Multimodal Emotion Recognition*

Anatoli de Bradké, Maël Fabien, Raphaël Lederman, and Stéphane Reynal

[1] Télécom ParisTech, Paris 75013, France
[2] Projet Fil Rouge 2018-2019

Abstract. In this paper, we are exploring state of the art models in multimodal emotion recognition. We have chosen to explore textual, sound and video inputs and develop an ensemble model that gathers the information from all these sources and displays it in a clear and interpretable way.

Keywords: Emotion recognition · Text · Sound · Video · Affective Computing

* A project for the French employment agency : Pole Emploi

Table of Contents

1 Context

1.1 Definitions

We are trying to provide definitions of affective computing and multimodal sentiment analysis in the context of our research. Those definitions may vary depending on the context.

Affective Computing Affective computing is a field of Machine Learning and Computer Science that studies the recognition and the processing of human affects.

Multimodal Emotion Recognition Multimodal Emotion Recognition is a relatively new discipline that aims to include text inputs, as well as sound and video. This field has been rising with the development of social networks that gave researchers access to a vast amount of data. Recent studies have been exploring potential metrics to measure the coherence between emotions from the different channels.

We are going to explore several categorical targets depending on the input considered.

Table 1 gives a summary of all the categorical targets we are evaluating depending on the data type.

Table 1: Categorical target depending on the input data type.

Data Type	Categorical target
Textual	Openness, Conscientiousness, Extraversion, Agreeableness, Neuroticism
Sound	Happy, Sad, Angry, Fearful, Surprise, Neutral and Disgust
Video	Happy, Sad, Angry, Fearful, Surprise, Neutral and Disgust

For the text inputs, we are going to focus on the so-called Big Five, widely used in personality surveys.

1.2 Research context

This research is made in the context of an exploratory analysis for the French employment agency (Pole Emploi), and is part of the Big Data program at Telecom ParisTech.

The research will explore state of the art multimodal sentiment analysis, but will also focus on compliance in the context of General Data Protection Regulation (GDPR).

The aim of this project is to provide candidates seeking for a job a platform that analyses their answers to a set of pre-defined questions, as well as the non-verbal part of a job interview through sound and video processing.

1.3 Data sources

We have chosen to diversify the data sources we used depending on the type of data considered. For the text input, we are using data that was gathered in a

study by Pennebaker and King [1999]. It consists of a total of 2,468 daily writing submissions from 34 psychology students (29 women and 5 men whose ages ranged from 18 to 67 with a mean of 26.4). The writing submissions were in the form of a course unrated assignment. For each assignment, students were expected to write a minimum of 20 minutes per day about a specific topic. The data was collected during a 2-week summer course between 1993 to 1996. Each student completed their daily writing for 10 consecutive days. Students' personality scores were assessed by answering the Big Five Inventory (BFI) [John et al., 1991]. The BFI is a 44-item self-report questionnaire that provides a score for each of the five personality traits. Each item consists of short phrases and is rated using a 5-point scale that ranges from 1 (disagree strongly) to 5 (agree strongly). An instance in the data source consists of an ID, the actual essay, and five classification labels of the Big Five personality traits. Labels were originally in the form of either yes ('y') or no ('n') to indicate scoring high or low for a given trait. It is important to note that the classification labels have been applied according to answers to a rather short self-report questionnaire : there might be a non-negligible bias in the data due to both the relative simplicity of the BFI test compared to the complexity of psychological features, and the cognitive biases preventing users from providing a perfectly accurate assessment of their own characteristics. For sound data sets,

we are using the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS). "The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) contains 7356 files (total size: 24.8 GB). The database contains 24 professional actors (12 females, 12 males), vocalizing two lexically-matched statements in a neutral North American accent. Speech includes calm, happy, sad, angry, fearful, surprise, and disgust expressions, and song contains calm, happy, sad, angry, and fearful emotions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression. All conditions are available in three modality formats: Audio-only (16bit, 48kHz .wav), Audio-Video (720p H.264, AAC 48kHz, .mp4), and Video-only (no sound)." For the video data sets, we are using the popular FER2013 Kaggle Challenge

data set. The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The data set remains quite challenging to use, since there are empty pictures, or wrongly classified images.

1.4 Methodology

Our aim is to develop a model able to provide real time sentiment analysis with a visual user interface using Tensorflow.js technology.

Therefore, we have decided to separate two types of inputs :

1. Textual input, such as answers to questions that would be asked to a person from the platform
2. Video input from a live webcam or stored from an MP4 or WAV file, from which we split the audio and the images.

2 Text mining for personality trait classification

2.1 Theoretical foundations

2.1.1 Introduction

Emotion recognition through text is a challenging task that goes beyond conventional sentiment analysis : instead of simply detecting neutral, positive or negative feelings from text, the goal is to identify a set of emotions characterized by a higher granularity. For instance, feelings like anger or happiness could be included in the classification. As recognizing such emotions can turn out to be complex even for the human eye, machine learning algorithms are likely to obtain mixed performances. It is important to note that nowadays, emotion recognition from facial expression tends to perform better than from textual expression. Indeed, many subtleties should be taken into account in order to perform an accurate detection of human emotions through text, context-dependency being one of the most crucial. This is the reason why using advanced natural language processing is required to obtain the best performance possible. There exists

different ways to tackle natural language processing problems, the two main ones being rule-based and learning-based approaches. While rule-based approaches tend to focus on pattern-matching and are largely based on grammar and regular expressions, learning-based approaches put the emphasis on probabilistic modeling and likelihood maximization. Here, we will mainly focus on learning-based methods, and review some of the central methods, from "traditional" classifiers to more advanced neural network architectures.

In the context of our study, we chose to use text mining in order not to detect regular emotions such as disgust or surprise, but to recognize personality traits based on the "Big Five" model in psychology. Even though emotion recognition and personality traits classification are two separate fields of studies based on different theoretical underpinnings, they use similar learning-based methods and literature from both areas can be interesting. The main motivation behind this choice is to offer a broader assessment to the user : as emotions can only be understood in the light of a person's own characteristics, we thought that analyzing personality traits would provide a new key to understanding emotional fluctuations. Our final goal is to enrich the user experience and improve the quality of our analysis : any appropriate and complementary information deepening our understanding of the user's idiosyncrasies is welcome

Many psychology researchers (starting with D. W. Fiske [1949], then Norman [1963] and Goldberg [1981]), believe that it is possible to exhibit five categories, or core factors, that determine one's personality. The acronym OCEAN (for openness, conscientiousness, extraversion, agreeableness, and neuroticism) is often used to refer to this model. We chose to use this precise model as it is nowadays the most popular in psychology : while the five dimensions don't capture the peculiarity of everyone's personality, it is the theoretical framework most recognized by

researchers and practitioners in this field. Many linguistic-oriented tools can be used to derive a person's personality traits, for instance the individual's linguistic markers (obtained using text analysis, psycholinguistic databases and lexicons for instance). Since one of the earliest studies in this particular field [Mairesse et al., 2007], researchers have introduced multiple linguistic features and have shown correlations between them and the Big Five. These features could therefore have a non-negligible impact on classification performances, but as we stated before, we will mainly focus machine learning methods and leave out the linguistic modeling as it does not fit into the spectrum of our study. Our main goal is to leverage on the use of statistical learning methods in order to build a tool capable of recognizing the personality traits of an individual given a text containing his answers to pre-established personal questions. Our first idea was to record a user's interview and convert the file from audio to text : in this way we would have been able to work with similar data for text, audio and video. Nevertheless, the good transcription of audio files to text requires the use of expensive APIs, and the tools available for free in the market don't provide sufficient quality. This is the reason why we chose to apply our personality traits detection model to short texts directly written by users : in this way we can easily target particular themes or questions and provide indications of the language level to use. As a result of this, we can make sure that the text data we use to perform the personality traits detection is consistent with the data used for training, and therefore ensure the highest possible quality of results. In the following sections, we will go through some of the learning techniques that are commonly used in order to perform personality traits recognition. These methods are usually applied in a sequential way through a pipeline including preprocessing steps in order to standardize the data, embedding in order to represent it as a numerical vector, then the classification algorithm in order to predict labels. Let's focus on a few technical aspects.

2.1.2 Preprocessing

The preprocessing is the first step of our NLP pipeline : this is where we convert raw text document to cleaned lists of words. In order to complete this process, we first need to *tokenize* the corpus. This means that sentences are split into a list of single words, also called tokens. Other preprocessing steps include the use of regular expressions in order to delete unwanted characters or reformat words. For instance, it is common to lowercase tokens, and delete some punctuation characters that are not crucial to the understanding of the text. The removing of stopwords in order to retain only words with meaning is also an important step : it allows to get rid of words that are too common like 'a', 'the' or 'an'. Finally, there are methods available in order to replace words by their grammatical *root* : the goal of both stemming and lemmatization is to reduce derivationally related forms of a word to a common base form. Families of derivationally related words

with similar meanings, such as 'am', 'are', 'is' would then be replace by the word 'be'. Finally, in the context of word sense disambiguation, part-of-speech tagging is used in order to mark up words in a corpus as corresponding to a particular part of speech, based on both its definition and its context. This can be used to improve the accuracy of the lemmatization process, or just to have a better understanding of the *meaning* of a sentence.

2.1.3 Embedding

2.1.3.1 Bag-of-Word approaches

In order to run machine learning algorithms we need to convert the text files into

numerical feature vectors : we convert a collection of text documents to a matrix of token counts, the number of features being equal to the vocabulary size found by analyzing the data (each unique word in our dictionary corresponding to a descriptive feature). The easiest and simplest way of counting this tokens is to use raw counts (term frequencies).

$$tf_{t,d} = f_{t,d}$$

Instead of using the raw frequencies of occurrence of tokens in a given document it is possible to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. In order to do this, we can use term frequencies adjusted for document length, also called TF-IDF (or term frequency times inverse document frequency). Common words like (a, an, the, etc.) will then have a lower weight.

$$tf_{t,d} = \sum_{t' \in d} f_{t',d}$$

Other constructions of term frequency weight also exist. The logarithmically scaled frequency uses log normalization to generate the document-term matrix.

$$f_{t,d} = log(1 + f_{t',d})$$

The double-normalization K (like the time frequencies adjusted for document length) provides augmented frequencies to prevent a bias towards longer documents.

$$f_{t,d} = 0.5 + 0.5 \frac{f_{t',d}}{max_{t' \in d} f_{t',d}}$$

This choice of embedding using a document-term matrix has some disadvantages. As with most embedding methodologies, it requires to choose the

vocabulary size (which is a parameter that will greatly impact the sparsity of the document representations) and the document-term matrix can end-up being very sparse. It is this scarcity, intrinsically linked to the structure of the word representation in bag-of-words approaches, that is the biggest disadvantage of this method, both from the point of view of computational efficiency and information retrieval (as there is little information in a large representation space). Finally, there can be a loss of valuable information through discarding word order.

2.1.3.2 Word2Vec embedding

The Word2Vec embedding was first proposed by Mikolov et al. in "Efficient Estimation of Word Representations in Vector Space" (2013). It generates distributed representations by assigning a real-valued vector for each word and representing the word by the vector : we call the vector *word embedding*. The idea is to introduce dependency between words : words with similar context should occupy close spatial positions. This is very different from the document-term matrix where all words were considered independent from each others. The Word2Vec method constructs the embedding using two methods in the context of neural networks: Skip Gram and Common Bag Of Words (CBOW). Both architectures can be used in order to produce embeddings. Using one-hot encoding and considering the context of each word, the goal of the CBOW methodology is to predict the word corresponding to the context. For a single input word, for instance the word "sunny" in the sentence "What a sunny weather today!", the objective is to predict the one-hot encoding of the target word "weather" and minimize the output error between the predicted vector and the target one. The vector representation of the target word is then learned in the prediction process. More precisely, the neural network first takes the V-dimensional one-hot encoding of the input word and maps it to the hidden layer using a first weight matrix. Then, another weight matrix is used to map the hidden layer outputs to the final V-dimensional prediction vector constructed with the softmax values. It is important to note that there is no use of non-linear activation functions (tanh, sigmoid, ReLu etc.) outside of the softmax calculations in the last layer: the outputs are passed as simple weighted combination of the inputs. More precisely :

$$Output\ from\ hidden\ node\ 'j' = u_j = \sum_{i=1}^{V} w_{i,j} x_i$$

With u_j being the input to the j-th hidden node, $w_i j$ is the weight of the connection from the i-th input node to the j-th hidden node and x_i is the value of the i-th input node (in the case of word vectors, only one element of x_i is equal to 1, remaining all are 0.

The output layer has V output nodes, one for each unique word (V corresponds here to the vocabulary size). The final output from each node is the softmax.

$$Value\ at\ output\ node\ 'k' = O_k = \frac{exp(u'_k)}{\sum_{q=1}^{V} exp(u'_q)}$$

where $u'_k = \sum_{j=1}^{N} w'_{jk} h_j$

u'_k is the input to the k-th output node, w'_{jk} is the weight of the connection from j-th hidden node to the k-th output node and h_j is the output of the j-th hidden node.

The cross-entropy function is used to compute the log-loss at each output node 'k':

$$Loss\ at\ output\ node\ 'k' = -y_k \times log(O_k) - (1 - y_k) \times log(1 - O_k)$$

where $y_k = 1$ if the actual output is the word at the k-th index else $y_k = 0$. The total loss for all output nodes (for a single training instance) is given as : The cross-entropy function is used to compute the log-loss at each output node 'k':

$$E = \sum_{k=1}^{V} [-y_k \times log(O_k) - (1 - y_k) \times log(1 - O_k)]$$

The weights w'_{jk} between hidden layer and the output layer are progressively updated using stochastic gradient descent technique as follows (the second equation being obtained by solving for the partial derivative in the first equation) :

$$w_{jk}^{\prime(t+1)} = w_{jk}^{\prime(t)} - \eta \times [\frac{\partial E}{\partial w'_{jk}}] = w_{jk}^{\prime(t)} - \eta \times (O_k - y_k) \times h_j$$

where η is the learning rate In the previous equation, $(O_k y_k)$ denotes the error in prediction at the k-th node. Similarly, using back-propagation, it is possible to obtain the update equations for the input weights w_{ij}. Since only one of the inputs is active at each training iteration, we need to update the input weights w_{ij} only for the node 'i' (the one for which the input $x_i = 1$) as follows: The above update equation is applied only for the node 'i', for which the input xi=1

$$w_{jk}^{(t+1)} = w_{jk}^{(t)} - \eta \times \sum_{k=1}^{V} (O_k - y_k) \times w_{jk}^{\prime(t+1)}$$

This model can be extended to non-single context words : it is possible to use multiple input context vectors, or a combination of them (sum or mean for instance) in order to improve predictions. Indeed, if we define a context size of 2,

we will consider a maximum of 2 words on the left and 2 words on the right as the surrounding words for each target word. For the sentence "read books instead of playing video games", the context words for "playing" with context size of 2 would be : ("instead", "of", "video", "games") Using the CBOW methodology, if the input is ("instead", "of", "video", "games"), then desired output for this precise example is ("playing").

In the case where we have multiple input context words, it is assumed that the same set of weights w_ij between input nodes and hidden nodes are used. This leads to computing the output of the hidden node 'j' as the average of the weighted inputs for all the context words. Assuming we have C number of context words, the output from hidden layer 'j' is given as :

$$h_j = \frac{1}{C} \times [\sum_{q=1}^{C} w_{i_q j} x_{i_q}]$$

where the inputs x_{i_q} is the value of x_i in the q-th input layer. The update equation for the output weights w'_{jk} remains the same as above. The update equations for the input weights w_{ij} is modified to be :

$$w_{i_q j}^{(t+1)} = w_{i_q j}^{(t)} - \eta \times \frac{1}{C} \times \sum_{k=1}^{V} (O_k - y_k) \times w_{jk}^{\prime(t+1)}$$

The concept behind the Skip Gram architecture is the exact opposite of the CBOW architecture : taking a word vector as input, the objective is to predict the surrounding words (the number of context words to predict being a parameter of the model). In the context of our previous example sentence with context size of 2, if the input is ("playing") then the desired output would be ("instead", "of", "video", "games"). For the Skip Gram architecture, we therefore assume that we have multiple output layers and all output layers share the same set of output weights w'_{jk}. The update equations for the output weights are modified as follows :

$$w_{jk}^{(t+1)} = w_{jk}^{(t)} - \eta \times h_j \times \sum_{k_q} (O_{k_q} - y_{k_q})$$

where k_q denotes the k-th output node in the q-th output layer. The term $\sum_{k_q}(O_{k_q} - y_{k_q})$ represents the sum of all the errors from C different context words predictions. In this way, the model is penalized for each context word mis-prediction. The update equation for the input weights remains unchanged.

According to various experimentations, it seems that the Skip Gram architecture performs slightly better than the CBOW architecture at constructing word embeddings : this might be linked to the fact that the varying impacts of different input context vectors are averaged out in the CBOW methodology.

For words which co-occur together many times in the text corpus, the model is more likely to predict one of them at the output layer when the other one is given as the input. For example, given "violin" as the input word to the Skip Gram model, it is more likely to predict context words such as "music" or "instrument" compared to words such as "computer" or "democracy". We need to update the output weights w'_{jk} at the output layer for all words V in the vocabulary (where V can be in billions). As this is the most time consuming computation in the model, the authors of Word2Vec tried two different techniques in order to reduce inefficiencies. The first one is the Hierarchical Softmax and the second one is Negative Sampling. With Hierarchical Softmax, it is possible to reduce the complexity of the output probabilities computation for each output node, from O(V) to O(log(V)). In HS, the output layer is arranged in the form of a binary tree, with the leaves being the output nodes, thus there are V leaf nodes in the tree (and V-1 internal nodes). Therefore there are no output weights with Hierarchical Softmax, but instead we need to update weights for each internal node (each internal node having a weight associated with it). The probability of a particular output node (leaf node in this case) is given by the product of the internal node probabilities on the path from the root to this leaf node. There is no need to normalize this probabilities as the sum of all the leaf node probabilities sum to 1.

Let n(k, j) for an internal node denote that it is the j-th node from the root to the word at index k in the vocabulary, along the path in the tree, then the output probabilities are given as :

$$O_k = \prod_{j=1}^{L(k)-1} F(k,j)$$

$$where\ F(k,j) = \begin{cases} \sigma(v(k,j)), & \text{if n(k,j+1) is the left child of n(k,j)} \\ \sigma(-v(k,j)), & \text{otherwise} \end{cases}$$

where $v(k,j) = \sum_{i=1}^{N} w'_{i,n(k,j)} h_i$ and $L(k)$ denotes the length of the path from the root of the tree to the word at index k, wi,n(k,j) denotes the weight of the connection from the hidden node 'i' to the internal node n(k, j) in the tree. It can be shown that :

$$\sum_{k=1}^{V} O_k = 1$$

Instead of using a balanced binary tree for Hierarchical Softmax, the authors have used Huffman Trees. In theses trees, words which are more frequent are placed closer to the root whereas words which are less frequent are placed farther from the root. This is done using the frequency of each output word from the training instances, and results in the weights of the internal nodes updating faster.

In the Negative Sampling approach, few words are sampled from the vocabulary

and only these words are used to update the output weights. The words that are sampled (apart from the actual output word) should be ones that are less likely to be predicted given the input word, so that input word vector is most affected by the actual output words (and least affected by the output vectors of the remaining sampled words). The main objective being to maximize the similarity between the input word vector and the output context word vectors, it seems appropriate to discard some of the words in the vocabulary that do not contribute much but only increase time complexity. The negative samples are selected from the vocabulary list using their "unigram distribution", such that more frequent words are more likely to be negative samples. One approach for

converting the word vector representations into the document-term matrix is to take the sum (average, min/max etc.) of the word vectors of all the words present in the document and use this as the vector representation for the document. The authors of Word2Vec have also developed another version of their methodology called Doc2Vec for directly training sentences and paragraphs with the Skip Gram architecture instead of averaging the word vectors in the text. We tried a similar approach in order to improve the accuracy of our classification: instead of giving the whole list of token vectors to our classifier, we gave it instead an average vector based on the TF-IDF weights. This method, while not improving our accuracy, yielded satisfying results considering the magnitude of the dimension reduction and therefore the potential information loss.

2.1.4 Classification algorithms

2.1.4.1 Multinomial Naïve Bayes and Support Vector Machines

Let's first briefly introduce two families of classifiers that have been extensively used in the context of NLP : multinomial naive bayes and support vector machines. The multinomial naive bayes algorithm applies Bayes theorem : it is based on the rather strong assumption that, in the context of classification, every feature is independent of the others. This classifier will always output the category with the highest *a priori* probability using Bayes theorem. This algorithm has a simple and intuitive design, and is a good benchmark for classification purposes.

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Let's take a concrete example to better understand the implications of this naive rule : what is the probability that the expression "bewitching melody" is classified as "music" ?

$$P(music|bewitching\ melody) = \frac{P(bewitching\ melody|music) \times P(music)}{P(bewitching\ melody)}$$

The goal here is only to determine whether or not the sequence "bewitching melody" can be classified as "music" : we can therefore discard the denominator and compare the two following values :

$$P(bewitching\ melody|music) \times P(music)$$
$$vs.$$
$$P(bewitching\ melody|not\ music) \times P(not\ music)$$

The problem in this case is that in order to determine what the value of $P(bewitching\ melody|music) \times P(music)$ is, we need to count the number of occurrences of "bewitching melody" in the sentences labelled as "music". But what if this particular expression never appears in our training corpus ? The *a priori* probability is null, leading to the value of $P(bewitching\ melody|music) \times P(music)$ being null as well. This is where the naive Bayes hypothesis comes in : as every word is supposed to be independent from the others, we can look at the occurrence of each word in the expression instead of the entire expression directly. The value we wish to compute can now be expressed as follows :

$$P(bewitching\ melody) = P(bewitching) \times P(melody)$$
$$P(bewitching\ melody|music) = P(bewitching|music) \times P(melody|music)$$

Here, we still have a problem : one of the words composing the sequence might not be present in the training corpus, in which case the value of the formula above will be null. An *a priori* frequency-based probability equal to zero can have the undesirable effect of wiping out all the information in the other probabilities. The solution is therefore to add some kind of smoothing, adding a correction term to every probability estimate. The most popular approach is called Laplace smoothing : given an observation $x = (x_1, \ldots, x_d)$ from a multinomial distribution with N trials and parameter vector $\Theta = (\Theta_1, \ldots, \Theta_d)$, the smoothed version of the data can be represented as follows :

$$\hat{\Theta}_i = \frac{x_i + \alpha}{N + \alpha \times d} \quad i = 1, ..., d,$$

where the pseudocount $alpha > 0$ is the smoothing parameter ($alpha = 0$ corresponds to no smoothing).

Let's now briefly present the support vector machine algorithm. This method does not focus on probabilities, but aims at creating a discriminant function $f : X \rightarrow y$. The intuition of SVM in the linearly separable case is to put a line in the middle of two classes, so that the distance to the nearest positive or negative instance is maximized. It is important to note that this ignores the class distribution $P(X|y)$. The SVM discriminant function has the form :

$$f(X) = w^T x + b$$

The classification rule is $sign(f(X))$, and the linear decision boundary is specified by $f(x) = 0$. If f separates the data, the geometric distance between a point x and the decision boundary is $\frac{yf(X)}{\|w\|}$

Given training data, the goal is to find a decision boundary w, b that maximizes the geometric distance of the closest point. The optimization objective is therefore :

$$\max_{w,b} \min_{i=1}^{n} \frac{y_i(w^T x_i + b)}{\|w\|}$$

This optimization objective can be re-written with an additional constraint, considering the fact that the objective is the same for $k\hat{w}$, $k\hat{b}$ for any non-zero scalng factor k :

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1,\ i = 1, \ldots, n. \end{aligned}$$

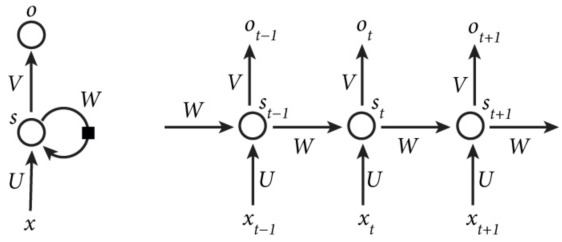
In the case where we don't make any assumption the linear separability of the training data, we relax the constraints by making the inequalities easier to satisfy. This is done with slack variables $\xi_i \geq 0$, one for each constraint. The sum of ξ_i is penalized in order to avoid points being on the wrong side of the decision boundary while still satisfying the constraint with large ξ_i. The new problem can in this case be expressed as follows :

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, \ldots, n,\ \xi_i \geq 0, \end{aligned}$$

Solving this objective leads to the dot product $x_i T x_j$, which allows SVM to be kernelized (using what is usually called the *kernel trick*), but we won't give much more details on the resolution of the equations.

2.1.4.2 Recurrent Neural Networks and LSTM

Recurrent neural networks leverage on the sequential nature of information : unlike regular neural network where inputs are assumed to be independent of each other, these architectures progressively accumulate and capture information through the sequences.



(a) RNN cell (b) Corresponding unfolded network

As we can see on the schema above, if the sequence we care about has a length of 5, the network would be unrolled into a 5-layer neural network, one layer for each instance. More precisely, x_t is the input at time step t. For example, x_0 could be embedding vector corresponding to the first word of a sentence. s_t is the hidden state at time step t : it corresponds to the *memory* of the network. s_t is generally computed through a non linear function based on the previous hidden state s_{t+1} and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. o_t is the output at step t. It could be a vector of probabilities across a corpus vocabulary if we wanted to predict the next word in a sentence (in this case for instance, $o_t = \text{softmax}(Vs_t)$).

In such an architecture, the same set of parameters (U, V, W) are shared across all steps of the sequence, only the inputs vary : this makes the learning process faster. It is important to note that in the context of classification, whether it is emotions or personality traits detection, we might not need to produce outputs at each step : in this case, only a final vector of probabilities would be required. As the RNN parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps : this is called Backpropagation Through Time (BPTT). This particular back-propagation phase can lead to a *vanishing gradient* phenomenon as the gradient signal can be multiplied by the weight matrix as many times as the number of steps in the sequence: in practice, these networks are limited to looking back only a few steps. If the weights in this matrix are small, the gradient signal ends up being so small that learning either slows down or stops completely. Moreover, it makes it more difficult to learn long-term dependencies in the data. Conversely, if the weights in this matrix are large, the very large gradient signal might cause learning to diverge (*exploding gradient* phenomenon).

This is one of the essential reasons why Long Short Term Memory architectures, introduced by Hochreiter Schmidhuber [1997], have an edge over conventional feed-forward neural networks and RNN. Indeed, LSTMs have the property of selectively remembering patterns for long durations of time. This is made possible by what is called a memory cell. Its unique structure is composed of four main elements : an input gate, a neuron with a self-recurrent connection, a forget gate and an output gate. The self-recurrent connection ensures that the state of a memory cell can remain constant from one timestep to another. The role of the

gates is to fine-tune the interactions between the memory cell and its environment using a sigmoid layer and a point-wise multiplication operation.

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

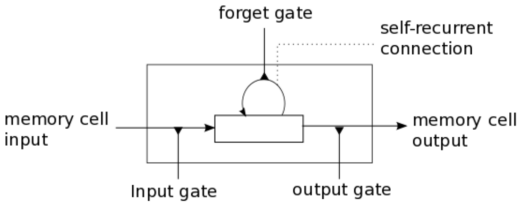


Fig. 1: LSTM memory cell

While the input gate can either allow incoming signal to alter the state of the memory cell or block it, the output gate can either allow the state of the memory cell to have an effect on other neurons or prevent it. Finally, the forget gate can influence the memory cell's self-recurrent connection, allowing the cell to *remember* or *forget* its previous state.

Let's now describe more precisely how a layer of memory cells is updated at each step t of the sequence. For the equations, we will use the following notations : x_t is the input to the memory cell layer at time t; W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o and V_o are weight matrices; b_i, b_f, b_c and bo are bias vectors. The first equations give the value for the input gate i_t and the candidate value for the states of the memory cells S_t at time t :

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$\widetilde{S_t} = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Then we compute the value for f_t, the activation function of the memory cells' forget gates at time t:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

Given the values of the input and forget gate activation functions i_t and f_t, and the candidate state value $\widetilde{S_t}, we can compute \mathrm{S}_t$ the memory cells' new state at time t:

$$S_t = i_t \times \widetilde{S_t} + f_t \times S_{t-1}$$

The new state of the memory cells allows us to compute their output gates values and finally their outputs:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o S_t + b_o)$$

$$h_t = o_t \times tanh(C_t)$$

2.2 Choice of model

Let's now simply provide a list of the different steps of our pipeline in order to sum up the information given above and give more details about our final choices. This precise pipeline is the one that obtained the highest accuracy among all the combinations we have tried.

1. Preprocessing
   - Tokenization of the document
   - Standardization of formulations using regular expressions (for instance replacing "can't" by "cannot", "'ve" by " have")
   - Deletion of the punctuation
   - Lowercasing the tokens
   - Removal of predefined stopwords (such as 'a', 'an' etc.)
   - Application of part-of-speech tags remaining tokens
   - Lemmatization of tokens using part-of-speech tags for more accuracy.
   - Padding the sequences of tokens of each document to constrain the shape of the input vectors. The input size has been fixed to 300 : all tokens beyond this index are deleted. If the input vector has less than 300 tokens, zeros are added at the beginning of the vector in order to normalize the shape. The dimension of the padded sequence has been determine using the characteristics of our training data. The average number of words in each essay was 652 before any preprocessing. After the standardization of formulations, and the removal of punctuation characters and stopwords, the average number of words dropped to with a standard deviation of . In order to make sure we incorporate in our classification the right number of words without discarding too much information, we set the padding dimension to 300, which is roughly equal to the average length plus two times the standard deviation.
2. Embedding
   - Each token is replaced by its embedding vector using Google's pre-trained Word2Vec vectors in 300 dimensions (which is the largest dimension available and therefore incorporates the most information), and this embedding is set to be trainable (our training corpus is to small to train our own embedding).
3. Classifier
   - Neural network architecture based on both one-dimensional convolutional neural networks and recurrent neural networks. The one-dimensional convolution layer plays a role comparable to feature extraction : it allows finding patterns in text data. The Long-Short Term Memory cell is then used in order to leverage on the sequential nature of natural language : unlike regular neural network where inputs are assumed to be independent of each other, these architectures progressively accumulate and capture information through the sequences. LSTMs have the property of selectively remembering patterns for long durations of time. Our final model first includes 3 consecutive blocks consisting of the following four layers : one-dimensional convolution layer - max pooling - spatial dropout

- batch normalization. The numbers of convolution filters are respectively 128, 256 and 512 for each block, kernel size is 8, max pooling size is 2 and dropout rate is 0.3. Following the three blocks, we chose to stack 3 LSTM cells with 180 outputs each. Finally, a fully connected layer of 128 nodes is added before the last classification layer.

2.3 Results

We tested different combinations of embeddings and classifiers in order to compare results. As explained at the end of the part on Word2Vec embeddings, we tried a hybrid model using averaged vector representations using TF-IDF weights : there is a loss of accuracy compared to the complete Word2Vec embedding, but results are better than the regular TF-IDF embedding. Let's provide the details of the accuracy obtained with each combination that we tested in our pipeline:

Model	EXT	NEU	AGR	CON	OPN
TF-IDF + MNB	45.34	45.11	45.24	45.31	45.12
TF-IDF + SVM	45.78	45.91	45.41	45.54	45.56
Word2Vec + MNB	45.02	46.01	46.34	46.38	45.97
Word2Vec + SVM	46.18	48.21	49.65	49.97	50.07
Word2Vec (TF-IDF averaging) + MNB	45.87	44.99	45.38	44.21	44.84
Word2Vec (TF-IDF averaging) + SVM	46.01	46.19	47.56	48.11	48.89
Word2Vec + NN (LSTM)	51.98	50.01	51.57	51.11	50.51
Word2Vec + NN (CONV + LSTM)	**55.07**	**50.17**	**54.57**	**53.23**	**53.84**

2.4 Possible improvements

In order to improve our accuracy, we could use hybrid models including both learning-based methods and linguistic features : adding lexicon-based features, using psycholinguistic databases, or even adding home-made features based on psychological research should have a positive impact on our accuracy scores.

3 Signal processing for emotion recognition

3.1 Theoretical foundations

3.1.1 Introduction

Speech emotion recognition purpose is to automatically identify the emotional or physical state of a human being from his voice. The emotional state of a person hidden in his speech is an important factor of human communication and interaction as it provides feedbacks in communication while not altering linguistic contents.

Speech emotion recognition is based on discrete emotion classification system. In most cases, literature focus only on 6 emotions labels introduced by Ekman, including happy, sad, angry, disgusted, fear and surprise. Although the emotion categories are more abundant and complex in real life.

The usual process for speech emotion recognition consists of three parts: signal processing, feature extraction and classification. Signal processing applies acoustic filter on original audio signals and splits it into meaningful units. The feature extraction is the sensitive point in speech emotion recognition because features need to both efficiently characterize the emotional content of a human speech and not depend on the lexical content or even the speaker. Finally, emotion classification will map feature matrix to emotion labels.

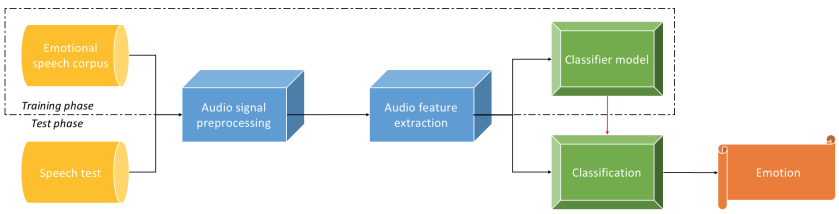


Fig. 2: Speech emotion recognition pipeline

In the following section, we will present in more details the audio feature extraction methodology and different relevant speech features typically used for speech emotion recognition.

3.1.2 Signal preprocessing

Following points describre audio signal preprocessing before audio features extraction.

3.1.2.1 Pre-emphasis filter

First, before starting feature extractions, it's advisable to apply a pre-emphasis filter on the audio signal to amplify all the high frequencies. A pre-emphasis filter has several advantages: it allows balancing the frequency spectrum since high frequencies usually have smaller magnitudes compared to lower ones and also avoid numerical problems on the Fourier Transform computation.

$$y_t = x_t - \alpha x_{t-1}$$

Typical values for the pre-emphasis filter coefficient α are 0.95 or 0.97.

3.1.2.2 Framing

After the pre-emphasis filter, we have to split audio signal into short-term windows called *frames*. For speech processing, window size is usually ranging from 20ms to 50ms with 40% to 50% overlap between two consecutive windows. Most popular settings are 25ms for the frame size with a 15ms overlap (10ms window step).

The main motivation behind this step is to avoid the loss of frequency contours of an audio signal over time because audio signals are non-stationary by nature. Indeed, frequency properties in a signal change over time, so it does not really make sense to apply the Discrete Fourier Transform across the entire sample. If we suppose that frequencies in a signal are constant over a very short period of time, we can apply Discrete Fourier Transform over those short time windows and obtain a good approximation of the frequency contours of the entire signal.

3.1.2.3 Hamming

After splitting the signal into multiple frames, we multiply each frame by a Hamming window function. It allows reducing spectral leakage or any signal discontinuities and improving the signal clarity. For example, if the beginning and end of a frame don't match then it will look like discontinuity in the signal and will show up as nonsense in the Discrete Fourier Transform. Applying Hamming function makes sure that beginning and end match up while smoothing the signal.

Following equation describes the Hamming window function:

$$H_n = \alpha - \beta cos(\frac{2\pi n}{N-1})$$

where $\alpha = 0.54$, $\beta = 0.46$ and $0 \leq n \leq N-1$ with N the window length.

3.1.2.4 Discrete Fourier Transform

The Discrete Fourier Transform is the most widely used transforms in all area of digital signal processing because it allows converting a sequence from the time

domain to the frequency domain. DCT provides a convenient representation of the distribution of the frequency content of an audio signal.

The majority of audio features used to analyze speech emotion are defined in the frequency domain because it reflects better the properties of an audio signal.

Given a discrete-time signal x_n $n = 0, \dots N-1$ (N samples long) the Discrete Fourier Transform can be defined as

$$X_k = \sum_{n=0}^{N-1} x_k e^{\frac{-i2\pi}{N} kn} \qquad k = 0, ..., N-1$$

Discrete Fourier Transform output is a sequence of N coefficient X_k constituting the frequency domain representation of a signal.

The inverse Discrete Fourier Transform takes Discrete Fourier coefficient and returns the original signal in time-domain:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} x_n e^{\frac{i2\pi}{N} kn} \qquad n = 0, ..., N-1$$

3.1.3 Short-term audio features

Once partitioning done, we can extract 34 features from time (3) and frequency (31) domain for each frame. Features from time domain are directly extracted from the raw signal samples while frequency features are based on the magnitude of the Discrete Fourier Transform.

3.1.3.1 Time-domain features

In following formulas, $x_i(n)$, $n = 0$, ..., $N-1$ is the nth discrete time signal of the ith frame and N the number of samples per frame (window size).

- **Energy**: Sum of squares of the signal values, normalized by the respective frame length

$$E_i = \frac{1}{N} \sum_{n=0}^{N-1} \mid x_i(n) \mid^2$$

- **Entropy of energy**: Entropy of sub-frames normalized energies. It permits to measure abrupt changes in the energy amplitude of an audio signal.
  To compute the Entropy of Energy of ith frame, we first divide each frame in K sub-frames of fixed duration. Then we compute Energy of each sub-frame and divide it by the total Energy of the frame E_i:

$$e_j = \frac{E_{subFrame_j}}{E_i}$$

where

$$E_i = \sum_{j=1}^{K} E_{subFrame_j}$$

Finally, the entropy H_i is computed according to the equation:

$$H_i = -\sum_{j=1}^{K} e_j.log_2(e_j)$$

- **Zero Crossing rate**: Rate of sign-changes of an audio signal

$$ZCR_i = \frac{1}{2N} \sum_{n=0}^{N-1} \mid sgn[x_i(n)] - sgn[x_i(n-1)] \mid$$

Where $sgn(.)$ is the sign function.

3.1.3.2 Frequency-domain features

In following formulas, $X_i(k)$, $k = 0, ..., N-1$ is the kth Discrete Fourier Transform (DCT) coefficient of the ith frame and N is the number of samples per frame (window size).

- **Spectral centroid**: Center of gravity of the sound spectrum.

$$C_i = \frac{\sum_{k=0}^{N-1} kX_i(k)}{\sum_{k=0}^{N-1} X_i(k)}$$

- **Spectral spread**: Second central moment of the sound spectrum.

$$S_i = \sqrt{\frac{\sum_{k=0}^{N-1} (k - C_i)^2 X_i(k)}{\sum_{k=0}^{N-1} X_i(k)}}$$

- **Spectral entropy**: Entropy of sub-frames normalized spectral energies.
  To compute the spectral entropy of ith frame, we first divide each frame in K sub-frames of fixed size. Then we compute spectral Energy (similar formula as time-domain energy) of each sub-frame and divide it by the total Energy of the frame. The spectral entropy H_i is then computed according to the equation:

$$H_i = -\sum_{k=1}^{K} n_k.log_2(n_n)$$

with

$$n_k = \frac{E_{subFrame_k}}{\sum_{j=1}^{K} E_{subFrame_j}}$$

- **Spectral flux**: Squared difference between the normalized magnitudes of the spectra of the two successive frames. It permits to mesure the spectral changes between to frame.

$$F_i = \sum_{k=0}^{N-1} [EN_i(k) - EN_{i-1}(k)]^2$$

with

$$EN_i(k) = \frac{X_i(k)}{\sum_{l=0}^{N-1} X_i(l)}$$

- **Spectral rolloff**: Frequency below which 90% of the magnitude distribution of the spectrum is concentrated. The lth DFT coefficient is corresponding to the spectral rolloff if it satisfies the following conditions:

$$\sum_{k=0}^{l-1} X_i(k) = 0.90 \sum_{k=0}^{N-1} X_i(k)$$

- **MFCCs**: Mel Frequency Cepstral Coefficients model the spectral energy distribution in a perceptually meaningful way. Those features are the most widely used audio features for speech emotion recognition. Following process permits to compute the MFCCs of the ith frame: Calculate the periodogram of the power spectrum of the ith frame:

$$P_i(k) = \frac{1}{N} \mid X_i(k) \mid^2$$

Apply the Mel-spaced filterbank (set of L triangular filters) to the periodogram and calculate the energy in each filter. Finally, we take the Discrete Cosinus Transform (DCT) of the logarithm of all filterbank energies and only keep first 12 DCT coefficients $C^l_{l=1,...,12}$:

$$C_i^l = \sum_{k=1}^{L} (log\tilde{E_i^k}) cos[l(k - \frac{1}{2})\frac{\pi}{L}] \qquad l = 1, ..., L$$

where $\tilde{E_k}$ is the energy at the ouptut of the kth filter on the ith frame.

3.2 Choice of model

3.2.1 Input

The **RAVDESS** database was used in order to evaluate our methodology in this paper. It contains acted emotions speech of male and female actors (gender balanced) that were asked to pretend six different emotions (happy, sad, angry, disgust, fear, surprise and neutral) at two levels of emotional intensity. Following table presents a summary of emotion distribution in **RAVDESS**:

RAVDESS			
Emotions	Man	Woman	**Total**
Happy	*96*	*96*	*192*
Sad	*96*	*96*	*192*
Angry	*96*	*96*	*192*
Scared	*96*	*96*	*192*
Neutral	*96*	*96*	*192*
Disgusted	*96*	*96*	*192*
Surprised	*96*	*96*	*192*
Total	**672**	**672**	**1344**

Table 2: RAVDESS database summary

Based on **Thurid Vogt and Elisabeth André** research: *Improving Automatic Emotion Recognition from Speech via Gender Differentiation (2006)*, we decided to separate out the male and female emotions using the identifiers provided by each database and to implement gender-dependent emotion classifiers rather than gender-independent ones. In their paper, separating male and female voices improved the overall recognition rate of their classifier by 2-4%.

3.2.2 Feature extraction

Once having extracted the speech features from the preprocessed audio signal (detailed on previous section) we obtain a matrix of features per audio file. We compute then the first and the second derivatives of each of those features to capture frame to frame changes in the signal. Finally, we calculate the following global statistics on these features: mean, standard deviation, kurtosis, skewness, 1% and 99% percentile. Thereby a vector of 360 candidate features is obtained for each audio signal.

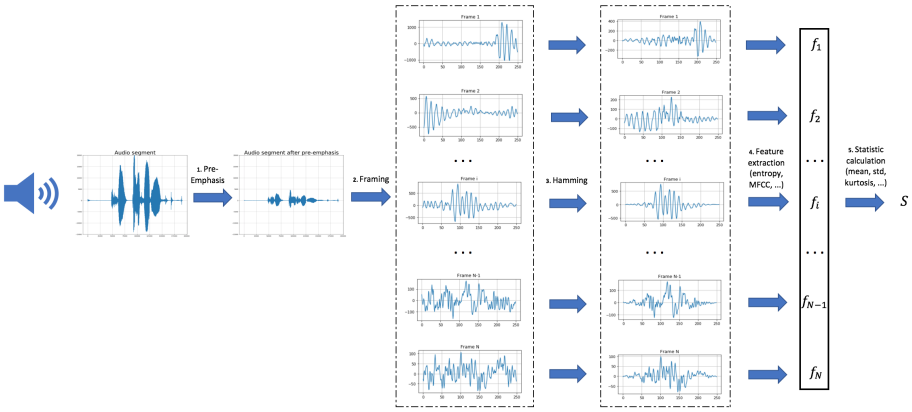


Fig. 3: Signal processing and audio feature extraction schema

Some post-processing also may be necessary before training and testing the classifier. First, normalization could be meaningful as extracted feature values have different orders of magnitude or different units. Secondly, it is also common to use dimensionality reduction techniques in order to reduce the memory and computation requirements of the classifier. There are two options for dimensionality reduction: features selection (statistical tests) and features transformation (Principal Component Analysis).

3.2.3 Classifier

In literature, various machine learning algorithms based on acoustic features (presented in previous section) are utilized to construct satisfying classifiers for hidden emotion detection in a human speech. Support Vector Machines (SVM) is the most popular and the most often successfully applied algorithm for speech emotion recognition. SVM is a non-linear classifier transforming the input feature vectors into a higher dimensional feature space using a kernel mapping function. By choosing appropriate non-linear kernels functions, classifiers that are non-linear in the original space can therefore become linear in the feature space. Most common kernel function are described below:

- **linear kernel**: $K(x_i, x_j) = x_i * x_j$
- **radial basis function (rbf) kernel**: $K(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$
- **d-degree polynomial kernel**: $K(x_i, x_j) = (x_i * x_j + \gamma)^d$

State of the art paper *"Speech emotion recognition: Features and classification models"* by **L. Chen**, **X. Mao**, **Y. Xue**, and **L. L. Cheng** achieved an accuracy of **86.5%** by combining principal component analysis and SVM respectively for dimensionality reduction and classification. Some sophisticated classifiers do

achieve higher recognition rates than simple SVM but not much. In the next section we will try to get as close as possible to the state of the art performances

3.3 Empirical results

We first implemented SVM classifiers based on different kernel functions (linear, polynomial and RBF), without dimensionality reduction and gender differentiation. Speech emotion recognition accuracies shown in next table were relatively low. However, the SVM with RBF kernel functions seems to be the best performer with an accuracy rate of 56.51%. Then we applied both feature selection (1%-Chi-squared test removed 75 features) and feature transformation (PCA) to reduce the dimension of the features. For PCA, three levels of explained variance were tested (90%, 95% and 98%) respectively leading to the following features dimensions : 100, 120 and 140. Our performances were still very low but the accuracy of polynomial and RBF increased respectively by 6% and 3% with the 140 feature dimension corresponding to the 98% contribution. RBF kernel still remains the best classifier.

PCA dimension	**linear**	**poly (2)**	**poly (3)**	**rbf**
None	*51.67%*	*54.28%*	*52.79%*	*56.51%*
140	*53.53%*	*50.19%*	*52.79%*	***59.48%***
120	*55.02%*	*50.56%*	*52.79%*	*58.74%*
100	*52.79%*	*48.33%*	*52.79%*	*58.36%*

Table 3: Different dimension and different kernel cross-validation accuracy rate

The first major improvement was observed with the implementation of gender differentiation as suggested in previous section. As shown in the following table, accuracy scores of almost all classifier (except for 3-degree polynomial) increased by almost 5%. The next figure illustrates accuracy rates obtained by cross-validation and the confusion matrix of the classifier with the highest accuracy score: RBF Kernel and PCA 180 features dimension (corresponding to 98% contribution).

PCA dimension	**linear**	**poly (2)**	**poly (3)**	**rbf**
None	*53.23%*	*55.02%*	*54.28%*	*60.59%*
180	*59.85%*	*55.39%*	*55.76%*	***64.20%***

Table 4: Gender differentiation - Cross-validation accuracy rate for different dimension and different kernel.

		Predicted labels						
		Happy	**Sad**	**Angry**	**Scared**	**Neutral**	**Disgusted**	**Surprised**
Actual labels	**Happy**	***65.9%***	*4.9%*	*7.3%*	*0.0%*	*7.3%*	*14.6%*	*0.0%*
	Sad	*17.9%*	***61.5%***	*7.7%*	*7.7%*	*0.0%*	*0.0%*	*5.1%*
	Angry	*7.9%*	*5.3%*	***63.2%***	*2.6%*	*0.0%*	*5.3%*	*15.8%*
	Scared	*5.3%*	*5.3%*	*0.0%*	***76.3%***	*7.9%*	*2.6%*	*2.6%*
	Neutral	*10.3%*	*5.1%*	*7.7%*	*5.1%*	***53.8%***	*10.3%*	*7.7%*
	Disgusted	*4.5%*	*0.0%*	*4.5%*	*4.5%*	*6.8%*	***72.7%***	*6.8%*
	Surprised	*3.3%*	*20.0%*	*3.3%*	*6.7%*	*6.7%*	*3.3%*	***56.7%***

Table 5: Confusion Matrix of best classifier

As can be seen above, *Surprise* and *Neutral* emotions were classified with the poorest accuracy compared to other emotions such as *Scared* and *Disgust* who achieved the highest results (respectively 76% and 73%). **RAVDESS** database contains speeches for 7 different emotions but we decided to remove *Surprise*, as our classifier had trouble differentiating it from other emotions. Final results have been quite satisfying. We have succeeded to obtain an accuracy score of almost 75% as shown in following table.

PCA dimension	**linear**	**poly (2)**	**poly (3)**	**rbf**
None	*63.34%*	*59.74%*	*65.80%*	*70.26%*
120	*54.50%*	*63.20%*	*64.94%*	***74.46%***

Table 6: 6-way emotions - Cross-validation accuracy rate for different dimension and different kernel.

		Predicted labels					
		Happy	**Sad**	**Angry**	**Scared**	**Neutral**	**Disgusted**
Actual labels	**Happy**	***80.0%***	*0.0%*	*5.7%*	*5.7%*	*5.7%*	*2.9%*
	Sad	*8.1%*	***81.1%***	*0.0%*	*0.0%*	*2.7%*	*8.1%*
	Angry	*6.3%*	*6.3%*	***75%***	*0.0%*	*6.3%*	*6.3%*
	Scared	*6.7%*	*0.0%*	*4.4%*	***71.1%***	*8.9%*	*8.9%*
	Neutral	*11.1%*	*5.6%*	*2.8%*	*8.3%*	***66.7%***	*5.6%*
	Disgusted	*0.0%*	*8.7%*	*0.0%*	*4.3%*	*2.2%*	***84.8%***

Table 7: Confusion Matrix of best classifier - 6-way emotions

3.4 Potential improvements

Our model presents reasonably satisfying results. Our prediction recognition rate is around 65% for 7-way (happy, sad, angry, scared, disgust, surprised, neutral) emotions and 75% for 6-way emotions (surprised removed).

In order to improve our results and to try to get closer to the state of the art, we will try to implement more sophisticated classifiers in second period of this project. For example, Hidden Markov Model (HMM) and Convolutional Neural Networks (CNN) seem to be potential good candidates for speech emotion recognition. Unlike SVM classifiers, those classifiers are train on short-term features and not on global statistics features. HMM and CNN are considered to be advantageous for better capturing the temporal dynamic incorporated in speech.

To implement a multimodal model for emotion recognition we will also need to set up the removal of silence and probably build a speaker identifier to not bias our emotion predictions in the speech domain.

4 Computer vision for emotion recognition

4.1 Theoretical foundations

4.1.1 Introduction Challenges such as emotion recognition can typically not be solved though classical machine learning techniques. All the recent research papers focus on several deep learning techniques, some of which include Artificial Neural Networks (ANN), Convolution Neural Network (CNN), Region-CNN (R-CNN), Fast R-CNN, Recurrent Neural Network (RNN) or Long Short-Term Memory (LSTM). The aim of the following section is to develop the bases that lead to Convolutional Neural Networks (CNN).

4.1.2 The perceptron

A perceptron is a single layer Neural Network. A perceptron can simply be seen as a set of inputs, that are weighted and to which we apply an activation function. This produces a weighted sum of inputs, resulting in an output. This is typically used for binary classification problems.

The perceptron was first introduced in 1957 by Franck Rosenblatt. Since then, it has been the core of Deep Learning.

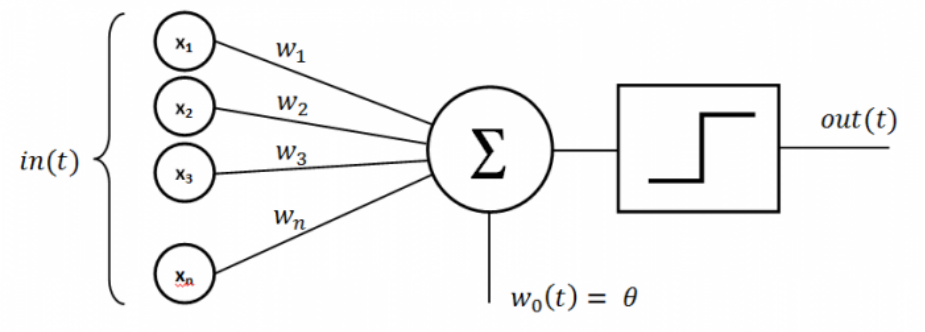


Fig. 4: A schema of the Rosenblatt Perceptron.

We have a set of inputs X_i, to which we apply :

- weights w_i to form a weight sum,
- a bias term b_i that can also be introduced as an input,
- and pass the overall function into an activation function.

The inputs can be seen as neurons, and will be called the input layer. The activation function might take several forms and should "send" the weighted sum into a smaller set of possible values that allows us to classify the output. One common activation function for a simple perceptron is a modified version of *sign*

function, noted $h_w(x^i)$, for which we assign the value 1 if the weighted sum is larger than some threshold, and 0 otherwise. This projects the weighted sum of inputs into a Heaviside function.

Then, a given observation can be either well classified, or in the wrong class. As in most optimization problems, we want to minimize the cost, i.e the sum of the individual losses on each training observation.

A pseudo code corresponding the Perceptron can be described as follows :

Algorithm 1 Perceptron parameters update

Require: A set of observations of the form (x_i, y_i), for $i = 1...n$
h an activation function to apply to the inputs
a random initialization of weights and biases close to 0
repeat
  for each (x_i, y_i) **do**
    compute $h(x_i)$
    if $\hat{y_i} \neq y_i$ **then** update the parameters
      $w_d \leftarrow w_d - \alpha \frac{\partial L(y_i, h(x_i))}{\partial w_d}$
      $b_d \leftarrow b_d - \alpha \frac{\partial L(y_i, h(x_i))}{\partial b_d}$
    end if
  end for
Ensure: A stopping criteria is met

In the most basic framework, we consider essentially a linear classification rule than can be represented as :

$$h(x) = sign(w_i + {b_i}^t x)$$

where w_i represents the bias term, and b_i the weights on each neuron.

In this framework, the empirical loss function to minimize when classifying is defined as :

$$min_{(x,\beta)} \hat{L}_n(g)$$

where :

$$\hat{L}_n(g) = \frac{1}{n} \sum I_n(-(\alpha + \beta^t x)y > 0)$$

However, due to the form of the *sign* function, we cannot apply a gradient descent to identify the minimum. We need to apply a stochastic gradient descent. The perceptron "learns" how to adapt the weights using back propagation. The weights and bias are firstly set randomly, and we compute an error rate. Then, we proceed to a back propagation to adjust the parameters that we did not correctly identify, and we start all over again for a given number of epoch.

We will further detail the concepts of stochastic gradient descent and back propagation in the context of Multilayer Perceptron.

Another approach to overcome the lack of derivability of the Heaviside function is to use the sigmoid function as an activation function :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

This produces a modified perceptron, which is called a logistic regression with 0 hidden layer.

The perceptron has another major drawback. If the categories are linearly separable for example, it identifies a single separating hyper-plane without taking into account the notion of margin we would like to maximize. This problem is solved by the Support Vector Machine (SVM) algorithm.

Finally, the perceptron classification rule is linear. This is due to the simple structure of the perceptron, and implies limitations when looking at complex problems such as emotion recognition.

4.1.3 Multilayer Perceptron (MLP)

A multilayer perceptron is a feedforward artificial neural network (ANN), made of an input layer, one or several hidden layers, and an output layer.

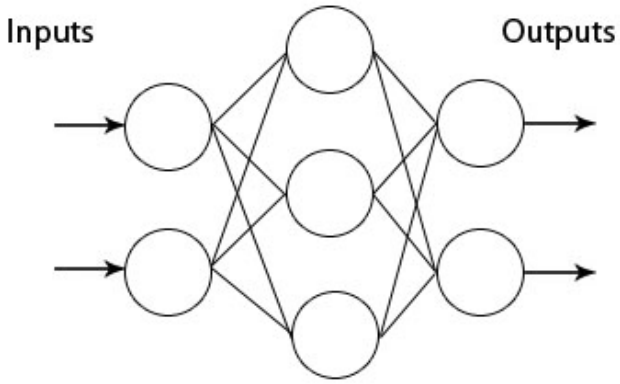


Fig. 5: A schema of a Multilayer Perceptron (MLP)

The phase of "learning" for a multilayer perceptron is reduced to determining for each relation between each neuron of two consecutive layers :

- the weights w_i
- the biases b_i

One major difference with the Rosenblatt perceptron is that the structure, due to the multiplicity of (hidden) layers, is no longer linear and allows us to model more complex data structures.

Such values are determined using the concept of maximum likelihood estimations (MLE). The maximum likelihood estimate $\hat{\theta}^{MLE}$ determines the parameters which maximize the probability of observing θ. For classification problems, the maximum likelihood optimization can be expressed as follows :

$$\hat{\theta}^{MLE} = argmax_{\theta} P(Y|X, \theta)$$

where $\theta = [W^{(l)}, b^{(l)}]$. We suppose that we have n training samples, and that each sample has the same number of classes (k).

Assuming that the Y_n are identically and independently distributed (iid), which will in most cases be the assumption, we know that : $Y_n \overset{iid}{\sim} P(Y|X, \theta)$. Therefore :

$$\hat{\theta}^{MLE} = argmax_{\theta} \prod P(Y_n|X_n, \theta)$$

In the context of classification, labels will follow a Bernouilli distribution. We can state that :

$$\hat{\theta}^{MLE} = argmax_{\theta} \prod_n \prod_k P(Y_n = k|X_n, \theta)^{t_{n,k}}$$

where $t_{n,k}$ is a one-hot encoded version of our labels Y. As a sample always belong to a class, we know that :

$$\sum_k P(Y_n = k|X_n, \theta) = 1$$

We can focus on the log-likelihood to make the computations easier :

$$\hat{\theta}^{MLE} = argmax_{\theta} \sum_n \sum_k t_{n,k} log(P(Y_n = k|X_n, \theta))$$

When dealing with classification problems, we want to minimize the classification error, which is expressed as the negative log-likelihood (NLL) :

$$NLL(\theta) = -l(\theta) = -\sum_n \sum_k t_{n,k} log(P(Y_n = k|X_n, \theta))$$

For a single sample, the measure is called the cross-entropy loss :

$$NLL(\theta) = -\sum_k t_{n,k} log(P(Y_n = k|X_n, \theta))$$

We can indeed prove the equivalence between minimizing the loss and maximizing the likelihood. We need to find θ^{MLE} that minimizes this loss. Note that the MLE produces an unbiaised estimator, i.e $E(\hat{\theta}^{MLE}) = \theta^*$.

As we do not have access to the potential entire data set, but to a sample only, we minimize the cost (the sum of the losses) given the concept of Empirical Risk Minimization (ERM)

4.1.4 Activation functions

There are several activation functions that can be used. The activation functions can vary depending on the layer we consider. The most popular activation functions are the following :

- sigmoid : $\sigma(z) = \frac{1}{1+e^{-z}}$
- hyperbolic tangent : $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- rectified linear unit (ReLU) : $g(z) = max(0, z)$
- leaky ReLU : $g(z) = max(0.01z, z)$
- parametric ReLU (PReLU) : $g(z) = max(\alpha z, z)$
- softplus function : $g(z) = log(1 + e^z)$

In practice, for our face emotion recognition problem, we tend to use the ReLU acitvation function. Indeed, the ReLU is not subject to the vanishing gradient problem. It has issues dealing with negative inputs, but since our input takes the form of arrays describing colors of pixels, we only have positive inputs.

These activation functions are commonly used on the hidden layers. For the output layer, we use another activation function called the $softmax$ activation. The $softmax$ allows a transformation of the output layer into a probabilities for each class. For classification purposes, we then only select the maximal probability as the class computed.

$$g(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

4.1.5 Gradient descent

There are two ways to solve this minimization problem :

- using traditional calculus, but this typically produces long and un-intuitive equations
- using stochastic gradient descent

Stochastic gradient descent offers a two step gradient descent (forward propagation, and back propagation) that allows us to approach the solution in a computationally efficient way. The stochastic gradient descent takes each observation individually, and computes a gradient descent. Then, the overall gradient is averaged. On the other hand, the batch gradient descent needs to wait until the optimal gradient has been computed on all training observations.

The forward propagation consist of applying a set of weights, bias and activation functions to an input layer in order to compute the output. Then,

while learning from our classification errors, we update the weights by moving backwards. This is the back propagation.

We won't cover the details of the back propagation, but for the intuition, it is enough to state that the concept of back propagation is a simple application of the chain rule that allows :

$$\frac{\partial}{\partial t} f(x(t)) = \frac{\partial f}{\partial x} \frac{\partial x}{\partial t}$$

However, in practice, the stochastic gradient descent is rarely used. Some improved alternatives are preferred, including :

- Mini batch gradient descent, which performs update for every mini-batch of n training examples, but cannot guarantee good convergence due to the choice of the learning rate
- Momentum, a method that helps accelerate stochastic gradient descent in the relevant direction and dampens oscillations by adding a fraction of the past time step to the current one
- Adaptive Moment Estimation (Adam), that computes adaptive learning rates for each parameter.

4.1.6 Regularization

One of the major challenges of deep learning is avoiding overfitting. Therefore, regularization offers a range of techniques to limit overfitting. They include :

- Weight decay
- Drop-out
- Batch normalization (BN)
- Data Augmentation

The weight decay is a way of implementing an L2-regularization term.

$$w_d \leftarrow w_d(1 - \frac{\alpha\lambda}{m}) - \alpha \frac{\partial L(y_i, h(x_i))}{\partial w_d}$$

where α is the learning rate and λ the L2-regularization term. The L2-regularization penalizes large coefficients and therefore avoids overfitting.

The drop-out technique allows us for each neuron, during training, to randomly turn-off a connection with a given probability. This prevents co-adaptation between units.

The batch normalization subtracts a measure of location, and divides by a measure of scale each input of each layer for faster learning. This reduces co-variate shift. The output of a layer $l-1$ is the input of the layer l.

Data augmentation is a popular way in image classification to prevent overfitting. The concept is to simply apply slight transformations on the input images (shift, scale...) to artificially increase the number of images.

4.1.7 Convolution Neural Network

In the field of computer vision, the new standard is the convolution neural network (CNN). CNNs are special types of neural networks for processing data with grid-like topology.

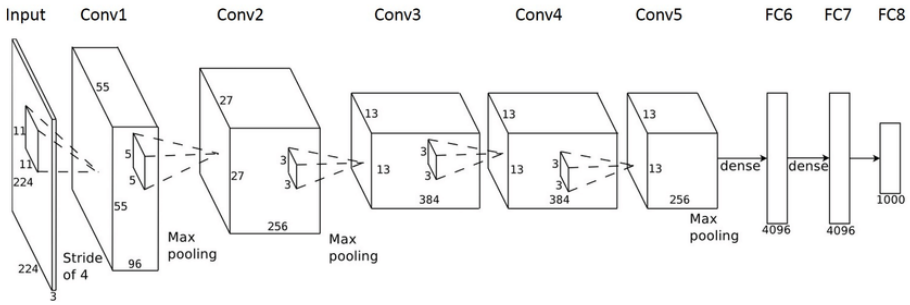


Fig. 6: Typical CNN architecture

In typical SVM approaches, a great part of the work was to select the filters (e.g Gabor filters) and the architecture of the filters in order to extract as much information from the image as possible. With the rise of deep learning and greater computation capacities, this work can now be automated. The name of the CNNs comes from the fact that we convolve the initial image input with a set of filters. The parameter to choose remains the number of filters to apply, and the dimension of the filters. The dimension of the filter is called the stride length. Typical values for the stride lie between 2 and 5.

In some sense, we are building a convolved output that has a volume. It's no longer a 2 dimensional picture. The filters are hardly humanly understandable, especially when we use a lot of them. Some are used to find curves, other edges, other textures... Once this volume has been extracted, it can be flattened and passed into a dense Neural Network.

Mathematically, the convolution is expressed as :

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)\partial\tau$$

The convolution represents the percentage of area of the filter g that overlaps with the input f at time τ over all time t. However, since $\tau < 0$ and $\tau > t$ have no meaning, the convolution can be reduce to :

$$(f * g)(t) = \int_{0}^{t} f(\tau)g(t-\tau)\partial\tau$$

At each convolution step, for each input, we apply an activation function (typically ReLU). So far, we have only added dimensionality to our initial image input. To reduce the dimension, we then apply a pooling step. Pooling involves a down sampling of features so that we need to learn less parameters when training. The most common form of pooling is max-pooling. For each of the dimension of each of the input image, we perform a max-pooling that takes, over a given height and width, typically 2x2, the maximum value among the 4 pixels. The intuition is that the maximal value has higher chances to be more significant when classifying an image.

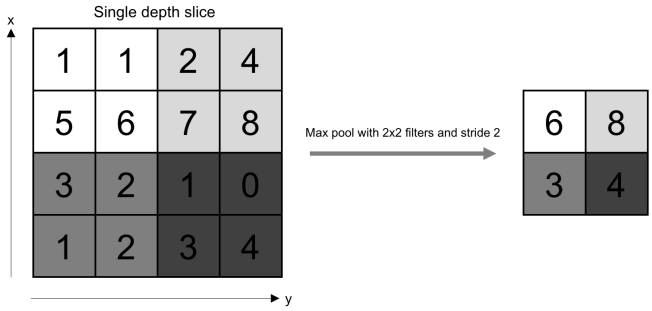


Fig. 7: Illustration of a max pooling step

We have now covered all the ingredients of a convolution neural network :

- the convolution layer
- the activation
- the pooling layer
- the fully connected layer, similar to a dense neural network

The order of the layers can be switched :
$ReLU(MaxPool(Conv(X))) = MaxPool(ReLU(Conv(X)))$
In image classification, we usually add several layers of convolution and pooling. This allows us to model more complex structures. Most of the model tuning in deep learning is to determine the optimal model structure. Some famous algorithms developed by Microsoft or Google reach a depth of more than 150 hidden layers.

4.2 Initial model

First of all, when exploring the data of the FER2013 data set, we observe that there is an imbalance in the number of images by class (emotion).

The labels are the following :

- 0 : Angry

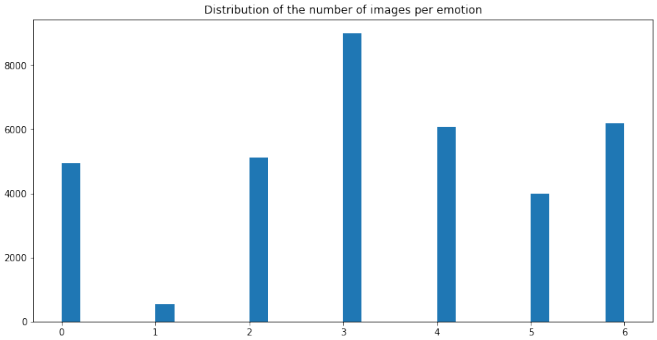


Fig. 8: Distribution of the number of images by class (emotion)

- 1 : Disgust
- 2 : Fear
- 3 : Happy
- 4 : Sad
- 3 : Surprise
- 3 : Neutral

We will therefore need to explore data augmentation techniques to solve this issue. The train set has 28709 images, the test set has 3589 images. For each image, the data set contains the grayscale color of 2304 pixels (48x48), as well as the emotion associated.

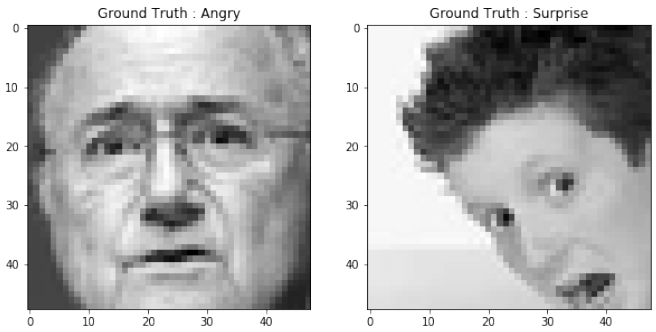


Fig. 9: Example of images contained in the data set

The challenge is that some pictures are miss-classified, while other images are only show part of the face. For example, the two images below seem rather clearly miss-classified.

Fig. 10: Example of miss-classified images

So, what can we expect from our model ? State of the art paper "Facial Expression Recognition using Convolutional Neural Networks: State of the Art" by Christopher Pramerdorfer and Martin Kampel has at the end of year 2016 achieved an accuracy of 75.2 %.

Not surprisingly, deep learning solutions perform better than more classical SVM algorithms in the literature.

The deep learning architecture we will be using is based on the best performing algorithm from the paper cited above. The corresponding architecture in Keras is the following.

```
def createModel2():

    model = Sequential()

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

    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())

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

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(nClasses, activation='softmax'))

    return model
```

This simple architecture produces over 440'000 parameters to estimate. The computation time is around 8 hours on local machine. In order to prevent overfitting, we also apply Keras built-in data generation module.

```
datagen = ImageDataGenerator(
        zoom_range=0.2,          # randomly zoom into images
        rotation_range=10,       # randomly rotate images
        width_shift_range=0.1,   # randomly shift images horizontally
        height_shift_range=0.1,  # randomly shift images vertically
        horizontal_flip=True,    # randomly flip images
        vertical_flip=False)     # randomly flip images
```

The optimizer we chose is RMSprop, an optimizer that divides the learning rate by an exponentially decaying average of squared gradients. The loss we use is the categorical cross-entropy, since we face a classification problem. Finally, the metric we use is the accuracy.

When plotting the accuracy and the loss in terms of the epochs, on both the training and the validation set, we observe a rather clear overfitting that occurs after approximately 20 epochs. Solutions will be addressed in the next section.

In the context of multimodal sentiment analysis, a key component is to be able to understand emotions from a video input, not only from pictures. The methodology to deploy our trained model on a webcam stream is the following :

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 48, 48, 32)	320
max_pooling2d_2 (MaxPooling2	(None, 24, 24, 32)	0
batch_normalization_1 (Batch	(None, 24, 24, 32)	128
conv2d_3 (Conv2D)	(None, 22, 22, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 11, 11, 32)	0
batch_normalization_2 (Batch	(None, 11, 11, 32)	128
conv2d_4 (Conv2D)	(None, 11, 11, 32)	9248
max_pooling2d_4 (MaxPooling2	(None, 5, 5, 32)	0
conv2d_5 (Conv2D)	(None, 5, 5, 32)	9248
flatten_1 (Flatten)	(None, 800)	0
dense_1 (Dense)	(None, 512)	410112
dense_2 (Dense)	(None, 7)	3591

Total params: 442,023
Trainable params: 441,895
Non-trainable params: 128

Fig. 11: Keras model summary

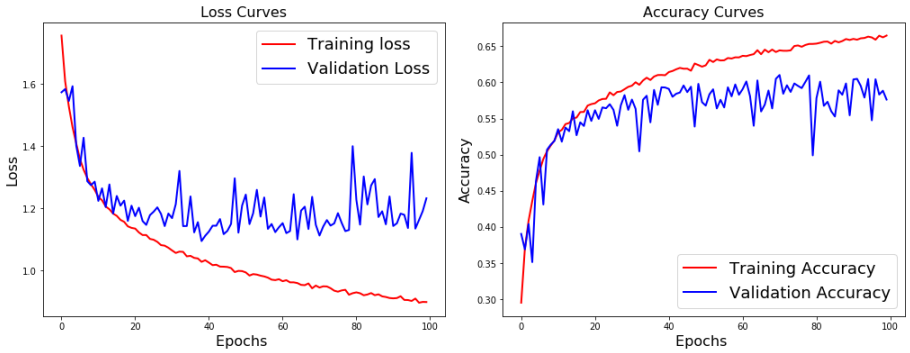


Fig. 12: Accuracy on training and validation set

- analyze the video image by image
- apply a grayscale filter to work with fewer inputs
- identify the face and zoom on it
- manage multiple faces
- reduce pixel density to same pixel density than the train set
- transform the input image to a model readable input
- predict the emotion of the input

An illustration of the process is proposed in Figure 20. The image processing is done with OpenCV on Python. The face detection is done with a Cascade Classifier.

Cascade classifiers is based on the concatenation of several classifiers, using and uses all the information from the output from the previous classifier as additional information for the next classifier in the cascade. Cascade Classifiers are typically applied to regions of interest of an image. The classifier will return 0 or 1 depending on whether the region is likely to show the object tested. The classifiers typically are Adaboost, Real Adaboost, Gentle Adaboost and Logitboost. OpenCV offers pre-trained cascade models for face detection.

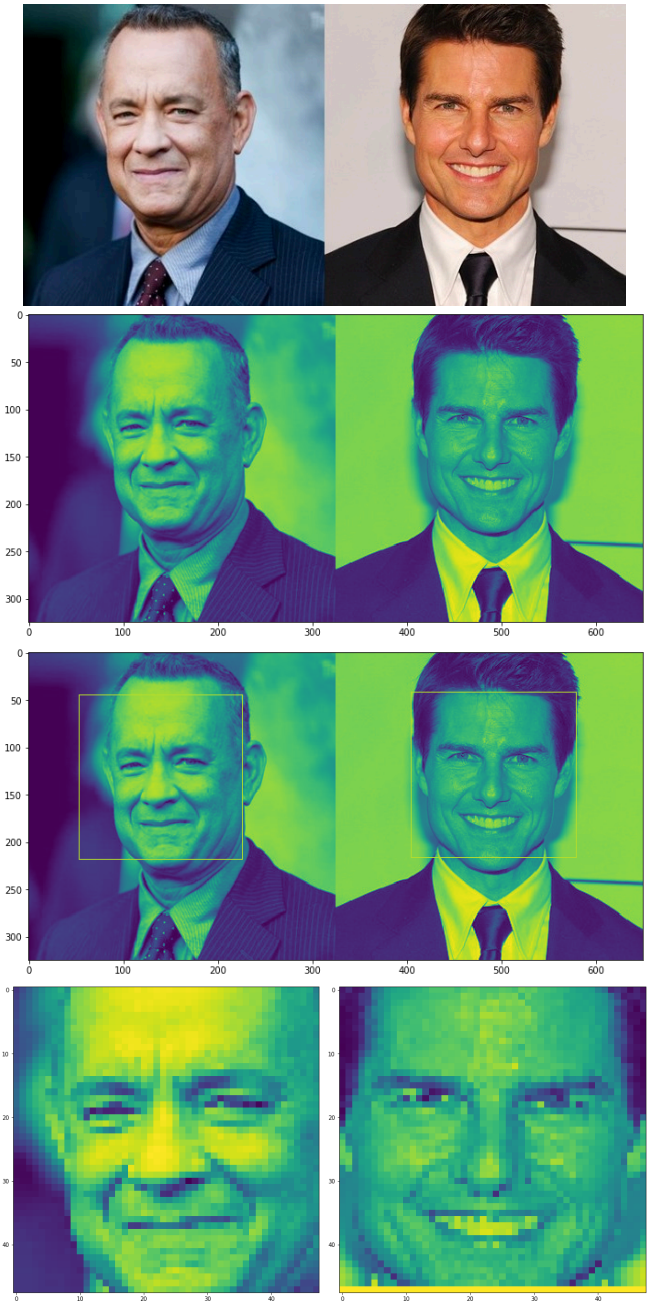


Fig. 13: Illustration of the process

4.3 Illustration

The webcam facial emotion algorithm presented above can be illustrated with a quick example. As shown on figure 10, the classification seems to work well in practice. There are however still many sources of improvements.

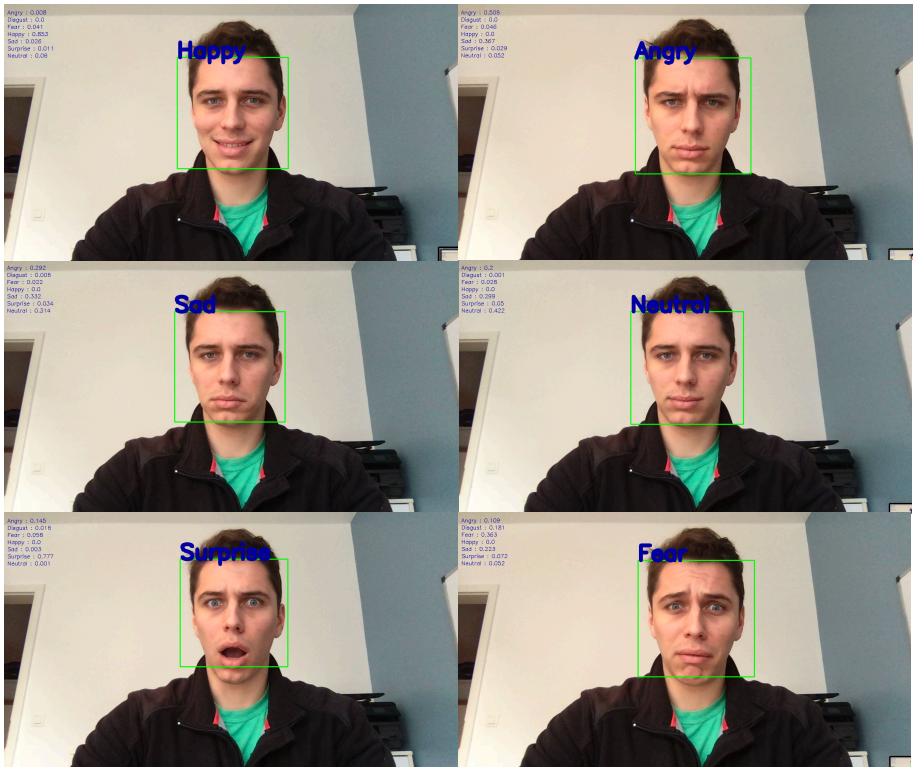


Fig. 14: Live facial emotion recognition

4.4 Potential improvements

Our model seems to work reasonably well. The accuracy reached 61%, which is still 14% behind state of the art models.

We seem to face an issue of overfitting. For this reason, we will include drop out layers in future model architectures. Moreover, it is very hard to obtain a prediction of "disgust". Our initial model is imbalanced, and this seems to have an effect on this class. We need targeted data augmentation on this specific class for example.

Another way to improve the model and approach state of the art solutions is to include extracted facial features in the design matrix. The extraction in state of the art models is usually made using histograms of oriented gradients (Hog) on sliding windows, but also take into account face landmarks.

Finally, in terms of graphical display, a potential improvement, instead of displaying probabilities on the top corner, would be to display graphs instead.

All these improvements will be explored further in the second period of this project.

5 Ensemble model

5.1 Theoretical foundations

5.2 Methodology

5.3 Model optimization

5.4 Illustration

5.5 Outcome

6 Deploying the model on a web interface

6.1 Tensorflow.js

6.2 Architecture

7 Compliance

7.1 GDPR Highlights

7.2 Limitations

8 Conclusion

References

1. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS), https://zenodo.org/record/1188976/?f=3.XAcEs5NKhQK
2. The Facial Emotion Recognition Challenge from Kaggle, https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data
3. End-to-End Multimodal Emotion Recognition using Deep Neural Networks, https://arxiv.org/pdf/1704.08619.pdf
4. Facial Expression Recognition using Convolutional Neural Networks: State of the Art, https://arxiv.org/pdf/1612.02903.pdf
5. B.Basharirad, and M.Moradhaseli. *Speech emotion recognition methods: A literature review.* AIP Conference Proceedings 2017. https://aip.scitation.org/doi/pdf/10.1063/1.5005438
6. L.Chen, M.Mao, Y.Xue and L.L.Cheng. *Speech emotion recognition: Features and classification models.* Digit. Signal Process, vol 22 Dec. 2012.
7. T.Vogt, E.André and J.Wagner. *Automatic Recognition of Emotions from Speech: A Review of the Literature and Recommendations for Practical Realisation.* Affect and Emotion in Human-Computer Interaction, 2008.
8. T.Vogt and E.André. *Improving Automatic Emotion Recognition from Speech via Gender Differentiation.* Language Resources and Evaluation Conference, 2006.
9. T.Giannakopoulos. *pyAudioAnalysis: An Open-Source Python Library for Audio Signal Analysis.* Dec. 2015https://doi.org/10.1371/journal.pone.0144610