A Deep Learning Approach for Italian Stereotype Detection in Social Media

Alessandro Cudazzo
alessandro@cudazzo.com

Giulia Volpi
giuliavolpi25.93@gmail.com

Department of Computer Science, University of Pisa, Pisa, Italy

Human Language Technologies Project
Academic Year: 2019/2020
July 5, 2021

Abstract

The Hate Speech Detection (HaSpeeDe 2) task presented at Evalita 2020 was composed of the main task (hate speech detection) and two Pilot tasks (stereotype and nominal utterance detection). This paper aims to investigate different models for solving the stereotype detection task. Our study includes different types of neural networks such as convolutional neural networks (CNNs), recurrent neural networks model (BiLSTM), and BiLSTM with a soft-attention module. We also evaluated a BERT model by using an Italian pre-trained BERT and then fine-tuned the entire model for our classification task. In our experiments, it emerged that the choice of model and the combination of features extracted from the deep models was important. Moreover, with Bert, we noticed how pre-trained models on large datasets can give a significant improvement when applied to other tasks.

***K*eywords** Stereotype Detection; Abusive Language; Sentiment Analysis; NLP; Italian Language

1. Introduction

Social media such as Twitter, Facebook, and Instagram are among the most popular platforms used by people to communicate and exchange information. These are generally unmonitored and content control over daily generated data stream is relatively low, so the nature of these platforms sometimes causes users to misuse them by creating and sharing aggressive and hateful content. Generally, hateful and stereotyped content is commonly defined as any communication that includes disparagement of an individual or a group based on some characteristic such as race, national origin, sex, religion, nationality, or other characteristics [1]. Moreover, the presence of these contents in social media can play a critical role in our society, and just like mass media, they may influence individual and collective identities. This can change the approach people have towards certain groups, topics, opinions, and information of all kinds, thus distorting the individual's approach towards them. This kind of influence and behavior can be harmful, there is evidence that the promotion of stereotypes, such as those towards ethnic groups, not only causes hostile behavior towards them but also lowers ethnic minority individuals' self-esteem [2].

Given the steady growth of social media content, the amount of online hate and stereotype speech is also on the rise, so the development of automated analysis methods has been necessary.

This paper aims to investigate different models for solving the stereotype detection task. The dataset provided at the HaSpeeDe 2 event organized within Evalita 2020 has been considered, since it has already been experimented with the most recent state-of-the-art literature [3], allowing us to compare our results appropriately. Our study includes different types of neural networks such as convolutional neural networks (CNNs), recurrent neural networks model (BiLSTM), and BiLSTM with a soft-attention module. We also evaluated a BERT model by using an Italian pre-trained BERT made available by the MDZ Digital Library and then fine-tuned the entire model for our classification task.

The remaining text is structured as follows. In Section 2, the dataset is stated and then the pre-processing task is illustrated. In Section 3 all the methods used in our analysis are formalized. Section 4 presents all the results obtained in our experiments, while Section 5 is devoted to conclusions.

2. Datasets and Data Preprocessing

The HaSpeeDe 2020 training set includes 6839 annotated tweets that target minority groups such as immigrants, Muslims, and Roma communities. Regarding the stereotypes detection task, 3042 are marked as stereotypes and the rest as non-stereotypes. The raw dataset was very noisy due to people's misspellings and their original way of writing text. In addition, the tweets have some special features such as emoji, emoticons, hashtags, and user mentions added to the more common ones such as URLs, email addresses, phone numbers, percentages, amounts of money, time, date, and generic numbers.

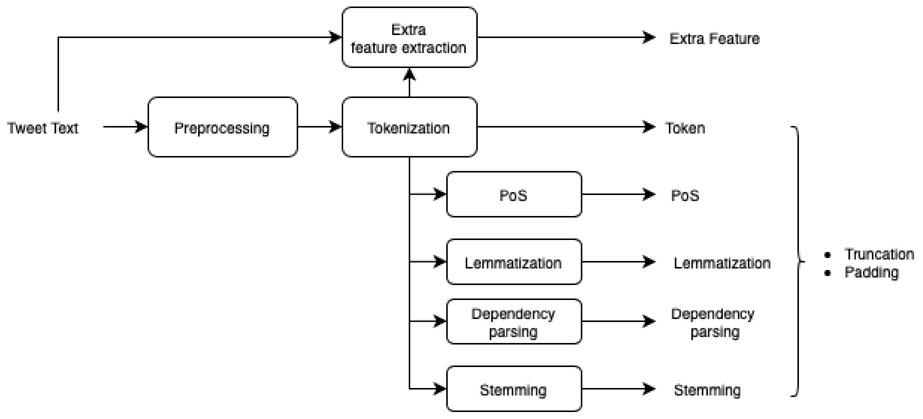


Figure 2.1: Pre-procressing pipeline

In order to obtain a more proper a dataset for machine learning models, some pre-processing techniques have been performed. The steps of the pre-processing phase are described below and our pipeline is summarized in Fig. 2.1.

2.1 Preprocessing

Our pre-procressing pipeline contains the following steps:

URLs and user mentions
In the raw dataset, the URLs and user mentions were already been normalized respectively into `URL` and `@user` keywords. So we applied the same process for all tweets and removed the URL and user tag.

Emoticons

They are sequences of textual characters intended to represent, in a stylized manner, human facial expressions as an emotion. In this work, the emoticons have been translated into a word that expresses the same feeling. To do this, the ekphrasis tool [1] [4] has been used, which enables the individuation of emoticons and replace them with normalized form. Then the normalized form has been converted into an Italian word. Some examples are reported in Tab. 2.1.

Emoticon	Normalized form	Italian
:* :x :-*	<kiss>	bacio
:-) :3 :-))))	<happy>	felice
:-D xD =D	<laugh>	risata
:-(:-c :'-(	<sad>	triste
>.< :S :-/	<annoyed>	annoiato

Table 2.1: Transformation of emoticons.

Emojis

These are pictographic symbols that represent different things, such as faces, weather, vehicles, feelings, or activities. Like emoticons, emojis have also been transformed to retain their meaning and remain consistent with the other words in the sentence. This process has been performed by combining the emoji tool[2] with the international table of official Italian emojis list[3]. Therefore, the emoji library has been extended and the commit was successfully accepted by adding Italian support to this library. Examples are reported in Tab. 2.2.

Emoji unicode	Normalized form	Italian
U+1F600	grinning face	faccina con un gran sorriso
U+1F976	cold face	faccina congelata
U+1F620	angry face	faccina arrabbiata
U+1F922	nauseated face	faccina nauseata
U+1F63B	smiling cat with heart-eyes	gatto innamorato

Table 2.2: Transformation of emojis.

Hashtags

Phrases preceded by the hash symbol # are called hashtags. These phrases can consist of letters, digits, and underscores; spaces and special characters are not allowed. In the pre-processing phase, using the python wordninja library[4], each hashtag has been split into its constituent words. In order to do that, this tool allows us to use custom dictionaries for the tokenization of the words, and in this case, we have used the Italian dictionary available online[5] that counts more or less 900.000 words. Moreover, a custom version of this dictionary has been developed to make it compatible with the library mentioned above. The resulting phrase is placed between two special tags, < and >. Hereafter an example of this procedure:

`#iostoconsalvini` $\longrightarrow$ `< io sto con salvini >`

Censored bad-words

Bad words can be often Censored with some of their middle letters replaced by special characters (symbols and punctuation as `@#%&$^.*`) and letter `x`. In this work, the bad words have been replaced by their

1. `https://github.com/cbaziotis/ekphrasis`
2. `https://github.com/carpedm20/emoji/`
3. `https://emojiterra.com/it/punti-di-codice/`
4. `https://github.com/keredson/wordninja`
5. `https://github.com/napolux/paroleitaliane`

uncensored version. So, a simple list of common bad words, written by us, has been used. By using regular expressions, we scanned the tweets and checked if there were any of these words, and replaced them.

Acronyms
These are names formed from sequences of one or more letters of individual words or certain words in a sentence or designation, read as if they were a single word. During the pre-processing phase, the acronyms have been replaced by the words that constitute them, using a dictionary (acronym-words) created by ourselves.

Abbreviations
Shortened form of a word or phrase, by any method. As for the acronyms, also the abbreviations have been replaced using a dictionary (abbreviation-word) created by ourselves.

Writing errors
In this work, the nearby equals vowels, and the nearby equal consonants, if they were more than two, have been removed (examples: `caaaane` $\longrightarrow$ `cane`, `gallllina` $\longrightarrow$ `gallina`).

Normalizing numbers
The same number normalization process showed in [5] has been used. More precisely: integers numbers between 0 and 2100 were kept as original; each integer number greater than 2100 is mapped in a string which represents the number of digits needed to store the number (ex: $10000 \longrightarrow$ `DIGLEN_5`); each digit in a string that is not convertible to a number must be converted with the following char: `@Dg`s. Example:

$$10,234 \longrightarrow \texttt{@Dg@Dg,@Dg@Dg@Dg}$$

Normalizing words
For the normalization of the words, the same work of Cimino, De Mattei, and Dell'Orletta [5] used for number normalization has been followed. In this case a string starting with lower case character must be lowercased (e.g.: `aNtoNio` $\longrightarrow$ `antonio`, `cane` $\longrightarrow$ `cane`), and a string starting with an upcased character must be capitalized (e.g.: `CANE` $\longrightarrow$ `Cane`, `Antonio` $\longrightarrow$ `Antonio`). Before doing this normalization, a procedure adds a space after each lower letter followed by an upper one. This was done because we realized that many users wrote everything attached, highlighting the end of one word and the beginning of another by simply switching from a lowercase letter to a capital letter. Hereafter an example:

`StupranoUccidonoEmuoionMartiri` $\longrightarrow$ `Stuprano Uccidono Emuoion Martiri`

Punctuation, capital letters, and stop words
Finally, this approach leaves untouched punctuation, stop words, and capital letters, to avoid the loss of information that could be useful.

2.2 Tokenization

Tokenization is a way of separating a piece of text into smaller units called tokens. In order to prepare the tweets for the models, this kind of process has been done with *spaCy* coupled with *UDPipe* library[6]. Then, our preprocessing procedure converts a message such as

@user @user infatti finché ci hanno guadagnato con i campi #rom tutto era ok con #Alemanno #Ipocriti"
to
['infatti', 'finché', 'ci', 'hanno', 'guadagnato', 'con', 'i', 'campi', '<', 'rom', '>', 'tutto', 'era', 'ok', 'con', '<', 'Alemanno', '>', '<', 'Ipocriti', '>'].

6. `https://github.com/TakeLab/spacy-udpipe`

2.3 Stemming, Lemmatization, PoS and Dependency parsing

Stemming:
This process chops off the ends of words and reduces them to their root words. The procedure makes use of the *Snowball stemmer* of the NLTK library[7] and converts a message such as:

È terrorismo anche questo , per mettere in uno stato di soggezione le persone e renderle innocue"
to
"è terror anche quest , per mett in uno stat di soggezion le person e rend innocu".

Instead, the following transformation were obtained with *spaCy* coupled with *UDPipe* library[8].

Lemmatization parsing:
The *Lemmatization* aims to transform a word into its base form in a less crude way than stemming. In fact, it usually does things properly with the use of a vocabulary and a morphological analysis of words, returning the base or dictionary form of a word, which is known as the lemma. E.g.

"È terrorismo anche questo , per mettere in uno stato di soggezione le persone e renderle innocue."
to
"essere terrorismo anche questo , per mettere in uno stato di soggezione il persona e rendere lo innocuo".

PoS Tagging:
Moreover, Words can be grouped into classes called *Part of Speech (PoS)* or morphological classes. These classes (noun, adjective, preposition, adverb, conjunction, etc.) provide fundamental information to determine the role of the word itself and its neighbors in the sentence.

Dependency parsing:
Finally, *Dependency parsing* is the task of extracting a dependency parse of a sentence that represents its grammatical structure and defines the relationships between "head" words and words, which modify those heads. In Fig 2.2, is shown how a sentence and its dependencies look like.

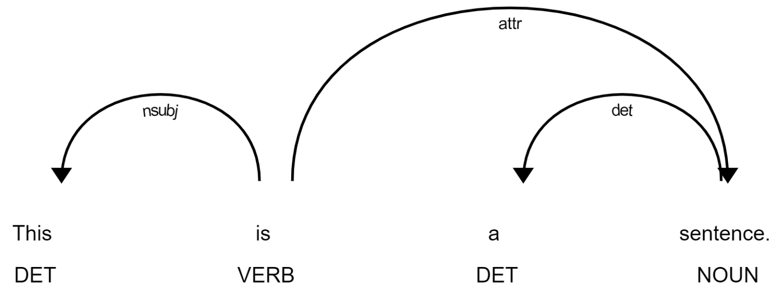


Figure 2.2: Visualization of the dependecy parse.

2.4 Feature Extraction

To preserve the information loss caused by the pre-processing phase we decided to extract some features before performing this phase. All the extracted features before the pre-pocessing are shown in the Tab 2.3.

7. `https://www.nltk.org/_modules/nltk/stem/snowball.html`
8. `https://github.com/TakeLab/spacy-udpipe`

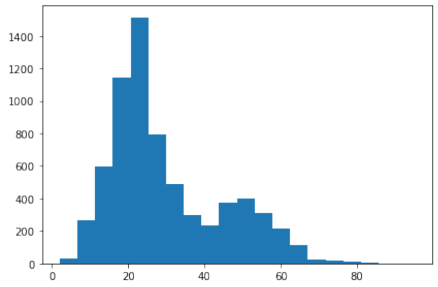
Feature name	**Description**
Text length	It was found that on average the messages are 144 characters long, ranging from a minimum of 8 to a maximum of 730.
#hashtags	Number of hashtag contained in the sentence.
%CAPS-LOCK words	People used to write with the CAPS-LOCK to simulate the spoken aloud. CAPS-LOCK words has been calculated as $$\%CAPS-LOCK = \frac{\#CAPS-LOCK\ words}{\#words} \times 100.$$
#Question and exclamation marks	Number of question/exclamation marks contained in the sentence.
%bad words	The percentage of bad words inside the tweet has been extracted, as $$\%Bad\ words = \frac{\#Bad\ words}{\#words} \times 100.$$

Table 2.3: Extra features extracted before the pre-processing utility.

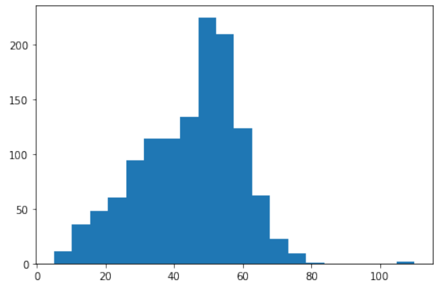
Instead, after pre-processing, by using the lemmatization transformation we have computed the positive/negative/neutral percentages sentence polarity:

Sentence polarity:
We used the *OpeNER Sentiment Lexicon for Italian* [9] [6] to extract the word polarity. The Italian Sentiment Lexicon provided us 24.293 lexical entries annotated for positive/negative/neutral polarity - a word not present in the lexicon has been set to neutral by default. Then we have computed the number of positive, negative, and neutral words in the sentence and divided by the total length, thus obtaining the percentage of positivity, negativity, and neutrality of the sentence.

2.5 Truncation and Padding



(a) Training set length distribution.



(b) Test set length distribution.

Figure 2.3: Length distribution of token lists for each sentence for training (a) and test set (b).

After the preprocessing phase, we decided to analyze the training distribution of the tokens list length obtained from each sentence. As shown in Fig. 2.3 it turned out that most of the sentences were at most 65 tokens long. Then, we decided to truncate the tokens and apply the same to all the remaining transformations (stemming, lemmatization, PoS and dependency parsing). Where the sentence was shorter than 65, padding was applied to align the sentences all at the same length.

9. `https://dspace-clarin-it.ilc.cnr.it/repository/xmlui/handle/20.500.11752/ILC-73`

3. Methods

The NLP classification problem is mainly divided into two steps. The first one is the word embeddings, which convert the text in a numerical form since machine learning algorithms don't work with plain text. The second one is the machine learning model itself. The model follows a two-step procedure. In the first step, some features are extracted with a deep learning technique. In the second step, those features are fed to a classifier (like a softmax) to make a prediction. In the following, we will analyze the two main components used in our experiments.

3.1 Word Embeddings with Word2Vec

Briefly, *word embeddings* are distributed vector representations of a particular word. We decided to use the pre-trained *Italian Twitter embeddings*[10] result of the work of Cimino, De Mattei, and Dell'Orletta [5]. These embeddings are 128-sized and were generated by *word2vec* [7], which can capture contextual word-to-word relationships in a multidimensional space, with the *CBOW* model that learns the embedding by predicting the current word based on its context.

3.2 Convolutional Neural Networks Architecture (CNN)

At the beginning of our study, we decided to start with a Convolutional Neural Network (CNN). This type of deep learning model has achieved remarkable results in the field of computer vision [8] and since it works well where detecting local and position-invariant patterns is important they become one of the most popular model architectures for text classification [9]. Over the years several CNN models have been presented to solve the text classification task, in particular our CNN architecture was inspired by the Kim's model [10] that is well illustrated in fig 3.1.

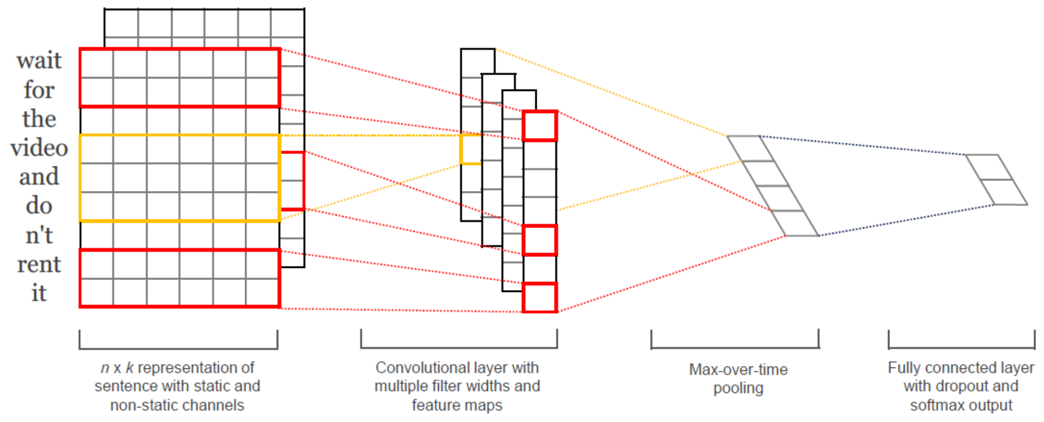

wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 3.1: Model architecture with two channels for an example sentence.

As we mentioned before, our max sentence length is 65 and each word is mapped into 128-dimensional real valued vectors by the embedding layer, this will be used as input layer (embedding layer) with a dimension of 65×128. Then, our model architecture consists in 3 parallel convolutions layers with different kernel size allowing to learn multiple filter with different width that will allow to extract feature maps by considering different n-gram over the sentence. Each feature map produced will be fed to the max-over-time pooling layer in order to reduces the output's dimensionality by preserving only the most significant n-gram feature in the entire sentence - *one* feature from *one* feature map. These features will be concatenate and passed to a

10. `http://www.italianlp.it/download-italian-twitter-embeddings/`

fully connected softmax layer whose output is the probability distribution over labels. Finally, to overcome the overfitting problem we employ the dropout [11] on the feedforward layer that prevents co-adaptation of hidden units by randomly dropping out some number of layer outputs during foward- backpropagation phase. From now on we will refer to this architecture as K-CNN.

3.3 Double Depth-wise CNN Architecture

This new architecture, named D-KCNN, comes as a direct evolution of the previous one presented in Fig. 3.1. At this point, we wanted to include more information in the neural network in order to improve the classification capability in stereotype detection. So we decided to include the PoS Tagging and all the extra features, extracted in the preprocessing phase, and couple them with the preprocessed tweet. To accomplish this we used two parallel K-CNN, one for the tweet and the order one for the PoS Tagging, then the output of both CNN has been concatenated with the extra feature and fed into the classification softmax layer as shown in Fig. 3.2.

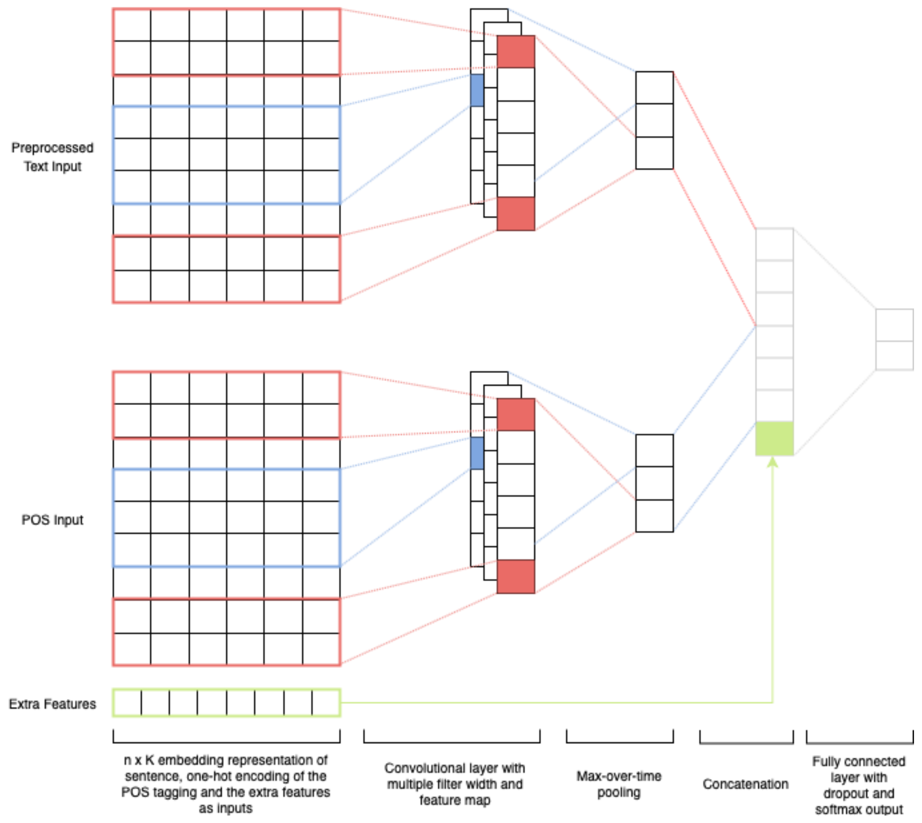

Preprocessed Text Input
POS Input
Extra Features
n x K embedding representation of sentence, one-hot encoding of the POS tagging and the extra features as inputs
Convolutional layer with multiple filter width and feature map
Max-over-time pooling
Concatenation
Fully connected layer with dropout and softmax output

Figure 3.2: Model architecture with two channels for an example sentence.

The PoS tagging method has been used to provide more information to the neural network in order to get an advantage of the meaning of a sentence based on the grammatical structure. As been showed that this could be a crucial point for what concerns the hate sentence because they could be clustered according to their structure. In fact, as shown in [12], a hate sentence could be a verbless one. Since most of the phrases containing stereotypes are hate phrases we decided to leverage this type of information. Although it might confuse the neural network that a stereotypes sentence is necessarily a hate sentence too. The latter problem could be solved automatically during the training phase since it is aimed at stereotype classification and the gradient back-propagate this information to the PoS K-CNN.

3.4 Double BiLSTM Architecture

After evaluating the two CNN-type architectures, we noticed some factors that could lead to some drawbacks in feature vector extraction. First, it is well known in the literature that K-CNN type networks are characterized by bottlenecks generated by max pooling, in fact, several solutions have been presented to solve this problem as in show in [13, 14]. Moreover, K-CNNs take advantage of a local perspective of the data and a global view could lead to better performance in some scenario. So, we have decided to move to the Recurrent Neural Network (RNN) architecture that is trained to recognize patterns across time and are intended to capture word dependencies and text structures, while a CNN learns to recognize patterns across space. Moreover, RNNs are designed to make use of sequential data, when the current step has some kind of relation with the previous steps. This makes them ideal for applications with a time component (audio, time-series data) and natural language processing.

Among many variants to RNNs, Long Short-Term Memory (LSTM) is the most popular architecture, which is designed to better capture long term dependencies and addresses the gradient vanishing/exploding problems suffered by the vanilla RNNs. Ultimately we decided to make us of a Bidirectional LSTM (BiLSTM) that consist of two LSTMs, one that takes the inpunt in a forward direction, and the other in backwards direction. This allow the model to increase the amount of information available for the softmax layer.

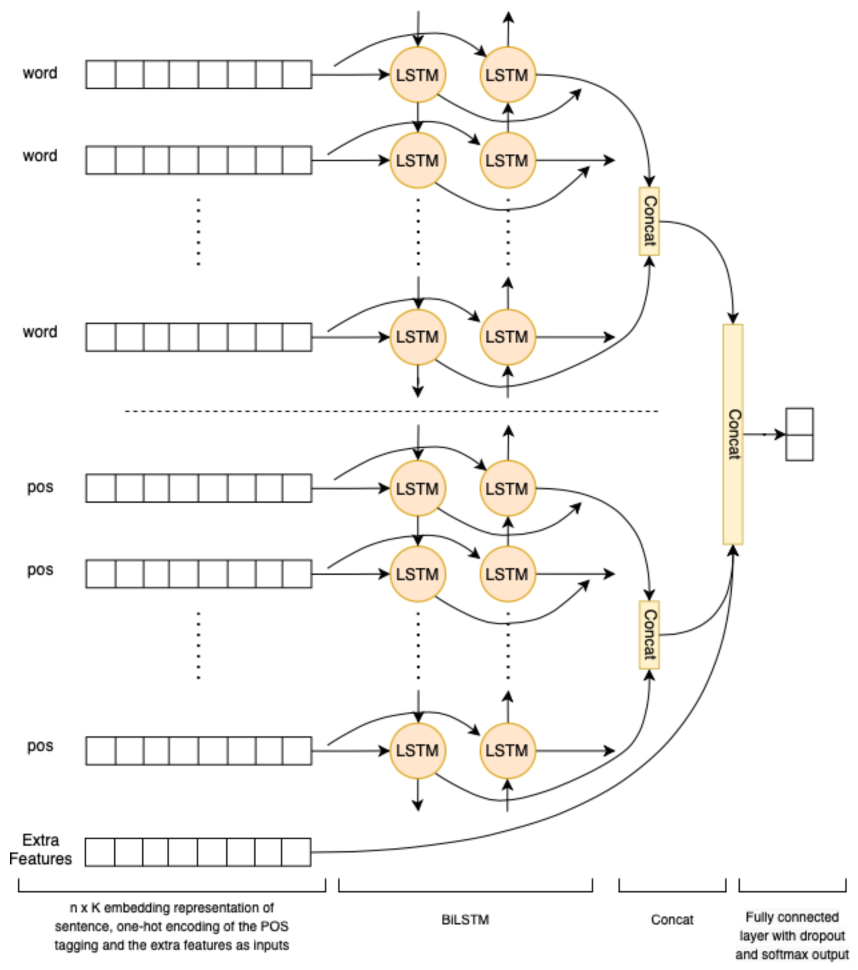

word
word
word
pos
pos
pos
Extra Features
LSTM
Concat
Concat
Concat
n x K embedding representation of sentence, one-hot encoding of the POS tagging and the extra features as inputs
BiLSTM
Concat
Fully connected layer with dropout and softmax output

Figure 3.3: Model architecture with two channels for an example sentence.

As shown in Fig. 3.3 our architecture, named D-BLSTM, is composed of two parallel BiLSTM, one that takes as input the word embedding and the other one takes the one-hot encoding of the PoS. Then, the output of the forward and backward processes of both BiLSTM is concatenated with the extra feature and fed to the softmax layer.

Moreover, the choice of BiLSTM seemed to fit the task of stereotype recognition. It has been shown [15] that when someone reads certain terms in a text, such as names that indicate a job, he or she usually associate a gender rather than another leading to a gender stereotype. If, for example, the term "elettricista" has been associated with the male gender and later in the text there is a female pronoun, it will make the reader understand that he or she has attributed the wrong gender to the subject of the text. This will cause the reader to go back and re-read the correct part and acquire the correct information.

3.5 Attention based BiLSTM Architecture

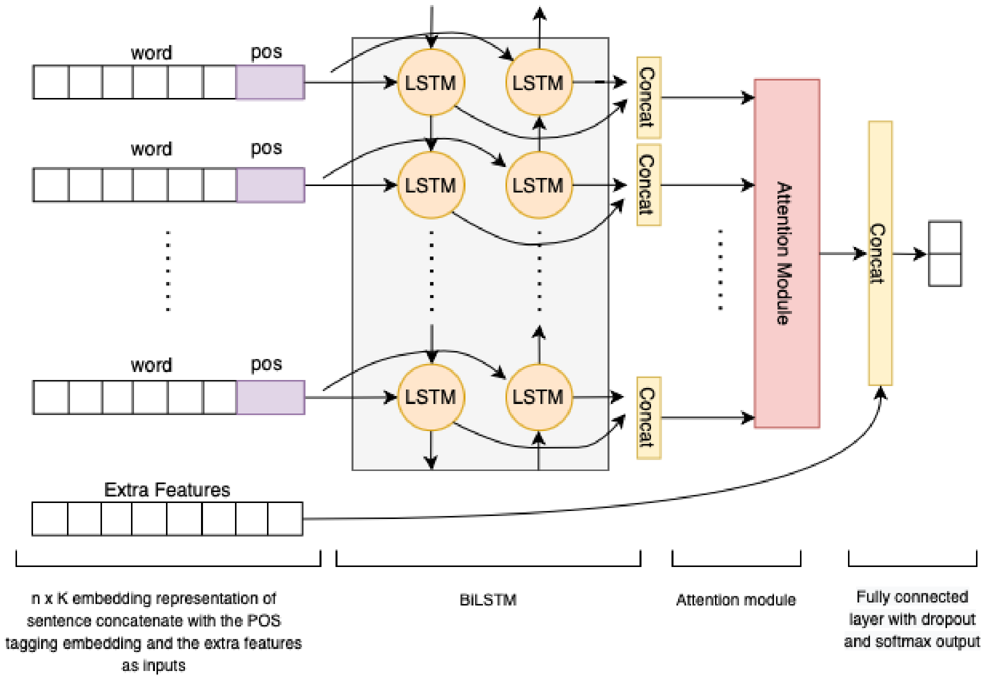

word
pos
LSTM
Concat
Attention Module
Extra Features
n x K embedding representation of sentence concatenate with the POS tagging embedding and the extra features as inputs
BiLSTM
Attention module
Fully connected layer with dropout and softmax output

Figure 3.4: Model architecture with two channels for an example sentence.

We realized that every previous model had some problem with the features vector extraction. In fact, they simply concatenate the features vector extracted from POS with the features vector extracted from words, increasing the input dimension of the following feed-forward network, and leading to the curse of dimensionality problem. Therefore, inspired by the work of A. Cimino, L. De Mattei, and F. Dell'Orletta [16], we changed the entry of the BiLSTM with pairs of (word, POS) to decrease the input dimension of the last fully connected layer. Furthermore, to take advantage as best as possible of the features extraction given by the BiLSTM, we weighted each output with an attention mechanism [17], as done by Q. Zhou, Z. Zhang, and H. Wu [18]. Not all words in a tweet contribute equally to the representation of a tweet, so, with this mechanism, the model learns which words to pay more attention to. Specifically, we have:

$$u_{ti} = tanh(Wh_{ti} + b),$$

$$\alpha_{ti} = \frac{exp(u_{ti}^T u_w)}{\sum_{j=1}^{n} exp(u_{tj}^T u_w)},$$
$$s_t = \sum_i \alpha_{ti} h_{ti}.$$

Where t represents t-th tweet, i represents i-th word in the tweet and n is the number of words in a tweet. h_{ti} is the concatenation output of the BiLSTM layer in our model, and u_{ti} is its hidden representation given by one-layer MLP (with weight matrix W and bias b). The set of $\{\alpha_{ti}\}$ are weights defining how much of each source hidden state should be considered for each output, and the bigger α_{ti} is, the more important the i-th word is for our classification task. Finally, s_t is the sentence vector represented as a weighted sum of the word annotations (h_{ti}). Lastly, the output of the attention layer is concatenated with the extra feature and fed to the softmax layer. This model was named A-BiLSTM.

3.6 BERT Architecture

Finally, due to well-known effectiveness in the use of context-dependent and task-free language understanding models (such as ELMo, GPT, and BERT), and after they have been proved to achieve the state of the art performance in numerous complex NLP tasks, we decided to try a pre-trained BERT [19]. BERT models are designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers, which allows the model to learn the context of a word based on all of its surroundings. Going into detail, the training is performed combining a "masked language model" (MLM) pre-training objective with a "next sentence prediction". The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Regarding the "next sentence prediction" task, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. In order to use a pre-trained BERT model, we have just to initialize the model with the pre-trained parameters and fine-tune them with our dataset of labeled text. There are different fine-tuning techniques, and we decided to train the entire pre-trained model on our dataset and feed the output to a softmax layer. In this case, the error is back-propagated through the entire architecture, and the pre-trained weights of the model are updated based on the new dataset. The Italian BERT model used is made available by the MDZ Digital Library team at the Bavarian State Library [11], and is pre-trained on recent Wikipedia dump and various texts from the OPUS corpora collection [12], with a final corpus size of 13GB and 2,050,057,573 tokens. The cased version was chosen, being more suitable for the proposed pre-processing method. Previously we faced the idea of using AlBERTo [20], a pre-trained BERT for the Italian language trained using a large number of Italian tweets retrieved from TWITA (a corpus of Tweets in the Italian language collected from February 2012 to nowadays from Twitter's official streaming API). However, as evidenced by A. Pota, M.Ventura, R.Catelli, and M. Esposito in [21], pre-trained BERT models on large corpus of plain text (instead of a collection of tweets) can perform better than AlBERTo for Twitter sentiment analysis. In addition, our pre-processing procedure converted emojis and emoticons into plain text, causing the structure of our data to move away from the format of a tweet. For these reasons, we abandoned AlBERTo, in favor of dbmdz BERT model. The dbmdz architecture is shown in in Fig. 3.5.

11. `https://huggingface.co/dbmdz/`
12. `https://opus.nlpl.eu/`

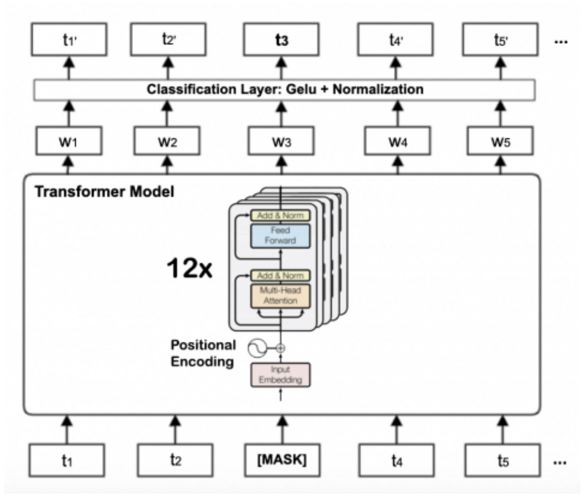


Figure 3.5: Dbmdz BERT model architecture.

There are two main architecture of BERT, the base and the large, and they differ mainly in four fundamental aspects: the number of hidden layers in the transformer encoder (transformer blocks), the number of attention heads (also known as self-attention [17]), the hidden size of the feed-forward networks, and finally the maximum sequence length parameter (the maximum accepted input vector size). In our work the base architecture is used, and the corresponding hyper-parameters are reported in Tab. 3.1.

Hyperparameters	**Value**
#Hidden layers	12
#Attention heads	12
Hidden size	768
Maximum sequence length	512

Table 3.1: Hyper-parameters of the fine-tuned Italian BERT Cased.

4. Experiments

We conducted a series of experiments to evaluate the proposed models for Italian stereotype detection tasks (CNN, Double CNN, Double BiLSTM, Attention BiLSTM, and BERT). In all experiments, we performed a binary classification task in which tweets were classified for having a stereotype or not. In this section, we describe the experiment setup, including the cross-validation method, model implementation software, hyperparameter tuning, evaluation metrics, and all results obtained for each model.

4.1 Experimental Settings

We randomly divided the original design set of HaSpeeDe 2020 into *Training Set* (80%) and *Validation Set* (20%). Concerning the experimental setting for the KCNN, D-KCNN, D-BiLSTM, and A-BiLSTM models, for the hyperparameters search, we performed a grid search for each model by using 5-folds cross-validation method, instead, for the BERT model a hold-out method was used. From the Training Set, a 10% has been extracted to implement the early stopping technique, which allowed us to stop the training once the model reached the minimum of the loss on the Validation Set. We chose the Binary Cross Entropy as a loss function. The early stopping allowed us to regularize the model, in addition to the dropout. Finally, the best model for each architecture has been retrained and evaluated on the *Test Set*. The Colab and the server provided during the course were used for all experiments.

4.2 Grid-search Results

All the results obtained with the grid search for each model will follow here.

4.2.1 K-CNN

After a screening phase that allowed us to isolate some hyperparameters ranges, a grid search with the values reported in Tab. 4.1 has been performed. Then, in in Tab. 4.2 the best three models resulting from the grid search sorted sorted by the macro F1 on the validation test.

Hyperparameters	Values range
#filters	32, 128, 256
filter sizes	[2, 3, 4], [3, 4, 5], [4, 5, 6]
dropout	0.1, 0.5, 0.9
#hidden neurons (HN)	64, 124, 512
η	0.1, 0.01, 0.001, 0.0001

Table 4.1: Range of Hyper-parameters for the grid search of the K-CNN.

#filters	filter sizes	dropout	#HN	η	loss (TR)	loss (VL)	macro F1 (TR)	macro F1 (VL)
256	[2, 3, 4]	0.5	64	0.0001	0.376 ± 0.035	0.560 ± 0.011	0.852 ± 0.018	0.702 ± 0.009
256	[2, 3, 4]	0.5	64	0.001	0.383 ± 0.068	0.571 ± 0.013	0.852 ± 0.037	0.700 ± 0.009
256	[2, 3, 4]	0.5	124	0.0001	0.383 ± 0.025	0.562 ± 0.005	0.849 ± 0.019	0.696 ± 0.007

Table 4.2: K-CNN best models results from the 5-folds cross-validation method. Mean and standard deviation are reported for the Binary Cross entropy Loss and macro F1-score over the Training and Validation sets. Models are sorted by the macro F1 (VL).

4.2.2 D-KCNN

Here, the same hyper-parameters ranges used for the K-CNN model was used (Tab. 4.1) and in Tab. 4.3 the best hyper-parameter configurations are shown.

#filters	filter sizes	dropout	#HN	η	loss (TR)	loss (VL)	macro F1 (TR)	macro F1 (VL)
512	[2, 3, 4]	0.5	512	0.001	0.3666 ± 0.067	0.561 ± 0.017	0.871 ± 0.042	0.704 ± 0.010
128	[2, 3, 4]	0.5	128	0.0001	0.396 ± 0.015	0.561 ± 0.008	0.836 ± 0.008	0.704 ± 0.011
512	[2, 3, 4]	0.5	128	0.0001	0.346 ± 0.034	0.560 ± 0.007	0.873 ± 0.022	0.703 ± 0.008

Table 4.3: D-CNN best models results from the 5-folds cross-validation method. Mean and standard deviation are reported for the Binary Cross entropy Loss and macro F1-score over the Training and Validation sets. Models are sorted by the macro F1 (VL).

4.2.3 D-BiLSTM

Hyper-parameters ranges and the best hyper-parameter configurations are shown in Tab. 4.4 and in Tab. 4.5.

Hyperparameters	Values range
#BiLSTM units	64, 128, 256, 512
Feed forward dropout	0.1, 0.3, 0.5, 0.7, 0.9
#Hidden Neurons (#HN)	64, 128, 256, 512
η	1e-2, 1e-3, 1e-4

Table 4.4: Range of Hyper-parameters for the grid search of the D-BiLSTM.

#B_units	**dropout**	**#HN**	η	**loss (TR)**	**loss (VL)**	**macro F1 (TR)**	**macro F1 (VL)**
512	0.7	128	0.001	0.624 ± 0.138	0.600 ± 0.047	0.767 ± 0.014	0.717 ± 0.006
128	0.7	64	0.001	0.623 ± 0.167	0.601 ± 0.040	0.792 ± 0.024	0.714 ± 0.004
256	0.9	256	0.001	0.631 ± 0.156	0.606 ± 0.037	0.789 ± 0.023	0.713 ± 0.007

Table 4.5: D-BiLSTM best models results from the 5-folds cross-validation method. Mean and standard deviation are reported for the Binary Cross entropy Loss and macro F1-score over the Training and Validation sets. Models are sorted by the macro F1 (VL).

4.2.4 A-BiLSTM

Ranges and the best hyper-parameter configurations are shown in Tab. 4.6 and in Tab. 4.7.

Hyperparameters	**Values range**
#BiLSTM units (#B_units)	64, 128, 256, 512
BiLSTM input dropout (B_dropout)	0.1, 0.3, 0.5, 0.7, 0.9
Feed forward dropout	0.1, 0.3, 0.5, 0.7, 0.9
#Hidden Neurons (#HN)	64, 128, 256, 512
η	1e-2, 1e-3, 1e-4

Table 4.6: Range of Hyper-parameters for the grid search of the A-BiLSTM.

#B_units	**B_dropout**	**dropout**	**#HN**	η	**loss (TR)**	**loss (VL)**	**macro F1 (TR)**	**macro F1 (VL)**
128	0.5	0.1	512	0.01	0.449 ± 0.025	0.554 ± 0.017	0.788 ± 0.019	0.722 ± 0.011
128	0.5	0.1	64	0.01	0.458 ± 0.020	0.554 ± 0.014	0.776 ± 0.017	0.721 ± 0.006
128	0.5	0.7	256	0.01	0.467 ± 0.027	0.557 ± 0.016	0.767 ± 0.020	0.721 ± 0.006

Table 4.7: A-BiLSTM best models results from the 5-folds cross-validation method. Mean and standard deviation are reported for the Binary Cross entropy Loss and macro F1-score over the Training and Validation sets. Models are sorted by the macro F1 (VL).

4.2.5 BERT

Ranges and the best hyper-parameter configurations are shown in Tab. 4.8 and in Tab. 4.9.

Hyperparameters	**Values range**
weight decay	0.01, 0.001
batch size (TR)	16, 32
η	5e-5, 1e-5, 1e-6

Table 4.8: Range of Hyper-parameters for the grid search of the ALBERTo.

weight decay	**batch size (TR)**	η	**loss (TR)**	**loss (VL)**	**macro F1 (TR)**	**macro F1 (VL)**
0.01	16	5e-5	0.389	0.538	0.826	0.732
0.01	32	1e-5	0.414	0.562	0.817	0.709
0.01	16	1e-6	0.478	0.572	0.768	0.694

Table 4.9: BERT best models results from the grid search. The Binary Cross entropy Loss and macro F1-score over the Training and Validation sets. Models are sorted by the macro F1 (VL).

4.3 Final Models Comparison

For each model, we have chosen the best hyper-parameters configuration (that leads to the best macro F1 score on the validation set) and then retrained all the models on the Training and Validation set. Then we evaluated them on the test set. In Tab 4.10 all models are compared with each other and also with the baseline, the results are sorted by the macro F1-score on the test set. All the learning curves for each model are shown in Fig 4.1, 4.2, 4.3, 4.4, 4.5.

Model	loss	val loss	test loss	macro F1	val macro F1	test macro F1
BERT	0.389	0.538	0.529	0.826	0.732	0.737
A-BiLSTM	0.465	0.537	0.542	0.765	0.731	0.722
D-KCNN	0.3821	0.5351	0.552	0.843	0.720	0.715
Baseline_SVC	-	-	-	-	-	0,714
D-BiLSTM	0.4970	0.5232	0.5630	0.752	0.727	0.703
KCNN	0.3665	0.5515	0.5631	0.862	0.722	0.700
Baseline_MFC	-	-	-	-	-	0,354

Table 4.10: Final results for all the models compared to the baseline (sorted by macro F1 score on test)

Looking at the results, we can see that the choices made in the various architectures improved the results of the test set. For example, the choice to add the PoS tagging as a parallel input in the D-KCNN improved the performance obtained with the KCNN from 0.700 to 0.715. Instead, it is clear that the D-BiLSTM was a more complex model and the results obtained align with the KCNN. In fact, the next network, A-BiLSTM performed better in comparison to all the no BERT models due to the initial combination of the PoS and the word given as input to the BiLSTM and the attention module that was able to efficiently combine all the outputs of it. In fact, we note that the A-BiLSTM deviates much less between the macro F1-score value obtained between the validation and the test set.

Regarding the BERT model, as expected, even in our task it performed much better than all other models reaching the value of 0.737 on the test set. With a much more efficient grid search, we deduce that we could have obtained much higher values. Furthermore, as mentioned by Risch and Krestel (2020) [22], the BERT model suffers from the small size of the dataset, in fact, transformer-type models suffer from high variance concerning the input.

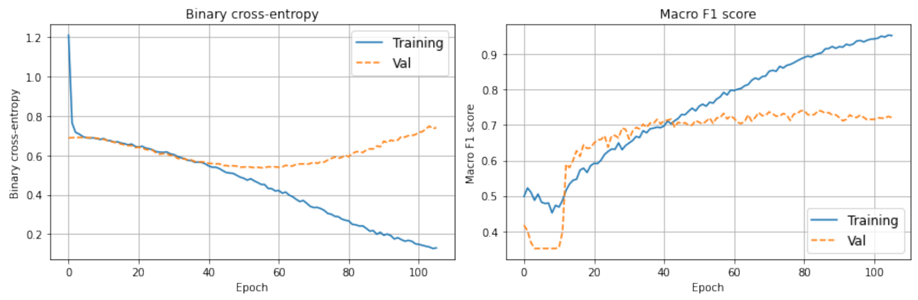


Figure 4.1: KCNN training and validation learning curves for the Binary cross-entropy and macro F1-score.

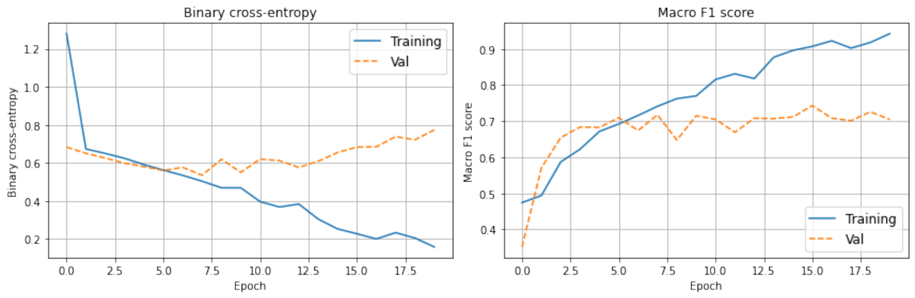


Figure 4.2: D-KCNN training and validation learning curves for the Binary cross-entropy and macro F1-score.

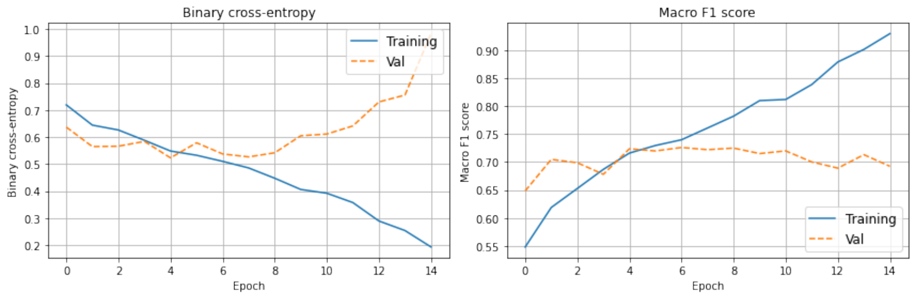


Figure 4.3: D-BiLSTM training and validation learning curves for the Binary cross-entropy and macro F1-score.

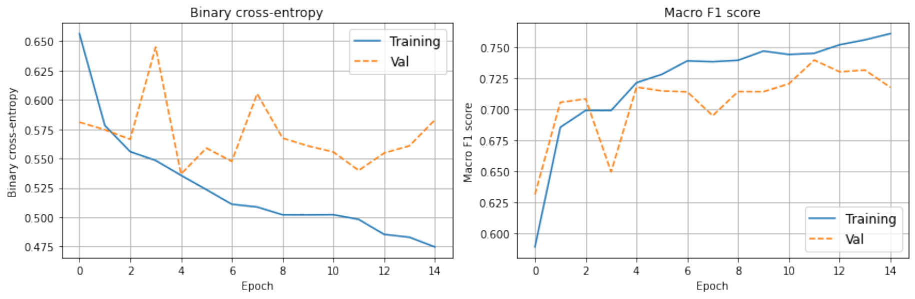


Figure 4.4: A-BiLSTM training and validation learning curves for the Binary cross-entropy and macro F1-score.

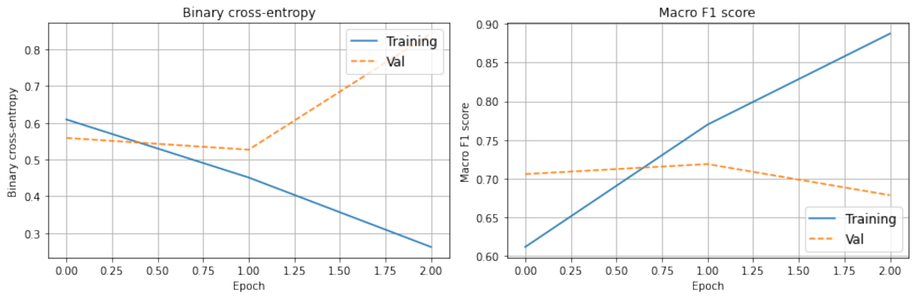


Figure 4.5: BERT training and validation learning curves for the Binary cross-entropy and macro F1-score.

4.4 Error Analysis

To understand the challenges of this task, we carried out further analysis of the errors made by our models. We sampled from the test set some sentences and predict the results with each model, the results are shown in Tab. 4.10. By looking at the second tweet, we can see how all the models were not able to label it correctly. It contains the word "immigrati" and some bad words. In this case, surely the decision was made considering that the tweet contained a phrase as a reference to a group stereotype. This scenario is also observed in the third sentence, again we see the presence of sentences related to ethnic groups, but this time all models manage to classify the sentence correctly. Thus, it is difficult for models to classify sentences containing hate phrases but without a stereotype.

Preprocessed Sentence	**True Value**	**KCC**	**D-KCNN**	**D-BiLSTM**	**A-BiLSTM**	**BERT**
Tutti i politici rubano , hanno rubato e ruberanno ! La Fornero ha rubato 10 mila euro a mio padre ! Ma basta con lecchini di immigrati che si ciondolano tutto il giorno con cellulari stracostosi e sigarette e per noia violentano o rubano ! Vuoi che tuo figlio conosca il Natale o no ?	1	1	1	0	1	1
Rosarno , le case popolari ? Solo a gli immigrati Hanno avuto bisogno di governi non eletti , di gente imposta ad un popolo disarmato . Una volta messi li , i Vigliacchi hanno dato inizio a la ns fine ! Se e quando si scatenerà la rabbia vera , ne farò parte ! !	0	1	1	1	1	1
i musulmano ammazzano tutti quelli che per il loro fumoso cervello sono " infedeli " . i nostri terroristi istituzionali ci obbligano ad accogliere e mantenere i nostri assassini	1	1	1	1	1	1
l' accusa di la sinistraglia cialtrona immigrazionista a < Salvini > è che ha impedito che altri parassiti afro islamici infestassero l' Italia e ha danneggiato il business di l' immigrazione clandestina . Solita feccia rosicona . < Salvini Non Mollare >	1	0	1	1	1	1
A le idiote pdinne gli serve questo trattamento forse solo dopo si ricrederanno e dico forse tanto sono coglione idiote dementi . Io dico unica salvezza Nuove Crociate Contro L' Islam Viva Riccardo Cuor Di Leone . Nuove Crociate Altro Che Accoglienza ! !	0	1	1	1	0	0

Table 4.11: Random sentences labeled by the models.

5. Conclusion and Future work

From the beginning, we realized that the stereotype detection task was very complex. In fact, the choice of architecture could easily lead to overfitting. Moreover, since the training set was characterized by a certain distribution of data, due to the sampling done by the organizers, we had to be very careful to avoid overfitting and loss of generalization. Finally, we learned that the variation in the distribution of tweets is really wide, caused by the fact that depending on the historical period, messages are more often about certain topics than others. From the study of the architectures, we drew two main conclusions. First, the correct architecture is crucial to obtain a good model. Thus, to build a good model it is necessary to consider the task at hand, taking advantage of feature extraction as much as possible. Hence, the right architecture, combined with an efficient hyper-parameters search, could lead to a small but sufficiently accurate model. Second, as expected, we also noticed how the BERT models manage to outperform the results obtained with the classical architectures known in the literature. But of course, they are also characterized by some drawbacks such as high variance due to fine-tuning with small datasets.

Finally, these two points lead to the fact that is important the correct choice of the model in the deploying phase. Depending on the required constraints (size, accuracy, predictive speed) a less accurate but more efficient model in terms of time and space could be considered better than a bigger and more accurate model (or vice versa). For example, it could happen that on embedded systems it could be necessary to use very small models but with good accuracy.

As regards future work:

- Initially, before contacting D'orletta to get access to the word embedding used in this work, we consider creating our own by using FastText and the tweets provided by the TWITA Datasets project of the University of Turin. We thought that by considering tweets related to certain keywords, the most common ones in the design set, eg, immigrants, Islam, migrants, and others, we could get a valid embedding.
- Perform a data augmentation technique, like *self-supervised data augmentation* [23], to improve learning models' performances since only a moderate amount of labeled data is available. This solution may help, in particular, the BERT model, which tends to have a high variance for the input dataset [22].
- In the D-KCNN model, we can have the same filter size both for POS and text. In that way, we can replace the concatenation of the two features maps with an average pooling, reducing the input dimension of the following feed-forward layer.
- In the K-CNN model, we could have used dynamic pooling to get more information out of data. However, being the tweet composed of few tokens, it seemed like a trivial effort to extract the n-grams dynamically, then we preferred to focus on BERT models.

References

[1] John T. Nockleby. *Encyclopedia of the American Constitution*, pages 1277–1279. Macmillan, 2nd edition, 2019.

[2] Tara Ross. *Media and Stereotypes*, pages 1–17. Springer Singapore, Singapore, 2019.

[3] Manuela Sanguinetti, Gloria Comandini, Elisa Nuovo, Simona Frenda, Marco Stranisci, Cristina Bosco, Tommaso Caselli, Viviana Patti, and Irene Russo. Haspeede 2 @ evalita2020: Overview of the evalita 2020 hate speech detection task. 12 2020.

[4] Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[5] De Mattei Cimino and Dell'Orletta. Multi-task learning in deep neural networks at evalita 2018. In *Proceedings of EVALITA '18, Evaluation of NLP and Speech Tools for Italian*, Turin, Italy, December 2018. Association for Computational Linguistics.

[6] I. Russo, Francesca Frontini, and Valeria Quochi. Opener sentiment lexicon italian - lmf. 2016.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: A comprehensive review. *ACM Comput. Surv.*, 54(3), April 2021.

[10] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[12] G. Comandini, M. Speranza, and B. Magnini. Effective communication without verbs? sure! identification of nominal utterances in italian social media texts. In *CLiC-it*, 2018.

[13] Anshul Mittal, Noveen Sachdeva, Sheshansh Agrawal, Sumeet Agarwal, Purushottam Kar, and Manik Varma. Eclare: Extreme classification with label graph correlations. In *Proceedings of the Web Conference 2021*, WWW '21, page 3721–3732, New York, NY, USA, 2021. Association for Computing Machinery.

[14] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning based text classification: A comprehensive review, 2021.

[15] Susan A. Duffy and Jessica A. Keir. Violating stereotypes: Eye movements and comprehension processes when text conflicts with world knowledge. *Memory and Cognition*, 32 (4), 2004.

[16] A. Cimino, Lorenzo De Mattei, and F. Dell'Orletta. Multi-task learning in deep neural networks at evalita 2018. In *EVALITA@CLiC-it*, 2018.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[18] Qimin Zhou, Zhengxin Zhang, and Hao Wu. Nlp at iest 2018: Bilstm-attention and lstm-attention via soft voting in emotion classification. 10 2018.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[20] Marco Polignano, Pierpaolo Basile, Marco de Gemmis, Giovanni Semeraro, and Valerio Basile. AlBERTo: Italian BERT Language Understanding Model for NLP Challenging Tasks Based on Tweets. In *Proceedings of the Sixth Italian Conference on Computational Linguistics (CLiC-it 2019)*, volume 2481. CEUR, 2019.

[21] Marco Pota, Mirko Ventura, Rosario Catelli, and Massimo Esposito. An effective bert-based pipeline for twitter sentiment analysis: A case study in italian. *Sensors*, 21(1), 2021.

[22] Julian Risch and Ralf Krestel. Bagging BERT models for robust aggression identification. In *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*, pages 55–61, Marseille, France, May 2020. European Language Resources Association (ELRA).

[23] Gabriele Sarti. Umberto-mtsa @ accompl-it: Improving complexity and acceptability prediction with multi-task learning on self-supervised annotations, 2020.