UNIVERSITÀ DI PISA

Human Language Teachnologies

Go Emotions

Tommaso Amarante, Iacopo Bicchierini

Github repository: https://github.com/Bicchie/News-Classification

Contents

Abstract

The problem of sentiment analysis is well known and can be addressed in several ways, in this work we approach two methods like Transformers, in particular we make a comparison between some members of the BERT family, and CNN. The aim of this paper is to perform the multi-label classification task on a dataset that contains Reddit comments, finding the best approach to assign emotions.

Chapter 1

Introduction

Nowadays the Human Computer Interaction is one of the most studied research area. Being the text, along with the speech, the main way to communicate between humans, understanding the mood of the user can be very useful to improve services like ChatBot, Review Analysis, Marketing Research and so on. It is also important to consider that human emotions are complex and non-binary and therefore there may be multiple emotions expressed by a sentence among a fairly large pool. Our aim is to review the most prominent approaches to automatically give emotions labels to a sentence.
Our work focuses on the performance analysis of the two most used architectures for the task of multi-label classification: **Transformers** and **CNN**. The Transformer architecture, in particular the BERT family, is analyzed in deep in most of its variants. We experimented two approaches to address the problem of class unbalancing: *class weight* and *data augmentation*.

Chapter 2

Data Collection and Preparation

2.1 Dataset

We decided to use the **Go Emotion** dataset from Hugging Face (1) because it is suitable for the multi label classification task. It is a collection of Reddit comments labeled with a set of emotions. Each row in the data is a different Reddit comment, the row has this structure:

- **Text** [string]: The Text content of the comment.
- **Tags** [list of integers]: Contains the indexes of the emotions related to the comment.

Positive		Negative		Ambiguous
admiration 👏	joy 😃	anger 😡	grief 😢	confusion 😕
amusement 😂	love ❤️	annoyance 😒	nervousness 😬	curiosity 🤔
approval 👍	optimism 🤞	disappointment	remorse 😔	realization 💡
caring 🤗	pride 😌	disapproval 👎	sadness 😞	surprise 😲
desire 😍	relief 😅	disgust 🤮		
excitement 🤩		embarrassment 😳		
gratitude 🙏		fear 😨		

Figure 2.1: Emotions in the Dataset

There are a total of 28 emotions divided into three different groups as shown in Figure 2.1, in addition to that set there is another emotions that is **Neutral**.

The comments were manually labeled by three English-speaking crowdworkers in India, this results in different biases. The most important is the inherent bias of Reddit itself, then we have to take into account that people from different cultures can have different sensibilities regarding the emotions conveyed from sentences.

2.2 Exploratory Data Analysis

This phase consists on understanding the characteristics of the Dataset. We extract the *frequency* in which each label appears in the sentences. The label that appears mostly is **Neutral**,

this can be understandable since it is frequent to make comment without expressing any emotion. There are a group of emotions with similar distribution like **Admiration**, **Amusement**, **Anger**, **Annoyance**, **Approval**, **Curiosity**, **Disapproval**, **Gratitude** and **Love**. Then some emotion are very rare like **Grief**, **Nervousness**, **Pride**, **Relief** and **Embarrassment**.

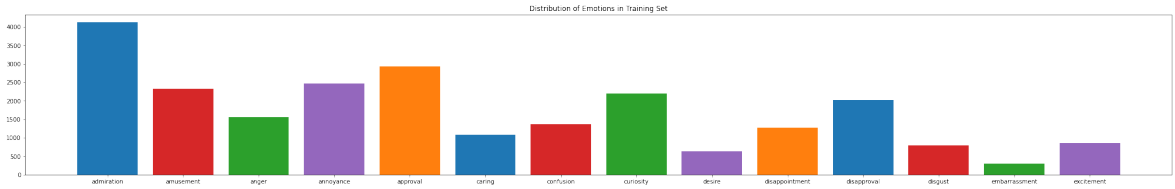

Figure 2.2: Number of Emotions

Figure 2.3: Number of Emotions

It's clear how the Dataset is **unbalanced**. Indeed in Section 2.3 we deal with this problem. It's worth to notice that being a Multi-Label classification problem, the distribution of emotions showed in Figure 2.2 and Figure 2.2 does not represent a one-to-one ratio with sentences since each comment could has up to 5 different emotions. As shown in the following report, the majority of the comments have only one label.

$$\{\mathbf{1} : 45446, \mathbf{2} : 8124, \mathbf{3} : 655, \mathbf{4} : 37, \mathbf{5} : 1\} \tag{2.1}$$

We find also the average length of the sentences per label, discovering how these values are included between 10 and 16 words and are pretty similar to each other.

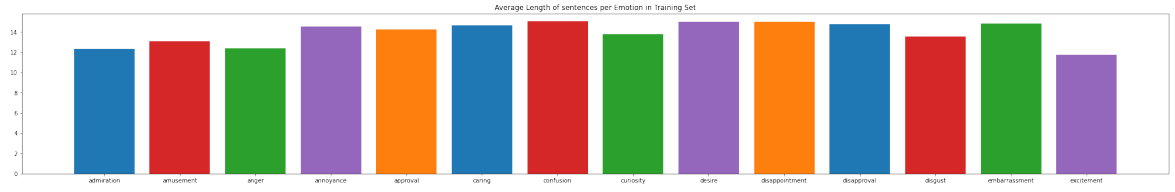

Figure 2.4: Average Length per Label

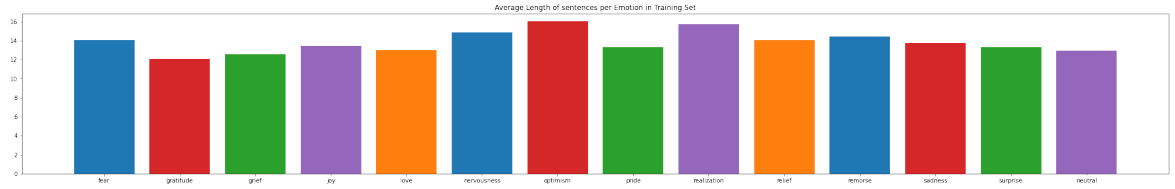

Figure 2.5: Average Length per Label

Another informal way to analyze the characteristics of the data is to scroll through them to see examples, by doing so we found some interesting aspects:

- the presence of **emoticon** that are facial expression representation using keyboard characters and punctuation.
- the presence of **emoji** that are small actual images used to express emotions or idea in text messages.
- some **names** are replaced with the token '[NAME]'.
- the presence of **acronym**.

2.3 Data Preprocessing

The dataset is loaded using HuggingFace API, the comments were already split into *Train*, *Validation* and *Test* set. The main operations to prepare our dataset are:

- *Conversion of the emojis into word*
- *Expand contracted forms*: for this purpose we can use the **contractions** library, this can help us to transform *"they're"* in *"they are"*.
- *Make all the words to lower case*: this steps helps to standardize the article's text.
- *Filter out all the chars that are not part of the normal alphabet*: using a regular expression we can remove all the non standard chars like *question or exclamation marks*, *parenthesis* or *quotation marks*.

The most interesting one is the *conversion of the emojis into words*. We found different approaches to address the presence of emojis, they are to completely eliminate them, to train a word embedding that considers the emoji like a word, to convert them into word. We convert them into word and the motivations behind this choice are that since we are performing sentiment analysis, can be dangerous to completely ignore them since they convey significant emotions. The possibility to train a word embedding was not achievable since we would need a huge corpus of data containing emojis.
In order to convert the emojis into words, we used a dictionary that map them, firstly we tried to use the UNICODE CLDR Short Name (2), this name does not provide the emotion but a description of the image (for example 'smiling face with heart-eyes'). So we built our dictionary based on the groups provided by UNICODE itself that divides emojis into emotions groups.

After this process we can add the column that contains the preprocessed text to the dataset and drop the raw text.

As we said in section 2.2, the dataset is heavily unbalanced as shown in figure **??**. To solve this problem we used two different approaches.
One approach is to use **Class Weight** the other one is to perform a **Data Augmentation** approach to increment the number of the minority emotions.

2.3.1 Class Weights

This method provides a weight or bias for each output label inversely proportional to the number of samples in that label. In particular in this work is used this formula:

$$\ln\left[\frac{\mu * total\_samples}{class\_samples}\right]$$

The resulting values are

- **admiration**: 2.158890852989586,
- **amusement**: 2.732158730093336,
- **anger**: 3.128005296588644,
- **annoyance**: 2.6729501093226418,
- **approval**: 2.4990988725257473,
- **caring**: 3.493746651823455,
- **confusion**: 3.263818440762169,
- **curiosity**: 2.7928101993491983,
- **desire**: 4.021894082023994,
- **disappointment**: 3.338939071230277,
- **disapproval**: 2.873081139364248,
- **disgust**: 3.8091003173098166,
- **embarrassment**: 4.771190733435295,
- **excitement**: 3.7361639914529854,
- **fear**: 4.094682871879315,
- **gratitude**: 2.5980905399896077,
- **grief**: 6.141118117090981,
- **joy**: 3.2042263435599234,
- **love**: 2.841919903383947,
- **nervousness**: 5.385057111120466,
- **optimism**: 3.1191107017351927,
- **pride**: 5.77539333763233,
- **realization**: 3.472808244638285,
- **relief**: 5.454485617552229,
- **remorse**: 4.18413774428142,
- **sadness**: 3.295001368198857,
- **surprise**: 3.518899351838552,
- **neutral**: 0.9225891615482902

2.3.2 Data Augmentation

This approach aim to reduce the unbalancing, especially for the least represented emotions mentioned in Section 2.2, creating new sample with that labels. Being a Multi-Label classification problem, this operation is not so straightforward like in the Multi-Class one, since augmenting samples belonging to one class could provoke increment in other classes. Because of that we decide to augment the sentences of these classes: Embarrassment, Grief, Nervousness, Pride, Relief, they are the ones with *less than 500 samples* in the training set. Since we do not found any standard approach we manage to augment this class using the following formula:

$$num\_aug = \lceil \frac{500}{class\_samples} \rceil \tag{2.2}$$

With the variable *num_aug* we mean the number of new samples to be created using the data augmentation function.
There were several techniques that could be used to make Data Augmentation in an NLP problem.

- **Back Translation**: translate the text data to some language and then translate it back to the original language. This can help to generate textual data with different words while preserving the context of the text data.
- **Easy Data Augmentation**: easy data augmentation uses traditional and very simple data augmentation methods.
- **NLP Albumentation**: this technique applies some of the ideas used in CV data augmentation in NLP. For example *shuffle sentences transform* that, given the text sample containing multiple sentences, they are shuffled to create new sample.

The first one, even if very interesting as approach, could be difficult because of the nature of the dataset that presents a lot of informal comment also with slang. The translation machine would have struggled in the translation process. The third one instead, can't be used because each comment has one or few sentences. So we decided to perform the **Easy Data Augmentation** we use code provided by Wei, Jason and Zou, Kai in *Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks* (3). These techniques are the following:

- **Synonym Replacement (SR)**: 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 (RI)**: 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. Do this n times.
- **Random Swap (RS)**: Randomly choose two words in the sentence and swap their positions. Do this n times.
- **Random Deletion (RD)**: For each word in the sentence, randomly remove it with probability p.

Chapter 3

Architectures

Here we present all the approaches used in our multi-class classification task. We use a SOTA approach like **Transformers** and **CNN**. Together with the description of the method, we present also the results and in the case of BERT, the internal comparison between the architectures.
It's worth to mention the methodology used in order to train, validate and test all the architectures. The dataset is already split into **Train**, **Validation** and **Test** set, we only need to preprocess each split and use it for out purposes. The comparison is valuable since the Test set is the same for every architecture.
The values that we consider useful in order to assess a method are:

- *Label Ranking Average Precision Score*
- *Training Time*
- *Inference Time*
- *F1 macro*

They are explained better in Chapter 4.

Multi-Label Classification

In order to perform the Multi-Label classification task we used the **Binary Crossentropy** as loss function and the **Sigmoid** as activation function in the last layer. We need to use this type of loss function because it can answer several binary question, in our case this loss function answers to 28 different questions regarding all the emotions, the possible answers can only be 0 or 1. It uses the following formula where $\hat{y}_i$ is the $ii-th$ scalar value in the model output, y_i is the corresponding target value, and *output size* is the number of scalar values in the model output.

$$\text{Loss} = -\frac{1}{\substack{\text{output}\\\text{size}}} \sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i) \tag{3.1}$$

Talking about the activation function, it is mandatory to use the sigmoid if we use the binary crossentropy for the loss, this is because the sigmoid can produce an **independent** probability for each output class in range $[0, 1]$. This factor is fundamental for the correct computation of the loss.

3.1 BERT family

Bidirectional **E**ncoder **R**epresentations from **T**ransformers "**BERT**" is a transformer-based machine learning technique for natural language processing, it is the SOTA for many tasks. This architecture is based on:

- *Self Attention*: is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence, so it will consider also the context of the word taking into account the tokens before and after it.
- *Positional Encoding*: that injects input word-position information inside the encodings, it allows the model to be trained fast with respect to the RNN.

The Progenitor of the whole set of models used in this project is the **BERTbase** model. There are 3 parameters to configure the model:

- L: the number of encoders to be stacked (the left portion of the Transformer in Figure 3.1).
- H: the hidden dimension, so the dimension of the embeddings inside the network.
- A: the number of attention heads, the concept of multi-head Attention is a module for attention mechanisms which runs through an attention mechanism several times in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension. Intuitively, multiple attention heads allows for attending to parts of the sequence differently (e.g. longer-term dependencies versus shorter-term dependencies).

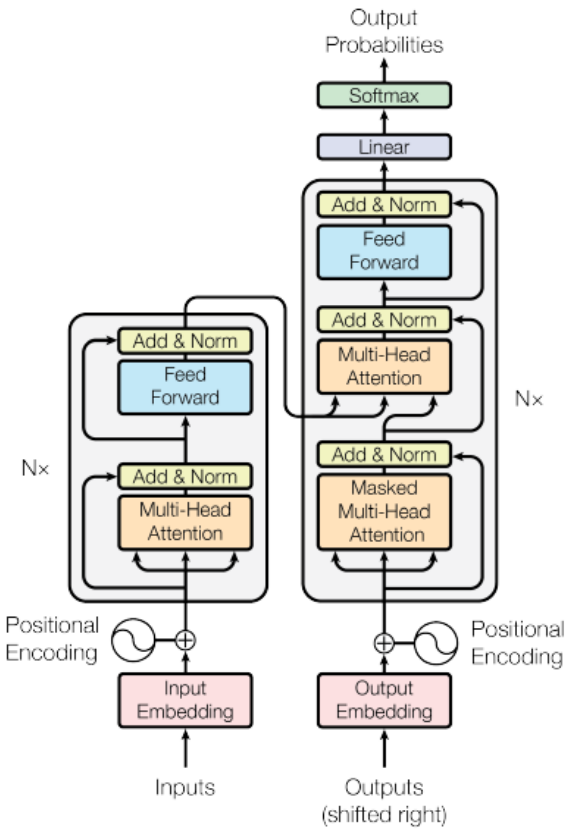

Figure 3.1: Architecture of Transformer from the paper "All you need is attention"

There are different configurations of the model, arranged using the parameters L, H and A. It is important to note that all these models have been trained in a self supervised way on *Wikipedia* and *BookCorpus* datasets using language modeling, specifically:

- *Masked Language Model*: as the name suggests, means we mask words from a sequence of input or sentences and the designed model needs to predict the masked words to complete the sentence, this is very useful for learning bidirectional representation of input.
- *Next Sentence Prediction*: consists of giving BERT two sentences, sentence A and sentence B. We then say, 'hey BERT, does sentence B come after sentence A?' — and BERT says either IsNextSentence or NotNextSentence. This is useful for capturing the relationship between pairs of input sentences.

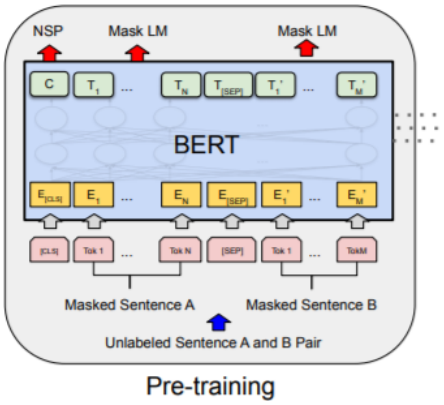

Figure 3.2: Illustrated pre-training of BERT

In order to fit our multi-label classification problem, we stack a **Dense Layer** with an output of 28 neurons, the number of emotions in our problem.

The **configuration of training** is the same for the whole set of BERT models:

- *BATCH SIZE*: 32.
- *EPOCHS*: 10.
- *Early Stopping* with *patience* equals to 2 on validation loss.
- *Optimizer*: RMSprop.

3.1.1 BERTbase

As we can see from this table, there are several configuration of BERT. The **BERTbase** is the configuration with **L=12**, **H=768** and **A=12**, highlighted in red. It has **110M** of parameters, where L = number of layers, H = hidden size and A = number of self-attention operations.

	H=128	**H=256**	**H=512**	**H=768**
L=2	4.4	9.7	22.8	39.2
L=4	4.8	11.3	29.1	53.4
L=6	5.2	12.8	35.4	67.5
L=8	5.6	14.4	41.7	81.7
L=10	6.0	16.0	48.0	95.9
L=12	6.4	17.6	54.3	110.1

Figure 3.3: Million of Parameters

Results with Class Weight Approach

In this part are shown the results obtained with this architecture. The Report shows an unbalanced behaviour on different classes, there are several classes with 0 Precision. Especially classes with few samples are struggling to be well classified. Also the F1-Macro is very very poor.

	precision	recall	f1-score	support
0	0.57	0.19	0.28	504
1	0.22	0.53	0.31	264
2	0.34	0.31	0.32	198
3	0.35	0.09	0.14	320
4	0.27	0.23	0.25	351
5	0.40	0.16	0.22	135
6	0.00	0.00	0.00	153
7	0.52	0.26	0.35	284
8	0.00	0.00	0.00	83
9	0.00	0.00	0.00	151
10	0.24	0.02	0.04	267
11	0.00	0.00	0.00	123
12	0.00	0.00	0.00	37
13	0.00	0.00	0.00	103
14	1.00	0.01	0.03	78
15	0.45	0.80	0.57	352
16	0.00	0.00	0.00	6
17	0.33	0.01	0.01	161
18	0.30	0.47	0.37	238
19	0.00	0.00	0.00	23
20	0.42	0.09	0.15	186
21	0.00	0.00	0.00	16
22	0.50	0.01	0.01	145
23	0.00	0.00	0.00	11
24	0.00	0.00	0.00	56
25	0.34	0.13	0.19	156
26	0.50	0.02	0.04	141
27	0.41	0.84	0.55	1787
micro avg	0.39	0.39	0.39	6329
macro avg	0.26	0.15	0.14	6329
weighted avg	0.36	0.39	0.30	6329
samples avg	0.40	0.40	0.40	6329

Figure 3.4: Report for all the emotions with Class Weight

The *training time* is 2495s in 5/10 Epochs thanks to Early Stopping, with an average of 499s per epoch.
The *inference time* on the test set is 53.17s.
The overall *LARP* is $54,46\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.14.

Results with Data Augmentation Approach

In this part are shown the results obtained with this architecture using the Data Augmentation. There is no sign of enhancement in the augmented classes, but the overall F1-Macro increases, as well as the LARP.

	precision	recall	f1-score	support
0	0.32	0.64	0.43	504
1	0.47	0.34	0.39	264
2	0.51	0.18	0.27	198
3	0.30	0.22	0.25	320
4	0.34	0.11	0.16	351
5	0.40	0.27	0.32	135
6	0.40	0.08	0.13	153
7	0.61	0.16	0.25	284
8	0.00	0.00	0.00	83
9	0.00	0.00	0.00	151
10	0.32	0.05	0.09	267
11	0.00	0.00	0.00	123
12	0.00	0.00	0.00	37
13	0.33	0.07	0.11	103
14	0.80	0.05	0.10	78
15	0.53	0.80	0.64	352
16	0.00	0.00	0.00	6
17	0.36	0.02	0.05	161
18	0.63	0.16	0.25	238
19	0.14	0.04	0.07	23
20	0.29	0.01	0.02	186
21	0.00	0.00	0.00	16
22	0.00	0.00	0.00	145
23	0.00	0.00	0.00	11
24	0.00	0.00	0.00	56
25	0.34	0.21	0.26	156
26	0.75	0.02	0.04	141
27	0.42	0.88	0.57	1787
micro avg	0.41	0.41	0.41	6329
macro avg	0.29	0.15	0.16	6329
weighted avg	0.38	0.41	0.32	6329
samples avg	0.42	0.42	0.42	6329

Figure 3.5: Report for all the emotions with Data Augmentation

The *training time* is 5149s in 10/10 Epochs, with an average of 515s per epoch.
The *inference time* on the test set is 53.10s.
The overall *LARP* is $56,84\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.16.

3.1.2 smallBERT

smallBERT has the same BERT architecture but fewer and smaller Transformer blocks and number of attention heads. The configuration is **L=4**, **H=512** and **A=8**, it is highlighted in green in the table 3.3. There is a novelty in this model, the usage of *knowledge distillation*. The idea is that the smaller student model is trained to recover the predictions of a highly accurate teacher using as ground truth, not only the hard labels of the dataset, but also soft labels related to the teacher's predictions, that allows better generalization. The soft labels coincide with the class probabilities produced by the teacher, obtained with a Softmax on the logits divided by a temperature parameter.

Results with Class Weight Approach

This architecture performs pretty well with respect to the previous one and also to the following BERT family solutions. It is interesting to observe how the F1-Macro is 27% so 10 points higher than the BERTbase one, even if the LARP metric is almost the same. This is due to the fact that LARP consider the overall prediction set, instead the Report (as we implemented It) considers only the max n predictions, with n the number of assigned labels in the ground truth.

	precision	recall	f1-score	support
0	0.55	0.47	0.51	504
1	0.50	0.62	0.55	264
2	0.36	0.35	0.36	198
3	0.23	0.23	0.23	320
4	0.30	0.16	0.21	351
5	0.27	0.15	0.19	135
6	0.42	0.12	0.18	153
7	0.43	0.39	0.41	284
8	0.30	0.19	0.24	83
9	0.18	0.11	0.14	151
10	0.22	0.34	0.27	267
11	0.72	0.11	0.18	123
12	0.00	0.00	0.00	37
13	0.36	0.16	0.22	103
14	0.57	0.22	0.31	78
15	0.68	0.85	0.75	352
16	0.00	0.00	0.00	6
17	0.48	0.15	0.23	161
18	0.47	0.71	0.56	238
19	0.00	0.00	0.00	23
20	0.25	0.47	0.33	186
21	0.00	0.00	0.00	16
22	0.26	0.12	0.17	145
23	0.00	0.00	0.00	11
24	0.38	0.38	0.38	56
25	0.43	0.24	0.31	156
26	0.33	0.16	0.22	141
27	0.51	0.67	0.58	1787
micro avg	0.44	0.44	0.44	6329
macro avg	0.33	0.26	0.27	6329
weighted avg	0.43	0.44	0.41	6329
samples avg	0.45	0.45	0.45	6329

Figure 3.6: Report for all the emotions with Class Weight

The *training time* is 1287s in 9/10 Epochs thanks to Early Stopping, with an average of 143s

per epoch.
The *inference time* on the test set is 14.77s.
The overall *LARP* is $59,86\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.27.

Results with Data Augmentation Approach

In this Report we can see how even if the F1-Macro decreases, the labels that was affected by the augmentation increase their performances as we can expect. Also the LARP increases by 0.5%.

	precision	recall	f1-score	support
0	0.49	0.49	0.49	504
1	0.51	0.54	0.53	264
2	0.46	0.06	0.11	198
3	0.38	0.12	0.18	320
4	0.42	0.05	0.10	351
5	0.31	0.13	0.19	135
6	0.38	0.09	0.15	153
7	0.47	0.31	0.37	284
8	0.50	0.06	0.11	83
9	0.00	0.00	0.00	151
10	0.28	0.09	0.13	267
11	0.82	0.07	0.13	123
12	0.50	0.03	0.05	37
13	0.38	0.05	0.09	103
14	0.62	0.17	0.26	78
15	0.77	0.82	0.79	352
16	0.25	0.33	0.29	6
17	0.45	0.11	0.18	161
18	0.59	0.56	0.57	238
19	0.00	0.00	0.00	23
20	0.44	0.27	0.34	186
21	0.22	0.12	0.16	16
22	0.35	0.08	0.12	145
23	0.00	0.00	0.00	11
24	0.33	0.02	0.03	56
25	0.31	0.29	0.30	156
26	0.52	0.09	0.15	141
27	0.42	0.92	0.57	1787
micro avg	0.45	0.45	0.45	6329
macro avg	0.40	0.21	0.23	6329
weighted avg	0.44	0.45	0.37	6329
samples avg	0.46	0.46	0.46	6329

Figure 3.7: Report for all the emotions with Data Augmentation

The *training time* is 1041s in 7/10 Epochs thanks to Early Stopping, with an average of 149s per epoch.
The *inference time* on the test set is 14.99s.
The overall *LARP* is $60,35\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.23.

3.1.3 BERTexpert

This model uses a BERTbase architecture pretrained from scratch on Wikipedia and BooksCorpus. This model is intended to be used for a variety of English NLP tasks. It is important to consider that the pre-training data contains more formal text and the model may not generalize to more colloquial text such as social media or messages.

Results with Class Weight Approach

In this report we can see that the F1-macro is quite low, moreover lots of classes have 0 Precision score. It important to notice that, even thougt the other metrics are pretty low, the LARP measure is acceptable.

	precision	recall	f1-score	support
0	0.39	0.27	0.32	504
1	0.38	0.48	0.42	264
2	0.21	0.12	0.15	198
3	0.29	0.08	0.12	320
4	0.20	0.09	0.12	351
5	0.20	0.21	0.21	135
6	0.22	0.01	0.02	153
7	0.47	0.20	0.28	284
8	0.33	0.01	0.02	83
9	0.00	0.00	0.00	151
10	0.31	0.10	0.15	267
11	0.33	0.02	0.03	123
12	0.00	0.00	0.00	37
13	0.00	0.00	0.00	103
14	0.00	0.00	0.00	78
15	0.54	0.54	0.54	352
16	0.00	0.00	0.00	6
17	0.29	0.05	0.08	161
18	0.46	0.37	0.41	238
19	0.00	0.00	0.00	23
20	0.21	0.23	0.22	186
21	0.00	0.00	0.00	16
22	1.00	0.01	0.03	145
23	0.00	0.00	0.00	11
24	0.00	0.00	0.00	56
25	0.25	0.01	0.01	156
26	0.26	0.04	0.06	141
27	0.38	0.87	0.53	1787
micro avg	0.37	0.37	0.37	6329
macro avg	0.24	0.13	0.13	6329
weighted avg	0.34	0.37	0.29	6329
samples avg	0.38	0.38	0.38	6329

Figure 3.8: Report for all the emotions with Class Weight

The *training time* is 4486s in 9/10 Epochs thanks to Early Stopping, with an average of 499s per epoch.
The *inference time* on the test set is 55.66s.
The overall *LARP* is $52,51\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.13.

Results with Data Augmentation Approach

BERTexpert seems not suitable for this tasks since the very poor values of the performances, this could be caused by the nature of the model that is pretrained using Wikipedia and BooksCorpus, so using formal language. Our dataset instead is composed mainly of informal comment on Reddit.

	precision	recall	f1-score	support
0	0.29	0.48	0.36	504
1	0.60	0.28	0.38	264
2	0.20	0.01	0.01	198
3	0.47	0.06	0.10	320
4	0.24	0.02	0.03	351
5	0.44	0.03	0.06	135
6	0.00	0.00	0.00	153
7	0.46	0.05	0.08	284
8	0.00	0.00	0.00	83
9	0.00	0.00	0.00	151
10	0.17	0.16	0.17	267
11	0.25	0.01	0.02	123
12	0.00	0.00	0.00	37
13	0.00	0.00	0.00	103
14	0.00	0.00	0.00	78
15	0.65	0.39	0.49	352
16	0.00	0.00	0.00	6
17	0.00	0.00	0.00	161
18	0.41	0.39	0.40	238
19	0.00	0.00	0.00	23
20	0.54	0.08	0.13	186
21	0.00	0.00	0.00	16
22	0.33	0.01	0.01	145
23	0.00	0.00	0.00	11
24	0.00	0.00	0.00	56
25	0.41	0.04	0.08	156
26	0.00	0.00	0.00	141
27	0.37	0.93	0.53	1787
micro avg	0.36	0.36	0.36	6329
macro avg	0.21	0.10	0.10	6329
weighted avg	0.32	0.36	0.26	6329
samples avg	0.38	0.38	0.38	6329

Figure 3.9: Report for all the emotions with Data Augmentation

The *training time* is 4119s in 8/10 Epochs thanks to Early Stopping, with an average of 515s per epoch.
The *inference time* on the test set is 53.35s.
The overall *LARP* is $51,75\%$, the metrics for all the classes is reported in the image above.
The *F1 - macro* is 0.10.

3.1.4 Electra

The **ELECTRA** model was proposed in the paper *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. ELECTRA is the pretraining approach, therefore there is nearly no changes done to the underlying model BERT, the configuration is **L=12**, **H=256** and **A=4**. This pretraining approach is the *replaced token detection* that randomly selects words in the text, but unlike the latter it does not mask them with a token, but replaces them with plausible alternatives returned by a small generator network, which samples from a proposal distribution. However this is not an adversarial model, in fact the generator is trained with maximum likelihood and not to fool the discriminator, due to the difficulty of applying GANs with text.

Results with Class Weight Approach

This model behaves very poorly with almost all the classes.

	precision	recall	f1-score	support
0	0.30	0.53	0.39	504
1	0.40	0.60	0.48	264
2	0.23	0.09	0.13	198
3	0.20	0.15	0.17	320
4	0.23	0.20	0.21	351
5	0.27	0.23	0.25	135
6	0.39	0.05	0.08	153
7	0.38	0.33	0.35	284
8	0.00	0.00	0.00	83
9	1.00	0.01	0.03	151
10	0.24	0.19	0.21	267
11	0.50	0.01	0.02	123
12	0.00	0.00	0.00	37
13	0.00	0.00	0.00	103
14	0.00	0.00	0.00	78
15	0.45	0.77	0.57	352
16	0.00	0.00	0.00	6
17	0.25	0.02	0.03	161
18	0.37	0.45	0.41	238
19	0.00	0.00	0.00	23
20	0.41	0.17	0.24	186
21	0.00	0.00	0.00	16
22	0.13	0.02	0.04	145
23	0.00	0.00	0.00	11
24	0.00	0.00	0.00	56
25	0.24	0.03	0.05	156
26	0.25	0.04	0.07	141
27	0.47	0.72	0.57	1787
micro avg	0.39	0.39	0.39	6329
macro avg	0.24	0.17	0.15	6329
weighted avg	0.35	0.39	0.33	6329
samples avg	0.40	0.40	0.40	6329

Figure 3.10: Report for all the emotions with Class Weight

The *training time* is 1094s in 7/10 Epochs thanks to Early Stopping, with an average of 156s per epoch.
The *inference time* on the test set is 16.59s.
The overall *LARP* is $53,76\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.15.

Results with Data Augmentation Approach

This model behaves very poorly with almost all the classes, the F1-macros is even lower with this approach.

	precision	recall	f1-score	support
0	0.36	0.40	0.38	504
1	0.57	0.52	0.54	264
2	0.38	0.02	0.03	198
3	0.33	0.09	0.14	320
4	0.27	0.08	0.12	351
5	0.34	0.15	0.21	135
6	0.60	0.02	0.04	153
7	0.51	0.15	0.24	284
8	0.00	0.00	0.00	83
9	0.67	0.01	0.03	151
10	0.29	0.04	0.07	267
11	0.00	0.00	0.00	123
12	0.00	0.00	0.00	37
13	0.00	0.00	0.00	103
14	0.00	0.00	0.00	78
15	0.61	0.60	0.60	352
16	0.00	0.00	0.00	6
17	0.00	0.00	0.00	161
18	0.49	0.36	0.42	238
19	0.00	0.00	0.00	23
20	0.33	0.02	0.04	186
21	0.00	0.00	0.00	16
22	0.00	0.00	0.00	145
23	0.00	0.00	0.00	11
24	0.25	0.02	0.03	56
25	0.75	0.02	0.04	156
26	0.00	0.00	0.00	141
27	0.37	0.96	0.54	1787
micro avg	0.39	0.39	0.39	6329
macro avg	0.25	0.12	0.12	6329
weighted avg	0.36	0.39	0.29	6329
samples avg	0.41	0.41	0.41	6329

Figure 3.11: Report for all the emotions with Data Augmentation

The *training time* is 1349s in 8/10 Epochs thanks to Early Stopping, with an average of 169s per epoch.
The *inference time* on the test set is 18.17s.
The overall *LARP* is $54,29\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.12.

3.1.5 Albert

Albert is light version of BERT with reduced number of parameters, that was presented in *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. This model introduces 3 main ideas to the original BERT's design choices:

- *Factorized embedding parameterization*: concerns to reduce the complexity of the representation of each word, that is not projected directly into the hidden space, but is first projected into a lower embedding space of dimension E.

- *Cross-layer parameter sharing*: states that all the parameters between transformer layers are shared, in particular both feed-forward network and sharing attention parameters.
- *Inter-sentence coherence loss*: changes the NSP task on which BERT is pre-trained, since with this approach the network learns more about topic prediction than coherence prediction. Here the same technique is used to generate positive samples (two consecutive segments of the same document), while for the negative examples are used the same two consecutive segments but with their order swapped.

Results with Class Weight Approach

In this part are shown the results obtained with this architecture. They are poor as we can see from the report.

	precision	recall	f1-score	support
0	0.20	0.66	0.30	504
1	0.63	0.51	0.56	264
2	0.33	0.01	0.01	198
3	0.31	0.11	0.16	320
4	0.00	0.00	0.00	351
5	0.29	0.17	0.21	135
6	0.17	0.01	0.01	153
7	0.39	0.30	0.34	284
8	0.00	0.00	0.00	83
9	0.50	0.01	0.01	151
10	0.22	0.03	0.05	267
11	0.00	0.00	0.00	123
12	0.00	0.00	0.00	37
13	0.00	0.00	0.00	103
14	0.00	0.00	0.00	78
15	0.61	0.58	0.59	352
16	0.00	0.00	0.00	6
17	0.00	0.00	0.00	161
18	0.47	0.07	0.12	238
19	0.00	0.00	0.00	23
20	0.13	0.50	0.21	186
21	0.00	0.00	0.00	16
22	0.00	0.00	0.00	145
23	0.00	0.00	0.00	11
24	0.00	0.00	0.00	56
25	0.18	0.11	0.14	156
26	0.00	0.00	0.00	141
27	0.44	0.70	0.54	1787
micro avg	0.35	0.35	0.35	6329
macro avg	0.17	0.13	0.12	6329
weighted avg	0.30	0.35	0.28	6329
samples avg	0.35	0.35	0.35	6329

Figure 3.12: Report for all the emotions with Class Weight

The *training time* is 3012s in $6/10$ Epochs thanks to Early Stopping, with an average of 502s per epoch.
The *inference time* on the test set is 54.30s.
The overall *LARP* is $49,72\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.12.

Results with Data Augmentation Approach

In this part are shown the results obtained with this architecture. They are poor as we can see from the report, the F1-Macro is even lower.

	precision	recall	f1-score	support
0	0.42	0.05	0.09	504
1	0.41	0.63	0.49	264
2	0.14	0.01	0.01	198
3	0.51	0.07	0.12	320
4	0.21	0.13	0.16	351
5	0.00	0.00	0.00	135
6	0.00	0.00	0.00	153
7	0.36	0.31	0.33	284
8	1.00	0.02	0.05	83
9	0.24	0.03	0.05	151
10	0.23	0.05	0.09	267
11	0.00	0.00	0.00	123
12	0.00	0.00	0.00	37
13	0.00	0.00	0.00	103
14	0.00	0.00	0.00	78
15	0.42	0.70	0.53	352
16	0.00	0.00	0.00	6
17	0.00	0.00	0.00	161
18	0.42	0.06	0.11	238
19	0.00	0.00	0.00	23
20	0.24	0.18	0.21	186
21	0.00	0.00	0.00	16
22	0.00	0.00	0.00	145
23	0.00	0.00	0.00	11
24	0.00	0.00	0.00	56
25	0.00	0.00	0.00	156
26	0.29	0.01	0.03	141
27	0.37	0.94	0.53	1787
micro avg	0.37	0.37	0.37	6329
macro avg	0.19	0.11	0.10	6329
weighted avg	0.29	0.37	0.25	6329
samples avg	0.38	0.38	0.38	6329

Figure 3.13: Report for all the emotions with Data Augmentation

The *training time* is 3191s in 6/10 Epochs thanks to Early Stopping, with an average of 532s per epoch.
The *inference time* on the test set is 54.64s.
The overall *LARP* is $51,56\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is 0.10.

3.1.6 Comparison

In this section we compare the models belonging to the **BERT family** in order to find the best one to be compared with the other methods.
The Table 3.1 shows a global overview about the models, the **number of parameters** ranges from the 12M of Albert to the 108M of *BERTbase* and *BERTexpert*. It is surprisingly instead the behaviour of the models with respect to the **training time** and **inference time**, since *Albert* has a quite high Inference Time despite being the tiniest model.
Another strange behaviour comes from *smallBERT* model that achieves the highest *LARP metric* despite being 4 times smaller in terms of number of parameters compared to the *BERTbase* and *BERTexpert* models. In addition we can say that **smallBERT** is the overall best model in all the evaluation metrics.
Another interesting aspect is the fact that Data Augmentation approach has always a smaller *F1-Macro* but an higher *LARP metric*.

Model	Parameters	LARP (%)	F1 - Macro	Inference Time (s)	Training Time (s)
BERTbase (CW)	108M	54.46	0.14	53.17	2495
BERTbase (DA)	108M	56.84	0.16	53.10	5149
smallBERT (CW)	29M	59.86	**0.27**	**14.77**	1287
smallBERT (DA)	29M	**60.35**	0.23	14.99	**1041**
BERTexpert (CW)	108M	52.51	0.13	55.66	4486
BERTexpert (DA)	108M	51.75	0.10	53.35	4119
Electra (CW)	14M	53.76	0.15	16.59	1094
Electra (DA)	14M	54.29	0.12	18.17	1349
Albert (CW)	12M	49.72	0.12	54.3	3012
Albert (DA)	12M	51.56	0.10	54.64	3191

Table 3.1

3.2 CNN

The structure of the CNN that we implemented is shown in Figure 3.14. The code in this section is inspired by the notebook (4).

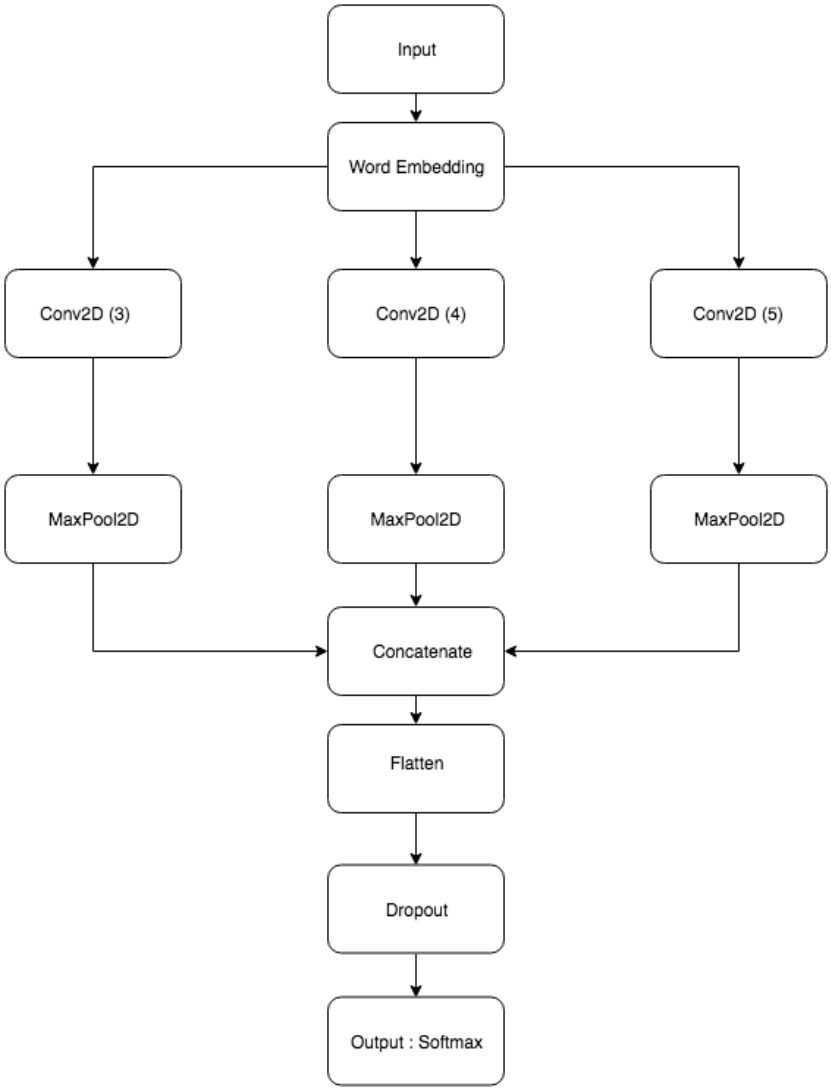

Figure 3.14: Architecture of the CNN

The input data needs to be preprocessed, we create an instance of a keras Tokenizer that takes as input the number of the most frequent word that have to be considered in the resulting dictionary, in our case 2000. This **dictionary** maps each word into an integer that correspond to a rank based on the frequency in which the word appears in the training set, the higher is the rank the more frequent the word is.

After this operation we use this vocabulary to transform each text in a **sequence of integers**, basically the tokenizer takes each word in the article and replaces it with with its corresponding rank value from the dictionary. Now each sequence is padded so that the length

is the same among all the comments, in particular we choose a length of 30; if a comment is longer it will be truncated otherwise it will be padded with zeroes.

The first layer of our CNN is dedicated to the **embeddings**, to build this layer we can use the **Glove** pretrained embeddings and give them as weights to the layer. Then we use three different **Convolutional** layers with different filter sizes; 3, 4 and 5, the second dimension of the filter matches the embedding length that is 100, this results in taking a window of words wide 3, 4 or 5 at a time. We can finally concatenate the results and use a **Dropout** layer to prevent the overfitting phenomenon. The last layer is a **Dense** layer that works as classifier.

Results with Class Weight Approach

In this part are shown the results obtained with this architecture. The results are really great. The *F1-Macro* is very high as well as the *LARP metric*. There is also only one label completely missed by the model that is Relief (23).

	precision	recall	f1-score	support
0	0.64	0.62	0.63	504
1	0.74	0.80	0.77	264
2	0.44	0.39	0.41	198
3	0.42	0.27	0.33	320
4	0.41	0.26	0.32	351
5	0.25	0.39	0.31	135
6	0.37	0.30	0.33	153
7	0.44	0.44	0.44	284
8	0.51	0.29	0.37	83
9	0.24	0.22	0.23	151
10	0.32	0.34	0.33	267
11	0.46	0.56	0.51	123
12	0.29	0.14	0.19	37
13	0.65	0.44	0.52	103
14	0.63	0.47	0.54	78
15	0.90	0.90	0.90	352
16	0.50	0.17	0.25	6
17	0.51	0.61	0.55	161
18	0.75	0.83	0.78	238
19	0.36	0.17	0.24	23
20	0.58	0.50	0.54	186
21	0.60	0.19	0.29	16
22	0.38	0.14	0.21	145
23	0.00	0.00	0.00	11
24	0.55	0.75	0.63	56
25	0.52	0.51	0.52	156
26	0.50	0.47	0.48	141
27	0.59	0.71	0.64	1787
micro avg	0.55	0.55	0.55	6329
macro avg	0.48	0.42	0.44	6329
weighted avg	0.54	0.55	0.54	6329
samples avg	0.55	0.55	0.55	6329

Figure 3.15: Report for all the emotions with Class Weight

The *training time* is 102s in 13/20 epochs.
The *inference time* on the test set is 0.82s.
The overall *LARP* is 68, 64%, the metrics for all the classes are reported in the image above.

The *F1 - macro* is and 0.44.

Results with Data Augmentation Approach

The CNN seems to react well to the usage of Data Augmentation, we can see how the augmented classes (12,16,19,21,23) increases, in particular considering F1-Macro: Embarrassment from 0.12 to 0.38, Grief from 0.25 to 0.4, Nervousness from 0.24 to 0.34, Pride from 0.29 to 0.40, Relief from 0 to 0.14.

	precision	recall	f1-score	support
0	0.61	0.66	0.64	504
1	0.74	0.80	0.77	264
2	0.54	0.35	0.43	198
3	0.42	0.26	0.32	320
4	0.43	0.26	0.32	351
5	0.56	0.13	0.22	135
6	0.51	0.18	0.27	153
7	0.49	0.32	0.38	284
8	0.60	0.33	0.42	83
9	0.27	0.16	0.20	151
10	0.34	0.13	0.19	267
11	0.61	0.28	0.38	123
12	0.42	0.35	0.38	37
13	0.70	0.31	0.43	103
14	0.57	0.44	0.49	78
15	0.89	0.90	0.89	352
16	0.33	0.50	0.40	6
17	0.56	0.54	0.55	161
18	0.78	0.81	0.80	238
19	0.33	0.35	0.34	23
20	0.55	0.53	0.54	186
21	0.56	0.31	0.40	16
22	0.46	0.12	0.20	145
23	0.12	0.18	0.14	11
24	0.53	0.68	0.59	56
25	0.62	0.42	0.50	156
26	0.54	0.43	0.47	141
27	0.53	0.87	0.66	1787
micro avg	0.56	0.56	0.56	6329
macro avg	0.52	0.41	0.44	6329
weighted avg	0.55	0.56	0.53	6329
samples avg	0.57	0.57	0.57	6329

Figure 3.16: Report for all the emotions with Data Augmentation

The *training time* is 116s in $13/20$ epochs.
The *inference time* on the test set is 0.81s.
The overall *LARP* is $69,47\%$, the metrics for all the classes are reported in the image above.
The *F1 - macro* is and 0.44.

Chapter 4

Evaluation Metrics

The metrics that we use in order to assess each model are:

- ***Label Ranking Average Precision Score (LARP)***: is the average over each ground truth label assigned to each sample, of the ratio of true vs. total labels with lower score. $LRAP(y, \hat{f}) = \frac{1}{n_{\text{gamples}}} \sum_{i=0}^{n_{\text{osmplo}}-1} \frac{1}{|yi|} \sum_{j:y_{jj}=1} \frac{|\mathcal{L}_{ij}|}{\text{rank}_{ij}}$
- ***Training Time***: this value shows how much time the model need to understand the characteristics of the dataset in order to carry out the task properly.
- ***Inference Time***: this value shows how much time the model needs to give a prediction on the category. Together with the *training time* they can be use to evaluate the trade-off of a given approach.
- ***F1-macro***: $\frac{\sum F1-scores}{numberofclasses}$, it gives each class equal importance. In order to obtain this value we have computed the binary label from the set of probabilities. We assign a number of labels to the sample equals to the number of labels It has in the ground truth, then we assigned a number of emotions with the highest probabilities using that number.

Chapter 5

Comparison

In this Chapter we propose a comparison between the two methods, in the case of multiple instances of the same approach we pick the best one, like in the case of BERT.

In Table 5.1 we can see that generally speaking the CNN approach is the best since it performs well in almost all the evaluation metrics, in particular it has outstanding results in term of *LARP* and *F1-Macro*.

Model	LARP (%)	F1 - Macro	Inference Time (s)	Training Time (s)
smallBERT(CW)	59.86	0.27	14.77	1287
CNN(DA)	**69.47**	**0.44**	**0.81**	**116**

Table 5.1: Comparison of evaluation metrics among all the approaches

Going deep into the values of this table, we can say that CNN training is very fast, as well as the inference time, this is due to the nature of the CNN that requires less time to be trained since CNN has 4M parameters with respect to the 29M of BERT. Moreover in CNN we set the Embedding Layer that has pretrained values (it uses Glove embedding) as not trainable, whereas in BERT we train all parameters.
Another interesting fact is the F1-score values that in CNN are all different from 0, instead with smallBERT emotions with low number of samples are often ignored. A possible hypothesis could be that smallBERT overfits on the training set, being almost 10 times bigger than CNN, since the number of samples labeled as neutral is quite high.
Out hypothesis could be confirmed by looking at the other BERT architectures, for instance BERTexpert and BERTbase, both with over 100M parameters, have the highest recall score in the neutral emotion and ignores the least present emotions in the dataset.

Conclusion

In conclusion we observe how a multi-label classification problem can be approached, using different architectures. The results in BERT are unexpectedly disappointing. The CNN instead behaves very well since an F1-Macro of 44% can be considered a very good result, comparing it to the sota result presented by HuggingFace related to this Dataset that reached an F1-Macro of 49% using *EmoRoBERTa*.

Bibliography

[1] https://huggingface.co/datasets/go_emotions

[2] https://unicode.org/emoji/charts/full-emoji-list.html

[3] https://github.com/jasonwei20/eda_nlp

[4] https://www.kaggle.com/code/patricia92fa/explaining-cnns-for-text-classification-using-shap/notebook