Text Emotion Recognition with BERT

Final Project for Human Language Technology

Matteo De Francesco **Chenxiang Zhang** **Gerlando Gramaglia**
Department of Computer Science
University of Pisa
{m.defrancesco4, c.zhang4, g.gramaglia1}@studenti.unipi.it

Abstract

Sentiment analysis is a must topic in Natural Language Processing. We present a baseline BERT model for the sentiment analysis over the GoEmotions dataset, comparing our results with those obtained in the original paper. Our results, with the same settings, show the same outcome of the original ones. Driven by this, we perform experiment over derived BERT models to assess the performance of them with respect to the original one. We show that a simplified model and a complex one achieve better result than the standard one.

1 Introduction

Correctly detecting emotional expressions is an essential task to achieve a better understanding of the complex human social interaction. In the last decade, autonomous system for text elaboration has made incredible advancements thanks to the Deep Learning field Krizhevsky et al. [2012]. Specifically in the last few years, a new state-of-the-art text model BERT Devlin et al. [2018] has been introduced and has since dominated the entire field of NLP as the best method for almost any text-related tasks. In this work we are going to study and build a system to automatically classify a text sentence's emotion. We assess the results by comparing various models from the wide family of BERT models. The dataset used for the experiments is the GoEmotions dataset Demszky et al. [2020], which is the largest English dataset, carefully human labeled with 27 fine-grained categories as target emotion.

2 Background

2.1 Attention

Mostly of modern state-of-the-art language models are characterized by deep layers of Transformers implementing the concept of attention Vaswani et al. [2017].
Recalling the definition of attention, we have: *"Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query"*
The weighted sum is basically a selective summary of the information contained in the values, and the query determines which values need more "focus". In this way, we obtain a fixed-size representation of an arbitrary set of representations (the values) dependent on the query.
Given some values $v_1, v_2, \ldots, v_N \in \mathbb{R}^{d_1}$, some keys $k_1, k_2, \ldots, k_N \in \mathbb{R}^{d_1}$ and a set of queries $q_1, q_2, \ldots, q_N \in \mathbb{R}^{d_1}$, computing the attentions for a given query $s \in \{q_1, q_2, \ldots, q_N\}$ is given by:‘

1. Compute the attention scores $e \in \mathbb{R}^N$

$$e_i = s^\intercal \cdot k_i$$

2. Use softmax to get an attention distribution α

$$\alpha = \text{softmax}(e) \in \mathbb{R}^{d_1}$$

3. Use attention distribution to take weighted sum of values

$$a = \sum_{i=1}^{N} \alpha_i \cdot v_i \in \mathbb{R}^{d_1}$$

In transformers architecture, we use the **self-attention** mechanism, where keys, queries and values are drawn from the same source. The steps above can be rewritten accordingly:

1. Compute key-query affinity

$$e_{ij} = q_i^\intercal \cdot k_j$$

2. Compute attention weights from affinities

$$\alpha_{ij} = \frac{\exp^{e_{ij}}}{\sum_j \exp^{e_{ij}}}$$

3. Compute outputs as the weighted sum of values

$$output_i = \sum_j \alpha_{ij} v_j$$

Attention layers have been chosen as an alternative to RNN due to their parallelizable nature. However, their nature brings in also some problems

- They do not have an inherent notion of order $\implies$ Solution: add position representations to the inputs
- They present nonlinearities in their standard form, just weighted averages $\implies$ Solution: apply the same feedforward network to each self-attention output
- Need to ensure that we don´t look at future words when predicting a sequence $\implies$ Solution: easy mask out the future by setting attention weights to 0

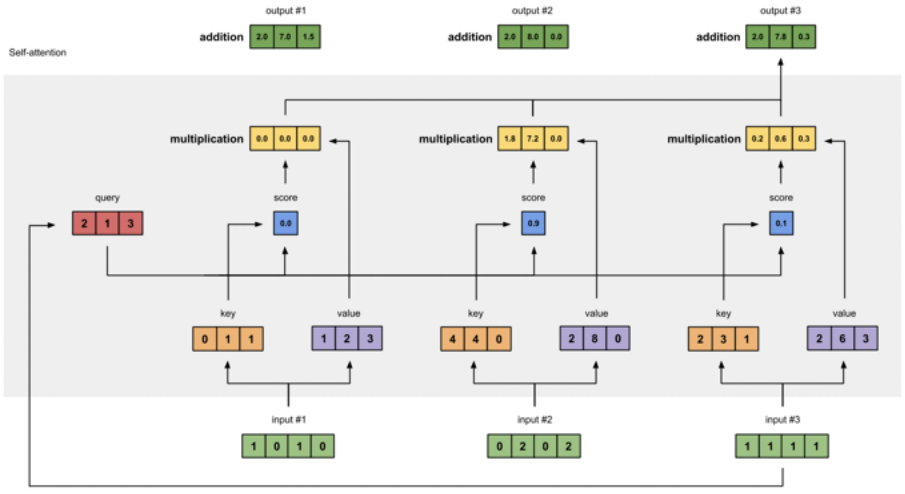

Figure 1: Self-attention layer

2.2 Transfomers

A transformer is an auto-regressive model consisting in two basic parts:

- An encoder, which maps an input sequence of symbol representations $(x_1, x_2, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, z_2, \ldots, z_n)$
- Given z, the decoder generates an output sequence $(y_1, \ldots, y_m)$ of symbols one element at time

It consumes the previously generated symbols as additional input when generating the next.

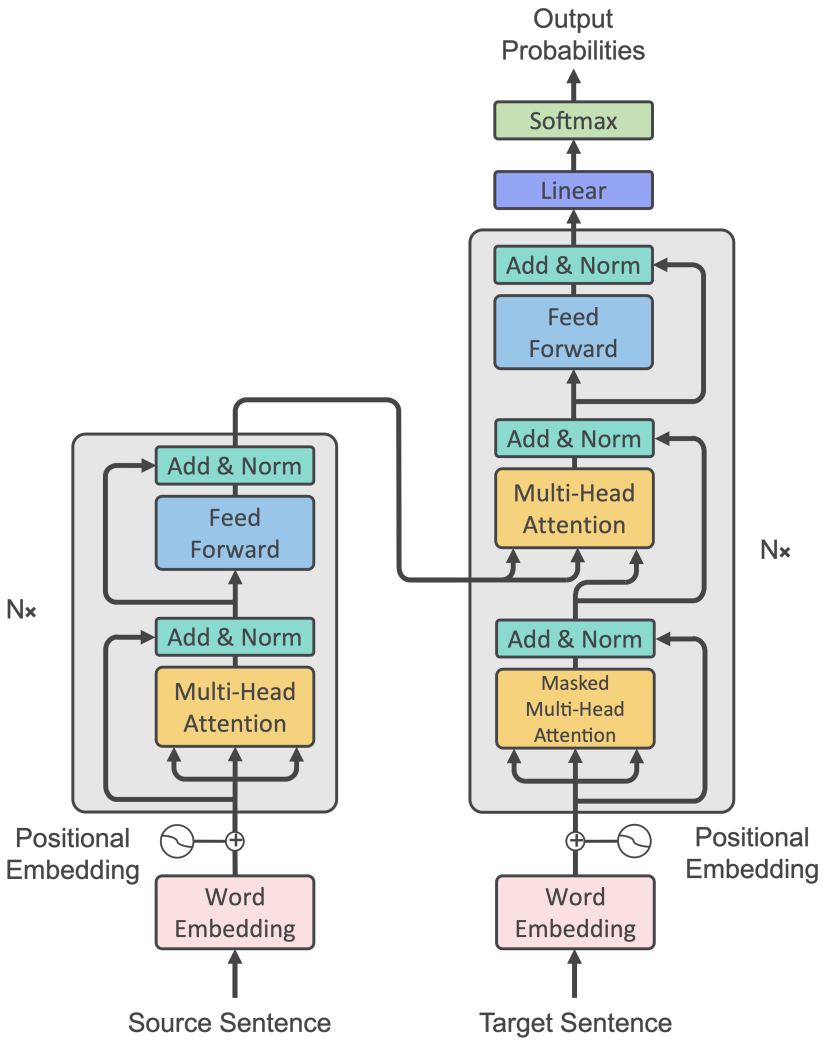

Figure 2: The original Transformer architecture

The basic transformer has a stack of encoders and a stack of decoders. The encoder consists of two layers: a self-attention layer and several feed-forward neural network. The decoder has both those layers too, but between them there is an attention layer that helps the decoder focus on relevant parts of the input sentence.
Each input word is turned into a word embedding vector. Those vectors flow through each of the two layers of the encoder. Whilst there are dependencies between paths in the self-attention layer, in contrast the feed-forward layers are independent and thus the various paths can be executed in

parallel.

As the model processes each word, self attention allows it to look at other words in the input sequence for clues that can help lead to a better encoding for this word.
For each word, we create a query vector, a key vector and a value vector. These are created by multiplying the embedding by three matrices that were trained during the training phase (hence the parameters of our model)

$$q_i = W^Q X_i \qquad k_i = W^K X_i \qquad v_i = W^V X_i$$

Then we perform the computation used to score the self-attention:

1. Calculate the score:
$$E = QK^T$$
2. Apply softmax
$$S = \text{softmax}(E)$$
3. Compute the weighted sum
$$Z = SV$$

In one step

$$Z = \text{softmax}(QK^T)V$$

We can notice how having only one self-attention, a word has only one way to interact with others. Instead of using only one self-attention layer, we can use several self-attention layers and then concatenate all the attention heads.

3 Dataset

The dataset used for the experiments is GoEmotions Demszky et al. [2020]. It is the largest human annotated dataset currently available with multiple annotations per example in order to have a consistent label quality. The dataset is well curated and consists of 58k comments extracted from Reddit's comments section and labeled by hand by humans with a choice among 27 different categories plus a Neutral label. The result of this work is a multilabel dataset where each sentence is classified in multiple categories as see in Table 1.

Sample Text	**Labels**
OMG, yep!!! That is the final answer. Thank you so much!	gratitude, approval
I'm not even sure what it is, why do people hate it	confusion
Guilty of doing this tbph	remorse
This caught me off guard for real. I'm actually off my bed laughing	surprise, amusement
I tried to send this to a friend but [NAME] knocked it away	disappointment

Table 1: Example samples from the GoEmotions dataset

3.1 Taxonomy

There are three different taxonomies that can be used to divide the classification of the expression in GoEmotions. Each taxonomy has a different level of emotion-granularity.

- *Original taxonomy*: the most granular taxonomy with 28 different classes;
- *Ekman taxonomy*: it groups together a portion of the categories producing 6 different macro-categories: anger, disgust, fear, joy, sadness and surprise;
- *Group taxonomy*: it groups together a portion of the categories producing 4 different macro-categories: positive, negative, ambiguous and neutral.

Using these different taxonomies we can choose the level of granularity we want to achieve for our automatic system.

4 Model Architectures

We experiment with various different models from the BERT family. For each model we extend each baseline model with different final layers to build a multi-label classifier based on the target chosen taxonomy. For the purpose of regularization, a Dropout layer is also added to the final layers. The architecture of different classifiers used can be observed in the Figure 3. Each ⟨*model_name*⟩**Layer** consists in a combination of an attention, a linear and an output layer. These are replicated 12 times in parallel in each ⟨*model_name*⟩**Encoder**. We give a brief summary highlighting the main points for each model we used:

- BERT Devlin et al. [2018]: the first original multi-layer bidirectional Transformer encoder. BERT uses positional encoding, self-attention layers, multiple heads. BERT is trained with two different objectives: masked language modeling (MLM) and next sentence prediction (NSP).
- DistilBERT Sanh et al. [2019]: this is a distilled version of the original BERT model. It preserves well the performance of the origin model while being smaller (40%) and thus running also faster (60%).
- RoBERTa Liu et al. [2019]: similar to the original BERT model but with a slightly different training procedure: different hyperparameters, and using only the masked language modeling (MLM) objective. It achieved better performance compared to BERT on different benchmarks.
- BigBird Zaheer et al. [2020]: it extends BERT to much longer sequences by applying sparse, global, and random attention which approximates full attention while being computationally efficient.

Additionally, we have to point out the following details in our extended architectures Figure 3 to handle the multi-label target:

- In the **RoBERTa** model, after the first **Pooler**, we denoted with a retroaction arrow that the output is passed again through the same layers, and, after the second flow through the **RobertaEncoder**, we get the output to the *Multilabel layers*.
- In the **DistilBERT** model we have 6 standard `Transformer` layers in parallel;
- Always in the **DistilBERT** model, the output of the last `Transformer` layer is given to a **MaxPooling** layer implemented by us, since the simplified nature of **DistilBERT** leave the pooling operation to the user.

Every model makes an extensive usage of attention layers, which is the core mechanism of any BERT family model. Each model is paired its corresponding Tokenizer and every input sentence is also modified by adding an initial `[CLS]` token and an ending `[SEP]` token to specify the starting and ending of a phrase, which is a process needed for all the BERT models. Subsequently, the input

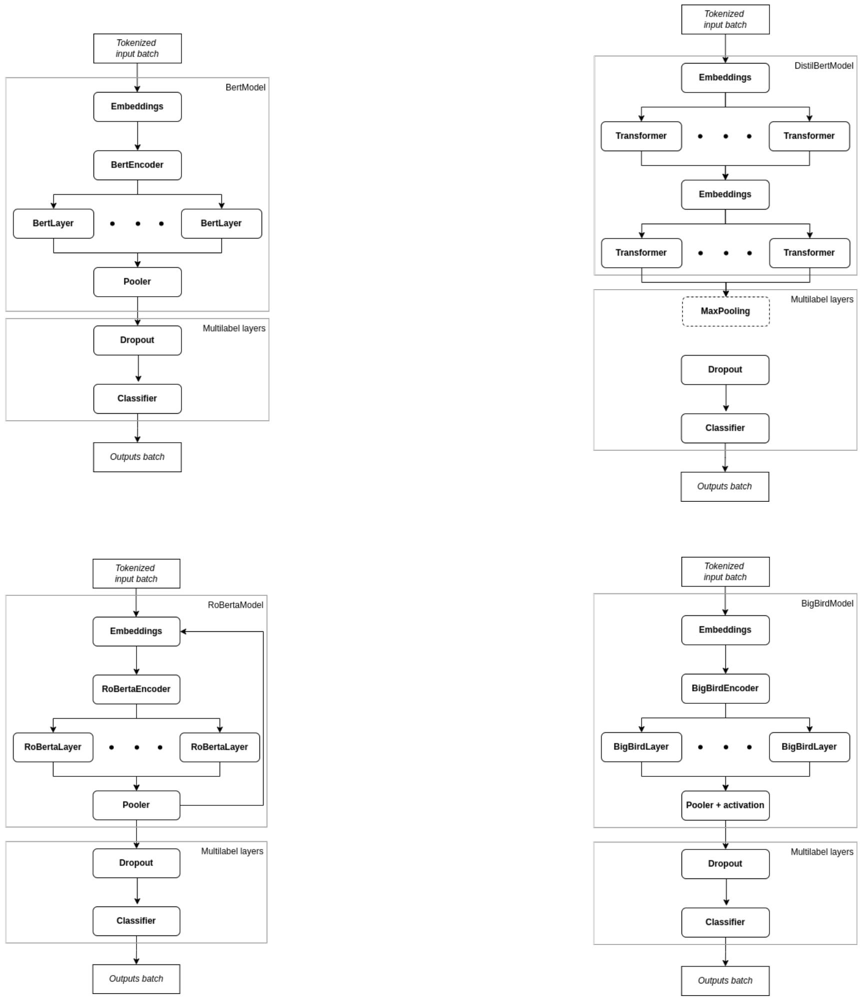

Figure 3: BERT family architectures adapted for multi-label classification

sentences are divided into the different batches and the *Tokenizer* conversion happens: given a batch of sentences as input, the Tokenizer produces 4 different output elements which are all used as input for the BERT model:

- `input_ids`: integers representation of the words in a sentence. We set a maximum length for each sequence and we pad the remaining;
- `attention_mask`: same length as the `input_ids`, each element is set to 1 when there is a corresponding input word in the `input_ids`, it is set to 0 for the padding portion;
- `token_type_ids`: used mainly in question answering systems, set to 1 for words belonging to the question, 0 for words in the answer. In our case, being a sentence classification task, we set everything to 0;
- `label`: the true label of the sentence, set to 1 for corresponding categories, 0 otherwise.

```
sentence: We need more boards and to create a bit more space for [NAME]. Then we’ll be good.
tokens: We need more boards and to create a bit more space for [NAME] . Then we ’ ll be good .
input_ids: 101 1284 1444 1167 8190 1105 1106 2561 170 2113 1167 2000 1111 1 119 1599 1195 787 1325 1129 1363 119 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
attention_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
token_type_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
label: 0 0 0 1
```

Figure 4: Example of Tokenization

As we can see from the example in the Figure 4, the `input_ids` contains the different integer representation of the tokens list. The `attention_mask` is set to 1 until the padding sequence starts, and the `token_type_ids` are all zeros. In this case we used the *group* taxonomy hence the resulting labels range is 4 (4 macro categories). We can notice also the use of special tokens: the original sentence contained the name of a person which has been replaced by the special token `[NAME]`. Different special tokens are included in the `BERT` vocabulary and used to tokenize sentences in this dataset.

5 Experiments and Results

5.1 Implementation Details

The experiments uses library Pytorch [1] together with the pretrained models from the HuggingFace [2]. For each baseline model, we extended it by adding the additional layers seen in the previous figure, and redefining the *forward()* function accordingly. We keep an universal *api* (such as same input parameters and output parameters for all functions) in order to reuse already written code.

Dataset. The creation of the dataset was one of the heaviest part. We read the input files containing the data, set up the different labels and also the possibility to save this data for the next tries. Looping through the scanned data files, we create different objects, first storing the information about text, label and an identifier (`InputExample` class), then we convert this set of objects using the *Tokenizer* into the set of elements needed by us: the `input_ids`, `attention_mask`, `token_type_ids` and `labels` (`InputFeature` class).

Finetuning. For all the models we load the pretrained models from the HuggingFace library and finetune them. We define the training and validation functions that execute the training / validation loop, which are controlled by different hyperparameters:

- set of integer values to define checkpoints for saving the actual model and execute validation phase;
- hyperparameters for controlling the learning rate, the gradient accumulation and the step to compute.

At the end of the finetuning process, we perform a testing phase to validate the final result. All in all, we can represent the workflow of the training/testing with the "narrow" algorithm 1.

Algorithm 1 GoEmotions

```
function GoEmotions(TAXONOMY, model_type)
    D = CreateDatasets(TAXONOMY)
    model = CreateModel(model_type, TAXONOMY)
    Set c_step, v_step, epochs                          ▷ Set checkpoints/validation steps
    for epoch ← 1 to epochs do
        loss, checkpoints = Train(model, D, c_step, v_step)   ▷ Calls validation inside
    end for
    model = CreateModel.from_last_checkpoint(checkpoint)      ▷ Recover model
    loss = Test(model, D)
end function
```

[1] https://pytorch.org/
[2] https://huggingface.co/

Parameter Settings. When finetuning the pre-trained BERT models, we use the same setup as in Demszky et al. [2020], where they keep most of the hyperparameters untouched from the original BERT Devlin et al. [2018] and modify only the batch size to 16, learning rate to 5e-5. The pretrained model is finetuned for a total of 5 epochs. We use the same setting for all the BERT architectures and for all the different taxonomies in the experiments.

Evaluation. The models performance are evaluated by using the results obtained from the test set. In particular, we compute different metrics for each model. We record the obtained loss, the accuracy computed and the F1-macro metric to combine precision and recall in a single shot.

5.2 Experiments

Firstly we replicate the experiment reported in the Demszky et al. [2020] using the **BERT** model to create a baseline. Then, we compare the baseline BERT with other language models from the same BERT family trained over the aforementioned corpus, and the same parameters. In order to have a fair comparison between different models we use the same evaluation metrics among all the models. All the experiments are perfomed using Google Colab and Kaggle's free GPU.

After the creation of all the BERT models, **RoBERTa** Liu et al. [2019], **DistilBERT** Sanh et al. [2019] and **Big Bird** Zaheer et al. [2020], we perform and average the results from three independent runs with different seed for each model and for each taxonomy group. Due to the stochastic nature of neural networks, this technique called "seed averaging", will allow to obtain a more fair results used to compare the different architectures.

We were able to perform around 4 experiments per day (due to `GPU` limit usage in `Google Colab` environment), but nevertheless this limitation we took advantage of the easy synchronization with the `Google Drive` to save the model, results and the different metrics computed. We performed also experiments in the `Kaggle` environment due to the advantage in speed, collecting the single results at the end and transferring them on the `Drive`.

After the first executions to debug the program and testing that everything worked, we performed a total of 36 final different experiments. We tested every BERT model, three different times per model changing the TAXONOMY: *original*, *group* and *ekman*. Then, we performed the same experiments again changing the starting seed of the models to collect the metrics for different seeds. In particular, we tried as seeds $42, 43, 44$.

In addition, we tried also the **XLM** language model. But due to the big complexity of it, `Colab` machines were not able to support it resulting in a crash, even after decreasing the `batch_size`.

We can remark the difference between the GPU used by `Colab` and those used by `Kaggle`. The first one deploy the NVIDIA Tesla K80, while the latter deploy the NVIDIA Tesla P100. This difference in GPUs results in a big difference of timings between the two platforms, coupled with the complexity of the models (`#parameters`) 2.

5.3 Results

From the experiments result reported in the Table 3, we can observe that all the models obtain a higher results on taxonomies with fewer target classes, *group* and *ekman*. This is because with fewer class, the task is generally easier than the *original* taxonomy with 27 classes. Moreover, we can see that there is no best model that outperforms other models on all the taxonomy. However it's clear that some model are better than others. Here we give some comments for each model's performance.

Taxonomy	Model	Tesla K80 (sec.)	Tesla P100 (sec.)	#params
Original	**BERT**	5670	1921	108.3 M
	BigBird	5815	2074	127.5 M
	DistilBERT	3245	1080	130.4 M
	RoBERTa	4610	1984	249.3 M
Ekman	**BERT**	5385	1943	108.3 M
	BigBird	5437	2094	127.5 M
	DistilBERT	2965	1135	130.4 M
	RoBERTa	3214	1973	249.3 M
Group	**BERT**	5205	1953	108.3 M
	BigBird	5461	2087	127.5 M
	DistilBERT	2710	1075	130.4 M
	RoBERTa	2595	1991	249.3 M

Table 2: Time elapsed for the finetuning of 5 epochs.

Taxonomy	Model	Loss	Accuracy	F1-macro
Original	BERT	0.10 ± 0.001	$\mathbf{0.43} \pm 0.005$	0.49 ± 0.005
	BigBird	0.11 ± 0.001	0.40 ± 0.007	0.36 ± 0.02
	DistilBERT	0.10 ± 0.009	0.40 ± 0.009	$\mathbf{0.52} \pm 0.01$
	RoBERTa	$\mathbf{0.09} \pm 0.003$	0.42 ± 0.01	0.51 ± 0.007
Group	BERT	0.60 ± 0.02	0.58 ± 0.003	0.67 ± 0.003
	BigBird	0.46 ± 0.009	0.52 ± 0.003	0.63 ± 0.003
	DistilBERT	0.53 ± 0.09	0.57 ± 0.002	0.67 ± 0.008
	RoBERTa	$\mathbf{0.41} \pm 0.04$	$\mathbf{0.59} \pm 0.007$	$\mathbf{0.69} \pm 0.006$
Ekman	BERT	0.36 ± 0.007	0.55 ± 0.003	0.61 ± 0.003
	BigBird	0.29 ± 0.006	0.50 ± 0.003	0.55 ± 0.009
	DistilBERT	0.32 ± 0.05	0.54 ± 0.008	0.61 ± 0.009
	RoBERTa	$\mathbf{0.26} \pm 0.02$	$\mathbf{0.56} \pm 0.003$	$\mathbf{0.63} \pm 0.008$

Table 3: BERT model results $(mean \pm std)$ on the test set averaged over three different seeds.

BERT With this baseline model we obtain a `macro F1-score` on the *original* taxonomy of 0.49 ± 0.005, which is a similar results as reported by Demszky et al. [2020] but with a better standard deviation, indicating a more stable finetuning process.

BigBird Since this model's architecture is specifically tuned to tackle very long sequences, it performs pretty poorly on this dataset composed by short sentences.

DistilBERT Interestingly, we can observe the unexpected results obtained by the smaller BERT model DistilBERT. Being a simpler and less complex model, we expected it to have a slightly worse performances than the baseline. Surprisingly, it retained almost always the same performance as the baseline and even obtaining the best F1-macro on the *original* taxonomy among all the BERT models.

RoBERTa This BERT model seems to perform slightly better than all the other models on the *group* and *ekman* taxonomy, and improving also the baseline BERT F1-macro on the *original* taxonomy. Overall, RoBERTa can be declared as the model with the best and consistent results among different taxonomies. This result was expected given that RoBERTa is basically the better trained version of the baseline BERT.

An important factor we had to pay attention during the training is the overfitting of the models. Being these very complex model, it is quite difficult to stabilize training and adopt heuristics to know when to stop. Also, we performed training only for 5 epochs, so even an `Early Stopping` technique with a certain degree of tolerance would have not stopped before the ending. The overfitting can be observed during the training phase of the *group* and *ekman* taxonomies for all the models in the Figure 6, 7.

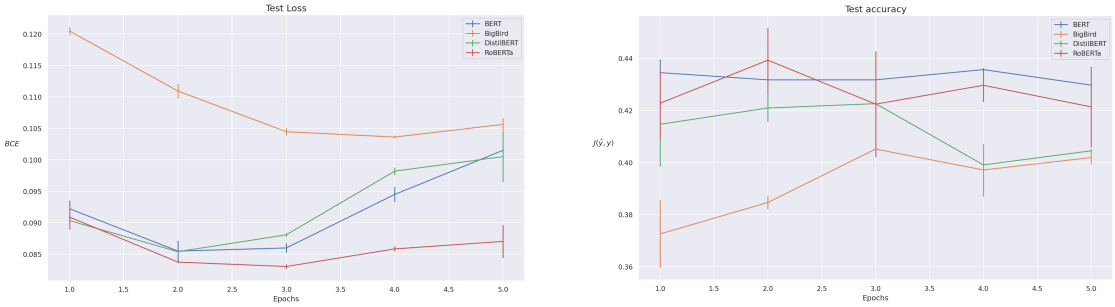

Figure 5: (left) Test loss on original taxonomy (right) Test accuracy on original taxonomy

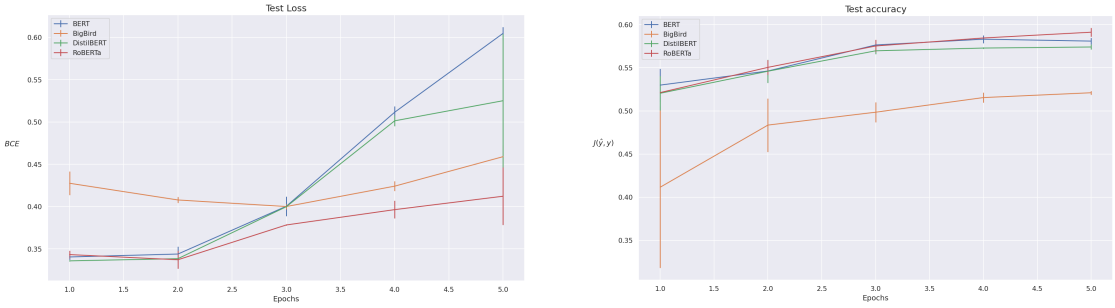

Figure 6: (left) Test loss on group taxonomy (right) Test accuracy on group taxonomy

Moreover we can see how the test loss of the models tend to increase by small portion. We can conclude that stopping the training after 5 epochs can be a good heuristic choice for the number of epochs hyperparameter, in order to avoid overfitting. To prevent the overfitting, we could have added also more layers of dropout inside each pretrained model.

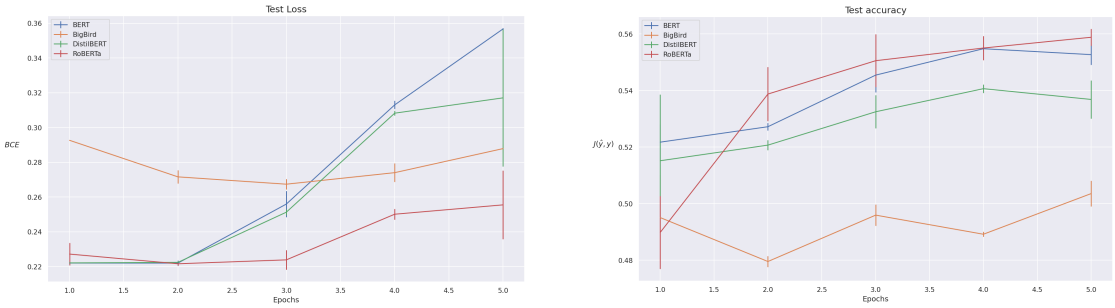

Figure 7: (left) Test loss on ekman taxonomy (right) Test accuracy on ekman taxonomy

6 Conclusion

In this work we have used different state-of-the-art deep neural models from the BERT family to create an autonomous text emotion recognition system. We have conducted various experiments to compare the different architectures using the biggest human labeled dataset GoEmotions Demszky et al. [2020]. We found out that the best BERT model is RoBERTa Liu et al. [2019], which consistently outperforms all the other models on different taxonomies. The limitation in this study is mainly due to the time and computational constraints. Some possible future work can explore better finetuning strategies, gather more data by combining multiple existing public dataset or modifying the BERT architecture to better adapt for short sentences for the classification task.

References

Dorottya Demszky, Dana Movshovitz-Attias, Jeongwoo Ko, Alan Cowen, Gaurav Nemade, and Sujith Ravi. GoEmotions: A Dataset of Fine-Grained Emotions. In *58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL `https://arxiv.org/abs/1706.03762`.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33, 2020.