Proposal of Real-Time Multimodal Emotion Recognition

AI Comprehensive Practice: Custom Project

Shengjie XIA
School of Information Science and Engineering
Lanzhou University
xiashj21@lzu.edu.cn

Abstract

In this paper, I am exploring state of the art models in multimodal emotion recognition. I 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.

This template is built on NeurIPS 2019 template[1] and provided for your convenience.

1 Key Information to include

- Mentor: Kun Zhan, Wei Su, Jizhao Liu, Cunlu Xu
- Sharing project: https://github.com/xiashj2021/AI-Comprehensive-Practice

2 Research paper summary

In selecting a representative example of close reading, I have chosen this text from a wide range of available literature.

Title	Deep learning for affective computing: Text-based emotion recognition in decision support
Journal	Decision support systems
Year	2018
URL	https://www.sciencedirect.com/science/article/pii/S0167923618301519

Table 1: Bibliographical information of affective computing [1]

2.1 Background

Based on the introduction, this paper focuses on improving text-based emotion recognition using deep learning techniques. The authors are motivated by several key factors:

Importance of emotions in human decision-making The authors highlight that emotions play a crucial role in human decision-making, communication, attention, and memory. They argue that computational systems that can recognize and adapt to human emotions (affective computing) can provide better decision support.

[1] https://www.overleaf.com/latex/templates/neurips-2019/tprktwxmqmgk

Limitations of current approaches While humans can naturally recognize emotions, this task remains challenging for computational systems. The authors note that previous research in affective computing has mainly used traditional machine learning methods, overlooking recent advances in deep learning.

Potential of deep learning The authors believe that deep learning techniques, particularly recurrent neural networks and transfer learning, can significantly improve the performance of text-based emotion recognition systems.

Lack of benchmarking datasets The authors identify a scarcity of widely-accepted datasets for text-based emotion recognition, which hinders fair comparisons and benchmarking of different methods.

The main problems the authors are attempting to solve include:

Improving the accuracy of text-based emotion recognition They propose to apply deep learning techniques, specifically long short-term memory networks (LSTMs), to enhance performance in detecting emotions from text.

Addressing class imbalances The authors introduce modifications to handle imbalanced distributions of emotion labels, including bidirectional text processing, dropout layers, and a weighted loss function.

Developing a novel transfer learning approach They propose "sent2affect," a method that first trains the network on sentiment analysis before fine-tuning it for emotion recognition.

Providing a comprehensive benchmark The authors aim to contribute a holistic comparison of different methods across various affect-labeled datasets used in prior research.

The knowledge they hope to discover includes:

1. The effectiveness of deep learning techniques in text-based emotion recognition compared to traditional machine learning methods.
2. The impact of their proposed modifications (bidirectional processing, dropout layers, weighted loss function) on handling class imbalances in affective computing.
3. The performance of their novel transfer learning approach (sent2affect) in improving emotion recognition accuracy.
4. A comprehensive understanding of how different methods perform across various affect-labeled datasets, considering factors such as linguistic style, domain, affective dimensions, and the structure of the outcome variable.

By addressing these issues and discovering this knowledge, the authors hope to advance the field of affective computing and improve its applications in various areas such as decision support, human-computer interaction, marketing, and social sciences.

2.2 Summary of contributions

The paper focuses on improving text-based emotion recognition using deep learning techniques. Here are the key contributions:

New Algorithms Proposes the use of **long short-term memory networks (LSTMs)** and **bidirectional text processing** to enhance emotion detection accuracy.

Experimental Results Introduces modifications like **dropout layers** and a **weighted loss function** to handle class imbalances.

Transfer Learning Approach Develops a novel method called "**sent2affect**", which trains on sentiment analysis before fine-tuning for emotion recognition.

Benchmarking Aims to provide a comprehensive comparison of different methods across various affect-labeled datasets.

2.3 Limitations and discussion

Based on the conclusion and discussion sections, the authors have indeed addressed some limitations of their work. However, there are still several areas where the paper could have been strengthened:

Dataset Limitations The authors acknowledge that the datasets used for affective computing are relatively small compared to other deep learning applications. While they suggest creating larger datasets in future research, they could have made their findings stronger by attempting to create or use a larger dataset themselves.

It's not clear if the authors evaluated their approach on diverse languages or text types. If they only used English texts, this limits the generalizability of their findings.

Comparative Analysis While the authors compare their approach to traditional machine learning baselines, they don't seem to compare against other state-of-the-art deep learning approaches for emotion recognition. This comparison could have provided more context for their performance improvements.

Error Analysis The paper doesn't appear to include a detailed error analysis. Understanding the types of errors their model makes compared to previous approaches could have provided valuable insights into the strengths and weaknesses of their method.

Robustness Testing The authors don't mention testing their model's robustness to adversarial examples or noise in the input text. This kind of evaluation could have strengthened their claims about the model's effectiveness.

Ethical Considerations While the authors discuss potential applications, they don't seem to address potential ethical concerns or biases that might arise from using their emotion recognition system in decision support contexts.

Computational Resources The authors don't provide details about the computational resources required for their approach. This information could be valuable for practitioners considering implementing their method.

Hyperparameter Optimization While the authors mention extensive training was required, they don't provide details about their hyperparameter optimization process, which could be crucial for reproducing their results.

Long-term Evaluation The paper doesn't discuss how their model might perform over time as language use evolves, which could be important for long-term decision support applications.

Despite these limitations, the paper still presents convincing findings. The authors' thorough approach to evaluating their method across multiple datasets and tasks, their novel transfer learning approach (sent2affect), and the consistent performance improvements they demonstrate all lend credibility to their work.

The authors are also transparent about the current limitations of affective computing datasets and the need for standardization in annotation schemes. This honesty about the field's challenges strengthens their overall argument.

Moreover, their proof-of-concept application to fake news detection demonstrates the potential real-world applicability of their approach, which adds to the paper's impact.

In conclusion, while there are areas where the research could have been expanded or strengthened, the paper still makes significant contributions to the field of affective computing and presents a convincing case for the use of customized deep learning approaches in text-based emotion recognition.

2.4 Why this paper?

Some reasons for selecting this paper include:

- Interest in the intersection of deep learning and affective computing
- Curiosity about improving text-based emotion recognition
- Exploring potential applications of emotion recognition in decision support systems
- Interest in novel transfer learning approaches in natural language processing

I've gained the insights into applying deep learning to affective computing from it.

2.5 Wider research context

This paper on "Deep learning for affective computing: Text-based emotion recognition in decision support" connects to several broad topics in NLP research:

Representing Language The paper explores how to represent emotional content in text, which is a crucial aspect of language representation. By using word embeddings, the authors reduce high-dimensional one-hot encoded vectors to lower-dimensional spaces, which aligns with modern approaches to dense word representations in NLP. This method of representation helps capture semantic relationships between words, including their emotional connotations.

Language Structure The use of Long Short-Term Memory (LSTM) networks, especially bidirectional LSTMs, demonstrates an attempt to capture the sequential nature of language. This approach recognizes that emotions in text are not just conveyed by individual words but by their sequence and context.

Challenges in Language Modeling The paper addresses several challenges inherent in modeling language, particularly for emotion recognition:

Class imbalance The authors propose a weighted loss function to handle imbalanced emotion labels, a common issue in many NLP tasks.

Contextual understanding By using bidirectional processing, the model attempts to capture both past and future context, which is crucial for accurate emotion recognition.

Deep Learning in NLP The paper showcases the application of deep learning techniques to a specific NLP task (emotion recognition). It demonstrates how customizing neural network architectures for specific tasks can lead to significant improvements over generic implementations.

Transfer Learning in NLP The proposed "sent2affect" method is particularly interesting. It demonstrates how knowledge from a related but distinct task (sentiment analysis) can be transferred to improve performance on emotion recognition. This concept of transfer learning is broadly applicable in NLP, as shown by models like BERT [2], which use pre-training on large corpora to improve performance on various downstream tasks.

The methods proposed in this paper have potential for broader applicability in NLP:

1. The weighted loss function for handling class imbalance could be applied to other classification tasks in NLP where class distribution is skewed.
2. The bidirectional processing technique could be useful in other sequence modeling tasks where both past and future context are important.
3. The "sent2affect" transfer learning approach suggests that knowledge transfer between related NLP tasks (not just datasets) can be beneficial. This idea could be explored for other pairs of related NLP tasks.
4. The use of dropout layers for regularization in LSTMs could be applied to other sequence modeling tasks in NLP to prevent overfitting.

While the paper focuses on emotion recognition, its contributions to handling imbalanced data, leveraging related tasks through transfer learning, and customizing neural architectures for specific NLP problems are broadly applicable. These techniques could potentially improve performance on other NLP tasks such as intent classification in dialogue systems, aspect-based sentiment analysis, or even more distantly related tasks like named entity recognition or part-of-speech tagging.

The paper's approach to transfer learning is particularly interesting when considered alongside more recent developments in NLP. For instance, the BERT model [2] and its variants have shown that pre-training on large corpora can create powerful language representations that transfer well to many downstream tasks. The "sent2affect" method in this paper, while more targeted, follows a similar principle of leveraging knowledge from one task to improve performance on another.

In conclusion, while this paper focuses on the specific task of emotion recognition, its methods and findings contribute to our broader understanding of how to effectively apply deep learning to NLP tasks, particularly in scenarios with limited labeled data and complex, imbalanced class structures.

3 Project description

3.1 Goal

Comparing LSTM (Long Short-Term Memory) and CNN (Convolutional Neural Network) performances across natural language processing, digital signal processing, and computer vision is a crucial, challenging, and potentially impactful research goal.

In natural language processing (NLP), LSTM excels in sequence data handling, language modeling, and text generation, while CNN finds extensive use in tasks like text classification and semantic analysis. Contrasting their strengths in text representation and semantic understanding helps optimize NLP tasks.

For digital signal processing, LSTM and CNN are applied in time series prediction, signal analysis, and sequence classification, each potentially more effective depending on data characteristics. Comparing their efficiency and accuracy in signal processing aids in improving data handling techniques.

Computer vision sees CNN dominate tasks like image classification, object detection, and segmentation, while LSTM holds promise in video analysis and action recognition. Evaluating their applicability in visual tasks advances intelligent vision systems.

By leveraging the strengths of LSTM and CNN, such as combining their capabilities in NLP for precise semantic understanding or using LSTM for temporal data in computer vision, overall model performance can be enhanced. Rapid advancements in deep learning technology make such comparisons pivotal in discovering effective model architectures and training methods to tackle diverse domain challenges.

Exploring hybrid models like LSTM-CNN or CNN-LSTM structures and optimizing parameters across different datasets and tasks further expands research scope and depth. Our approach contrasts with selected papers that focus on specific applications like sentiment analysis in NLP or object detection in computer vision, aiming instead to explore and analyze the versatility and general applicability of deep learning models across multiple domains.

Through these comparisons and research efforts, insightful deep learning solutions can be developed for practical applications across various fields, driving further advancements in artificial intelligence technologies.

3.2 Task

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.

Our aim is to develop a model able to provide a live sentiment analysis with a visual user interface.

3.2.1 Input

For the implementation of our models, I chose to create an open-source web application. The purpose of this platform is to make available to all our emotion recognition models in an intuitive and easily accessible way. It allows users to obtain in real time a personalized assessment of their emotions or personality traits based on the analysis of a video, audio or text extract sent directly via the platform. As our research work was made for the French employment agency Pole Emploi, this tool is initially dedicated to job seekers looking to practice their interview skills : the candidates can train themselves as much as they want to answer the questions, and each time obtain a summary of the emotions/personality traits perceived by our algorithms as well as a comparison with the other candidates.

A page is dedicated to each communication channel (audio, video, text) and allows the user to be evaluated. A typical interview question is asked on each page, for instance : "Tell us about the last time you showed leadership". The audio/video extract (recorded via computer microphones/webcam) or text block can be retrieved once saved, and processed by our algorithms (in the case of the text channel the user can also upload a .pdf document that will be parsed by our tool).

3.2.2 Output

Once the user has recorded or typed his answer, he is redirected to a summary page. In the case of the video interview for example, this assessment allows him not only to know his "score" in each of the emotions identifyed by our model, but also the average score of the other candidates: in this way he can re-position himself, and adjust his attitude at will. I believe that including a kind of benchmark in the analysis helps the user becoming aware of his or her position in relation to the average candidate.

The text and video/audio summaries are slightly different: for the text interview summary, not only I chose to display the percentage score of identifyed personality traits for both the user and the other candidates, but also the most frequently used word in the answer. For the video and audio interview summaries, I displayed the perceived emotions scores of the user and the other candidates. Following are the summary pages for both the text and video interviews.

3.3 Data

I have chosen to diversify the data sources I used depending on the type of data considered.

3.3.1 Natural Language Processing

For the text input, I are using data that was gathered in a study by Pennebaker and King [3]. 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. Students personality scores were assessed by answering the Big Five Inventory (BFI) [4]. 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.

The preprocessing is the first step of our NLP pipeline: this is where I convert raw text document to cleaned lists of words.

Tokenization In order to complete this process, I first need to *tokenize* the corpus. This means that sentences are split into a list of single words, also called tokens.

Deletion 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'.

Reformatting 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'.

Tagging 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.

Padding 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 168 with a standard deviation of 68. In order to make sure I incorporate in our classification the right number of words without discarding too much information, I set the padding dimension to 300, which is roughly equal to the average length plus two times the standard deviation.

3.3.2 Digital Signal Processing

For sound data sets, I am using the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) [5]. "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)."

Following points describre audio signal preprocessing before audio features extraction.

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.

Framing After the pre-emphasis filter, I 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 I suppose that frequencies in a signal are constant over a very short period of time, I can apply Discrete Fourier Transform over those short time windows and obtain a good approximation of the frequency contours of the entire signal.

Hamming After splitting the signal into multiple frames, I 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 $0 \leqslant n \leqslant N-1$ with N the window length.

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. Discrete Cosine Transform (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 $n = 0, N-1$ (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:

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

3.3.3 Computer Vision

For the video data sets, I am using the popular FER2013 Kaggle Challenge data set [6]. 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.

Here are several methods for preprocessing data in computer vision.

Resizing Image Dimensions Image dimensions may vary due to different capturing devices, and models often perform better with specific image sizes. Therefore, resizing images to a uniform size is a common preprocessing method. Here, I crop images to 48x48 pixels.

Image Standardization Standardizing pixel values makes the model less sensitive to variations in brightness and contrast. I normalize pixel values by dividing by 255 and use label encoding to categorize emotions into seven classes (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

Image Enhancement Enhancing images improves quality and information content through methods like rotation, flipping, cropping, and adjusting brightness/contrast. I convert images to grayscale to highlight image features.

Image Smoothing Image data can be affected by noise interference, so it is necessary to perform denoising. This can be achieved using methods such as mean filtering, median filtering, and Gaussian filtering.

Data Augmentation This method generates new training data by transforming original data, effectively increasing the training set's size and diversity to improve model generalization. I augment the original dataset by zooming, rotating, shifting, and flipping images.

Feature Extraction Extracting features such as edges, textures, and colors reduces input data dimensions and computational complexity. Methods include sliding windows, Histogram of Oriented Gradients (HOG), and facial landmark detection.

3.4 Methods

3.4.1 Natural Language Processing

Bag-of-Word approaches In order to run machine learning algorithms I need to convert the text files into numerical feature vectors : I 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, I 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.

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 : I 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.

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

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

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) :

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

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, I 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 I define a context size of 2, I 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 I have multiple input context words, it is assumed that the same set of weights $w_i j$ 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 I 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, I therefore assume that I 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". I 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 I 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, w'i,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. I 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, I 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.

Multinomial Naïve Bayes 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": I 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, I 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, I can look at the occurrence of each word in the expression instead of the entire expression directly. The value I 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, I 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 $x = (x_1, , x_d)$ from a multinomial distribution with N trials and parameter vector $\Theta = (\Theta_1, , \Theta_d)$, 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).

Support Vector Machines This method does not focus on probabilities, but aims at creating a discriminant function $f : X \to y$. 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:

$$f(X) = w^\intercal Tx + b$$

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:

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

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:

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

In the case where I don't make any assumption the linear separability of the training data, I 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:

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

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

LSTM 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!

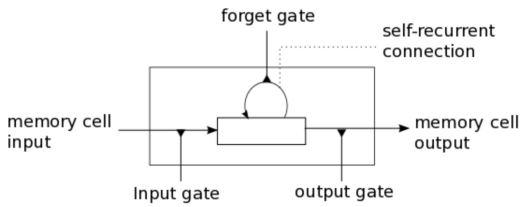

Figure 1: LSTM memory cell

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 cells 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, I 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 b_o 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 I 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 $\widetilde{S}_t$, I can compute 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:

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

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

3.4.2 Digital Signal Processing

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.

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).

Spectrogram The spectrogram of a nonstationary signal is an estimate of the time evolution of its frequency content. It shows on a two-axis diagram 3 parameters: time in x-axis, frequency on y-axis and sound power (in dB) by different color intensity. To compute signal spectrogram you only have to convert DCT coeffcients from power to decibels. The relationship between power and decibels is:

$$ydB = 10 * \log_{10}(y)$$

Log-mel-spectrogram The mel-frequency scale is a quasi-logarithmic spacing roughly resembling the resolution of the human auditory system. To compute log-mel-spectrogram you only have to apply the Mel-spaced filterbank (set of L triangular filters) to the audio spectrogram and get the logarithm of the result.

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, I first divide each frame in K sub-frames of fixed duration. Then I 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.

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, I first divide each frame in K sub-frames of fixed size. Then I 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, I take the Discrete Cosinus Transform (DCT) of the logarithm of all filterbank energies and only keep first 12 DCT coefficients $C^l_{l=1,...,12}$:

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

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

Once having extracted the speech features from the preprocessed audio signal (detailed on previous section) I obtain a matrix of features per audio file. I compute then the first and the second derivatives of each of those features to capture frame to frame changes in the signal. Finally, I 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.

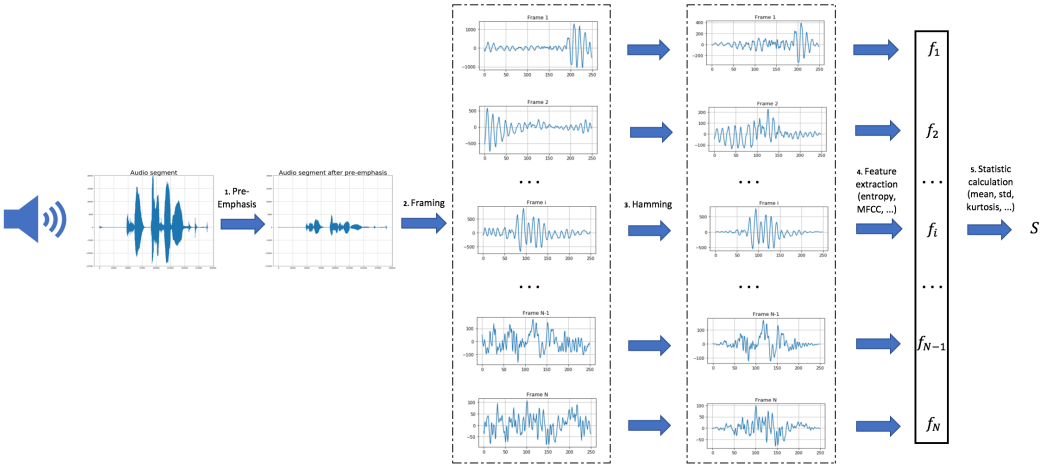

Figure 2: Signal processing and audio feature extraction schema

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).

SVM 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**: $K(x_i, x_j) = (x_i * x_j + \gamma)^d$

Time distributed convolutional neural network Convolutional Neural Networks (CNNs) show remarkable recognition performance for computer vision tasks while Recurrent Neural Networks (RNNs) show impressive achievement in many sequential data processing tasks. The concept of time distributed convolutional neural network is to combine a deep hierarchical CNNs feature extraction architecture with a recurrent neural network model that can learn to recognize sequential dynamics in a speech signal.

Unlike the SVM approach, I will no longer work on global statistics generated on features from time and frequency domain. This network only takes the log-mel-spectrogram (presented in previous section) as input.

The main idea of a Time Distributed Convolutional Neural Network is to apply a rolling window (fixed size and time-step) all along the log-mel-spectrogram. Each of these windows will be the entry of a convolutional neural network, composed by four Local Feature Learning Blocks (LFLBs) and the output of each of these convolutional networks will be fed into a recurrent neural network composed by 2 cells LSTM (Long Short Term Memory) to learn the long-term contextual dependencies. Finally, a fully connected layer with softmax activation is used to predict the emotion detected in the voice.

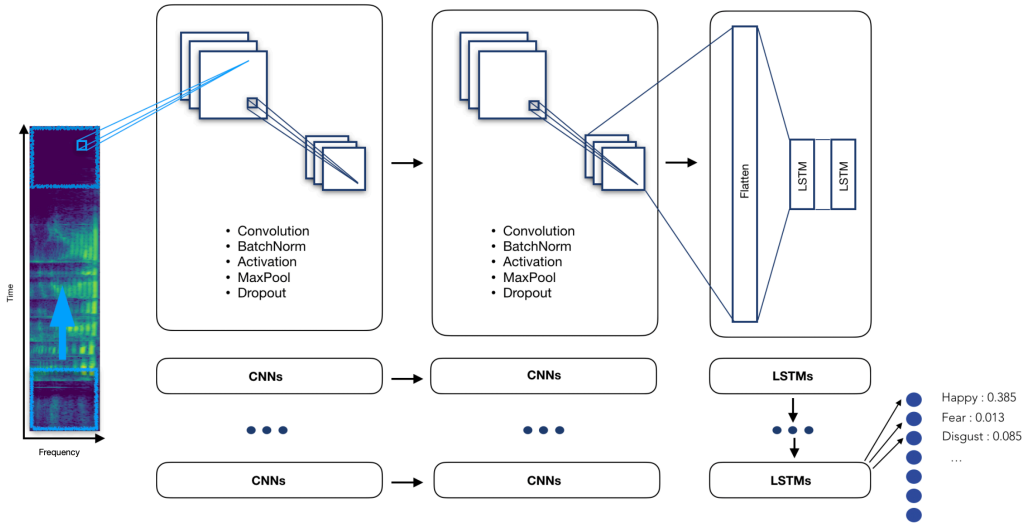

Figure 3: Time distributed Convolutional Neural Network schema

3.4.3 Computer Vision

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.

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 I 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, I am building a convolved output that has a volume. It's no longer a 2 dimensional picture. The filters are hardly humanly understandable, especially when I 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:

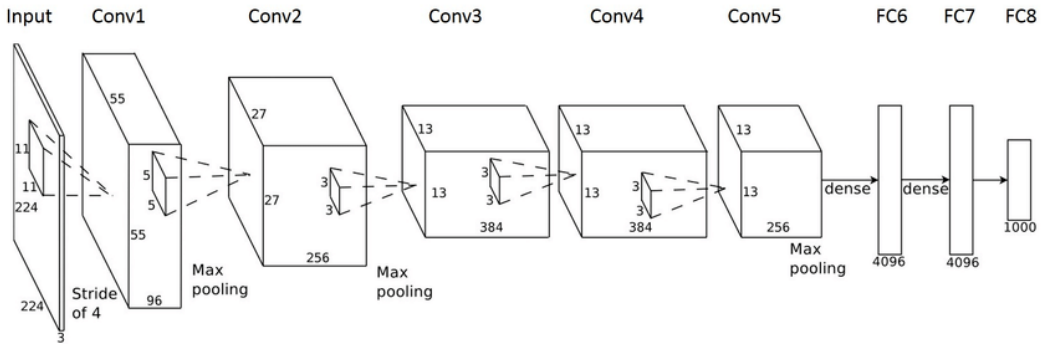

Figure 4: Typical CNN architecture

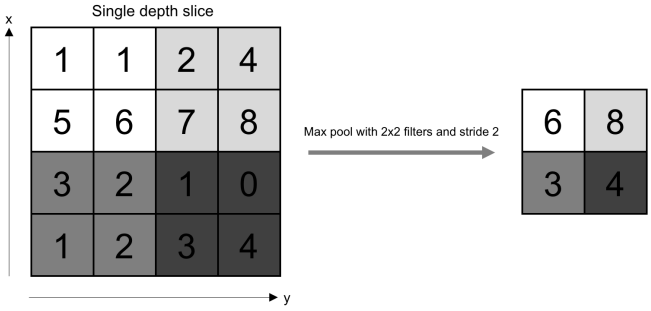

Figure 5: Illustration of a max pooling step

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

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, I apply an activation function (typically ReLU). So far, I have only added dimensionality to our initial image input. To reduce the dimension, I then apply a pooling step. Pooling involves a down sampling of features so that I 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, I 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.

I 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, I 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.

XCeption and Depthwise Separable convolutions Xception is a deep convolutional neural network architecture that involves Depthwise Separable Convolutions. It was developed by Google researchers. Google presented an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads them to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions. The data frst goes through the entry flow, then through the middle flow which is repeated eight times, and fnally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization.

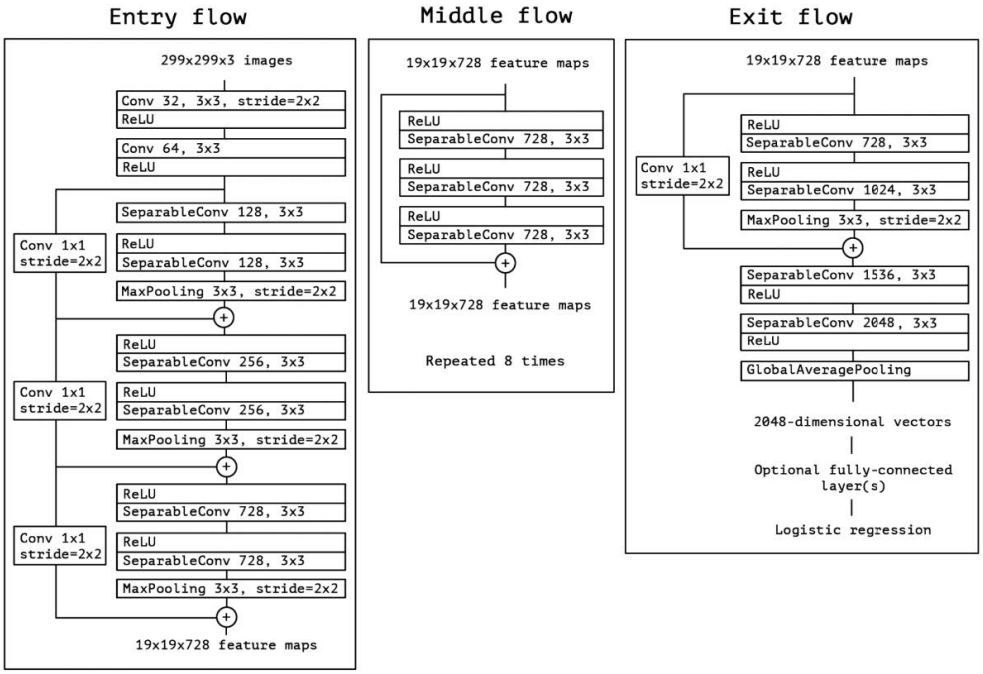

Figure 6: XCeption Structure

Xception architecture has overperformed VGG-16, ResNet and Inception V3 in most classical classification challenges.It is an effcient architecture that relies on two main points:

- Depthwise Separable Convolution
- Shortcuts between Convolution blocks as in ResNet

For the above methods, aside from Word2Vec which I will use as a pretrained model, the rest are implemented by myself. The Time Distributed Convolutional Neural Network is an original creation.

3.5 Baselines

The following baseline methods will be implemented by me.

Natural Language Processing I will use a combination of Polynomial Naive Bayes, Support Vector Machine (SVM), and Bag-of-Words as the baseline methods.

Digital Signal Processing I will combine principal component analysis and SVM respectively for dimensionality reduction and classification as a baseline method.

Computer Vision I will design a simple CNN model as a baseline method.

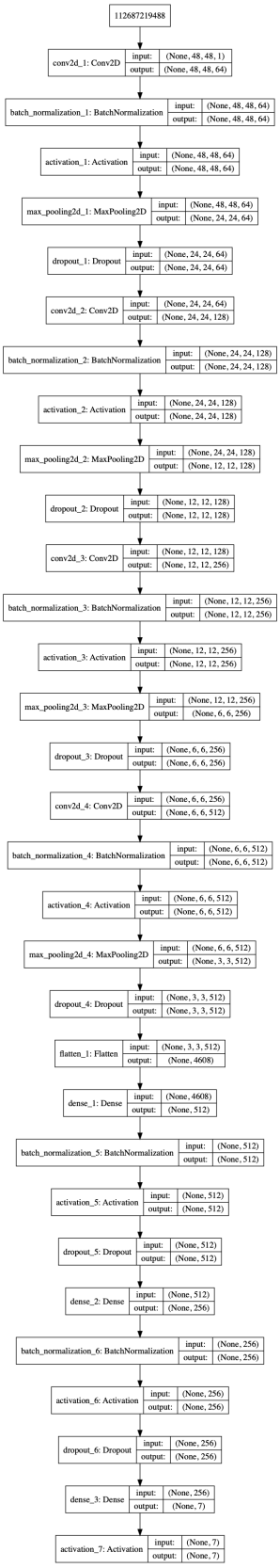

Figure 7: Simple architecture of CNN model

This CNN model, defined within a Sequential container, begins with a series of convolutional layers each followed by BatchNormalization, ReLU activation, max pooling with 2x2 kernel and 'same' padding, and dropout regularization at 25% rate: starting with a 64-filter 3x3 Conv2D layer on 48x48 grayscale input images, followed by 128 and 256-filter layers, and a 512-filter layer with 'same' padding. After flattening the feature maps, it connects to three dense layers of 512, 256, and 7 neurons respectively, with BatchNormalization, ReLU activation, and dropout, culminating in a softmax activation for 7-class classification tasks.

3.6 Evaluation

For quantitative evaluation, I will define **accuracy** as the evaluation metric. Accuracy measures the proportion of correctly predicted instances out of the total instances evaluated. It is widely used in classification tasks to assess the correctness of predictions made by machine learning models.

To benchmark the performance of my method using accuracy, I will compare against several existing scores:

State-of-the-art performance If applying an existing method to a new task, I will compare against the best reported accuracy achieved by current state-of-the-art models on that specific task or dataset. This provides a baseline for understanding where my approach stands relative to the top-performing methods.

Previous results If I am reimplementing or extending a method, I will compare against the accuracy scores reported in the original publication or in similar implementations. This helps validate the reproducibility and improvement (if any) of my method compared to the original.

Regarding qualitative evaluation, in addition to accuracy, I might consider:

Confusion matrix analysis This qualitative evaluation technique can provide insights into specific types of errors made by the model (e.g., false positives, false negatives). It helps understand where the model is struggling and where improvements might be focused.

Error analysis Examining individual predictions that the model gets wrong can uncover patterns or specific cases where the model fails. This qualitative assessment can guide improvements in model design or data preprocessing.

By combining both quantitative (accuracy) and qualitative evaluations, I aim to provide a comprehensive assessment of the performance and capabilities of the method relative to existing approaches in the field.

References

[1] Bernhard Kratzwald, Suzana Ilić, Mathias Kraus, Stefan Feuerriegel, and Helmut Prendinger. Deep learning for affective computing: Text-based emotion recognition in decision support. *Decision support systems*, 115:24–35, 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] James W Pennebaker and Laura A King. Linguistic styles: language use as an individual difference. *Journal of personality and social psychology*, 77(6):1296, 1999.

[4] Oliver P John, Eileen M Donahue, and Robert L Kentle. Big five inventory. *Journal of personality and social psychology*, 1991.

[5] Steven R Livingstone and Frank A Russo. The ryerson audio-visual database of emotional speech and song (ravdess): A dynamic, multimodal set of facial and vocal expressions in north american english. *PloS one*, 13(5):e0196391, 2018.

[6] Pierre-Luc Carrier and Aaron Courville. Challenges in representation learning: Facial expression recognition challenge. *Kaggle Compet*, 2013.