SAPIENZA
UNIVERSITÀ DI ROMA

A Multimodal Approach to Visual Sentiment Analysis for Twitter Data Streams

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Data Science

Candidate

Egon Ferri

ID number 1700963

Thesis Advisor

Prof. Pierpaolo Brutti

Co-Advisor

Dott. Lorenzo Baiocco

Academic Year 2019/2020

Thesis defended on 21 January 2021
in front of a Board of Examiners composed by:

Prof. Pierpaolo Brutti (chairman)
Prof. Casalicchio Emiliano
Prof. Chatzigiannakis Ioannis
Prof. Crespi Mattia
Prof. Lembo Domenico
Prof. Marcucci Juri
Prof. Petti Manuela

A Multimodal Approach to Visual Sentiment Analysis for Twitter Data Streams
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: egon.ferri@gmail.com

Abstract

In this master thesis I will describe the work of four months of work at ELIS consultingLabs. The work is divided into three parts. The first two are projects developed for an Italian large broadcasting company regarding twitter data: one consisted of the automatic detection of celebrities in images, the other in the extraction of text from the image to enhance the performance of the already up-and-running sentiment analysis model. These projects have consisted mainly in the application and put in production of pre-trained-model from Microsoft Azure cloud platform. The last project, motivated by the scarce performance of the old sentiment analysis model, is a more research-oriented approach to multimodal sentiment analysis, from the data collection and labeling to the development of a model using the state-of-the-art theory both for NLP (google BERT) and for computer vision (ResNet CNN).

Dedicated to my mother, Laura

Contents

Introduction

Sentiment analysis, also called sometimes emotion AI or opinion mining, is a set of techniques, tools, methods, and algorithms used to systematically identify, extract, quantify, and study emotions and states and subjective information, such as opinions and attitudes.[1][2]
To analyze, understand, and convey public opinion has always been a matter of primary importance for leaders and politicians. Only in the last twenty years, however, thanks to the internet, people started to share their opinions worldwide, directly writing them in high volume on social media, reviews, or indirectly via traffic volumes. This made possible the development of modern sentiment analysis, one of the fastest-growing fields in IT, useful in an incredible number of domains. For example, from an economic perspective, sentiment analysis could obviously be used to detect customers' feelings about products and services, at the same time opinion mining could be leveraged by policymakers and leaders to understand the feelings about laws and government programs or to detect verbal violence and cyberbullying.

The final aim of this thesis is to propose a model to perform multimodal (based not only on textual data, but on textual data enriched with other modality of data, in this case with images) sentiment analysis, and to apply it to tweets with images about the topic of Italian television programs. The high specificity of this task and its relative novelty hides different pitfalls, requiring research, experimentation, and some degree of creativity, since the proposed model, without the possibility of mimicking something already working and popular, will be the first of his kind (although it will make use of state-of-the-art instruments and ideas published in the last few years for similar tasks).

This work is divided into three parts:

In chapters 1 and 2, the theoretical background needed will be presented. Chapter 1 is an overview of deep learning and neural networks, starting from the very basics

to the architecture of transformers and the concept of transfer learning. Chapter 2 is an overview of sentiment analysis, with a historical introduction, an explanation of the basics, and a review of the state of the art for both textual and multimedia types of data.

In chapter 3 I will expose the project that I developed at ELIS ConsultingLabs under the funding of one of the largest broadcasting companies in Italy. Even if at the beginning the aim of the project seemed the same as the research project, the customer's requirements concerning scopes and instruments made them diverge, as expected, to a certain point, when a project is developed with AGILE methodology. The work resulted in two models:

- A model for VIP recognition, based almost entirely on an already-trained model developed on Microsoft Azure cloud computing platform
- A basic technique to enhance the performance of an already trained model for sentiment analysis using text detected from images embodied in the tweet

In chapter 4 I'll present MmaSAI TVpT, the Multimodal Sentiment Analysis classifier trained on tweets composed by image and text in Italian. This project has required different steps:

- Data downloading
- Hand data labeling
- Development of a pipeline to automatically extract text from images.
- Identification of a suitable approach to train and validate the algorithm (identified in a two-input neural network)
- Exploration and fine-tuning of different text encoding branches
- Exploration and fine-tuning of different image encoding branches
- Exploration and fine-tuning of different fusion layers
- Results presentation

Chapter 1

Deep Learning and Neural Networks

1.1 Introduction

The main theoretical instrument that will be exploited in this thesis is the deep learning framework, in particular in the form of neural network architectures.

Before diving into deep learning, it's necessary to take a step back and say a few words about Artificial Intelligence, Machine Learning, and the relationship between these topics and Deep Learning. We can consider Turing's paper "Computing

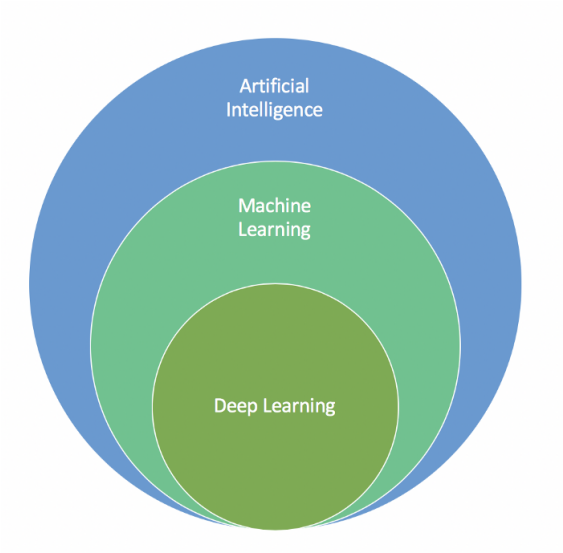


Figure 1.1. Relation between AI, ML and DL

Machinery and Intelligence" [3], in 1950, the first milestone of AI. In this paper, he proposes a simple question: "Can Machines think?", defining the fundamental goal and vision of the field. At its very basics, AI is just the area of research that put the efforts into trying to answer Turing's question in the affirmative, making machines able to think to perform tasks commonly associated with intelligent beings. If this definition seems vague, is because it is, and even if many attempts were made to make it more rigorous, the debate is still very hot. Furthermore, with every step in the state of the art, the bar of what can be considered "Artificial Intelligence" is raised a little more, putting out of the cauldron tasks that once upon a time was supposed to need "human intelligence". In an enlightening paper, "on the measure of intelligence"[4], F. Chollet interrogate himself on the whole concept of "intelligence", arguing that being able to perform (even if with incredible peaks of skill) just one task in one specific environment is not enough to be reputed intelligent at all, and we have to rethink completely the field to take some serious leap forward.

Machine Learning programming, a subfield of artificial intelligence, is a little bit easier to define. The main concept is the change of paradigm from standard programming. In classical programming, rules and data are inserted in the program to get some

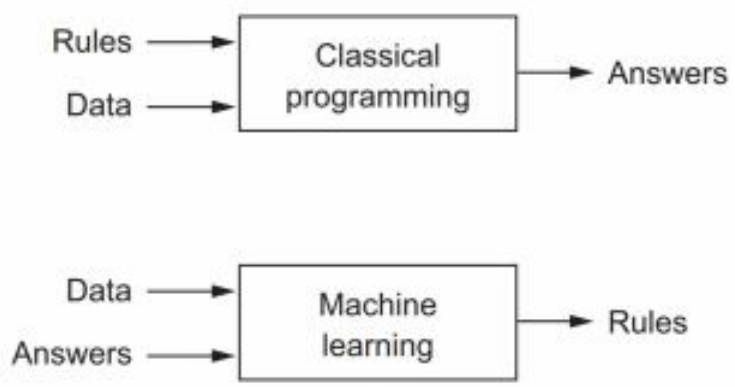


Figure 1.2. Machine Learning paradigm

answers. On the contrary, in ML programming we insert data and desired answers to "train" the model to be able to process other non-labeled datasets. Machine Learning flourished during the nineties, quickly becoming the most popular and successful subfield of AI, exploiting the possibility to train the models on larger data-sets with more computational power.

Machine learning models can be based either on supervised or unsupervised algorithms. Supervised learning is the most utilized and studied type of machine learning

method. Given a labeled training set, a model is built through a training process where predictions are made on the input data and then adjusted when the predictions are wrong. The training process lasts until the model achieves some desired performance or stopping criterion, such as reaching a precise training iteration or low error value. Supervised learning is used mainly for classification and regression tasks. An unsupervised learning algorithm is instead based on non-labeled data. Roughly speaking, this kind of algorithm finds common patterns within the data and helps them to be understood. The common tasks solved by unsupervised learning is clustering.

Deep Learning is a subfield of Machine Learning: a new take on learning representation that lays the foundation for a new way to extract information from data. The word "deep" in "deep learning" refers to the number of layers through which the data is transformed, which is almost always identified as a model called neural network. This term comes from neurobiology, but despite this term being coined taking inspiration from our brain, deep learning models are not a close representation of our brain (see 1.3), but the study of this phenomenon has led scientists to deduce that the great information processing capacity of biological neurons depends on the parallelization of the processes that are distributed among all the neurons in the network, so they tried to build similar architectures to exploit this process. Deep Neural networks are widely used in order to perform almost all the current Machine learning more important tasks (including the two needed in this work: computer vision and sentiment analysis). Moreover, DNNs quickly achieve excellent performance, and they are easy to customize due to their modularity and flexibility.

1.2 Basics

1.2.1 Perceptron

The perceptron [6] is the basic element of neural networks. It mimics the natural mechanism of the human neuron that works through electrochemical signals: the cell body is stimulated by impulses that come from dendrites and, if there is a large enough amount of voltage changes over a short time interval, it can generate an electrochemical pulse. This voltage is then passed to synapses through the axon (see 1.3).

The perceptron is a binary classifier that replicates this structure 1.4 with a simple formula:

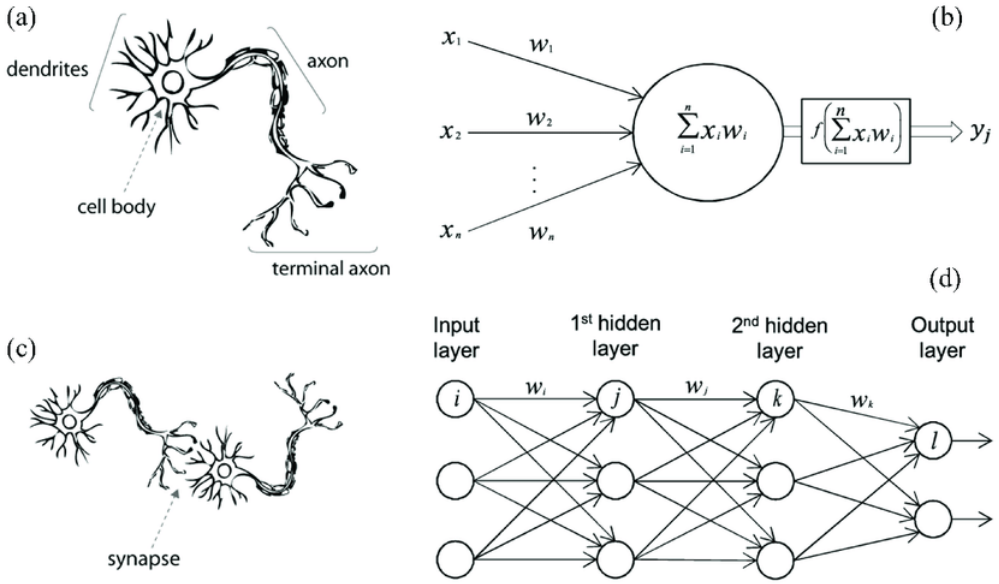


Figure 1.3. A biological neuron in comparison to an artificial neural network: (a) human neuron; (b) artificial neuron; (c) biological synapse; and (d) ANN synapses [5]

$$f\left(x_1, \ldots, x_n\right) = \begin{cases} 1, & \text{if } \sum_i w_i x_i + b > 0 \\ -1, & \text{otherwise} \end{cases}$$

Where each w_i is a real-valued constant, the weight, that determines the contribution of input x_i to the perceptron output, and f is a function that project a scalar into a binary space (like $f(x) = sign(x)$).

This simple neuron is not very useful by itself. It's just a weak binary classifier that cannot tackle non-linearly separable problems. The true potential of neural networks is exploited when multiple layers are interconnected with each other. Even just with two nodes, we can create a non-linear model by the stack of two of them, we just have to be careful to interleave them with some element-wise nonlinearity ϕ, to avoid the collapse of them that would happen otherwise: Without the activation function ϕ:

$$\mathrm{h} = \mathrm{W}\mathrm{x}$$

$$f(\mathrm{h}) = \mathrm{w}^T\mathrm{h}$$

$$f(\mathrm{x}) = \left(\mathrm{w}^\top \mathrm{W}\right)\mathrm{x}$$

With the activation function ϕ:

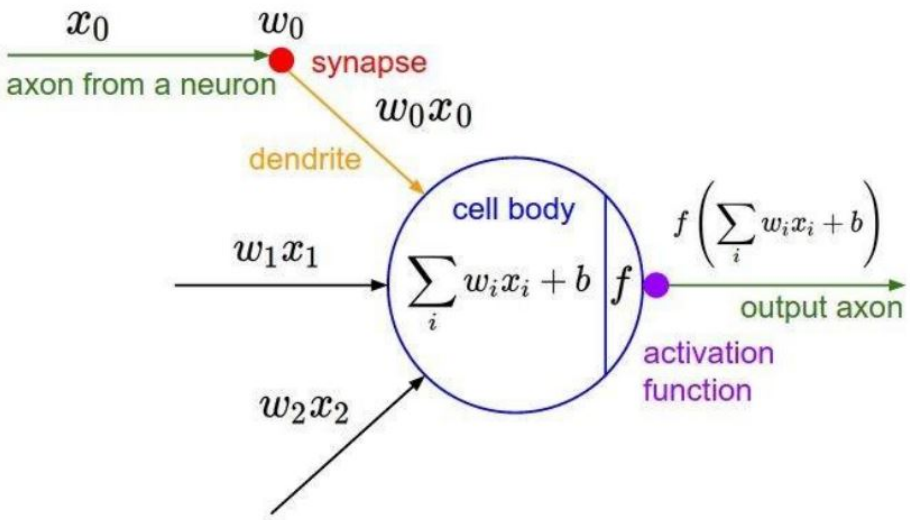


Figure 1.4. A perceptron mimicking a natural neuron

$$\mathrm{h} = \phi(\mathrm{Wx})$$
$$f(\mathrm{h}) = \mathrm{w}^T \mathrm{h}$$

We can expand this concept and keep increasing the number of hidden layers as long as the computational power allows us (although it is not always true that more layers are better), and a basic dense neural network is a simple as that.

1.2.2 Activation functions

As said, we need an activation function for two main reasons: to achieve the nonlinearity, and to compress the outputs to small numbers to help computation. The most common activation functions are explained in 1.6.

Historically the first and more popular has been the sigmoid, which squashes numbers in the range $[0, 1]$ with its easy interpretation as a saturating "firing rate" of a neuron. Despite this clean shape, it presents three main problems:

- Saturated neurons "kill" the gradients, interrupting the learning
- Sigmoid outputs are not zero-centered, and this makes the learning less smooth
- It has an exponential in the formula, and this is computationally expensive

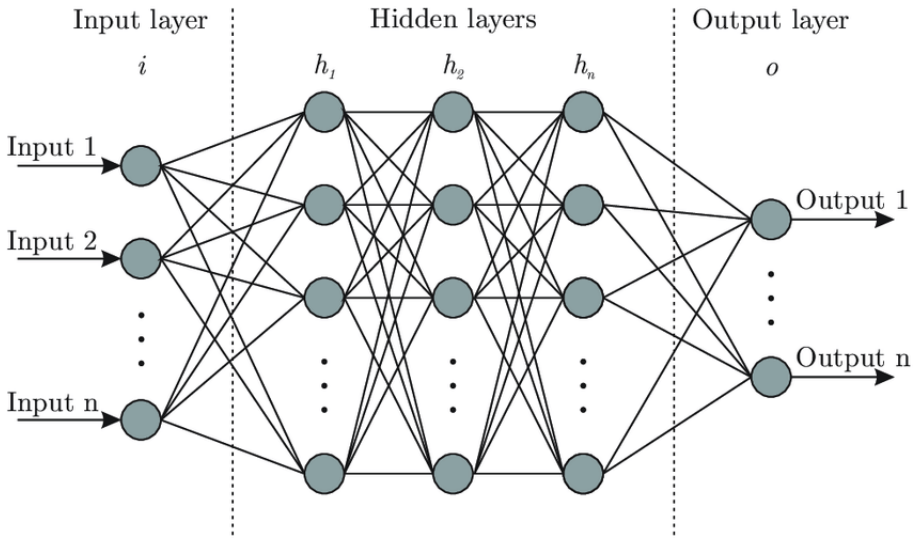


Figure 1.5. A dense neural network

The tanh function only solves the second problem, but the others remain. In contrast, the ReLU (rectified linear unit) keeps the problem of not being zero centered, but it solves the other two. In fact, it does not saturate (at least in the positive region) and is very computationally efficient. It's the best one, empirically speaking, and at the moment is the best choice. Sometimes Leaky ReLU and parametrized ReLU perform slightly better. Their strength is that the neuron is never killed (but it is actually not always better in terms of results). ELU and Max Out are also good but are computationally heavy. -

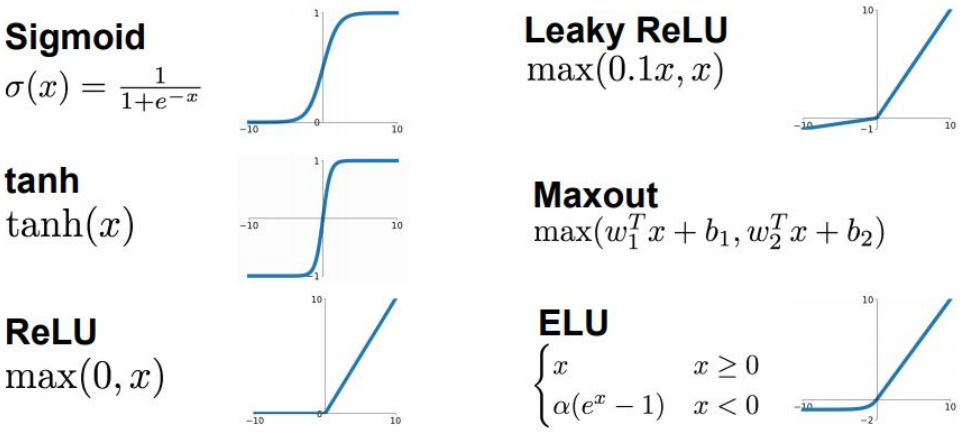


Figure 1.6. Most common activation functions

1.2.3 Loss functions and optimizers

In order to train a deep neural network, two more ingredients are essential: a loss function and an optimizer. The loss function, or objective function, expresses the quantity that will be minimized during the training phase and measures the qualitative progress of the training process. Given a loss function f, and a vector of training x data the optimal vector of weights w that solves the problem can be expressed as:

$$\mathrm{w} = argmin f(\mathrm{x})$$

In order to solve a specific type of problem, a suitable loss function is required. For a binary classification problem, for example, the use of a binary cross-entropy loss function is recommended; for a multi-class classification problem, a categorical cross-entropy is required; for a regression problem, a mean-squared error loss function is suitable, and so on.

The optimizer plays the main role in solving the problem and gradient descent is the dominant method in deep learning problems. The gradient descent is a method to minimize the cost function $J(\theta)$, which is parametrized by a model's parameter $\theta \in \mathrm{R}^d$, by updating the parameters in the opposite direction of the gradient of the cost function $\nabla_\theta J(\theta)$ with respect to the parameters θ. The hyperparameter η called learning rate determines the size of the step taken in the descending direction to reach the local minimum of the cost function. There exist many variants of the gradient descent algorithm. The vanilla gradient descent, computes the gradient of the loss function with respect to the parameters θ for the entire dataset. The updating rule is represented by the following equation:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

The Stochastic Gradient Descent (SGD) is a variant of the vanilla gradient descent, with the difference that it performs parameter updates for one element of the dataset each time. It is represented by the following expression:

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^i; y^i\right)$$

This is obviously a lot faster, but too unstable to converge to a reliable global optimum, so it's usually used a compromise between the two, in the form of the Mini-Batch Stochastic Gradient Descent, expressed by the following equation:

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{i:i+n}; y^{i:i+n}\right)$$

This variant takes the best from the two previous ones and updates the parameters of each mini-batch of the n training examples. In this way, we have a good compromise between stability and speed.

In practice, more complex optimizers are used on mini-batches. For example AdaGrad, which maintains a per-parameter learning rate that improves performance on problems with sparse gradients, RMSProp also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight, (good performance on online and non-stationary problems), or Adam that combines the benefits of the two, making use of the average of the second moments of the gradients.

1.2.4 Back-propagation

When we consider deep neural networks, we have to think of multi-layer structures in which several hidden layers are interconnected in a cascade. During the training process, the network is considered as a computational graph, in which nonlinearities introduced by the activation functions make the network able to switch on its neurons in an expressive way. In the case of deep networks, directly computing the gradient with respect to each weight individually in a naive way would be very inefficient. The solution is the backpropagation algorithm[7]. The backpropagation algorithm is a method for calculating the gradient through the recursive application of the chain rule. So the neural network model consists of two parts: the forward phase and the backward phase. During the forward phase, given the weight, the computational graph is consumed to calculate the loss. In the backward pass, we travel the graph from the end to the start to get, exploiting the chain rule, the partial derivatives of the weights with respect to the loss function, in order to get the "direction" in which we have to change the value of the weights to minimize the loss function. Then, we modify the weights, and we can start again.

1.3 Convolutional neural network

Convolutional neural networks are a type of network that uses structured tensors as input, specifically images, and through the operation of convolution, from which the name, extracts those that are the characteristics of the dependence of a pixel with respect to neighboring pixels [8]. Despite some theoretical concept have been discussed since the sixties, it's only in 2012 that, thanks to bigger models, more data, GPU computation, and better regularization techniques, CNN achieved the state of the art with AlexNet [9]. From that moment, CNN became the main instrument

to tackle image-based tasks, and a myriad of papers and architectures have been published.

From a mathematical point of view, the image is seen as a matrix whose values describe the level of intensity of the pixel. In addition, in the usual case of RGB encoding, you can represent the image as a tensor of depth 3, as shown in figure 1.7. The 3D-tensor is the input that is given at the first convolutional level and on which the first convolution operations are performed .

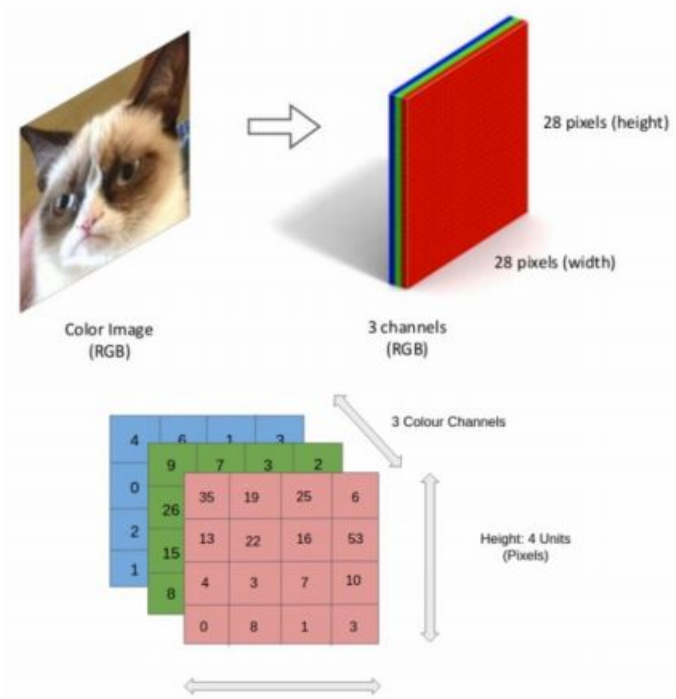


Figure 1.7. Images as h **x** w **x** *channel* vectors

The key concept of this kind of network is the convolution operation. As previously said, convolutions operate over 3D-tensors, called feature maps. These maps have a structured dimension with two spatial axes, for weight and height, and one for the channels also called the depth axis. The "convolution operation extracts patches from its input patches from its input feature map and applies the same transformation to all of those patches, producing an output feature map", which is still a 3D-tensor. Practically speaking, the convolution works by sliding a filter over a small portion of the image and let it slide across the whole image, stopping it at every possible location. Each 3-D patch is then transformed into a 1-D vector applying a tensor product with the same filter, called the convolutional kernel.

Another important concept for CNN architecture is the concept of pooling. Pooling is

important for three main reasons. First, to reduce the dimensions of the feature maps and the number of parameters to learn, and the amount of computation performed in the network. Second, the pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image. Third, to help CNN to elaborate features "hierarchically", in fact, in this way, the deeper layer of the network is focused on larger details.

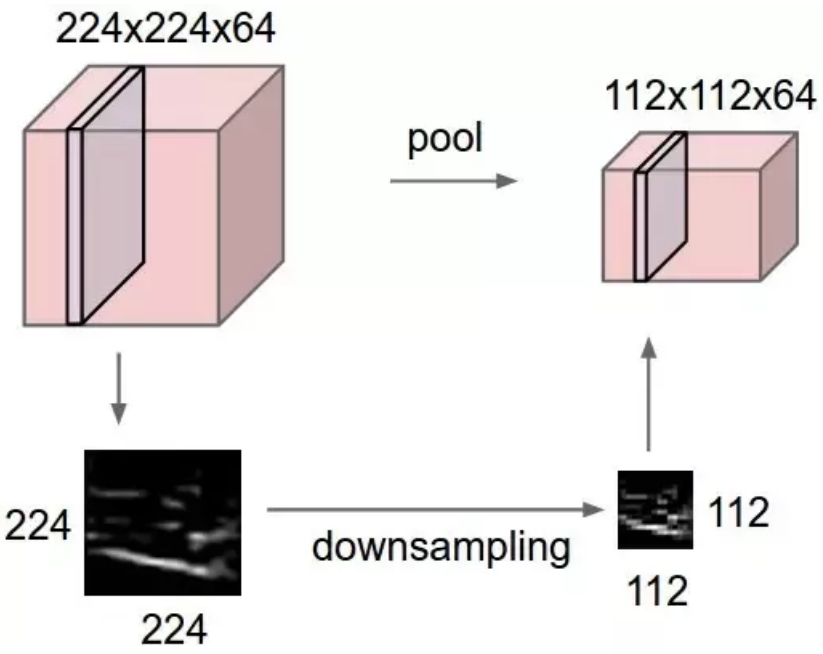


Figure 1.8. Pooling example

Pooling can be applied in various ways. In practice, usually, the simple technique of max-pooling is used, being empirically very efficient.

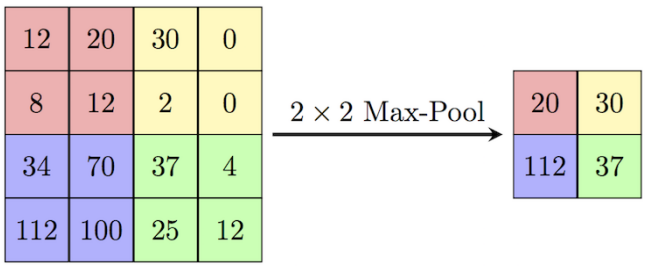


Figure 1.9. Max pooling

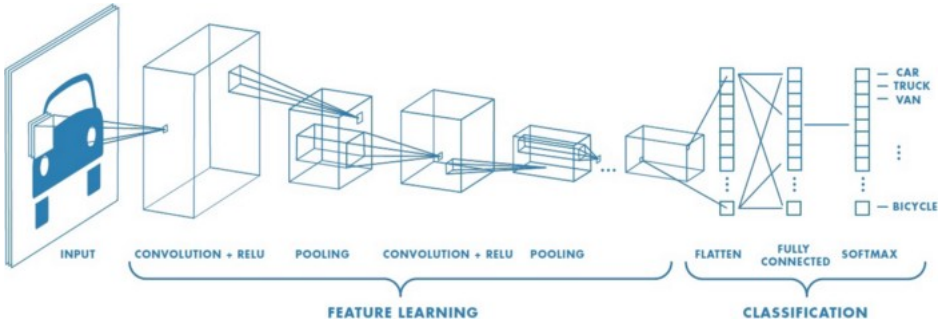


Figure 1.10. CNN full architecture

The final output features map computed by passing through convolutional and pooling layers can be seen as a final "encoding" of the image, and fed to input to some classic Fully Connected layers, that can end the job of classification or of whichever needed task.

1.4 Recurrent neural network

The most common neural network architectures can not correlate information through time. CNNs are powerful but are limited for handling some particular types of sequential data: sequence with significant variant length and sequence with long term dependencies. The Recurrent Neural Network (RNN) [10] was proposed to solve this issue, using internal loops that allow information to persist as a sort of memory.

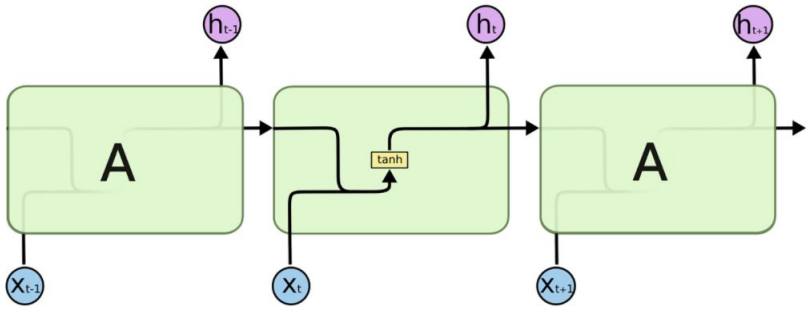


Figure 1.11. Recurrent NN architecture

The RNN architecture presents a network unit A that analyses, at a generic time step t, its x_t input, and generates an h_t output. In addition, unit A passes information encoded in the previous cell from one time-step to the next one. As can be seen further in Fig.2.7, the RNN can be represented like a chain of units, where each unit passes information to its successor. In formulas, taken $x_1, ..., x_t \in \mathrm{R}$ as inputs, the network computes $h_1, ..., h_n \in \mathrm{R}$ outputs, according to the following equation:

$$h_t = \sigma \left(W_{xh} x_t + W_{hh} h_{t-1} + b_h \right)$$

where h_t represents the state at time t for an hidden neuron; σ indicates the sigmoid activation function (e.g. tanh function); W_{xh} is a weight matrix used in the connection between the input and hidden layers; W_{hh} is a weight matrix among recursive connections in a layer; W_{hy} denotes a weight matrix among the hidden and output layers, while the b_h vector is the bias.

The two most common RNN architectures are GRu (Gated Recursive unit)[11] and LSTM (Long Short Term Memory)[12].

The Long-Short Term Memory (LSTM) architecture, unlike RNNs, was designed to leverage long-term dependencies and it is able to retain information (called cell state) over long periods of time, by using dedicated gates inside the single units. The

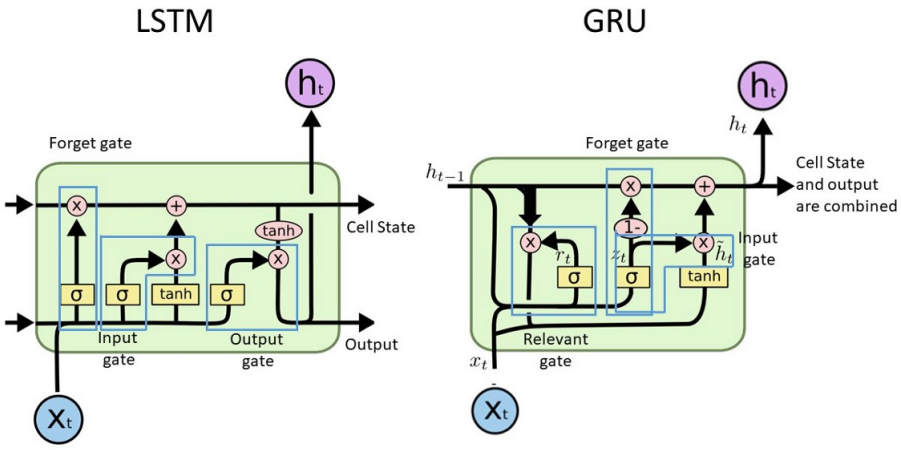


Figure 1.12. Comparison between the two most famous RNN: LSTM and GRu

major innovation of LSTM consists of conveying the cell state directly to the next unit via linear operations, choosing which information to keep or remove through the dedicated gates.

The GRU is like an LSTM with a forget gate, but with fewer parameters, as it lacks an output gate. GRU's performance on certain tasks of polyphonic music modeling, speech signal modeling, and natural language processing was found to be similar to that of LSTM. GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets.[13]

1.5 Seq2Seq models and attention

Sequence-to-sequence models are deep learning models that have achieved a lot of success in NLP tasks like machine translation, text summarization, image captioning, and, as will be discussed in this thesis, sentiment analysis.

Most text information is in a sequence format, e.g., words, sentences, and documents. Seq2Seq is a two-part deep learning architecture to map sequence inputs into sequence outputs. It was initially proposed for the machine translation task, but can be applied for other sequence-to-sequence mapping tasks such as captioning and question retrieval. These models were presented in two independent papers in 2014[14][15].

The encoder reads a sequence input with variable lengths, for example, English words, and the decoder produces a sequence output, for example corresponding Italian words, considering the hidden state from the encoder. The hidden state sends source information from the encoder to the decoder, linking the two. Both the encoder and decoder consist of RNN cells or their variants such as LSTM and GRU.

The context vector turned out to be a bottleneck for these types of models. It made it challenging for the models to deal with long sentences. A solution was proposed in Bahdanau et al., 2014 [16] and Luong et al., 2015 [17]. These papers introduced and refined a technique called "Attention", which highly improved the quality of machine translation systems. Attention allows the model to focus on the relevant parts of the input sequence as needed. Attention is implemented as follows [18]:

First, the encoder, instead of passing only the last hidden state of the encoding stage, passes all the hidden states to the decoder (see fig 1.13):

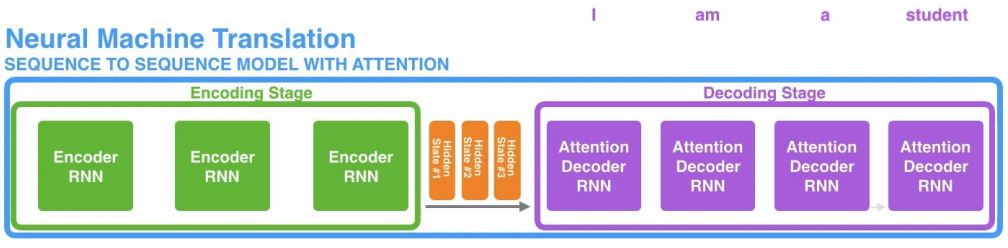


Figure 1.13. Data flow between encoding and decoding stage[18]

Second, an attention decoder goes an extra step before producing its output. In order to focus on the parts of the input that are relevant to this decoding time step,

the decoder does the following:

- Look at the set of encoder hidden states it received – each encoder hidden state is most correlated with a certain word in the input sentence.
- Assign each hidden state a score (explained in a few lines).
- Multiply each hidden state by its soft-maxed score, amplifying hidden states with high scores, and squeezing hidden states with low scores. (see fig 1.14)

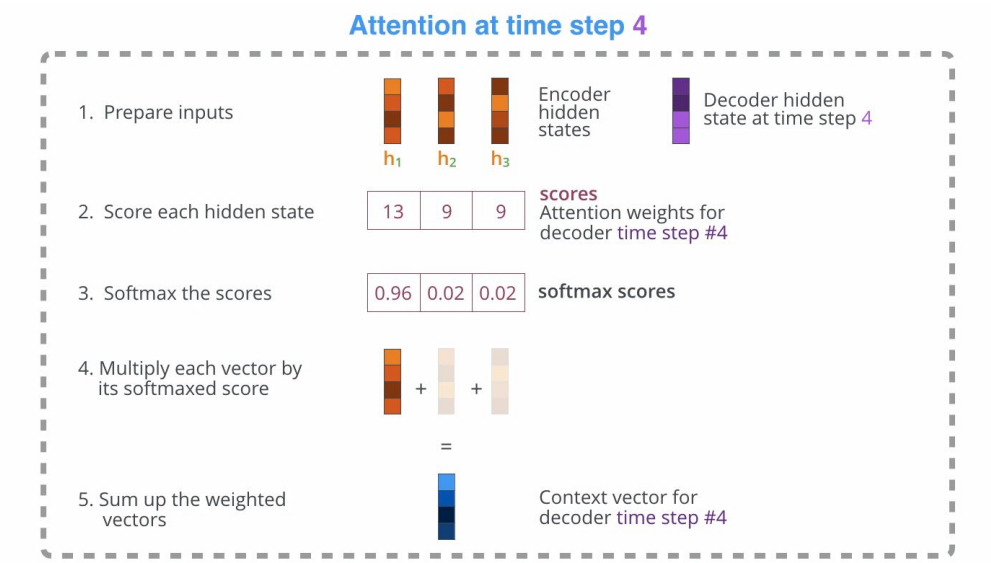


Figure 1.14. Visualization of the building of the "context vector" [18]

This scoring exercise is done at each time step on the decoder side.

In fig 1.15 the whole model with attention process is shown:

- The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
- The RNN processes its inputs, producing an output and a new hidden state vector (h_4). The output is discarded.
- The encoder hidden states and the h_4 vector are used to calculate a context vector (C_4) for this time step. Then, the vector h_4 and C_4 are concatenated and passed through a feed-forward neural network (trained jointly with the model).
- The output of the feed-forward neural networks indicates the output word of this time step.
- This process is repeated for the next time step.

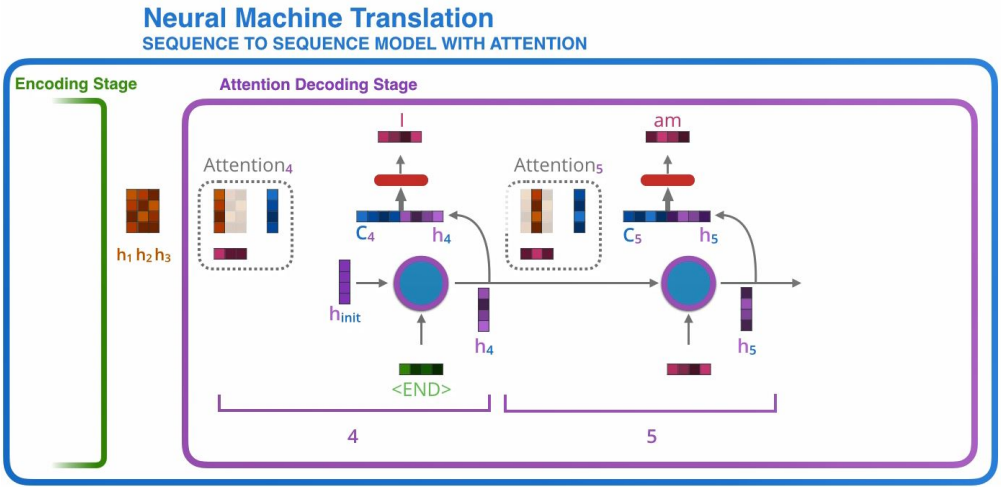


Figure 1.15. Visualization of the whole model [18]

1.6 Transformers

The Transformer neural network architecture proposed by Vaswani et al. in 2017 [19] marked one of the major breakthroughs of the decade in the NLP field. Transformers use the new concepts of self-attention and positional-encoding, going beyond the seq2seq recurrent neural networks by building a new architecture that, in contrast to RNN, does not need to process the sequences of data in order, thus allowing the structure to be parallelized in order to train the models in a more efficient way with less effort in computational view.

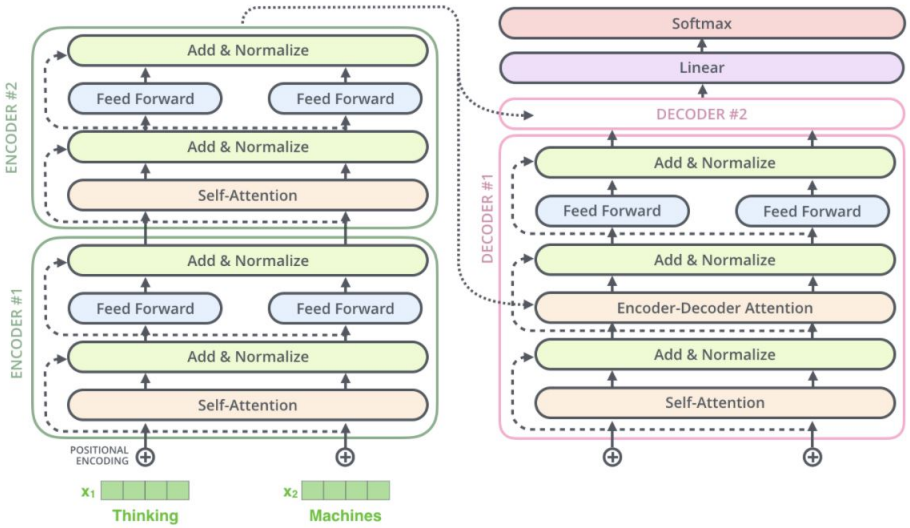


Figure 1.16. Transformer architecture [20]

1.6.1 Self-attention

The key idea to make transformers possible is self-attention. The self-attention formula is:

$$\text{Attention } (Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

and it's implemented as follows (see 1.17):

- Three vectors from each of the encoder's input vectors (in this case, the embedding of each word) are created: a Query vector, a Key vector, and a Value vector. These vectors are created by multiplying the embedding by three matrices trained during the training process.
- A score is calculated for each word. We need to score each word of the input sentence against this word. The score determines how much focus to place on other parts of the input sentence encoding a word at a certain position. The score is calculated by taking the dot product of the query vector with the key vector of the respective word. So, processing the self-attention for the word in position 1, the first score would be the dot product of q_1 and k_1. The second score would be the dot product of q_1 and k_2 and so on and so forth.
- Divide the scores by the square root of the chosen dimension of the key vectors. This leads to having more stable gradients.
- Pass the result through a softmax operation. Softmax normalizes the scores so they're all positive and add up to 1.
- Multiply each value vector by the softmax score. The intuition here is to keep intact the values of important words we want to focus on and squeeze the others
- Sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for a given word).

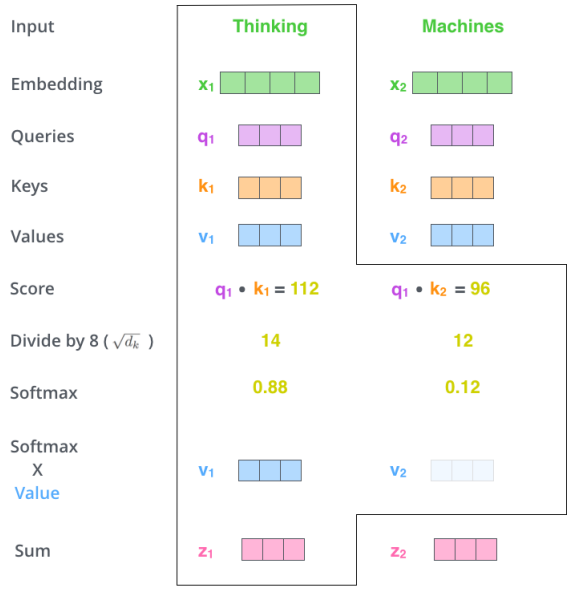


Figure 1.17. Illustrated self-attention [20]

1.6.2 Multi-head attention

Another new concept of the transformer architecture is "multi-headed" attention, that improves the performance of the attention layer in two ways:

- It expands the model's ability to focus on different positions.
- It gives the attention layer multiple "representation subspaces". With multi-headed attention, we have not only one, but multiple sets of Query/Key/Value weight matrices (the Transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder). Each of these sets is randomly initialized. Then, after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace. This helps to represent different semantic relations.

The same self-attention calculation outlined above is replicated n different times with different weight matrices, to end up with n different Z matrices. Since the feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word), a way is needed to condense these n matrices down into a single one. This can be achieved by concatenating the matrices, and then multiplying them by an additional weights matrix W^{O}.

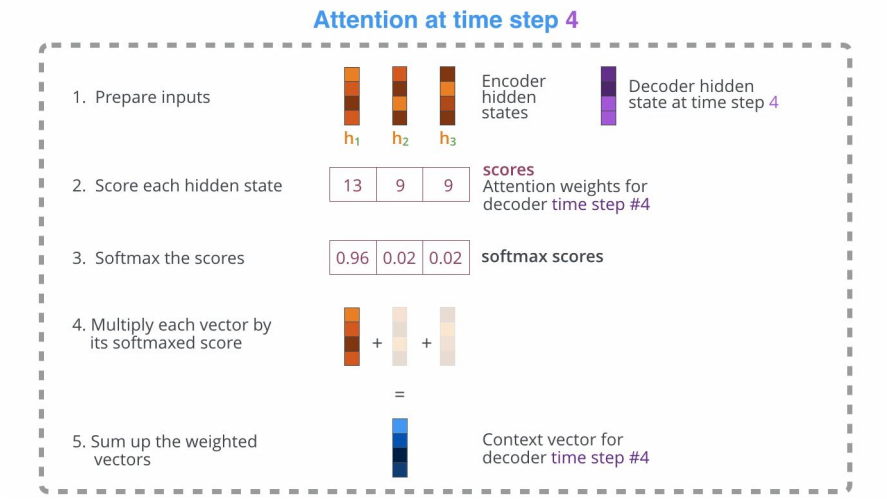


Figure 1.18. Multi-head attention [20]

1.6.3 Positional encoding

Even if the data are not processed sequentially, there is still the need to account for the order of the words in the input sequence.

To address this, the transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word or the distance between different words in the sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

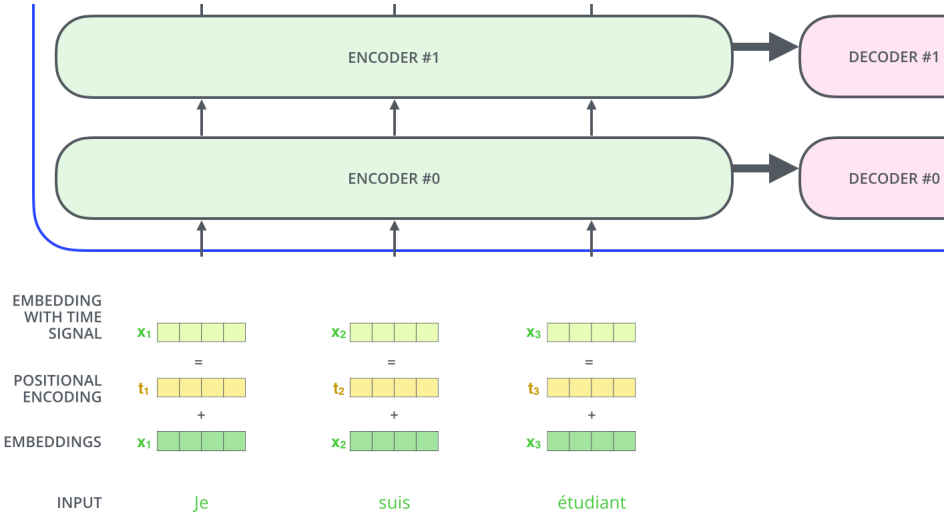


Figure 1.19. Positional encoding [20]

1.7 Transfer learning

The last concept that needs to be presented in order to have a full wealth of skills to develop this thesis is Transfer Learning. The intuition behind transfer learning is to allow AIs to cross-utilize knowledge acquired in different tasks, similar to what humans commonly do (for example, learning to drive a bicycle could help, until a certain point, to speed up the learning required to drive a motorbike).

The key motivation that has driven the development of this tool in the context of deep learning is that most models that solve complex problems need a whole lot of data and getting vast amounts of labeled data for supervised models can be really difficult, considering the time and effort it takes to label data points. However, the idea behind transfer learning is not exclusive of deep learning but can be generalized to each machine learning model (see fig 1.20).

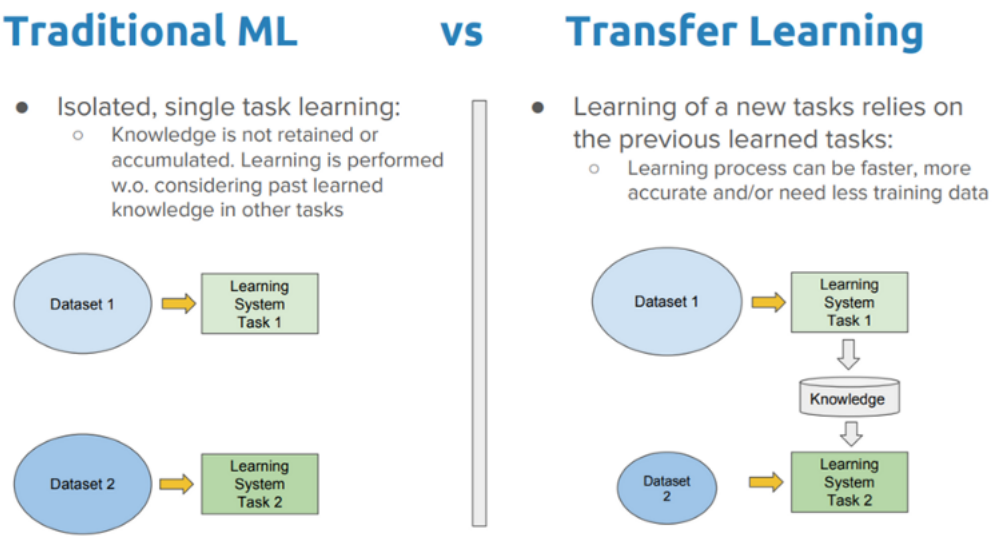


Figure 1.20. Transfer learning in ML [21]

Coming to what is needed for this work, there are various deep learning networks that grant state-of-the-art results that have been developed, tested, and shared for others to use. These works cover different domains, such as computer vision and natural language processing that will be exploited in this thesis.

As shown in figure 1.21, it is a very common choice to take a network that's already pre-trained on a large amount of data of a different set to adapt it to the target task. There are a lot of ways to perform this adaptation. The simplest one is to simply cut the upper layers to modify the final output. For instance, if a trained CNN is utilized without the final classification layer, it will transform images into

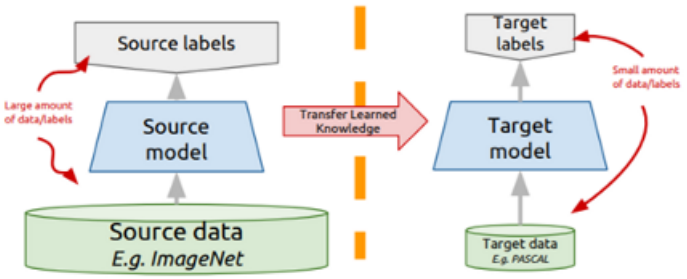


Figure 1.21. Transfer learning in DL [21]

an n-dimensional vector (where n is the number of nodes of the last layer) based on its hidden states, thus enabling the extraction features from a new domain task, utilizing the knowledge from a source-domain task. A little more advanced technique is the fine-tuning of pre-trained models, that impose not only the cut or replacement of final layers, but also the re-training of some of them; since the initial layers have been seen to capture generic features, while the later ones focus gradually on more specific ones, it is possible to "freeze" weights of certain layers while re-training, and fine-tune the rest of them to suit our needs. In this way, knowledge is used as the starting point for re-training. This helps to achieve better performance in less time.

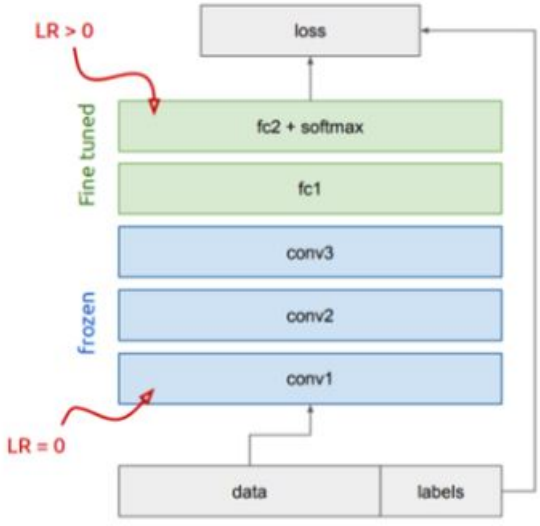


Figure 1.22. Feeze or fine tune? [21]

The choice between freeze or let free for the fine-tuning of each layer is not always easy and is usually driven by empirical trials. However, roughly speaking, is better

to freeze more layers when the train data is scarce and we need to avoid overfitting.

Chapter 2

Sentiment analysis

2.1 History

Some sort of sentiment analysis ancestor can already be found around the '40s [1]. The first papers that tried to analyze systematically the public opinion, in fact, aimed to understand the feelings of the population in countries that suffered during WWII[22][23][24]. Opinion mining continued to evolve, but it's only with the computing revolution that the basis of the modern sentiment analysis could be set, with the help of the Association for Computational Linguistics.[1] Some interesting papers of this period tried to use computers to pool opinion [25]or, for example, to detect subjective sentences from the narrative.[26]

At the beginning of the 21^{st} century, the number of papers slowly began to rise. In 2002, Bang published "Thumbs up? Sentiment Classification using Machine Learning Techniques"[27] a paper in which he describes the use of movie review data and machine learning classification to outperform human-produced baselines, and Turney published: "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews"[28] where he used online reviews of automobiles, banks, movies, and travel destinations to achieve an average accuracy of 74% for recommendations. These two papers can be considered the first two papers of the "modern Sentiment analysis".

As is possible to see from the figure, from that time, interest and research in opinion mining increased year after year, following the growth of data. From that moment, Sentiment analysis became every year a hotter topic, with more and more articles published about methods and application, thanks to the fact that with the evolution of the internet and the rise of social media and web-reviews of products and services,

researchers could manage an enormous amount of data and develop new interesting applications.

The techniques developed in the last years can be divided into two main categories: the unsupervised lexicon-based approach and the supervised machine learning approach. In the very last years, as will be explained in the next paragraphs, the latter took the lead in the form of deep-learning, thanks to the power of neural networks, and at the moment the stat of the art is represented by BERT.

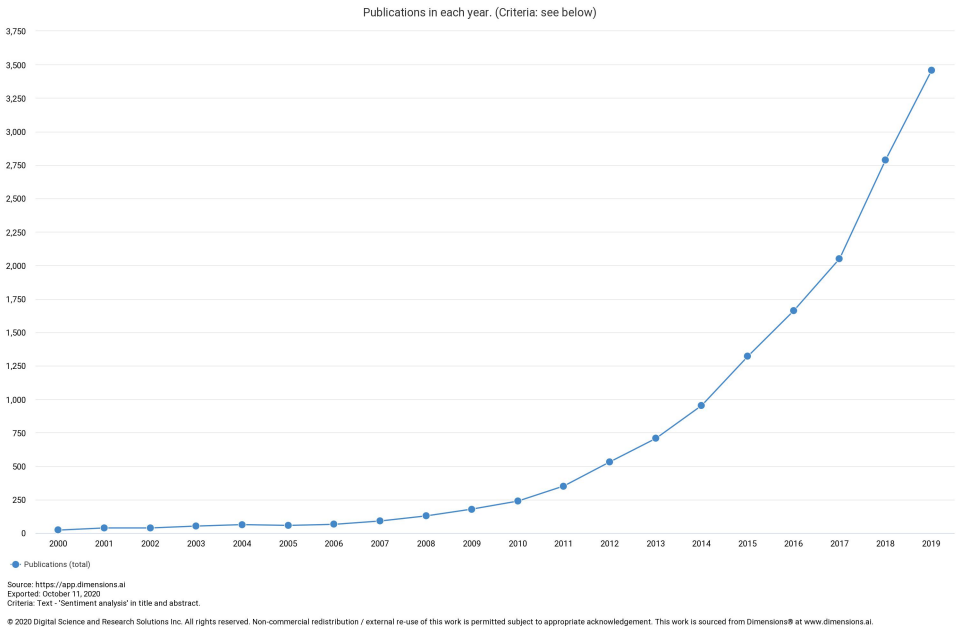


Figure 2.1. Numbers of papers published per year about sentiment analysis

2.2 Basics

Good reviews of the state of the art before the explosion of neural-networks methods and models are "Opinion Mining and Sentiment Analysis"[29] and "Sentiment analysis algorithms and applications: A survey"[30], which are the main resources of this paragraph.

The main division between the SA techniques is between Lexicon based and Machine Learning Approaches. The former needs a human annotation of words to build sentiment lexicons (collection of known and pre-compiled associations term-sentiment), instead the second is automated. This difference is reflected not only in the algorithms but also in the feature selection.

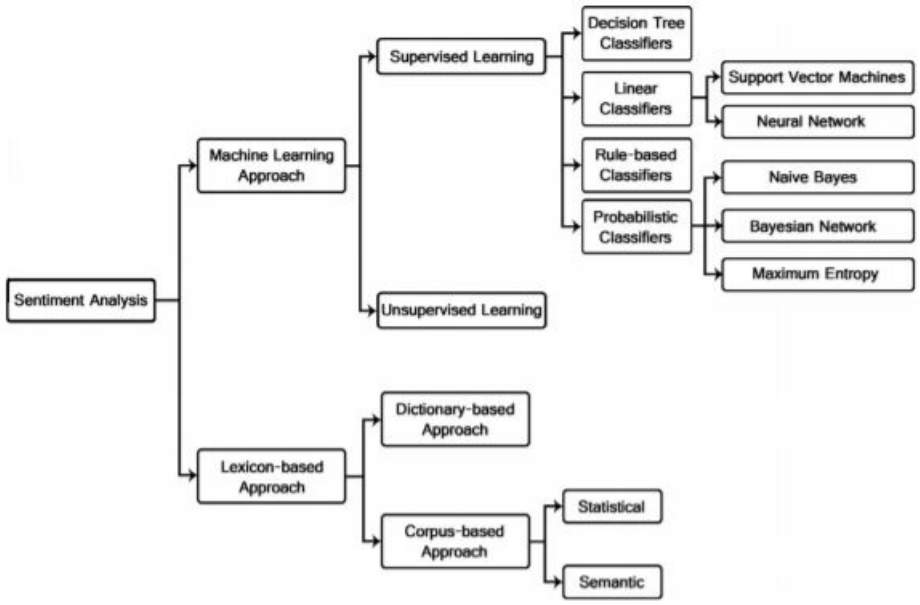


Figure 2.2. Sentiment classification techniques [30]

2.2.1 Feature Selection

The first step of each method is to extract and select text features. Some of the main features, that are highly responsible for the sentiment of the sentences are[31]:

- Terms presence and frequency: The simplest feature, with the frequency that grants that the relative importance is taken into account.
- Parts of speech (POS): The semantic position of the words in the sentences, can be important since some POS carry more semantic value (ex: adjectives).
- Opinion words and phrases: Words and phrases commonly used to carry semantic value (ex: good, bad, like, hate).
- Negations: the appearance of negative usually changes the opinion orientation (ex: not good is equal to bad).

Most of the algorithms treat every sentence as a "bag of words", meaning that the position in which the word appears is not important, but, as will be seen, not all of them. Is also common to preprocess the tokenized (split into parts) sentences, removing stop words (ex: the, at, which...) and reducing the words to the stem (ex: "frightening", "frightened", "frightens" are all reduced to "frighten").

After the tokenization and the preprocessing, a lot of statistical techniques have

been used to assign the polarity (apart from the obvious hand-labeling).

Point-wise mutual information, a formula that comes from information theory[32], is the easiest formal way to model the mutual information between the features and the classes. The point-wise mutual information (PMI) $M_i(w)$ between the word w and the class i is derived from the co-occurrence between the class i and word w. The expected co-occurrence of class i and word w, on the basis of mutual independence, is given by $F(w) \cdot P_i$, and the true co-occurrence is given by $F(w) \cdot p_i(w)$. The mutual information is defined in terms of the ratio between these two values and is given by the following equation:

$$M_i(w) = \log\left(\frac{F(w) \cdot p_i(w)}{F(w) \cdot P_i}\right) = \log\left(\frac{p_i(w)}{P_i}\right)$$

The word w is positively correlated to the class i, when $M_i(w)$ is greater than 0. The word w is negatively correlated to the class i when $M_i(w)$ is less than 0.

In order to get more comparable values across terms in the same category, the χ^2 can be used, since its output is a normalized value:

$$\chi_i^2 = \frac{n \cdot F(w)^2 \cdot (p_i(w) - P_i)^2}{F(w) \cdot (1 - F(w)) \cdot P_i \cdot (1 - P_i)}$$

With n as the total number of documents in the collection, $p_i(w)$ as the conditional probability of class i for documents which contain w, P_i as the global fraction of documents containing the class i, and $F(w)$ as the global fraction of documents which contain the word w.

2.2.2 Lexicon-Based methods

Methods based on lexicons need a hand-crafted one. Obviously, labelling by hand an entire vocabulary would be extremely consuming, in terms of time and energy. The techniques that have been deployed to speed up this process are divided into two families: dictionary-based and corpus based.

In dictionary-based annotation, the first step is to label by hand a conspicuous set of words. After that, the algorithm searches for synonyms and antonyms in a word-bank and propagates the label. The process is repeated until no new words are labeled.

In corpus-based annotation, the first step is the same, but instead of being propagated searching synonyms and antonyms in vocabularies, the words are propagated trough

documents of the corpus, thanks to connectives (ex: "and", "but", "or") and then corrected by hand. [30]

2.2.3 Machine Learning methods

Machine Learning methods instead lighten the human from the tedious hand labeling. Machine learning for text classification is usually supervised (although unsupervised methods have been used for automatic clustering). The supervised Text Classification Problem can be defined as this: We have a set of training records $D_{train} = X_1, X_2, ..., X_n$ where each record is labeled to a class. We have to train a model that, given a test set $D_{test} = X_1, X_2, ..., X_n$ can predict the class label with high accuracy. When only one label is assigned to an instance, we call it a hard classification problem. Instead, when a probabilistic value of labels is assigned to an instance, we have a soft classification problem.[30] We can divide the main classifiers into three main families: probabilistic, linear, and based on decision trees.

Probabilistic classifiers

Probabilistic classifiers use mixture models for classification, giving a probability distribution over a set of classes, rather than only outputting the most likely class that the observation should belong to.

The simplest one is the Naive Bayes Classifier, which computes the posterior probability of the class based on the words of the document, treating it like a bag of words.

$$P(\text{label}|\text{features}) = \frac{P(\text{label}) * P(\text{features} \mid \text{label})}{P(\text{features})}$$

With $P(label)$ as the prior probability of a label, $P(features|label)$ as the likelihood that the given feature set appears given the label, $P(features)$ as the prior probability of occurring of the given set. Since the NBC assumes also that all features are independent, the equation could be rewritten as follows:

$$P(\text{label}|\text{features}) = \frac{P(\text{label}) * P(f1 \mid \text{label}) * \ldots * P(\text{fn} \mid \text{label})}{P(\text{features})}$$

The NB classifier assumes independence of the features. The Bayesian Network model, on the other extreme, assumes that all the features are fully dependent. This leads to a directed acyclic graph with nodes that represent variables and edges that represent conditional dependencies. Therefore, is specified for the model a complete

joint probability distribution over all the variables. In text mining, the computation complexity of BN is very expensive; that is why it is not frequently used. [31]

The Maximum entropy Classifier encodes labeled feature sets to vectors. This encoded vector is then used to calculate weights for each feature that can then be combined to determine the most likely label for a feature set. This classifier is parameterized by a set of *weights*, which is used to combine the joint features that are generated from a feature set by an *encoding*. In particular, the encoding maps each $(featureset, label)$ pair to a vector. The probability of each label is then computed using the following equation:

$$P(fs \mid \text{label}) = \frac{\text{dotprod(weights, encode}(fs,\ \text{label}))}{\text{sum(dotprod(weights, encode }(fs, l))\text{ for l in labels)}}$$

Linear classifiers

Given $\bar{X} = x_1, x_2, ..., x_n$ as then normalized word frequency vector, and vector $\bar{A} = A_1, A_2, ..., A_n$ as the vector of linear coefficients, and b as a scalar; the output of the linear predictor is defined as $p = \bar{A}.\bar{B} + b$, which is the output of the linear classifier. The predictor p is a separating hyper-plane between different classes. To find the optimal separating hyper-plane is the objective of linear classifiers. Among them, one of the most famous and used is the support vector machine. The main principle of SVMs is to determine linear separators in the search space which can best separate the different classes by optimizing in a way that the normal distance of any of the data points from the separator hyper-plane is the largest. A perceptron is just a linear classifier, so in a [30] simple neural network with only one layer it is also cited. As will be shown in the next paragraph, more complex neural networks are been developed later, until they become the state of the art.

Decision tree classifiers

Decision tree classifiers provide a hierarchical decomposition of the training data space in which a condition on the attribute value is used to divide the data [33]. In-text classification, the condition is given by the presence or absence of one or more words. Space is divided recursively until the leaf nodes contain a given number of records.

2.3 State of the art: Neural networks and BERT

In the last ten years, the field of artificial intelligence made giant leaps. The whole industry reached incredible peaks of performance, with state of the art results achieved mainly by deep learning models. Natural language processing and sentiment analysis have not been an exception, and a very accurate classification model demonstrated that the deep learning framework is the best approach at the moment to suit this task. At the same time, judging from the hotness of the topic in research, these techniques are still not mature. In "Sentiment analysis with deep neural networks: comparative study and performance assessment"[34], Wadawagi and Pagi examined the best architectures and hyper-parameters to get the sentiment from text at the state of the art in 2020.

2.3.1 Word encoding

Deep learning models utilized for sentiment analysis and Natural language processing, are heavily dependent on word-embeddings as input features. The idea behind the concept of word-embeddings is to optimize the translation from word into vectors in a way that words with more similar semantic value are closer in the feature space. This method, as opposed to One-hot (or other encoding based only on the relative frequency, like tf-idf) vectors, present another advantage. While One-hot encoding is high-dimensional and sparse, word-embeddings are low-dimensional and dense, the latter being more computationally efficient than the former, which is directly correlated with the vocabulary size. The building of the word-embeddings can be realized via dense Neural Networks or via matrix factorization.

Word2Vec and GloVe

The most famous two are word2vec and GloVe. Word2vec is a shallow two-layer neural network model that allows the creation of encoding of any size starting from selected parameters. The main choice is that it can run based on two architectures: Skip-Gram (predicts the context window for a given word) or CBoW (predicts the given word from a context window). Other parameters are the training algorithm, hierarchical softmax (better for infrequent words) vs negative sampling (better for frequent words, better with low dimensional vectors), the context window size, etc [35]. To understand better the result, an image can help more than a thousand words, and in figure 2.3 we can see that not only words with similar semantic value are closer, but also some slightly more complex "semantic feature" are captured by the model and expressed very clearly.

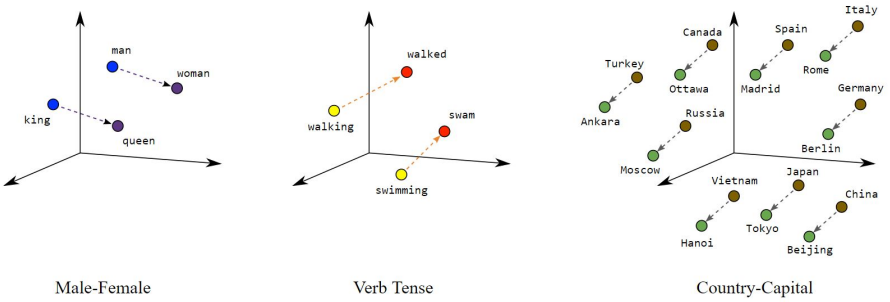


Figure 2.3. Image from developers.google.com

Another word embedding method is Glove (stays for "Global Vectors"). Glove is based on a matrix factorization performed on the word-context matrix. The first step is to construct a large matrix of co-occurrence information: for each word (rows), the occurrence in a large corpus in a given context is taken into account (the columns). Then this matrix is factorized to a lower-dimensional (word x features) matrix, where each row now stores a vector representation for each word via a “reconstruction loss”, a loss that tries to minimize the dimension of the representations while keeping high the portion of the variance in the high-dimensional data explained. [36]

ELMo and Bert

After Word2Vec (2013) and GloVe (2015), in 2018 have been born ELMo (Embeddings from Language Models) and BERT (Bidirectional Encoder Representations from Transformers). The main strength of ELMo is the new concept that the vector assigned to a word (or a larger token) is dependent on context, being a function not only of the word but of all sentences containing it. ELMo representations are also based just on characters, allowing the network to get information from unseen tokens. These word vectors are learned functions of a bi-directional LSTM (a type of LSTM that will be shown in the next paragraph) pre-trained on a large text corpus. When ELMo came out, due to its power and to fact that it could be easily added to existing models, it improved state of the art across a broad range of challenging NLP problems, including sentiment analysis[37]. Last but not Least, BERT represents the state of the art at the moment. BERT and other Transformers-based architecture (like GPT-2 and CTRL) produce context-sensitive embeddings like ELMo. But differently from ELMo, Transformers do not use recurrent neural networks, and so, using a complex architecture that will be shown in the next paragraph, they are free from the need to process all the tokens sequentially. All words in the sentence

are processed in parallel, this approach speeds up processing and solves vanishing gradient problem, allowing them to train more data for more time, and to achieve outstanding peaks of performance.

2.3.2 Basic architectures

Wadawagi and Pagi[34] analyzed the performance of a lot of different neural network architectures. Here are presented some of the best-performing ones.

CNN

Despite being designed for working with computer vision applications, they are widely used in the field of sentiment analysis. In this model, each row represents a k-dimensional word-embeddings (like the previously cited Word2Vec or GloVe).

So a sentence of length n tokens, each with d-dimensional embedding is represented by a $n \times d$ matrix and is fed to the network. A sentence of length n is represented as the concatenation of all associated word-vectors in the sentence with x_i as the embedding of the i^{th} word. In the convolution layer, chosen the size k and an activation function $f(x)$, the filter is applied to generate a convoluted feature vector c_i in a feedforward manner:

$$c_i = f\left(x_{i:[i+k-1]}\right) + \text{bias}$$

The process is pretty standard for a CNN, with an architecture that thanks to activation functions and pooling, processes, in the end, a vector of size $d \times 1$ that can be passed to a classifier (usually a softmax one). CNNs are very good to detect local correlation (a very useful feature for Sentiment analysis) but fails in modeling long-term ones.

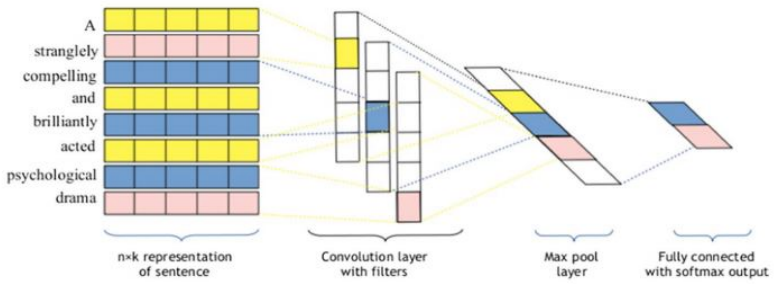
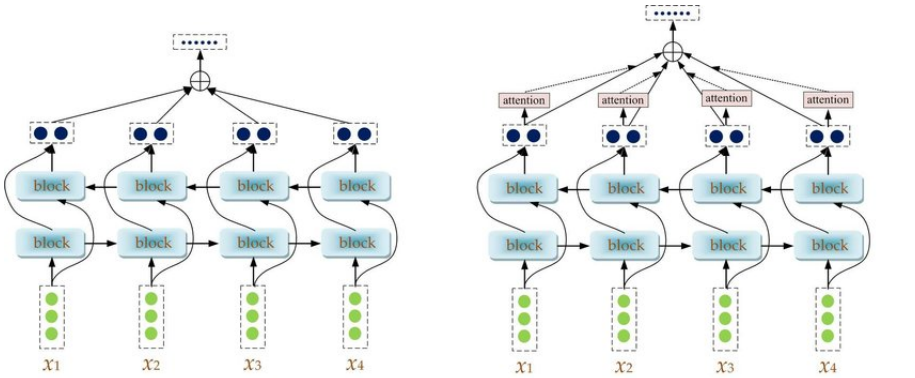


Figure 2.4. Cnn for sentiment analysis [34]

LSTM

As seen in the first chapter, LSTM is one of the best architectures to model seq2seq tasks, however, it is often good to consider both the future and the past contexts in natural language processing tasks. To overcome these limitations of LSTM is proposed the bidirectional LSTM (Bi-LSTM). Bi-LSTM, adding a second layer where the information flows also in reverse temporal order, extend the basic LSTM as depicted in fig 2.4. Therefore, the model can be trained with information coming from the tokens coming before and after.

Although, LSTM and Bi-LSTM can address the problem of long-range dependencies, in practice, handling long-term dependencies is very challenging. So, in order to strengthen the model, the attention mechanism (explained in the first chapter) is plugged in.



(a) A bi-diredctional LSTM structure (b) A bi-diredctional LSTM structure with attention

Figure 2.5. bi-lstm for sentiment analysis [34]

Hybrid network CNN+RNN

Another architecture that performed quite well is the hybrid between CNN and RNN: it extracts local features from the input sequence using CNN and captures long term dependencies from RNN.

2.3.3 Bidirectional Encoder Representations from Transformers

In the last three years a milestone for NLP machine learning models has been marked. It's been defined the "NLP's ImageNet moment" [38],in reference to how the diffusion of model pre-trained on enormous dataset has sped up the development of Computer Vision tasks as we moved from just initializing the first layer of our models to pretraining the entire model with hierarchical representations. In this revolution,

BERT certainly played a major role. The "Bidirectional Encoder Representations from Transformers" (BERT) has been presented in 2019 by Devlin et al.[39]. BERT makes use of the transformers (see par. 1.6) architecture to build an encoder that, as other advanced encoders like ELMo, is trained to take into account the context, but thanks to the new architecture doesn't need to process data in a sequential way, allowing the training process to be parallelized to enhance the computing power.

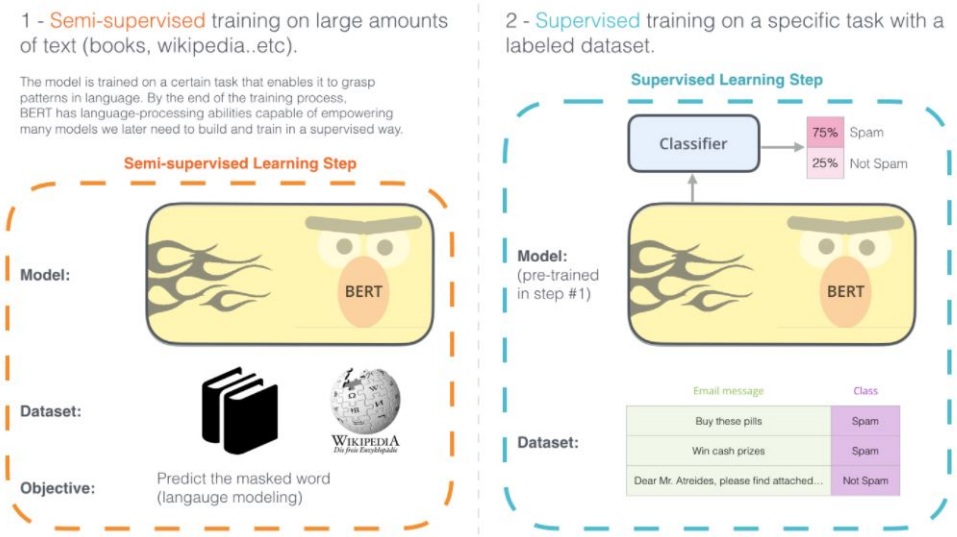


Figure 2.6. The two steps of how BERT is developed [40]

So, the core of the architecture of the model is the same as Vaswani et al. [19], only bigger; BERT comes in two version:

- BERT-Base: 12-layer, 768-hidden, 12-head (to compare it to other models with a similar number of parameters).
- BERT-Large: 24-layer, 1024-hidden, 16-head (to destroy the state of the art using not only ideas but also computing power and amount of data).

What really changes is the pre-training framework. The idea is to split the training phase into two tasks.

- Mask out $k\%$ (with k around 15, not too small to keep training computationally feasible and not too high to give enough context) of the input words, and then predict the masked words.
- To learn relationships between sentences predict whether Sentence B is an actual sentence that proceeds Sentence A, or a random sentence.

This produces the input embeddings as in fig 2.7 that is the sum of token embeddings, the segmentation embeddings and the position embeddings. Since the semi-unsupervised learning that we described above is not necessary for the human hand in the construction of the dataset, almost everything can be fed to BERT. In the paper, the corpus consists of the BooksCorpus (800M words) and English Wikipedia (2,500M words).

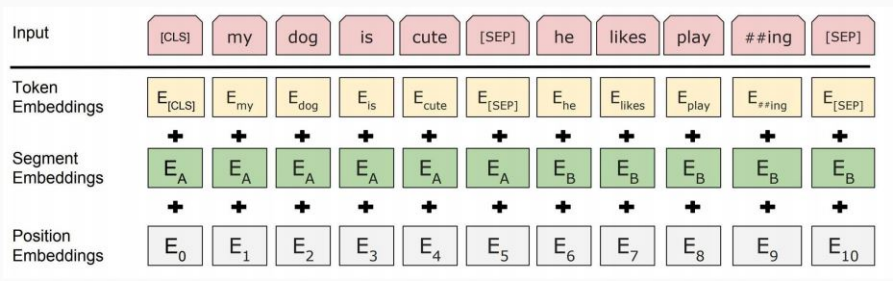


Figure 2.7. Word embeddings coming from Bert [39]

Once you have pre-trained your BERT, you can build on top of the encoding specific NLP tasks with relatively low effort and fine-tune, as long as the language isn't changing (there are also multi-language BERTs, with some small drawbacks in representation power). For example, for text classification, it's enough to take the architecture and change the last layers, from the predict-the-masked-word layer to some fully connected layers, and a softmax on top of that.

2.4 Visual sentiment analysis

The purpose of this project is to develop a visual sentiment analysis architecture. Visual sentiment analysis is a new field, is still not well defined and there is no published official benchmark data set. In this section it is presented a review of interesting previous works that can be used to better understand the problem, the challenges and possible solutions. The field can be divided into "uni media" and "multimedia" sentiment analysis. In the first case, the image or gif is analyzed as a stand-alone, in the second case is used in addition to a text to better characterize the whole data element. Another distinction is the type of classification. There are mainly three types of classification: a three-class positive-negative-neutral, a fine-grained pos/neg in a one to five scales, and a classification based on emotion (like happiness-sadness-anger-fear-love or similar set).

2.4.1 Image or GIF sentiment analysis

Image classification purpose is to classify the sentiment of a standalone image". An interesting pioneer work is "Emotion Detection and Sentiment Analysis of Images"[41] published in 2015 in which Gajarla and Gupta classified Flickr images (automatically downloaded scraping from keywords) using basic the basic approach of fine-tuning the best public architectures (like RESNET, VGG-ImageNet, etc.). They did not reach very good metrics but had the merit of trying for the first time a rough deep learning experiment for this task.

Another interesting work is "Visual Sentiment Analysis for Review Images with Item-Oriented and User-Oriented CNN" [42] published in 2017 by Truong and Lauw, that tried a different variation of Convolutional neural networks to label as positive or negative reviews of different products, trying to take into account the user-factor and item-factor.

A slightly different field, that is worth citing, is the emotional recognition of human face task. This specific task has been explored more and has a benchmark dataset, affect net [**affect net**]. It consists of 1,000,000 facial images collected from the internet, of which about one-half were manually annotated for the presence of seven discrete facial expressions and the intensity of valence and arousal.

The attempt of analyzing the sentiment of gifs also was made [43].

2.4.2 Multimodal sentiment analysis

Multimodal sentiment analysis includes other modalities such as audio and visual data to enrich textual sentiment analysis. It's relatively new because it's only a few years that comments are consistently accompanied by other types of media on social networks and in reviews of products and services.

Similar to the traditional sentiment analysis, one of the most basic tasks in multimodal sentiment analysis is sentiment classification, which classifies different sentiments into categories such as positive, negative, or neutral. The complexity of analyzing multiple types of features to perform such a task requires the application of different fusion techniques.

A pioneer work on the subject is "Convolutional Neural Networks for Multimedia Sentiment Analysis" [44] in which Cai and Xia, in 2015, demonstrate that despite a very simple architecture (see fig 2.8), the information contained in text can be combined with information contained in the image to achieve higher accuracy with respect to both the pieces of data taken alone.

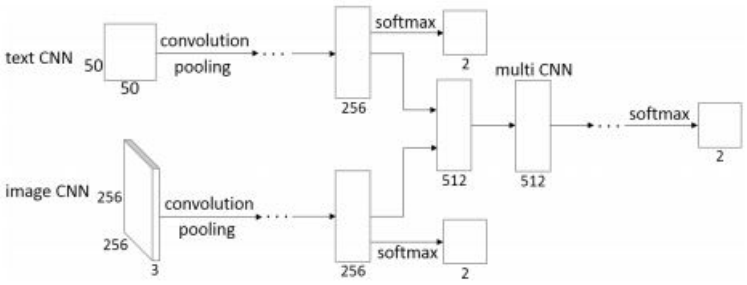


Figure 2.8. Architecture of multimodal sentiment analysis [44]

Another work that shows that the multimodality of comments can be very important to get good results is "Sentiment analysis of multimodal Twitter data" by Kumar and Garg[45]. In their experiment conducted on LGBT verdict of Indian Penal Court (IPC) section 377 in India using hashtag section377, they got an accuracy of 77.63% using only the Image module, 84.62% using only the Text module, and 91.32% with their proposed Multimodal. Their architecture is very interesting see **??**, with the idea of extracting the text from the image to elaborate it with the rest of the comment. The main limit is the fact that it uses some external services as a black box for example Sentibank [46], sentiWordNet, and the non-specified API used to perform optical character recognition.

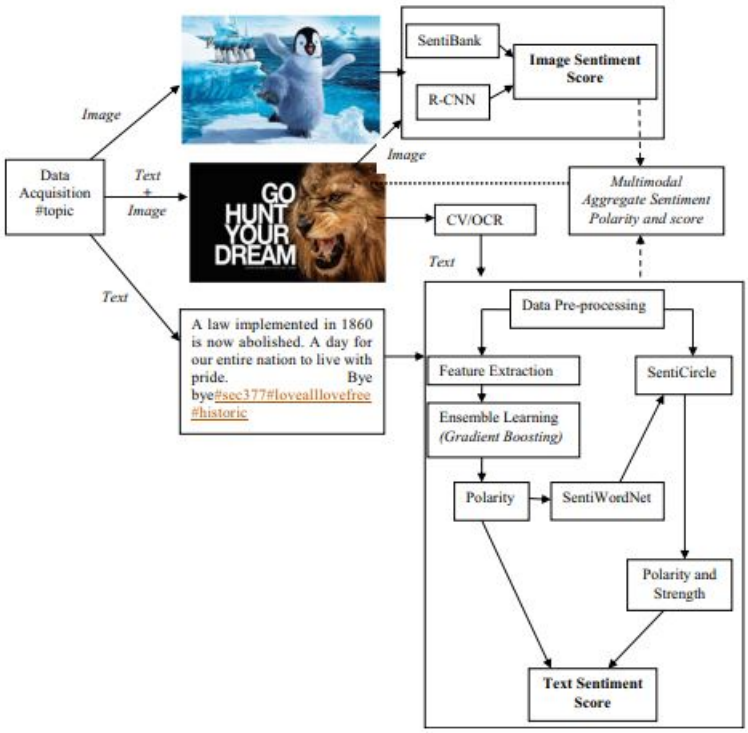


Figure 2.9. Architecture of multimodal sentiment analysis [45]

In "Bi-Fusion Techniques for Deep Meme Emotion Analysis"[47], Gupta et al. present different bimodal fusion techniques (see fig 2.10) to leverage the inter-modal dependency between text (analyzed via bi-LSTM or BERT) and images (analyzed via classic CNN architectures) in memes to classify sentiment and humor. A more extensive study on the effectiveness of different fusion blocks in neural networks can be found in "Gated Multimodal Units for Information Fusion" by Arevalo et al. [48].

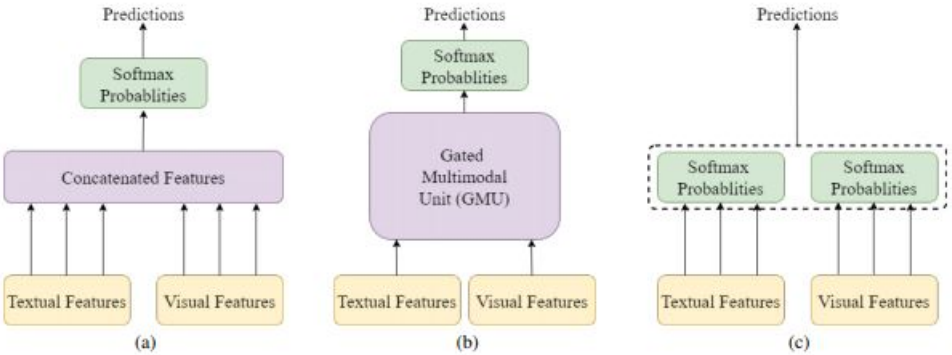


Figure 2.10. Architecture of multimodal sentiment analysis [45]

Chapter 3

An end-to-end real world application

3.1 Introduction

In this chapter, I am going to present the work that I developed at ELIS Consulting&Labs under the funding of one of the largest broadcasting companies in Italy.

The project was organized according to the Agile methodology[49]. The client requested two deliverables :

- An algorithm able to take as input an image and return the list of celebrities in it.
- A solution to integrate the already up and running sentiment-analysis algorithm to increase the performance on tweets accompanied by an image.

Apart from these requests, the client stressed the fact that a smooth system with a clean and simple architecture based on the Microsoft Azure cloud platform was highly preferable.

3.2 Methods and instruments: Microsoft Azure

Microsoft Azure is a cloud computing service for building, testing, deploying, and managing applications and services through Microsoft-managed data centers.[50] The possibilities that it gives are almost infinite, and a complete review would be

out of scope, so I'll summarize just the very basics of things that we have actually used.

3.2.1 Cognitive services: Computer vision API

Computer vision API is useful for a lot of things, such as automatic text extraction or analysis of videos and images in real-time, allowing everyone to perform very easily some tasks like object recognition or image captioning. We used two functionalities:

- "Celebrities Recognition", which recognizes celebrities in an image (though just from a small subset, as explained later)
- "Read", a very powerful OCR that can read text from noisy images.

Azure APIs can be called in different ways: the native way to interrogate them is via REST, but client library SDKs are also available in different programming languages (we used python), and some features are also implemented as logic app connectors, which will be further analyzed in the next paragraphs.

3.2.2 Cognitive Services: Face API

The Face API is similar to the previous one, but, as indicated by the appellative, it is more focused on human faces. The features include: face detection that perceives faces with different attributes in an image; person identification that matches an individual in a pre-compiled private repository of up to 1 million people; emotion detection, recognition, and grouping of similar faces in images. The main functionalities that we exploited are:

- Emotion detection
- The creation of a repository of people to recognize them in images.

3.2.3 Logic Apps

Azure Logic Apps is a cloud service that allows to schedule, automate, and orchestrate tasks, and workflows helping to integrate apps, data, systems, and services across enterprises or organizations. Logic Apps provides a simplified path to design and build scalable solutions for the integration of apps, data and systems.

Every logic app workflow starts with a trigger, which fires when a specified event happens.

Whenever the trigger fires, the Logic Apps engine creates a logic app instance that runs the actions in the workflow. These actions can be data conversions and workflow controls, such as conditional statements, switch statements, loops, and branching.

3.3 Vip recognition

The first task we had was to build an algorithm that, given an image, could return as output the list of celebrities identified in it. This could serve the client both as a standalone solution (to monitor the volume of engagement referred to a VIP) and in pair with other services (for example, with a sentiment analysis algorithm to analyze t1he sentiment toward a character). The customer requested to solve this problem using the Azure cloud platform, both for economical reasons (they already pay it and they want to get value from it) and practical reasons (to be able to integrate our solution with the environment already in production). Setting goals with the client, it came out that it was also important to deliver a solution that included the possibility to update dynamically the list of recognizable celebrities in a simple way, minimizing the effort from the user.

3.3.1 Architecture

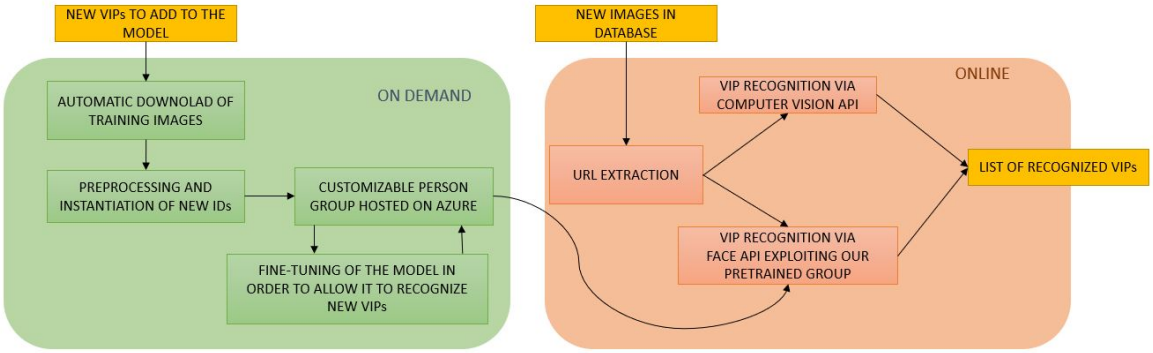


Figure 3.1. vip recognition pipeline

The architecture is quite simple. There are two main blocks: one works online while the other is on demand.

The former runs whenever a new tweet with media is inserted in the database. It extracts the URL of the image, or of the first frame if it's a video or a gif, and sends

the request to two different azure cognitive services.

To computer vision API, which has a function to directly perform celebrities recognition, this is quite good as it provides a starting point detecting with high accuracy worldwide celebrities, but from a limited list. To Face API, that has a function to detect faces in the image and another one to query a pre-trained group to check if detected faces belong to one of the people indexed in the group. Then it performs a simple join and outputs the merged list.

The second main blocks operates on-demand: whenever a new VIP is manually added to the list, the algorithm automatically:

- Searches images of that VIP on Google
- Discards any image not containing one and only one face.
- Instantiates a new ID on the pre-trained group and adds to it all the faces.
- Fine-tunes the model in order to allow it to recognize new VIPs.

3.3.2 Testing environment set up

To assess if our idea was viable, especially to check if the automatic choice of training images could lead to satisfying results, we set up a test as this:

- We downloaded 60 images per VIP.
- We used 30 of them as a training set (without cleaning them in order to simulate the proposed production environment).
- We used 5 of them, chosen manually between the other 30, as a test set (chosen to assure that in each of them there was the considered VIP, alone).
- With 600 VIPs, we had 18.000 training images and 3.000 test images.

The testing list was given by the client, but we had to modify it a little:

- We removed all the groups (comedy couples, music bands,...)
- We modified the name of the VIPs called by the nickname that could be confused. For example, searching for "Gazelle" (an Italian indie singer-songwriter) we got only photos of gazelles (animal) and Adidas shoes (see fig 3.2, 3.3). However, a good policy to avoid this problem is to always feed the algorithm with the VIP's name and surname (or name-nickname-surname).

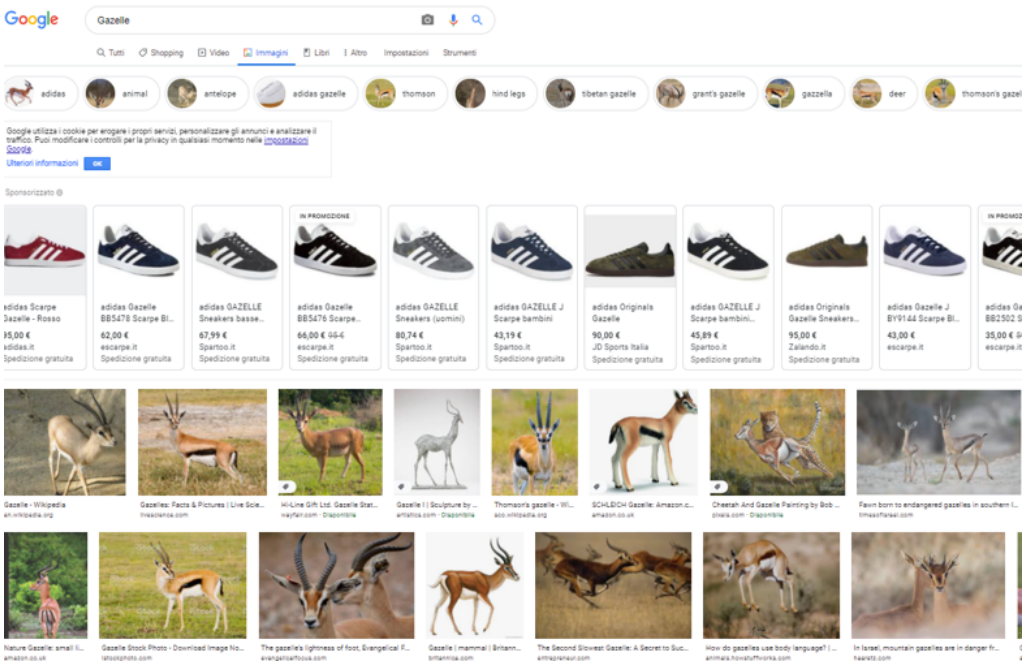


Figure 3.2. Gazzelle on google images https://rb.gy/lgusvr

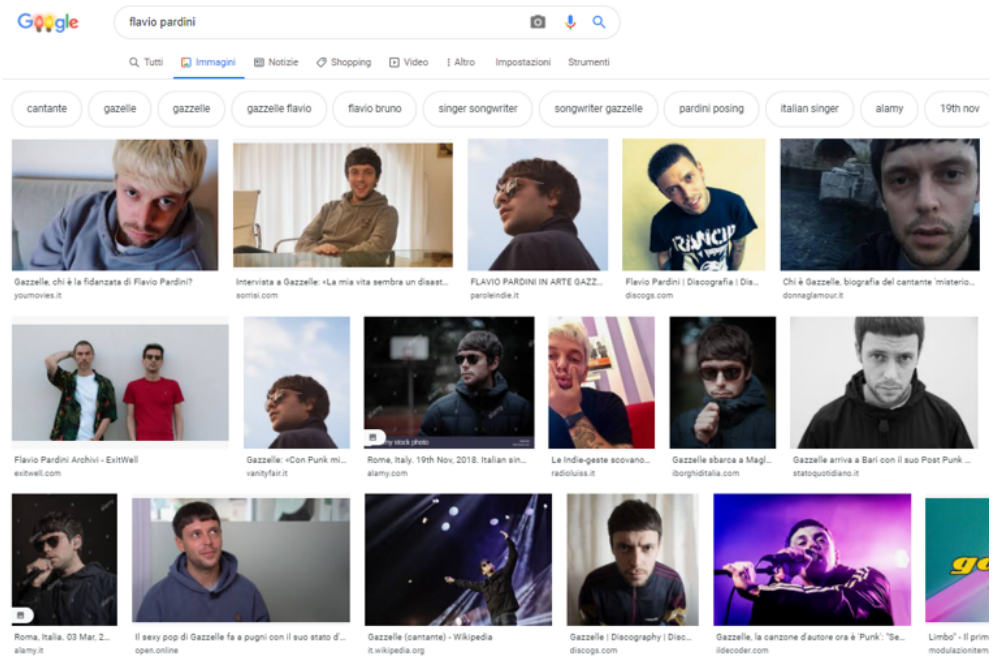


Figure 3.3. Flavio Pardini on google images https://rb.gy/sj9tha

3.3.3 Result

The result of this test showed a very good performance. We got an overall accuracy of 95% and almost every VIP considered has been recognized 4 or 5 times over the 5 photos. (As it is possible to see in fig 3.4, 3.5)

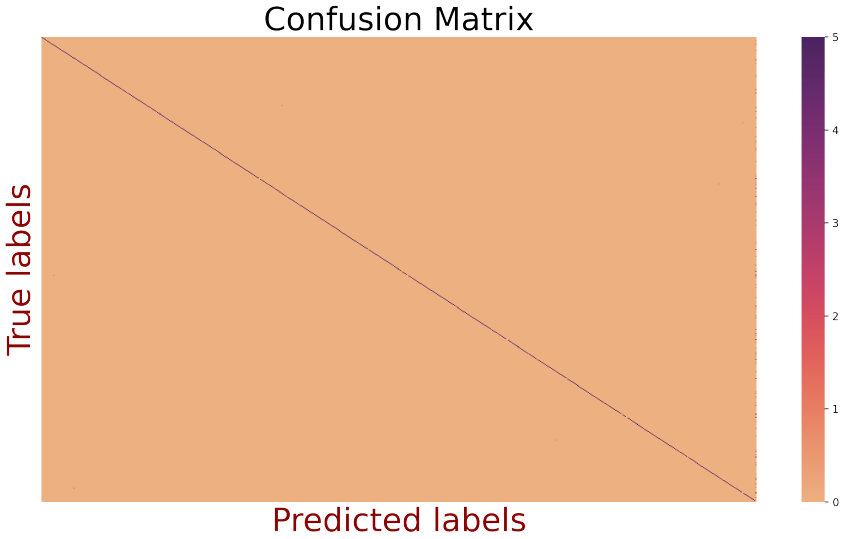


Figure 3.4. Whole confusion Matrix

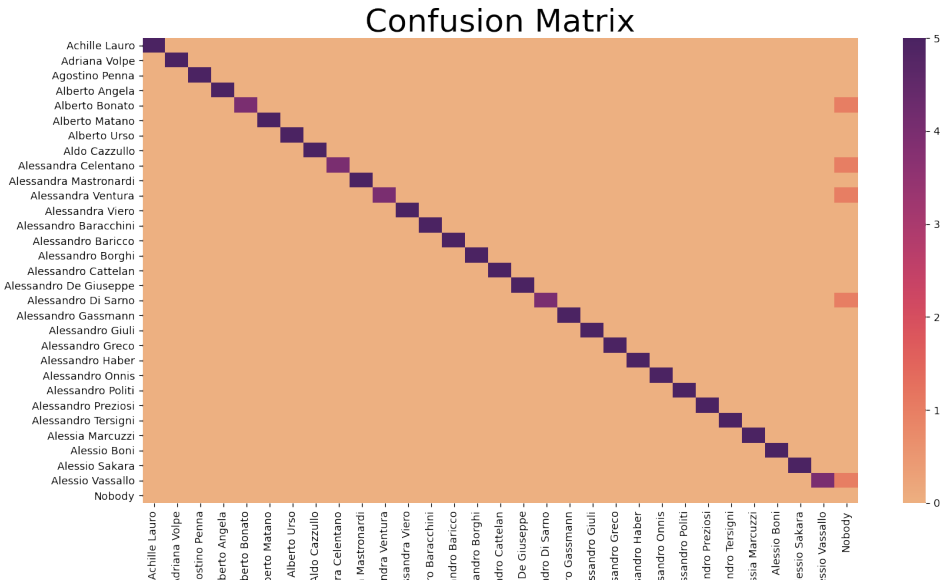


Figure 3.5. Zoom-on confusion Matrix

It's important to know that this test is slightly different from the effective environment of production; the images of the test-set are generally "easier" because in the "wild" we will have more photos in which the VIPs are occluded, turned, caught with heavy facial expressions, etc. This is because a test set like this is both easier to build and more adept to help us analyze the part of the performance over which we have some power. In fact, as will be shown in the next paragraph, a big portion of errors is given by face-detection problems that depend on the performance of Azure face detectors. What matters to us, instead, is that, given that the face is detected, the VIP is correctly parametrized to be recognized. The accuracy is quite high considering the fact that we could manage to totally avoid human supervision in the creation of the train set.

3.3.4 Error analysis

The great majority of errors, as said, came from missing face detection, dependent only on Azure pre-trained algorithm (see fig 3.6):

- Occluded faces.
- Faces distorted due to heavy facial expressions.
- Turned faces.

Figure 3.6. Occluded, turning, distorted faces

But we experienced also errors depending on the train set. Some of them on single images(see fig 3.7):

- VIP not recognized because in that photo it was disguised.

- VIP not recognized because the photo comes from a period of VIP lives in which it was very different from his appearance in the stage of life of maximum stardom.

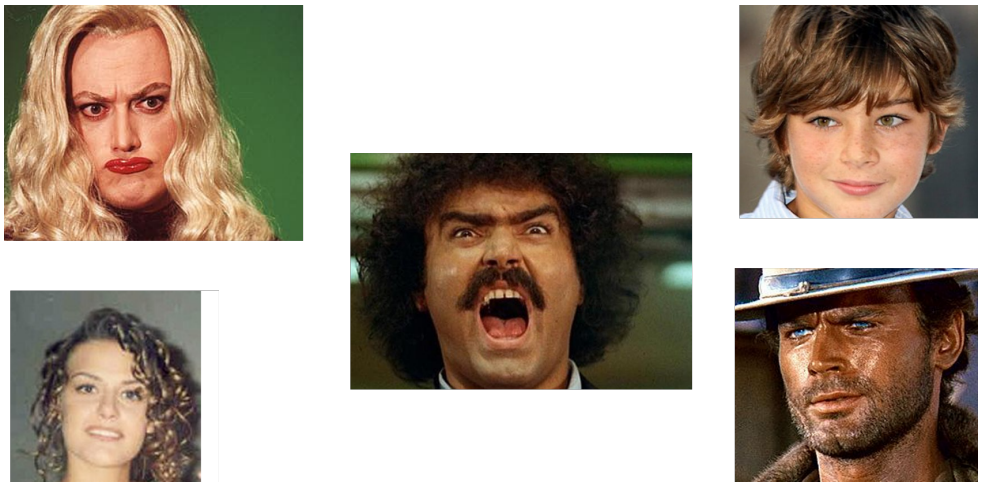


Figure 3.7. Disguised and aged vips

We had only a handful of VIPs that were not parametrized correctly (less than 2%) and were not recognized in more than three photos. We found some patterns between them.

- VIP is simply not very famous and there are only a few photos of him on the web.
- VIP comes from radio.
- VIP comes from a group, a comic duo or trio, and it never appears in photos alone (thus reducing heavily the train set).
- VIP has a very common name and is not the one appearing the most in researchers.
- VIP always wears a mask. (see 3.8)

Figure 3.8. Miss keta, always wears a mask https://rb.gy/58vlla

Even if our algorithm is quite good in dealing with a difficult train set, so that usually even if one of these conditions is happening it succeeds in solving the task, when two or more of these conditions are verified, it can fail.

3.3.5 Deployment

For the production deployment, we opted for a three-headed solution:

- A Python script to initialize the group and to start populating it.
- A logic app to easily manage the online stream of data. A.1.2
- A rest API back end to allow dynamic updates of the VIP list. At the moment only the back-end is written but soon enough a front-end interface will be deployed.

3.3.6 Conclusion

In the end, we are quite satisfied with the solution. It fulfills all the requirements of the client: it is clean, simple, easy to conjugate with the production environment, but at the same time it is very accurate.

3.4 Multimodal sentiment analysis

For this task, the client asked to improve the performance of their textual sentiment analysis model on tweets with images by finding a way to allow it to extract information also from the visual part. This was not easy, because they put some severe constraints;

- They didn't want a whole new model, since they put a lot of money and efforts in training and implementation of the old one.
- They didn't want to fine-tune it.
- They wanted us to use Azure APIs for these improvements.

In other words, we just tried to:

- Extract some relevant information using Azure tools.
- Try to encode this information in a way that the old model could understand.

On top of all of that, the company providing the old model delivered it as a black box, and we only knew that it was not a state-of-art technology probably based on the bag-of-words technique.

3.4.1 Architecture

By the analysis of the dataset, we identified two macro groups of images accompanying tweets:

- Side pics, where the images show the subject of the tweet.
- Reaction pics, where the images express the sentiment.

Obviously, the only type that can help us in sentiment detection, is the second one. The reaction pic usually is characterized by two features: bottom text with some catchphrase, and highly expressive faces.

So, we decided to try to exploit Azure APIs to extract these features. For the text, the choice was easy, since Azure provides a very powerful read API that allows to extract the text.

Regarding the collection of the emotion, we tried the only Azure API available that could match our need: the Face API that is able to identify the emotion of faces.

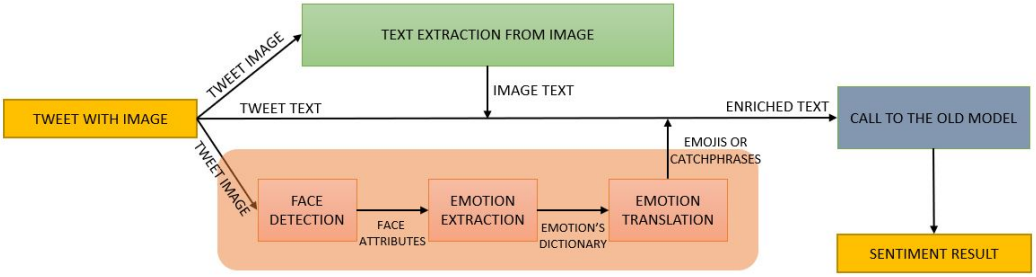


Figure 3.9. Full pipeline

The whole pipeline is depicted in figure 3.9.

3.4.2 Test

To text these functionalities, we had to create a dataset. We labeled by hand tweets about TV programs collected from 5 different hashtags. (amici19, ballandoconlestelle, docnelletuemani, chediociaiuti, CTFC). After some down-sampling to balance the set, we came up with 2253 tweets equally divided between positive, negative and neutral. Labeling policy could be slightly different from the one that leads to the model resulting in inconsistency and lower accuracy, but a bigger pattern should emerge nonetheless.

The results are shown below.

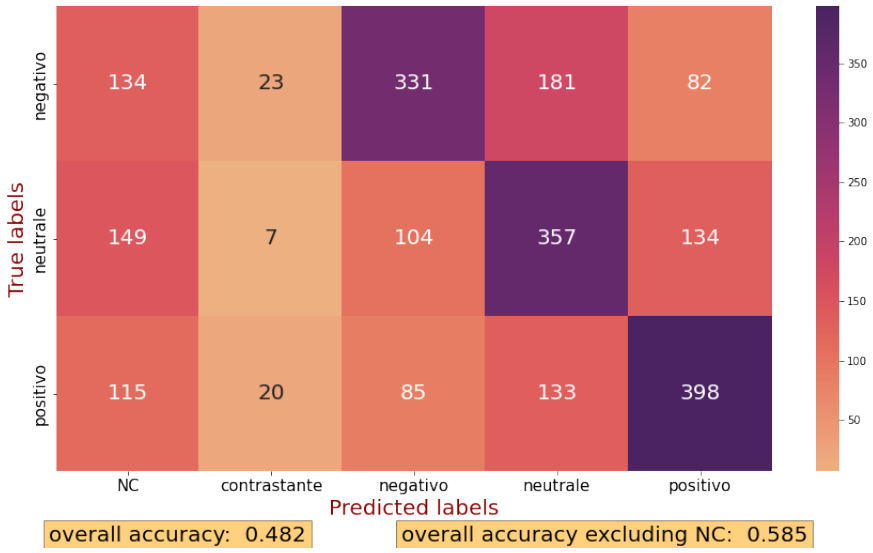


Figure 3.10. Only tweet text

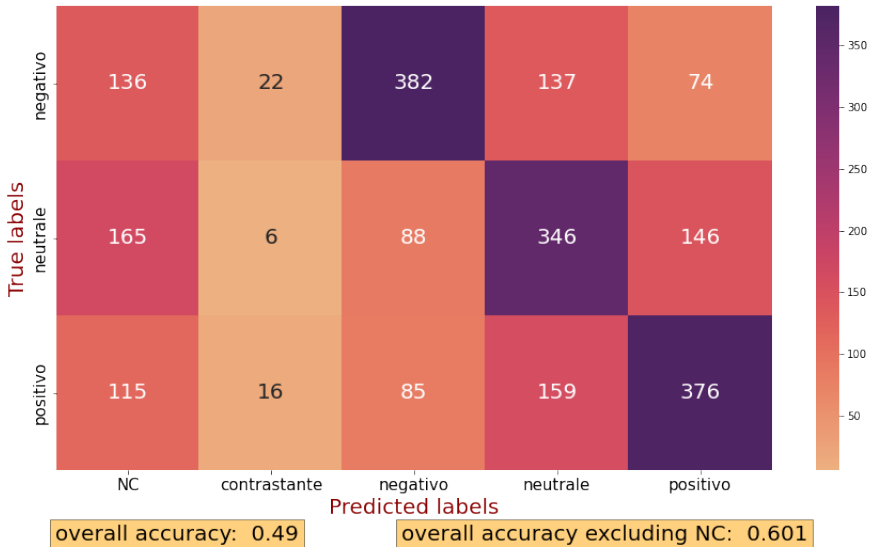


Figure 3.11. Tweet text and image text

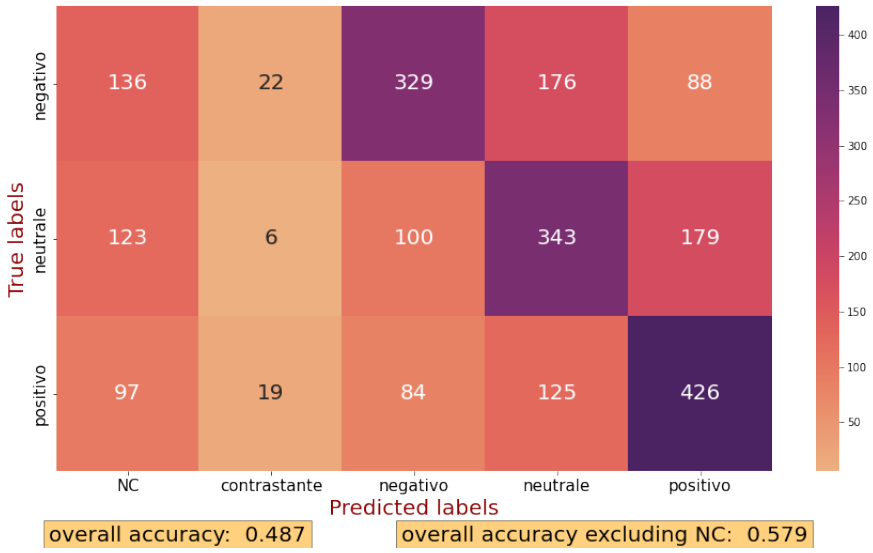


Figure 3.12. Tweet text and emotion transformed in emoticon

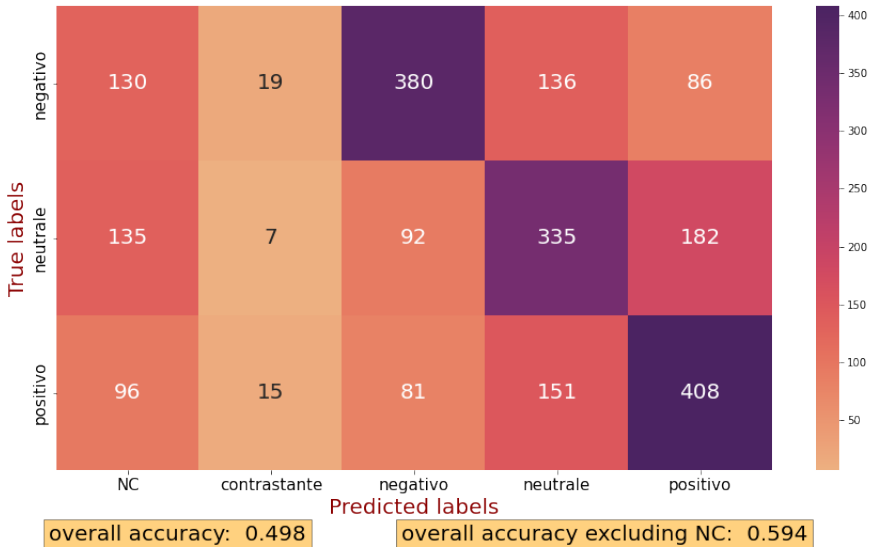


Figure 3.13. Tweet text, image text, and emotion transformed in emoticon

From this experiment emerges that including the in-image-text helps to detect the sentiment of the tweet. On the contrary, the utility of using expression of the faces is not very useful, as it can bee seen from fig 3.14 and 3.15. This could happen for several reasons:

- The expression of faces in an image is not always correlated with the general sentiment, especially when there is more than one person.
- The algorithm that detects emotion, although being very good, is not perfect.
- The translation from the emotion to the emoticon or catchphrase, even if it is "readable" from the old model, adds noise and volatility.
- There are a lot of "side pics" where image is only used to show the subject of the comment. When the subject is a person it is usually smiling (regardless the sentiment of the comment).

	precision	recall	f1-score	support
NC	0.00	0.00	0.00	0
contrastante	0.00	0.00	0.00	0
negativo	0.69	0.51	0.58	751
neutrale	0.54	0.46	0.50	751
positivo	0.63	0.50	0.56	751
accuracy			0.49	2253
macro avg	0.37	0.29	0.33	2253
weighted avg	0.62	0.49	0.55	2253

Figure 3.14. Tweet text and image text

	precision	recall	f1-score	support
NC	0.00	0.00	0.00	0
contrastante	0.00	0.00	0.00	0
negativo	0.69	0.51	0.58	751
neutrale	0.54	0.45	0.49	751
positivo	0.60	0.54	0.57	751
accuracy			0.50	2253
macro avg	0.37	0.30	0.33	2253
weighted avg	0.61	0.50	0.55	2253

Figure 3.15. Tweet text, image text, and emotion transformed in emoticon

3.4.3 Conclusion

We demonstrated that this approach, even if gives a small improvement on the old model, is not the best way to manage the task. In the next chapter, I present a more

rigorous and up-to-date method to tackle multimodal sentiment analysis.

Chapter 4

A MultiModal approach to Sentiment Analysis for Italian TV programs tweet

4.1 Introduction

MmaSAI TVpT is the acronym for Multi-modal approach to Sentiment analysis for Italian TV programs Tweets.

After the development of a very basic technique to increase the performance of the client's model already in production, I wanted to try to build a model that could process natively multi-modal data, thus having the possibility to get the maximum informative value from it. I decided to use only open-source and free instruments and to apply state of the art embedding techniques to develop a multi-stream classification neural network.

In order to train the model, I had to extract and label my own data set. This task is pretty specific and, as expected, a public data set with the needed characteristics does not exist, and in general, due to the fact that Italian speakers are very few relative to the world, Italian dataset availability is very scarce.

After the labeling, the only important feature engineering step left before feeding the main model was to optimize an OCR in order to extract text from images.

4.2 Data collection and preprocessing

The final data set is composed of 6027 tweets, with a train set of 4746 and a test set of 1281. This set comes from a pipeline composed of data collection via scraping, data labeling, data balancing, and extraction of text inside the image.

4.2.1 Data collection

I downloaded the tweet with a simple python script (see code A.2.1) that, given a hashtag, explores the last 5000 tweets about that hashtag downloading and saving text, date, and image (or the first frame of GIFs and videos) of all the tweets where the text was accompanied by another media.

4.2.2 Data labeling and balancing

To label the data, I used pigeon jupyter annotator. (see code A.2.2, fig 4.1). The data labeling operation is not as straightforward as it is in other tasks. Labeling a picture of a puppy as a dog or cat is objective, labeling a tweet as positive/negative/neutral, unfortunately, is not. To mitigate the subjectivity of the labeling, I gave myself some policies. I decided to interpret in a quite wide way "positive" and "negative", labeling as that also slight criticism and appreciation. This obviously makes the task a little more difficult, since sometimes the separation line between the two classes is nuanced. I also decided to discard all the tweets in languages different from Italian and all the tweets not referred to the program.

After the labeling, I had to correct the test set due to a strong imbalance of different classes, by taking all the negative comments (less frequent class) but only a random sample of that size of positive and neutral comments.

Figure 4.1. Pigeon Jupyter annotator

4.2.3 In-image-Text Extraction

To extract text from images, the best free open-source tool is Tesseract, an OCR created by Hewlett Packard and actually sponsored by Google [51]. Unfortunately, Tesseract is trained to read from documents written black on white and is not able to process automatically more complicated images where the background is noisy and unstable.

To solve this problem we had to pre-process images with four transformations:

- Bilateral filter
- Grayscaling
- Binarization
- Inversion

Then, we had to properly set Tesseract, taking only alphabetical characters and setting it to exploit Italian vocabulary.

This pipeline has been chosen to maximize performance over images formatted like memes because we found that most reaction pics on Twitter are characterized by this configuration (for the code see A.2.3).

Bilateral filter

The first function that we applied to our image is bilateral filtering. In a nutshell, this filter helps to remove the noise, but, in contrast with other filters, preserves

edges instead of blurring them. This operation is performed by excluding from the blurring of a point the neighbors that do not present similar intensities.

Grayscaling

The second operation is pretty simple: we project our RGB images in grayscale.

Binarization and inversion

The last transformation is binarization. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to 255. Since we have white text, we want to blackout everything is not almost perfectly white (not exactly perfect since usually text is not "255-white". We found that 240 was a threshold that could do the work. Since Tesseract is trained to recognize black text, we also need to invert the colors. The function threshold from open cv can do the two operations jointly, by selecting the inverted binarization.

Figure 4.2. (a) Normal image (b) Bilateral Filtered (c) Grayscaled (d) Inv-binarized

4.3 Architecture

The basic architecture (after the preprocessing described in the previous paragraphs) is quite simple: there is a branch where the text is embedded and a branch where the image is embedded. Then a fusion layer where the two are merged, and in the end a block of classification layers.

All these "logical" blocks have been declined in different ways to empirically explore the best choices.

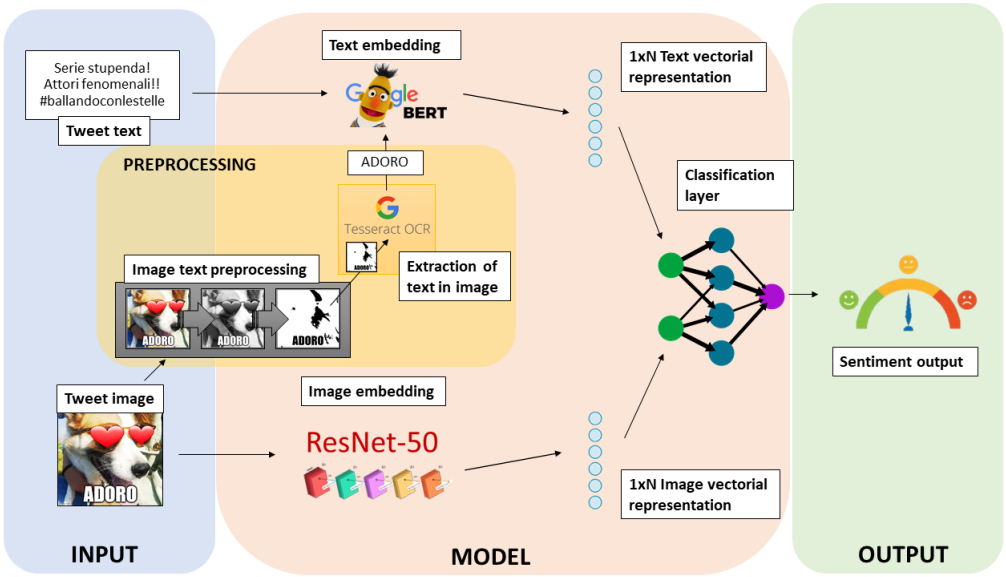


Figure 4.3. Final model confusion matrix

4.3.1 Text block

Choosing best pre-trained model

	BEST TEST ACCURACY
poliBERT_sa	0.6159
neuraly Italian BERT	0.5917
multilingual distilBERT	0.4715
BERTino	0.4785
poliBERT_sa + emoticons	0.6362

For the text block, I choose to embed the text using BERT (see 2.3), since it assures state-of-the-art performance and is also very easy to include in TensorFlow models, thanks to the hugging face transformer library. Although there are a lot of different implementations of BERT fine-tuned on different datasets and tasks in English, finding architectures pre-trained on Italian datasets is harder. I tried four different backbones:

- polibert_SA : POLItic BERT based Sentiment Analysis
- neuraly : Italian BERT Sentiment model
- BERTino : an Italian DistilBERT model
- DistilBERT base multilingual model (cased)

Of these, only the first two were fine-tuned specifically for sentiment classification (and this is clearly visible from the result table). The one that performed better is actually fine-tuned for sentiment on tweets (even if on political ones). Ideally, it would be best to be able to fine-tune a little bit more but unfortunately, we do not have the data, nor the computational power.

After the choice of BERT backbone, I decided to apply a small preprocessing to strings.

- Remove URLs since are not informative.
- Remove hashtags, since the dataset is balanced in its whole but not for single hashtags, and this could bias the training.
- Transform some emoji into text with similar meaning, since the tokenizer of this architectures is not trained to recognize them but they often carry strong sentiment information (see 4.4).

```
def emoticon_replacer(sentence):
    """
    replace emoticons in a text with words with similiar semantic value
    """
    sentence=re.sub(r'😍|🥰|😘|😻|❤️|🧡|💛|💚|💙|💜|🖤|🤍|🤎|💞|💋|♥️|💗|💖|💘|💝', 'adoro ', sentence)
    sentence=re.sub(r'😀|😃|😄|😁|😆|😂|🤣|😹|😸', 'ahah ', sentence)
    sentence=re.sub(r'😡|🤬|👿|🤡', 'infame ', sentence)
    sentence=re.sub(r'✈️|🔥|🛸|⭐|🌟|✨|💥|🌫️|💨|〰|🚀', 'wow ', sentence)
    sentence=re.sub(r'😢|😭|😿', 'piango ', sentence)
    sentence=re.sub(r'🤢|🤮', 'schifo ', sentence)
    return sentence
```

Figure 4.4. emoticon replacer

fusion between text and text in image

	BEST TEST ACCURACY
no image text	0.6362
early concatenation	0.6542
late concatenation	0.6448
late GMU	0.6151

I've tried different ways to incorporate information coming from text extracted from images.

- early fusion: concatenating the tokenized texts before passing them to BERT
- late fusion: concatenating the tokenized texts after passing them to BERT
- late GMU (see 4.5)

I obtained the best result by concatenating them before the embedding.

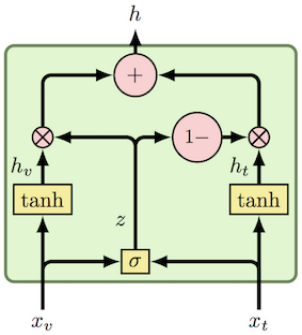


Figure 4.5. Gated multimodal unit [48]

4.3.2 Image block

	BEST TEST ACCURACY
resnet50v2	0.4762
resnet101v2	0.4610
inception_resnetv2	0.4461
resenet50v2 + data agumentation	0.4707

For the image feature extraction, I tried different CNN backbones. The accuracies are significantly lower, but this is expected, since usually the sentiment is carried predominantly by the text, and in a lot of tweets the image has no information for the sentiment at all, serving only as a picture of the object of the comment. However, for the next steps, I fixed them as choice resnet50v2.

4.3.3 Fusion layer between image and text

	BEST TEST ACCURACY
without images	0.6542
concatenation	0.6601
GMU	0.6518
linear sum	0.6440

Again, the best fusion layer is a simple concatenation.

4.3.4 Other blocks and hyperparameters

The classification block on top of the embedding is pretty standard; a layer of BatchNormalization and three dense layers (of size: 512, 128, 3) interspersed by a 0.2 dropout. I used a batch size of 4 (the best that could fit on my system). The text branch run on my NVIDIA GeForce MX150, while the image branch, due to the big size, had to run on my CPU. Each experiment runs for ten epochs, although all the information was usually extracted in the first three epochs. As optimizer, I used RMSprop with an exponential decaying learning rate starting at 0.0001. The loss is the classic cross-entropy. I tried to conduct a little bit of empirical tuning of these parameters but I didn't have any luck.

4.3.5 Final architecture and results

This is the whole model architecture:

Figure 4.6. Final model architecture

I obtained a test accuracy of 0.66, and the errors are distributed quite symmetrically (see the confusion matrix in fig 4.8).

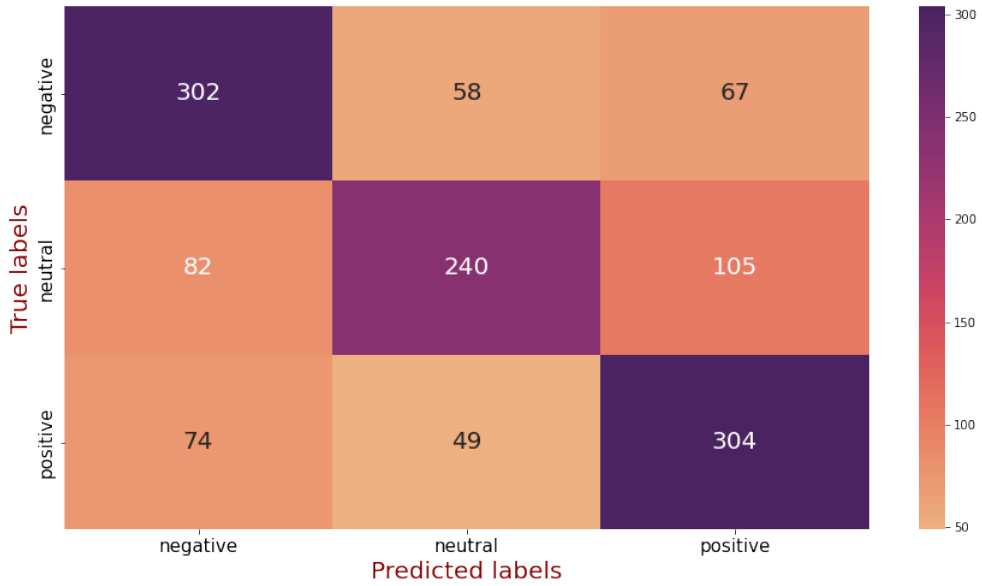


Figure 4.7. Final model confusion matrix

	precision	recall	f1-score	support
negative	0.66	0.71	0.68	427
neutral	0.69	0.56	0.62	427
positive	0.64	0.71	0.67	427
accuracy			0.66	1281
macro avg	0.66	0.66	0.66	1281
weighted avg	0.66	0.66	0.66	1281

Figure 4.8. Final model metrics

4.4 Error analysis

Inspecting the errors, I found different interesting patterns, that could help understand the limits of the model.

4.4.1 Soft edges

The first error is quite endemic for this task, and it was expected, since we decided to deal with this problem treating it as a categorical one, simplifying the reality. Indeed, to classify a tweet in "positive" or "negative" is not as objective as it is to

divide pictures between cats and dogs, and the subject that labels the text sometimes simply has to make a choice. Even if sometimes some borderline cases are labeled following the brain of the human annotator, thanks to the law of the large numbers, after training on a large data set a model can learn the relevant patterns and should be able to understand the strong cases (and our model actually is quite good in solving the majority of them), but a little bias will always be present (although in the next paragraph I will propose some techniques to reduce it).

Anyway, these are some soft cases where the model failed in matching the idea of the annotator.

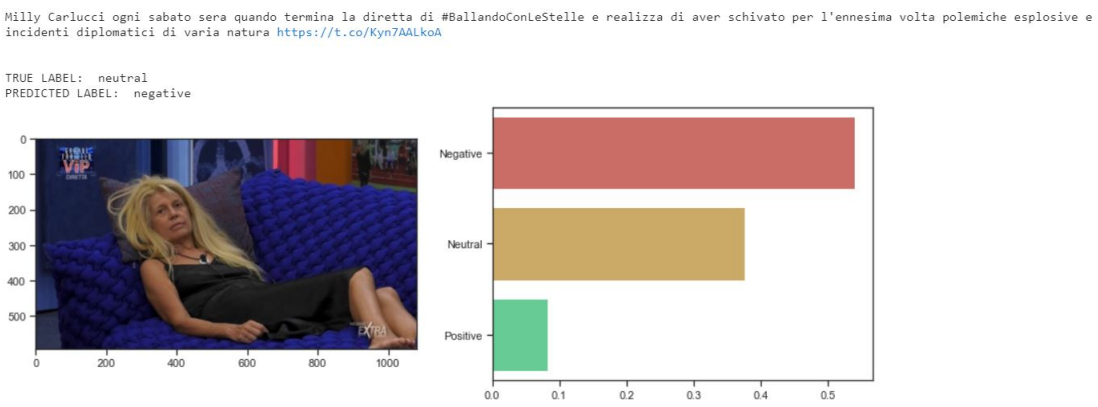


Figure 4.9. Hard to say if this is a negative or a neutral comment

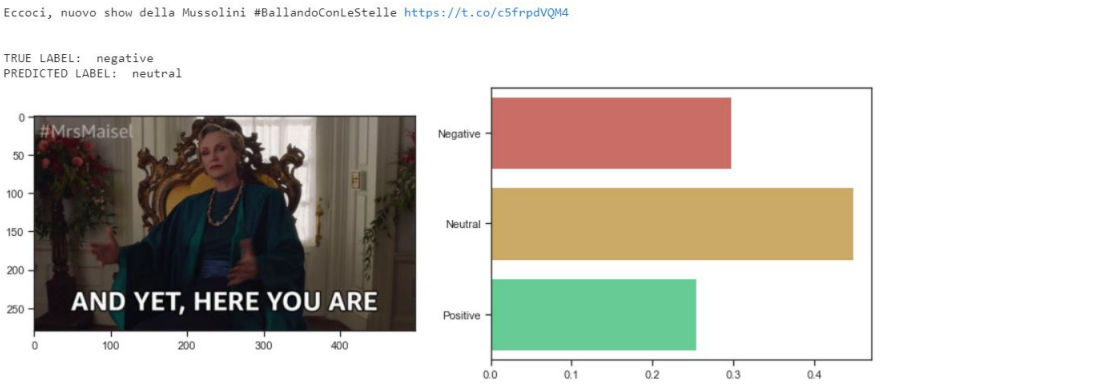


Figure 4.10. Hard to say if this is a negative or a neutral comment

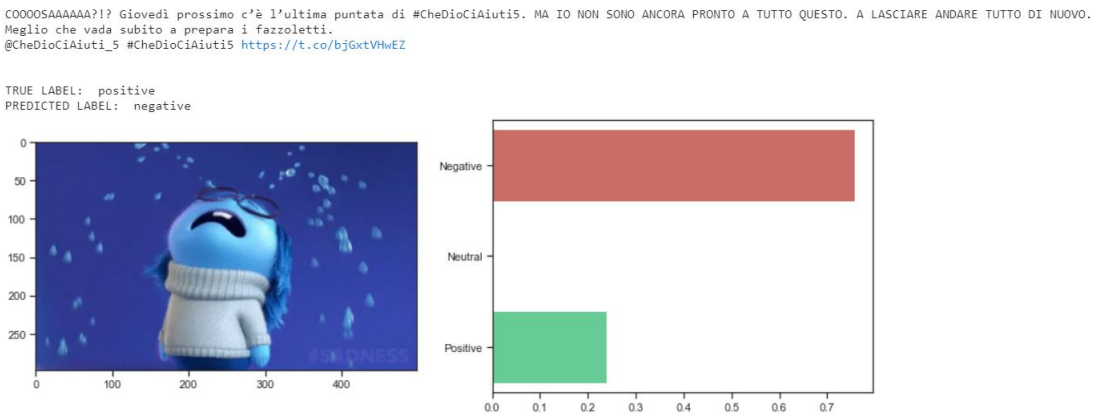


Figure 4.11. I labeled it as positive but I understand why it could be seen as negative

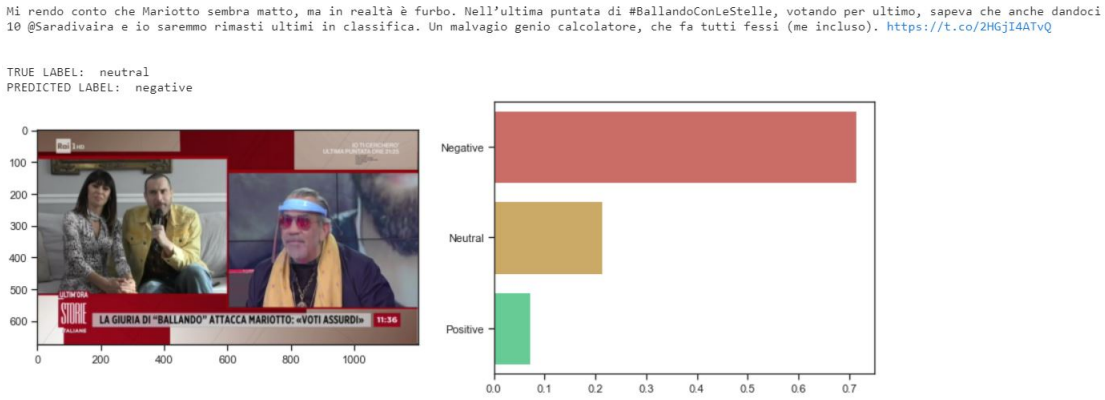


Figure 4.12. I labeled it as neutral because the tweet is only a quote, and usually quotes are labeled as neutral if there is not also something that express the idea of the user about it, but the model sees a lot of negativity and behave accordingly.

4.4.2 Conflicting

In a similar way, when the tweets are conflicting, in the sense that expresses positivity towards something but negativity against something else, the model can decide differently from the annotator.

La coppia più bella delle serie tv.

Sono riusciti a distruggerla e ancora mi chiedo perché 🤔 che senso ha 🤔 #GuidoCorsi #AzzurraLeonardi #CheDioCiAiuti #CheDioCiAiuti5 #LinoGuanciale #FrancescaChillemi https://t.co/8uMgyGK8mM

TRUE LABEL: negative
PREDICTED LABEL: positive

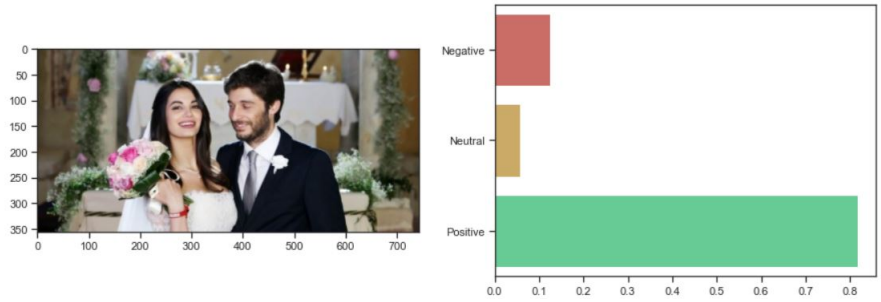


Figure 4.13

Parole sante! Brava #rosalindacelentano questa sera la giuria non ha trasmesso armonia e serenità ... #BallandoConLeStelle https://t.co/TEZFHd747p

TRUE LABEL: negative
PREDICTED LABEL: positive

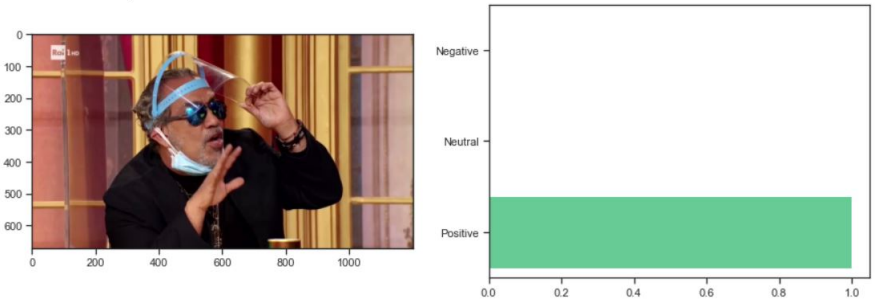


Figure 4.14

Martina Beltrami davvero vuole mettersi contro di loro? Ma davvero?😂🛫 #karaoke #amici19 https://t.co/U6yKrCRm7W

TRUE LABEL: negative
PREDICTED LABEL: positive

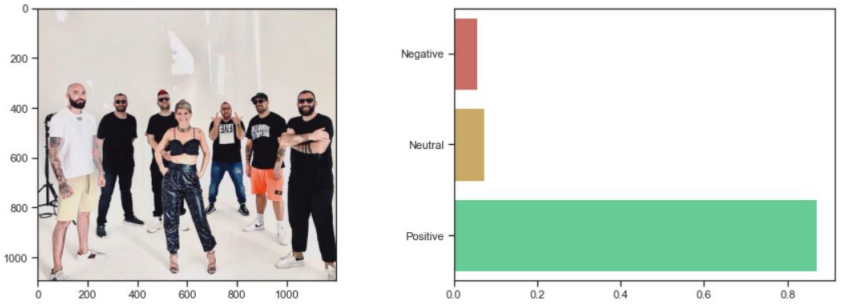


Figure 4.15

Posso dire? Questa puntata si è meritata di essere vista fino alle una. Quelle di Amici19 mi facevano cadere i coglioni in terra dopo 10 minuti. #amicispeciali https://t.co/tchH4tlpH3

TRUE LABEL: positive
PREDICTED LABEL: negative

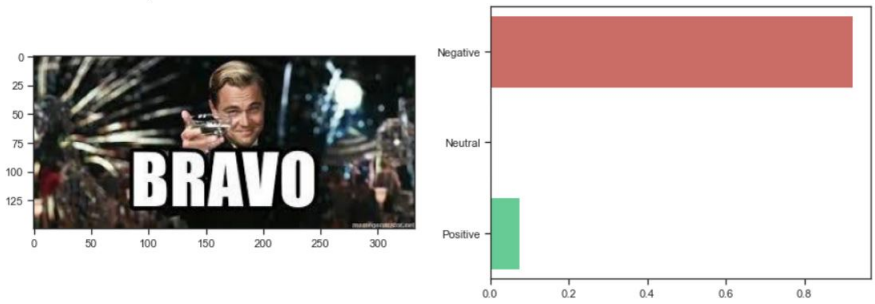


Figure 4.16

4.4.3 Sentiment carried predominantly by image.

Unfortunately, the model is not able to understand images as well as it understands texts. The reasons why this is happening will be discussed in the next paragraph, however, this is the reason why the tweets where the sentiment is only carried by the image (not the text on the image but the "visual" part of the image, like facial expression), is often misclassified.

Stasera alla giuria di #BallandoConLeStelle https://t.co/29XlLeBDMY

TRUE LABEL: negative
PREDICTED LABEL: neutral

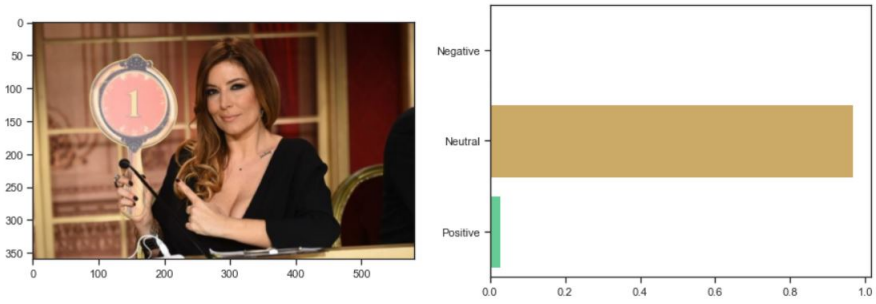


Figure 4.17. The algorithm is not able to understand that the vote in the picture is the vote that the writer of the tweet is giving to the program jury

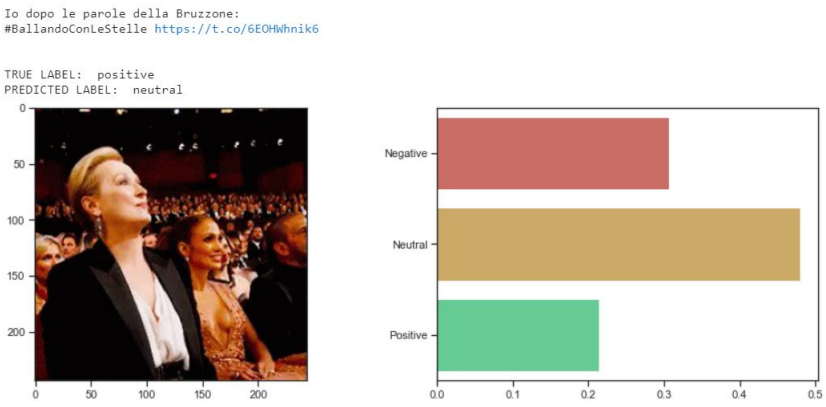


Figure 4.18. The algorithm is not able to detect the positivity expressed by the image

4.4.4 Strange use of the text

Sometimes the model is confused by strange use of the text, for example when the writer of the tweet uses ironic words that usually carry negative sentiment, or when a strange or new word is used, or when a word is not written properly.

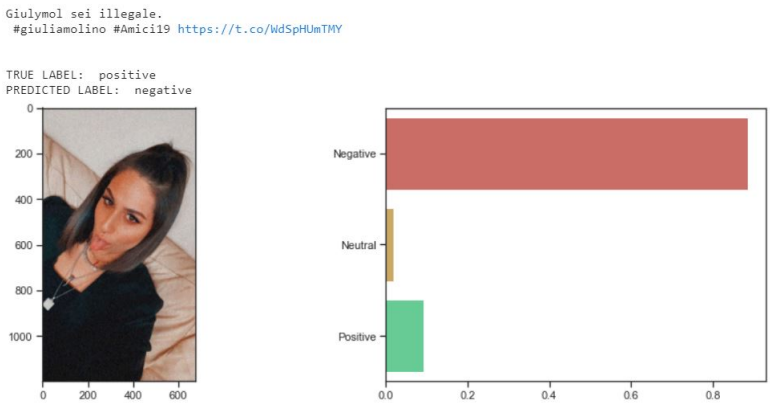


Figure 4.19. "illegale" used ironically.

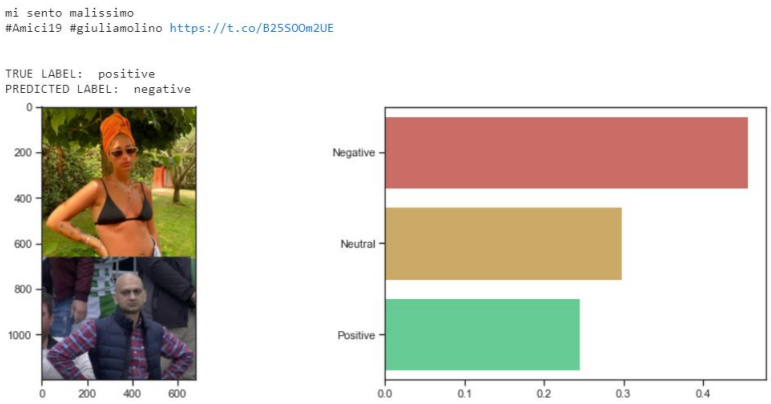


Figure 4.20. "Mi sento malissimo" used ironically.

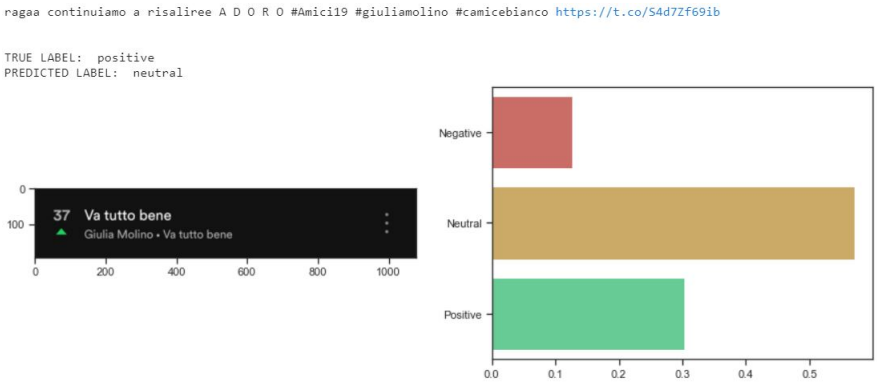


Figure 4.21. "A D O R O" is "adoro" that it means "I love it", and it is written decomposed to enphatize it, but the model is not prepared to tokenize properly it.

Figure 4.22. "Congiunto" is a techincal word meaning "lover", an obslolet term that came back to use in Italy since it has been used in goverment directives.

4.5 Discussion

Considering the low baseline defined by the old model in production (accuracy around 0.5), the result obtained can be described as quite promising, since we got an accuracy of 0.66 with a balanced confusion matrix. In this paragraph, I'll analyze what worked, what didn't, and what could be done to enhance this prototype.

4.5.1 What performed well

What granted the best performance gain is certainly the different approach to analyze text. Since BERT came out, it has revolutionized the NLP field helping to boost incredibly the ability to embed the semantic value of words into numbers, and our case confirmed his strength. Having the possibility to exploit a model that has been pre-trained two times (the first one in the classical BERT way on a very big Italian corpus, the second one specifically on a sentiment analysis task on Twitter) has assured a high-level generalization despite the small data-set. Two other ideas allowed the model to arrive almost at his best form: the use of text written in the images, and the encoding of the emoticons. These features, even if roughly engineered, demonstrated to carry a lot of sentiment.

4.5.2 What did not perform well

What certainly didn't work as hoped, is the direct embedding of images via CNN. This could be happening for multiple reasons, and not all of them are easily solvable. The size of data is for sure playing a part. With only 4000 tweets it's hard for the algorithm to model properly the correlation between image and text features. The problem is that the distribution could simply be too noisy. As we explained briefly in paragraph3.4.1, the images in tweets can be divided in two macro families: side pics (that just shows the object of the comment) and reaction pics (that express a sentiment, usually via text and/or image). Obviously, this division is not known a priori by the network, and to understand this difference and exploit it properly, the context should be understood in a more complex way, so much so that it would not be granted that just a lot more data would be enough to model. Apart from this, the reaction pics come in a lot of forms, so much so that it's difficult to understand them without some form of pre-existing knowledge.

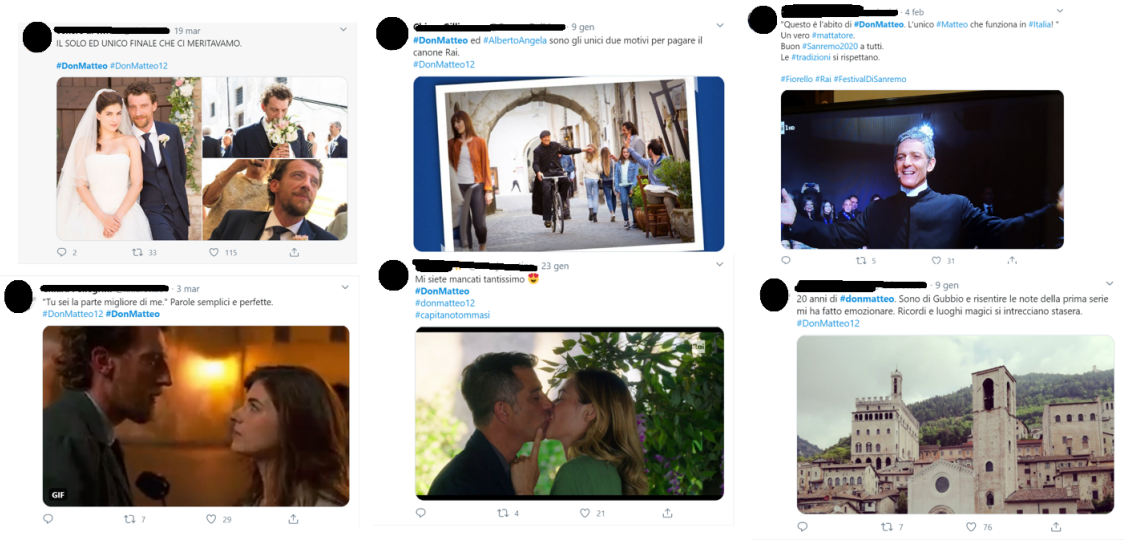


Figure 4.23. Side pics examples

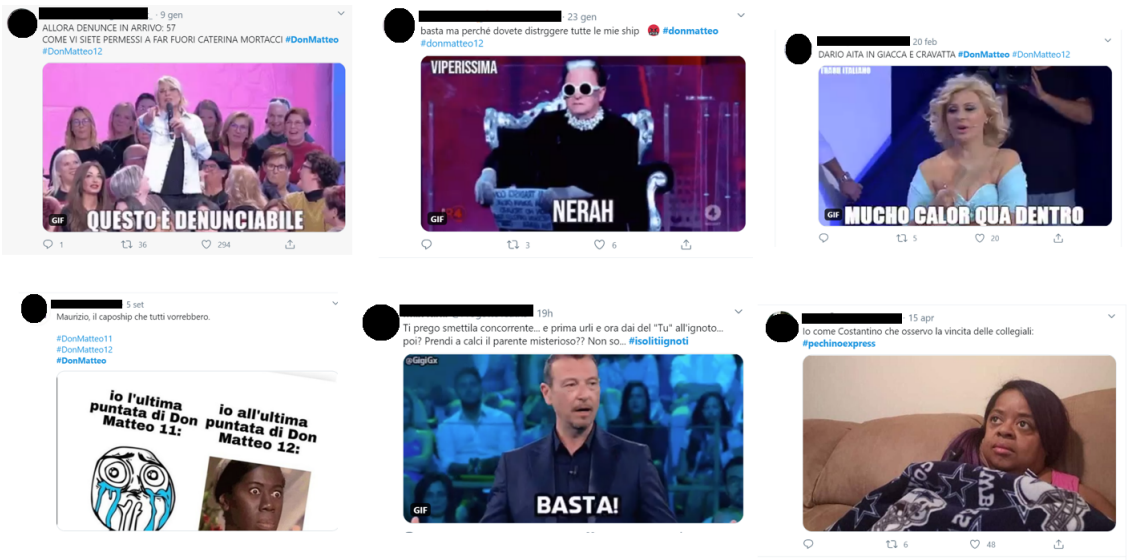


Figure 4.24. Reaction pics examples

4.5.3 Possible improvements

There is a saying in data science: "More data is better than better algorithms". It is quite depressing to admit, and not totally true, but to some extent, this is certainly something that we have to consider. The first thing to do to improve these results would be to get and label more data, and maybe improve the quality of the data assuring that the labeling is not done by one person only but by a group of different people, to try to smooth at least a little bit of the subjectivity. With more time and

computational power, moreover, a more systematic grid search and cross-validation could be done to present more stable results.

Apart from these obvious but true statements, a possible path to increase the performance could be to try to make the CNN branch work a little bit more. Two possible things to try would be pre-training the branch on a dataset more appropriate than the general Imagenet, or restricting the search on the image by using a Regional CNN to take only an object on the image.

4.5.4 Conclusions

Working on this task has been really challenging since it featured multiple novelties and the track that we decided to walk was not previously beaten. Sadly, even if the result was quite good in general, the main target, exploiting the information contained in images, is only partly solved. Extracting relevant information from images' visual features has been harder than we thought. By the way, when you lose, you learn, and this project taught me multiple things, for example how insidious it is to develop a project from data collection to result's presentation.

Bibliography

[1] Mika V. Mäntylä, Daniel Graziotin, and Miikka Kuutila. "The Evolution of Sentiment Analysis - A Review of Research Topics, Venues, and Top Cited Papers". In: *Computer Science Review* 27 (Feb. 2018), pp. 16–32.

[2] Wikipedia. *Sentiment analysis*. URL: https://en.wikipedia.org/wiki/Sentiment_analysis.

[3] A.M. Turing. "Computing Machinery and Intelligence". In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

[4] François Chollet. *On the Measure of Intelligence*. 2019. arXiv: 1911.01547 [cs.AI].

[5] Zhenzhu Meng, Yating Hu, and Christophe Ancey. "Using a Data Driven Approach to Predict Waves Generated by Gravity Driven Mass Flows". In: *Water* 12 (Feb. 2020). DOI: 10.3390/w12020600.

[6] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* (1958), pp. 65–386.

[7] Robert H. Nielsen. "Theory of the Backpropagation Neural Network". In: *Proceedings of the International Joint Conference on Neural Networks* (Washington, DC). Vol. I. Piscataway, NJ: IEEE, 1989, pp. 593–605.

[8] A. Rosebrock. *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch, 2017. URL: https://books.google.it/books?id=9Ul-tgEACAAJ.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[10] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[11] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[12] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[13] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 3104–3112.

[15] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation. 2014. URL: http://arxiv.org/abs/1409.0473.

[17] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *CoRR* abs/1508.04025 (2015). arXiv: 1508.04025. URL: http://arxiv.org/abs/1508.04025.

[18] Jay Alammar. *Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)*. URL: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/.

[19] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[20] Jay Alammar. *Illustrated transformer*. URL: http://jalammar.github.io/illustrated-transformer/.

[21] Dipanjan (DJ) Sarkar. *A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning*. URL: https://towardsdatascience.

com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

[22] A. L. Knutson. "Japanese opinion surveys: the special need and the special difficulties". In: *Public Opinion Quarterly* 9.3 (1945), pp. 313–319.

[23] P. L. Fegiz. "Italian Public Opinion". In: *Public Opinion Quarterly* 11.1 (1947), pp. 92–96.

[24] Ĉ. ADAMEC and I. Viden. "Polls come to Czechoslovakia". In: *Public Opinion Quarterly* 11.4 (1947), pp. 548–552.

[25] D. Dubois S. A. Sandri and H. W. Kalfsbeek. ""Elicitation, assessment, and pooling of expert judgments using possibility theory". In: *IEEE transactions on fuzzy systems* (1995), pp. 313–335.

[26] R. F. Bruce J. M. Wiebe and T. P. O'Hara. ""Development and use of a gold-standard data set for subjectivity classifications". In: *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics* 3.3 (1999), pp. 246–253.

[27] L. Lee B. Pang and S. Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques". In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing* 10 (2002), pp. 79–86.

[28] P. D. Turney. "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews". In: *Proceedings of the 40th annual meeting on association for computational linguistics* (2002), pp. 417–424.

[29] Bo Pang and Lillian Lee. "Opinion Mining and Sentiment Analysis". In: *Found. Trends Inf. Retr.* 2.1–2 (Jan. 2008), pp. 1–135. ISSN: 1554-0669. DOI: 10.1561/1500000011. URL: https://doi.org/10.1561/1500000011.

[30] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. "Sentiment analysis algorithms and applications: A survey". In: *Ain Shams Engineering Journal* 5.4 (2014), pp. 1093–1113. ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2014.04.011.

[31] ChengXiang Zhai Charu C. Aggarwal. *Mining Text Data*. Springer-Verlag New York, 2012. DOI: 10.1007/978-1-4614-3223-4.

[32] Thomas JA Cover TM. *Elements of information theory*. New York: John Wiley and Sons, 1991.

[33] J. R. Quinlan. "Induction of Decision Trees". In: *Mach. Learn.* 1.1 (Mar. 1986), pp. 81–106. ISSN: 0885-6125. DOI: 10.1023/A:1022643204877. URL: https://doi.org/10.1023/A:1022643204877.

[34] Ramesh Wadawadagi and Veerappa Pagi. "Sentiment analysis with deep neural networks: comparative study and performance assessment". In: *Artificial Intelligence Review* (May 2020). DOI: 10.1007/s10462-020-09845-2.

[35] Google. *word2vec*. URL: https://code.google.com/archive/p/word2vec/.

[36] Stanford. *GloVe*. URL: http://nlp.stanford.edu/projects/glove/.

[37] Matthew E. Peters et al. "Deep contextualized word representations". In: *CoRR* abs/1802.05365 (2018). arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

[38] Sebastian Ruder. *NLP's ImageNet moment has arrived*. URL: https://ruder.io/nlp-imagenet/.

[39] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[40] Jay Alammar. *Illustrated Bert*. URL: http://jalammar.github.io/illustrated-bert/.

[41] Vasavi Gajarla and Aditi Gupta. "Emotion detection and sentiment analysis of images". In: ().

[42] Quoc-Tuan Truong and Hady W. Lauw. "Visual Sentiment Analysis for Review Images with Item-Oriented and User-Oriented CNN". In: *Proceedings of the 25th ACM International Conference on Multimedia*. MM '17. Mountain View, California, USA: Association for Computing Machinery, 2017, pp. 1274–1282. ISBN: 9781450349062. DOI: 10.1145/3123266.3123374. URL: https://doi.org/10.1145/3123266.3123374.

[43] Dazhen Lin et al. "GIF Video Sentiment Detection Using Semantic Sequence". In: *Mathematical Problems in Engineering* 2017 (May 2017), pp. 1–11. DOI: 10.1155/2017/6863174.

[44] Guoyong Cai and Binbin Xia. "Convolutional Neural Networks for Multimedia Sentiment Analysis". In: Oct. 2015, pp. 159–167. ISBN: 978-3-319-25206-3. DOI: 10.1007/978-3-319-25207-0_14.

[45] A. Kumar and Geetanjali Garg. "Sentiment analysis of multimodal twitter data". In: *Multimedia Tools and Applications* (2019), pp. 1–17.

[46] D. Borth et al. "SentiBank: large-scale ontology and classifiers for detecting sentiment and emotions in visual content". In: *MM '13*. 2013.

[47] Pradyumna Gupta, Himanshu Gupta, and Aman Sinha. *DSC IIT-ISM at SemEval-2020 Task 8: Bi-Fusion Techniques for Deep Meme Emotion Analysis*. 2020. arXiv: 2008.00825 [cs.CV].

[48] John Arevalo et al. *Gated Multimodal Units for Information Fusion*. 2017. arXiv: 1702.01992 [stat.ML].

[49] Kent Beck, Robert C. Martin, and Martin Fowler. *Manifesto for Agile Software Development*. URL: http://agilemanifesto.org/iso/en/manifesto.html.

[50] Microsoft. *Microsoft Azure Cloud Platform*. URL: https://azure.microsoft.com/it-it/.

[51] Hewlett-Packard. *Tesseract OCR*. URL: https://github.com/tesseract-ocr/tesseract.

Appendix A

Code

The whole code can be found at https://github.com/EgonFerri/MMasaiTVpT.

A.1 Vip recognition

A.1.1 REST API written in pyhton

```
ENDPOINT = '*******'
KEY = '*******'
face_client = FaceClient(ENDPOINT, CognitiveServicesCredentials(KEY))
PERSON_GROUP_ID = 'raicelebs'


def vip_adder(name, n_photos=40, group=PERSON_GROUP_ID):
    print('adding'+name)
    response().download(name, n_photos)
    path = glob.glob('simple_images\\'+name.replace(' ', '_')+'\*')
    vip = face_client.person_group_person.create(group, name)
    vip_images = [file for file in path]
    for image in vip_images:
        with open(image, 'r+b') as w:
            try:
                face_client.person_group_person.add_face_from_stream(
                    group, vip.person_id, w)
            except:
                pass
        os.remove(image)
    os.rmdir(('simple_images\\'+name.replace(' ', '_')))


def trainer(group=PERSON_GROUP_ID):
```

```
25      print('Training the person group...')
26      # Train the person group
27      face_client.person_group.train(PERSON_GROUP_ID)
28
29      while (True):
30          training_status = face_client.person_group.
        get_training_status(
31              PERSON_GROUP_ID)
32          print("Training status: {}.".format(training_status.status))
33          print()
34          if (training_status.status is TrainingStatusType.succeeded):
35              break
36          elif (training_status.status is TrainingStatusType.failed):
37              sys.exit('Training the person group has failed.')
38          time.sleep(5)
39
40
41  def main(req: func.HttpRequest) -> func.HttpResponse:
42      logging.info('Python HTTP trigger function processed a request.')
43
44      name = req.params.get('name')
45      if not name:
46          try:
47              req_body = req.get_json()
48          except ValueError:
49              pass
50          else:
51              name = req_body.get('name')
52
53      if name:
54          vip_adder(name, 10)
55          trainer()
56          return func.HttpResponse(f"Adding of {name} executed
        successfully.")
57      else:
58          return func.HttpResponse(
59              "This HTTP triggered function executed successfully. Pass
         a name in the query string or in the request body for a
        personalized response.",
60              status_code=200
61          )
```

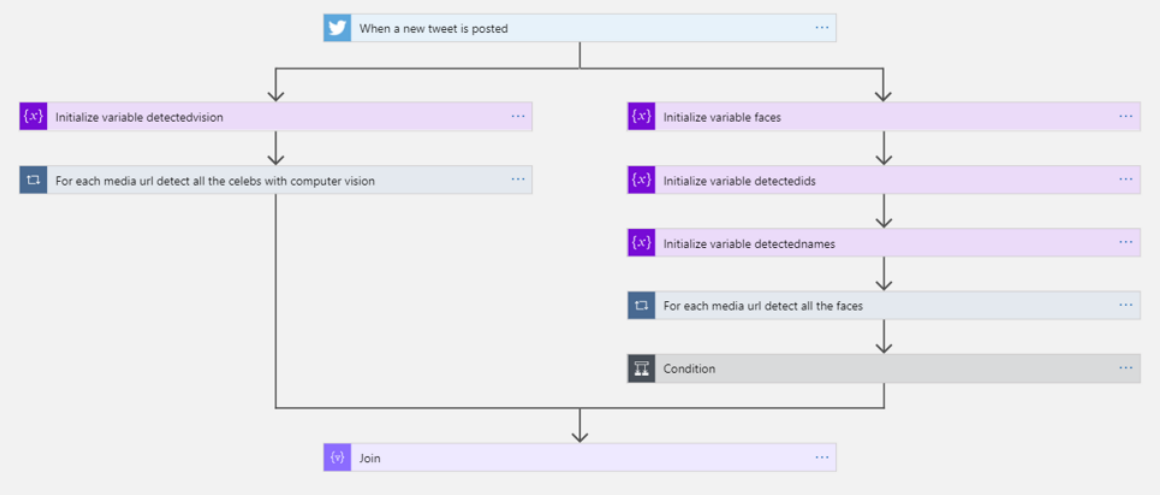


Figure A.1. Schema of the logic app of Vip recognition

A.1.2 Logic app schema

A.2 Sentiment analysis

A.2.1 scraper

```
import pandas as pd
import requests
import os
import sys
import time
import snscrape.modules.twitter as sntwitter


class TwitterScraper:

    def __init__(self, hashtag, count, output_folder, image_folder):
        self.hashtag = hashtag
        self.count = count
        self.output_folder = output_folder + hashtag
        self.image_folder = self.output_folder + image_folder
        print(self.image_folder)
        try:
            os.makedirs(self.output_folder)
            os.makedirs(self.image_folder)
            print('*** Folders created! ***')
        except:
            pass

    def df_maker(self):
```

```
25          lista = []
26          for i, tweet in enumerate(sntwitter.TwitterSearchScraper(
    hashtag).get_items()):
27              if i > count:
28                  break
29              if tweet.media != None:
30                  try:
31                      media = ((tweet.media[0].previewUrl).replace(
32                          '?format=', '.').replace('&name=small', ''))
33                  except:
34                      media = (tweet.media[0].thumbnailUrl)
35                  lista.append(
36                      [tweet.id, tweet.date, tweet.renderedContent,
    media])
37
38          tweets_df = pd.DataFrame(lista, columns=['id', 'date',  'text
    ', 'url'])
39          return tweets_df
40
41      def image_saver(self, df):
42          for idx, row in df.iterrows():
43              index = row['id']
44              url = row['url']
45              response = requests.get(url)
46              with open(self.image_folder + f'{index}.png', "wb") as f:
47                  f.write(response.content)
48
49      def scrape_tweets(self):
50          dataframe = self.df_maker()
51          if len(dataframe) > 0:
52              self.image_saver(dataframe)
53              try:
54                  old = pd.read_csv(self.output_folder + '/data.csv')
55                  dataframe = old.append(dataframe)
56                  dataframe = dataframe.drop_duplicates(subset='id')
57              except:
58                  pass
59              print(len(dataframe))
60              dataframe.to_csv(path_or_buf=self.output_folder +
61                               '/data.csv', index=False)
62              print('** Data written! ** ')
63          else:
64              print('I found no data')
65
66
67  if __name__ == '__main__':
```

```
68      # os.getcwd()  + '/data/'
69      output_folder = 'C:/Users/Egon/projects/visual_sentiment_analysis
        /twitter_scraper/data/'
70      # output_folder = os.getcwd() + '/data/'
71      image_folder = '/images/'
72      lista_hashtag = ['donmatteo12']
73      count = 5000
74
75      for hashtag in lista_hashtag:
76          print(f'scraping for {hashtag}')
77          scraper = TwitterScraper(hashtag=hashtag, count=count,
78                                   output_folder=output_folder,
        image_folder=image_folder)
79          scraper.scrape_tweets()
80          time.sleep(10)
81      print('----FINISHED!----')
82      # print(os.path.dirname(os.path.abspath(sys.argv[0])))
```

A.2.2 labeler

```
1  from pigeon import annotate
2  import pandas as pd
3  from IPython.display import display, Image
4
5
6  def displayer(dataframe, file, hashtag):
7      print(dataframe[dataframe['id'] == file]['text'], file)
8      display(Image(f'../DATA/{hashtag}/images/{str(file)}.png'))
9
10
11 hashtag = input()
12 path = f'../data/{hashtag}'
13 dataframe = pd.read_csv(path+'/data.csv')
14
15 annotations = annotate(
16     list(dataframe['id']),
17     options=['positive', 'negative', 'neutral', 'discard'],
18     display_fn=lambda filename: displayer(dataframe, filename,
       hashtag),
19     include_skip=False
20 )
21
22 res = [i[1] for i in annotations]
23 data_cleaned = dataframe.copy()
24 data_cleaned['sentiment'] = res
25 data_cleaned['hashtag'] = hashtag
26
```

```
data_cleaned.to_csv(path+'/data_labeled.csv', index=False)
```

A.2.3 text extraction

```
# to load and process images
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pytesseract

# change this path if you install pytesseract in another folder:
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'


im = np.array(Image.open(r'../meme sissi/sissi2.jpg'))
plt.figure(figsize=(10, 10))
plt.imshow(im)
plt.xticks([])
plt.yticks([])
plt.savefig('img1.png', bbox_inches='tight')

im = cv2.bilateralFilter(im, 5, 55, 60)
plt.figure(figsize=(10, 10))
plt.imshow(im)
plt.xticks([])
plt.yticks([])
plt.savefig('img2.png', bbox_inches='tight')

im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(10, 10))
plt.imshow(im, cmap='gray')
plt.xticks([])
plt.yticks([])
plt.savefig('img3.png', bbox_inches='tight')

_, im = cv2.threshold(im, 240, 255, 1)
plt.figure(figsize=(10, 10))
plt.imshow(im, cmap='gray')
plt.xticks([])
plt.yticks([])
plt.savefig('img4.png', bbox_inches='tight')


def preprocess_final(im):
    im = cv2.bilateralFilter(im, 5, 55, 60)
```

```
    im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    _, im = cv2.threshold(im, 240, 255, 1)
    return im


custom_config = r"--oem 3 --psm 11 -c tessedit_char_whitelist=
    ABCDEFGHIJKLMNOPQRSTUVWXYZ"

img = np.array(Image.open(r'../meme sissi/sissi2.jpg'))
im = preprocess_final(img)
text = pytesseract.image_to_string(im, lang='ita', config=
    custom_config)
print('text')
```

A.2.4 tensorflow utilities and models

```
import pandas as pd
from glob import glob
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import re
import seaborn as sn
import os

from tensorflow.keras.layers import Input, Dense, Bidirectional,
    Conv2D, MaxPooling2D, Flatten, concatenate, GlobalAveragePooling2D
    , BatchNormalization, Lambda, Add, Multiply
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam, SGD
import tensorflow.keras.backend as K

from transformers import AutoTokenizer, TFAutoModel, TFBertModel,
    logging

from utilities import *

logging.set_verbosity_error()

train = pd.read_csv('train_final.csv', index_col=[0]).fillna(' ')
test = pd.read_csv('test_final.csv', index_col=[0]).fillna(' ')


def class_block(inputs):
    X = tf.keras.layers.BatchNormalization()(inputs)
    X = tf.keras.layers.Dense(512, activation='relu')(X)
    X = tf.keras.layers.Dropout(0.2)(X)
```

```
29      X = tf.keras.layers.Dense(128, activation='relu')(X)
30      X = tf.keras.layers.Dropout(0.2)(X)
31
32      y = tf.keras.layers.Dense(3, activation='softmax', name='outputs'
        )(
33          X)  # 3 labels due to three sentiment classes
34
35      return y
36
37
38  #optimizer = tf.keras.optimizers.Adam(0.0001)
39  # Define configuration parameters
40  initial_learning_rate = 0.0001
41  lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
42      initial_learning_rate,
43      decay_steps=100000,
44      decay_rate=0.96,
45      staircase=True)
46
47  optimizer = tf.keras.optimizers.RMSprop(learning_rate=lr_schedule)
48  loss = tf.keras.losses.CategoricalCrossentropy()
49  acc = tf.keras.metrics.CategoricalAccuracy('accuracy')
50  iterations = 30
51
52  seq_len = 50
53  seq_len2 = 10
54
55  checkpoint_filepath = '/tmp/checkpoint'
56  model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
57      filepath=checkpoint_filepath,
58      save_weights_only=True,
59      monitor='val_accuracy',
60      mode='max',
61      save_best_only=True)
62
63  model = "unideeplearning/polibert_sa"
64  tokenizer = AutoTokenizer.from_pretrained(model)
65  bert = TFBertModel.from_pretrained(model)
66
67  CNN = tf.keras.applications.ResNet50V2(
68      include_top=False, weights='imagenet', input_tensor=None,
        input_shape=(224, 224, 3), pooling=False, classes=3)
69
70  dataset = df_to_tf_data(df=train, tokenizer=tokenizer,
71                          SEQ_LEN=seq_len, SEQ_LEN2=seq_len2, emotic=
        True,
```

```
72                          txt=True, imtxt=True, image=True, shuffle=
    True)
73
74 valset = df_to_tf_data(df=test, tokenizer=tokenizer,
75                         SEQ_LEN=seq_len, SEQ_LEN2=seq_len2, emotic=
    True,
76                         txt=True, imtxt=True, image=True, shuffle=
    False)
77
78 input_ids = tf.keras.layers.Input(
79     shape=(seq_len,), name='input_ids', dtype='int32')
80 mask = tf.keras.layers.Input(
81     shape=(seq_len,), name='attention_mask', dtype='int32')
82 input_ids2 = tf.keras.layers.Input(
83     shape=(seq_len2,), name='input_ids2', dtype='int32')
84 mask2 = tf.keras.layers.Input(
85     shape=(seq_len2,), name='attention_mask2', dtype='int32')
86
87 input_ids_c = tf.keras.layers.concatenate([input_ids, input_ids2])
88 mask_c = tf.keras.layers.concatenate([mask, mask2])
89
90 image_inputs = tf.keras.layers.Input(shape=(224, 224, 3), name="
    images")
91
92 text_embeddings = bert(input_ids_c, attention_mask=mask_c)[
93     0]  # we only keep tensor 0 (last_hidden_state)
94 text_1d = tf.keras.layers.GlobalMaxPool1D()(
95     text_embeddings)  # reduce tensor dimensionality
96
97 image_embeddings = CNN(image_inputs)
98 image_1d = GlobalAveragePooling2D()(image_embeddings)
99
100 X = tf.keras.layers.concatenate([text_1d, image_1d])
101
102 y = class_block(X)
103
104 model = tf.keras.Model(
105     inputs=[input_ids, mask, input_ids2, mask2, image_inputs],
    outputs=y)
106
107 # freeze the DistilBERT layer
108 model.layers[7].trainable = False
109 model.layers[8].trainable = False
110
111 model.compile(optimizer=optimizer, loss=loss, metrics=[acc])
112 model.summary()
```

```
113
114 os.environ['PATH'] = os.environ['PATH']+';' + \
115     os.environ['CONDA_PREFIX']+r"\Library\bin\graphviz"
116 tf.keras.utils.plot_model(model, to_file='final_model.png')
117
118 #history = model.fit(dataset,validation_data=valset, epochs=3,
        callbacks=[model_checkpoint_callback])
119
120 model.load_weights(checkpoint_filepath)
121
122 # model.save_weights('saves/final_weights')
123
124
125 model.load_weights('saves/final_weights')
126
127 y_pred = model.predict(valset)
128 y_true = np.concatenate([y for x, y in valset], axis=0)
129 y_pred_n = np.argmax(y_pred, axis=1)
130 y_true_n = np.argmax(y_true, axis=1)
131 test2 = test.copy()
132 test2['pred'] = y_pred_n
133 test2['pred'] = y_pred_n
134
135 confusion_matrix_plotter(true=y_true_n, pred=y_pred_n, normalize='
        true',
136                          title='full model confusion matrix', save=
        True)
137
138 predictions = test2.replace(0, 'negative').replace(
139     1, 'neutral').replace(2, 'positive')
140
141 predictions.to_csv('predict.csv', index=False)
```

Ringraziamenti

Questo spazio lo dedico alle persone che, con il loro supporto, mi hanno aiutato in questo percorso di durante i cinque anni universitari.

Grazie a mia madre, che mi ha sempre supportato in ogni mia scelta, e a mio fratello.

Grazie ad Eleonora, dolce compagna di vita.

Grazie al mio relatore Pierpaolo Brutti, non solo per avermi seguito in questo ultimi mesi, ma per essere stato presente e disponibile dal primo all'ultimo giorno di questi due anni. Per aver reso divertente perfino la statistica. Per i cioccolatini del giovedì. Per la dedizione all'insegnamento.

Grazie a tutto lo staff dell'azienda Elis, in cui ho svolto un tirocinio formativo della durata di 4 mesi complementare alla redazione della tesi, per l'ospitalità e per le skills acquisite sul campo, specialmente al mio Tutor Lorenzo Baiocco che ha saputo attutire al meglio il mio impatto col mondo del lavoro, a Luigi de Costanzo, a Lorenzo Ricciardi Celsi, ed ad Andrea B. Leone.

Grazie a Lorenzo M., che mi ha insegnato a studiare.

Grazie a Claudio, che mi ha insegnato a programmare.

Grazie ai miei colleghi universitari, sia Giorgio, Giulia, Livia, Lorenzo G., Federica, che hanno reso leggere le mie giornate.

Grazie alla truppa catalana composta da Giovanni, Leonardo, Roberto, Noemi, Vito e Luca, che hanno reso il mio erasmus indimenticabile.

Grazie ai miei amici per esserci sempre stati, in particolare a Simone, Tiziano, Giacomo, Alice, Giovanni e Federico.