection, usable separately, and then apply to their single prediction a postprocessing phase that allows the user to have a broader view of what the person in the video is saying, both in their voice and facial expressions.

## 4.3 Technological framework

In this work, Tensorflow has been used as main framework to build most of the system. It is Google's open source AI framework for machine learning and high performance numerical computation. TensorFlow is also a Python library that invokes C++ to construct and execute dataflow graphs. It supports many classification and regression algorithms, and more generally, deep learning and neural networks. Furthermore, we used Keras [2], that is an advanced neural network API that can run on top of Tensorflow. It can be quickly experimented with high-level, user-friendly, modular and extensible API. Keras can

also run on CPU and GPU. Keras includes the implementation of many commonly used neural network building blocks, such as layers, goals, activation functions, optimizers, and many tools that make it easier to process image and text data, thereby simplifying the coding required to write deep neural network code.

## 4.4 Proposed Solution

To give an overview of the multimodal model, we can look at the following scheme:

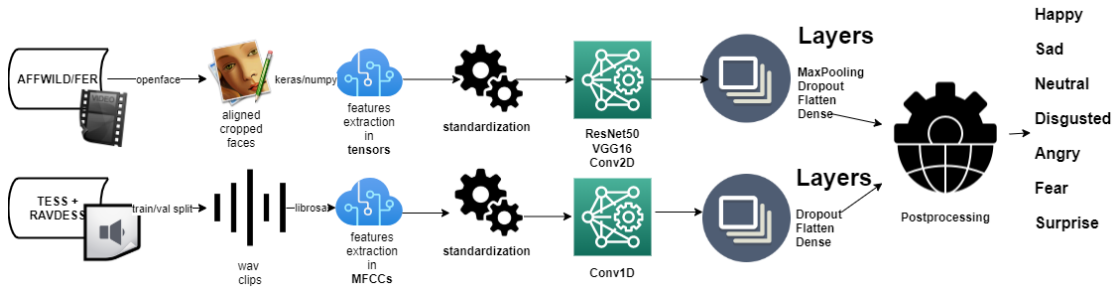


Figure 4.1: Multimodal architecture

It has been entirely built in Python on Ubuntu 18.04 LTS and it can be considered as a first approach to a combined emotion detection model in audio and video. We will focus on the single network in the next paragraphs. The main feature of this architecture is that each network can work independently and together, so they can be easily upgraded or fully replaced in the future. The thing that distinguishes it from the other architectures analyzed in the previous chapter is that



it can be run with a single command that takes in the input video and put its frames to the video subsystem and the audio chunks to the audio one.

## 4.5 Hyperparameters selection and optimization

As we all know every ML model has hyperparameters. Hyperparameters are selection points or configuration points that allow machine learning models to be customized for specific tasks or data sets. The machine learning model also has parameters, which are internal coefficients set by training or optimizing the model on the training data set. The optimization process involves defining a search space. It can be regarded as an  $n$ -dimensional volume geometrically, where each hyperparameter represents a different dimension, and the ratio of the dimensions is the value that the hyperparameter may take, such as a real value, an integer value or a categorical value. The goal of the optimization process is to find a vector that makes the learned model have the best performance, such as maximum accuracy or minimum error. In this work we have been used scikit-learn libraries to carry out this step. They provide techniques to adjust model hyperparameters. Specifically, they provide `RandomizedSearchCV` function for random search and `GridSearchCV` for grid search. Both techniques use cross-

validation to evaluate the model for a given hyperparameter vector. Both classes require two parameters. The first is the model you are optimizing. This is an example of a model with hyperparameter settings to be optimized. The second is the search space. In particular, we use three Python files to select hyperparameters:

- one for epochs and batch size selection,
- one for activation function selection,
- one for optimizer selection

The automatic selection was only made on Fer dataset, while for Affwild dataset everything was selected manually for two reasons: first one is that we already start with 96% accuracy, so no need to improve a lot; the second one is that Affwild datasets have more than 70 thousands images to be loaded every time, so the GridSearch would have take days to perform automatic selection. In the following screens we look at the main results of the automatic hyperparameters selection over Fer dataset using the custom CNN and ResNet50.

```
0.5663729310035706 {'batch_size': 100, 'epochs': 50}  
0.28666846752166747 {'batch_size': 10, 'epochs': 5}  
0.29391332864761355 {'batch_size': 10, 'epochs': 10}  
0.3023772776126862 {'batch_size': 10, 'epochs': 50}  
0.4666832506656647 {'batch_size': 20, 'epochs': 5}  
0.5069489657878876 {'batch_size': 20, 'epochs': 10}  
0.48319592475891116 {'batch_size': 20, 'epochs': 50}  
0.5021423399448395 {'batch_size': 50, 'epochs': 5}  
0.5426523327827454 {'batch_size': 50, 'epochs': 10}  
0.5589890241622925 {'batch_size': 50, 'epochs': 50}  
0.5072979688644409 {'batch_size': 100, 'epochs': 5}  
0.5466932892799378 {'batch_size': 100, 'epochs': 10}  
0.5663729310035706 {'batch_size': 100, 'epochs': 50}
```

Figure 4.2: Epochs selection on Conv2D

```
0.5566544890403747 {'activation': 'tanh'}  
0.5512562751770019 {'activation': 'softmax'}  
0.5544603705406189 {'activation': 'softplus'}  
0.5459618330001831 {'activation': 'softsign'}  
0.5525445342063904 {'activation': 'relu'}  
0.5566544890403747 {'activation': 'tanh'}  
0.4894308507442474 {'activation': 'sigmoid'}  
0.5539730191230774 {'activation': 'hard_sigmoid'}  
0.5516741275787354 {'activation': 'linear'}
```

Figure 4.3: Activation functions selection on Conv2D

```
0.5516736745834351 {'optimizer': 'RMSprop'}  
0.5507683396339417 {'optimizer': 'SGD'}  
0.5516736745834351 {'optimizer': 'RMSprop'}  
0.5502807974815369 {'optimizer': 'Adagrad'}  
0.5436626076698303 {'optimizer': 'Adadelat'}  
0.5488177299499511 {'optimizer': 'Adam'}  
0.5459961891174316 {'optimizer': 'Adamax'}  
0.5478074789047241 {'optimizer': 'Nadam'}
```

Figure 4.4: Optimizer selection on Conv2D

```
0.44999831914901733 {'optimizer': 'Nadam'}  
0.439792662858963 {'optimizer': 'SGD'}  
0.44595823884010316 {'optimizer': 'RMSprop'}  
0.4442513346672058 {'optimizer': 'Adagrad'}  
0.4421260833740234 {'optimizer': 'Adadelta'}  
0.4429273843765259 {'optimizer': 'Adam'}  
0.4404196798801422 {'optimizer': 'Adamax'}  
0.44999831914901733 {'optimizer': 'Nadam'}
```

Figure 4.5: Optimizer selection on Resnet

```
0.44515679478645326 {'batch_size': 64, 'epochs': 30}  
0.39492911100387573 {'batch_size': 64, 'epochs': 10}  
0.42199326157569883 {'batch_size': 64, 'epochs': 20}  
0.44515679478645326 {'batch_size': 64, 'epochs': 30}  
0.3722531974315643 {'batch_size': 128, 'epochs': 10}  
0.39994436502456665 {'batch_size': 128, 'epochs': 20}  
0.4070501565933228 {'batch_size': 128, 'epochs': 30}
```

Figure 4.6: Activation functions selection on Resnet

## 4.6 Training model

The training model built for the audio system model contains some features of a previous work [6]. It is a 1D CNN with a ReLu activation function, dropout of 20% and a max-pooling function 2x2. Later others dropout and flatten layers have been applied, and finally the fully connected dense layer with a softmax activation function for the emotions probabilities estimation. We can take a look at the cascade architecture in the following diagram.

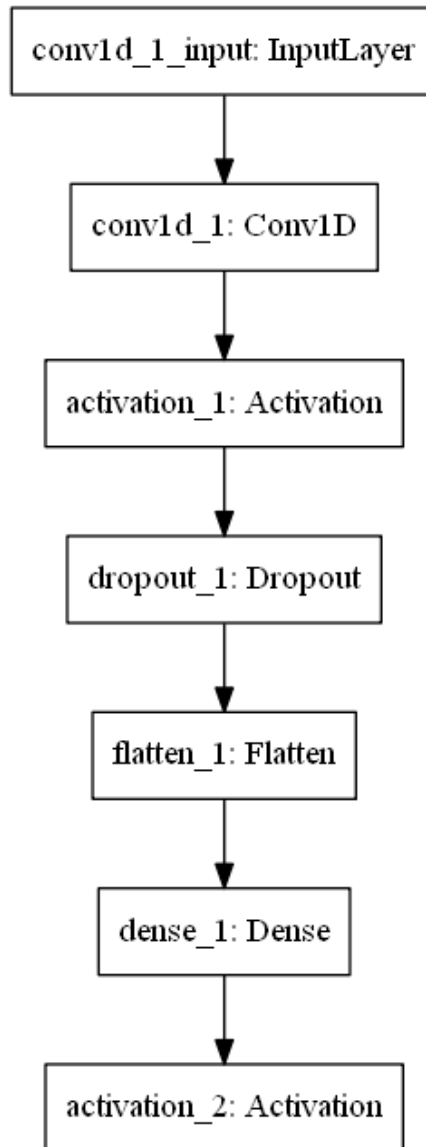


Figure 4.7: Audio model

For the video system model, we used a 2D-CNN with a ReLu activation function, dropout of 50% and a max-pooling function 2x2. Later others dropout and flatten layers have been applied, and finally the fully connected dense layer with a softmax activation function for the emotions probabilities estimation. We can take a look at the cascade architecture in the following diagram.

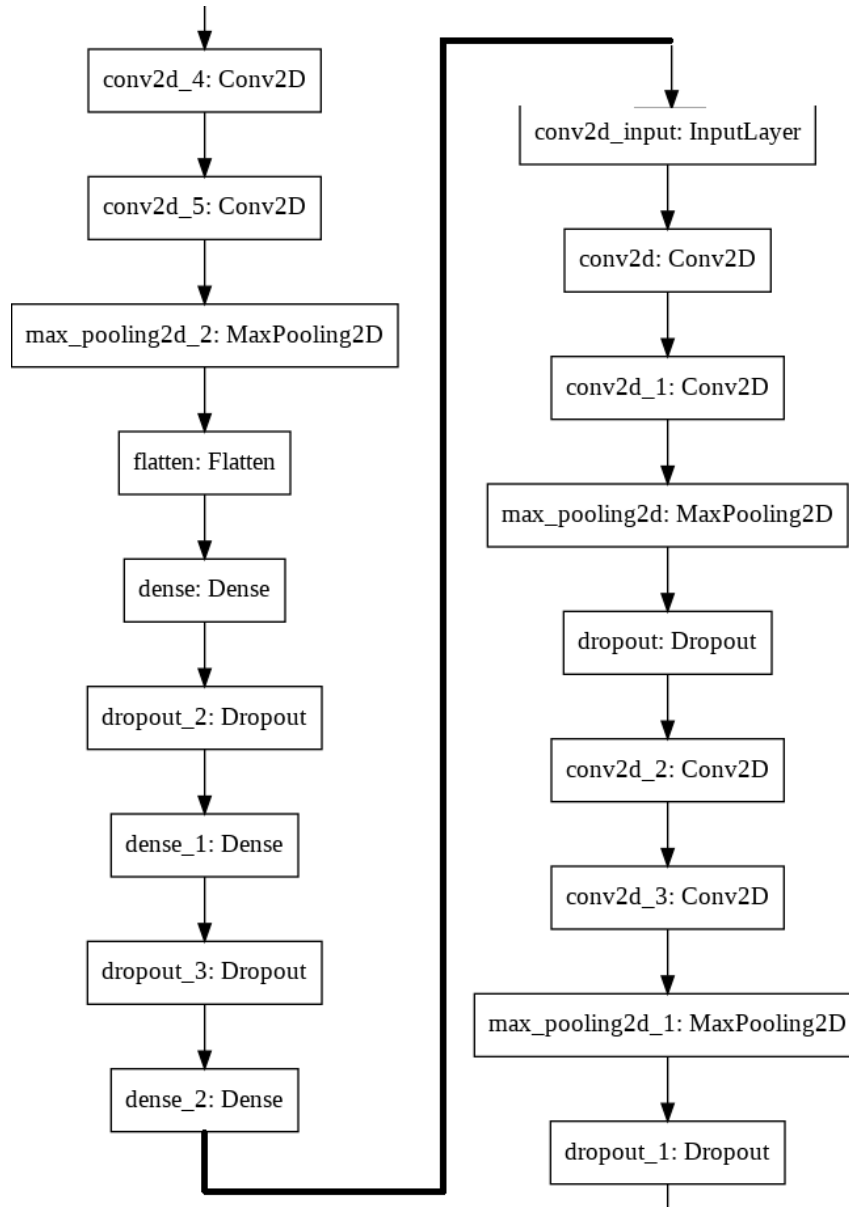


Figure 4.8: Custom 2D-CNN

## 4.7 Data preparation and preprocessing

In the video system we processed the Affwild [1] dataset as follows. First, all videos have been converted into avi format so that OpenFace toolkit could process them. So we used Openface to detect faces

into every single video in the training set. Face frames are extracted and cropped from every single video, they have also been aligned and distributed into 7 folders, one for every emotion. Each folder has the same frame number, in order to avoid any kind of bias in the model. After that we checked the integrity of the frames metadata, cleaned them from empty or corrupted one and shuffled so that any kind of bias could be removed apriori. As regard Fer dataset, it is already ready to be loaded into the model in csv format. The dataset presents three columns: the first one is for the emotion's labelling, so it is an integer from 0 to 6; the second one is a pixel array of each frame; the third one is a label that tells if the frame needs to be considered as a training, validation or test set. Regarding the audio system, we used two public datasets named RAVDESS[5] and TESS[4]. RAVDESS contains 1440 speech files and 1012 song files, it includes recordings of 24 professional actors (12 female, 12 male), vocalizing two lexically matched statements in a neutral North American accent. From TESS dataset we have 2800 files in total. Two actresses were recruited from the Toronto area. Both actresses speak English as their first language, are university educated, and have musical training. Audiometric testing indicated that both actresses have thresholds within the normal range. Therefore we merged the two databases into one and extracted features (MFCCs) from it in the joblib format. For each subsystem, we provide a separate feature extractor. Later, they are inputted to

the CNNs to train the model.

## 4.8 Validation model

The validation step has been carried out using the fit built-in function by keras that implements the holdout validation. We used a separate validation set to evaluate the model's performance during the training, and it is 15% of the entire dataset for Affwild and Fer, while for RAVDESS/TESS dataset in the audio system the validation set has been automatically created by keras fit function, with a 33% split ratio. In the following screens we can take a look at the results, the best ones have been achieved on Affwild with a validation accuracy always higher than 90%.

2247/2247	[=====]	-	596s	265ms/step	-	loss: 0.1879	-	accuracy: 0.9362	-	val_loss: 0.2571	-	val_accuracy: 0.9146
Epoch 10/30												
2247/2247	[=====]	-	595s	265ms/step	-	loss: 0.1914	-	accuracy: 0.9349	-	val_loss: 0.1647	-	val_accuracy: 0.9449
Epoch 11/30												
2247/2247	[=====]	-	595s	265ms/step	-	loss: 0.1700	-	accuracy: 0.9422	-	val_loss: 0.1693	-	val_accuracy: 0.9455
Epoch 12/30												
2247/2247	[=====]	-	595s	265ms/step	-	loss: 0.1702	-	accuracy: 0.9460	-	val_loss: 0.3119	-	val_accuracy: 0.9023
Epoch 13/30												
2247/2247	[=====]	-	594s	264ms/step	-	loss: 0.1912	-	accuracy: 0.9334	-	val_loss: 0.1995	-	val_accuracy: 0.9342
Epoch 14/30												
2247/2247	[=====]	-	595s	265ms/step	-	loss: 0.1406	-	accuracy: 0.9494	-	val_loss: 0.1831	-	val_accuracy: 0.9392
Epoch 15/30												
2247/2247	[=====]	-	595s	265ms/step	-	loss: 0.1487	-	accuracy: 0.9507	-	val_loss: 0.1726	-	val_accuracy: 0.9460
Epoch 16/30												
2247/2247	[=====]	-	594s	265ms/step	-	loss: 0.1277	-	accuracy: 0.9573	-	val_loss: 0.1670	-	val_accuracy: 0.9472
Epoch 17/30												
2247/2247	[=====]	-	595s	265ms/step	-	loss: 0.1376	-	accuracy: 0.9533	-	val_loss: 0.5395	-	val_accuracy: 0.9492
Epoch 18/30												
2247/2247	[=====]	-	594s	264ms/step	-	loss: 0.2300	-	accuracy: 0.9284	-	val_loss: 0.1883	-	val_accuracy: 0.9452
Epoch 19/30												
2247/2247	[=====]	-	595s	265ms/step	-	loss: 0.1138	-	accuracy: 0.9615	-	val_loss: 0.1522	-	val_accuracy: 0.9527
Epoch 20/30												
2247/2247	[=====]	-	594s	264ms/step	-	loss: 0.1277	-	accuracy: 0.9579	-	val_loss: 0.2311	-	val_accuracy: 0.9534
Epoch 21/30												
2247/2247	[=====]	-	594s	264ms/step	-	loss: 0.1050	-	accuracy: 0.9650	-	val_loss: 0.1496	-	val_accuracy: 0.9547
Epoch 22/30												
2247/2247	[=====]	-	594s	264ms/step	-	loss: 0.5527	-	accuracy: 0.8252	-	val_loss: 0.6870	-	val_accuracy: 0.7785
Epoch 23/30												
2247/2247	[=====]	-	594s	264ms/step	-	loss: 0.5612	-	accuracy: 0.8095	-	val_loss: 0.4201	-	val_accuracy: 0.8808
Epoch 24/30												
2247/2247	[=====]	-	594s	264ms/step	-	loss: 0.5030	-	accuracy: 0.8260	-	val_loss: 0.5691	-	val_accuracy: 0.8034

Figure 4.9: Audio validation results



```
Epoch 2/20
2247/2247 [=====] - 594s 264ms/step - loss: 0.5432 - accuracy: 0.8043 - val_loss: 0.3938 - val_accuracy: 0.8605
Epoch 3/20
2247/2247 [=====] - 594s 264ms/step - loss: 0.3798 - accuracy: 0.8636 - val_loss: 0.3033 - val_accuracy: 0.8999
Epoch 4/20
2247/2247 [=====] - 594s 265ms/step - loss: 0.2932 - accuracy: 0.8964 - val_loss: 0.2793 - val_accuracy: 0.9044
Epoch 5/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.2714 - accuracy: 0.9051 - val_loss: 0.2453 - val_accuracy: 0.9137
Epoch 6/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.2074 - accuracy: 0.9261 - val_loss: 0.2385 - val_accuracy: 0.9236
Epoch 7/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1922 - accuracy: 0.9340 - val_loss: 0.1932 - val_accuracy: 0.9409
Epoch 8/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1760 - accuracy: 0.9393 - val_loss: 0.2190 - val_accuracy: 0.9274
Epoch 9/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1612 - accuracy: 0.9440 - val_loss: 0.2725 - val_accuracy: 0.9128
Epoch 10/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.1441 - accuracy: 0.9501 - val_loss: 1.0915 - val_accuracy: 0.6750
Epoch 11/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1538 - accuracy: 0.9480 - val_loss: 0.1633 - val_accuracy: 0.9468
Epoch 12/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.1340 - accuracy: 0.9546 - val_loss: 0.1715 - val_accuracy: 0.9468
Epoch 13/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1260 - accuracy: 0.9583 - val_loss: 0.1840 - val_accuracy: 0.9422
Epoch 14/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.4535 - accuracy: 0.8596 - val_loss: 0.5680 - val_accuracy: 0.8027
Epoch 15/20
2247/2247 [=====] - 594s 264ms/step - loss: 0.4965 - accuracy: 0.8250 - val_loss: 0.3227 - val_accuracy: 0.8860
Epoch 16/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.3241 - accuracy: 0.8843 - val_loss: 0.2881 - val_accuracy: 0.9010
```

Figure 4.10: Conv2D validation results on Affwild

```
2247/2247 [=====] - 464s 188ms/step - loss: 2.4469 - accuracy: 0.5616 - val_loss: 0.7699 - val_accuracy: 0.7621
Epoch 2/15
2247/2247 [=====] - 252s 112ms/step - loss: 0.5775 - accuracy: 0.7953 - val_loss: 0.5292 - val_accuracy: 0.8253
Epoch 3/15
2247/2247 [=====] - 249s 111ms/step - loss: 0.4362 - accuracy: 0.8454 - val_loss: 0.4532 - val_accuracy: 0.8593
Epoch 4/15
2247/2247 [=====] - 248s 110ms/step - loss: 0.3606 - accuracy: 0.8737 - val_loss: 0.5349 - val_accuracy: 0.8260
Epoch 5/15
2247/2247 [=====] - 248s 111ms/step - loss: 0.3131 - accuracy: 0.8893 - val_loss: 0.3389 - val_accuracy: 0.8894
Epoch 6/15
2247/2247 [=====] - 249s 111ms/step - loss: 0.2830 - accuracy: 0.9009 - val_loss: 0.4759 - val_accuracy: 0.8449
Epoch 7/15
2247/2247 [=====] - 248s 110ms/step - loss: 0.2577 - accuracy: 0.9104 - val_loss: 0.3505 - val_accuracy: 0.8814
Epoch 8/15
2247/2247 [=====] - 248s 110ms/step - loss: 0.2632 - accuracy: 0.9087 - val_loss: 0.3831 - val_accuracy: 0.8734
Epoch 9/15
2247/2247 [=====] - 247s 110ms/step - loss: 0.2256 - accuracy: 0.9203 - val_loss: 0.3065 - val_accuracy: 0.9067
Epoch 10/15
2247/2247 [=====] - 247s 110ms/step - loss: 0.2216 - accuracy: 0.9221 - val_loss: 0.3202 - val_accuracy: 0.8991
Epoch 11/15
2247/2247 [=====] - 247s 110ms/step - loss: 0.2065 - accuracy: 0.9275 - val_loss: 0.2709 - val_accuracy: 0.9106
Epoch 12/15
2247/2247 [=====] - 248s 110ms/step - loss: 0.2084 - accuracy: 0.9279 - val_loss: 0.4307 - val_accuracy: 0.8571
Epoch 13/15
2247/2247 [=====] - 247s 110ms/step - loss: 0.1962 - accuracy: 0.9329 - val_loss: 0.7501 - val_accuracy: 0.8102
Epoch 14/15
2247/2247 [=====] - 246s 109ms/step - loss: 0.2158 - accuracy: 0.9270 - val_loss: 0.1853 - val_accuracy: 0.9412
Epoch 15/15
2247/2247 [=====] - 246s 109ms/step - loss: 0.1903 - accuracy: 0.9349 - val_loss: 0.3118 - val_accuracy: 0.9003
```

Figure 4.11: VGG16 validation results on Affwild

```

2247/2247 [=====] - 555s 247ms/step - loss: 0.0510 - accuracy: 0.9824 - val_loss: 0.0842 - val_
Epoch 24/30
2247/2247 [=====] - 555s 247ms/step - loss: 0.0455 - accuracy: 0.9839 - val_loss: 0.0784 - val_
Epoch 25/30
2247/2247 [=====] - 555s 247ms/step - loss: 0.0445 - accuracy: 0.9846 - val_loss: 0.0979 - val_
Epoch 26/30
2247/2247 [=====] - 554s 246ms/step - loss: 0.0455 - accuracy: 0.9842 - val_loss: 0.0840 - val_
Epoch 27/30
2247/2247 [=====] - 552s 245ms/step - loss: 0.0437 - accuracy: 0.9848 - val_loss: 0.0800 - val_
Epoch 28/30
2247/2247 [=====] - 551s 245ms/step - loss: 0.0455 - accuracy: 0.9843 - val_loss: 0.0800 - val_
Epoch 29/30
2247/2247 [=====] - 552s 246ms/step - loss: 0.0426 - accuracy: 0.9850 - val_loss: 0.0819 - val_
Epoch 30/30
2247/2247 [=====] - 552s 245ms/step - loss: 0.0434 - accuracy: 0.9848 - val_loss: 0.0838 - val_
<Figure size 640x480 with 1 Axes>
<Figure size 640x480 with 1 Axes>
482/482 [=====] - 40s 81ms/step - loss: 0.0816 - accuracy: 0.9760

Test loss: 0.08161714673042297, Test Accuracy: 0.9760358333587646
Model saved to disk

```

Figure 4.12: Resnet validation results on Affwild

## 4.9 Training and Validation Evaluation

To validate the models, we use the evaluate function by Keras. The evaluate method is one of the main functions in TensorFlow and Keras. It predicts the output for the given input and then computes the metrics function specified in the compilation phase and based on labels and corrected predictions, it returns the computed metric value as the output. As regards the video subsystem, we can take a look at the results in the following screens, for each subsystem and for each configuration.

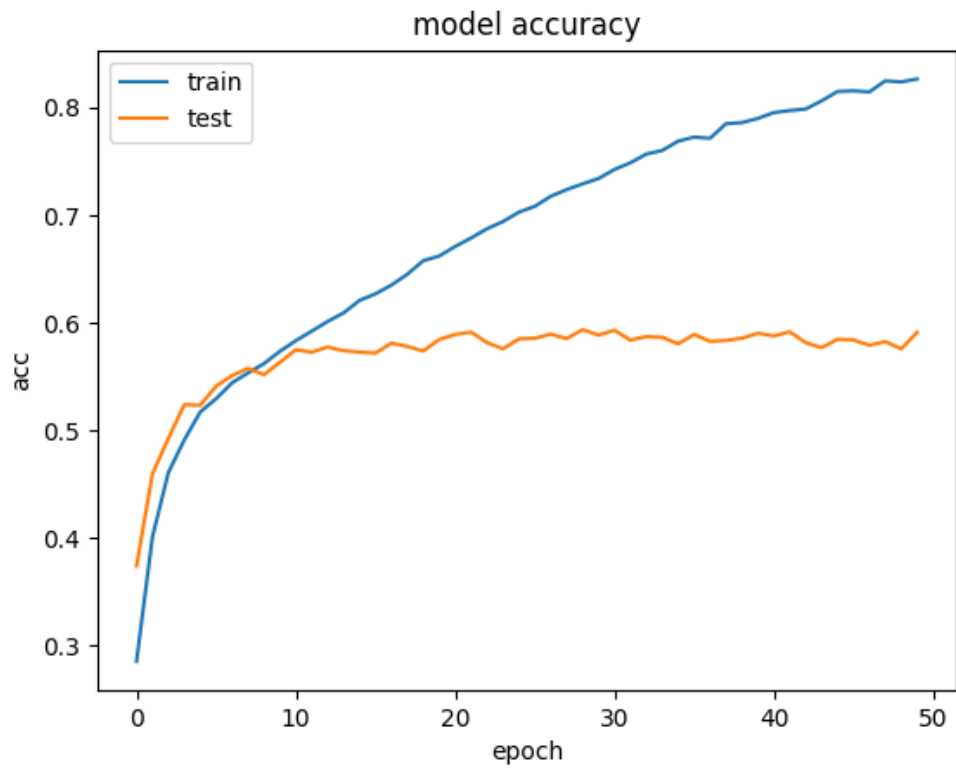


Figure 4.13: Accuracy function of Conv2D on Fer

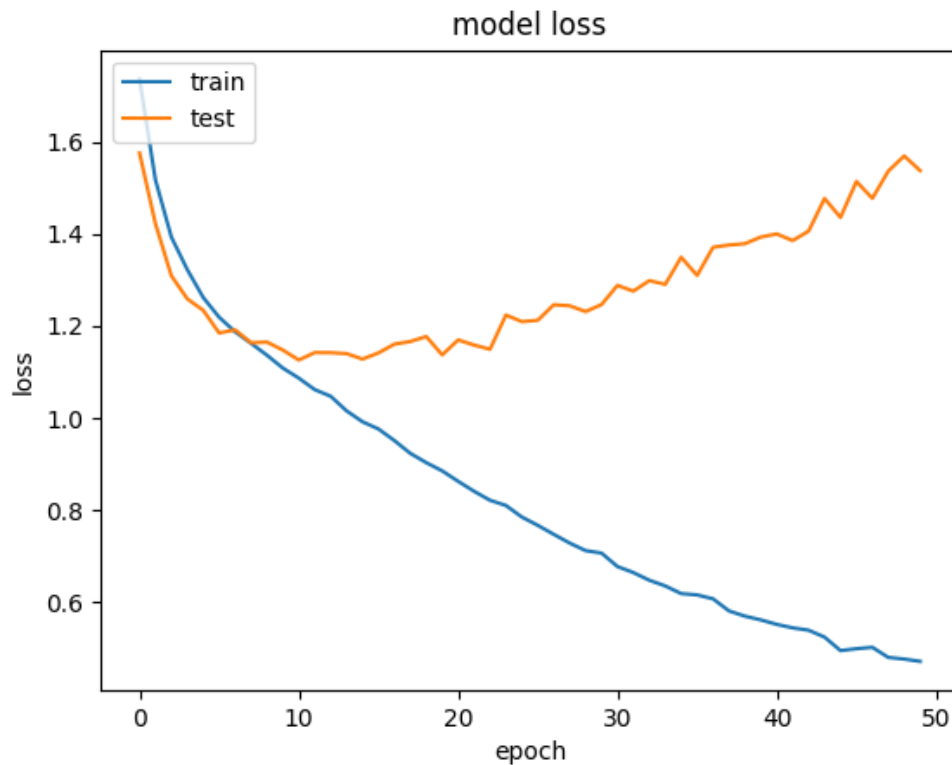


Figure 4.14: Loss function of Conv2D on Fer

What we can say about these evaluation graphics is that the training and the validation accuracy start to diverge around 10 epochs. This means that after 10 epochs the model overfits to training data, indeed it performs better on that data. So we realized that a good parameter for epochs would have been 10 on this network. Dual reasoning can be done for the loss.

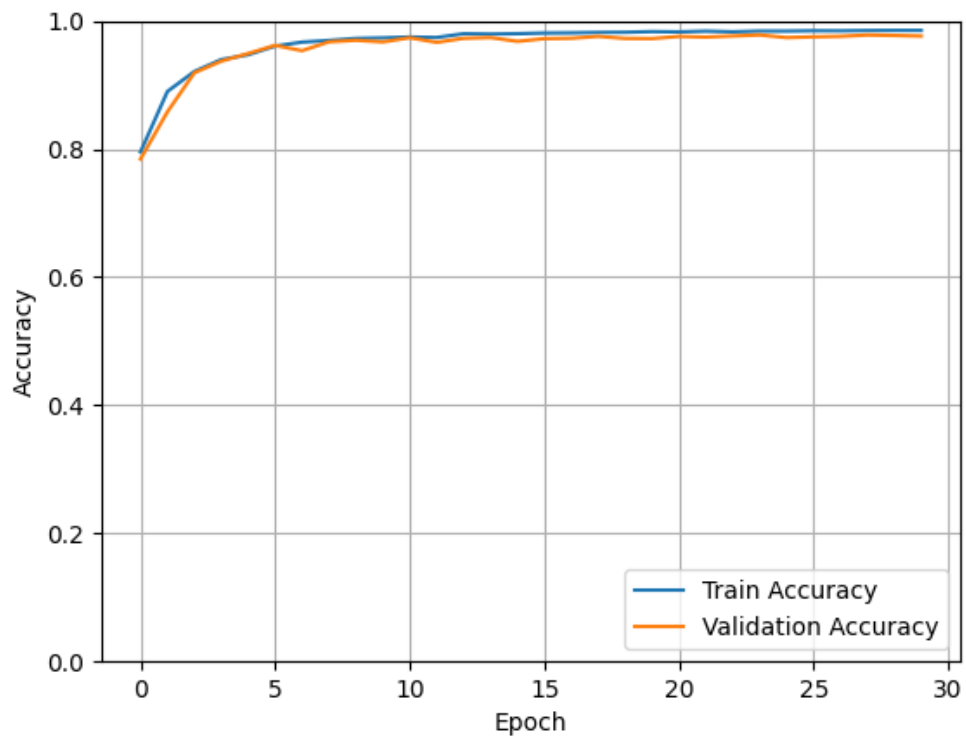


Figure 4.15: Accuracy function of ResNet50 on Affwild

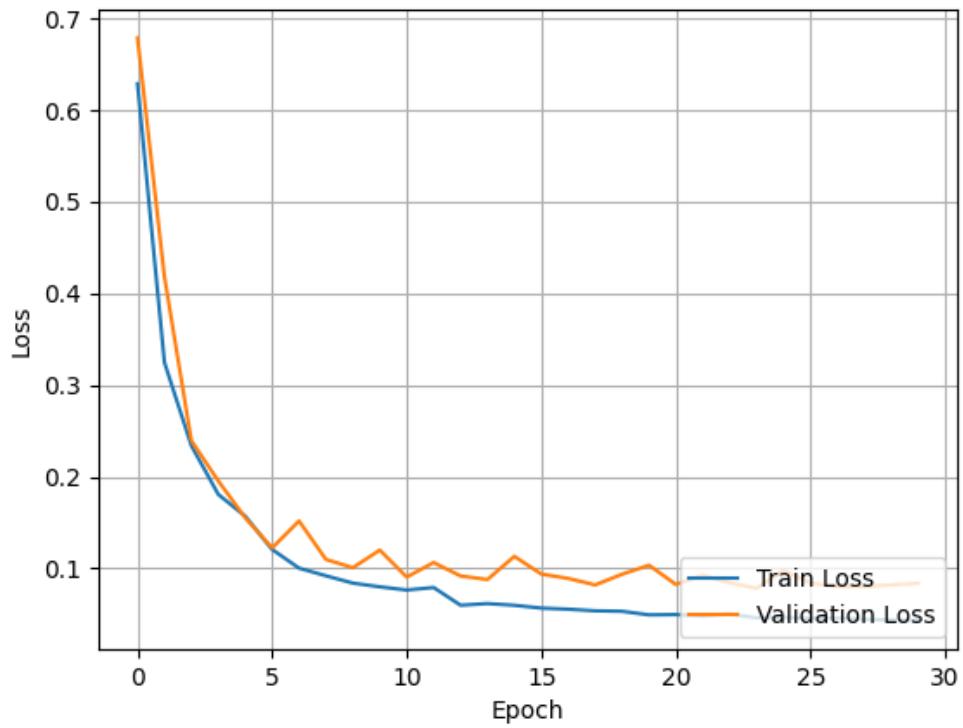


Figure 4.16: Loss function of ResNet50 on Affwild

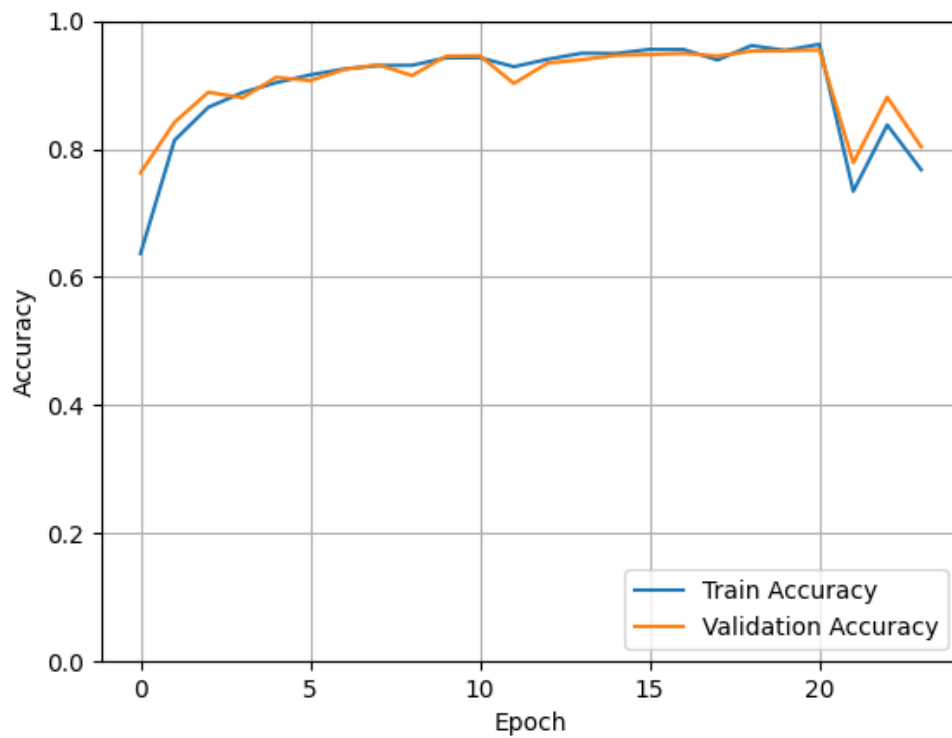


Figure 4.17: Accuracy function of VGG16 on Affwild

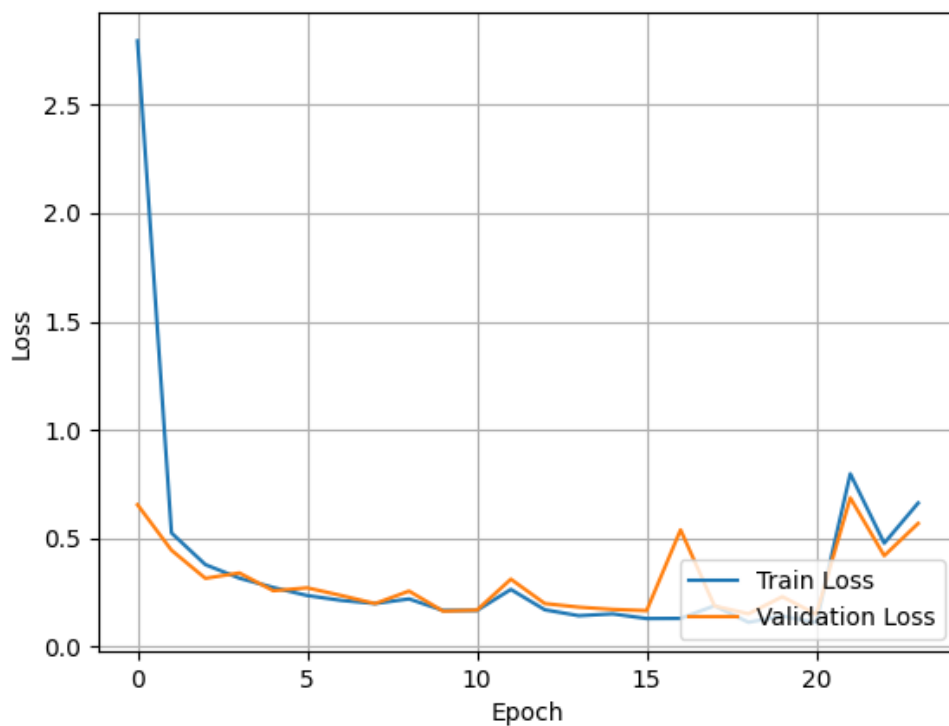
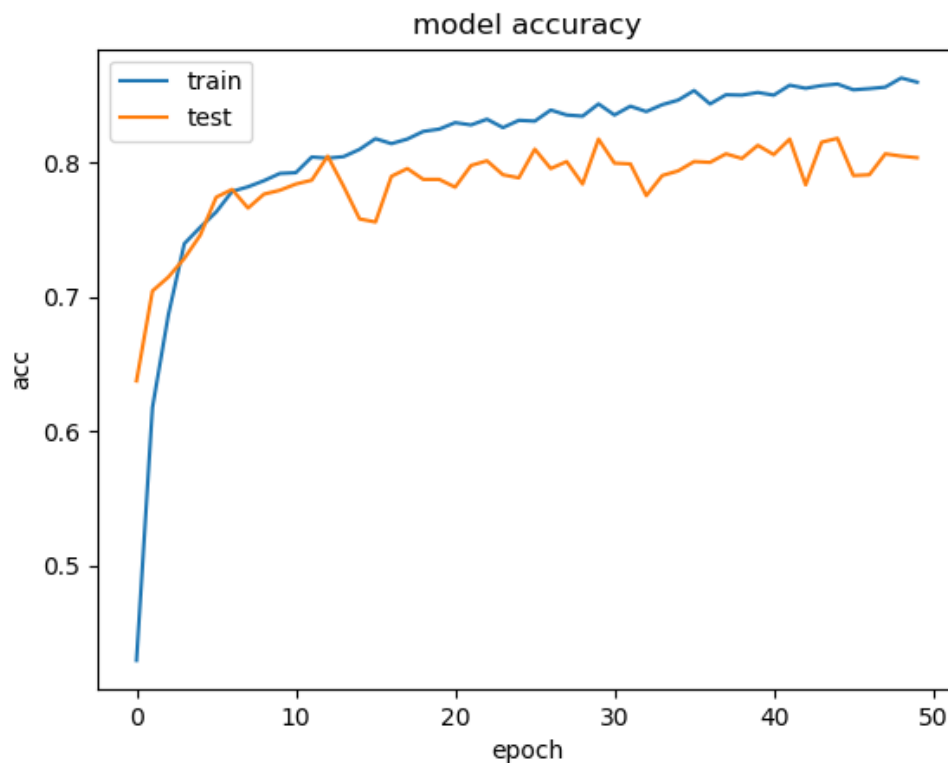
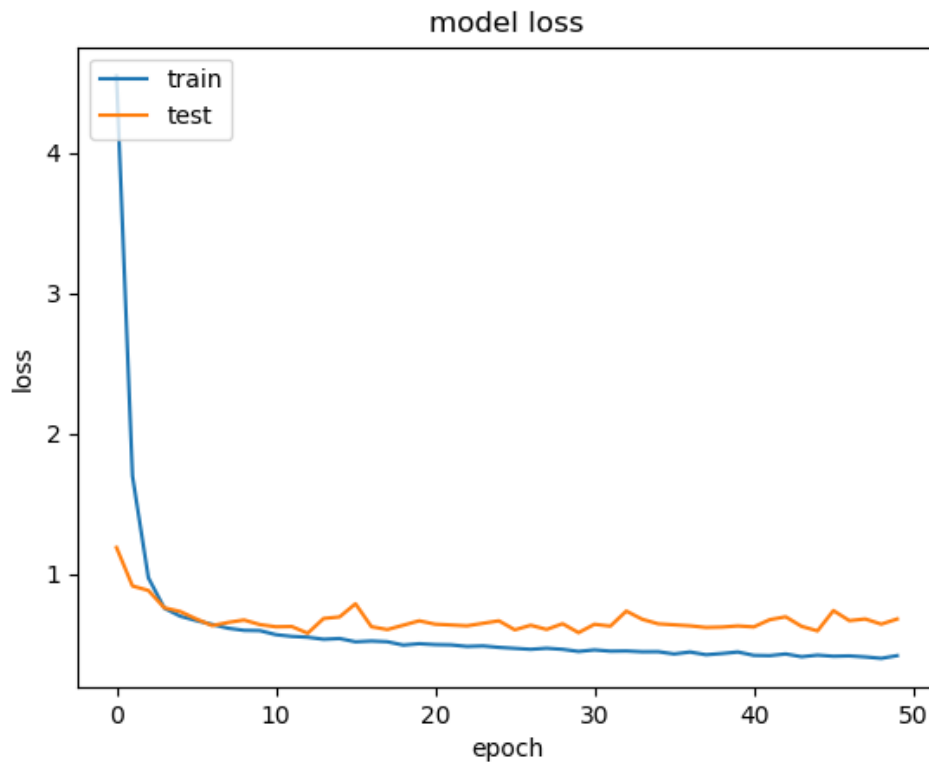


Figure 4.18: Loss function of VGG16 on Affwild

Similar evaluation can be done for ResNet50 and VGG16 graphics, where we see that the training and the validation accuracy start to diverge around 30 epochs for the first one and at 20 for the second one. This means that after these values the models overfit to training data, indeed they perform better on that data. So we realized that a good parameter for epochs would have been 30 for ResNet50 and 20 for VGG16. Dual reasoning can be done for the loss. As regards the audio subsystem, to evaluate the model, we need to interpret the loss and the accuracy history during the training phase. We can do it looking at the following graphics.







What we can say about these graphs is that the training and the validation accuracy start to diverge around 40 epochs. This means that after 40 epochs the model overfits to training data, indeed it performs better on that data if you use more than 40 epochs. So we realized that a good parameter for epochs would have been 30. Dual reasoning can be done for the loss.

## Chapter 5

# Testing, Experiments and Result Analysis

### 5.1 Testing model and evaluation

The testing was made for each subsystem using the evaluate function by Keras. As regard the video part, we divided the dataset into training, validation, and test set with the following percentages: 60% - 20% - 20%. As regard the audio part, the ratios were: 70% - 15% - 15%. We chose the accuracy as the main metric for both systems.

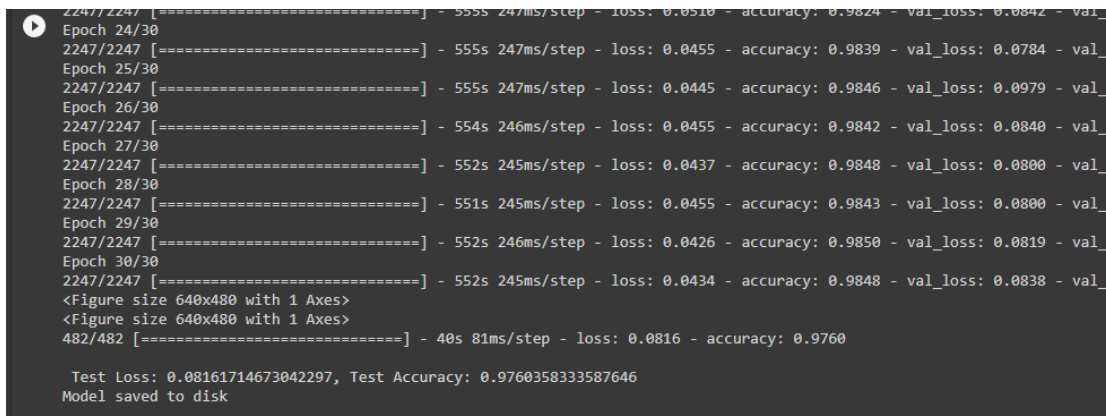
### 5.2 Set of Experiments

As regard the video subsystem, experiments were done training all networks on two different datasets, Fer2013 and Affwild and the results

of accuracy were:

- 97% using ResNet50 on Affwild
- 46% using ResNet50 on Fer
- 90% using VGG16 on Affwild
- 53% using Conv2D on Fer
- 80% using VGG16 on Fer
- 90% using Conv2D on Affwild

In the following screen we can take a look at the test results of all configurations.



```
2247/2247 [=====] - 555s 247ms/step - loss: 0.0510 - accuracy: 0.9824 - val_loss: 0.0842 - val_
Epoch 24/30
2247/2247 [=====] - 555s 247ms/step - loss: 0.0455 - accuracy: 0.9839 - val_loss: 0.0784 - val_
Epoch 25/30
2247/2247 [=====] - 555s 247ms/step - loss: 0.0445 - accuracy: 0.9846 - val_loss: 0.0979 - val_
Epoch 26/30
2247/2247 [=====] - 554s 246ms/step - loss: 0.0455 - accuracy: 0.9842 - val_loss: 0.0840 - val_
Epoch 27/30
2247/2247 [=====] - 552s 245ms/step - loss: 0.0437 - accuracy: 0.9848 - val_loss: 0.0800 - val_
Epoch 28/30
2247/2247 [=====] - 551s 245ms/step - loss: 0.0455 - accuracy: 0.9843 - val_loss: 0.0800 - val_
Epoch 29/30
2247/2247 [=====] - 552s 246ms/step - loss: 0.0426 - accuracy: 0.9850 - val_loss: 0.0819 - val_
Epoch 30/30
2247/2247 [=====] - 552s 245ms/step - loss: 0.0434 - accuracy: 0.9848 - val_loss: 0.0838 - val_
<Figure size 640x480 with 1 Axes>
<Figure size 640x480 with 1 Axes>
482/482 [=====] - 40s 81ms/step - loss: 0.0816 - accuracy: 0.9760

Test Loss: 0.08161714673042297, Test Accuracy: 0.9760358333587646
Model saved to disk
```

Figure 5.1: ResNe50 on Affwild

```
539/539 [=====] - 64s 119ms/step - loss: 0.2569 - accuracy: 0.9098 - val_loss: 2.5870 - val_accuracy: 0.4311
Epoch 18/30
539/539 [=====] - 64s 119ms/step - loss: 0.2247 - accuracy: 0.9227 - val_loss: 2.6257 - val_accuracy: 0.4514
Epoch 19/30
539/539 [=====] - 64s 119ms/step - loss: 0.2186 - accuracy: 0.9228 - val_loss: 2.6976 - val_accuracy: 0.4307
Epoch 20/30
539/539 [=====] - 64s 119ms/step - loss: 0.1754 - accuracy: 0.9392 - val_loss: 2.6737 - val_accuracy: 0.4509
Epoch 21/30
539/539 [=====] - 64s 119ms/step - loss: 0.2017 - accuracy: 0.9336 - val_loss: 2.7109 - val_accuracy: 0.4225
Epoch 22/30
539/539 [=====] - 64s 119ms/step - loss: 0.1754 - accuracy: 0.9393 - val_loss: 2.8956 - val_accuracy: 0.4363
Epoch 23/30
539/539 [=====] - 64s 119ms/step - loss: 0.1814 - accuracy: 0.9368 - val_loss: 2.8505 - val_accuracy: 0.4350
Epoch 24/30
539/539 [=====] - 64s 120ms/step - loss: 0.1538 - accuracy: 0.9485 - val_loss: 2.8457 - val_accuracy: 0.4269
Epoch 25/30
539/539 [=====] - 64s 119ms/step - loss: 0.1555 - accuracy: 0.9502 - val_loss: 2.8981 - val_accuracy: 0.4516
Epoch 26/30
539/539 [=====] - 64s 119ms/step - loss: 0.1358 - accuracy: 0.9532 - val_loss: 2.8080 - val_accuracy: 0.4478
Epoch 27/30
539/539 [=====] - 64s 119ms/step - loss: 0.1467 - accuracy: 0.9508 - val_loss: 2.9598 - val_accuracy: 0.4108
Epoch 28/30
539/539 [=====] - 64s 119ms/step - loss: 0.1392 - accuracy: 0.9522 - val_loss: 3.1126 - val_accuracy: 0.4586
Epoch 29/30
539/539 [=====] - 64s 119ms/step - loss: 0.1449 - accuracy: 0.9478 - val_loss: 2.7210 - val_accuracy: 0.4433
Epoch 30/30
539/539 [=====] - 64s 119ms/step - loss: 0.1230 - accuracy: 0.9586 - val_loss: 2.9796 - val_accuracy: 0.4530
113/113 [=====] - 3s 30ms/step - loss: 2.9914 - accuracy: 0.4620
Elapsed time (s): 3.4014155864715576
Test loss: 2.9913523197174072
Test accuracy: 0.4619671106338501
113/113 [=====] - 3s 27ms/step - loss: 2.9914 - accuracy: 0.4620
```

Figure 5.2: ResNet50 on Fer

```
TFM - 23.100.8.198:3389 - Connessione Desktop remoto
File Edit View Search Terminal Help
Epoch 2/20
2247/2247 [=====] - 594s 264ms/step - loss: 0.5432 - accuracy: 0.8043 - val_loss: 0.3938 - val_accuracy: 0.8605
Epoch 3/20
2247/2247 [=====] - 594s 264ms/step - loss: 0.3798 - accuracy: 0.8636 - val_loss: 0.3033 - val_accuracy: 0.8999
Epoch 4/20
2247/2247 [=====] - 594s 265ms/step - loss: 0.2932 - accuracy: 0.8964 - val_loss: 0.2793 - val_accuracy: 0.9044
Epoch 5/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.2714 - accuracy: 0.9051 - val_loss: 0.2453 - val_accuracy: 0.9137
Epoch 6/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.2074 - accuracy: 0.9261 - val_loss: 0.2385 - val_accuracy: 0.9236
Epoch 7/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1922 - accuracy: 0.9340 - val_loss: 0.1932 - val_accuracy: 0.9409
Epoch 8/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1760 - accuracy: 0.9393 - val_loss: 0.2190 - val_accuracy: 0.9274
Epoch 9/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1612 - accuracy: 0.9440 - val_loss: 0.2725 - val_accuracy: 0.9128
Epoch 10/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.1441 - accuracy: 0.9501 - val_loss: 1.0915 - val_accuracy: 0.6750
Epoch 11/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1538 - accuracy: 0.9480 - val_loss: 0.1633 - val_accuracy: 0.9468
Epoch 12/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.1340 - accuracy: 0.9546 - val_loss: 0.1715 - val_accuracy: 0.9468
Epoch 13/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.1260 - accuracy: 0.9583 - val_loss: 0.1840 - val_accuracy: 0.9422
Epoch 14/20
2247/2247 [=====] - 596s 265ms/step - loss: 0.4535 - accuracy: 0.8596 - val_loss: 0.5600 - val_accuracy: 0.8027
Epoch 15/20
2247/2247 [=====] - 594s 264ms/step - loss: 0.4965 - accuracy: 0.8250 - val_loss: 0.3227 - val_accuracy: 0.8860
Epoch 16/20
2247/2247 [=====] - 595s 265ms/step - loss: 0.3241 - accuracy: 0.8843 - val_loss: 0.2881 - val_accuracy: 0.9010
482/482 [=====] - 38s 79ms/step - loss: 0.2765 - accuracy: 0.9006
Test Loss: 0.2765204906463623, Test Accuracy: 0.9005715250968933
Model saved to disk
root@TFM: /home/username/TFM - AGUIDataDetection/emotion_classifier/xxprint/sffu1de
```

Figure 5.3: VGG16 on Affwild

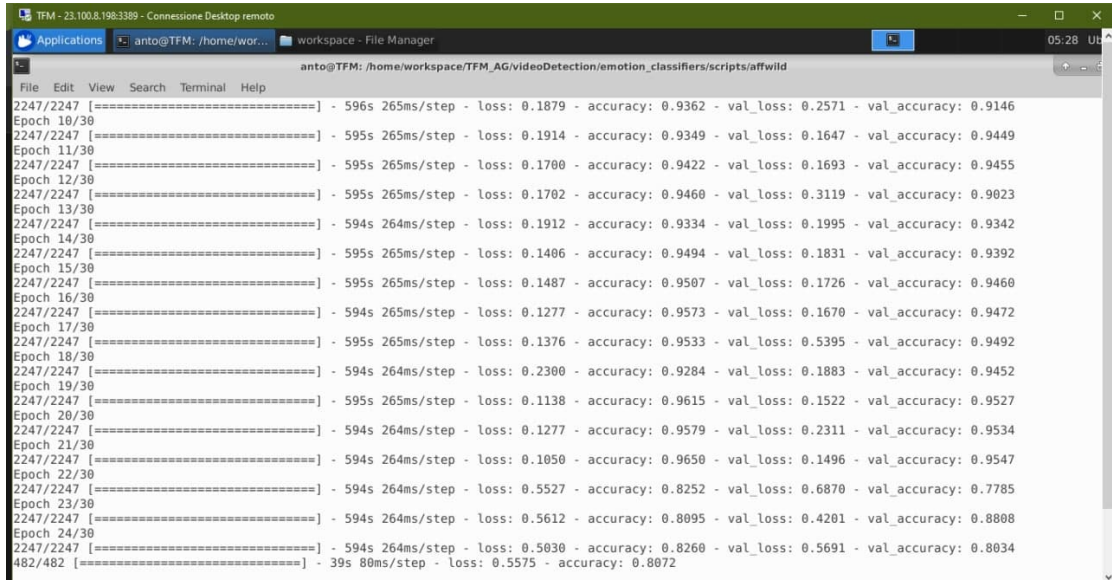


Figure 5.4: VGG16 on Fer

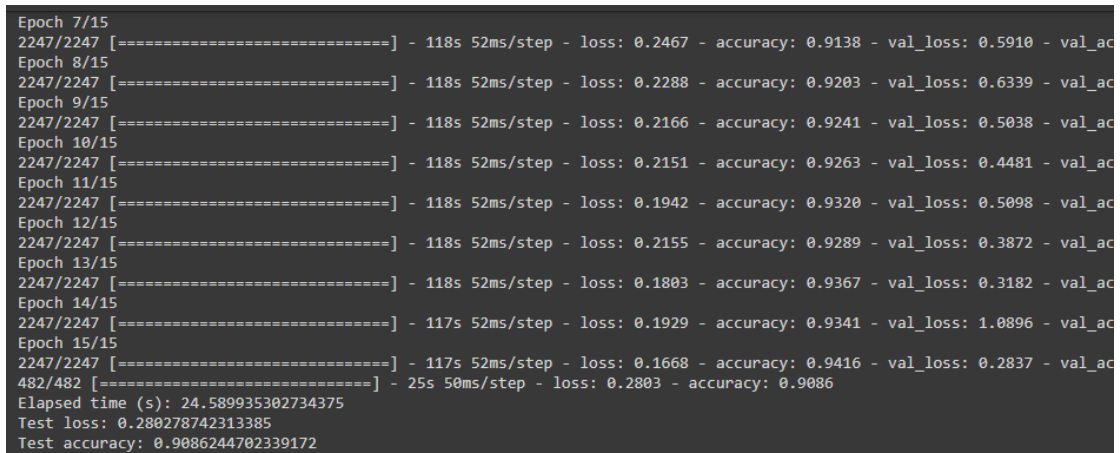


Figure 5.5: Conv2D on Affwild

```
Epoch 22/30
270/270 [=====] - 5s 20ms/step - loss: 0.9231 - accuracy: 0.6522 - val_loss: 1.2704 - val_accuracy: 0.5452
Epoch 23/30
270/270 [=====] - 5s 20ms/step - loss: 0.8927 - accuracy: 0.6571 - val_loss: 1.2698 - val_accuracy: 0.5546
Epoch 24/30
270/270 [=====] - 5s 20ms/step - loss: 0.8661 - accuracy: 0.6698 - val_loss: 1.2914 - val_accuracy: 0.5509
Epoch 25/30
270/270 [=====] - 5s 20ms/step - loss: 0.8598 - accuracy: 0.6703 - val_loss: 1.2916 - val_accuracy: 0.5582
Epoch 26/30
270/270 [=====] - 5s 20ms/step - loss: 0.8411 - accuracy: 0.6815 - val_loss: 1.3039 - val_accuracy: 0.5521
Epoch 27/30
270/270 [=====] - 5s 20ms/step - loss: 0.8203 - accuracy: 0.6874 - val_loss: 1.3011 - val_accuracy: 0.5623
Epoch 28/30
270/270 [=====] - 5s 20ms/step - loss: 0.7896 - accuracy: 0.6965 - val_loss: 1.3375 - val_accuracy: 0.5408
Epoch 29/30
270/270 [=====] - 5s 20ms/step - loss: 0.7720 - accuracy: 0.7059 - val_loss: 1.3203 - val_accuracy: 0.5600
Epoch 30/30
270/270 [=====] - 5s 20ms/step - loss: 0.7536 - accuracy: 0.7129 - val_loss: 1.3134 - val_accuracy: 0.5582
113/113 [=====] - 1s 4ms/step - loss: 1.3753 - accuracy: 0.5414
Elapsed time (s): 0.5259091854095459
Test loss: 1.3752899169921875
Test accuracy: 0.5413764119148254
```

Figure 5.6: Conv2D on Fer

As regard the audio subsystem, we reached over 80% accuracy on both RAVDESS[5] and TESS[4] datasets.

```

0      0.78      0.93      0.85      198
1      0.65      0.84      0.73      135
2      0.88      0.72      0.79      260
3      0.90      0.70      0.79      241
4      0.82      0.87      0.85      266
5      0.82      0.80      0.81      253
6      0.80      0.83      0.81      189
7      0.78      0.80      0.79      192

accuracy
macro avg      0.80      0.81      0.80      1734
weighted avg   0.81      0.81      0.81      1734

[[185  4  2  1  2  0  3  1]
 [ 14 113  2  1  0  2  2  1]
 [  7 18 187  3 15  8  4 18]
 [ 13 19  6 168  6 19  8  2]
 [  3  0  3  0 231 12 10  7]
 [  6  6  5 11 12 203  3  7]
 [  2 12  1  2  7  1 157  7]
 [  8  3  6  1  7  4 10 153]]
55/55 [=====] - 0s 2ms/step - loss: 0.7077 - accuracy: 0.8057
Elapsed time (s): 0.19033074378967285
Test loss: 0.7077444791793823
Test accuracy: 0.8056516647338867
```

Figure 5.7: Conv1D on RAVDESS/TESS

## 5.3 Results Analysis

As the main consideration over these results, we can definitely say that the best combination as regard the video part, is ResNet50 over Affwild dataset, but also VGG16 has reached 90%. It could be predictable for two reasons: first because ResNet50 was designed for images' classification tasks; second one because we used a large balanced and shuffled dataset, so no bias was added during the training step. On the other hand, we consider as worst system the Conv2D over Fer dataset, even if it can not be considered a fail because the winner in Kaggle competition [3] reached 70% accuracy. The reason comes mainly from the fact that Fer dataset it is not shuffled, this means that consecutive frames can come from the same video, and this introduces a bias into the model. In addition we can not ignore the fact that it is a half of the Affwild. Since we have not many networks to compare with on the audio dataset, there is not much we can say on its behaviour, but we can surely affirm that the merger of two separate datasets has brought more variability into the system and for this reason made it capable of processing a wider range of voices and tonalities.

## Chapter 6

# Conclusions and Future Work

### 6.1 Conclusions

In conclusion, we have presented a combined approach to emotion detection based on two subsystems, usable separately and that can work together too. The video subsystem has been built using two datasets (Affwild[1] and Fer[3]) with a huge preprocessing, carried out using OpenFace toolkit and many Python scripts. The main steps for the dataset generation were: frame extraction, integrity checking, cleaning, balancing and shuffling. We trained three different networks for the video system: ResNet50 [10], VGG16 [7] and a custom 2D-CNN. For the audio part, we merged two different datasets (RAVDESS [5] and TESS[4]) and trained the model over 26 actor voices, with different



ages and tonalities. For both systems hyperparameters selection was applied, to choose the best optimizer, activation function and epoch number. Finally a postprocessing step allows the user to run both systems together, taking in input a single video and extracting its audio and its frames. Singularly, each system reaches a max accuracy of 97% (ResNet50 on Affwild dataset) for the video system and 82% for the audio one.

## 6.2 Future Work

As future development of this work I think it could be integrated with another subsystem that can takes into account the trascriptions of the multimodal files, to give a more accurate result to the user. Furthermore using more features obtained by OpenFace toolkit and from other datasets would be an extra point to upgrade the architecture. Finally, a fusion technique between the two networks could be applied in order to generate a single network that will be able to learn from a unique multimedial file.

# List of Figures

2.1	Deep learning concept . . . . .	8
2.2	Neurons structure . . . . .	17
2.3	Neural Network structure . . . . .	17
2.4	Metrics . . . . .	22
3.1	Emotion classifiers comparative table . . . . .	32
4.1	Multimodal architecture . . . . .	36
4.2	Epochs selection on Conv2D . . . . .	39
4.3	Activation functions selection on Conv2D . . . . .	39
4.4	Optimizer selection on Conv2D . . . . .	39
4.5	Optimizer selection on Resnet . . . . .	40
4.6	Activation functions selection on Resnet . . . . .	40
4.7	Audio model . . . . .	41
4.8	Custom 2D-CNN . . . . .	42
4.9	Audio validation results . . . . .	44
4.10	Conv2D validation results on Affwild . . . . .	45
4.11	VGG16 validation results on Affwild . . . . .	45

4.12	Resnet validation results on Affwild . . . . .	46
4.13	Accuracy function of Conv2D on Fer . . . . .	47
4.14	Loss function of Conv2D on Fer . . . . .	48
4.15	Accuracy function of ResNet50 on Affwild . . . . .	49
4.16	Loss function of ResNet50 on Affwild . . . . .	50
4.17	Accuracy function of VGG16 on Affwild . . . . .	51
4.18	Loss function of VGG16 on Affwild . . . . .	51
5.1	ResNe50 on Affwild . . . . .	55
5.2	ResNet50 on Fer . . . . .	56
5.3	VGG16 on Affwild . . . . .	56
5.4	VGG16 on Fer . . . . .	57
5.5	Conv2D on Affwild . . . . .	57
5.6	Conv2D on Fer . . . . .	58
5.7	Conv1D on RAVDESS/TESS . . . . .	58

# Bibliography

- [1] D. Kollias, S. Zafeiriou. Aff-wild2 database, 2019. <https://ibug.doc.ic.ac.uk/resources/aff-wild2/>.
- [2] Google. Keras, 2020. [keras.io](https://keras.io).
- [3] Kaggle. Fer 2013 dataset, 2013. <https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-data>.
- [4] Kate Dupuis, M. Kathleen Pichora-Fuller. Tess database, 2020. <https://tspace.library.utoronto.ca/handle/1807/24487>.
- [5] Livingstone SR, Russo FA. Ravdess database, 2018. <https://zenodo.org/record/1188976#.YB520OhKjIU>.
- [6] Marco Giuseppe de Pinto, Marco Polignano, Pasquale Lops, Giovanni Semeraro. Emotions understanding model from spoken language using deep neural networks and mel-frequency cep-

- stral coefficients, 2020. <https://ieeexplore.ieee.org/document/9122698>.
- [7] Muneeb ul Hassan. Vgg16, 2020. <https://neurohive.io/en/popular-networks/vgg16/>.
- [8] roboticsbiz. Different types of deep learning models explained, 2020. <https://roboticsbiz.com/different-types-of-deep-learning-models-explained>.
- [9] Wikipedia contributors. Deep learning — Wikipedia, the free encyclopedia, 2020. [https://en.wikipedia.org/wiki/Deep\\_learning#Deep\\_neural\\_networks](https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks).
- [10] Wikipedia contributors. Tensorflow train and evaluate, 2020. [https://en.wikipedia.org/wiki/Residual\\_neural\\_network](https://en.wikipedia.org/wiki/Residual_neural_network).
- [11] Wikipedia contributors. Unsupervised learning — Wikipedia, the free encyclopedia, 2020. [https://en.wikipedia.org/wiki/Unsupervised\\_learning](https://en.wikipedia.org/wiki/Unsupervised_learning).
- [12] Wikipedia contributors. Machine translation, 2021. [https://en.wikipedia.org/wiki/Machine\\_translation](https://en.wikipedia.org/wiki/Machine_translation).
- [13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives, 2021. <https://arxiv.org/pdf/1206.5538.pdf>.