UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

DIPLOMA PROJECT

Offensive Language Detection

Gabriel-Răzvan Busuioc

Thesis advisors:

Prof. dr. ing. Mihai Dascălu
Drd. ing. Andrei Paraschiv

BUCHAREST
2022

UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE

PROIECT DE DIPLOMĂ

Identificarea Limbajului Ofensator

Gabriel-Răzvan Busuioc

Coordonatori științifici:

Prof. dr. ing. Mihai Dascălu
Drd. ing. Andrei Paraschiv

BUCUREȘTI
2022

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SINOPSIS

În ultimul deceniu, popularitatea platformelor de social-media a crescut în rândul oamenilor de pe tot globul, unii dintre ei profitând de această oportunitate pentru a prolifera limbaj ofensator și instigator la ura. Deși unele platforme folosesc metode pentru filtrarea textului, acelea care aleg să nu filtreze sunt exploatate de utilizatorii ce tind să folosească limbaj ofensator și abuziv. Această lucrare prezintă crearea și adnotarea corpusului FB-RO-Offense, un set de date ce conține 8 mii de comentarii generate de utilizatori în cadrul transmisiilor live de pe Facebook. Munca noastră a implicat experimente cu acest set de date, dar și cu alt corpus în limba română pentru detecția limbajului ofensator, RO-Offense. Pentru clasificarea ce implica 4 clase, rezultatele obținute prezinta un F1-score de 0.83, iar pentru clasificarea binara un F1-score de 0.90.

ABSTRACT

In the past decade, social media platforms have gained a lot of popularity amongst people all around the globe, some of them seizing this opportunity in order to proliferate offensive language and hate speech. Although some platforms use text filtering techniques, those that choose not to do this are being exploited by users that tend to use offensive and abusive language. This work presents the creation and annotation of the FB-RO-Offense corpus, an offensive speech dataset containing 8 thousand user-generated comments from Facebook live broadcasts. Our work involved experimenting with the FB-RO-Offense dataset as well as with another Romanian language dataset for offensive language, RO-Offense. The results show a F1-score of 0.83 for a four-way classification and a F1-score of 0.90 for a binary classification.

1 INTRODUCTION

In this part of our thesis we briefly describe our objectives alongside proposed solutions in order to help our readers acquaint the thesis subject. We also touch bases on the context of our thesis and the problems that our experiments could potentially solve. In the end we present a brief overview of the thesis structure.

1.1 Context

According to Perrin (2015), the first 5 years of the past decade showed a massive popularity growth of social networks all around the globe. Moreover, Islm et al. (2021) presents a big increase in use of social media platforms during the COVID-19 pandemic. At a first glance, the usage growth of these platforms might represent a beneficial thing as it proves the fact that social networks goal is being achieved: connecting people around the globe. On the other hand, such an explosion of users might prove tough to handle and many unwanted situations might occur both on and off these social media platforms.

1.2 Problem

As stated in Section 1.1, the growth of social media platforms, forums and any other kind of human interaction on the internet comes at a certain price. In the case presented by our thesis, this price is represented by offensive speech pollution of the online platforms. This phenomenon has grown especially on platforms that do not use filtering techniques for such language. Therefore, the problem of offensive speech on social media platforms has become worrying and represents a big concern not only for the developers of such platforms but also for the overall well being and mental health of people around the globe.

1.3 Objectives

Recent researches have came up with feasible and performing solutions in the fields of Natural Language Processing. Technologies such as BERT(Devlin et al., 2019) proved to be a good base for building programs that are able to detect and classify certain forms of language.

One of our objectives is to build a reliable set of data for offensive speech detection that can be later on extended and used in other classification problems as well. We aspire to use this corpus and other similar ones in order to train and experiment with multiple kinds of neural architectures.

1.4 Proposed Solution

In terms of dataset, we propose a Romanian language corpus that consists of 8 thousand comments extracted from Facebook live broadcasts of Romanian political figures. The data was extracted in the months of February and March 2022 during the first phases of the Ukraine invasion. The comments were manually annotated by 1 person following a hierarchical labeling schema. An inter-rater agreement of 91% was obtained for the proposed dataset.

In terms of offensive language classification models, we propose a SVM method and 2 BERT based architectures. The 2 neural architectures consist of a BERT layer followed by a Feed Forward layer, respectively a Convolutional Neural Network layer. Using another corpus that contains the same labeling scheme as the previous presented one, we also propose a transfer learning approach.

1.5 Thesis Structure

In Chapter 2 of our thesis, we present 3 state of the art datasets used for offensive speech detection. Before diving into neural architectures used in Machine Learning, linear methods for classification are also discussed. The last part of the chapter presents the structure and importance of Transformer-based architectures and their real-world application.

Chapter 3 describes the corpora used for our experiments as well as an in-depth look at our workflow and experiments with neural architectures.

Further, Chapter 4 presents the best results obtained in our experiments. Discussions based on these results, as well as an error analysis and a real-world experiment on one of our models are presented in Chapter 5.

Conclusions, future work and improvements are briefly tackled in Chapter 6 of our thesis.

2 STATE OF THE ART

In this section we present methods, architectures and frameworks underlying the field of Machine Learning. The emphasis is on the methods that are currently used in some of the cutting-edge products pertaining to the domain of Natural Language Processing.

2.1 Datasets and Methods for Offensive Speech

2.1.1 Hate Speech and Offensive Language

Hate Speech and Offensive Language (Davidson et al., 2017) is a dataset for hate speech detection. The dataset was computed by cross referencing a lexicon of hate speech phrases compiled by *Hatesbase.org* [1] with data gathered from Twitter. Twitter is known for its light way handling of offensive tweets, thus being a very good and productive source for these kinds of datasets. Twitter API is also more permissive than other social media APIs, allowing developers to easily retrieve data and metrics in order to use them for research purposes.

Applying the above mentioned technique, the authors of HSOL managed to gather 85.4 million tweets from 33.458 distinct twitter users. From this corpus, they extracted a random sample of 25 thousand tweets that was later on manually annotated by *CrowdFlower*[2] workers. The tweets were labeled as one of the following three categories:

- Hate speech.
- Offensive language but not hate speech.
- Neither of them.

In order to obtain a reliable dataset, inter-coder agreement was enforced by having any tweet labeled by a minimum of 3 *CrowdFlower* workers, some isolated cases being labeled by as many as 9 workers. The category of one tweet was decided based on the majority decision. Some tweets were not labeled because of a perfectly split decision, thus not existing a majority to decide a category for those particular tweets.

The distribution of the HSOL corpus is unbalanced, the majority of the tweets being labeld as Offensive.

[1] `https://hatebase.org/`
[2] `https://appen.com/`

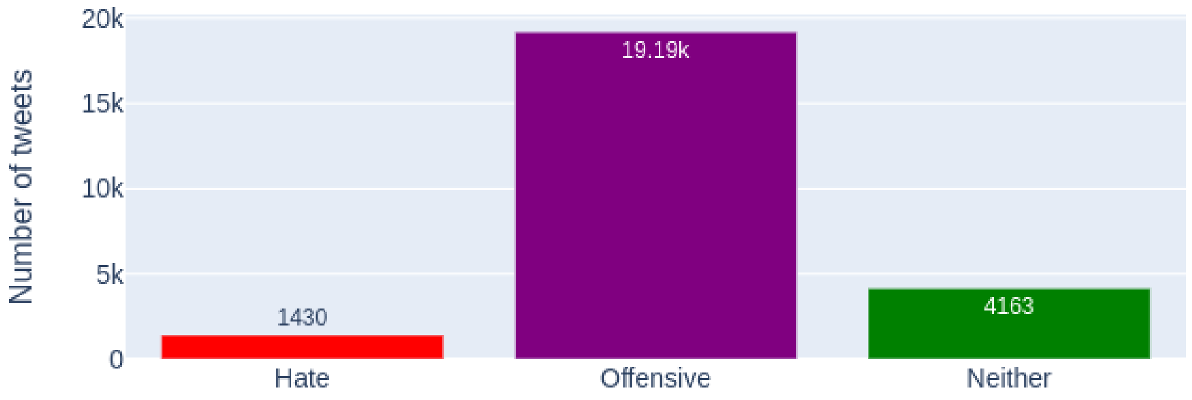


Figure 1: Tweets distribution in HSOL Dataset

More than 75% of the entire dataset is represented by offensive tweets, while the ones containing hate speech make up only 5% of the corpus.

2.1.2 Hate Speech from a White Supremacist Forum

Hate Speech Dataset from a White Supremacy Forum (de Gibert et al., 2018) is a dataset of textual hate speech. The dataset is made up of approximately 10.5 thousand sentences from *Stormfront*[3], a racial hate forum.The extracting process was carried out using web-scraping techniques, the data being arranged into sub-forums, conversation threads and posts. After extracting the data, posts were randomly selected from 22 sub-forums covering various topics. An automatic language detector was used in order to skip the non English textual data. The next step involved breaking the posts into sentences allowing to work with the minimum unit of hate speech and reducing the noise from other sentences.The annotation process was carried out manually, dividing the dataset in batches of approximately 1000 sentences. According to the authors of the dataset, this decision was taken in order to better control the process of labeling. For experiments that were carried out with this dataset, sentences shorter than 3 words or longer than 50 words were discarded in order to preserve the corpus as compact as possible. For this corpus, a 4 class labeling scheme was chosen. The sentences were labeled as one of the following 4 categories:

- Hate.
- NoHate.
- Relation.
- Skip.

According to Davidson et al. (2017), sentences labeled as *hate* should contain any form of hate speech. The sentences that do not fall under the previously mentioned category should

[3]`https://www.stormfront.org/forum/`

be labeled as *noHate*. The sentences labeled as *relation* are usually consecutive posts (of the same user) that do not contain hate speech if interpreted isolated, but their combination succeeds in spreading hate speech. *Skip* label is used for sentences that are not written in English or that do not contain relevant information in order to be labeled as *hate* or *noHate*

This corpus is yet another one that contains unbalanced data but, in contrast with the previously presented one, it is mostly made up of sentences that do not contain hate speech.

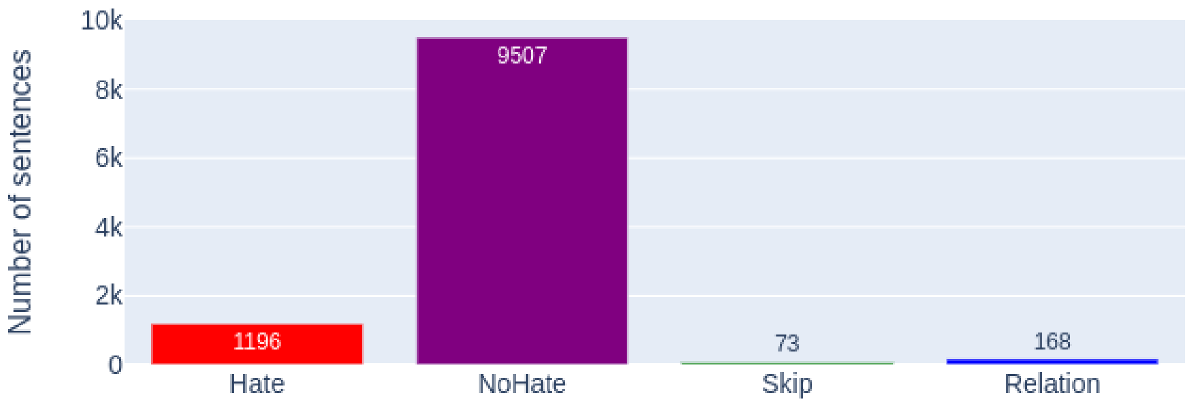


Figure 2: Tweets distribution in HSWS Dataset

In this corpus, 86% sentences are categorized as *noHate* while the rest of 14% is mostly composed of sentences labeled as *hate*.

2.1.3 Offensive Language Identification Dataset

Offensive Language Identification Dataset (Zampieri et al., 2019) or OLID is a dataset used for identifying not only the offensive language but also the target and type of offensive language in social media texts. OLID was computed by extracting data from Twitter, the final corpus containing 14.1 thousand tweets divided into a batch for training that contains 13.240 tweets and a batch for testing that contains 860 tweets. The labeling scheme for this dataset is a hierarchical one based on 3 levels.

Task A	OFF				NOT
Task B	TIN			UNT	NULL
Task C	IND	GRP	OTH	NULL	

Figure 3: Annotation scheme for OLID

The first level involves dividing the tweets in 2 categories: offensive and not-offensive. The second level aims at further dividing the offensive tweets in 2 categories: targeted and untargeted. The third level aims at dividing the offensive targeted tweets in 3 categories: individual, group, other.

The relationship between the 3 presented levels is a hierarchical one. An offensive tweet can be targeted or not targeted. If an offensive tweet is targeted, it could be targeted towards an individual, group or other entity.

2.2 Linear Methods For Text Classification

2.2.1 Naive Bayes

Naive Bayes is a simple method for assigning labels to instances that are usually represented by a list of features. The interesting part of this method is that it assumes that every feature of one instance is independent of the other features belonging to the same instance. In a real life problem, that indeed might not be the wisest choice as features could be strongly connected, but this method represents a good starting point for a baseline or proof-of-concept model. The previous presented assumption that this classifier is based on represents the origin of the method‘s name: *Naive* Bayes.

Therefore, Naive Bayes classifier is reduced to a simple conditional probability problem. Considering that a set of features $X = [x_1, x_2, .., x_n]$ should be labeled as Y (where Y can take a value from the following list $[y_1, y_2, ...y_k]$), this classifier computes the following probability for every y_i element from the previously mentioned list:

$$P(y_i|x_1, x_2, ..., x_n) \tag{1}$$

For a small number of features, this presented technique can be feasible, but there is a problem when the instances have a large number of features. When this is the case, Bayes‘ theorem can help to decompose the formula:

$$P(yi|x_1, x_2, ..., x_n) = \frac{(P(y_1) * P(x_1, x_2, ..., x_n|y_1))}{P(x_1, , x_2, ...x_n)} \tag{2}$$

As the denominator of this formula is always constant, the focus of the formula shifts on the numerator of the fraction. Because the Naive Bayes methods assumes that every feature is independent the numerator is equal to:

$$P(y_1) * P(x_1|y_1) * P(x_2|y_1) * .. * P(x_n|y_1) \tag{3}$$

which can be easily calculated from the set of features in the training dataset.

When working with continuous data, Gaussian Naive Bayes technique can be used. This method involves assuming that the values associated with a certain class follow a Gaussian distribution.

In the field of Natural Language Processing, Multinomial Naive Bayes is often used. This approach states that the set of features should actually be represented by a set of frequencies corresponding to a list of tokens.

2.2.2 Decision Trees and Random Forests

Random Forest is a simple yet effective technique that can be used for classification problems. This method is based on the concept of decision trees, concept that is covered next.

Decision trees are represented by a rooted tree structure that can point to a decision based on a certain path from the root to a leaf. Each level of the tree involves further splitting the decision paths based on a certain feature that has not been covered previously. Therefore, being given a dataset with N instances, every instance containing a set of features $X = (x_1, x_2, ...x_k)$ and a set of classes $(y_1, y_2, ...y_n)$, we can construct a decision tree with a maximum level of k (the number of features for an instance), every level corresponding to a choice based on that level's feature. This tree shall have at least one leaf that would output a class y_i from the above mentioned set.

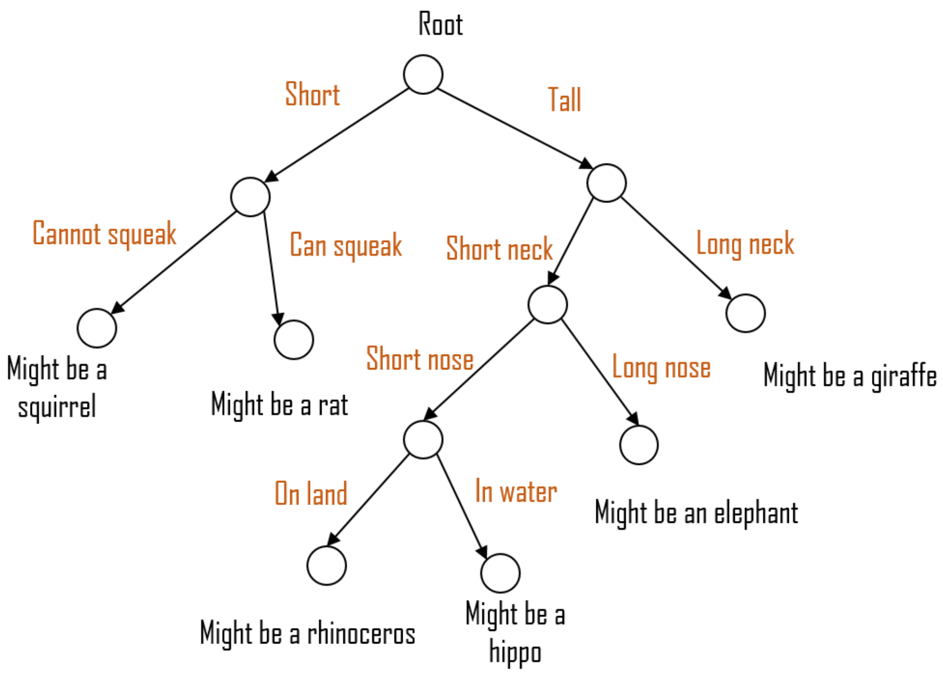


Figure 4: Example of decision tree. Animal classification[4]

Figure 4 presents a decision tree architecture for classifying an animal. First level of decision is based on the animal's height. The following level takes a decision either based on the length of the neck or on the squeaking ability of the animal. Further levels take into consideration features such as nose length or the natural habitat of the animal. All the leaves represent an according classification for the animal given as input.

Based on the nature of a dataset, decision tree accuracy may vary, some cases being proven

[4]https://forum.huawei.com/enterprise/en/machine-learning-algorithms-decision-trees/thread/710283-895

as suitable for a decision tree approach, while others being proven otherwise. In any of the above mentioned cases, decision tree results may be different depending on the order of the features. Selecting the order of the features might seem a tricky task, although it all comes to choosing the one that implies a minimum entropy for every decision layer. Following the given statement, features that weight more shall be placed closer to the root than features that are not correlated with the label of an entry. A problem with this kind of architectures is that for instances with a large number of features, big decision trees might be created, consistently slowing down the process of classification. In order to resolve such problems, pruning techniques are used.

Random Forest classifiers involve breaking the training dataset into multiple batches and building a decision tree for each batch of data. Therefore, each decision tree labels a given instance and the final label chosen by the Random Forest classifier is represented by the majority vote.

2.2.3 Support Vector Machines

Support Vector Machines or SVMs are yet another type of linear method used for classification problems. This technique was firstly introduced by Cortes & Vapnik (1995) as a solution for binary classification problems. Given a training dataset of entries, each entry containing N features, a Support Vector Machine classifier builds a N-dimensional space in which it places every entry of the dataset. For a better partitioning of the 2 categories, the points are then projected from the N-dimensional space to a M-dimensional space. Having all the dataset entries projected in this space, the approach involves separating the 2 categories of points by building a $(M-1)$ dimensional hyperplane. The points from each class that are the closest to this hyperplane are called support vectors. SVMs make sure that the distance from these support vectors to the chosen hyperplane is the longest, thus allowing for a maximum separation between the features of the 2 classes.

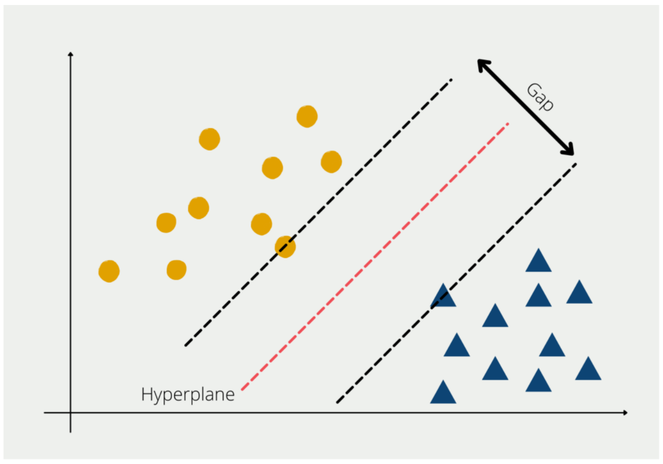


Figure 5: SVM representation in a 2D space [5]

Basically, what Support Vector Machines try to achieve is to build a M-dimensional space where it can plot all the features of the instances in order to provide a linear separation between the attributes of 2 classes.

Although this method was proposed as a solution for binary classification problems, it was later extended for multi-class scenarios as well (Duan & Keerthi, 2005). This is achieved by reducing the multi-class problem into a series of binary classification problems. Therefore a classification problem of this kind can be solved with the binary SVM classification method presented previously. One way of reducing a multi-class problem to a binary one is to chose an one-versus-all approach, this involving differentiating one class from the remaining. Another approach is to distinguish every existing pair of classes (one-versus-one method).

We cover the impact and accuracy of a Support Vector Machine model on a multi-class problem in the next chapters as this thesis's baseline model for experimenting was built using a SVM approach.

2.3 Neural Networks

2.3.1 Convolutional Neural Networks

One type of Artificial Neural Networks is the one of Convolutional Neural Networks or CNNs. CNNs have been primarily used to solve Computer Vision tasks as it's architecture is suitable for this kind of problems.

A CNN architecture contains 3 types of layers: convolutional layers, pooling layers and fully connected layers.

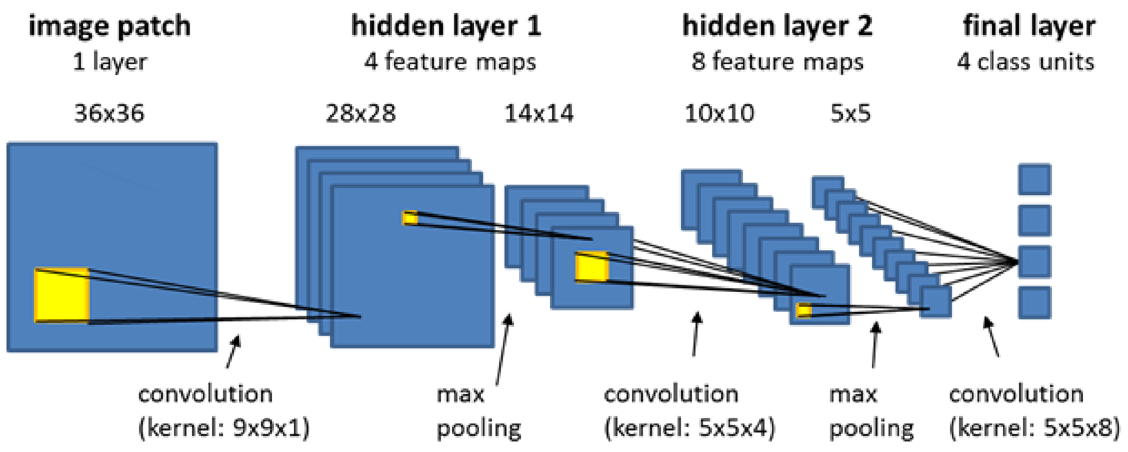


Figure 6: CNN architecture[6]

[5]https://databasecamp.de/en/ml/svm-explained

[6]https://docs.ecognition.com/v10.0.2/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm

The convolution layers are using certain kernels or filters in order to extract features from the previous layer. Following a convolutional layer, a pooling layer is usually placed. This assures that the size of the samples shrinks in order to reduce computation, keeping the loss of information at a minimum. CNN architectures can contain one or multiple such layers. The activation function for these layers is also important. In the field of image recognition, ReLu is preferred as it drops all the negative values from the features. At the end of the convolutional layers, the set of features is usually reshaped using a flatten layer and then linked to one or more fully connected layers. The output is represented by a layer of N neurons that should activate for the according prediction.

Comparing it with an usual fully connected Artificial Neural Network, the above presented architecture involves that it‘s neurons are connected to a small portion of the neurons from the previous layer. This results in a much smaller number of parameters to train, therefore a faster trainable architecture.

Even though the main focus of CNNs is pointed towards the computer vision field, their architecture can also be used, with some variations, in the field of Natural Language Processing. According to Kim (2014), a way of using CNNs for tasks such as sentence classification is to use convolution windows that are applied to one or more neighbor words in a sentence in order to discover a new feature.

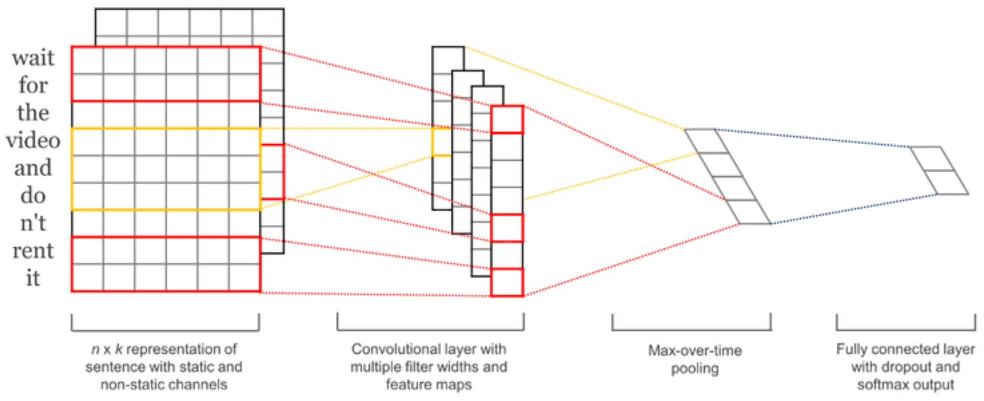

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 7: Example of CNN architecture for Natural Language Processing. Kim (2014)

This kind of model involves using multiple filters with different window sizes in order to obtain the best feature for every type of filter. The best fitting feature for a certain filter is then be passed on to a fully connected layer that helps making a prediction. This presented approach involves a word-level embedding. Techniques such as word2vec can be used in order to obtain such a model.

According to Zhang et al. (2015) a character-level embedding can be used in order to build a Convolutional Neural Network for sentence classification. Characters are encoded using a one-hot approach and passed to the neural network as a list of vectors. According to the authors, a possible architecture for this kind of embedding would contain 6 convolutional layers followed by one flatten layer and 2 fully connected layers.

2.3.2 Recurrent Neural Networks

One problem with the above mentioned Artificial Neural Networks is that they lack understanding context. When trying to solve problems where input order does matter, such as sentiment analysis or detection of offensive language, Recurrent Neural Networks bring a considerable improvement.

Recurrent Neural Networks introduce the notion of feedback, this meaning that cell signals can travel in both directions, having the potential to provide information to other cells or to themselves. Another big difference between RNNs and Feed Forward Networks is that input is fed sequentially into the initial layer. Every input fragment is processed and its output is fed back into the network in order to process the next section. Therefore, parts of input that require context or that are strongly connected with previous parts can now be interpreted accordingly.

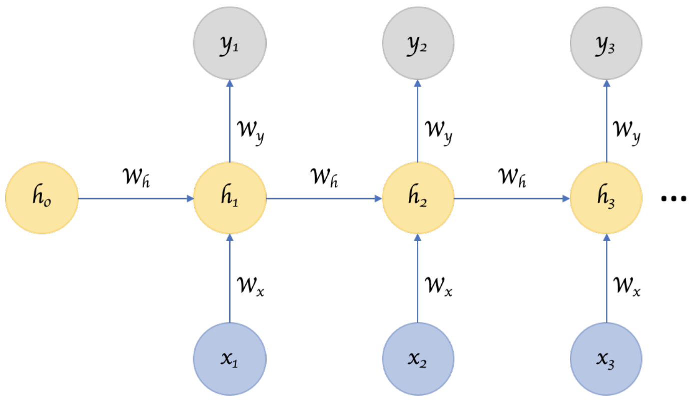


Figure 8: Recurrent Neural Networks architecture[7]

Figure 8 presents how input tokens (x_1, x_2, x_3) are fed into the network in a sequential manner. Every RNN cell processes the given input and the output of previous cell, computing and providing a vector (W_h) to the next cell and to global output.

NLP tasks can benefit from this kind of approach, as language processing is strongly dependent on context and on the connection between words. For example, feeding the sentence "What time is it?" to a normal Neural Network would result in poor results compared to feeding the sentence to a model which addresses the RNN architecture.

Although this method has clear benefits, it still falls short performance wise. Firstly, training such a Recurrent Neural Network is time consuming because of the sequential way of feeding

[7]https://gotensor.com/2019/02/28/recurrent-neural-networks-remembering-whats-important/

input. A normal network architecture processes all the input at once which, even though has its disadvantages, it can save a lot of training time.

Another problem with this architecture lies in the back propagation process of network training. After one batch of training is over, a gradient value is computed based on the current loss function of the training batch. The gradient propagates backwards to the start of the network helping to adjust internal weights of the network for better future predictions. The problem is that with every layer the gradient value can decrease. Therefore, the gradient value could diminish up to a point where it is insignificant for the earlier layers. Therefore, the starting layers fail to learn and the model‘s memory is shorten. Because of this, context is lost over time especially when working with long textual data.

The above presented problem is called the Vanishing Gradient Problem and in the following parts of the thesis we discuss approaches that can help solve it.

2.3.3 Long Short Term Memory

As mentioned in the previous section, Recurrent Neural Networks suffer from short term memory problems, making it very hard for a RNN to remember context when processing long inputs. Long Short Term Memory or LSTMs are represented by recurrent architectures that, as their name states, can hold prior information from the input for a longer period of time. In this manner, over-time loss of context is reduced and the network can have a better understanding of the overall input.

The LSTM architecture is a bit more complex compared to the one of RNN presented previously. A LSTM cell takes as input not only the hidden state of the previous cell and the current timestamp input but also the state from the previous cell. This previously mentioned state represents the memory of the cell and can be changed throughout the cell‘s gates. A LSTM cell consists of 3 gates:

- Forget gate.
- Input gate.
- Output gate.

The forget gate decides what information from the previous cells should be kept. Based on the current timestamp input, the previous context might be totally irrelevant so this gate helps in dumping deprecated and unnecessary information. The input gate assures that the current cell state is updated accordingly. This gate is very important when the current part of input is context-important as it is concatenated with the current cell state and then passed to the next cells. Based on the computations from these two states, the current cell state can be calculated. The output gate helps computing the hidden state that helps prediction and that is fed to the next LSTM Unit.

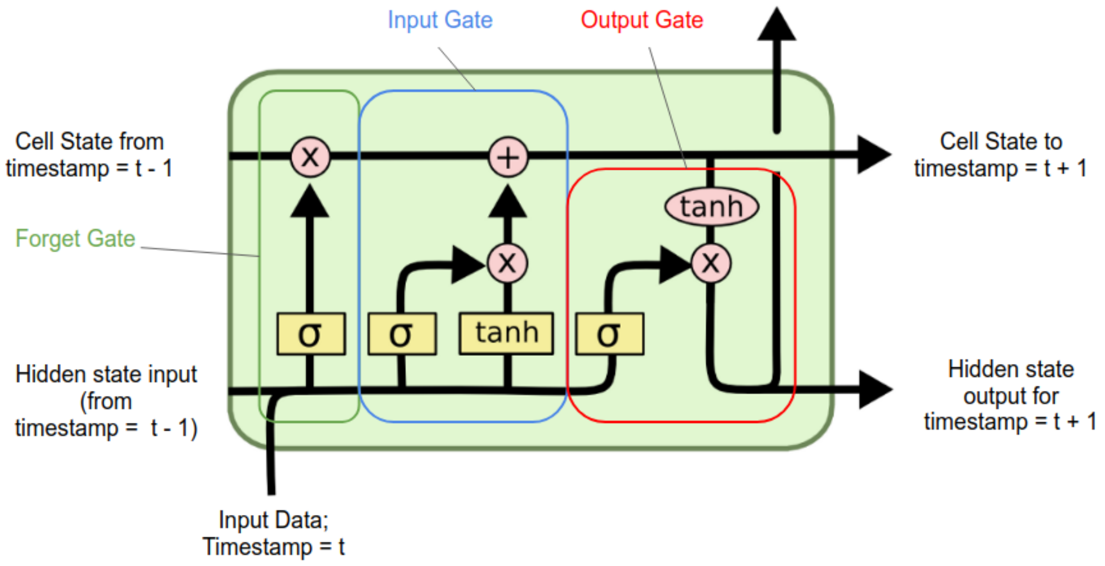

Input Gate
Output Gate
Cell State from timestamp = t - 1
Forget Gate
Hidden state input (from timestamp = t - 1)
Input Data; Timestamp = t
Cell State to timestamp = t + 1
Hidden state output for timestamp = t + 1
σ
σ
tanh
σ
tanh
X
+
X
X

Figure 9: LSTM cell architecture[8]

Even though LSTM architectures solve the Vanishing Gradient problem, they require a lot of time and resources for training. The sequential feeding of the input alongside with the complexity of the architecture represent the main drawbacks of this approach.

2.3.4 Gated Recurrent Units

As stated in the previous section, LSTMs lack performance as far as training time is concerned. In order to reduce this problem, Gated Recurrent Unit cells are used. A GRU cell removes the concept of a cell state and uses a hidden state in order to transfer information throughout the network. This alone represents a consistent improvement over the LSTM. In addition to this, a GRU cell is built upon only 2 gates:

- Update gate.
- Reset gate.

The update gate is a combination of the LSTMs forget and input gates. This gate decides what information might be deprecated and needs to be dumped and what information from the current timestamp input might be relevant in order to build the hidden state that connects to the next GRU cell. The reset gate is responsible for trimming past information that might occur as irrelevant for the current input‘s context.

[8]`https://www.turing.com/kb/comprehensive-guide-to-lstm-rnn`

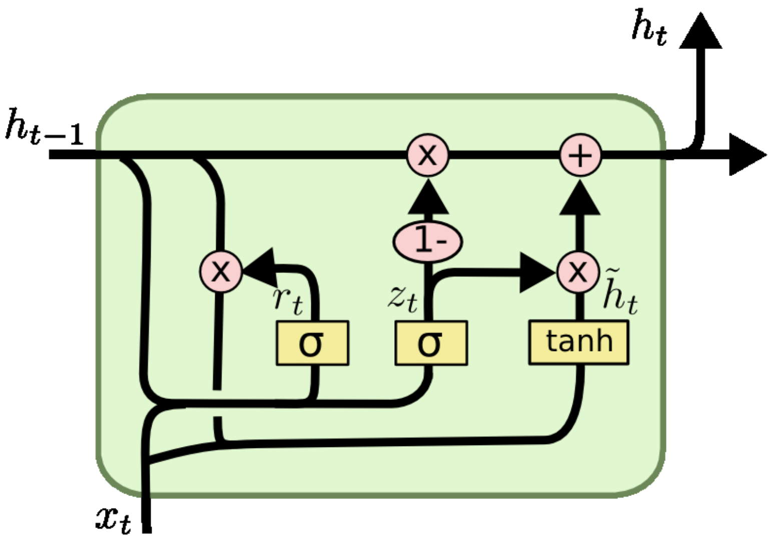


Figure 10: GRU cell architecture [9]

Because of its reduced number of gates, the GRU cell performs less computations than the LSTM one and can obtain better training times. In view of the fact that the concept of cell state is removed, GRU based networks are more compact and present a small number of connections between the nodes.

2.4 Transformer Based Models

As presented in the previous section, Recurrent Neural Networks architectures present a few disadvantages. The most important of them is related to the fact that their process of training is slow. Therefore, training such a model on big textual data would require a lot of time. This problem is due to the fact that RNNs can‘t be parallelized and, therefore can not make use of today‘s GPU technology.

Humans can comprehend multiple meanings of an input based on how that certain input is processed. For example, you can go through a sentence both ways and form multiple contexts. Therefore, a one-way traversing of a sentence could not be enough in order to determine its full meaning. Having this in mind it seems that trying to parallelize such a task of bilateral sequential processing of input might prove difficult and ambiguous.

This is the problem that Transformer architectures overcome. This architecture was firstly introduced by Vaswani et al. (2017) as a method for text translation, being later adopted and modified in order to perform other tasks like classification. A Transformer architecture is made up of 2 crucial parts: Encoder and Decoder.

[9]https://www.tutorialexample.com/understand-gru-gated-recurrent-unit-difference-between-gru-and-lstm-deep-learning-tutorial/

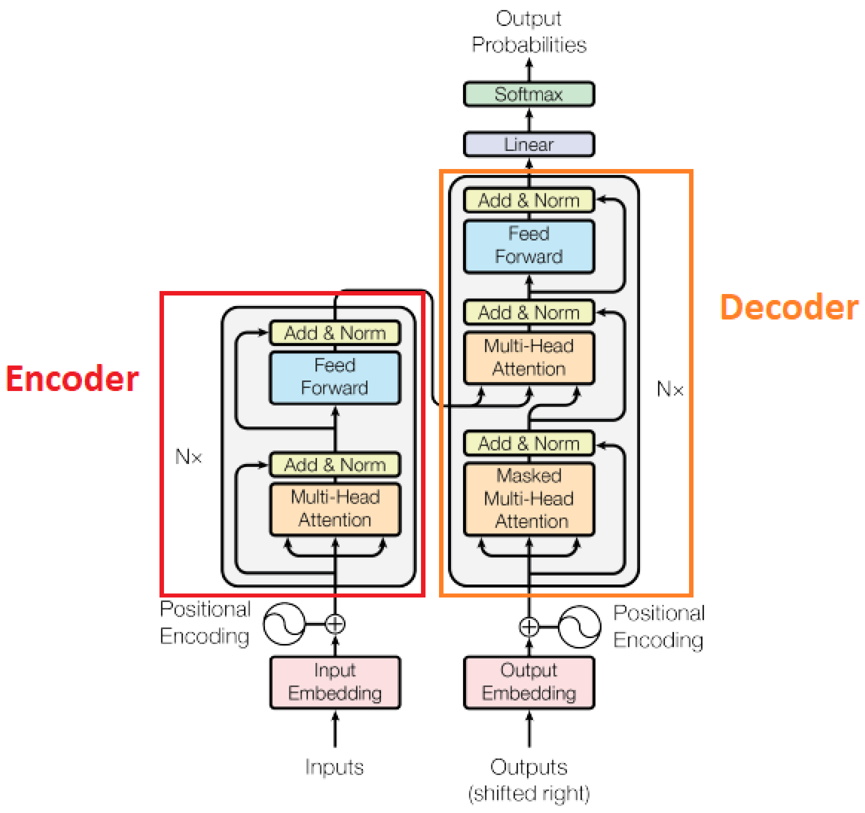


Figure 11: Transformer architecture [10]

The encoder takes all the input and generates embeddings for it. This task can be easily parallelized as there is no need for sequential processing of the input. But how can an encoder manage to generate accurate embeddings for a sequential input if all the parts of the input are processed simultaneously? Every part of the input is embedded and passed a positional encoding which is a representation of its position in the given input. This gets passed to the encoder where it goes through a Multi-Head Attention layer and a Feed Forward layer. The Multi-Head Attention block helps in computing what parts of the whole input are relevant for one part of an input. As an example, for a sentence which contains words that are strongly connected, this block should output attention vectors that suggest the strong connection between the 2 words. In this way, the network can learn how to manage context and connect parts of the input without sequential parsing. The Feed Forward layer is represented by a simple neural network that processes the attention vectors from the previous layer.

The decoder works sequentially, outputting a word that gets then passed as input to itself. Therefore, at a time t, the decoder receives as input the output it gave at the time $t-1$. This presented input is embedded just like the encoder input and passed into a Masked Multi-Head Attention layer which determines the attention vectors for every word considering that the right hand side of the sentence has not been predicted. This means that at a time t, this attention layer outputs a vector that has just t relevant elements, the remaining elements of

[10] `https://www.hearai.pl/post/13-slt/`

the vector being unconsidered. The output vectors from the previously presented layer and the output of the encoder serve as input for the next block of the decoder: the Multi-Head Attention block. This block has the same function as the one from the encoder. The last block of the decoder is represented by a simple Feed-Forward layer. The output of this last block goes through a linear layer of neurons that represents possible predictions. One prediction is chosen from these and passed to the decoder until the end of the input is reached.

The Transformer architecture has a clear and modular structure. The encoder can basically understand language, grammar and context being able to output accurate representations of a given input. The decoder can understand language and can generate text sequences. Therefore, these 2 components can be used individually in order to build more specific task-oriented models.

One model that uses only decoders is Generative Pre-trained Transformer 3. GPT-3 was introduced by OpenAI (Brown et al., 2020) as "an autoregressive language model with 175 billion parameters". GPT-3 can generate text in multiple configurations, being currently opened to the public. As this thesis focuses more on classification models, we next cover more details about architectures that use encoders.

Bidirectional Encoder Representations from Transformers or BERT is a model developed by Google (Devlin et al., 2019) that is currently used in Google‘s majority of applications that require natural language processing. BERT solves problems that require language understanding such as text summarization, sentiment analysis and question answering.

The architecture of BERT is fairly simple and concise: multiple stacks of encoders. BERT architectures can contain from 2 to 24 layers of encoders thus succeeding to fit any kind of environments. BERT takes as input a set of word tokens that are then transformed into embeddings using the same approach as the previously described encoder. BERT also takes as input 2 special tokens:

- *[SEP]* token.
- *[CLS]* token.

The [SEP] token is used to separate inputs when giving BERT more that one input (for example in question-answer problems). The [CLS] token is used in order to help with classification problems. These embeddings represent the input for the stack of encoders that form the BERT hidden layers. The output of BERT is represented by a set of hidden-state vectors with fixed size, every vector corresponding to a token from input. The BERT output also contains a special fixed size vector, [C], used for classification tasks.

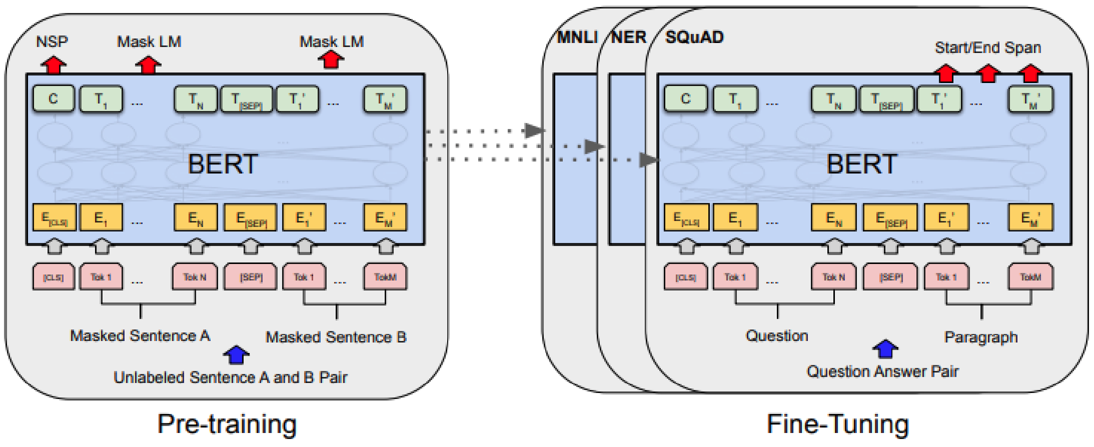


Figure 12: BERT architecture[11]

According to Devlin et al. (2019), a BERT model requires pre-training in order to learn language and context. This phase is divided in 2 sub-tasks:

- Masked Language Model.
- Next Sentence Prediction.

Masked Language Model (MLM) involves training BERT with sentences that have words randomly masked. For this task, BERT learns to predict accurate words for the given masks. Besides this, MLM enables understanding not only the left-to-right context of the input sentences but also the right-to-left context.

Next Sentence Prediction (NSP) helps BERT to learn the probability for one sentence following another one. Inputs consisting of 2 sentences are fed into the model and a binary output is expected: correct or incorrect order.

After the pretraining phase, a BERT model has an understanding of the language and can be fine-tuned in order to learn specific tasks such as offensive language detection. The fine-tuning phase involves taking the pretrained model, adding some extra layers at the end of the BERT encoders and training this network in a supervised environment.

In our experiments, BERT layers were used for all our neural architectures proving to be a very good choice when compared to the baseline SVM model.

[11]https://towardsdatascience.com/bert-why-its-been-revolutionizing-nlp-5d1bcae76a13

2.5 Frameworks and Development Environments

2.5.1 Tensorflow

Tensorflow (Abadi et al., 2015) is a free machine learning software library that operates at large scale and in heterogeneous environments. Tensorflow was developed by Google Brain who had previous experience in this field with their first generation system, DistBelief (Le, 2013). The initial version of Tensorflow was released in 2015. Four years later, in 2019, an updated version of the library, Tensorflow 2.0, was released.

The library focuses on the fields of Machine Learning and Artificial Intelligence and although it can be used for a wide range of tasks within these fields, Tensorflow has a particular target on training and inferencing on deep neural networks. Tensorflow offers high-level APIs like Keras, which provides immediate model iteration and easy debugging. In addition to building and training a model, TensorFlow can also help load the data to train the model and deploy it using TensorFlow Serving. During our experiments, Keras API proved to be very easy to use, helping to develop multiple neural architectures over a short period of time.

According to Hari (2022), Tensorflow is currently one of the most popular Machine Learning frameworks, being constantly maintained and expanded by an active community of developers. One of the reasons for its popularity is that building and training models is very easy and intuitive.

2.5.2 Scikit-learn

Scikit-learn (Pedregosa et al., 2011) is a free and open-source machine learning library for Python. The library started as a Google Summer of Code project in 2007 and its codebase was rewritten by other developers in 2010.

Scikit-learn aims to provide integration with a wide variety of state-of-the-art machine learning algorithms focusing on helping machine learning beginners to better understand this field and practice more, as it emphasizes on ease of use, performance, documentation and API consistency. Even though the API of this library is easy to use, Scikit-learn requires, however, basic Python knowledge.

A notable difference between this library and other Python based libraries is that Scikit-learn depends only on NumPy and SciPy libraries. This facilitates easy distribution for the library across different environments.

Scikit-learn offers easy access to classic linear machine learning methods for classifications such as Support Vector Machines, Random Forest or Naive Bayes which were covered in the previous section. This aspect of easy usability and fast development represents the main reason for choosing to use Scikit-learn in our experiments with linear models.

2.5.3 spaCy

SpaCy (Honnibal & Montani, 2017) is an open-source software library for advanced Natural Language Processing that was firstly released in 2015. SpaCy focuses on providing state of the art solutions especially for production usage thus helping developers to build applications that better understand large volumes of textual data.

The spaCy library provides a variety of practical tools such as pretrained word vectors, named entity recognition, text classification, lemmatization and morphological analysis. SpaCy offers support for over 66 languages and has 73 trained pipelines for 22 languages. The spaCy library would be a good choice for our further research in anonymizing our corpus.

SpaCy uses Thinc, a lightweight deep learning library, that offers an API for creating models based on layers imported from other Python libraries such as PyTorch or Tensorflow. Therefore, Thinc can be used as a wrapper around already defined models and workflows in order to allow easy integration for spaCy.

2.5.4 Google Colaboratory

Google Colaboratory[12] is a free development environment that allows one to write and execute Python code in the browser. As it can offer GPU and TPU support, Google Colab. is mostly used for machine learning tasks such as building and training models. This platform is easy to use, enabling developers to write code in the browser without any prior setup needed.

This development environment offers connection to a cloud server thus offering all the infrastructure needed for computation (CPU/GPU/TPU and RAM) as well as a way of storing data persistently. Basically users get their own machine in the cloud that can execute Python code, with the possibility of using its file system to their advantage. Therefore, the platform allows users to upload different datasets or models and load them with just a few lines of code.

The platform uses what is called a Notebook in order to remember and store all the code written by an user, thus allowing for easy shifting between projects and environments. The Notebooks offer a light weight revision control system and allow for simultaneously editing by multiple users.

Our experiments were entirely conducted using Google Colab. as it offered us a great development environment at a very low cost.

[12] `https://colab.research.google.com/`

3 METHOD

3.1 Corpora

In this section we present key aspects of our corpora focusing on data extraction approaches, annotation rules and examples.

3.1.1 FB-RO-Offense

Early thoughts on building a dataset were pointed towards using information either from Wikipedia comments or from online local newspapers platforms. We considered these 2 options because we knew that these platforms do not filter offensive language, thus being a perfect match for our requirements at a first glance. However, these trials proved to be unsuccessful due to the fact that not that much offensive language was present. Therefore, we considered to gather data from multiple newspapers platforms. The problem with this approach was that multiple web-scraping methods were required as every platform architecture was different. Another approach that we did not consider at the start of the experiments was gathering data from a social media platform. We knew Twitter does not get much attention in Romania, having only 420 thousand active users in 2021[1]. Therefore, focus had to be shifted towards Facebook which presented a number of 11.43 million active users in 2021[2]. Although Facebook uses offensive language filtration methods, its live broadcasts do not use on the spot filtering techniques. We had insight that certain Romanian political figures tend to use Facebook live broadcasting features and that a lot of offensive language is present through those broadcast comments. This seemed like the perfect combination, but another problem was encountered. Unlike other social media APIs, Facebook‘s API is a lot more restrictive and obtaining permissions for gathering other people‘s information such as comments and reacts was unfortunately not possible. Previous web-scraping experiments with various techniques pointed us in the direction of using the Selenium library[3] for Python. Selenium is an open source library that can automate browsers. It provides full access to the DOM of a web page, allowing access to any UI element and its information. Besides that, Selenium provides support for interacting with UI elements such as buttons and pop-ups.

Using the previous mentioned approach we were able to gather as much as 30 thousand comments from over 20 different live broadcasts. We initially set out to build a small dataset

[1]https://www.statista.com/forecasts/1143811/twitter-users-in-romania
[2]https://www.statista.com/forecasts/1136361/facebook-users-in-romania
[3]https://www.selenium.dev/

containing around 3 to 4 thousand entries, but having gathered this much data, building a larger corpus was taken into consideration. We cross-referenced all the 30 thousand comments with a list of Romanian offensive words and phrases and for every match we took into consideration the given match and it‘s surrounding comments. Further sanitization was performed by choosing comments with a minimum length of 15 characters and a maximum one of 187 and by removing comments that would contain URL‘s or phone numbers. After all the sanitization, 8017 entries remained in order to construct the final dataset.

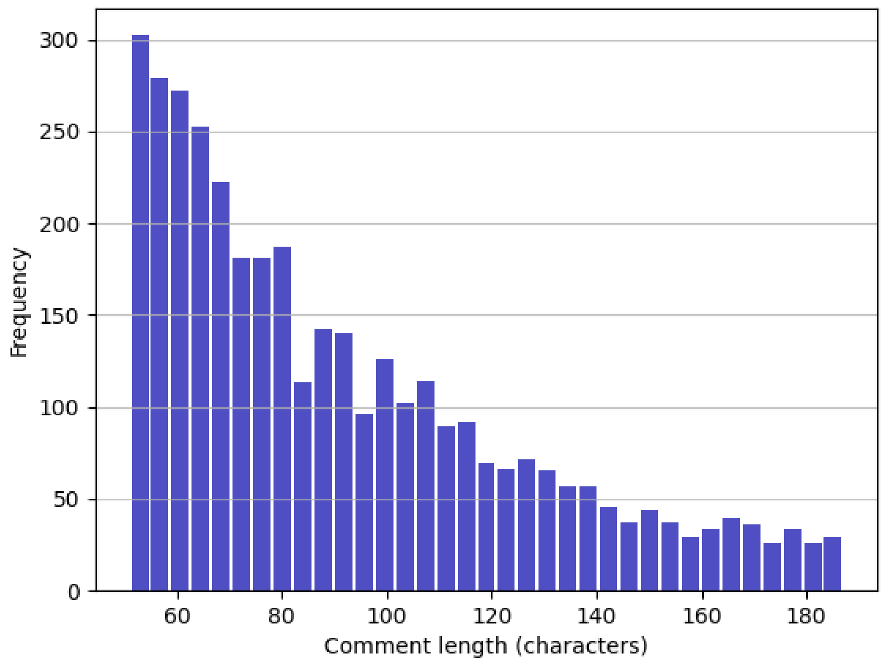


Figure 13: Comment length distribution for FB-RO-Offense dataset

Figure 13 highlights the nature of the dataset as most of the present text is represented by short comments (between 20 to 100 characters). It was expected to extract a large number of small length comments as we knew people do not write long texts in broadcasts because they would not get the chance to be read.

The annotation scheme for FB-RO-Offense dataset, adopted from Wiegand et al. (2018), is hierarchical, the first level of annotation being represented by a coarse-grained approach that involves dividing the dataset in 2 categories:

- OFFENSIVE.
- OTHER.

Although it might seem as a simple task, the tedious part was to decide what level of offensiveness we consider as OFFENSIVE. One important aspect regarding this dataset is that it was computed during the first phases of the 2022 Ukraine invasion. Therefore, this topic was addressed in the majority of the broadcasts and has weighted a lot throughout the labeling process. Having this in mind, we decided to label as OTHER phrases that would refer to leaders or mentalities regarding the 2 countries involved in the conflict. Therefore, sentences such as "Esti un putinist", "Esti un rusofil" were not labeled as offensive even though they might come as offensive or insulting to some individuals. Another type of language behavior that we labeled as OTHER is the one of booing. Phrases such as "esti dus cu capul", "generalul lui peste", "mancator de laturi" were also labeled as OTHER because we wanted to keep the level of offensiveness for the offensive category of entries as high as possible. The sentences that fall into the OFFENSIVE class should contain any shape of offensive speech such as cursing, insulting, abusing and profanity. Examples for this category of text are covered in the section for fine grained annotation.

The fine-grained level of annotation involves a further 3-way classification for the offensive comments:

- INSULT.
- ABUSE.
- PROFANITY.

The comments that were considered as INSULT were those light weight targeted curses that would make an individual feel unworthy or disrespected. We labeled as INSULT the following particular cases:

- Any resemblance with an animal: "esti un bou", "sarpe ce esti", "esti o capusa".
- Expressions that do not contain swear words but succeed in making a person feel insulted: "tusea si junghiul", "ai pavilionul bleg", "bai nefericitule", "esti un terminat".
- Sentences that reflect the sender‘s anger and contempt towards the receiver: "mars ma de aici!", "iesi", "valea", "hai sictir".
- Usual insults: "esti un fraier", "esti un prost", "dobitocule", "derbedeule".

The comments labeled as ABUSE tend to be more serious than the ones labeled as INSULT. We considered as abusive that language that would offend an individual by associating that individual with a certain identity or group that is negatively judged by the community. Some cases of ABUSE are as follows:

- Racist comments and associations with groups that are negatively perceived by the Romanian community : "esti o cioara borata", "esti un nazist", "esti o curva", "esti un bulangiu", "esti un bastard".
- Sexist comments and any type of sexual harassment: "esti buna la pat!", "e buna rau!".
- References to serious or deadly diseases: "sa te termine cancerul!", "sa te manance cancerul!".
- Death wishes or sexual harassment targeted towards a person or its relatives: "futu-ti mortii ma-tii!", "sa moara ma-ta!".

- Combinations of curses that refer to hell: "du-te dracu!", "proasta dracului!".

In the case of PROFANITY, a comment is not targeted to an individual or a group but contains swear words. This kind of speech would remain unchanged in terms of meaning if the swear words would be left out. Therefore, comments such as "sa moara mamaie ca e adevarat", "ce dracu! iar ploua?" were classified as PROFANITY.

Even though strict rules were considered for the labeling of one class, we also needed to define rules for comments that would contain features from 2 or 3 classes. These are as follows:

- Comments containing both PROFANITY and INSULT features would be labeled as INSULT.
- Comments containing both PROFANITY and ABUSE features would be labeled as ABUSE.
- Comments containing both INSULT and ABUSE features would be labeled as ABUSE.

Once the annotation process was carried out, inter-coder agreement had to be measured. Firstly, 100 random comments from the dataset were labeled by a different person in order to understand and get used to the rules. At this phase, an inter-rater agreement of 84% was reached. Later on, another random batch of 100 comments was extracted in order to be labeled by the same person. Based on these last 100 comments, we obtained an inter-coder agreement of 91%.

Even though the dataset contained 8017 comments, after the annotation process we observed that it was unbalanced in terms of class distribution.

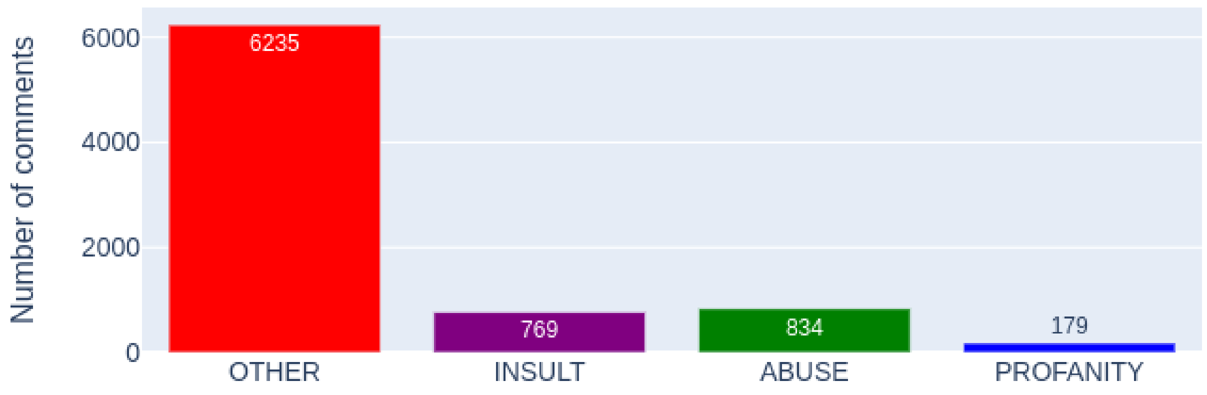


Figure 14: Fine grained class distribution for FB-RO-Offense dataset

Over 77% of the comments are labeled as OTHER, while classes such as INSULT or ABUSE weight only 10% each. In order to use a more balanced set of data, we kept all the comments labeled as OFFENSIVE (INSULT, ABUSE and PROFANITY) and removed 3562 comments labeled as OTHER. All our experiments were conducted using this reduced version of the dataset.

Table 1: Distribution of comments over categories in Reduced FB-RO-Offense dataset

Assigned label	**# Comments**	**% of Total**
OTHER	2,673	60%
INSULT	769	17.26%
ABUSE	834	18.72%
PROFANITY	179	4.01%
Total	4,455	100%

For our experiments, the reduced version of the dataset was statically divided into 3 batches:

- **Train batch**: 2851 comments (63.99%).
- **Validation batch**: 713 comments (16%).
- **Test batch**: 892 comments (20%).

3.1.2 RO-Offense

In this section we present a Romanian language dataset for offensive speech detection[4]. This corpus consists of over 12 thousand manually annotated comments extracted from a sports news website. The main principles for building this dataset are as follows:

- Focus on comments that reflects the online interaction on Romanian online platforms.
- Vast range of offensive textual data.
- Balanced data.

The data was extracted by crawling the comment section of Romanian sports website *Gazeta Sporturilor*[5]. This action took place in October 2020 and involved collecting all the available comments from 2008 until September 2020. Therefore, a total of 4,958,302 comments were gathered. It appeared that the website used some form of text filtering starting from 2015, therefore annotation focus was shifted to comments dating from 2008 until 2015. Only comment-threads with 20 to 50 messages were took into consideration. The selection of the sample was further restricted by choosing comments with a minimum length of 50 characters and a maximum length of 500 characters. Furthermore, comments containing URL's or phone numbers were discarded. All this sanitization resulted in a pool of 410,000 comments from which 30,000 were selected for the annotation process.

The final dataset consists of 12,445 comments with a length between 50 an 500 characters. The noticeable aspect is that the RO-Offense dataset contains much longer entries and,

[4]https://github.com/readerbench/ro-offense
[5]https://www.gsp.ro/

therefore, we expect our models build upon transfer learning approach to have a certain level of maturity and to be able to predict accurately different length inputs.

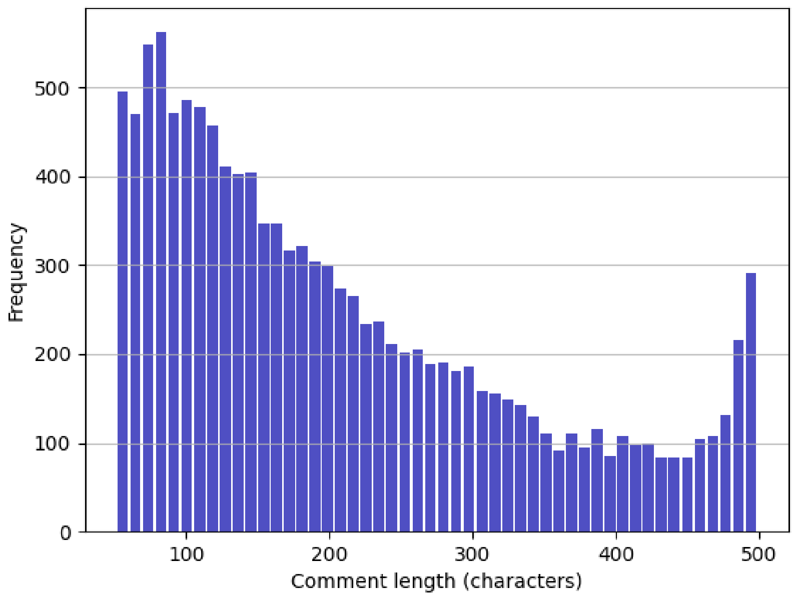


Figure 15: Comment length distribution for RO-Offense dataset

This dataset was annotated upon the same schema presented previously for the FB-RO-Offense dataset. Therefore, the classes and tasks of the 2 datasets are identical, allowing us to experiment with a transfer learning approach. The set of rules for annotation is similar, as it was adopted from Wiegand et al. (2018). In order to obtain a reliable dataset, the labeling process was carried out by 2 different annotators, obtaining an overall inter-rater agreement of 77.8%.

Anyhow, there exists differences between the offensiveness level chosen by the 2 datasets. The RO-Offense dataset contains more severe curses and a vast range of profane words, whereas the FB-RO-Offense dataset contains a lighter form of offensive speech. Therefore, a certain level of discrepancy between the datasets is observed, especially in the areas of INSULT and ABUSE comments. We further present differences between the 2 datasets focusing on examples and reasons for discrepancy.

Table 2: Annotation differences between RO-Offense and FB-RO-Offense datasets

Type of text	**RO-Offense label**	**FB-RO-Offense label**
esti o mizerie de om!	INSULT	ABUSE
du-te dracu!	INSULT	ABUSE
esti o javra ordinara!	INSULT	ABUSE
esti un jegos!	INSULT	ABUSE
dute-n pizda ma-tii!	INSULT	ABUSE

The different level of offensiveness between the 2 datasets can be observed in Table 2. Many sentences that were considered as ABUSE in FB-RO-Offense dataset are labeled as INSULT in the RO-Offense dataset. This aspect comes as normal given the fact that RO-Offense dataset contains a much larger number of profane and swear words and a higher overall level of offensiveness. On the other hand, FB-RO-Offense dataset contains less offensive speech, this being the reason for considering some of the above texts as ABUSE.

Currently, the RO-Offense dataset contains 7,873 offensive messages distributed over the PROFANITY, INSULT and ABUSE classes, and 4,572 non-offensive messages, thus presenting a much higher rate of offensive entries than FB-RO-Offense dataset.

Table 3: Distribution of comments over categories in RO-Offense dataset

Assigned label	**# Comments**	**% of Total**
OTHER	4,572	36.73%
INSULT	2,795	22.45%
ABUSE	3,461	27.81%
PROFANITY	1,617	12.99%
Total	12,445	100%

For our experiments, the presented dataset was used in a transfer learning approach. Therefore a train-validation-test split was not required as the training was conducted on the entire corpus.

3.2 Workflow

In this section we present all the steps that were taken in order to conduct and evaluate our experiments specifying the methods and frameworks that were used.

In the incipient phases of our investigations, we set out to build a simple baseline model for offensive language detection. Methods such as Multinomial Naive Bayes, Random Forest or Support Vector Machines were used. In terms of feature extraction, all these models were built upon a term frequency-inverse document frequency (TF-IDF) approach as we considered this method simple and effective. After conducting the experiments with linear classifiers, we decided to hold on to the SVM approach and refer to it in our thesis as it presented the best overall results.

In the training process we approached the 2 following methods:

- Training only on the FB-RO-Offense dataset.
- Transfer learning using both datasets described previously.

In the first case, the model was trained on the validation and train batches from FB-RO-Offense dataset and evaluation was done using the test batch from the same dataset. In the case of transfer learning, the model was trained on the whole RO-Offense dataset and on the validation and train batches from FB-RO-Offense dataset. In the same manner as the first approach, evaluation was performed on the test batch from the FB-RO-Offense dataset.

In order to properly evaluate the SVM model, confusion matrices were build for both fine and coarse grained levels of classification as well as for every training approach. Furthermore, for every classification level, evaluation metrics such as Accuracy, Precision, Recall and F1-score were computed. The last step in evaluating this model was to manually test its prediction on some on the spot texts. This step helped us in estimating the model's capabilities in a real-world environment.

The next experiment involved building a model using BERT. More specific, this model was built using a Romanian language pre-trained BERT version, RoBERT (Masala et al., 2020), followed by a fully connected feed forward layer. It's neural architecture are described in detail in Section 3.3.1. Training the BERT layer was not conducted in our experiments in order to reduce time. Therefore, only the feed forward network parameters were trained. Early results for this model did not prove as satisfying and, therefore, multiple trials were conducted using this model. A hyperparameter optimization was approached using Optuna[6] and, later on, using Weights and Biases platform[7]. This optimization method brought many improvements and was adopted in the following experiments as well.

In the same manner as for the SVM methodology, the training and evaluation of this model was conducted using the 2 already presented approaches: training on the FB-RO-Offense dataset and transfer learning using the entire RO-Offense dataset and the training batch from FB-RO-Offense dataset.

Evaluation metrics such as Confusion Matrices, Accuracy, Precision, Recall and F1-score were computed in order to estimate the performance of the model. In the same manner as for the SVM approach, manual testing was conducted for this model where we noticed a clear improvement in context understanding.

This architecture proved to be more accurate than the linear one but we aspired to build a better model. Therefore, after experimenting with different parameters we decided that a more mature and complex architecture had to be built in order to obtain even better results.

Our last experiment also involved a RoBert-base layer. Analyzing the results and model architectures from Safaya et al. (2020), we decided to insert a Convolutional Neural Network between the BERT layer and the feed forward network. As for tokenizer we chose the same pre-trained RoBERT-base one in order to keep a certain level of similarity between the 2 BERT models. In the same manner as the previously described approaches, we proceeded towards a 2-way training of the model. Once again, the BERT layer was set as not trainable. For both methods a hyperparameter optimization was performed, each optimization run containing 50 trials.

[6]https://optuna.org/

[7]https://wandb.ai/site

The same evaluation metrics as for the vanilla BERT model were computed. As the improvement from the previous model was clearly visible we decided to stop experimenting with model architectures and try to fine-tune this last presented model.

As mentioned previously, a hyperparameter optimization was performed for the 2 BERT architectures. This optimization involved adjustments for the following 4 parameters:

- Batch size.
- Learning rate.
- Maximum number of tokens to be consumed for an input.
- Size of the penultimate fully connected layer of the architecture.

Statistics such as parameter importance or parameter values are discussed in Chapter 5.

The frameworks and methods that were used in order to experiment and evaluate the previously specified models are Weights and Biases, Optuna, Scikit-learn and Tensorflow. We chose to work with these technologies because of their vast support for linear models and easy way of building and integration of neural networks.

3.3 Neural Architectures

3.3.1 BERT and FFN

This architecture uses a RoBert-base layer as encoder. The CLS token from the encoder‘s output is passed to a feed forward network. Hyperparameter optimization was performed on the size of the fully connected layer.

The BERT encoder receives as input the followings:

- A list of tokens represented by their identifier.
- An attention mask used to prevent the model from taking into consideration padding tokens.
- A list of token type ids. As this form of input is used mainly for Next Sentence Prediction tasks, it is filled with zeros during our experiments.

Even though BERT output contains the list of hidden-state vectors corresponding to the output tokens, for this model, only the CLS token is passed forward to the next layers.

The feed forward network is build upon 3 layers:

- Input layer which is represented by the CLS token from BERT.
- Fully connected layer with a *Tanh* activation function.

- Output layer with a *Softmax* activation function. This layer consists of 4 neurons which represent the prediction.

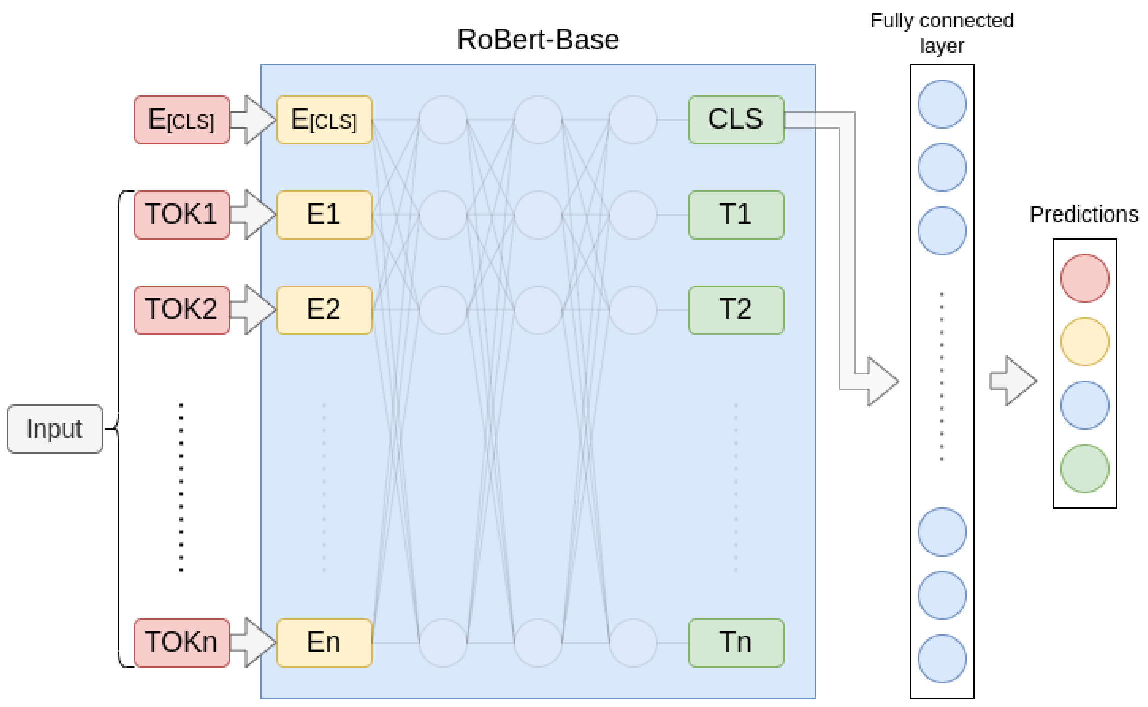


Figure 16: Architecture of BERT-FFN

This model contains a total of 115,087,268 parameters. Our experiments involved setting the BERT layer as untrainable, therefore only 24 thousand parameters being trainable. Adopting an early stopping approach, the training time for this architecture proved to be between 4 and 46 minutes for the transfer learning method. On the other hand, training this architecture only on the FB-RO-Offense dataset took between 2 and 7 minutes. The training process was supported by a NVIDIA Tesla P100 GPU[8].

3.3.2 BERT and CNN

This second architecture is composed of a RoBert-base encoder that passes its output to a CNN. After propagation throughout CNN layers, the resulted output is passed to a feed forward network that contains a dropout layer in the middle. The input for BERT is identical with the one from previous architecture. The CNN is build upon 3 layers:

- 1D convolutional layer with kernel size of 2, 32 filters and *ReLu* activation function.
- 1D convolutional layer with kernel size of 2, 64 filters and *ReLu* activation function.
- Pooling layer with a maximum pooling approach.

[8]`https://www.nvidia.com/en-us/data-center/tesla-p100/`

In contrast to the previous experiment, this one passes the whole output of BERT to the Convolutional Neural Network.

The feed forward network is build upon 4 layers:

- Input layer which is represented by the output layer of the CNN.
- Fully connected layer with a *Tanh* activation function.
- Dropout layer with a rate of 0.1.
- Output layer with a *Softmax* activation function. This layer consists of 4 neurons which represent the prediction.

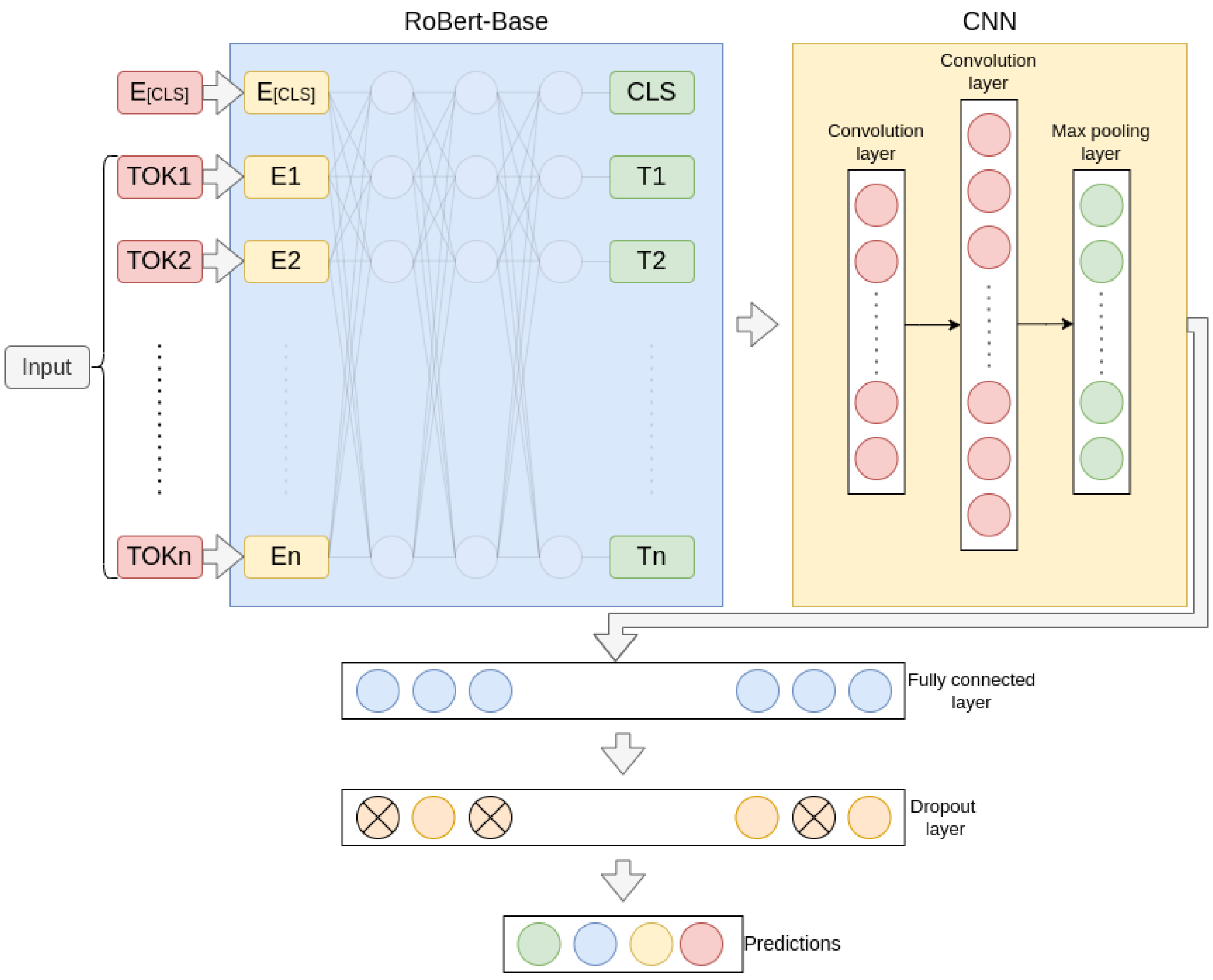


Figure 17: Architecture of BERT-CNN

This model contains a total of 115,151,204 parameters. Our experiments involved setting the BERT layer as untrainable, therefore only 88 thousand parameters being trainable. Adopting an early stopping approach, the training time for this architecture proved to be between 9 and 40 minutes for the transfer learning method. On the other hand, training this architecture only on the FB-RO-Offense dataset takes between 2 and 10 minutes. The training process was supported by a NVIDIA Tesla P100 GPU.

4 RESULTS

As presented in the previous sections of the thesis, our experiments involved three configurations: SVM, BERT with FFN and BERT with CNN.

For every configuration we have also adopted a transfer learning approach. Therefore, there exists a transfer learning version for every type of architecture. Moreover, for all neural architectures (BERT+FFN and BERT+CNN) we performed a hyperparameter optimization on both learning approaches. In the end, for every neural architecture, 4 different models were built:

- Model trained on FB-RO-Offense.
- Hyperparameter optimized model trained on FB-RO-Offense.
- Model trained on RO-Offense and FB-RO-Offense.
- Hyperparameter optimized model trained on RO-Offense and FB-RO-Offense.

For the approaches that did not involve hyperparameter optimization, the first model created during the experiments phases was selected, while for the approaches that did involve optimization, we selected the model that presented the highest performance.

The results were split in 2 sections: fine grained task and coarse grained task

Table 4: Results for Fine Grained task

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
SVM	0.60	0.45	0.45	0.60
SVM transfer	0.61	0.47	0.42	0.61
RoBERT+FFN	0.73	0.79	0.66	0.72
RoBERT+FFN optimized	0.75	0.80	0.70	0.74
RoBERT+FFN transfer	0.73	0.80	0.62	0.70
RoBERT+FFN transfer, optimized	0.75	0.81	0.67	0.73
RoBERT+CNN	0.82	0.84	0.80	0.82
RoBERT+CNN optimized	**0.84**	**0.86**	**0.81**	**0.83**
RoBERT+CNN transfer	0.78	0.82	0.74	0.78
RoBERT+CNN transfer, optimized	0.82	0.84	0.79	0.81

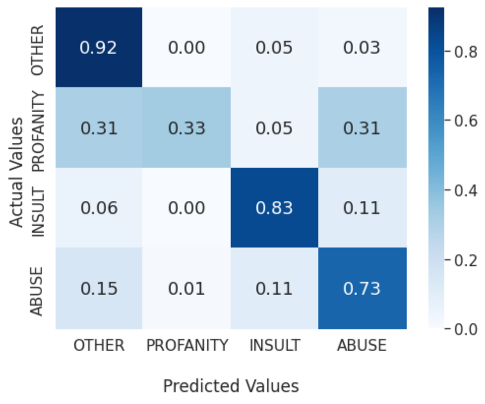
Table 4 presents the performance metrics of all the experimented models. At a simple examination, it can be stated that the "RoBERT+CNN" model obtained the best overall results.

The coarse grained task involved evaluating the models built for the fine grained task from a binary classification perspective. Therefore, predictions from the 3 offensive classes (PROFANITY, INSULT, ABUSE) have been unified under the OFFENSIVE label. A binary evaluation of the model was performed, obtaining the results displayed in Table 5

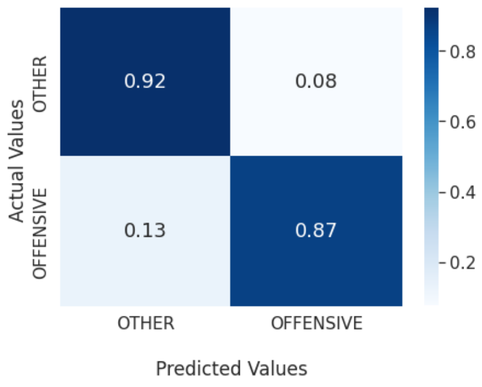
Table 5: Results for Coarse Grained task

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
SVM	0.69	0.68	0.67	0.69
SVM transfer	0.69	0.69	0.63	0.68
RoBERT+FFN	0.83	0.86	0.70	0.83
RoBERT+FFN optimized	0.84	0.89	0.69	0.84
RoBERT+FFN transfer	0.83	0.87	0.67	0.83
RoBERT+FFN transfer, optimized	0.83	0.90	0.64	0.83
RoBERT+CNN	0.88	0.90	0.79	0.88
RoBERT+CNN optimized	**0.90**	0.88	**0.87**	**0.90**
RoBERT+CNN transfer	0.88	**0.91**	0.80	0.88
RoBERT+CNN transfer, optimized	0.89	0.90	0.82	0.89

The above presented tables highlight the performance of the "RoBert+CNN optimized" model. Therefore, we next present its confusion matrices in order to have a better understanding of the model‘s performances and limitations.



(a) Fine Grained task

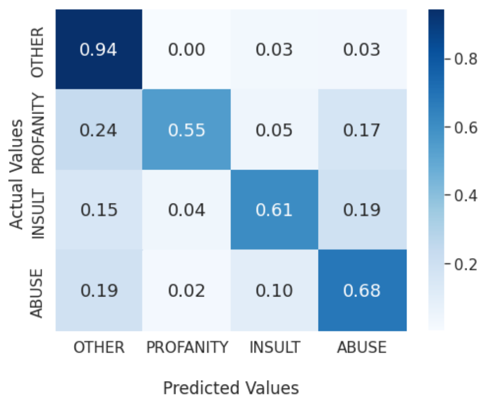


(b) Coarse Grained task

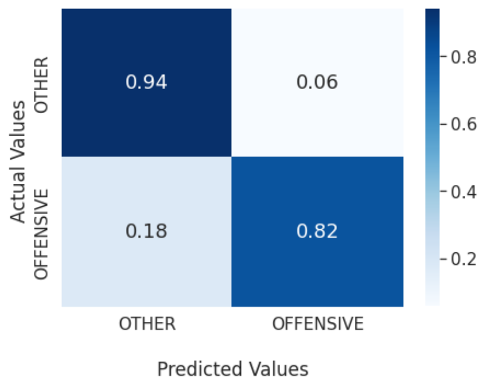
Figure 18: Confusion Matrices for the "RoBERT+CNN optimized" model

In terms of fine grained classification, the confusion matrix shows that the model lacks prediction accuracy when it comes to PROFANITY comments. It can be seen that many ABUSE and OTHER comments are labeled as PROFANITY. In the case of binary classification, the model seems able to accurately predict both types of comments. Nevertheless, this model presents a slightly high rate of false negatives. All these mentioned aspects are further discussed in Chapter 5.

In order to better understand the performance of the "RoBERT+CNN transfer, optimized" model, we will next present the confusion matrices for both fine and coarse tasks.



(a) Fine Grained task



(b) Coarse Grained task

Figure 19: Confusion Matrices for the "RoBERT+CNN transfer, optimized" model

It can be observed that this model has a better accuracy when it comes to predicting PROFANITY comments but, compared to the previous model, it falls short predicting other types of offensive comments.

As presented in Section 3.2 a 4-way hyperparameter optimization was approached during our experiments with neural architectures. Figures 20 and 21 present the results of this process for "RoBERT + CNN" model. Figures 22 and 23 present the hyperparameter optimization results for "RoBERT + CNN transfer" model.

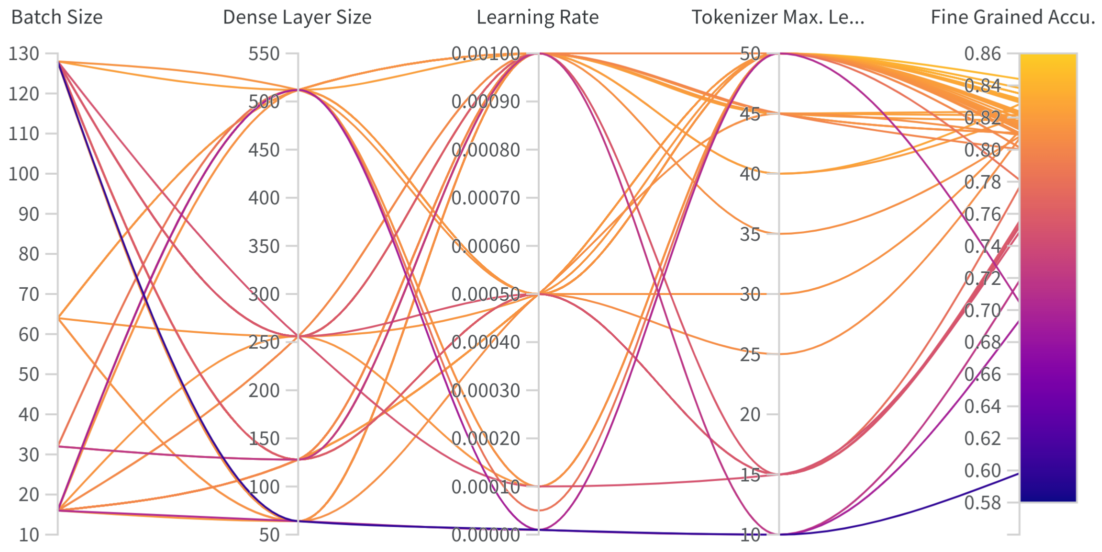


Figure 20: Hyperparameter optimization for the "RoBERT+CNN" model on fine grained task

5 DISCUSSION

In this chapter we evaluate the previously presented results, emphasizing on the best performing models, hyperparameter optimization, examples on wrong predictions and trials with real world examples. We also discuss about dataset and model limitations.

We next evaluate the fine grained results presented in Table 4. As stated previously, all our models have been trained using 2 approaches. We experimented hyperparameter optimization on every approach and this method proved to be beneficial for our models given the fact that the optimized versions obtained the best results.

Starting with the SVM models, from a simple examination we can state that even though their accuracy seems to be close to the one of the neural architectures, the precision and recall scores highlight the model‘s real capabilities. The SVM lacks in finding context for the input and predicts most of the comments as OTHER. The reason for its high accuracy is represented by the unbalanced nature of the training dataset. One positive aspect of this method is represented by the transfer learning approach. It can be seen that this method obtains slightly better results than the model trained only on FB-RO-Offfense dataset.

The "RoBERT+FFN" models present a clear improvement over the SVM approaches. Despite the improvement, this model still lacks understanding context. Even though it is able to accurately predict classes such as OTHER, INSULT or ABUSE, its major problem comes when trying to predict PROFANITY comments. The F1-score for this class is approximately 0.1, the lowest F1 class score for this model. Most of the actual PROFANITY comments are predicted as ABUSE or OTHER. On the other hand, having a high recall score might prove as beneficial in a real-world application as it is preferred to filter most profanities even though false-positives could occur during this process. Predicting them as ABUSE is understandable given the fact that these 2 classes tend to contain similar words and expressions, the only difference between them being the offensive speech target. Predicting the PROFANITY comments as OTHER represents a concern since comments labeled as OTHER do not contain any profane or swear words whatsoever. Even though it presents the same results, the transfer learning approach of this architecture proves to be more mature than the previously presented model. Given the fact that the "RoBERT+FFN transfer, optimized" model is trained on approximately 14 thousands comments from 2 different sources, its ability to understand context is more accurate. The F1-score for the PROFANITY class is almost 3 times higher (0.28) than its sibling‘s. It looks like this model is able to distinguish PROFANITY and ABUSE comments which is a good sign but it is still unable to differentiate between OTHER and PROFANITY classes. Another concern for this model is that it wrongly predicts as OTHER a large number of comments. These problems are partially solved by our next models. A positive aspect regarding this architecture is represented by the hyperparameter optimization of the models. The optimized versions present a growth of 0.02 in matter of fine grained accuracy as well as consistent growths regarding other metrics.

Our best performing architecture, "RoBERT + CNN", presents a consistent improvement over the previous ones. As highlighted in Table 4 The "RoBert + CNN optimized" model obtained the best results for every score, which reflects the maturity and balance of the model. As its confusion matrix (Figure 18a) highlights, this model solves the previously described problems. Its rate for wrong PROFANITY predictions has dropped as well as its rate for predicting offensive comments as OTHER. In terms of accuracy of predicting PROFANITY comments, the "RoBERT+CNN transfer" model presents a clear improvement (Figure 19a), even though it lacks accuracy in predicting other classes. Nevertheless, there could still be improvements in this area which are discussed in Chapter 6. Another important aspect about these models that should be touched is represented by the fact that the transfer learning approaches did not bring any improvements in matter of fine grained results. It can be seen in Table 4 that the transfer models did not overcome the models trained only on FB-RO-Offense dataset. We assume that the reason for this degradation lies in the difference between the datasets presented in Section 3.1.2. As we knew from the beginning of our experiments that the 2 datasets present differences, it was expected to obtain this kind of results. The hyperparameter optimization approach for this architecture proved to be beneficial one more time as better overall results were achieved. As multiple configurations are tested, the optimizer tries to guess the parameter importance and build stronger models. Figure 20 highlights the importance of "Tokenizer Max. Length" parameter as most of the best performing results have this parameter set to 50 tokens. The "Learning Rate" is yet another important parameter that can influence the result of the model.

In terms of binary classification, Table 5 highlights one more time the performance of "RoBERT + CNN" models. Similar to the fine grained task, SVM approach presents the poorest results. Unfortunately, the SVM transfer learning approach does not succeed to provide a better performing model. In the case of "RoBERT+FFN" models, a consistent improvement can be seen. The optimized models reach an accuracy score of approximately 0.84. The significantly low recall score in comparison to the precision score highlights the fact that this model presents a slightly large number of false negatives. Our best performing models, "RoBERT+CNN" obtain an accuracy score close to 0.90 with a more balanced distribution between the precision and recall scores. In the same manner as other approaches, these models do not present a clear improvement for the transfer learning approaches.

5.1 Error Analysis

In this section we focus on wrong predictions and try to explain the reason behind them. As presented previously, some models tend to not fully understand context, this being the reason for a low score regarding the PROFANITY class. Other models tend to predict a large amount of the comments as OTHER, this being the most concerning aspect of our experiments.

We next present a list of comments extracted from the testing dataset that were wrongly predicted by our best performing models: "RoBERT+CNN optimized" and "RoBERT+CNN transfer, optimized".

Table 6: Classification error analysis

Comment	**RoBERT+CNN optimized**	**RoBERT+CNN transfer, optimized**	**True label**
Bah voi nu vedeti ce hal de patriot ! Hai la barfa **dao draq** de politica!	ABUSE	PROFANITY	PROFANITY
DEMISIA hoților din guvern IOHANNIS ARAFAT CÂȚU ORBAN TOȚI **HULIGANI NENOROCIȚI** MAFIOȚI!	INSULT	ABUSE	INSULT
Marș criminalilor	OTHER	INSULT	INSULT
vorbește frumos ,nenea ,sau ești în târg cu precupețele!	INSULT	ABUSE	OTHER
Nu va spune nimeni , nimic ! Totul se face pe ascuns, dar nu se știe ce soartă va avea **nazistul** criminal împreună cu Arafat.	OTHER	OTHER	ABUSE
Români **sunt proști** înțelege..sunt puțini cu coi...in pantaloni	INSULT	OTHER	INSULT
Români **PROSTI** Din Naștere vor Război vor sa își omoare copiii	INSULT	OTHER	INSULT
Ce **varza e cucu**	INSULT	PROFANITY	INSULT
Silviu Eusebiu Hogas Ce fumezi, **saracie**?!	OTHER	OTHER	ABUSE
Numai **prosti vorbesti** nu iesti tu in masura dute acasa LA oale	INSULT	INSULT	OTHER
Ești mic dar ai mare manipulare	INSULT	INSULT	OTHER
Chiriac Și totii NİSTE **BOİ BALTİİ**	OTHER	OTHER	INSULT
Toti sunt **pupin curisti**.	INSULT	OTHER	INSULT
Asta duce lipsă de **mă.ciucă**!	ABUSE	OTHER	ABUSE
lasă-ne cu inepțiile, devii **penibil**!	ABUSE	INSULT	OTHER

Table 6 exemplifies the errors presented previously. Most common model error is to wrongly predict an INSULT comment as ABUSE or vice-versa. This is understandable given the fact that both kinds refer to a targeted offensive comment. Moreover, because of the differences in agreement between the 2 datasets, the transfer learning approach tends to make these errors occasionally. Another error is assigning an INSULT or ABUSE label to a non-offensive comment. As seen in the table, this happens when the model does not understand the context and just takes the words as they are (e.g. "vorbesti prostii") or when the model just interprets non offensive words as offensive (e.g. "Ești mic dar ai mare manipulare").

In order to test our models in a real-world environment we decided to build a small dataset and test their predictions. The dataset consists of 50 sentences and it was built on this simple premise: "try to trick an offensive language detection AI". We decided to not label the sentences and tried to approach an unsupervised test method. Table 7 presents examples that occurred as interesting while doing this experiment.

Table 7: Examples of predictions on a real-world dataset. Possible correct predictions in bold

Sentence	**RoBERT+CNN optimized**	**RoBERT+CNN transfer, optimized**
baga-ti-as surubul in piulita!	OTHER	**ABUSE**
sa-ti bag lebarul in sacosa!	OTHER	**ABUSE**
daca esti prost de mic, cand esti mare doar te joci	**INSULT**	**INSULT**
sunteti niste prapaditi!	**INSULT**	**INSULT**
sa-mi iei ouale in gura!	OTHER	**ABUSE**
sa-ti dau tot pixul sa il sugi	**ABUSE**	INSULT
grijania mă-tii de copil	**ABUSE**	**ABUSE**
sa-mi iei pu....l..a in gura	**ABUSE**	**ABUSE**
ce bozgoer esti...	INSULT	**ABUSE**
esti un bolsevic	**ABUSE**	**ABUSE**
ma-ta este foarte frumoasa!	**OTHER**	**OTHER**
sa-ti bag banii la cec!	OTHER	OTHER
canta-mi-ai la ce atarna	**INSULT**	OTHER
sa-mi lingi banana!	OTHER	**INSULT**
sa-mi fac schiuri din crucea ma-tii	**ABUSE**	**ABUSE**
Manca-mi-ai fleașca	**ABUSE**	**ABUSE**
Du-te-n puii mei	**INSULT**	**INSULT**
Manca-mi-ai ciocolata	OTHER	OTHER
Culca-te-ar vaca	**INSULT**	OTHER
Sa ma iei de pițigoi	INSULT	**ABUSE**
ai capul mare dar nici creierul nu sta inghesuit	**INSULT**	OTHER
sa imi ciufulesti barzoiul	INSULT	**ABUSE**

We were surprised that the models seemed to correctly predicted some of these sentences. Texts such as "baga-ti-as surubul in piulita" or "ai capul mare dar nici creierul nu sta inghesuit" demand a high level of understanding language and context. On the other side, errors still exist among these predictions. For example, the sentence "manca-mi-ai ciocolata" whose substrate is abusive was predicted as OTHER. Romanian language is vast and contains many forms and shapes of offensive speech, which can represent a barrier in training a mature and capable model for offensive language detection.

5.2 Limitations

In this section we briefly present all the problems we encountered and the limitations of our dataset and experiments.

Regarding data extraction for the FB-RO-Offense dataset, a big limitation is represented by our infrastructure. As stated in Section 3.1.1 the data extraction process was carried on using web-scraping techniques provided by the Selenium library for Python. This approach helped us build our dataset but does not represent a sustainable method for future experiments, as web page elements can change frequently and break our infrastructure. An improvement in this area would be to gain permissions and use the Facebook API[1] in order to extract comments from live broadcasts. Gaining permissions to do such things might prove as difficult, but, nevertheless, it would bring an improvement to our data collection infrastructure. Another limitation regarding the data extraction infrastructure is represented by its static architecture. The comments have been collected by manually running a script when a certain political person would start a broadcast. This method is rudimentary and unsustainable in the long run. The infrastructure should be upgraded to a more advanced architecture that would automate the whole process of looking for broadcasts and launching a data extraction session. In this manner we would be able to build a pipeline that would train our model in an unsupervised manner.

Another limitation regarding our dataset is represented by the topics of the broadcasts from which the data was gathered. The extraction process took place during the 2022 Ukraine invasion and, therefore, our dataset contains many comments related to politics and war.

Moving on the the annotation process, the biggest limitation of this area was coherence related. Given the fact that the annotation process was carried out by one person, it was hard to keep a certain level of coherence throughout the whole process. Even though we obtained a good inter-coder agreement score based on a random selection of 100 comments, our feeling is that is still work that needs to be done in this area. Another problem regarding the annotation process is related to the form of the comments. We encountered a lot of cases where a certain sentence would not be understandable, making it hard to be labeled. There were many cases of wrong or misspelled comments and it only made our work tougher.

As stated before, all of our models that involved a transfer learning approach proved to be underperforming when compared to the ones trained only on the FB-RO-Offens dataset. The reason for this is represented by a discrepancy between the 2 datasets. Given the fact that they were build from different sources and annotated by different persons who followed dissimilar annotation rules, the 2 datasets present a certain labeling disagreement especially between the INSULT and ABUSE classes. We assumed from the start that this limitation will have an impact over our experiments and it finally proved so. Nevertheless, improvements such as annotation reiteration could be made in order to build a more mature and accurate model using the transfer learning approach.

[1]`https://developers.facebook.com/docs/graph-api/`

6 CONCLUSIONS AND FUTURE WORK

In this thesis we presented all our trials and experiments on building an offensive speech dataset. We focused on building a reliable corpus with data collected from Facebook live broadcasts of Romanian political figures. Limitations and drawbacks regarding this area were also covered focusing on future improvements that could be adopted. Regarding the dataset, we displayed our hierarchical annotation schema focusing on presenting the majority of our labeling rules alongside explanation for each of them. In the view of the fact that we proposed to experiment with a transfer learning approach, all the important aspects of a second corpus, RO-Offense, were touched. On the other hand, all the discrepancies between the 2 datasets were displayed focusing on what could be improved in this area.

For our experiments, we firstly presented a baseline model involving a SVM approach. Even though the focus was not targeted towards this part of the thesis, the results obtained by the baseline model were displayed and discussed.

The second step of our experiment involved working with BERT based architectures. A first architecture was created using a BERT and a FFN layer. Besides a simple training on our dataset, a transfer learning approach was adopted for this architecture resulting in 2 models. We experimented a hyperparameter optimization for each model and later presented the benefits of this method. Results for this architecture were displayed and discussed emphasizing on potential improvements.

The last step of our experiment brought in another BERT based architecture, this time involving a CNN layer. In the same manner as for previous experiments, we obtained 4 models for this architecture. Comparing them with previous models, better results were noticed. For these models we presented an error analysis in order to enhance on future improvements that could be adopted.

Regarding future work, we proposed to enhance our data collection infrastructure in order to be more sustainable and independent. In this manner, we could continuously collect data and perform unsupervised learning on our models in order to obtain better results. Improving the infrastructure could also help in generating a big dataset that could serve not only for offensive speech detection but also for other classification tasks as well. In what involves transfer learning approaches, we proposed to enhance the inter-coder agreement between the 2 datasets in order to obtain better results with the models that use transfer learning. We have also proposed to build better performing model in the future. Regarding this, it is intended to perform a hyperparameter optimization involving a larger number of parameters and more runs. Moreover, we have proposed to experiment in the future with other BERT based architectures such as "RoBERT + BiLSTM" in order to potentially obtain better results.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *Proceedings of the international aaai conference on web and social media* (Vol. 11, pp. 512–515).

de Gibert, O., Pérez, N., García-Pablos, A., & Cuadros, M. (2018). Hate speech dataset from a white supremacy forum. In *Proceedings of the 2nd workshop on abusive language online (alw2)* (pp. 11–20).

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171–4186).

Duan, K.-B., & Keerthi, S. S. (2005). Which is the best multiclass svm method? an empirical study. In *International workshop on multiple classifier systems* (pp. 278–285).

Hari, S. (2022). *Best machine learning frameworks(ml) for experts in 2022.* `https://hackr.io/blog/machine-learning-frameworks`. (Last accessed 12 June 2022)

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.*

Islm, T., Meng, H., Pitafi, A. H., Zafar, A. U., Sheikh, Z., Mubarik, M. S., & Liang, X. (2021). Why do citizens engage in government social media accounts during covid-19 pandemic? a comparative study. *Telematics and Informatics*, *62*, 101619.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1746–1751).

Le, Q. V. (2013). Building high-level features using large scale unsupervised learning. In *2013 ieee international conference on acoustics, speech and signal processing* (pp. 8595–8598).

Masala, M., Ruseti, S., & Dascalu, M. (2020). Robert–a romanian bert model. In *Proceedings of the 28th international conference on computational linguistics* (pp. 6626–6637).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Perrin, A. (2015). Social media usage.

Safaya, A., Abdullatif, M., & Yuret, D. (2020). Kuisail at semeval-2020 task 12: Bert-cnn for offensive speech identification in social media. In *Proceedings of the fourteenth workshop on semantic evaluation* (pp. 2054–2059).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Retrieved from `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`

Wiegand, M., Siegel, M., & Ruppenhofer, J. (2018). Overview of the germeval 2018 shared task on the identification of offensive language. In *14th conference on natural language processing konvens 2018* (p. 1).

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., & Kumar, R. (2019). Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). In *Proceedings of the 13th international workshop on semantic evaluation* (pp. 75–86).

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28). Curran Associates, Inc. Retrieved from `https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf`

APPENDIX - HYPERPARAMETER FINETUNING

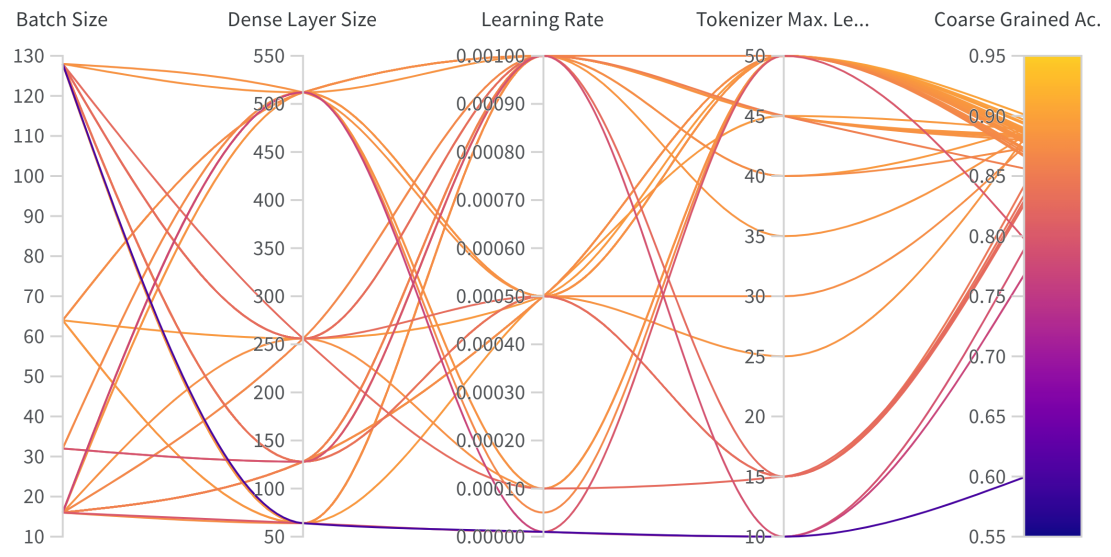


Figure 21: Hyperparameter optimization for the "RoBERT+CNN" model on coarse grained task

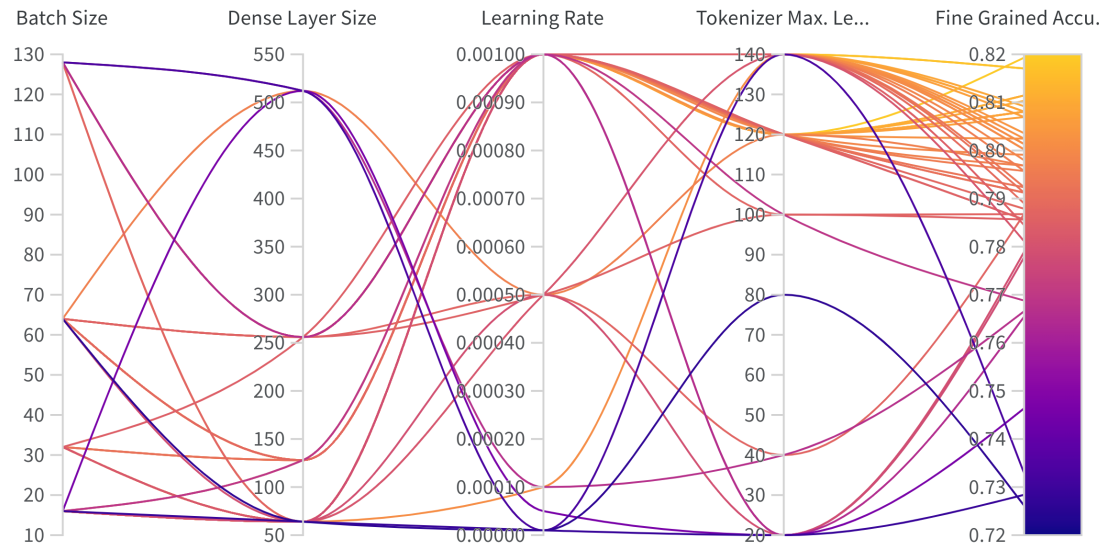


Figure 22: Hyperparameter optimization for the "RoBERT+CNN transfer" model on fine grained task

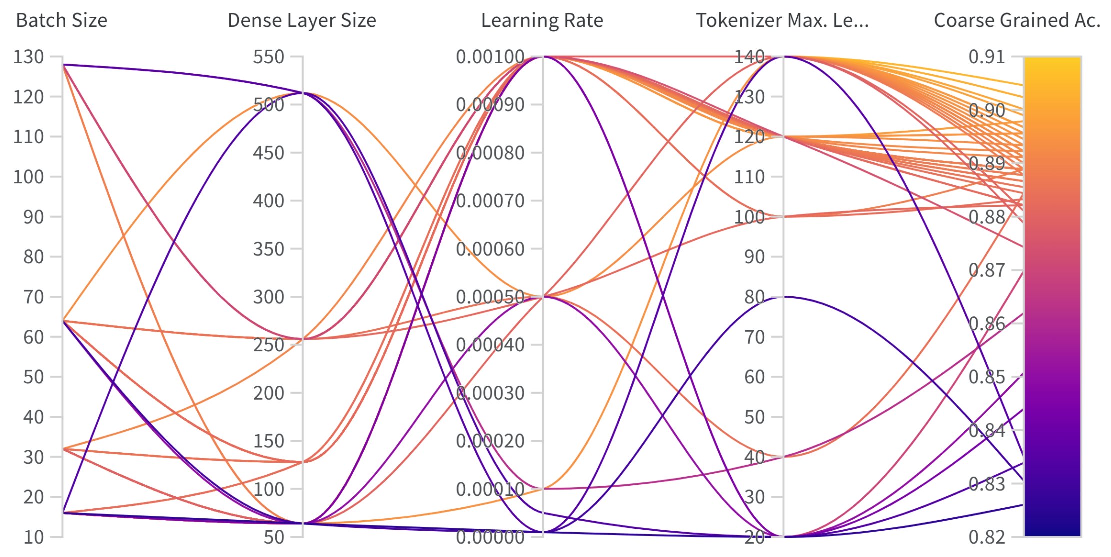


Figure 23: Hyperparameter optimization for the "RoBERT+CNN transfer" model on coarse grained task