eman ta zabal zazu

Universidad del País Vasco Euskal Herriko Unibertsitatea

Bachelor Degree in Computer Engineering
Computation

Thesis

Deep neural networks and data augmentation for semantic labelling in a dialogue corpus

Author
David Sandström Daparte

informatika fakultatea facultad de informática

2020

Abstract

Sentiment analysis, also known as opinion mining, refers to the use of Natural Language Processing (NLP), among other techniques, in order to extract and analyze subjective information from text, such as emotions or the topic of a text. These techniques are normally applied to reviews or data from social media but, in this project, we will apply these techniques to the analysis of coaching dialogues involving senior adults. These dialogues have been collected as part of the EMPATHIC project.

EMPATHIC is an European project whose goal is to implement a virtual agent designed to help elderly to live a healthy and independent life as they age [1][2]. Within this implementation, a Natural-language Understanding (NLU) component plays the role of classifying the utterance (spoken words) of the user into semantic components. This is a machine learning classification problem where there are multiple classes and a model has to be taught to classify the text into these classes.

Currently, the NLU model implementation is based on seq2seq models (a variant of Recurrent Neural Network (RNN) networks). However, convolutional neural networks have been also proposed for text classification in different contexts [3][4][5].

The main objective of this project will be to address a topic classification problem using Convolutional Neural Network (CNN) based architectures in order to classify the data from the Empathic project's dataset. Besides that, we will also propose and test a number of architectures based on RNN in order to provide some comparison of the performance from each model.

Contents

List of Figures

Table index

CHAPTER 1

Introduction

Large amount of text are being written everyday for the ever expanding storage of information in text format. All these documents do not have any mark or identifier and, given the vast volume of published data, it is mandatory to find an efficient, stable and accurate mechanism to extract and classify all this information according to our needs. That is where NLP has its niche and is constantly finding new ways to perform, for different type of problems, a successful method to extract, classify or manage all this information and apply it in a suitable context of interest.

The main objective of the project is to semantically label data, so that the label assigned to the text corresponds with the context to which the data belongs to, and represents accordingly its particular characteristics. To solve this task, we will use a Deep Neural Network (DNN) based model. This model will have the task of finding the relation between a set of features and its corresponding label, assigning the correct one to each unit of data.

More concretely, the DNN that we will use is a CNN, frequently used in image or video related tasks (e.g. Image classification or Video generation), and also capable of being used in the NLP scope. For this concrete task, it specially stands out in the following:

- Taking into account the order of the words.
- Evaluating the semantic connotation of a word in a particular context.
- Evaluating the surroundings of each word, to complement the current one and finding correlations that words only by themselves lack of.

1.1 Summary of sentiment analysis

Sentiment analysis consists on the use of Natural Language Processing and text analytics, among other disciplines, in order to analyze subjective characteristics of the data to, for example, identify the topic of a sentence or separate positive opinions from negative ones. This is also known as **Opinion Mining** [6].

These techniques can be applied to many different dimensions of text. As described in [7], among others, they can be applied to full documents or, as it will be the case in this project, to sentences. Also, some use cases of these techniques are:

- Analysis in the area of reviews of consumer products and services.
- Monitoring the reputation of a specific brand on social media.
- They enable campaign managers to track how voters feel about different issues and how they relate to the speeches and actions of the candidates.

1.2 Description of the problem

Our work in this project will be driven by the Empathic project [1] [8]. In this project, a dataset containing sentences collected from conversations between elder people, has been obtained. For this dataset, each sentence has been given a label by experts according to the topic it belongs to. Some examples of labels might be *family* or *nutrition*.

The problem that the provided dataset labelling carries is that the labels needed to be assigned manually. Therefore, the main objective of this project will be to propose a mechanism that is able to automate this process using machine learning techniques. More concretely, our objective will be to use techniques based on CNN models.

Lastly, we will also propose solutions based on RNN models as their use is widely spreaded in the context of sequential data[1] and because the current solution for this automation is based on them. This way, we will also be able to compare the obtained results provided by both architectures.

In addition to the model design, this project investigates and solves a number of related subproblems such as: word embedding selection, hyperparameter optimization and data augmentation techniques.

[1]A sentence can be interpreted as a sequence of words with a certain meaning.

1.3 Summary of the procedure

The main objective of this project will be to automate the labelling of the sentences from the Empathic's dataset according to the topic each sentence belongs to. For this purpose, we need o address the following problems:

1. The first problem that we will have to solve comes from the size of the dataset. It is formed by approximately 2000 sentences. Clearly, insufficient amount to train a DNN model.
2. A second problem comes from the design of the DNN architectures. We need to find an architecture that is able to properly solve the problem, without having too many parameters.
3. The third problem comes from the need to obtain the best possible performance out of the DNN models that we will propose. Because the performance of an architecture can have great variations just changing the value of some hyperparameters.

These problems will need to be solved in the same order they were mentioned. To solve them, we will propose, develop and implement the following solutions:

1. In order to have enough data to train the DNN models, we will propose two data augmentation techniques that will be used to generate a new dataset large enough to train a DNN. This process is detailed in Chapter 6.
2. In order to find an architecture that is able to solve the problem satisfactorily, we will first investigate which embedding is more appropriate and provides better results. Then, we will need to find a DNN model that can solve this problem with a reasonable amount of parameters. For that, we will propose and test several CNN and RNN configurations. This process is detailed in Chapter 7.
3. In order to obtain the best possible performance out of our DNN, we will need to optimize some of its hyperparameters. For that, we will use two optimization techniques. The optimization process is detailed in Chapter 8.

Besides the development of the solution, we will first cover some theory in the topic of Neural Networks that will be needed along the project. We will also cover what will be our background for the project, as well as a summary of the current state of the art in DNN for text related tasks.

CHAPTER 2

Project Management

The project management will be taken on the basis of a traditional evolutionary approach, as we believe that the complexity of the project does not require a more sophisticated methodology, such as PMR or similar. In this chapter, we will focus on the following two Project Management topics:

- The **risks** assessment and management approach.
- The **Gantt chart** of the project.

2.1 Risk assessment

We identified the following meaningful assets for the purpose of a risk assessment:

- The original dataset. The dataset provided from the Empathic project.
- The augmented dataset. A new dataset, as a results of the data augmentation process.
- The data augmentation algorithm.
- The DNN's architectures.
- The hyperparameter optimization method.

For all of them, we have undertaken a risk assessment and identified the most relevant ones. For each of the risks we have assessed the likelihood and the potential impact.

2.1.1 Identified risks

The identified risks for each asset are presented in Table 2.1.

ID	Asset	Risk	Likelihood	Impact
1	Original Dataset	Not predictable.	Low	High
2	Data Augmentation Algorithm	The generated dataset is not large enough to obtain sufficiently accurate results.	Medium	Medium
3	DAA	The computational load is too high.	High	Low
4	Augmented dataset	It does not contain enough information to classify the original dataset.	Low	High
5	Architectures	It is not capable of capturing enough information from the data.	Medium	High
6	Architectures	The designed architecture is too complex.	High	High
7	Optimization method	It is not capable of finding an hyperparameter combination that provides accurate enough results.	High	High
8	Optimization method	Loss or lack of consistency in the obtained results.	Low	High
9	Optimization method	The computational load is too high.	High	Low

Table 2.1: Likelihood and potential impact of each risk.

2.1.2 Risk evaluation criteria

In order to classify the risks, we applied the criteria exposed in Table 2.2, which gives us a classification for each risk according to the likelihood and impact.

Risk		Likelihood		
		Low	Medium	High
Impact	Low	Low	Low	Medium
	Medium	Low	Medium	High
	High	Medium	High	Very High

Table 2.2: Risk evaluation criteria

Regarding the risk evaluation criteria, we will focus in those risks classified as **High** or **Very High**. For those, we shall propose mitigation initiatives. We consider that it is not needed to set concrete mitigation initiatives (beyond common sense) for risks classified as Low or Medium.

2.1.3 Risks classification

According to the risk evaluation criteria stated in Table 2.2, the identified risks described in Table 2.1 results in the following classification, as shown in Table 2.3.

ID	**Asset**	**Risk**	**Risk**
1	Original Dataset	Not predictable.	Medium
2	DAA	The generated dataset is not large enough to obtain conclusive and consistent results.	Medium
3	DAA	The computational load is very high.	Medium
4	Augmented dataset	It does not contain enough information to classify the original dataset.	Medium
5	Architectures	The designed architecture is not capable of capturing enough information from the training data.	High
6	Architectures	The designed architecture is too complex, generating unacceptable computational load.	Very High
7	Optimization method	It is not capable of finding a hyperparameter combination that provides accurate enough results.	Very High
8	Optimization method	Loss or lack of consistency in the obtained results.	Medium
9	Optimization method	The computational load is too high.	Medium

Table 2.3: Risk's classification

2.1.4 Risk management

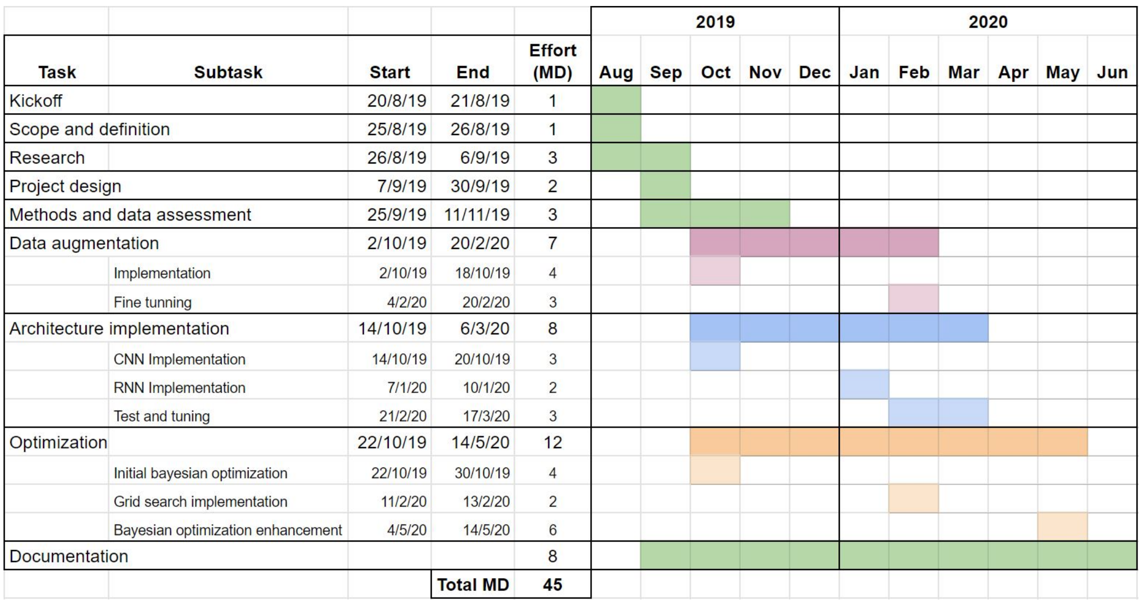
The risk mitigation initiatives proposed in order to manage the **High** and **Very High** risks identified in Table 2.3, are explained in Table 2.4.

ID	Asset	Risk Mitigation Initiative
5	Architectures	In order to improve the performance of our models, we will test architectures based in CNN and RNN. We assume that by applying the initiative, the likelihood will be reduced to low. Resulting in a Medium risk.
6	Architectures	In order to keep the complexity of the models in a reasonable level, we will use sequential as well as parallel architectures. We will apply it for both, the CNN and RNN. We assume that by applying the initiative, the likelihood and the impact will be reduced to medium. Resulting in a Medium risk.
7	Optimization method	We will extend the research period in order to find the suitable region of the hyperparameter space that is more likely to provide suitable results. On top of that, we will combine the results obtained using two different algorithms. Grid Search and Bayesian Optimization. We assume that by applying both initiatives, the likelihood will be reduced to low and the impact to medium. Resulting at the end in a Low risk.

Table 2.4: Risk mitigation initiatives

2.2 Gantt chart

Figure 2.1 shows the distribution of all the tasks performed in order to complete the project. Also, the starting and ending date of each task and the effort that each one took.



					2019					2020					
Task	**Subtask**	**Start**	**End**	**Effort (MD)**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**
Kickoff		20/8/19	21/8/19	1	🟩										
Scope and definition		25/8/19	26/8/19	1	🟩										
Research		26/8/19	6/9/19	3	🟩	🟩									
Project design		7/9/19	30/9/19	2		🟩									
Methods and data assessment		25/9/19	11/11/19	3		🟩	🟩	🟩							
Data augmentation		2/10/19	20/2/20	7			🟥	🟥	🟥	🟥	🟥				
	Implementation	2/10/19	18/10/19	4			🟥								
	Fine tunning	4/2/20	20/2/20	3							🟥				
Architecture implementation		14/10/19	6/3/20	8			🟦	🟦	🟦	🟦	🟦	🟦			
	CNN Implementation	14/10/19	20/10/19	3			🟦								
	RNN Implementation	7/1/20	10/1/20	2						🟦					
	Test and tuning	21/2/20	17/3/20	3							🟦	🟦			
Optimization		22/10/19	14/5/20	12			🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	
	Initial bayesian optimization	22/10/19	30/10/19	4			🟧								
	Grid search implementation	11/2/20	13/2/20	2							🟧				
	Bayesian optimization enhancement	4/5/20	14/5/20	6										🟧	
Documentation				8		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
			Total MD	**45**											

Figure 2.1: Gantt chart

CHAPTER 3

Introduction to Neural Networks

An Artificial Neural Network (ANN)[9] is a machine learning model inspired by the biological neural networks [10] that compose the brain of a human being. Same as humans, this model will need to learn from previous experience in order to solve a task. This "previous experience" will normally be represented as a dataset in this context.

The structure of an ANN is based on a collection of connected nodes named artificial neurons, which are used to process all the information that has to go through the network. Each neuron will be connected to other neurons in order to transmit the information and to process the data to produce the output[1].

The artificial neurons are normally grouped in layers, each layer receiving the output of the previous layer as input, processing it, and forwarding the result to the following layer. This class of models are known as feed-forward neural networks and are the main focus of this project.

The layers of an ANN can be classified using the following criteria:

- **Input layer**: receives the input data and perform the first processing.
- **Output layer**: this layer takes care on generating the output of the model.
- **Hidden layers**: these layers are optional and are placed between the input and the

[1]Depending on the architecture, there are some kind of Neural Networks that not necessary follow this scheme. But the architectures that will be used for this project will.

output layer. These extra layers will help the network to perform more complex operations to the input data, extending the capacity of the model.

Depending on the amount of hidden layers of the model, the networks can be classified as:

1. Shallow Neural Networks.
2. Deep Neural Network (DNN).

And in Table 3.1, a few of their more relevant characteristics are presented:

Shallow Neural Networks	Deep Neural Networks
Small number of hidden layers. Used, in general, to learn simpler problems than those addressed with DNNs. Usually, they are very homogeneous in terms of the activation functions they use.	Formed by many hidden layers, being able to solve more complex problems. Integrate the steps of feature selection and feature understanding. Can learn decomposable representations of complex patterns into simpler patterns. Organized as hierarchical features, from simpler patterns in the initial layers to more complex patterns in subsequent layers.

Table 3.1: Comparison of some of the most relevant differences between Shallow Neural Networks ans Deep Neural Networks.

It is important to mention that within each of the two categories presented in Table 3.1 there are many subtypes, each one designed for a certain task. Here are some examples of three different problems that are addressed with different types of networks:

- Image Recognition; Convolutional Neural Networks.

- Dimensionality reduction; Autoencoders.
- Time series prediction; Recurrent Neural Networks.

Among other models. Since there are many types of Neural Networks to cover them all, we will focus in the most important types regarding this project.

3.1 Artificial Neural Networks

ANN are the most recognizable type of network and can be used on their own or as a part of a more complex network. This type of networks are mostly used for supervised machine learning problems, as the following ones:

- **Classification**: tasks were the value to predict is a categorical value. It can only take a certain set of discrete values, e.g. predicting whether it will be rainy tomorrow.
- **Regression**: the value to predict is a continuous value, e.g. predicting the amount of rain expected for tomorrow.

Depending on the complexity of the problem, we will need a simpler or more complex model (perceptron or multilayer perceptron) in order to solve the problem[2]. These networks are formed by the following components:

- **Weights**: these are the parameters of the Neural Network. They will be used to obtain a linear transformation of the input features in order to compute the output of the model.
- **Bias**: this is a parameter added to the linear transformation to adjust the output value of the model in order to fit the problem.
- **Activation Function**: will be used to determine how the neuron should be activated. Some of the most popular activation functions are: Sigmoid, Tanh and ReLU.

Besides from that, all the weights of the network are initialized randomly and therefore their values will have to be learned in order to fit the problem. This process will be solved as an optimization problem where all the weights will be iteratively updated. Also, for any NN model there are two concepts that should be kept in mind:

[2]The perceptron is a model with only one neuron and, therefore, its processing capabilities are very limited.

- Parameters: the biases and weights.
- Hyperparameters: the value of these parameters are set when the DNN is created. Their value will remain constant through the learning process and afterwards (e.g. the number of neurons in a layer).

3.1.1 Multilayer Perceptron

This network is formed by connecting many perceptrons in order to extend their complexity and capabilities. In this model, the perceptrons are grouped in layers and represented as neurons. Each neuron is connected to all the neurons in the previous layer (receiving their outputs as input), and using its output as input of all the neurons in the next layer. Because of that, these layers of neurons are also knows as **fully connected layers**.

3.2 Convolutional Neural Networks

In the spectrum of the different coexisting neural network models, CNN [11] are a reliable neural network model used for many applications. In this case, the CNN have been used for NLP. In fact, they can be used for any problem were the inputs can be represented as matrices, even accepting multi-channel inputs (e.g. images in the three colour channels). In order to extract the features of the input data, these networks rely on the convolutional operation.

3.2.1 Architecture

Convolutional neural networks are mainly formed by three different types of layers:

- **Convolutional layers**: these layers are applied to the input data in order to produce a transformation in it that exposes the most important features, regarding the problem that the Network is trying to solve. This transformation can also hide irrelevant information. For example, it could blur the background of an image and highlight the foreground.

  An important difference to highlight in the convolutional layers with respect to the previously presented fully-connected layers, is that the weights of these layers are shared.

- **Pooling layers**: this layer's main objective is to subsample the input feature map in order to reduce the computational load, the memory usage, and the number of parameters.
- **Fully connected layers**: these layers will receive all the information, extracted by the previous layers[3], from the input data. With that information, these layers must produce the final output of the model. The output type will vary depending on the problem that the network is designed for.

  This can be seen as that this layer's task is to interpret the information extracted by the convolutional and pooling layers to generate an output.

Taking this structure into consideration, one of the most relevant applications for convolutional neural networks is **image classification**, since they can extract and shrink into a more compressed format all the relevant information from the image in order to differentiate different types of images. Obtaining these intermediate images, the computational cost of processing images is reduced considerably.

For obtaining a compressed yet explanatory version of the input data, feature maps are created transforming the data applying the filters to it. Through this process, a more condensed and meaningful decomposition of the input is possible. Now, we will cover each type of layer that compose this type of models.

3.2.2 Layers of the network

Convolutional Layers. The first component of the Network is a **convolutional layer**. The input data is modified by the **filters** of the layer in order to produce an intended result. These filters are also known as **kernels** or **masks**. The size of the filters has to be defined by the user, defining the dimension of the area that the filter will be covering to apply a **convolution** to the input data. Depending on the complexity of the input data, we will need to vary the depth of our net. Because, it is to say that the filters start by showing very subtle and more focused information of each region to end up gathering the different parts and understanding the information contained in the sample.

Another thing to consider in a convolutional layers is **how many receptive fields**[4] should be in total from which to extract information. This number is determined by the following parameters of the layer:

[3] After a process called flattening.

[4] Receptive field: the region of the input data to which the convolution is going to be applied.

- **Kernel size**: this determines the size of the receptive fields from which the information will be extracted.

- **Stride**: This decides how many units will the area of the filter be moved before applying the filter once again. This essentially works advancing the filter from one side to the other and from top to bottom.

Pooling Layers. After, at least, one convolutional layer has processed the input, a pooling layer can be applied. It is used to **reduce** the size of the output from the previous convolutional layer and reduce the computational load. There are **no parameters** to learn in the pooling layer. Instead of that, the output of the previous layer is divided in regions and, to each region, a specified statistical operation is performed to compute the output of each region. This is done following a predefined criteria based on a statistical transformation which will reduce the size of the region into a single value.

Fully Connected Layers. These layers are the **last component** of the convolutional neural networks. They behave similarly to the MLP discussed in Section 3.1.1 and their purpose is to process the results obtained by the previous layers[5] and produce a result based on that information.

3.3 1D Convolutional Neural Network

This type is used for **text related** tasks. The input data is represented by n vectors of dimension k. These vectors are known as the **embeddings** of the words that will be covered in Section 7.1.1, each vector identifying one word of the input text in the position it appears on. This architecture is very similar to the 2D convolutional neural networks used in image related tasks.

The filter, in this case, will process the **sequence of embeddings** (the text). The width of the filter is defined as a hyperparameter, usually called the window size in the context of NLP with 1D CNN. The height is always fixed to the embedding dimension and, therefore, the convolution is performed in only one direction, from the beginning to the end of the text. Because of this fact, this type of CNN is said to be **one dimensional** despite processing matrices.

[5] Taking into consideration convolutional and pooling layers.

Padding is only applied at the beginning and the end of the embedding sequence. Otherwise, the filter dimension would have to increase in order to cover the entire input as it will be different than the embedding size.

In the **pooling** stage, in the case of text classification, the pooling will be performed in only one dimension. In this case, the **pooling size** will be $(1, X)$, been always a one dimensional array.

The hyperparameters needed to define a convolutional layer inside a CNN are the following: 1) Number of filters. 2) Kernel size. 3) Stride. 4) Padding. 5) Activation function. 6) Regularization term.

3.4 Recurrent Neural Networks

Unlike the Convolutional Neural Network, this type of Neural Networks are meant to process **sequential information**.

Recurrent Neural Network are a family of models for processing sequential data. For this purpose, the neurons will store a hidden state of what the network has previously processed (the previous inputs of the sequence). Besides that, as the neuron will have to process the complete sequence, the weights of the neuron will be shared across all time steps, and the structure of the model will have the same size regardless of the sequence's length.

RNN neurons can be structured in layers and each neuron will have to process the complete sequence, same as convolutional filters. Besides that, the following three types of recurrent networks are worth mentioning for the course of the project:

- **Sequence to Sequence**: this network receives a sequence as input and outputs a complete sequence instead of a value. These are also known as Encoder-Decoder and process the information in two stages. First, an initial encoding stage that processes the input sequence producing a hidden state. Second, in the decoding stage, the output sequence is produced, conditioned on the hidden state from the encoding stage.

- **RNN with context**: to the input sequence we can add some extra information in the form of an initial state or an extra input in order to provide some context to condition the output of the model.

- **Bidirectional** RNN: this type of RNN processes the input sequence not only from beginning to end, but also from the end to the beginning. Depending on the problem, this can extract more information than processing the sequence only in one direction.

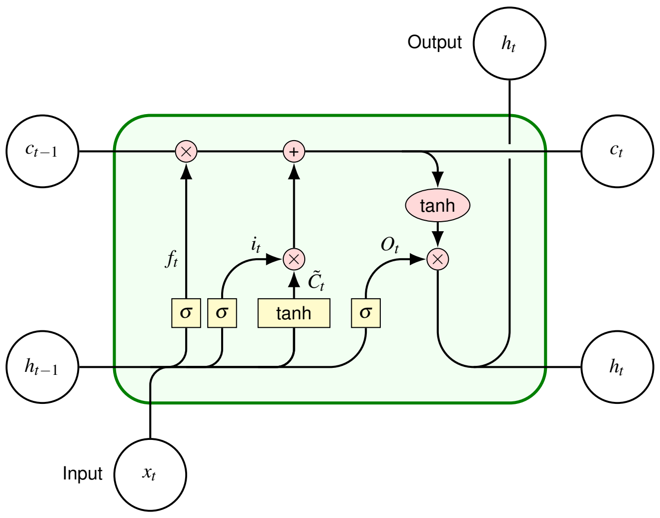
The remaining problem of these three architectures are the **Long term dependencies**.

3.4.1 Long term dependencies

Gradients propagated over many stages tend to either vanish or (rarely) to explode. This is caused because exponentially smaller weights are given to long-term interactions compared to short-term ones.

Among the existing solutions for this problem, for this project we will use the Gated RNNs. This networks are:

- Long Short-Term Memory (LSTM): it addresses the limitations of simple recurrent cells to represent long term dependencies in the data. The key idea is that the network can learn what to store in the long-term state, what to throw away, and what to read from it. For this purpose, the network will have two recurrent paths. One for the long term information and one for the short term information. The representation of an LSTM is shown in Figure 3.1[6]:

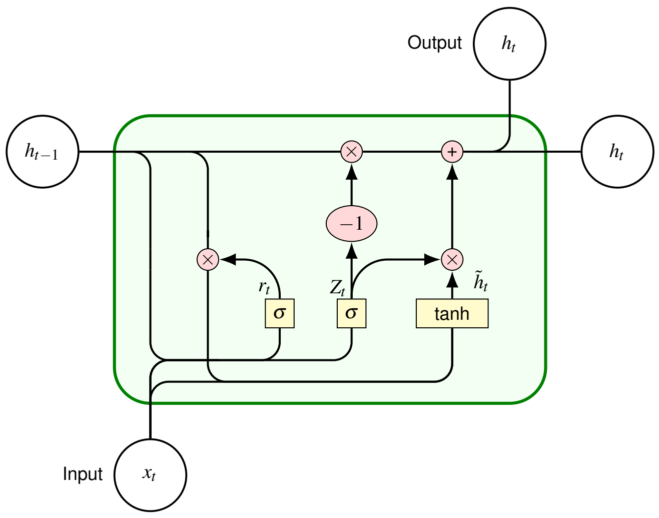


$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$
$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C)$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$C_t = i_t \cdot \tilde{C}_{t-1} + f_t \cdot C_{t-1}$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \cdot tanh(C_t)$$

Figure 3.1: Representation of an LSTM unit along its equations.

[6]This is a modification of the original, available in the following link.

- Gated Recurrent Unit (GRU): this is a simplification of the previously presented LSTM with similar performance. In this type, both state vectors are merged into a single output vector $h_{(t)}$. Only a single controller will control the forget state and the input state and there is no output gate. A representation of this model is shown in Figure 3.2:



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = tanh(W \cdot [r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3.2: Representation of a GRU unit along its equations

3.5 Other concepts

Now we will cover other important concepts for the project that are not necessary part of a CNN or RNN.

3.5.1 Dropout

This is a **regularization method** [12, 13]. Applying dropout to the network, implies that in each training step certain neurons from each of the layers[7] are totally ignored. Each neuron in each layer will be dropped in a random way, according to the **dropout probability** set as p.

[7]Including the input layer and excluding the output layer.

3.5.2 Data Augmentation

When the dataset for the problem is not large enough, the obtained results will be underwhelming. One approach to tackle this problem consists in "obtaining" more data in some way. This "added" data has to share the same context with the original data in order to provide satisfactory results. All in all, it is a method to "create" new input data for training from the original training data.

This technique provides a decrease in **generalization error**. Therefore, it is not far-fetched to consider data augmentation as a **regularization method**, as the generalization error will be reduced if this technique is implemented correctly. As a result, the performance of the model is heightened, thanks to a wider array of input data in the training session. Among the many approaches to data augmentation, there are some that aim to modify slightly the content in a way that still preserves the original label.

3.5.3 Loss Functions

Once our model is trained we will need a way to distinguish a good performing model from other models that might not perform as well. Besides from that, neural networks are optimized using a method called stochastic gradient descent [14], that will work by maximizing or minimizing the value of a concrete function. This function is called **loss function**.

Some examples of these functions are:

- **Binary Crossentropy**. This function is used for **binary classification tasks**. This function is computed as follows:

$$BCE = -\frac{1}{N} \times \sum_{i=1}^{N} y_i \times \log_2(p(y_i)) + (1 - y_i) \times \log_2(1 - p(y_i)) \tag{3.1}$$

- **Categorical Crossentropy**. This loss function is used for **multi-class classification tasks**. In order to use it, the output of the network should be processed by a softmax activation. This way, the output of each node is transformed in a probability value between $[0,1]$. This function is computed as follows:

$$L(y, \hat{y}) = -\sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} \times \log(\hat{y}_{ij})) \tag{3.2}$$

CHAPTER 4

State of the Art

The chapter summarizes some of the most important aspects from the current state of the art of machine learning applied to text related problems.

For that purpose, we will cover:

1. Some of the most common text related problems that are solved using CNN models.
2. A few techniques that can be used to optimize the hyperparameters of the network, as this process will be an important part of the project later on.
3. Some of the most relevant text related problems that are addressed using machine learning techniques.
4. Some of the most used implementations of NN models for this type of problems.
5. Data augmentation techniques applied to text.

We will finish this chapter explaining very briefly the functioning of some novel implementations that are used to create more powerful tools to solve these problems, as well as more computationally efficient.

4.1 Convolutional Neural Networks for text

Convolutional Neural Network are mainly known for its use cases in video and image based tasks. Such as image generation or video classification. Nevertheless, they are a very strong candidate to be used in text related tasks as sometimes they even outperform the classical RNN in this field. Some of the most common use cases for this kind of networks inside the NLP scope include:

- *Text recognition.*

  This problem consists on, given an image that might contain text, extract the text from it. An example implementation is presented in [15].

- *Text classification.*

  In these problems, the model will have to classify the text according to the correct class it belongs to. The work presented in [16] introduces an example of how this kind of architectures, without the need of either too many parameters or significant hyperparameter tuning, are able to obtain remarkable results in this task. In [17], an example of how to solve this task in a medical environment using a CNN is presented.

CNNs are not always used alone. Sometimes they are combined with other types of networks, such as RNN, in order to improve their capabilities. In [18], an example implementation that combines both types of networks in order to perform "handwritten text recognition" is presented.

4.1.1 Hyperparameter optimization

In order to obtain the maximum possible performance out of our architecture it is mandatory to optimize the value of its hyperparameters. For this purpose, many different techniques have been proposed. The following are some of the most commonly used techniques:

- **Manual Search**: based on previous experience or tests, tune the value of the hyperparameters by hand.

- **Grid Search**: define a set of values for each hyperparameter, generate all possible combinations and test each of them. Saving only the combination that provided the best results.
- **Bayesian Optimization**: this is an optimization technique based on the Bayes theorem. It will be covered in detail in Chapter 8 as it will be used in this project.
- **Genetic Algorithms**: these algorithms apply natural selection mechanisms in order to optimize the values of the hyperparameters.

A typical solution to optimize a network consists on not only using one of them, but a combination of at least two. This way, we can get a wider variety of tests, normally leading to better results.

4.2 Text related tasks solved using Machine Learning

Some of the most relevant text related problems that are solved using Machine Learning based approaches are:

- **Part of Speech Tagging**: this type of problem [19, 20] is also known under the name *grammatical tagging* or *word-category disambiguation*. This is the process of marking up a word in a text as corresponding to a particular part of speech (e.g. noun, verb, adjective, etc.), based on the context and its definition.
- **Named Entity Recognition**: this task [21, 22] consists on locating and classifying among a set of predefined categories, such as person names, locations, organizations, all the recognized name entities in a text.
- **Sentiment Analysis**: this task, also known by the name of opinion mining, refers to the use of natural language processing, computational linguistics and text analysis in order to extract the conceptual meaning and subjective information from text.

  This method is widely applied to social media [23, 24] in order to extract feedback from huge amounts of users in certain topics, among other use cases.
- **Latent Semantic Analysis and Indexing**: LSA [25, 26], in natural language processing, is a procedure that aims to analyze the relationships between a set of documents and the terms they contain. For that purpose, it analyzes the terms that they

contain, and produces a set of concepts related to the documents and, more concretely, to the previously found terms.

Once each set of concepts is generated, the sets generated by two documents are transformed to a mathematical representation and compared in order to obtain the similarity in terms of content of both documents.

4.3 Types of Architectures

In order to solve the task, we will not only need to choose a viable architecture that, given enough information, can produce the correct output. We will also need a representation of the data that contains as much information as possible, and is efficient. Some of the most commons representations of the data are:

- **One hot encoding**: this technique represents the categorical value of words into a binary vector. In this vector, all values are zeros but the index of the word in the vector, that is marked as 1.
- **Word embeddings**: this procedure is explained in Section 7.1.1.
- **Bag of words**: this technique is a representation of the occurrences of each word within a text without taking into consideration the order of the words. From a text, it produces a vector with as many positions as the vocabulary of the text. In each position, stores the amount of occurrences of each word.

Among the presented representations, the best will depend on the task to be performed. After that, we will also need to choose the architecture that best can fit our needs. For that, some possible solutions are:

- Recurrent Neural Networks.

  As this type of networks are used to process sequential data, it is to be expected that these networks are used in the scope of text related problems. Some problems like [27] are solved using this kind of networks.

  Normally, they are used in its Bidirectional variant, as there is some information that can be obtained from a sentence when it is processed backwards that can not be obtained processing it forward [28].

- Convolutional Neural Networks.

  This is the type of solution that we will explore in this project, following previous implementations such as [3].

- Networks that combine both CNN and RNN parts.

  Sometimes the combination of two tools might be much more powerful than the tools separately. Therefore, both types of architectures can also be combined to produce more powerful models, as proposed in [29].

4.4 Data augmentation for text

Some of the models that are widely used for NLP tasks need vast amount of data in order to produce proper results. Sometimes, that amount of data does not exist or is too expensive to obtain, as it is the case of hand labelled text. In order to extend the amount of data available to work at a reasonable cost, many possible solutions have been proposed. Out of them, we have decided to highlight the following:

- Synonym Replacement and Stemming.

  In this method, the original sentence will be modified to generate a new one applying two possible modifications at word level. We will choose a word and it will be replaced either by the most similar synonym or its stem. This process can be repeated *n* times.

- Word appearance labelling.

  This method is proposed in the following article [30] and will be tested for our concrete problem later on. This method is used to assign a label to unlabelled sentences when needed. In this method, we will define a set of relevant words for each possible label, count the occurrences of these sets of words in the sentence and assign the label corresponding to the most frequent set of words.

- Randomly manipulate a sentence.

  This method is proposed in the following article [31], and will be tested for our concrete problem later on. This method consists in applying the following modifications to the original sentences in order to generate new ones:

  - Synonym Replacement: Randomly choose n words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.
  - Random Insertion: Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position in the sentence. This process can be repeated n times.
  - Random Swap: Randomly choose two words in the sentence and swap their positions. This process can be repeated n times.
  - Random Deletion: Randomly remove each word in the sentence with probability p.

- Generative models.

  In this approach, generative models, such as GANNs [32], are used in order to generate new text that fit the needs of our project. An example of this kind of implementations is [33].

4.5 Novel implementations

Besides the classical RNN and CNN implementations, more modern solutions have been proposed in order to provide efficient and more powerful tools to address NLP related problems. Although we do not use these advanced models in this project, we have decided to highlight the following ones:

- **Attention Networks**

  Instead of focusing on the complete sentence, this architecture will focus on the most important part of the sentence for the current task. Some sentences might contain extra information, that is not needed for the current task and that can only add noise to the decision that will be performed by the network. This type of networks have been applied in different problems, such as [34] or [35].

- **Transformer**

  Traditional RNN have a computational limitation: they process the values of the sequence one by one. This architecture tries to address this problem, proposing an encoder-decoder[1] based architecture that is able to process the complete input sequence in parallel and not sequentially word by word.

This architecture works in the following way. For example, if the task was to translate from English to Spanish, the **encoder** will learn how the English language works. On the other hand, the **decoder** will learn how it can, given the information provided by the encoder, translate a sentence from English to Spanish. Using this kind of models some problems like [36] have been solved.

- **BERT**. Bidirectional Encoder Representation from Transformer [37, 38].

  These networks come from the idea of stacking the encoders from the transformer network, in order to generate a model that can understand language. In order to train these networks, the following two steps are needed:

  1. Pretrain: the objective of this stage is to teach the BERT what language and context are.
  2. Fine-tuning: fully-connected layers are connected to the output of the previously pretrained component, and trained to solve the concrete task that is needed.

  These networks can be used to solve problems like question answering, sentiment analysis and text summarization. Some problems that have been solved using this kind of networks include solutions such as [39] and [40].

[1]It should be noted that both parts include attention components.

CHAPTER 5

Deep Neural Networks for Semantic Labelling

In this chapter, we are going to present an architecture described in the article [41] as it presents a CNN used for text classification. This architecture is presented because it is going to be considered as the base architecture for the project, and all implementations of CNN used in this project are based on it. The explanation will be divided in two parts:

1. Explanation of the full pipeline for the model used in a binary class classification problem, going from the original data to the final classification.
2. How to use this model for predicting a multi-class problem.

The previously mentioned model is composed by the steps presented in Figure 5.1:

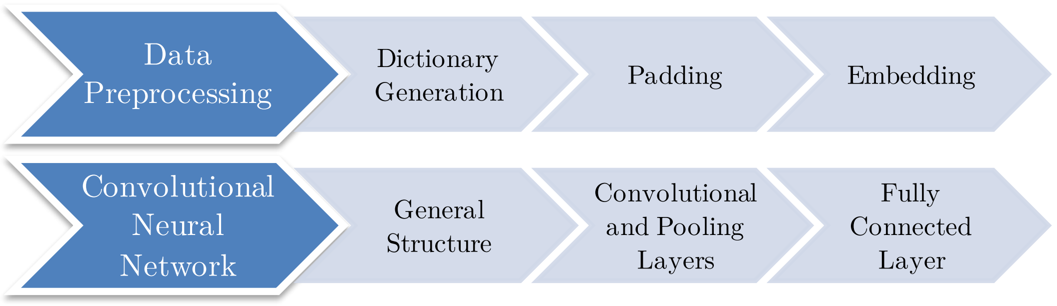


Figure 5.1: Pipeline Diagram

The purpose of this pipeline is:

1. Transform our inputs, all of them written in natural language, into a mathematical representation that our CNN [11] can recognize.
2. Extract the main features from the input data with a convolutional process.
3. Combine the information extracted to classify the inputs.

Once we have presented the model, it will be described step by step.

5.1 Preprocessing of the data

As we explained before, this model is based on a DNN architecture. Therefore, we cannot use natural language as input of our model. We first need to transform it to a mathematical representation in order to use it.

5.1.1 Dictionary Generation

The first step in this process will be to generate a **dictionary** with only the words that have a relevant appearance in the text. This means that only the words that appear in the text with a frequency above a fixed threshold will be considered in the classification process, and will have an entry in the dictionary. Once that dictionary is fully generated, we will replace all words in the text by the value of its entrance in the dictionary. In Table 5.1 we have an example of how a dictionary generated by this process looks like:

Word	Index
a	32
motor	654
car	342
fast	121
phone	76

Table 5.1: Example of the dictionary generated.

Taking into consideration the previous dictionary, for each sentence, we will substitude each word by its index in the dictionary, as the following example illustrates:

$$[\text{A, fast, car}] \rightarrow [32,\ 121,\ 342]$$

Once the input is transformed to a mathematical representation we can proceed with the next step.

5.1.2 Padding

We will first proceed by applying a **padding** process to the transformed input data. In this case, we will use the following parameters:

1. Padding is only applied to the end of each sequence.
2. The value of the elements added to the input is *0*, and therefore we are applying zero padding.
3. The amount of elements added to each row is at least *2*.

The neural network needs a fixed sequence length in order to perform all the computations. Therefore, this process will also fit all the sequences to the length of the longest sequence in order to not waste any information from the longest sequences.

After applying the padding, we can proceed with the **embedding**.

5.1.3 Embedding

Since embedding will transform each word to a fixed dimension array, we will need to set the following parameters beforehand:

1. The amount of words contained in the vocabulary computed in the previous part. This parameter will control the amount of embeddings that will be generated.
2. As the embedding process transforms each input value to a vector of real values with fixed length, this length needs to be specified.
3. Whether any kind of padding has been applied to the input.

After this step, each index from the vocabulary is transformed in a vector of real numbers, as illustrated in Figure 5.2, where each row of the matrix is an embedding:

$$[\text{A fast car}] \rightarrow [32\ 121\ 342] \rightarrow \begin{Vmatrix} 1.2 & 2.3 & 0.7 \\ 0.5 & 7.5 & 2.3 \\ 0.9 & 4.2 & 3.4 \end{Vmatrix}$$

Figure 5.2: Result of the preprocessing applied to the input text.

Something to notice is that the embeddings are generated without any knowledge of the word that they will represent or its context. Therefore, they will be generated randomly. It is during the training phase that the embeddings will be adjusted to the concrete meanings of the words in the input text that the DNN is processing.

After applying all those procedures, the data can be forwarded to the CNN.

5.2 Convolutional Neural Network

The NN model used is a non sequential CNN. The architecture of the network is described in Figure 5.3:

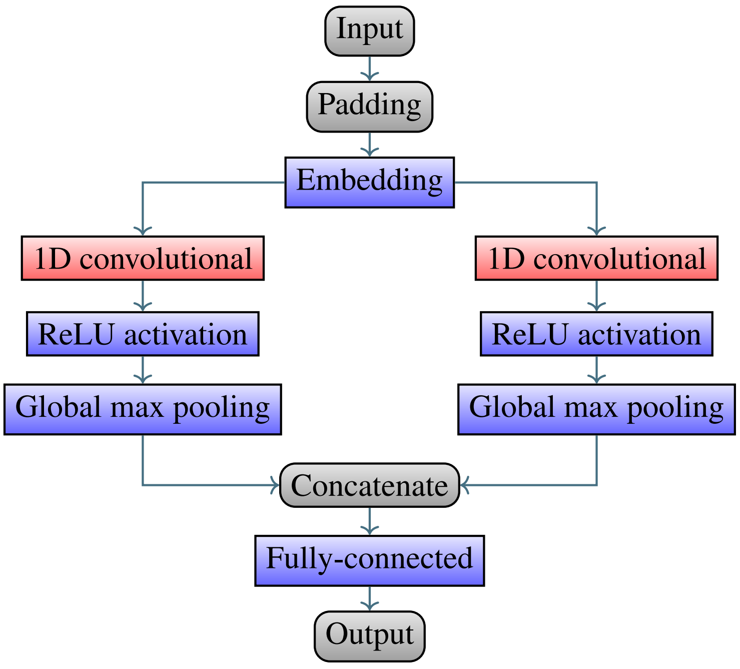


Figure 5.3: Diagram of the non sequential CNN model built, showing the structure of layers.

Some comments should be made about the architecture presented in Figure 5.3:

1. Both Convolutional Layers operate in parallel in order to extract all the possible information from the input data.
2. The intermediate processing, consisting on the ReLU activation function and a pooling layer, are performed individually for the output of each convolutional layer.

We will explain the model layer by layer in the following sections.

5.2.1 Convolutional layer

After the input text has been processed in the way that is described in Section 5.1, this data is forwarded to both convolutional layers, as both will process this information in parallel and not in a sequential way. This pair of layers will have the following characteristics:

- They both have the same amount of **filters**. The amount of convolutional layers can be extended from two to more layers.
- Each group of filters will use a different **window size** in order to extract different features from the text. The selected sizes will be 3 and 4.
- The **stride** of the convolution will be set to 1 so that we don't loose any information during the process.

Each layer will produce a separated output that will be forwarded separately to its correspondent ReLU layer.

5.2.2 Intermediate Processing

First, the results obtained by the convolution are transformed using an activation function, ReLU in this case. After the activation function is applied, all the results will go through a **GlobalMaxPooling** layer. This layer is a very similar utility layer to the regular max pooling previously presented to subsample the data, but with a very specific distinction. The term "global" is referred to this max pooling layer, since the **maximum value from each row**[1] of the input matrix is selected and forwarded to the following layer.

[1] Instead of the maximum of each $k \times k$ submatrix as in the regular max pooling.

Another interesting aspect is that the global max pooling is applied just before forwarding the values to the fully connected layers, without the presence of a "Flatten" layer. The global max pooling technique will receive as input a matrix, and will provide as output a one dimensional vector. Therefore, there is no need to introduce a "Flatten" layer as many CNN use.

As a result of this aspect, the input size of the fully-connected layer is reduced significantly, saving computational load, parameters of the model and memory space.

5.2.3 Fully Connected Layer

The last step needed in the network will be to generate the prediction given the features extracted by the previous layers. For this purpose, we will use a linear combination of this features, in the same way as a fully-connected layer would do. This combination is computed as follows, being x all the features extracted and w the associated weight of each feature:

$$y = \sum_{i=1}^{n} x_i \times w_i \tag{5.1}$$

As we can see in the equation 5.1, there is no independent term and we need to add it to complete the linear combination.

It is worth to notice that, in this case, the bias of the neurons in this layer is implemented by adding a new term to the input vector of this layer. In this way, this new term will act as the bias[2] and its value will be controlled using its associated weight. Once the new term is added, we obtain the following expression:

$$y = \sum_{i=1}^{n} (x_i \times w_i) + 1 \times w_0 \tag{5.2}$$

This way, the term w_0 will measure the impact of that added independent term (the bias) in the computation. This layer can be represented in the following way:

[2]This is the way that the author of the article decided to implement the bias.

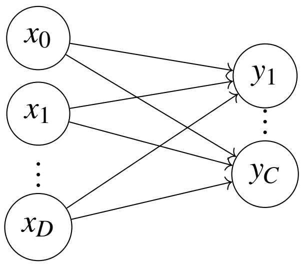


Figure 5.4: Representation of the Dense Layer of the CNN.

The dense layer of the network will have as many **inputs** as outputs the Global max pooling layers have, and as many **outputs** as possible values that the classification problem can take. This way, the output of the model will be generated as a linear combination of the features extracted by the convolutional layers.

Once every output of the model has been computed, each input of the model will be assigned to the corresponding class of the highest output value.

5.3 Multi-class Problem

We have presented the model used to solve the binary classification task but, in order to use the model from the previous section for a non-binary problem, the following should be taken into consideration:

1. The amount of outputs of the last layer of the network must be extended to the number of possible classes.
2. The loss function, *cross-entropy*, is already suitable for multiple (more than two) possible classes.

CHAPTER 6

Data Augmentation

In the data augmentation step, as it has been explained in Section 3.5.2, the purpose is to increase the amount of data from the original dataset provided for the project. This dataset, as previously mentioned, is **far from being large enough**[1] as shown in Table 6.1.

	Nutrition	**Leisure**	**Family**
Number of sentences	1680	360	129

Table 6.1: Distribution of topics in the original dataset

In this chapter, we will cover the data augmentation methods that we have used to expand the dataset, in order to be able to train a Deep Neural Network. These two methods are:

1. The first method consists on taking a dataset that contains many sentences extracted from subtitles, and try to label as many of them as possible. For that, we have to take into consideration the occurrence of some key words in each sentence. This approach has been originally proposed in [30].

2. A second method that consists on deleting and swapping randomly some words from the sentences obtained by the previous method, in order to improve the performance of the neural network. This approach has been introduced in [31].

[1]We estimate that something around 10 times larger should be enough to start working.

One aspect to take into consideration, is that the implementation of the models has been done using TensorFlow [42] and Keras [43][44] libraries.

Once this is clarified, we will go on with the detailed explanation of the methods implemented.

6.1 Implementation of the first method

To implement this first method, we will obtain a **new set of sentences** from the OpenSubtitles repository[2] in order to have enough data to train the models. The database contains millions of sentences extracted from subtitles in Spanish, as the dataset that we want to predict is in Spanish. A limitation of the dataset is that the new sentences are not labelled, and have to be labelled before they can be used.

For this purpose, we will define a set of approximately 30 relevant words for each of the three topics in the original dataset, in order to assign a label to each of the new sentences. These sets of words will be in English always, regardless the language of the sentences to label. This is because the words in the needed language, Spanish in this case, will be obtained later on. The three topics are the ones listed in Table 6.1:

- **Nutrition**: Words related to food or meal. They have some common information about the many types of eating habits and specific language in the field of gastronomy, as well as the importance of nutrition in the biological processes.
  - **Examples**: food, breakfast or catering.
- **Leisure**: Words related to different activities that can be part of the free time a person has available. Also, ways in which that free time can be conveniently used, or specific circumstances that only occur in that kind of situation.
  - **Examples**: game, play or recreation.
- **Family**: Words related to the environment in which familiar events may happen. Includes also the familiar relationships between members of the same family, and many of the possible exercises of family bond situations that could be in line with the topic.

[2]The database can be downloaded in many languages in the following link.

- **Examples**: parents, brother or fraternal.

However, as previously mentioned, this set of words is in English, not Spanish and is not very large. In order to expand the number of words of each topic and to have them in the corresponding language, we will use the **WordNet** [45][46]. *WordNet* is a lexical database which places together words that are synonyms, or have a related significance of each other in a common set called a **synset**. These synsets are represented as **nodes in a graph**. When two synsets are related to one another, this relationship is represented by connecting an edge between the two nodes.

This representation **establishes relationships** between synsets. Those relationships are useful to identify two synsets with similar meaning or used in a similar context. For example, if both of the synsets were to be present in the database and for the synset of *vegetables* and the synset of *food* there is a connection between their synsets, it would mean that their rate of apparition together is high enough to consider them similar.

For labelling the phrases of the Spanish subtitles dataset, (which are unlabelled and require this data augmentation process in order to have a class associated) we will count the number of matching words from each set of relevant terms (each set related to one topic), for every sentence. From there, the criterion to determine the label of each of the phrases is to check which one of the three sets is more frequent than the other two.

6.1.1 Datasets generated

From this process we will obtain two different datasets, depending on the minimum threshold of matching words set to label a sentence:

1. The first dataset will contain all the sentences where the most frequent topic only has **one occurrence** of a relevant word from the topic. This will result in a large dataset but with a low quality, as each sentence is weakly related to its assigned class.

2. The second dataset will contain only those sentences where the most frequent topic has, at least, **the occurrence of two** different words from the topic. This will result in a very small dataset but with much higher quality than the previous one, as the sentences from this dataset are expected to be more related to its class than the ones in the previous dataset (the dataset with only one occurrence of a relevant word).

Those sentences that don't have any occurrence of any relevant word from any topic are discarded and will not be used in the following training stages.

The two datasets generated by this process contain the distribution of topics presented in Table 6.2.

	Nutrition	**Leisure**	**Family**
One occurrence	670 008	723 116	2 397 546
At least two occurrences	14 568	9581	122 021

Table 6.2: Distribution of topics in the augmented datasets

As it can be observed in Table 6.2, the datasets are very unbalanced. We have to keep an eye on this during the testing, as it will heavily affect the performance of our model. These datasets will be referenced later on as the **high quality dataset** (At least two occurrences) and the **low quality dataset** (One occurrence).

6.1.2 Infrastructure required

The execution of this technique required a meaningful computational load. The dataset was formed by more than 20 000 000 sentences contained in thousands of XML files, all of them compressed in a zip file. The computational requirement of this method was too high for the hardware we had available locally. Therefore, we used two virtual machines hosted in the Google Cloud Platform in the Compute engine service.

Each machine had 24 CPU cores and 22 GB of memory. Using these machines, the generation of the datasets took approximately one day.

6.2 Second Data augmentation technique

The particular implementation of the second data augmentation technique has been done using a *Keras DataGenerator*[3] instance. The important thing to mention about it, is that this method will not modify the dataset itself but will modify the training procedure. More concretely, when the DNN needs a new batch of data to train, this method will be applied directly in that batch and not to create a new dataset.

[3]For more concrete information about this class please refer to the following link.

This method will apply the following modifications in each sentence of the current batch:

1. Randomly deleting **n** words from the sentence.
2. Randomly flipping the order of two words in the sentence. This modification is repeated **k** times.

We remark that this technique does not directly expand the amount of sentences in the datasets. Even though, the DNN will receive different training data in each training epoch as a procedure to improve its performance and its generalization capabilities.

Also, it should be mentioned that, as this method is based on applying random modifications to the data, the results can change between different training trials.

6.3 Testing and improvements

Now that we have a dataset that is large enough to train a DNN, we need to measure the performance that we can get from these datasets, so that we have a baseline for all the improvements that we will apply later on[4]. This testing will be performed following the testing methodology detailed in Section 6.3.1.

6.3.1 Testing methodology

In order to measure the performance obtained by the data augmentation, we will divide all our data[5] in three different datasets as follows:

1. **Validation set**: Comprises the first half of the original dataset[6].
2. **Test set**: Comprises the second half of the original dataset.
3. **Train set**: Composed by the augmented datasets, separating the high quality dataset and the low quality, as they will be used separately.

[4]Some improvements are also detailed in Section 6.4.

[5]The original dataset provided for the project and the augmented datasets.

[6]Before dividing this dataset into two, the dataset was shuffled in random order so that the class distribution of both halves is approximately the same.

Since the accuracy metric alone is not suitable to describe how our model performs because the dataset is very unbalanced, as presented in Table 6.1, we will additionally compute the **balanced accuracy** and the **confusion matrix** for the validation and test sets.

The DNN that will be used as a reference model in order to measure the performance obtained by these techniques, will be constructed using the architecture described in Figure 6.1:

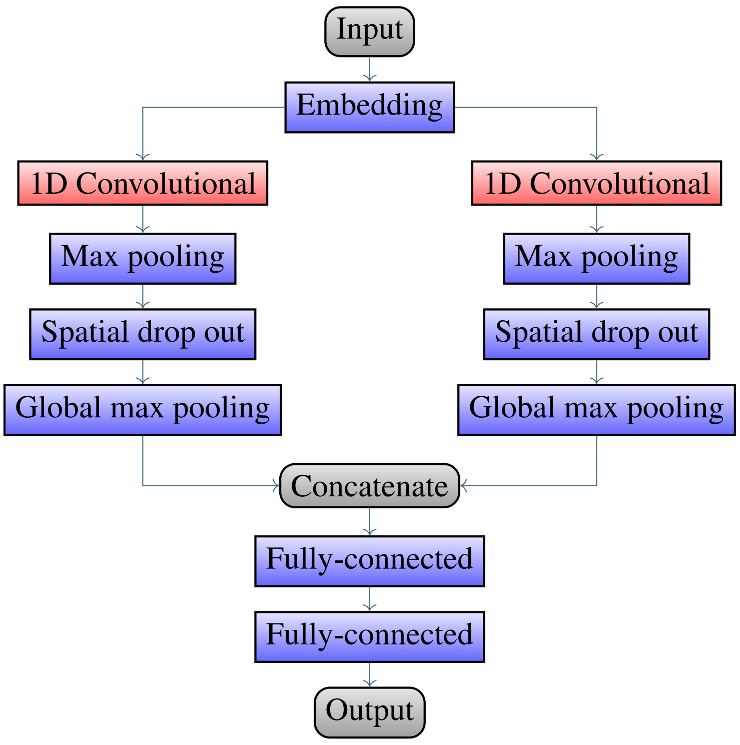


Figure 6.1: Reference DNN for the Data Augmentation

The training of the DNN was performed in two different ways:

1. Using as training set exclusively the high quality dataset.
2. Using the low quality dataset to perform a first train and the high quality dataset afterwards to perform a fine-tuning of the network.

Lastly, as it is not known in advance how many training epochs will be needed in order to get the best performance from the model, the number of epochs that each training takes will be controlled with a *Callback*[7] from the Keras library. This method will ensure that,

[7]More concretely, we used the EarlyStopping. For more information about it, refer to the following url: `https://keras.io/callbacks/`

as soon as the model starts to overfit, the training will stop and the weights from the network that provided the best results will be restored.

6.3.2 Results obtained from single training stage

Following the methodology that is explained in Section 6.3.1, Table 6.3 shows the results obtained training exclusively with the high quality dataset. In order to try to improve the results, the dataset will be modified. For this purpose, we will only take into consideration the results obtained in the validation set.

	Validation Set	**Test set**
Accuracy	59.59%	57.14%
Balanced Accuracy	48.47%	53.15%

Table 6.3: Accuracy and Balanced accuracy with one training stage

The analysis of the metrics shown in Table 6.3 is sufficient to confirm that the first data augmentation technique was successful. It is able to predict the sentences from the original dataset, only training with the new augmented sentences. Moreover, the results obtained in the test set are very similar to the results obtained in the validation set, making our results more consistent.

However, this information is not sufficient to determine how the dataset can be improved. Therefore, we will also check the confusion matrices, presented in Table 6.4 from both datasets in order to extract some information that can lead us to improve the datasets:

		Validation set				**Test set**		
		Predicted Value				**Predicted Value**		
		Family	**Leisure**	**Nutrition**		**Family**	**Leisure**	**Nutrition**
Real	**Family**	58	0	22		44	0	5
	Leisure	108	5	58		126	4	59
value	**Nutrition**	243	7	583		267	8	572

Table 6.4: Confusion matrices with single train

The analysis of the results presented in Table 6.4, reveals that the DNN trained with this dataset shows a trend of trying to classify every sentence as *Family*. These results are satisfactory only for sentences that belong to this class, as they are mostly well classified. But, if we look at the distribution of the predictions for the other two classes, the prediction has a high error rate as there are many sentences that are being missclassified as *Family*. Once again, the test set presents similar results as the validation set.

This can be due to the distribution of the sentences (the amount of sentences of each class) in the training dataset. As it can be seen in Table 6.2, the number of sentences from class *Family* is by far superior than the number of the other two. Because of that, we will reevaluate the results after balancing the dataset later on.

6.3.3 Results obtained from Train and fine-tuning

The results for both datasets, using the strategy that adds a fine-tuning step, are shown in Table 6.5.

	Validation Set	**Test set**
Accuracy	41.33%	34.19%
Balanced Accuracy	48.52%	46.42%

Table 6.5: Accuracy and Balanced accuracy using the strategy that adds a fine-tuning step

We can clearly see that the results presented in Table 6.5 are worse than the results obtained in the previous experiment as presented in Table 6.3. For this particular dataset and neural network, the approach of training first with a lower quality dataset and fine-tuning the network afterwards with a higher quality dataset, might not be as effective as it seemed in advance. This situation is even worse if we look at the results obtained in the test set.

To better understand the behaviour of the model, we inspect the confusion matrices which are shown in Table 6.6.

		Validation set				**Test set**		
		Predicted Value				**Predicted Value**		
		Family	**Leisure**	**Nutrition**		**Family**	**Leisure**	**Nutrition**
Real	**Family**	80	0	0		49	0	0
	Leisure	153	3	15		174	3	12
value	**Nutrition**	464	4	365		526	2	319

Table 6.6: Confusion matrices with train and fine tuning

Table 6.6 shows the same trend as the results presented in Table 6.4, but in a more severe way. In fact, if we only look at the results obtained in the sentences that belong to the class *Family*, we can see that they are all well classified. But if we also look at the results obtained in the other two classes, even more sentences than in the previous experiment are being missclassified as Family. This makes the problem even more severe using this training technique than the previous one.

Therefore, we will now repeat the same testing with both training techniques after balancing the amount of sentences from each class in the training datasets.

6.3.4 Results after balancing the datasets

Once the two datasets have been balanced, the distribution of sentences ends as shown in Table 6.7.

	Nutrition	**Leisure**	**Family**
One occurrence	670 008	670 008	670 008
At least two occurrences	9581	9581	9581

Table 6.7: Augmented datasets' distribution after balancing the number of sentences per class

It can be observed in Table 6.7 that, if we compare it to the distribution presented in Table 6.2, the previous problem that the number of sentences from the class *Family* is far higher than the other two is gone. The results achieved by the DNN training with this balanced dataset using the single training stage approach are shown in Table 6.8.

	Validation Set	**Test set**
Accuracy	71.03%	68.39%
Balanced Accuracy	58.41%	58.14%

Table 6.8: Results obtained with one training stage and the augmented datasets balanced

The results presented in Table 6.8 are much better than the ones previously presented in Table 6.3. Therefore, the distribution of the sentences in the datasets was heavily conditioning the results. Further more, the results have improved in both datasets in a similar ratio, making the model consistent and more capable of generalizing to new unseen data.

In the next step, we analyzed the confusion matrices, shown in Table 6.9, to determine whether the previous trend of missclassifying many sentences as *Family* is still present.

		Validation set				Test set		
		Predicted Value				Predicted Value		
		Family	**Leisure**	**Nutrition**		**Family**	**Leisure**	**Nutrition**
Real	**Family**	44	14	16		34	13	8
	Leisure	43	62	63		47	70	75
value	**Nutrition**	52	126	664		92	108	638

Table 6.9: Confusion matrices using a single training stage and the augmented datasets balanced

We can clearly see in Table 6.9 that the previous trend has disappeared. If we look at the results obtained by the class *Family*, they are worse than before as it is missclassifying more sentences from this class. However, considering the dataset as a whole, we can see that there are less sentences missclassified and much less missclassified as *Family*.

Another improvement that balancing the datasets brings is that, as the datasets has shrunken, the computational load needed to train the DNN has decreased significantly, improving as well the training times.

The results using the balanced datasets and the training plus fine-tuning strategy, are shown in Table 6.10. These results are presented because balancing the datasets clearly improved the results obtained in the single training stage method. Also, because the train and fine-tuning procedure shows the previously solved trend even clearer.

	Validation Set	**Test set**
Accuracy	40.13%	40.65%
Balanced Accuracy	48.90%	49.62%

Table 6.10: Accuracy and Balanced accuracy using the strategy that adds a fine-tuning step and the balanced datasets.

The results in Table 6.10 have improved with respect to those presented in Table 6.5. Nevertheless, they are not as good as we expected, mainly because this technique is not getting as big benefits from the balancing of the training datasets as the previously presented strategy. The corresponding confusion matrices are shown in Table 6.11. They allow to further analyse the results.

		Validation set				Test set		
		Predicted Value				Predicted Value		
		Family	**Leisure**	**Nutrition**		**Family**	**Leisure**	**Nutrition**
Real	**Family**	74	0	0		55	0	0
	Leisure	149	8	11		166	7	19
value	**Nutrition**	477	12	353		453	6	379

Table 6.11: Confusion matrices using the strategy that adds a fine-tuning step and the augmented datasets balanced.

Contrary to what could be expected by the results presented in Table 6.9, the trend of predicting most of the sentences as *Family* has not disappeared using this technique. This can be due to the noise that the low quality dataset has added to the DNN is larger than the information that it contributes to properly classify the sentences.

6.3.5 Results using the second data augmentation technique

As the performance has been clearly better using the single training stage method than the training and fine-tuning approach, only the first method will be tested with this new data augmentation technique. The results are shown in Table 6.12.

	Validation Set	**Test set**
Accuracy	66.33%	64.79%
Balanced Accuracy	60.79%	59.92%

Table 6.12: Accuracy and Balanced accuracy using both data augmentation techniques

If we compare the results from Table 6.12 to the previously obtained and presented in Table 6.8, we can see that the accuracy is slightly worse, but the balanced accuracy has improved. This is caused because the performance is more stable across the classes and worse in the most frequent class in this concrete dataset. This seems to indicate that our model will perform better regardless the concrete distribution of the dataset that it will have to face.

This observation can be confirmed from the analysis of the confusion matrices shown in Table 6.13.

		Validation set				**Test set**		
		Predicted Value				**Predicted Value**		
		Family	**Leisure**	**Nutrition**		**Family**	**Leisure**	**Nutrition**
Real	**Family**	52	16	6		38	14	3
	Leisure	44	69	55		55	78	59
value	**Nutrition**	89	155	598		79	172	587

Table 6.13: Confusion matrices using both data augmentation techniques

Comparing to the results presented in Table 6.9 this method performs slightly worse in the most frequent class but performs more evenly along all classes.

6.3.6 Summary of the results

During this chapter we have presented two data augmentation methods, as well as a technique that improved the obtained results. In Table 6.14, a summary of the results obtained in this chapter is presented. All of them, measured over the **validation set**.

	Accuracy	**Balanced Accuracy**
Single train stage	59.59%	48.47%
Train and fine-tuning	41.33%	48.52%
Single training stage, balanced dataset	71.03%	58.41%
Train and fine-tuning, balanced dataset	40.13%	48.90%
Second data augmentation method	66.33%	60.79%

Table 6.14: Summary of the results obtaining in this stage.

As shown in Table 6.14, the best results were obtained when only using the smaller dataset. With and without the second data augmentation method. Taking into consideration those results, we have decided that, from now on, we will only use the smaller dataset without the second data augmentation method. That decision comes from the instability of the results produced when using the second method, and that the produced results are not significantly better.

6.4 Lessons learned

This section presents some of the most relevant problems that were faced during the development of this phase of the project, and that might as well be helpful for other people trying to solve similar problems. These problems were related to:

- Subsequences.
- Accents in the Spanish language.
- Techniques for expanding the original set of words used in the first data augmentation technique.

6.4.1 Subsequences

This problem appeared during the implementation of the first data augmentation technique, covered in Section 6.1. For this data augmentation technique we need to check which words[8] are **subsequences** of the whole sentence, to count how many of them are included in the sentence and assign the corresponding label.

This carries the following challenge. For example, if the sentence is "El coche está averiado" and the word is "ave", the word will count for the sentence even though the word "ave" is not included in the sentence as a word but it is as a subword of "avería". This point was not taken into consideration during the implementation.

Instead of just checking if the word appears in the sentence, it is also needed to ensure that it appears as a complete word, and not as part of a different word. This way, the word will not be counted if it appears as a subsequence of other word. Adding this consideration to the implementation decreased the size of the dataset considerably. Approximately dividing it by 3 but, far from reducing the performance of the model, this improved its performance. A plausible explanation for this improvement is that many sentences that were removed from the dataset by this process were wrongly labelled, or not fully correctly labelled.

[8]From the set of relevant words of each topic.

6.4.2 Spanish accents

In Spanish, some words are written with accent such as "mamá" or "móvil". The problem with this characteristic is that we do not know if the dataset from where we will try to label the sentences in the first data augmentation technique, will or will not contain these words with accent. The previous words shown as examples could appear as well as "mama" and "movil".

If accents are not taken into account in the dataset, it can cause that many sentences will not be labelled simply because the word appears with or without accent. In order to solve that problem, for each word that we have in the set of Spanish words that has accent, the word without the accent will also be added to the set of words. By doing this, we can ensure that the sentence will be labelled regardless if it is written with or without accent.

6.4.3 Expand the English set of words using WordNet

In the first data augmentation technique, from a set of words in English we use wordnet to obtain an expanded set of words in the needed language (Spanish in this case) being those Spanish words searched in the corresponding sentences.

We can also add another step to this process in the following way: this original set of words (the set in English) can be expanded using wordnet (e.g. to go from 60 to 100 words), and use these 100 words instead of the original 60 to obtain the words in Spanish.

This method is consistent as the English words obtained from WordNet are related to the words that were added manually. Therefore, it might add some noise to the set, but not too much. The results obtained by this method did not improve significantly the performance of the model, but increased significantly the computational load needed to train a model and. Because of that, this method is not used for the final testing.

CHAPTER 7

Implementations of DNN and Embeddings

In this chapter, we will cover all the concrete implementations of DNN that have been performed in order to address the main tasks of this project. For this purpose, the following types of DNN have been implemented:

- Convolutional Neural Networks.
- Recurrent Neural Networks.

We used TensorFlow 2.1 for the implementation of all the DNN and the version of the Keras library that is included in TensorFlow 2.1, in order to avoid version compatibility issues.

Also, as the concrete representation of the words that is used in the DNN are word embeddings. Different word-embedding representations have been tested, in particular, the following two types:

- Embeddings learned during the training stage.
- Precomputed embeddings obtained from *fastText*[47][48].

These precomputed embeddings can remain frozen during the training phase or be fine-tuned during training in order to adjust their values to the concrete sentences found in the text.

7.1 Embeddings

In this section we will cover in depth what word embeddings are, as they have only been briefly explained and they are used all along the project. We will also detail all the experiments that have been performed with this representation of the data.

We have tested a total of three different embeddings in order to obtain the best possible results out of our implementation. The results obtained by these tests are covered in Section 7.1.2. It should be remarked that one of the embeddings tested was finally discarded due to the appearance of many English words. The embeddings tested are the following:

- Embeddings learned during training.
- Spanish embeddings from fastText frozen or fine-tuned during training.
- Embeddings obtained from *GloVe*[49].

The discarded set of embeddings is the last one. A precomputed set of embeddings, is composed by a list where each element contains a word and its corresponding embedding. In the vocabulary of these embeddings we found words in Spanish as well as in English as it was learned in a multilingual corpus. As the presence of these English words could influence the embedding obtained for each of the Spanish words, making them not fully accurate, we preferred to keep only the embeddings learned during training, and the precomputed ones obtained from fastText with and without fine-tuning. Those are the ones that will be tested later on.

Word embeddings can be learned [50] using algorithms such as Word2Vec[51] from a large corpus, but this topic will not be covered in this project.

7.1.1 Embedding theory

An embedding is a **vector of real values** associated to a word. This is a common representation of words used in NLP, with the purpose of encoding the words in such way that, not only can be identified from other words, but also can be related with similar words.

This **representation** is related with the fact that words have different meaning **depending on the context** in which they are used. Therefore, finding a representation for a word is not always exactly correct. Rather than that, the real objective is to find a **representation**

for the specific meaning of a word in a context. This is a real challenge, and is why the surrounding words play a very important role in identifying the semantically accurate description for that word and its rightful representation.

As a summary, the most important characteristics of the embeddings are the following:

- Embeddings quantify semantic similarities found in large datasets.
- They follow the algebraic model of a vector space model representation, which consists in using vectors to identify words and extract information.
- The context of each word is used to characterize the word's purpose semantically in the context.

In Figure 7.1, an example of the result of this process can be found, where each word is transformed to its corresponding embedding. Each row represents the learned embedding of each word. In this case, just as an example, a five dimensional embedding.

$$\begin{Vmatrix} \text{Window} \\ \text{Person} \\ \text{Car} \\ \text{Bottle} \\ \text{Building} \\ \text{Book} \end{Vmatrix} \Longrightarrow \begin{Vmatrix} 0.2 & 1.7 & 0.0 & 1.1 & 2.1 \\ 5.2 & 1.8 & 2.0 & 1.1 & 3.2 \\ 2.2 & 2.5 & 0.8 & 5.1 & 2.7 \\ 0.7 & 0.5 & 2.3 & 3.2 & 2.9 \\ 0.3 & 2.3 & 4.2 & 0.1 & 1.3 \\ 1.6 & 1.4 & 1.1 & 0.2 & 0.3 \end{Vmatrix}$$

Figure 7.1: Embedding result

This type of representation also has some **limitations** associated. For instance, the inability to represent two meanings of the same word simultaneously. This causes that the set of embeddings will contain only one semantic vector for a word, regardless the number of different meanings in which it is used in the original text. This can be a problem, for example, with polysemic words as they can fit in different contexts but their embedding will not.

7.1.2 Testing of the embeddings

In this section, we will cover the tests performed over the two[1] different embeddings that we are using for this project. This is to keep only the embedding, frozen or trainable, that has a better performance. For this purpose, we will only look at the accuracy and balanced accuracy obtained using always the same model. The concrete architecture of the model used for this testing phase is detailed in Figure 6.1. As previously mentioned, since the models are stochastic, the results can change if the training procedure is repeated. Therefore, the results might change slightly from any result previously presented. The results of testing the embeddings are detailed in Table 7.1.

	Validation set		**Test set**	
	Accuracy	**Balanced Accuracy**	**Accuracy**	**Balanced Accuracy**
Learned during training using the CNN	67.53	58.78	69.95	59.61
Fixed fastText	72.42	64.73	71.71	64.38
Trainable fastText	61.27	61.32	69.77	61.41

Table 7.1: Percentage of accuracy and balanced accuracy obtained by using the different embeddings tested.

As shown in Table 7.1, the best results are obtained using the fastText embeddings without the option of being trainable. This option performs clearly better than the "learned during training" ones according to all the metrics. Moreover, the "Fixed fastText" also performs slightly better than the "Trainable fastText" according to all metrics. Due to its better performance measured in the validation set, we will use the "Fixed fastText" embeddings from now on.

These results can be achieved because the embeddings are learned separately, without the influence of the backpropagation algorithm that is used to optimize the rest of the parameters of the DNN. Also, because they are learned in a larger corpus. Due to these results, from now on, we will use the precomputed embeddings without the possibility of modifying them during training.

[1]The embeddings learned during training and the ones from fastText with and without fine-tuning.

7.2 CNNs implemented

For the concrete implementation of the CNNs, we have used 1D Convolutional layers. More concretely, two different convolutional architectures have been implemented:

1. A **sequential** convolutional neural network.
2. A **non sequential** convolutional neural network.

These networks are implemented based on the **Sequential** Keras API and the **Functional** Keras API respectively. The concrete neural network architectures used are presented in Sections 7.2.1 and 7.2.2 respectively.

7.2.1 Sequential model

The model is built layer by layer, with only one path from the input layer to the output layer, processing the data only on one layer at the same time. Therefore, the output of one layer is the input of only one layer. That will change in the parallel model explained later. The diagram of layers that compose the Sequential CNN model implemented, is presented in Figure 7.2.

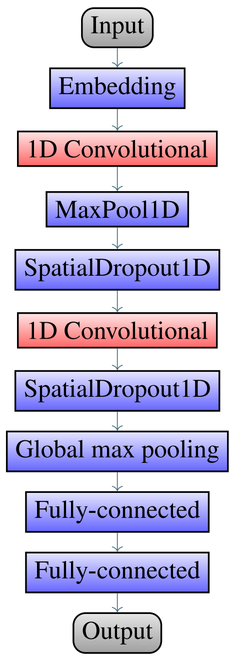


Figure 7.2: Diagram representing the sequential convolutional neural network model built, showing the structure of layers.

The functionality of CNN, and their specific variant 1D CNN, was covered previously in Section 3.2. However, there are certain hyperparameters of the network that have been used with their default values from the keras library, and some that have been changed. This was in order to improve the accuracy of the model. The changed values[2] are the following:

1. Embeddings layer: the embeddings can be:
   - The embeddings learned by the model from the data.
   - The precomputed embeddings loaded from the *fastText* library.
2. First convolutional layer:
   - Filters: 256
   - Window Size: 3
   - Activation function: ReLU
   - L1 regularizer
3. Size of the Max poolings: 2
4. Drop out percent for the neurons in both spatial drop out layers: 30%
5. Second convolutional layer:
   - Filters: 256
   - Window Size: 4
   - Activation function: ReLU
6. Neurons in the first fully connected layer: 256
7. Neurons in the second fully connected layer: 128
8. Activation function of the output layer: SoftMax
9. Optimizer for the backpropagation: Adam
10. Loss function: Sparse categorical cross entropy

These values have been adjusted in order to improve the performance but, in order to obtain the best possible performance out of this architecture, it is necessary to perform the optimization process that will be detailed in Chapter 8.

[2]These are commonly used parameters when using TensorFlow or any other library.

7.2.2 Parallel model

This model corresponds to a "non-sequential" topology, implemented using the Keras **functional** API. As a result of this:

1. A model can have multiple inputs and outputs.
2. The input of a layer can be the combination of the outputs of multiple layers, and its output can be the input of more than one layer.

The main idea behind using this kind of models instead of the sequential ones is that, as more than one convolutional layer can process the input in parallel, these layers can be built using different values of their hyperparameters. This approach allows to extract different information from the sentence, information that will be combined later on. This way, the data can provide more information to the following layers of the CNN and contribute to perform a more accurate prediction.

In order to obtain different information from both convolutional layers, we will use **different window sizes**. In Figure 7.3, the structure of layers that compose this model is detailed.

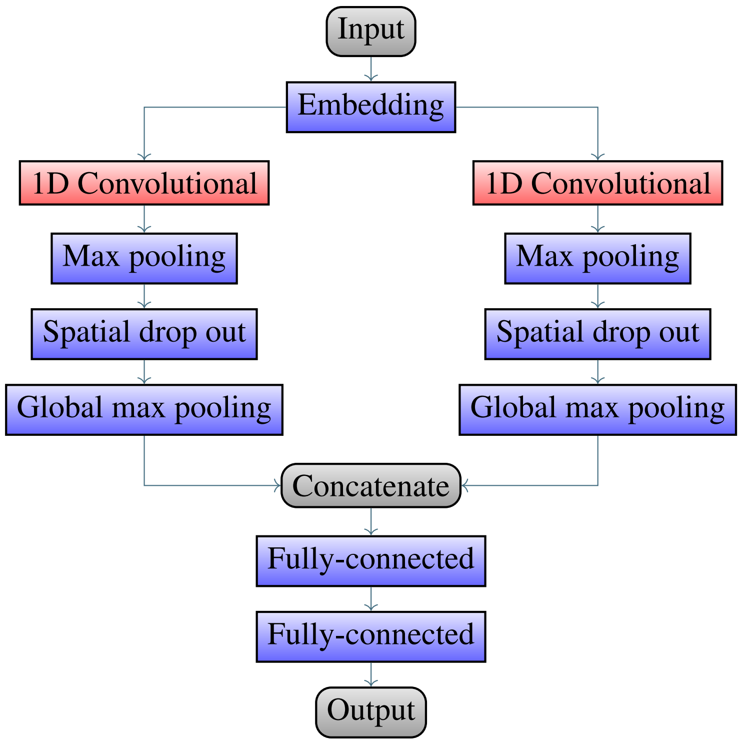


Figure 7.3: Diagram of the parallel CNN model built, showing the structure of layers.

Some hyperparameters of the network are used with the default values from the Keras library. The values of the following hyperparameters have been changed in order to improve the performance of the model:

1. Embedding layer: the concrete embeddings can be:
   - The embeddings learned by the model from the data.
   - The precomputed embeddings loaded from the fastText library.
2. Filters in both convolutional layers: 128.
3. Window Sizes for each Convolutional layer: 3 and 4.
4. Pooling size in both MaxPool layers: 2.
5. Drop out rate in both Spatial DropOut layers: 30%.
6. Nodes in the dense layers in order of appearance: 64 and 32.
7. Activation functions:
   (a) ReLU for the internal layers.
   (b) SoftMax for the output layer.
8. Optimizer for the backpropagation: Adam.
9. Loss function: Sparse categorical cross entropy.

These values have been tested for the sake of performance improvement. But, in order to obtain the best possible performance out of this architecture, it is necessary to perform the optimization process that will be detailed in Chapter 8.

7.3 RNNs Implemented

Up to now, we have only evaluated Convolutional Neural Network but, as a sentence can be seen as a sequence of words, we could use Recurrent Neural Network [52][53] for this problem. As we did with the CNN, we will test sequential as well as parallel architectures. We will also investigate the effect of using different types of recurrent neurons, such as GRU or LSTM.

In all the implementations, the recurrent layers will process the sequences in a bidirectional [28] way. This is because there can be some extra information that is obtained from the sentence when it is processed backwards, that can not be obtained when processing it forward.

7.3.1 Sequential models

The purpose of the following tests will be to measure the performance of GRU and LSTM separately. Depending on their separate results, we will decide to use only one or both to build a more complex DNN. For this purpose, we will build two simple DNN where the only difference is that the first uses GRU recurrent neurons and the second uses LSTM recurrent neurons.

The concrete architecture of layers that uses GRU recurrent units is detailed in Figure 7.4.

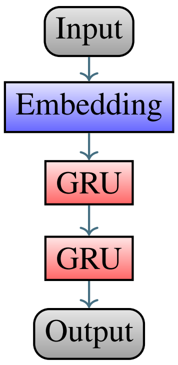


Figure 7.4: RNN architecture based on GRU.

All the hyperparameters of the DNN are set as default values except the following:

1. Embeddings: precomputed embedding from *fastText*[3].
2. Recurrent Units:
    (a) First layer: 128
    (b) Second layer: 32
3. Activation functions:

[3]This selection is based on the results obtained in the previous section 7.1.2, where this embedding obtained the best results among all tested.

   (a) Recurrent layers: tanh.
   (b) Output layer: SoftMax.
4. Optimizer for the backpropagation: Adam
5. Loss function: Sparse categorical cross entropy

The architecture based on the LSTM recurrent units is detailed in Figure 7.5.

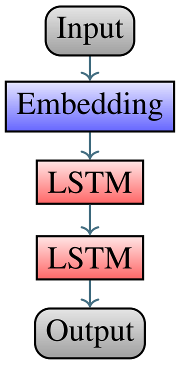


Figure 7.5: RNN architecture based on LSTM.

All the hyperparameters of the DNN are set as default values except the following:

1. Embeddings: precomputed embeddings from *fastText*[4].
2. Recurrent Units:
   (a) First layer: 128
   (b) Second layer: 32
3. Activation functions:
   (a) Recurrent layers: tanh.
   (b) Output layer: SoftMax.
4. Optimizer for the backpropagation: Adam
5. Loss function: Sparse categorical cross entropy

[4]This selection is based on the results obtained in the previous section 7.1.2, where this embedding obtained the best results among all tested.

The performance obtained by both models is presented in Table 7.2:

	Validation set		**Test set**	
	Accuracy	**Balanced Accuracy**	**Accuracy**	**Balanced Accuracy**
GRU	69.65	62.70	69.49	61.33
LSTM	69.83	60.65	69.86	58.02

Table 7.2: Percentage of accuracy and balanced accuracy obtained by using the LSTM and GRU separately.

As shown in Table 7.2, both models provide similar performance. Therefore, we will combine both types of recurrent units in a more complex network to try to create a model that performs better.

7.3.2 Parallel model

The rational behind this implementation is to combine the information extracted by both types of recurrent neurons (GRU and LSTM) in order to improve the performance of the model. For this purpose, we will build the following model, detailed in Figure 7.6.

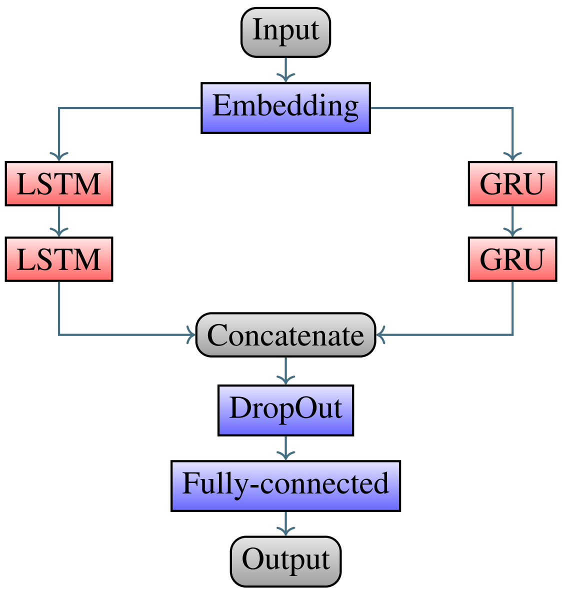


Figure 7.6: RNN architecture based on a combination of LSTM and GRU

It is important to mention that the value of some hyperparameters are different from the ones in the sequential models. They were changed in order to improve the performance of this new model. All the hyperparameters of the DNN are set to default values except the following:

1. Embeddings: precomputed embeddings from fast text[5].
2. Recurrent Units in both types of recurrent layers:
    (a) First layer: 64.
    (b) Second layer: 32.
3. Drop Out rate: 30%.
4. Neurons in the dense layer: 64.
5. Activation functions:
    (a) Recurrent and fully connected layers: tanh.
    (b) Output layer: SoftMax.
6. Optimizer for the backpropagation: Adam.
7. Loss function: Sparse categorical cross entropy.

The results obtained by this architecture are detailed in Table 7.3.

	Validation set		**Test set**	
	Accuracy	**Balanced Accuracy**	**Accuracy**	**Balanced Accuracy**
LSTM + GRU	73.43	65.85	70.69	63.09

Table 7.3: Percentage of accuracy and balanced accuracy obtained by the architecture that combines LSTM and GRU neurons

Looking at these results and comparing them to the results obtained by the sequential architectures, detailed in Table 7.2, we can see that the results have improved according to all measured metrics. The fact that the improvement is not very significant, in some of them, can be because of a non optimal value of the hyperparameters.

[5]Idem [3]

In the following Chapter 8 we will select only some of the most relevant architectures and will take them through the full optimization process to extract the maximum performance out of each neural network.

7.4 Lessons learned

In this section, the main challenge we faced during the implementation, and that might as well be helpful when trying to solve similar problems, will be presented below.

7.4.1 Spanish words in the English embeddings

As previously explained, during the evaluation of the architectures we tested the influence of different precomputed embeddings. Among them, we picked by mistake one set of embeddings that was conceived to be used in problems with English text. This means that this set was trained using text in English, and therefore was supposed to contain none or very few words in Spanish.

The set of sentences that we will try to predict is in Spanish and if we try to use English embedding for this purpose, the predictions should be remarkably incorrect. Just stating the obvious, each embeddings would be formed by a random array of numbers, as non of the needed embeddings should be found in the precomputed set[6].

Surprisingly, the results achieved with these embeddings were similar to those obtained by using the correct embeddings[7] (57% versus 65% accuracy).

After double checking, we discovered that there were many Spanish words in the set of embeddings[8] and, for those words, the embeddings used were accurate enough. Because of its performance, and for the sake of clearness, we discarded this set of embeddings because there were many English words and this could have affected the vectors obtained for the Spanish words.

[6]When the embedding of a word is not found it can be initialized to an array of random values or to an array where all values are 0.

[7]The results were slightly worse than using the Keras generated embeddings.

[8]The application found approximately 100 000 Spanish words.

CHAPTER 8

Hyperparameter optimization

As previously mentioned, in order to expose the full potential of our DNN models, we need to optimize some of their hyperparameters. For this purpose, we will use the following methods:

1. Grid Search
2. Bayesian Optimization

We first perform the Grid Search, since all the information obtained from that process will be used in the bayesian optimization later on. Due to the high computational load that optimizing a DNN requires, it will only be executed with the following architectures, as we consider them the most interesting ones to optimize:

- The parallel CNN architecture. This architecture is detailed in Figure 7.3.
- The RNN that combines LSTM and GRU recurrent neurons. This architecture is detailed in Figure 7.6.

In order to store all the results obtained during these processes, and to ensure its consistency, we will use a MySQL database. For these processes, we will use the training data included in the **high quality balanced dataset**, that we obtained in the first data augmentation technique. The performance of each model will be measured using the **validation set**.

8.1 Range of values for each hyperparameter

To perform the optimization, we first need to place upper and lower bounds for each hyperparameter's value. The range of values for each of the hyperparameters that will be used in both architectures, are the following:

- **Convolutional Neural Network**:
  - Filters in each of the convolutional layers: [16, 256].
  - Window size for each of the convolutional layers: [3, 6].
  - Neurons in the first fully-connected layer: [32, 160].
  - Neurons in the second fully-connected layer: [16, 80].
- **Recurrent Neural Network**:
  - Neurons in the first LSTM and GRU layer: [32, 160].
  - Neurons in the second LSTM and GRU layer: [16, 80].
  - Neurons in the fully-connected layer: [16, 80].

These ranges will be used in both optimization algorithms. From this ranges, the algorithms will not be able to choose any combination, but a subset of them. The combinations that will be allowed, and the restrictions that will be placed for the Grid Search and for the Bayesian Optimization, will be detailed later on.

We have chosen these ranges of values after testing many possible combinations and evaluating the obtained results. After that, these ranges of values provided very convincing results without generating models with too many parameters.

One aspect that should be noted in the CNN is that the total amount of hyperparameters to be optimized are **6**. Two in each convolutional layer and one in each fully-connected layer. Despite this fact, only **5** hyperparameters will be optimized. This is because, in order to reduce the total number of models, we created a restriction so that both convolutional layers are created with the **same number of filters**. Therefore, as both hyperparameters will have the same value, we will use them as only one from now on.

Contrary to that, in the RNN, each recurrent layer will have its own number of neurons without any dependence of the neurons in other recurrent layers.

8.2 Grid Search

This method will be used to define and explore shallowly[1] the hyperparameter space. It is not possible to evaluate all solutions in the hyperparameter space because, given the range of values for each hyperparameter covered in Section 8.1, there are more than 100 000 000 different possible combinations. Moreover, training a DNN model requires an important computational load, making barely impossible at a reasonable cost to test so many combinations.

8.2.1 Method explanation

There are two types of Grid Search that we should pay attention to: the 1-dimensional and the m-dimensional [57].

The 1-dimensional Grid Search

For this case, we will consider a function $f : R^1 \longrightarrow R^1$ that we need to optimize and a set composed by n test points such that $L \leq x_1 \leq \cdots \leq x_n \leq U$. We will evaluate this set of points in f so that $f(x_1), \ldots, f(x_n)$ can provide us with some indication of the behaviour of f in $\mathcal{I} = [L, U]$. We need to be very careful with the number of test points, as the higher we go the better the function will be defined, but the execution time will heavily increase. Specially if f is very costly to evaluate, as it is the case in this project. This n points can be selected manually or in an equi-distance way.

The m-dimensional Grid Search

We will try to optimize a function f in a region of the space $\mathcal{I} = [a_i, a_2] \times [b_1, b_2] \times \cdots \in R^m$. For each parameter we will need to specify the values that will be used to define the grid. Again, they can be selected manually or equi-distant. As the amount of combinations grows exponentially with m, this method is impractical in higher dimensional problems. Therefore, in our implementation, we will limit the amount of hyperparameters to optimize, and will not choose more than 6 possible values for each hyperparameter[2].

In our particular implementation, each point will represent a configuration of the values of the hyperparameters that will be used to create a DNN model. The function f will have

[1]We will not test all the possible combinations of the hyperparameters, only a few selected combinations.

[2]For this problem we will limit it to 6 in order to keep the execution time under control. For a more precise definition of the hyperparameter space, more than 6 values can be chosen.

as input the values of the hyperparameters and, as output, a value that will address the performance of the model created with those values as hyperparameters. As we need to obtain as much information as possible about the hyperparameter space, we will choose values that are scattered by the complete hyperparameter space, and not concentrated around one single value. Because of that, the values that we will select are detailed in Section 8.2.2.

8.2.2 Values tested for each hyperparameter

As previously mentioned, not every value in the ranges presented in Section 8.1 will be tested. Moreover, the selected values should try to cover the complete range of values of each hyperparameter. Because of that, the values of the hyperparameters that we will test in both architectures are the following:

- **Convolutional Neural Network**:
  - Filters in each of the convolutional layers: 16, 64, 128, 160, 224, 256.
  - Window size for each of the convolutional layers: 3, 4, 5.
  - Neurons in the first fully-connected layer: 32, 64, 128, 160.
  - Neurons in the second fully-connected layer: 16, 32, 64, 80.
- **Recurrent Neural Network**:
  - Neurons in the first LSTM and GRU layer: 32, 64, 128, 160.
  - Neurons in the second LSTM and GRU layer: 16, 32, 64, 80.
  - Neurons in the fully-connected layer: 16, 32, 64, 80.

Notice that, in our implementation of grid search, the values assigned to the filters and to the neurons were not uniformly distributed in the ranges of the variables. This was due to the fact that this values were also chosen during the selection of the intervals. We selected the values that, after some manual testing, produced more accurate results[3].

[3]Only a few combinations of them were tested manually.

8.2.3 Results obtained running the CNN

In Table 8.1, the best and worst configurations obtained in the process are detailed. In order to choose the best and worst configurations we used the balanced accuracy metric as it is, from our point of view, the metric that better describes the performance out of both. In case of a tie in this metric, we will use the accuracy to decide which one is better.

Taking into consideration that both models are based on the same architecture, and the only difference between them is the value of some hyperparameters, it is somehow obvious the importance of executing these processes in your NN before rolling out in production. This process tested a total of 864 different hyperparameters combinations.

	Accuracy	Balanced Accuracy
Best Configuration	71.31	70.94
Worst Configuration	74.44	37.96

Table 8.1: Comparison of the accuracy and balanced accuracy, measured in percentage, obtained optimizing the hyperparameters of the CNN architecture.

If we compare both results shown in Table 8.1, we observe that the difference is above 30% in the balanced accuracy, far higher than the one present in the accuracy metric of both models. This is caused because of the distribution of the Empathic dataset, detailed in Table 6.1. In this case, if a model is not able to capture enough information to predict all the classes, just by classifying all inputs as the most frequent class, its results can go beyond 70% in the accuracy metric. Results that look very convincing, but being very lame if we pay attention to the balanced accuracy metric. This is the case for the worst model. Therefore, we can clearly see that the worst combination found is not able to correctly differentiate the three different classes of the dataset, whereas the best combination found is perfectly able to do so.

The concrete values of the hyperparameters that produced the best results so far are: 256 filters, 5 and 4 as window sizes, 160 neurons in the first fully-connected layer and 16 neurons in the second fully-connected layer.

8.2.4 Results obtained running the RNN

Regarding the RNN model, the results obtained are very similar to the results obtained running this method with the CNN. In this case, the figures for accuracy do not worsen in the best case. These results are presented in Table 8.2.

	Accuracy	Balanced Accuracy
Best Configuration	66.97	69.18
Worst Configuration	66.32	41.69

Table 8.2: Comparison of the accuracy and balanced accuracy, measured in percentage, obtained optimizing the hyperparameters of the RNN architecture.

The results exposed in Table 8.2 not only remark again the importance of optimizing at least some hyperparameters of the architectures before using them, but also the importance of using more than one metric in order to measure the performance of a model. From the results presented in Table 8.2 we can extract pretty similar conclusions as with the CNN results.

The best model is formed by 128 and 16 neurons in the LSTM based layers respectively, 160 and 80 in the GRU based layers and 32 neurons in the fully-connected layer.

8.3 Bayesian Optimization

The objective of an optimization method is to minimize (or maximize depending on the situation) the cost function of the task to solve by readjusting the values of the parameters[4]. In this section, an **optimization method** to choose the **best possible hyperparameters** is presented[5]. Once the performance of the model with a set of hyperparameters is computed, the optimization method will make some inference over it and try to approximate the best selection of values for the hyperparameters. For this purpose, the more information it has about the function to optimize, the better it will perform.

One method for the optimization of the model's hyperparameters is the **Bayesian optimization** method [58]. In this method, the Bayes theorem is used for making inference and find the **preferable optimization** (maximum or minimum) of an objective function. It is a particularly suitable strategy when the objective function is costly to evaluate, just like training a DNN.

[4]In this case, the parameters of the function will be the hyperparameters of the model, and the result of the function to optimize will be the results of training a model using those values in the hyperparameters.

[5]Given certain limitations in terms of computational cost, and the time in which we need the results.

8.3.1 Methodology

The Bayesian optimization is considered to be a **black-box** optimization method, that is to say, the only known things are:

- The set of parameters that were entered into the function to be optimized.
- The result that the objective function provides, using as input the provided values of the parameters. This is without having any information about how the result was generated.

Therefore, the method takes **sets of values**[6] and evaluates a function in those values, **obtaining a result** related to the problem. For instance, the function could result in training a model, where the input of the function will be the hyperparameters to configure the model. In the result, the training procedure of the model could give some metric to assess the performance of the selection of the hyperparameters.

This means that the values entered into the objective function and the results can be **organized by pairs**. These pairs will be formed by a set of input values s_i and the value that the objective function gives as result f_i. This is used to build a **knowledge base** (KB) that can be used for future iterations of the algorithm and it defines our **prior knowledge** about the problem. In this case, the likelihood function would be the probability of the knowledge base (KB) conditioned by the function (f). This likelihood function is subjected to change as more sets of values are fed into the function to optimize.

$$P(f|KB) = P(KB|f) * P(f) \tag{8.1}$$

Since there are several possible inputs to the function to evaluate, each time a new entry is added to the knowledge base the posterior probability is updated in the search of a better set of values to optimize the function. This is accomplished by evaluating the value of the **posterior probability** when new entries are added. In consequence, the posterior probability is a surrogate function of the objective function and the posterior probability has information about the updated beliefs taken from the results of the objective function. In a sense, the posterior probability is an estimation of the objective function which has the advantage of being efficiently sampled.

[6]In this case, it will take the set of hyperparameters.

From the results obtained with the posterior probability, the knowledge base is updated and it is time for the next set of values to be sampled. Through this process, new values to be tested in the optimization process are generated. For this, a function called the **acquisition function** is used. This function will look for new values in the search space (e.g. hyperparameter space) by optimizing the conditional probability of the areas of the **search space** to create new samples, so that it can choose the areas that are more likely to improve the previous results.

Every time a new sample of values is created by the acquisition function and fed to the objective function, a new entry is added in the knowledge base with the results. This improves the ability to search for the **best sample's values** in the search space. This process is repeated until a given stop condition is satisfied[7].

8.3.2 Configuration and Hyperparameters to optimize

In order to execute this method, we have used **Expected Improvement** as acquisition function and **RBF** as the kernel of the Gaussian Process used as model. Moreover, this process has been executed 4 times for each architecture in order to obtain the results that will be presented later on.

The hyperparameters that will be optimized and their range of values used in this method are the same as the ones detailed in Section 8.1. During this process, the algorithm will be allowed to test each possible combination of the values in those ranges with only **one restriction**. For the amounts of convolutional filters, recurrent units and neurons in the fully-connected layers, it will only be allowed to choose values that are **multiple of 8** (e.g. it can test with 16, 24, 32... and not 16, 17, 18...). The rational behind this restriction is the following:

- Adding one neuron or filter to the network will not change the results significantly. Adding 8, the likelihood is higher that it produces different results.
- This helps to reduce the total amount of combinations that the algorithm can test.
- Reduce the chance of falling in a local minimum of the function to optimize, as the values that can be tested with this approach are more likely to produce different results.

[7]This can be the maximum time for the search or space for the knowledge base among others.

For our concrete implementation we will use all the results obtained in the Grid Search as the initial knowledge base. This knowledge base can guide[8] the inference process in the Bayesian Optimization as a starting point instead of starting from scratch.

This optimization method has been implemented using the GPyOpt [59] library.

8.3.3 Results obtained optimizing the CNN

After finishing the execution of this procedure, a total of 1264 combinations have been tested[9]. These represent only 1.6% of the total amount of models that can be tested as there are 75 888 possible combinations, given the ranges of values chosen for the hyperparameters that we tried to optimize and the restrictions imposed. The best configuration obtained by this method is compared to the best configuration obtained in the Grid Search in Table 8.3.

	Accuracy	Balanced Accuracy	Number of parameters
Bayesian Optimization	72.04	71.14	316 723
Grid Search	71.31	70.94	776 419

Table 8.3: Comparison of the best configuration obtained using Grid Search and the best using Bayesian Optimization for the CNN. Accuracy and balanced accuracy measured in percentage.

The values of the hyperparameters that form the best model found using the Bayesian Optimization are: 128 filters, window sizes of 4 and 3 and 144 and 72 neurons in the fully-connected layers respectively.

As shown in Table 8.3, the results obtained using the best configuration from the Bayesian optimization are slightly better in both metrics, the accuracy and balanced accuracy. Providing more consistent results across all classes. Moreover, this new configuration also has less and smaller filters, resulting in a model with less than half the parameters of the previous one. Overall, this new configuration is less intense in terms of computing resources required. These results can be improved by allowing more time for the algorithm to test more combinations of the hyperparameters, as only 1.6% of them have been evaluated. In Figure 8.1, we show the distributions of the tests performed by each method during the optimization of the CNN architecture.

[8] In the case of this projects, the method will have some information about the hyperparameter space instead of starting from scratch.

[9] It should be taken into consideration that 864 come from the Grid Search. Therefore only 400 new combinations were tested during the Bayesian optimization.

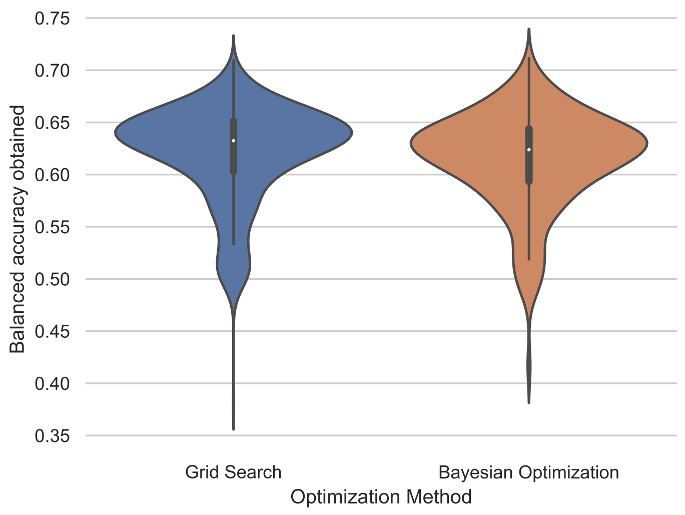


Figure 8.1: Distribution of tests performed for the CNN

In Figure 8.1 we can see how, even though the distributions are similar, the distribution of the results obtained by the Grid Search is slightly more concentrated to the medium range of the values than the Bayesian Optimization one, located a bit more to the high values. This is because the optimization method will always try to find the best possible combination, exploring the areas that are more prone to produce successful results, whereas the Grid Search will simply try a set of predefined combinations. It is to be expected that this results are affected by the stochastic nature of the DNN models.

Therefore, the results could be improved even more by allowing more computing time for the Bayesian Optimization to evaluate more configurations of the hyperparameters.

8.3.4 Results obtained optimizing the RNN

If we compare the results produced by the best configuration previously obtained using the Grid Search with the best configuration obtained using the Bayesian Optimization, we can see the following in Table 8.4:

	Accuracy	Balanced Accuracy	Number of parameters
Bayesian Optimization	72.41	69.25	1 206 387
Grid Search	66.97	69.18	1 116 995

Table 8.4: Comparison of the best configuration obtained using Grid Search and the best using Bayesian Optimization for the RNN. Accuracy and balanced accuracy measured in percentage.

The results are roughly the same when looking at the balanced accuracy, and improved by 5.44% when looking at the accuracy metric. This provides a model whose performance is better. Moreover, the number of parameters of this new model is slightly higher than the previous best. Something that we should remark, is that the number of parameters of the new best RNN is more than 3 times larger than the best CNN configuration found.

This new best RNN is formed by: 128 and 40 neurons in the LSTM layers, 160 and 80 neurons in the GRU layers and 72 neurons in the fully-connected layer.

Once again, these results could be improved by allowing more computing time for the optimization algorithm to run. In Figure 8.2, we will show the distributions of tests performed by each optimization method during the optimization of the RNN.

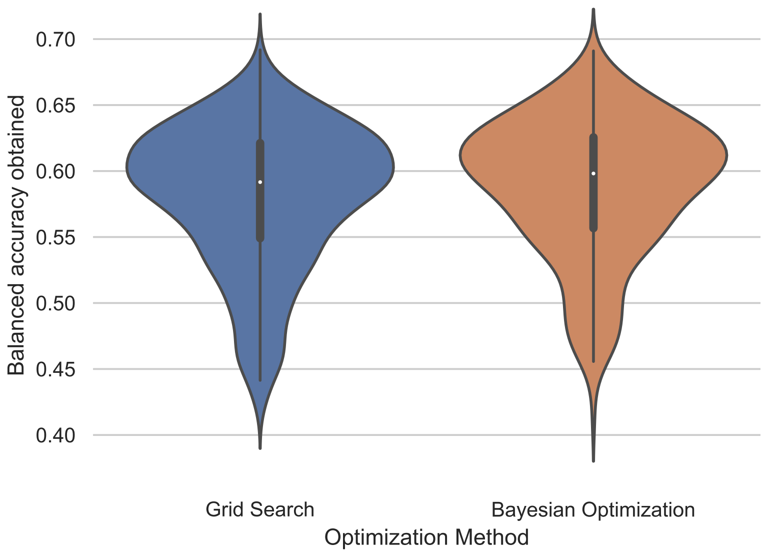


Figure 8.2: Distribution of tests performed for the RNN

Figure 8.2 shows very similar results as the CNN optimization did. Both distributions are very similar, but the distribution of the tests performed by the Bayesian optimization is a bit more concentrated in the high values of the table than the Grid Search ones.

8.4 Comparison of the architectures

In Table 8.5 we compare the best model obtained optimizing the CNN and the best model obtained optimizing the RNN.

	Accuracy	Balanced Accuracy	Number of parameters
CNN	72.04	71.14	316 723
RNN	72.41	69.25	1 206 387

Table 8.5: Comparison of the best configuration obtained optimizing the RNN and the CNN. Accuracy and balanced accuracy measured in percentage.

As shown in table 8.5, the best model obtained optimizing the CNN performs better according to the balanced accuracy metric, and also is formed by less than one third of the parameters that form the RNN. Therefore, we can conclude that the best performing architecture for this problem, out of both, is the CNN.

CHAPTER 9

Conclusions

In this chapter we summarize the conclusions that we can extract from the project. The conclusions will be organized according to the three main components of the project.

9.1 Data augmentation

It is not necessary to go for the most complicated and complex solutions for every problem. In our case it was better to start with the simplest approach and, from there, increase the complexity if needed and only if that increase improves the results.

Something to pay close attention is the class distribution of the dataset as it can heavily affect the results produced by our model. As it has been observed, balancing the class distribution of the dataset improved the performance in a similar ratio to what the optimization process did. Because of that, it is mandatory to study the training data distribution in close comparison to the obtained results.

Normally, when dealing with data augmentation techniques there is an appetite to extend the dataset as much as possible, with the rational that it will generate better results. As shown in the results, it is not necessarily the case. A right balance between dataset size and quality delivers better results than putting all energy on the size.

This part of the project could be expanded by implementing other methods such as generative models, as they can improve the performance in this kind of problems. The drawback of this technique is that it requires extensive knowledge in generative models and it is difficult to obtain the optimal results.

9.2 Implementations

On top of performing the tests with the DNN that is going to be used, it is also important to test different representations of the data. The main reason for this recommendation is that in certain cases, it is not obvious which data representation will provide the most accurate results. Moreover, a set of embeddings learned in a larger, and maybe even higher quality corpus, can provide better results than the ones learned using the corpus of the project. At the end, it can heavily affect the performance of the model.

Sometimes, the best solution for a problem comes with the use of a non sequential model. With the same amount of computational requirements, it might be able to obtain more information from the original data than a sequential architecture could, leading the model to obtain better results. The problem associated to non sequential models is that they are more difficult to design and it is required to fully understand the architecture.

Combining LSTM and GRU recurrent neurons can really boost performance. This idea came straight from us during the design phase of the architectures, even though this approach might have been used before. We did not find previous report on the combination of these two types of neurons.

9.3 Optimization

We obtained different results running the Grid Search and the Bayesian Optimization in both architectures. This should motivate the user to try more than one optimization technique if possible, as it can improve our results even further. We should also take into consideration the computational cost of these algorithms.

When it comes to optimize the network, we normally take the paradigm of “The more complex in terms of parameters, the higher the quality will be”. It is not always the case. Testing different approaches makes sense as it is possible to find a new model that performs better with a smaller number of parameters.

Acronyms

Bibliography

[1] M. I. Torres, J. M. Olaso, C. Montenegro, R. Santana, A. Vázquez, R. Justo, J. A. Lozano, S. Schlögl, G. Chollet, N. Dugan, M. Irvine, N. Glackin, C. Pickard, A. Esposito, G. Cordasco, A. Troncone, D. Petrovska-Delacretaz, A. Mtibaa, M. A. Hmani, M. S. Korsnes, L. J. Martinussen, S. Escalera, C. P. Cantariño, O. Deroo, O. Gordeeva, J. Tenorio-Laranga, E. Gonzalez-Fraile, B. Fernandez-Ruanova, and A. Gonzalez-Pinto, "The EMPATHIC project: Mid-term achievements," in *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '19, (New York, NY, USA), p. 629–638, Association for Computing Machinery, 2019.

[2] C. Montenegro, A. López Zorrilla, J. Mikel Olaso, R. Santana, R. Justo, J. A. Lozano, and M. I. Torres, "A dialogue-act taxonomy for a virtual coach designed to improve the life of elderly," *Multimodal Technologies and Interaction*, vol. 3, no. 3, p. 52, 2019.

[3] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[4] A. Jacovi, O. S. Shalom, and Y. Goldberg, "Understanding convolutional neural networks for text classification," *arXiv preprint arXiv:1809.08037*, 2018.

[5] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, pp. 649–657, 2015.

[6] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008.

[7] B. Liu, "Sentiment analysis and opinion mining," *Synthesis lectures on human language technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[8] C. Montenegro, R. Santana, and J. A. Lozano, "Data generation approaches for topic classification in multilingual spoken dialog system," in *12th Conference on PErvasive Technologies Related to Assistive Environments Conference (PETRA-19)*, ACM, 2019.

[9] P. Liang and N. Bose, "Neural network fundamentals with graphs, algorithms and applications," *Mac Graw-Hill*, 1996.

[10] M. A. Arbib, ed., *The handbook of brain theory and neural networks*. The MIT press, 2003.

[11] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, p. 1237, Barcelona, Spain, 2011.

[12] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton, "Regularizing neural networks by penalizing confident output distributions," *arXiv preprint arXiv:1701.06548*, 2017.

[13] J. Larsen and L. K. Hansen, "Generalization performance of regularized neural network models," in *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pp. 42–51, IEEE, 1994.

[14] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini–batch gradient descent,"

[15] A. Baldominos, Y. Saez, and P. Isasi, "Evolutionary convolutional neural networks: An application to handwriting recognition," *Neurocomputing*, vol. 283, pp. 38–52, 2018.

[16] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.

[17] M. Hughes, I. Li, S. Kotoulas, and T. Suzumura, "Medical text classification using convolutional neural networks.," *Studies in Health Technology and Informatics*, vol. 235, pp. 246–250, 2017.

[18] H. Scheidl, "Build a handwritten text recognition system using tensorflow," Jan 2019. Available at https://towardsdatascience.com/build-

a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5.

[19] H. Schmid, "Part-of-speech tagging with neural networks," in *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pp. 172–176, Association for Computational Linguistics, 1994.

[20] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith, "Part-of-speech tagging for twitter: Annotation, features, and experiments," tech. rep., Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, 2010.

[21] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," *arXiv preprint arXiv:1603.01360*, 2016.

[22] A. Ritter, S. Clark, O. Etzioni, *et al.*, "Named entity recognition in tweets: an experimental study," in *Proceedings of the conference on empirical methods in natural language processing*, pp. 1524–1534, Association for Computational Linguistics, 2011.

[23] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining.," in *LREc*, vol. 10, pp. 1320–1326, 2010.

[24] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. J. Passonneau, "Sentiment analysis of twitter data," in *Proceedings of the Workshop on Language in Social Media (LSM 2011)*, pp. 30–38, 2011.

[25] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse Processes*, vol. 25, pp. 259–284.

[26] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman, "Using latent semantic analysis to improve access to textual information," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '88, (New York, NY, USA), p. 281–285, Association for Computing Machinery, 1988.

[27] O. Irsoy and C. Cardie, "Opinion mining with deep recurrent neural networks," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 720–728, 2014.

[28] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions on*, vol. 45, pp. 2673 – 2681, 12 1997.

[29] X. Wang, W. Jiang, and Z. Luo, "Combination of convolutional and recurrent neural network for sentiment analysis of short texts," in *Proceedings of COLING 2016, the 26th international conference on computational linguistics: Technical papers*, pp. 2428–2437, 2016.

[30] C. Montenegro, R. Santana, and J. Lozano, "Data generation approaches for topic classification in multilingual spoken dialog systems," in *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, pp. 211–217, ACM, 2019.

[31] J. Wei and K. Zou, "EDA: Easy data augmentation techniques for boosting performance on text classification tasks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, (Hong Kong, China), pp. 6383–6389, Association for Computational Linguistics, Nov. 2019.

[32] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, and F.-Y. Wang, "Generative adversarial networks: introduction and outlook," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 4, pp. 588–598, 2017.

[33] K. Wang and X. Wan, "Sentigan: Generating sentimental texts via mixture adversarial networks.," in *IJCAI*, pp. 4446–4452, 2018.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[35] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 1480–1489, 2016.

[36] K. Ahmed, N. S. Keskar, and R. Socher, "Weighted transformer network for machine translation," *arXiv preprint arXiv:1711.02132*, 2017.

[37] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does BERT look at? an analysis of BERT's attention," in *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, (Florence, Italy), pp. 276–286, Association for Computational Linguistics, Aug. 2019.

[38] I. Tenney, D. Das, and E. Pavlick, "BERT rediscovers the classical NLP pipeline," *arXiv preprint arXiv:1905.05950*, 2019.

[39] H. Xu, B. Liu, L. Shu, and P. S. Yu, "BERT post-training for review reading comprehension and aspect-based sentiment analysis," *arXiv preprint arXiv:1904.02232*, 2019.

[40] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune BERT for text classification?," *arXiv preprint arXiv:1905.05583*, 2019.

[41] A. Jacovi, O. S. Shalom, and Y. Goldberg, "Understanding convolutional neural networks for text classification," *arXiv preprint arXiv:1809.08037*, 2018.

[42] M. Abadi, M. Isard, and D. G. Murray, "A computational model for tensorflow: an introduction," in *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 1–7, 2017.

[43] J. Brownlee, *Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras*. Machine Learning Mastery, 2016.

[44] C. Antona Cortés, "Herramientas modernas en redes neuronales: la librería keras," B.S. thesis, 2017.

[45] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[46] J. Kamps, M. Marx, R. J. Mokken, M. De Rijke, *et al.*, "Using WordNet to measure semantic orientations of adjectives.," in *LREC*, vol. 4, pp. 1115–1118, Citeseer, 2004.

[47] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[48] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[49] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.

[50] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, (Miyazaki, Japan), European Language Resources Association (ELRA), May 2018.

[51] X. Rong, "word2vec parameter learning explained," *arXiv preprint arXiv:1411.2738*, 2014.

[52] M. Boden, "A guide to recurrent neural networks and backpropagation," *the Dallas project*, 2002.

[53] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[54] L. Chen and C. M. Lee, "Convolutional neural network for humor recognition," *arXiv preprint arXiv:1702.02584*, 2017.

[55] C. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, (Dublin, Ireland), pp. 69–78, Dublin City University and Association for Computational Linguistics, Aug. 2014.

[56] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, pp. 649–657, 2015.

[57] J. Kim, *Iterated grid search algorithm on unimodal criteria*. PhD thesis, Virginia Tech, 1997.

[58] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 2951–2959, Curran Associates, Inc., 2012.

[59] T. G. authors, "GPyOpt: A bayesian optimization framework in python." `http://github.com/SheffieldML/GPyOpt`, 2016.

[60] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010.