City, University of London MSc in Data Science

Project Report

2022

The use of Pre-Trained Word Embeddings in Automatic Hate Speech Detection Models

Dimitrios Megkos

Supervised by: Oleksandr Galkin

26 September 2022

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work, I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed: ***Dimitrios Megkos***

Abstract

The purpose of this project was to investigate whether the use of pre-trained, contextualised word embedding methods improve the performance of automatic hate speech detection models. The research was based on the "Hate Speech Dataset from a White Supremacy Forum" paper by de Gibert et al., (de Gibert *et al*., 2018). The aim of the research was to improve upon the published results and add to the body of knowledge. The models that were selected for this study were a Support Vector Machine, Convolutional Neural Network, and Long Short-Term Memory model. The pre-trained word embedding methods that were tested with these models were the Word2Vec, GloVe, ELMo, and BERT methods, replacing the traditional methods used in the reference paper. The trials revealed that ELMo and BERT improved all three models' performance in both hate and no hate speech classification. A new hybrid BERT CNN-LSTM model was introduced that achieved an overall accuracy of 82%, 9% more than the best model in the reference paper. The model was applied to a different dataset containing hateful tweets, achieving an overall accuracy of 73%. Fine-tuning the model for just one training epoch increased the overall accuracy to 85%, suggesting that the hybrid model can be used for Transfer Learning.

Keywords: Natural Language Processing, Sentiment Analysis, Hate Speech Detection, Contextualised Word Embeddings, Deep Learning

Table of Contents

Chapter 1. Introduction

1.1. Background

Social media have become one of the main ways of communication in modern society, with more than four billion users globally (*Internet and social media users in the world 2022*) spending on average 147 minutes daily communicating, sharing ideas, and opinions (*Global daily social media usage 2022*). The mass adoption of social media and subsequent increased available content online provide benefits to users in terms of cost and speed of communication, though can also have ill consequences, such as exposure to expressions of hate speech. Hate speech not only has an immediate detrimental psychological impact on the targeted person, but can also have wider consequences on society, as these behaviours can exacerbate tensions between groups and even jeopardise the positive progress that has been made on topics such as racism and human rights (Iginio *et al.*, 2015). For the purpose of this paper, hate speech is considered as "any communication that disparages a target group of people based on some characteristic such as race, colour, ethnicity, gender, sexual orientation, nationality, religion, or other characteristics" (Nockleby, 2000), (de Gibert *et al.*, 2018).

In the past few years, social media companies, such as Facebook and Twitter, have been heavily criticised for the ineffectiveness of their methods in managing phenomena of hate speech. The topic has also received attention from regulators and institutions (UNESCO and United Nations Office on Genocide Prevention and the Responsibility to Protect, 2021) and the academic community (Mondal, Silva and Benevenuto, 2017), (Mathew *et al.*, 2019).

Online hate speech detection is difficult to analyse from a methodological and technological standpoint due to the inconsistency of detection systems, the opaque nature of proprietary algorithms, and the difficulty of access to data stored by firms. Clarity on how to solve these problems is critical for gaining a better understanding of how online hate speech arises and spreads, and then creating effective countermeasures (UNESCO and United Nations Office on Genocide Prevention and the Responsibility to Protect, 2021).

1.2. Purpose and Beneficiaries

The purpose of this research is to contribute to the advancement of detecting mechanisms, which can be utilised by social media websites to accurately detect and classify hate speech language, protecting their users from falling victims to it. These detecting mechanisms come in the form of deep learning neural networks, which can classify sentences into distinct

categories. The result of this research will be a deep learning model, which can recognize the context of words inside a sentence and classify it either as containing hate speech or not.

Improving these detecting tools can benefit social media users and operators, create a safer online environment, identify the origins and potential impact of hate speech in social interactions, and provide solutions to specific operational issues social media platforms face. Furthermore, this research will benefit the Data Science and Natural Language Processing community, by experimenting with a combination of different neural networks and word embedding methods, improving previously published results, and adding to the collective knowledge. Finally, the suggested methods could have broader applications in the field of Natural Language Processing that are not hate speech detection related.

1.3. Objectives

The research will be conducted based on the following research questions:

1) "To what extent do pre-trained contextualised word embeddings improve hate speech detection?"
2) "Can models that take advantage of pre-trained word embeddings be used for transfer learning themselves?"

The following objectives will contribute to answering the research questions, in the form of project steps:

- Reproduce the results of the reference paper, i.e. (de Gibert *et al.*, 2018)
- Apply and compare different pre-trained word embedding methods
- Create, tune, test, and save the best performing model
- Assess whether contextualized pre-trained word embeddings improve hate speech detection
- Assess whether the best performing model qualifies for transfer learning

1.4. Methods

A thorough literature search was conducted to improve the understanding of Natural Language Processing and text classification techniques and methods. Several internet articles and tutorials were reviewed to decide which Python Machine Learning and Deep Learning libraries would be best fitted for this research. Customizability and ease of use were the main attributes that were considered when making the choice. Several Deep Learning and Natural Language Processing DataCamp courses were completed to solidify my knowledge of these techniques.

All three models mentioned in de Gibert et al.'s work (de Gibert *et al*., 2018) were recreated, following the instructions mentioned in their paper, and their results were reproduced locally. This was the foundation of my research, as it enabled me to use the reproduced results for benchmarking and make direct comparisons between the different variations of the models that were created.

Four different pre-trained word embedding methods were tested with Support Vector Machine, Convolutional Neural Network, and Long Short-Term Memory models, comparing all twelve models' performance with the originals' and with each other's. For each word embedding method, a visualization was created using the t-SNE technique and plotly express library, in order to discover and explain their strength and weaknesses.

The two best-performing models were tuned using the Hypeband tuner to try and push them to their limits. During this process, a hybrid model combining the architecture of both was also tuned and the top performer was put to the test to answer the first research question. Lastly, to answer the second research question, the model was tested twice on a bigger similar dataset, first as it was, and second after fine-tuning it on the new dataset.

1.5. Testing and Performance Metrics

To decide whether the project's objectives were met or not, each model's performance was evaluated by calculating a variety of performance metrics typically used in classification problems i.e., the confusion matrix, overall accuracy, and the accuracy, precision, recall, and F1 Score of each of the two classes. Each of the model's results were compared with the published results reported in the paper, and against each other.

All tests were performed on the same set of hate speech sentences used by the paper's authors to publish their results. A final round of tests was performed using the best-performing model and a second, bigger dataset consisting of tweets, to evaluate the model's performance on similar types of data.

1.6. Work Plan

Figure 1 below shows the work plan as it was carried out from the beginning of the research to the end. An initial literature search was performed to identify a published paper related to hate speech detection with the potential to be further improved. An additional literature review provided knowledge on the variety of methods that are used in Natural Language Processing

and state-of-the-art. Initially, the "visualization of word embeddings with t-SNE" and "tuning of the models with the HyperBand algorithm" stages were not part of the original work plan. They were added during research to visualise the differences between the four pre-trained word embedding methods and find the best combination of hyperparameters for the models, respectively.

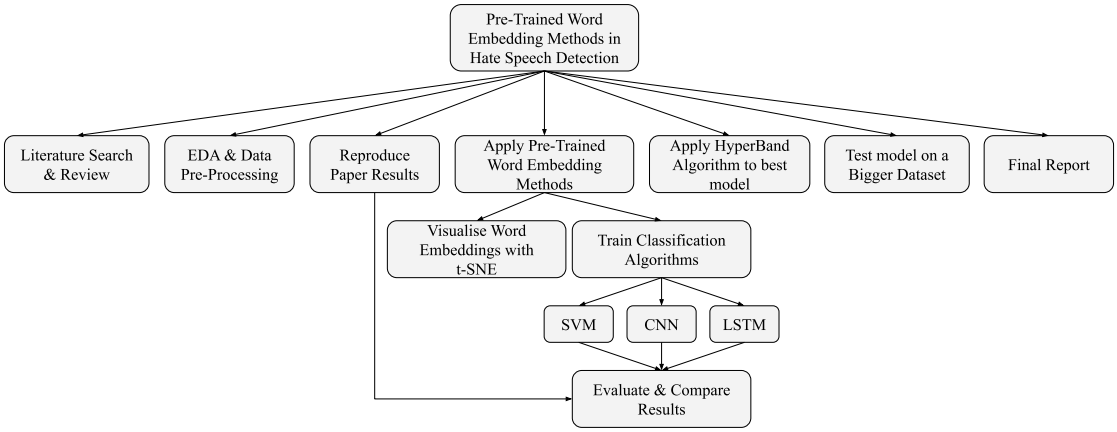

Figure 1: Project Work Plan

1.7. Structure of the Report

Chapter 2 explains the context of the research, identifies which methods have been applied before to similar problems, and what is currently state-of-the-art. Chapter 3 describes in detail all the methods, which were used, as specified in 1.4. above. In Chapter 4, all performance metrics and visualisations that were used to evaluate the models and pre-trained word embedding methods are shown, along with experimental results and obstacles that emerged during the development of the models. In Chapter 5, the results are discussed in comparison with the initial objectives of the research and the research questions. Chapter 6 evaluates the project work as a whole, reflecting on the original choice of objective, highlighting the project's limitations while also suggesting future work and ways to improve the model. A detailed reference list is also included in the final pages.

The coding part of this research was completed in an iPython Notebook document, using Google Colab Pro+ for the extra processing power of a GPU (Tesla V100-SXM2-16GB), and bigger RAM size (51GB).

Chapter 2. Context

2.1. Introduction

This chapter describes the motivation for the project and the similar work that has been done in recent studies involving hate speech detection. In addition, the chapter sets the background for the methods and techniques that were used throughout this research.

2.2. Project Motivation and Similar Work

Hate speech detection is an active field where a significant work has been done using both traditional machine learning and deep learning methods. This project was motivated by the work of (de Gibert *et al*., 2018) in "Hate Speech Dataset from a White Supremacy Forum" and their research in the field of Natural Language Processing and more specifically in Hate Speech detection. Through their work, they introduced a new dataset composed of thousands of sentences extracted from Stormfront, a white supremacist forum. Using a new custom annotation tool they developed, all sentences were manually labelled as either containing, not containing, and being related to hate speech language. Based on their annotation guidelines, a sentence was categorized as hate speech if it was "a deliberate attack, directed towards a specific group of people, motivated by aspects of the group's identity". Sentences that did not fall under the above definition were categorized as No Hate. Finally, the label "Relation" was created for specific cases where the sentences in a post did not contain hate speech on their own, but the combination of several sentences did. The dataset was introduced with the purpose of being used to train models that can detect hate speech language, treating the problem as a supervised classification task. To the best of my knowledge, it is the only publicly available dataset containing posts from a white supremacist forum, annotated at sentence level. A set of baseline experiments were conducted by the authors, to check the validity of the annotations, based on a balanced subset of sentences. The three algorithms that were evaluated were Support Vector Machines utilizing Bag-of-Words vectors, Convolutional Neural Networks and Recurrent Neural Networks with Long Short-Term Memory using randomly initialized word embeddings. The experiments revealed that the LSTM model with the randomly initialized word embeddings achieved the best performance, returning an overall accuracy of 73%, 71% for the hate class and 75% for the no hate class, when including sentences that required additional context for manual annotation. The model achieved an overall accuracy of 78%, 76% for the hate class and 80% for the no hate class, when excluding sentences that required additional context for manual annotation. Hyperparameters were left to the values reported in

the literature, leaving tuning, experimentation, and research with word knowledge and context for future work.

An attempt to take advantage of the context and Transfer Learning to improve these results was done by (Alatawi, Alhothali and Moria, 2021) in "Detecting White Supremacist Hate Speech Using Domain Specific Word Embedding With Deep Learning and BERT". They applied two approaches in their work: First, they experimented with a Bidirectional Long Short-Term Memory classifier, comparing domain-agnostic (e.g., Word2Vec, GloVe.Wikipedia, and GloVe.Twitter) and domain-specific word embeddings (Word2Vec) learned from a collection of one million tweets from white supremacist accounts and hashtags. This approach was evaluated on multiple datasets, including the Stormfront dataset, with the domain-specific embeddings outperforming the other three methods, achieving F1-scores ranging from 0.49 to 0.75. For their second approach, they used the pre-trained language model BERT with a dense neural network layer, which they fine-tuned using a subset of the one million tweets dataset. The fine-tuned BERT model was evaluated on the same datasets as their first approach, achieving F1-scores ranging from 0.59 to 0.80, making it their top-performing model. The model outperformed the LSTM with randomly initialized word embeddings reported in (de Gibert *et al*., 2018), using the Stormfront dataset and excluding sentences that required additional context for manual annotation. They achieved an overall accuracy of 84%, 82% for the hate class, 86% for the no hate class, and an F1-score of 0.84, proving that the BERT model can assign closer, more meaningful vectors to the words due to its training strategy, improving hate speech classification tasks.

In "Automated Hate Speech Detection and the Problem of Offensive Language" by (Davidson *et al*., 2017), a variety of traditional machine learning models were tested in a multi-class classification problem, using tweets that were classified as either hate speech, offensive, and no hate speech language. Dataset features were extracted using unigrams, bigrams, and trigrams, each weighted by TF-IDF. They found that Logistic Regression and Support Vector Machines performed better than the rest of the models and decided to conduct the rest of their experiments with the former. Results were impressive for the offensive and no hate class, however, the model performed poorer in classifying the hate class, which is a phenomenon seen in many studies.

In “Hate Speech Detection Using Multi-Channel Convolutional Neural Network” by (Naidu and Kumar, 2021), a way of combining multiple channels and deep learning layers was proposed, to improve the efficiency of the existing base models. They introduced a three-channel Convolutional Neural Network implementation, with each channel containing an input layer, embedding layer, the same number of convolutional layers, a max pooling layer and a flatten layer. All channels were merged and connected to a fully connected layer which returned the output. The new multi-channel network was tested on hate detection tasks using two datasets containing hateful tweets, returning promising results when compared to base CNN and LSTM implementations.

The idea of combining architectures was also seen in “Detecting Online Hate Speech Using Context Aware Models” by (Texas A&M University, USA, Gao and Huang, 2017). A dataset consisting of annotated comments extracted from the Fox News website was used for their trials. Comments were labelled as either hate or no hate. Two types of hate speech detection models were proposed: a logistic regression model with context features, and a bidirectional Long Short-Term Memory model with word2vec embeddings. Both models outperformed a strong baseline by 3% and 4% in F1-score, however, combining these two models further improved the performance by another 7%. This further proved that combining different model architectures in hate speech detection tasks can yield promising results.

There have been more and more studies recently that have been using transformer-based models, like BERT, to get better performance in hate speech detection. In (Swamy, Jamatia and Gambäck, 2019), the authors experimented with four different approaches. Among them was an ELMo approach, where they used the model for feature extraction passing its output through an LSTM and a dense layer, and a BERT approach, where they loaded a pre-trained model and fine-tuned it. Both approaches were compared with a Support Vector Machine model utilizing TF-IDF vectors and an LSTM model utilizing GloVe vectors. Results revealed that BERT yielded the best results out of all approaches, followed by the LSTM with GloVe vectors.

Finally, in (Mozafari, Farahbakhsh and Crespi, 2020), a novel transfer learning approach based on a pre-trained BERT model was introduced. The authors experimented with different strategies by passing the output of BERT through Nonlinear, BiLSTM, and CNN layers,

measuring each strategy's performance on hate speech detection using two datasets. According to them, "inserting a CNN to pre-trained BERT model for fine-tuning on downstream task provides the best results in both datasets and it clearly exceeds the baselines".

2.3. Deep Learning

Deep Learning is a subfield of Artificial Intelligence and a type of machine learning based on artificial neural networks (McCulloch and Pitts, 1943) that utilizes multiple layers of processing that can perform complex operations. The field of Deep Learning can be characterized as "data hungry" and has become extremely popular in the last few years, due to the great amount of data that is being generated each day. Deep Learning is responsible for the progress of many difficult tasks such as image recognition, language translation, and speech recognition.

2.4. Natural Language Processing

Natural Language Processing is described in (Chowdhury, 2003) as "an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things." Any kind of computer manipulation of the natural language can be described as NLP (Bird, Klein and Loper, 2009). It has become a prominent field with many applications, such as machine translation, question answering, and text classification (Manning and Schütze, 1999). One of the most common tasks of this field is text classification, also known as sentiment analysis, where the sentiment of one or multiple sentences is predicted using traditional supervised machine learning, deep learning, and the more recent Transformer-based models.

2.5. Traditional Word Embeddings

Computers do not understand text the same way as humans do, as they can only understand numbers. The first step in any NLP task is to transform the data (the text in the case of sentiment analysis) into a form that can be recognized and manipulated by a computer. This transformation came in the form of word representations, or word embedding methods. These methods can be categorised into two groups: Traditional word embedding models (e.g., bag-of-words, word2vec, GloVe) and the more recent contextualised word embedding models (e.g., ELMo, BERT).

2.5.1. Bag-of-words

Bag-of-words is a basic model widely used for text classification problems due to its simple and low-cost computation. Bag-of-words was first introduced as a linguistic context in "Distributional Structure" (Harris, 1954) and was much later seen in NLP tasks. The model

creates fixed-sized vectors from text by defining a vocabulary, which is a set of all the words found in the document, and counting how many times each word appears. The result representations can then be passed into a machine learning model to carry out NLP tasks.

2.5.2. Word2Vec

The Word2Vec method was introduced in "Linguistic Regularities in Continuous Space Word Representations" (Mikolov, Yih and Zweig, 2013) as an attempt to extract better word representations from text. The traditional word count representation was replaced with word vectors that represented the words' relationships. The cosine similarity between the vectors describes the semantic similarity between the words represented by the vectors. The authors stated in their paper that "these representations are surprisingly good at capturing syntactic and semantic regularities in language and that each relationship is characterized by a relation-specific vector offset."

2.5.3. GloVe

GloVe (Global Vectors for Word Representation) was introduced in "Glove: Global Vectors for Word Representation" (Pennington, Socher and Manning, 2014) as a new method to create better word representations. Word vectors are obtained using an unsupervised learning algorithm trained on aggregated global word-word cooccurrence statistics from a corpus (*GloVe: Global Vectors for Word Representation*). As stated by the authors, the GloVe model "combines the advantages of the two major model families in the literature: global matrix factorization and local context window methods". Like Word2Vec, it maps words into a meaningful vector space where the distance between words is related to semantic similarity (Abad *et al.*, 2016).

2.6. Contextualized Word Embeddings

The rise of more complex NLP tasks and the need to solve them introduced newer methods in 2018 that achieved state-of-the-art results, such as ELMo and BERT. Unlike the traditional context-independent word embedding methods, these models were able to capture the context of a word within a sentence and generate different vectors for a word, based on its position in the sentence.

2.6.1. ELMo

ELMo (Embeddings from Language Models) was introduced in "Deep contextualized word representations" (Peters *et al.*, 2018) as "a new type of deep contextualized word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy)". The authors

demonstrated a novel way of adding ELMo to existing models and achieving state-of-the-art results in several NLP tasks. The difference with traditional word embeddings was that because of how ELMo assigned vectors to words, the same words with different meanings could have different vectors.

2.6.2. BERT

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model introduced in "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin *et al*., 2019) and since then has achieved many state-of-the-art results in many studies. The main difference compared to other word embedding methods is that the model itself is also required when generating word vectors. As stated by the authors, the biggest advantage of the BERT model is that it can be fine-tuned using only just one additional neural network layer to create models that can achieve state-of-the-art results in difficult NLP tasks such as language translation or question answering.

2.7. Transfer Learning

Transfer learning is the ability to take models that have been pre-trained on larger datasets and specific problems, apply them to different problems, and achieve similar results. In the field of NLP, transfer learning has become very popular following the introduction of context-aware models, especially BERT. Many studies utilize pre-trained word representation models that have been trained on very large corpora, to extract word vectors, save time and resources and achieve great results.

Chapter 3. Methods

3.1. Introduction

This section describes in detail the methods that were necessary for conducting this research, in line with the objectives mentioned in section 1.3. The research attempts to answer the two research questions, as described in section 1.3., and test the two hypotheses below:

1) Replacing a model's traditional word representation methods with newer pre-trained contextualized word embedding methods will improve automatic hate speech detection across the board, taking advantage of Transfer Learning and understanding of context.
2) The end product can itself take advantage of Transfer Learning, being able to perform decently in similar problems as it is, or by fine-tuning it on the new dataset.

3.2. Structure of the Work

The structure followed in this section was based on the work plan as shown in section 1.6., following the literature review. First, section 3.3. describes briefly the main dataset used in this research, along with the way it was loaded and the pre-processing techniques that were applied with the Natural Language Toolkit (NLTK) python library. Second, section 3.4. describes how the three models, Support Vector Machine (SVM), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM), and their results were able to be reproduced as reported by (de Gibert *et al*., 2018) in their paper. Section 3.5., describes how the four major pre-trained word embedding methods (i.e., Word2Vec, GloVe, ELMo, and BERT) that were selected during the literature review were applied to all models, including Support Vector Machine which required an extra process, due to its architecture and the lack of an embedding layer, compared to the other two neural networks. In section 3.6., the top two performing models were selected and their hyperparameters were tuned using the HyperBand algorithm (Li *et al*., 2018). During this process, a third hybrid model was also created, tuned, and tested, combining the network architecture of the two models. Finally, in section 3.7., the hybrid model was experimented on a new, similar, and bigger dataset, to observe its performance and conclude whether it qualifies for Transfer Learning.

Figure 2 below shows the structure of the work, as described in this section.

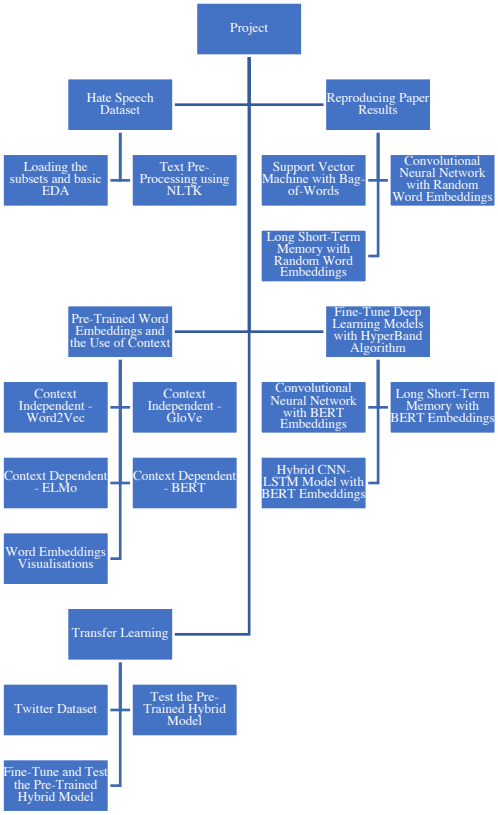

Figure 2: Structure of the work

3.3. Hate Speech Dataset

The dataset chosen for this research is the Hate Speech Dataset from a White Supremacy Forum, introduced in (de Gibert *et al.*, 2018). According to the authors, "A random set of forum posts have been sampled from several subforums and split into sentences. The sentences have been manually labelled as containing hate speech or not, according to certain annotation guidelines." The dataset was downloaded from the authors' GitHub repository.

The experiments of this research were based on a balanced subset of labelled sentences, which were provided by the authors in the sampled_train and sampled_test folders. According to the authors, "All the sentences labelled as hate have been collected, and an equivalent number of no Hate sentences have been randomly sampled, summing up 2k labelled sentences. From this amount, 80% was used for training and the remaining 20% for testing." Each sentence was saved as a separate text file. The files include sentences that required additional context during manual annotation, which provided an extra challenge for the models that were created and tested and proved the importance of introducing more recent input methods that attempt to understand the context of a sentence.

3.3.1. Loading the subsets and basic EDA

The authors' provided an annotations_metadata.csv file containing the label information of each text file name which was utilised to load and map each sentence with its label. The text files were loaded into two Pandas Data Frames, one for the training data and one for the test data.

	file_id	user_id	subforum_id	num_contexts	label
0	12834217_1	572066	1346	0	noHate
1	12834217_2	572066	1346	0	noHate
2	12834217_3	572066	1346	0	noHate
3	12834217_4	572066	1346	0	hate
4	12834217_5	572066	1346	0	noHate

Figure 3: The annotations Data Frame

The Data Frame has five columns, as shown in figure 3 on the left: file_id (the name of the text file), user_id (the unique ID of the user that created the forum post), subforum_id (the unique ID of the forum each sentence belongs), num_contexts (sentence information related to the Relation class), and label (the class each sentence belongs to). Sentences are classified into four unique labels: no Hate (the sentence does not contain hate speech content), hate (the sentence contains hate speech content), IDK/skip (non-English sentences), and relation (sentences that belong in a forum post that contains other hate speech sentences).

This research experimented with a binary classification problem, predicting if a sentence classifies as Hate Speech or No Hate Speech. For this reason, rows with IDK/skip and relation labels were dropped. Furthermore, the columns user_id, subforum_id, and num_contexts were also dropped, since only the file_id and label columns were required. A new column named sentence was created containing each sentence.

The Word Clouds depicted in figure 4 below revealed the most common words found in sentences classified as Hate Speech and No Hate Speech, with both train and test subsets having similar results. Some readers may find the below words offensive. They are mentioned only for research purposes. These words do not express my beliefs and opinions, and neither do I associate myself with hate speech.

Figure 4: Word Clouds of Hate and No Hate Speech sentences

Words like "black", "white", "Jew", "Asian", "negro", "race", and "ape", were a few of the most common words found in the Word Cloud of Hate Sentences, words that are often found in hate speech language. On the other hand, while words like "black" or "white" also appeared in the Word Cloud of No Hate Sentences, their meaning is not necessarily hate speech related, as they were among other seemingly "innocent" words like "YouTube", "school", "people", "time", "good" and others.

3.3.2. Text Pre-processing using NLTK

According to their paper, the authors conducted their experiments without exploiting any external resources such as lexicons, heuristics, or rules. Although this does not specify exactly how they pre-processed the data, I assumed that the authors did not apply any heavy text pre-processing to the sentences, when conducting their experiments. To reproduce their published results, a similar approach was taken, applying only very basic pre-processing, like lowercasing and removing punctuation and non-alpha characters. Each sentence in the training and test subsets was tokenized using the word_tokenize function from the Natural Language Toolkit library. All words that did not contain letters found in the English alphabet were filtered out, and the remaining words were put back together in sentences while also applying lowercasing. A new "clean_sentence" column was created for both train and test subsets, containing each pre-processed sentence.

3.4. Reproducing Paper Results

The purpose of this section was to recreate the results reported in the paper, use them as a benchmarking tool and compare them with the rest of the results of this research.

In the paper, the below three algorithms were used for Hate Speech Classification:

- Support Vector Machines (SVM) over Bag-of-Words vectors. Word-count-based vectors were computed and fed into a Python Scikit-learn LinearSVM classifier to separate Hate and No Hate instances.
- Convolutional Neural Networks (CNN). The implementation was a simplified version using a single input channel of randomly initialized word embeddings

- Recurrent Neural Networks with Long Short-term Memory (LSTM). An LSTM layer of size 128 over word embeddings of size 300

All the hyperparameters were left to the usual values reported in the literature. No hyperparameter tuning was performed. More comprehensive experimentation and research were left for future work. The authors did not provide the code for each of their three models, apart from a reference code for the Convolutional Neural Network model. Therefore, all three models were coded by me from scratch, using the above descriptions as a guideline.

3.4.1. Support Vector Machine with Bag-of-Words

The first model that was reproduced was the Support Vector Machine, using the Scikit Learn library and the LinearSVC function. The word representation method used to create vectors from text was Bag-of-Words, in accordance with the guidelines mentioned in 3.4.. Using the Count Vectorizer function from the Scikit Learn library (*CountVectorizer*), a bag-of-words vocabulary was created by fitting the training data. The reason why only the training subset was used to create the vocabulary was to avoid having any concerns about leaking the test data to the Support Vector Machine model. Both training and test subsets were then transformed to document-term matrices using the constructed vocabulary and were passed to the Support Vector Machine model for training and testing. This implementation was based on Scikit Learn documentation and examples.

3.4.2. Convolutional Neural Network with Random Word Embeddings

The second model that was reproduced was the Convolutional Neural Network. The code implementation referenced by the authors was written using the TensorFlow neural network library. The neural network library that I decided to utilise to carry out this research was Keras (*Keras: the Python deep learning API*), due to its customizability of layers, ease of use, and the tools it comes bundled with, such as Keras Visualisation Utilities, and HyperBand Tuner. Therefore, the neural network was coded from scratch, converting the TensorFlow implementation to Keras.

The network is a simplified implementation of Kim's CNN-Rand as described in his paper, (Kim, 2014). Figure 5 shows the whole network architecture, broken down into layers. The model was visualised using the plot_model function from Keras Visualisation Utilities. Visualising the model helped confirm the layers' order and shapes.

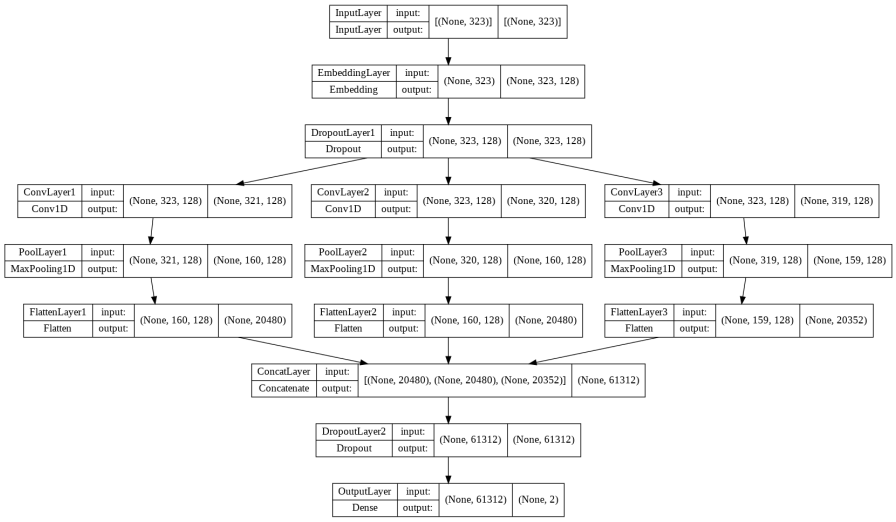

Figure 5: CNN Network architecture

The Neural Network consists of one input layer, one non-trainable embedding layer of randomly initialized word embeddings with an output dimension of 128, and one dropout layer with 0.5 dropout probability that reduces overfitting. Three pairs of Convolution with 128 filters and Pooling layers, one for each kernel size of three, four and five, were used to capture the semantic patterns of each sentence, which were then concatenated together. Finally, a second dropout layer was added followed by a dense output layer that classified each sentence. As a result, the network had 1,029,890 total parameters, with 319,618 of them being trainable.

The dataset required extra preparation before it could be used with the CNN Model. First, a vocabulary was created with Keras Tokenizer using the training subset, and all sentences were converted to lists of number sequences. Second, all sequences were post-padded to the same length, equal to the length of the longest sentence, 323. Finally, the train and test subset labels were one hot encoded.

The model was compiled with Adam (Kingma and Ba, 2017) as the preferred optimizer with a learning rate of 1e-3, Categorical Cross entropy as a loss function and Accuracy as a performance metric. A batch size of sixty-four was used to train the network for one hundred training epochs, storing, and visualising the training history. Finally, the trained model was tested on the test subset, calculating the Confusion Matrix, Precision, Recall, Accuracy, and F1 Score.

3.4.3. Long Short-Term Memory with Random Word Embeddings

The third model that was reproduced was the Long Short-Term Memory network, implementing it from scratch in Keras, based on section 3.4. guidelines. As with the

Convolutional Neural Network, figure 6 below shows the whole network architecture, broken down into layers.

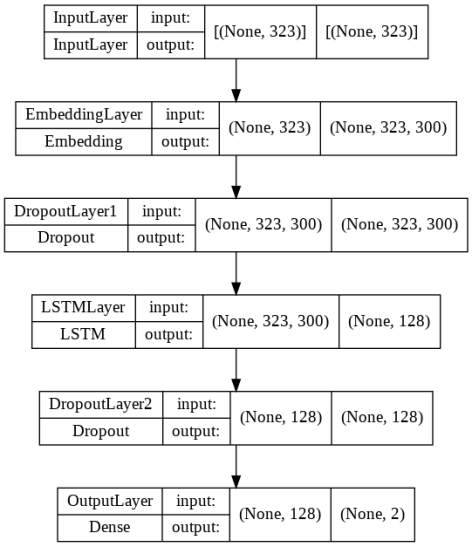

Figure 6: LSTM Network architecture

The network consists of one input layer, one non-trainable embedding layer of randomly initialized word embeddings with an output dimension of 300, and one dropout layer with 0.5 dropout probability that reduces overfitting. A single LSTM layer with 128 units was then placed, capable of predicting the long sequences of data like sentences. Finally, like the CNN network, a second dropout layer was added followed by a dense output layer that classified each sentence. As a result, the network had 1,884,606 total parameters, with 219,906 of them being trainable.

The dataset was prepared in the same way as with the Convolutional Neural Network, with the only difference being the type of padding applied to the sentences. Unlike CNN, my LSTM would only learn when pre-padding was applied to the sentences, instead of post-padding. The reason behind this behavior will be explored and discussed in section 4.5.. The model was compiled using the similar settings as the CNN, but with a smaller learning rate of 5e-5, a batch size of thirty-two and fifty training epochs. The trained model was tested on the test subset, calculating the same performance metrics as the CNN.

3.5. Pre-Trained Word Embeddings and the Use of Context

This section aimed to answer the research question "To what extent pre-trained contextualised word embedding methods improve hate speech detection?" by introducing pre-trained word embeddings to the above models, replacing the traditional bag-of-word model used in the Support Vector Machine model and the random word embeddings used in both the

Convolutional Neural Network and Long Short-Term Memory models. First, the experiments were conducted using the context-independent models Word2vec and GloVe. Their performance was then compared with that of the context-dependent models ELMo and BERT.

3.5.1. Context Independent – Word2Vec

The Word2Vec method replaces the classic word count method, by learning and representing words' relationships and placing them in a continuous vector space. There are two methods to obtain Word2Vec vectors. Either by creating a Word2Vec model and training it from scratch or by loading an existing model, pre-trained on a big corpus. Since this research's aim was to experiment with pre-trained word embeddings, the second method was selected (Keras, pre-trained word embeddings). The model was imported from the Gensim library and was trained on part of the Google News dataset (about one hundred billion words), containing 300-dimensional vectors for three million words and phrases.

The vectors were utilized by creating a corresponding embedding matrix that could be used in the Keras Embedding layer. Like before, a vocabulary was created first with the Keras Tokenizer function. Each entry at index i of the embedding matrix was the pre-trained vector for the word of index i in the vocabulary. A total of 494 words from the vocabulary were missing from the pre-trained vectors.

Training and test data were prepared in a similar manner as before. Sentences were converted to a list of sequences and were pre-padded to a common length, only this time a different length size was selected. It is common practice to pad all sentences to the max sentence length. However, figure 7 on the left shows that almost all the sentences have a length between zero and fifty, with a small number of sentences going up to one hundred. Therefore, the max sentence length used for applying padding was changed to one hundred.

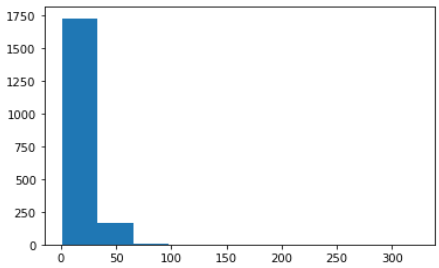

Figure 7: Sentences length

The embedding matrix was used in the embedding layer of both Convolutional Neural Network and Long Short-Term Memory Keras Models, replacing the previous randomly initialized vectors. Both networks' architecture remained the same, adjusting slightly the dropout keep probability, number of convolution filters and the output dimension of the LSTM layer. A

learning rate scheduler was introduced for both models, that handled the adjusting of the learning rate during training. The learning rate was first initialized with a higher number to prevent the optimizer from getting stuck at a local minimum which then started decaying exponentially to minimize the movement and find the optimal value.

Unlike the Keras Deep Learning Networks, the Scikit learn Support Vector Machine model does not incorporate an embedding layer. Therefore, the mapping of the word embedding vectors to the training and test data was applied manually. For every word found in every padded sentence in the list of sequences, the word's vector was extracted from the embedding matrix and appended to a list, which was then transformed into a NumPy array. The returned array had a three-dimensional shape of (number of sentences, sentence length, 300-dimensional word vector). Because Support Vector Machines cannot handle three-dimensional data, each 300-dimensional word vector was averaged before training and testing the model.

3.5.2. Context Independent – GloVe

The GloVe word embeddings were introduced as an extension to the Word2Vec method as an attempt to map words into a more meaningful vector space. Like the Word2Vec method, a pre-trained existing model was used to extract the vectors (Keras, pre-trained word embeddings). The 200-dimensional word vectors were downloaded from the Stanford website and were pre-trained on the combined Wikipedia 2014 and Gigaword 5th Edition corpora (6B tokens, 400K vocabulary size).

Despite the difference in the dimension of the vectors, the process of utilizing the GloVe word embeddings and the experimentation with the three models was identical to the Word2Vec method. A corresponding embedding matrix was created and used to initialize the embeddings of the Keras embedding layers. A total of 313 words from the vocabulary were missing from the pre-trained vectors. Training and test data were prepared in a similar manner, manually mapping each word to its corresponding GloVe vector, and reducing its dimensionality so it can be used by the Support Vector Machine.

3.5.3. Context Dependent – ELMo

Unlike Word2Vec and GloVe embedding methods, each ELMo vector assigned to a word is a function of the entire sentence containing that word. This means that the ELMo model considers the word order of a sentence, subsequently understanding the context and polysemy of words. Since ELMo generates each word vector based on the context of a sentence, the

model that was used to train the vectors is also required, even after training, when utilizing the word embeddings. It is possible to just use the raw trained vectors, however, this would defeat the purpose of ELMo. For my research, I used the whole model, passing to it the hate speech sentences and feeding its output to the Support Vector Machine, Convolutional Neural Network, and Long Short-Term Memory models.

The pre-trained ELMo model was loaded from TensorFlow Hub, a library containing a variety of machine learning models used for Transfer Learning. The model was trained by Google on the one-billion-word benchmark. Since ELMo was used only for its raw output, it was not fine-tuned. This was achieved by freezing its weights and setting the trainable flag to False. The model was loaded with hub.Module, a function that works only with TensorFlow version 1. Since TensorFlow version 2 was utilized throughout the whole project, version 2 behaviour had to be disabled just for the ELMo section.

ELMo computes its vectors by receiving batches of tokenized sentences and their sequence length. First, all sentences were tokenized and padded to a length of one hundred. Next, they were split into batches of sixty-four sentences per batch. Finally, each batch was passed to the ELMo model using a custom function and a for loop, returning the contextualized word embeddings in a three-dimensional NumPy array with a shape of (number of sentences, sentence length, 1024-dimensional word vector). Like before, because Support Vector Machines cannot handle three-dimensional data, each 1024-dimensional word vector was averaged before training and testing the model (Joshi, 2019). This whole process was time-consuming, therefore the prepared datasets were exported as PyTorch Tensors, ready to be loaded at one's convenience.

The Support Vector Machine model was trained and tested with the ELMo embeddings in the same way as before. Regarding the two Keras Neural Network models, since this time their input data was already in their word embedding form, their Keras Embedding layer was dropped from their network architecture. The rest of the networks' architecture and hyperparameter values remained the same, apart from the training epochs which were decreased, because the networks would start to overfit. As a result, the optimiser was also changed to Adam with weight decay (AdamW) (Loshchilov and Hutter, 2019), since it was performing better than Adam with the learning rate scheduler when trained for fewer epochs.

3.5.4. Context Dependent – BERT

Like the ELMo word embeddings method, BERT word vectors are generated based on the context of a sentence. The BERT model can be used in three different ways:

1) It can be trained completely from scratch and used as a classifier itself
2) Its pre-trained word vectors can be extracted and used in a Keras embedding layer, similar to how Word2Vec and GloVe vectors were used
3) A pre-trained model can be fine-tuned using its output as input for a Deep Learning model

The third option was deemed as the most appropriate for this research, taking full advantage of Transfer Learning. The pre-trained model was loaded from the Hugging Face Transformers package, using the TFBertModel.from_pretrained function, which is compatible with Keras (Pietro, 2022). The BASE BERT model was used, a smaller version which has 12 transformer blocks, a hidden layer size of 768, and 12 attention heads. Since BERT was used only for its "raw" output, its weights were frozen by setting the trainable flag equal to False.

Data preparation this time was slightly different compared to before, since the data needed to be transformed into a format that BERT recognized and was trained on. More specifically, the BERT model required:

- A special separator [SEP] token that marks the end of a sentence or the separation between two sentences
- A special class [CLS] token at the beginning of each text, used for classification tasks
- Tokens that comply with BERT's fixed vocabulary
- The Token IDs from BERT's tokenizer
- The Mask IDs to distinguish tokens from padding elements
- The Segment IDs to distinguish different sentences
- The Positional Embeddings that show each token's position within the sentence

All of the above requirements were taken care of using the tokenizer.encode_plus function from the Transformers BertTokenizer package on each sentence. The function returned two feature matrices, containing the encoded sentences and the attention masks, serving as the actual inputs of the BERT model.

The word embeddings that were used to train and test the Support Vector Machine were extracted using the bert.embeddings method and passing the encoded sentences and attention

masks as input. Like before, since the method output was three-dimensional vectors with a shape of (number of sentences, sentence length, 768-dimensional word vector), each 768-dimensional word vector was averaged before training and testing the model. Regarding the two Keras Neural Network models, the embedding layer was replaced by the whole pre-trained BERT model which was added to each network's architecture. BERT returns two outputs when called: the last hidden state which contains batches of sequences with a shape of (batch size, sentence length, 768-dimensional vector) and the pooler output. The last hidden state was the one that was passed on through the next layers since it contained the actual sentences. Each network's input layer was also adapted to accept the encoded sentences and attention masks. Figure 8 below shows the newly added layers that replaced the previous input and embedding layers of both Keras Neural Network models.

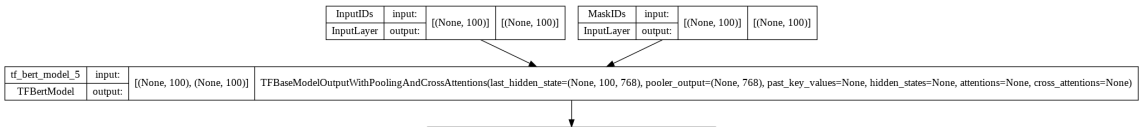

Figure 8: The Neural Networks' new input and BERT layers

The rest of the networks' architecture and hyperparameter values remained the same, apart from the training epochs which were decreased to four for the Convolutional Neural Network model and two for the Long Short-Term Memory model, as suggested in (Devlin *et al*., 2019). Higher training epochs would result in overfitting. Like with the ELMo models, the optimiser was changed to Adam with weight decay (AdamW), for the same reasons.

3.5.5. Word Embeddings Visualisations

For each of the four pre-trained word embedding methods, their output vectors and the similarity between objects were visualized using the t-SNE method. This method takes a group of high-dimensional word vectors and compresses them down to two-dimensional coordinate pairs that can be visualized in a two-dimensional plane. Ideally, similar words will be close together on the plane, while dissimilar words will have a longer distance between them. The process was similar for all four methods, with the only change being the way each method produced its vectors. The top fifty most common words found in hate and no Hate sentences were visualised in a two-dimensional plane, using colour to distinguish them based on their label.

My implementation of the above was done as such: First, all training sentences were divided based on the class they belonged to. Second, every word of every sentence that wasn't an

English stop word was added to a list, creating one list for each class. Then, using a Python Counter, the top fifty most common words found in each list were stored in new lists. Next, for every top word, its word vector was extracted using each of the four word embedding methods. For the Word2Vec and Glove embeddings, the get_vector function and embedding matrix were utilised respectively. For the ELMo and BERT embeddings, the process was more difficult since these models output word vectors by taking as input the whole sentence. First, word vectors were extracted for the two groups of sentences. Then, each of the top fifty words was searched in each embedded sentence, extracting the first resulting vector before moving to the next word.

For each of the four word embedding methods, the final vectors were created by reducing the dimensionality of both classes' vectors using the t-SNE method. A Pandas Data Frame was created containing the one hundred words, their vectors, and the class they belong to. Duplicate words i.e., common words that appeared in both classes, were assigned a different label to distinguish them during visualization. Finally, the words were visualized with a scatterplot, using different colours based on the class they belonged to.

3.6. Fine-Tune Deep Learning Models with HyperBand Algorithm

Out of all fifteen models, the BERT variations of the Convolutional Neural Network and Lon Short-Term Memory models demonstrated great potential, with each model performing slightly better in each one of the classes than the other. A third hybrid BERT CNN-LSTM model was created, combining the two architectures and their advantage in each of the two classes. The purpose of this section was to fine-tune the three models using Keras Hyperband Tuner and find which of them performed the best in the Hate Speech Classification problem.

The Hyperband algorithm (Li *et al*., 2018) focuses on speeding up random search through adaptive resource allocation and early stopping. To achieve that, Keras Tuner requires a model-building function to be passed that specifies the hyperparameters which Keras Tuner aims to tune and builds the model. During each trial, the Tuner creates an instance of the model using a random combination of the provided list of hyperparameters, and trains it for a specific number of training epochs while monitoring and storing a specific metric (*Introduction to the Keras Tuner*).

The tuned model that yielded the best results was saved as a Keras model, ready to be loaded at one's convenience.

3.6.1. Convolutional Neural Network with BERT Embeddings

The BERT Convolutional Neural Network model hyperparameters the Tuner focused on optimizing were:

- The dropout keep probability of the Dropout Layer, searching for optimal values between 0 and 0.5, with a step of 0.1
- The number of filters in the three Convolution Layers, searching for optimal values between 32 and 128, with a step of 32
- The kernel sizes of the three Convolution Layers, searching for optimal values between 2 and 3
- The decay rate of the AdamW optimizer, searching for optimal values between 0.001, 0.25, and 0.5
- The learning rate of the AdamW optimizer, searching for optimal values between 6e-5, 5e-4, 1e-4, and 1e-3

The hyperparameter values that were tested were selected based on the literature review.

The Hyperband tuner was instantiated with the objective of monitoring the changes in validation loss during each epoch. An early stopping criterion was defined that monitored the validation loss and stored the hyperparameter combination that returned the minimum value. The early stopping would activate if the best-found validation loss would not change for five consecutive trials. A new model was created using the best hyperparameter combination and was trained for two training epochs. The model was then tested one final time using the test subset. Figure 9 below shows the full network architecture of the Hyperband Tuned BERT Convolutional Neural Network model.

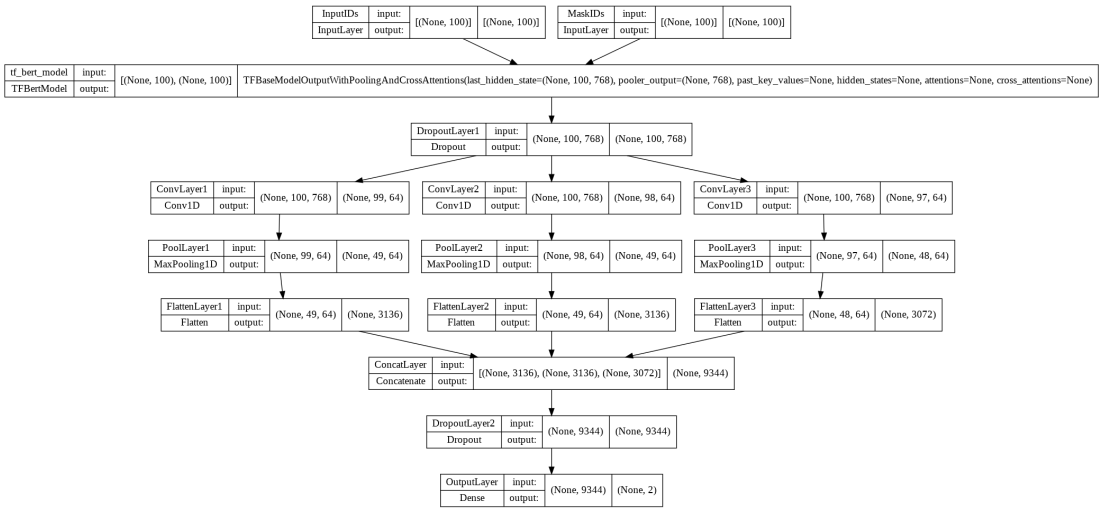

Figure 9: Architecture of the tuned BERT Convolutional Neural Network model

3.6.2. Long Short-Term Memory with BERT Embeddings

The BERT Long Short-Term Memory model hyperparameters the Tuner focused on optimizing were:

- The dropout keep probability of the Dropout Layer, searching for optimal values between 0 and 0.5, with a step of 0.1
- The number of the LSTM layers, searching for optimal values between 1 and 2
- The type of the LSTM layers, searching for the optimal type between Unidirectional and Bidirectional
- The number of units in the LSTM layers, searching for optimal values between 32 and 128, with a step of 32
- The decay rate of the AdamW optimizer, searching for optimal values between 0.001, 0.25, and 0.5
- The learning rate of the AdamW optimizer, searching for optimal values between 1e-4, 6e-5, 5e-4, and 1e-3

The hyperparameter values that were tested were selected based on the literature review.

Like the Tuned BERT Convolutional Neural Network, the Hyperband tuner was instantiated with the objective of monitoring the changes in validation loss during each epoch. The same early stopping criterion was defined, monitoring and storing the validation loss, activating when the best-found validation loss would stop changing for five consecutive trials. The new model was created with the best hyperparameter combination and was trained for two training

epochs. The model was then tested one final time using the test subset. Figure 10 below shows the full network architecture of the Hyperband Tuned BERT Long Short-Term Memory model.

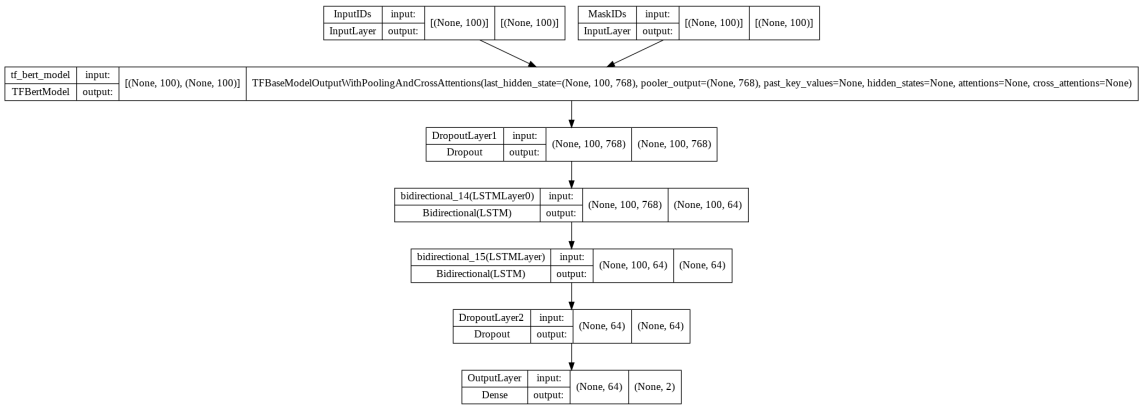

Figure 10: Architecture of the tuned BERT Long Short-Term Memory model

3.6.3. Hybrid CNN-LSTM Model with BERT Embeddings

A new hybrid model was tuned, trained, and tested that combined the architecture of both Convolutional Neural Network and Long Short-Term Memory models. The dropout layer that was placed after the BERT layer was dropped, to retain the full features of the word embeddings. The last hidden layer of the BERT model was passed to two convolution layers with kernel size two and three that captured the features of two and three consecutive words respectively. Each of the two layers' output was then pooled and passed to an LSTM layer. The two LSTM layers were then concatenated together leading to a Dense layer that handled the classification.

The Hybrid BERT CNN-LSTM model hyperparameters the Tuner focused on optimizing were:

- The dropout keep probability of the Dropout Layer, searching for optimal values between 0 and 0.5, with a step of 0.1
- The number of filters in the two Convolution Layers, searching for optimal values between 32 and 128, with a step of 32
- The type of the LSTM layers, searching for the optimal type between Unidirectional and Bidirectional
- The number of units in the LSTM layers, searching for optimal values between 32 and 128, with a step of 32
- The decay rate of the AdamW optimizer, searching for optimal values between 0.001, 0.25, and 0.5

- The learning rate of the AdamW optimizer, searching for optimal values between 1e-4, 6e-5, 5e-4, and 1e-3

The hyperparameter values that were tested were selected based on the literature review.

Like the two previous Tuned BERT models, the Hyperband tuner monitored the changes in validation loss during each epoch, early stopping when the best-found validation loss would stop changing for five consecutive trials. The new model was created with the best hyperparameter combination and was trained for two training epochs. The model was then tested one final time using the test subset. Figure 11 below shows the full network architecture of the Hyperband Tuned Hybrid BERT CNN-LSTM model.

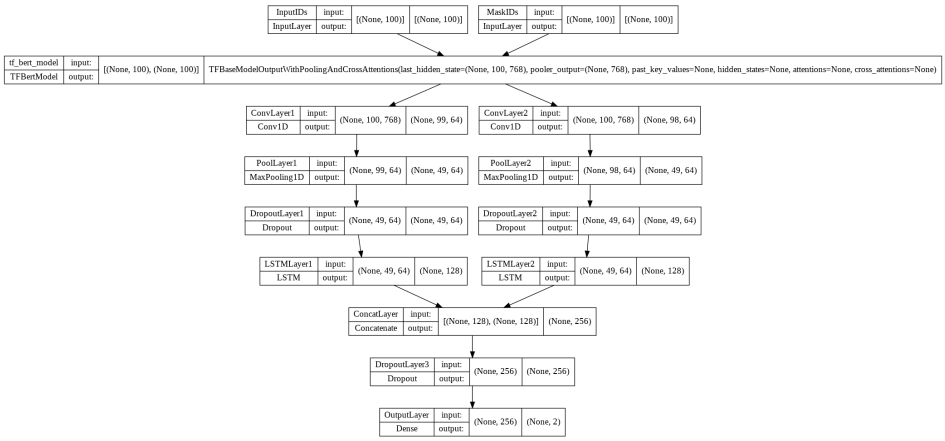

Figure 11: Architecture of the tuned Hybrid BERT CNN-LSTM model

3.7. Transfer Learning

To test the robustness of the best-tuned model and whether it qualified for Transfer Learning, a second test was conducted using two bigger, combined datasets containing tweets that can be classified as either Hate or No Hate. The datasets were downloaded from the repositories for the papers "Automated Hate Speech Detection and the Problem of Offensive Language", ICWSM 2017 (Davidson *et al.*, 2017) and "Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior" (Founta *et al.*, 2018) respectively. A final test was conducted, fine-tuning the model first with a training subset of the Twitter dataset and then measuring its performance again on a test subset.

3.7.1. Twitter Dataset

The two datasets were loaded individually first to bring them to a format that allowed them to be combined and preprocessed together. The first dataset was loaded from a CSV file into a Pandas Data Frame. The dataset was created in a format that was convenient to me, as shown in figure 12 below.

	count	hate_speech	offensive_language	neither	class	tweet
0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

Figure 12: The first Twitter Dataset, as provided by (Davidson et al., 2017)

The Data Frame had six columns: count, hate_speech, offensive_language, neither, class, and tweet. Only the class and tweet columns were required for conducting the experiments, therefore the rest columns were dropped. The Data Frame contained 24783 Tweets and three classes. Class 0 represented hate speech, Class 1 the offensive language, and Class 2 neither of the two. Since this research was focused on classifying whether a sentence (or Tweet in this case) is Hate Speech or not, all tweets that belonged to class 1 were dropped altogether, to convert this to a binary classification problem. A total of 5593 tweets remained after the removal of class 1, classified as either Hate or No hate. This number was considerably smaller compared to before however, the dataset was still considered a good benchmark for the tuned model since it had a bigger size than the initial dataset used for training the model.

Loading the tweets from the second dataset required more effort compared to the first one, as each data entry contained the class and the tweet id instead of the actual tweet message. Utilizing the Twitter API, tweet messages were fetched from Twitter's servers using their tweet ID. A Twitter developer account had to be created first to get access to the tokens required by the API. The fetching functions were downloaded from the Twitter Developer repository ('Twitter API v2 sample code', 2022) and were adapted to work with my research. The dataset initially had five classes: abusive, normal, hateful, spam and nan. Like the first twitter dataset, only the classes normal and hateful were kept, making it appropriate for this research, and were renamed to match the classes of the first dataset. A total of 56470 data entries remained after the removal of the non-relevant classes. The function that handled the fetching of the tweets took a little over two hours to complete in Google Colab Pro+. For this reason, the results were saved to a CSV file, ready to be loaded at one's convenience. The API was able to fetch only 3846 tweets out of the 56470. The rest of the tweets must have been either deleted by their users or taken down by Twitter moderators. This is the main disadvantage of providing the tweet ids instead of the tweet messages, with the advantage being the considerably smaller size of the dataset.

The two datasets were combined, concatenating them into one Pandas Data Frame. The Word Clouds in figure 13 below show the most common words found in Hate Speech and No Hate Speech tweets.

Figure 13: Word Clouds of Hate and No Hate Speech tweets

Words like "bitch", "white", "nigger", "faggot", and "nigga" were a few of the most common words found in the Word Cloud of Hate tweets, words that are often found in hate speech language. While words like "yellow" or "trash" also appeared in the Word Cloud of No Hate tweets, their meaning is not necessarily hate speech related, as they were among other seemingly innocent words like "bird", "people", "game", "ghetto" and others. Some of these words might be offensive to some people but do not qualify as Hate Speech. While the Hate Tweet Word Cloud was not the same as the Hate Forum Word Cloud, many of these words had a similar meaning, which made for a good benchmark to see whether the tuned model was able to understand the context of these words, with the help of BERT.

Before feeding the tweets into the tuned model, they had to be pre-processed accordingly first. Apart from standard Natural Language Toolkit techniques like lowercasing and keeping only words with alphabet letters, tweet-specific pre-processing techniques were applied, like the removal of URLs, username tags, hashtags, the retweet tag (RT), and the ampersand character (&amp). This was achieved using regular expressions. A new "clean_tweet" column was created containing each tweet processed according to the above methods. Lastly, because the Data Frame was heavily imbalanced, with 82.91% of the total tweets not being related to Hate Speech and 17.09% of them qualifying as Hate Speech, a random under sampler was applied, bringing the total number of class-balanced tweets to 3,216.

3.7.2. Test the Pre-Trained Hybrid Model

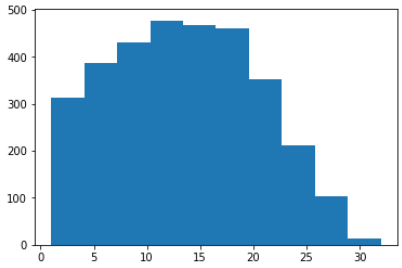

Figure 14: Tweet messages length

For the first test, the tuned model was used as it was. The data frame was prepared accordingly to work with the tuned BERT model. According to the histogram shown in figure 14 on the left, the tweets have a length between zero and twenty-five, with a small number of tweets going up to thirty. However, since the tuned model only accepts sequences with a length of one hundred, the tweets were also padded to the same length.

3.7.3. Fine-Tune and Test the Pre-Trained Hybrid Model

For the second test, the tuned model was further fine-tuned on the Twitter dataset. First, the dataset was split into training and test subsets and each subset was prepared accordingly to work with the model. Then, the tuned model was trained for one epoch, using a batch size of sixty-four. Finally, both the tuned and fine-tuned models were tested on the new test set, comparing their results with each other.

Chapter 4. Results

4.1. Recreating Paper Results

Despite not having access to the original code used in (de Gibert *et al.*, 2018), I was able to reproduce the results published in the paper. Each model's performance in the paper was measured using only the accuracy metric. In my implementation, I also included the Confusion Matrix and the Precision, Recall, and F1 Scores for each of the two classes to better evaluate the models.

The Bag-of-Words Support Vector Machine Model did a decent job in correctly classifying both classes, having an overall accuracy of 73%, an accuracy of 72% for the Hate Class, and 74% for the No Hate Class. The results were in line with the original Support Vector Machine results reported in the paper, which returned an overall accuracy of 71%, 69% for the Hate Class, and 73% for the No Hate Class. There is a small deviation between the returned and reported results which could be due to the different pre-processing techniques that were applied to the dataset before training.

The Convolutional Neural Network model with randomly initialized word embeddings did a decent job in correctly classifying No Hate sentences but performed poorly in classifying Hate sentences, having an overall accuracy of 66%, a Hate accuracy of 57%, and a No Hate accuracy of 74%. The randomly initialized word embeddings and the convolution layers failed to capture the content of the sentences that contain Hate Speech. The model was unable to distinguish the different meanings of the two classes, misclassifying almost half of them as No Hate. Apart from the No Hate accuracy score, the results were in line with the original Convolutional Neural Network results reported in the paper, which returned an overall accuracy of 66%, 55% for the Hate Class, and 79% for the No Hate Class. This 5% difference could be due to the different pre-processing techniques that were applied to the dataset before training and the randomness of the initialized word embeddings. Figure 15 below shows the model's training and validation loss and accuracy, suggesting that the model started overfitting on the noHate class early, with the validation loss increasing after the first twenty training epochs.

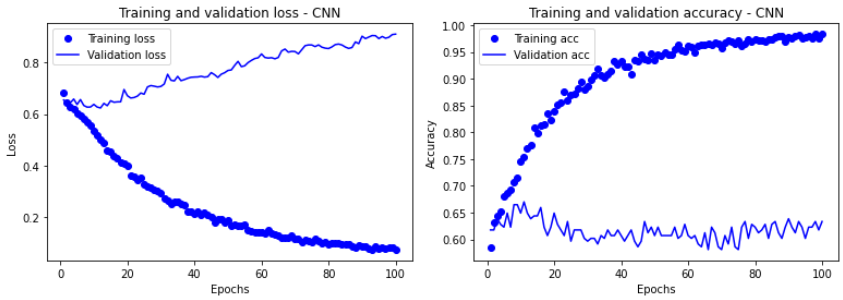

Figure 15: CNN Training and Validation Loss and Accuracy

The Long Short-Term Memory model with randomly initialized word embeddings performed decently in correctly classifying Hate and No Hate sentences, having an overall accuracy of 71%, a Hate accuracy of 67%, and a No Hate accuracy of 76%. Despite utilizing randomly initialized word embeddings like the Convolutional Neural Network model, the Long Short-Term Memory model was better at identifying, storing, and remembering the important words of each class, being able to better distinguish them, compared to Convolutional Neural Network. Apart from the Hate accuracy score, the results were in line with the original Long Short-Term Memory results reported in the paper, which returned an overall accuracy of 73%, 71% for the Hate Class, and 75% for the No Hate Class. Similarly, the small difference between returned and reported results could be due to the different pre-processing techniques that were applied to the dataset before training and the randomness of the initialized word embeddings. Figure 16 below shows the model's training and validation loss and accuracy, showing the difference in a network's validation loss when it's not overfitting.

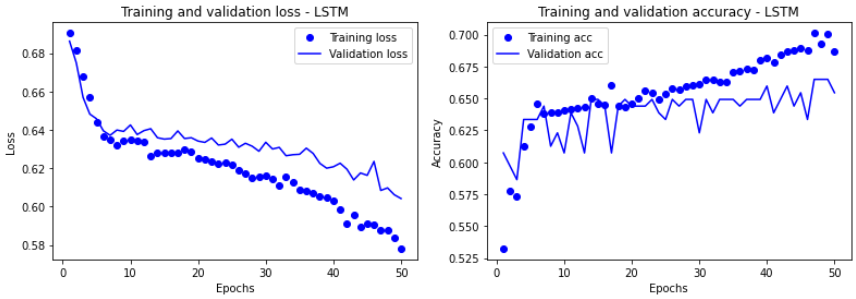

Figure 16: LSTM Training and Validation Loss and Accuracy

Figure 17 below shows the confusion matrix of each of the three reproduced models.

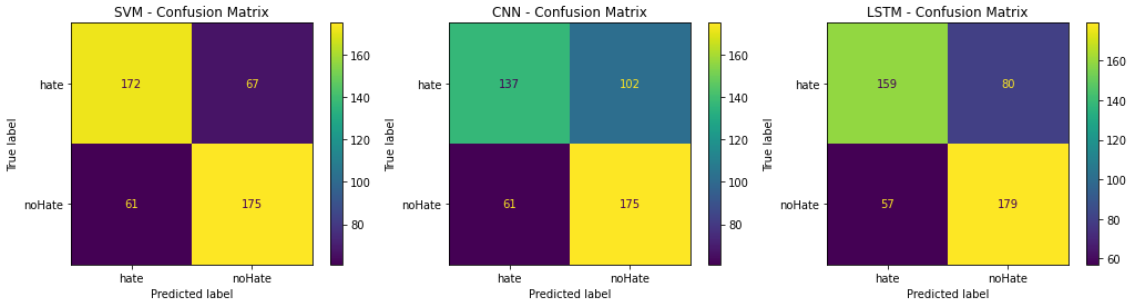

Figure 17: SVM, CNN, and LSTM Confusion Matrix

Table 1 shows performance metrics for all three reproduced models. Overall, the Long Short-Term Memory model was able to achieve the highest score numbers, with the Support Vector Machine model being the most reliable. The Convolutional Neural Network model's performance was the worst of the three.

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
SVM_{REP}	**0.72**	0.74	**0.73**	**0.74**	**0.72**	**0.72**	0.74	**0.73**	**0.73**
CNN_{REP}	0.57	0.74	0.66	0.69	0.63	0.57	0.74	0.63	0.68
$LSTM_{REP}$	0.67	**0.76**	0.71	**0.74**	0.69	0.67	**0.76**	0.70	0.72

Table 1: Metrics for the reproduced SVM, CNN, LSTM models

4.2. Word2Vec Results and Visualisation

Using Word2Vec embeddings with the Support Vector Machine model greatly improved the model's ability to correctly classify No Hate Sentences, compared to the Bag-of-Words Support Vector Machine model. However, the Word2Vec vectors hurt the model's ability to correctly classify Hate sentences. The model returned an overall accuracy of 71%, a Hate accuracy of 53%, and a No Hate accuracy of 89%. Unlike Deep Learning Neural Networks, traditional Machine Learning models like the Support Vector Machine, cannot work with three-dimensional data. The embedding vectors had to be averaged to maintain the dimensionality that works with the model, losing the spatial features of the Word2Vec vectors.

The Word2Vec embeddings improved overall the Convolutional Neural Network model's ability to correctly classify No Hate sentences, compared to the Convolutional Neural Network model with randomly initialized word embeddings. Like the Word2Vec Support Vector Machine model, it still had difficulty recognizing Hate sentences, having an overall accuracy of 69%, a Hate accuracy of 57%, and a No Hate accuracy of 81%. Although the training history

in figure 18 below suggests that the model did not overfit this time, performance in one class was much better than in the other, possibly highlighting the embeddings' weaknesses.

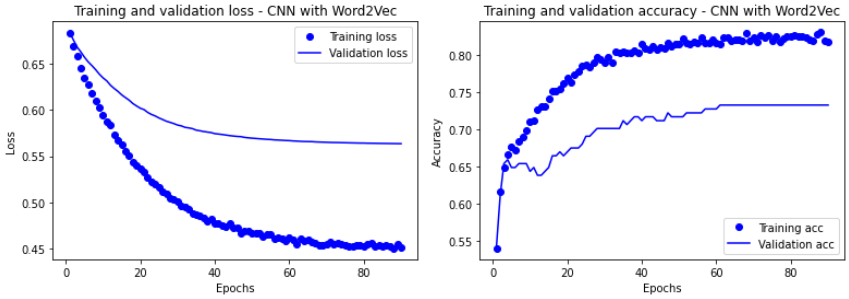

Figure 18: Word2Vec CNN Training and Validation Loss and Accuracy

Like the two models above, Word2Vec embeddings greatly improved the Long Short-Term Memory model's ability to correctly classify No Hate sentences, compared to the original model with randomly initialized word embeddings. However, it hurt the model's ability in recognizing Hate sentences, classifying correctly only half of them. The model returned an overall accuracy of 69%, a Hate accuracy of 53%, and a No Hate accuracy of 85%. The model's training history is shown in figure 19 below.

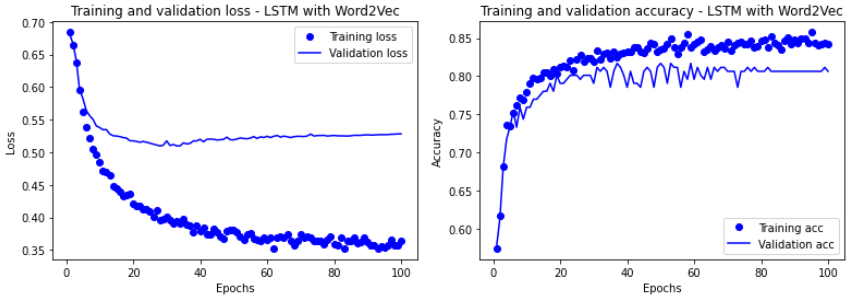

Figure 19: Word2Vec LSTM Training and Validation Loss and Accuracy

Figure 20 below shows the confusion matrix of each of the three Word2Vec model variations.

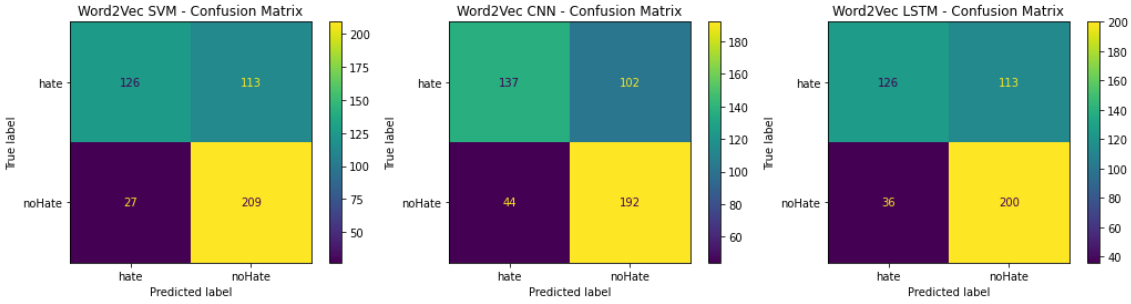

Figure 20: Word2Vec SVM, CNN, and LSTM Confusion Matrix

Table 2 shows performance metrics for all three Word2Vec variations of the models. Overall, the Support Vector Machine model was able to achieve the highest score numbers, with all the

models performing poorly in the Hate class, showing the weakness of the Word2Vec word embeddings.

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
SVM_{W2V}	0.53	**0.89**	**0.71**	**0.82**	**0.65**	0.53	**0.89**	0.64	**0.75**
CNN_{W2V}	**0.57**	0.81	0.69	0.76	**0.65**	**0.57**	0.81	**0.65**	0.72
$LSTM_{W2V}$	0.53	0.85	0.69	0.78	0.64	0.53	0.85	0.63	0.73

Table 2: Metrics for the Word2Vec SVM, CNN, LSTM models

The scatterplot in figure 21 below showed that most of the words that belonged to no Hate sentences were closer together and placed towards the middle of the plane. Word2Vec was able to capture the context of a few no Hate words, like YouTube and video, or country and world. Although these examples highlighted the advantage of using pre-trained over randomly initialised word embeddings, there were still a few words that seemed like they were placed randomly in the two-dimensional plane (e.g., children and kids, or year with years). Some of the offensive words that were found in hate sentences (e.g., black, blacks, negro, white) were placed together, however, while the words jew and jews were grouped together they were placed further away from the other group of hate speech-related words. This inability to group hate words together, was reflected in the performance of all three models.

Figure 21: Word2Vec vectors visualisation

4.3. GloVe Results and Visualisation

Using GloVe embeddings with the Support Vector Machine model improved the model's ability to correctly classify No Hate Sentences, compared to the Bag-of-Words model. However, it also misclassified more Hate Sentences than the Bag-of-Words variation. The model returned an overall accuracy of 74%, a Hate accuracy of 67%, and a No Hate accuracy of 81%. Like before, embedding vectors had to be averaged to maintain the dimensionality that works with Support Vector machines, losing the spatial features of the GloVe vectors.

The GloVe embeddings improved overall the Convolutional Neural Network model's ability to correctly classify both Hate and No Hate sentences, compared to the model with randomly initialized word embeddings. Although the model's performance in classifying No Hate sentences did not change, its ability to recognise and classify Hate sentences improved greatly. The model returned an overall accuracy of 72%, a Hate accuracy of 72%, and a No Hate accuracy of 71%. Its training history is shown in figure 22 below.

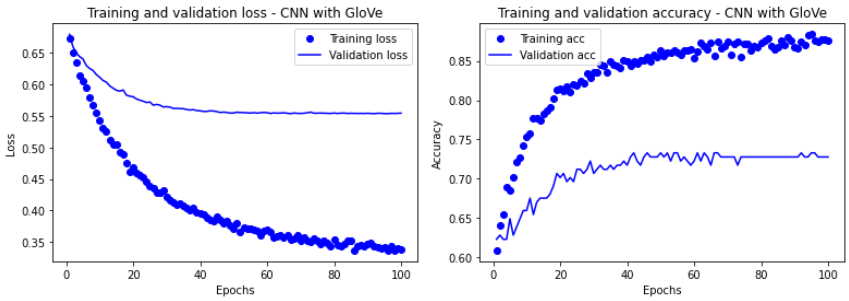

Figure 22: GloVe CNN Training and Validation Loss and Accuracy

Using Pre-Trained GloVe embeddings with the Long Short-Term Memory model did not affect the model's ability to correctly classify No Hate sentences, compared to the model with randomly initialized word embeddings. However, like the GloVe Convolutional Neural Network, the model was able to better understand the words that were related to hate speech, returning an overall accuracy of 74%, a Hate accuracy of 72%, and a No Hate accuracy of 75%. The model's training history is shown in figure 23 below.

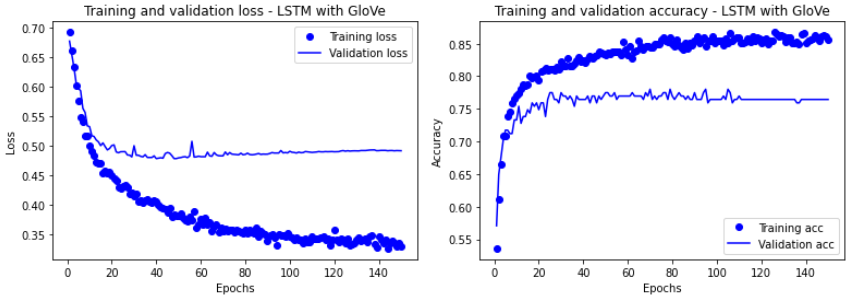

Figure 23: GloVe LSTM Training and Validation Loss and Accuracy

Figure 24 below shows the confusion matrix of each of the three GloVe model variations.

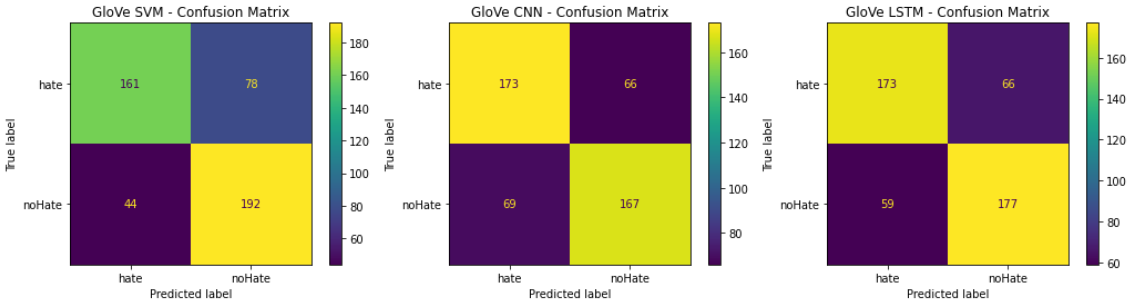

Figure 24: GloVe SVM, CNN, and LSTM Confusion Matrix

Table 3 shows performance metrics for all three GloVe variations of the models. The Support Vector Machine model was able to achieve the highest score numbers, with the Long Short-Term Memory model having the best overall performance.

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
SVM_{GLV}	0.67	**0.81**	**0.74**	**0.79**	0.71	0.67	**0.81**	**0.73**	**0.76**
CNN_{GLV}	**0.72**	0.71	0.72	0.71	0.72	**0.72**	0.71	0.72	0.71
$LSTM_{GLV}$	**0.72**	0.75	**0.74**	0.75	**0.73**	**0.72**	0.75	**0.73**	0.74

Table 3: Metrics for the GloVe SVM, CNN, LSTM models

Like the Word2Vec visualisation, the scatterplot in figure 25 below shows that most of the words that belonged to no Hate sentences were closer together and placed towards the middle of the plane. GloVe was able to capture the context of a few of the no Hate words, like YouTube and video, or year and day. Like before, there were still a few words that seemed like they were placed randomly in the two-dimensional plane (e.g., school and children, or children and kids). There was a slight difference in the way offensive words were placed in the two-dimensional plane. Although words like white and black were grouped together, they were placed away from their plural versions and other offensive words found in hate speech sentences. However, GloVe was able to place together the words negro, africa, and jews indicating that it understood that these words are targeted toward a group of people.

Figure 25: GloVe vectors visualisation

4.4. ELMo Results and Visualisation

Using the output of ELMo with the Support Vector Machine model improved its ability to correctly classify Hate Sentences, compared to the Bag-of-Words model. Although its ability to classify No Hate sentences was slightly affected, the overall accuracy was increased. The model returned an overall accuracy of 74%, a Hate accuracy of 76%, and a No Hate accuracy of 72%. Like before, the embedding vectors had to be averaged, losing the spatial features of

the ELMo vectors. The whole process of preparing the data, extracting the ELMo vectors, and training and testing the model took a great amount of time, which is something that also needs to be considered.

The ELMo outputs improved overall the Convolutional Neural Network model's ability to correctly classify both Hate and No Hate sentences, compared to the model with randomly initialized word embeddings. Although the model's performance in classifying No Hate sentences improved slightly, its ability to recognise and classify Hate sentences was greatly improved. The model returned an overall accuracy of 77%, a Hate accuracy of 75%, and a No Hate accuracy of 79%. Its training history is shown in figure 26 below.

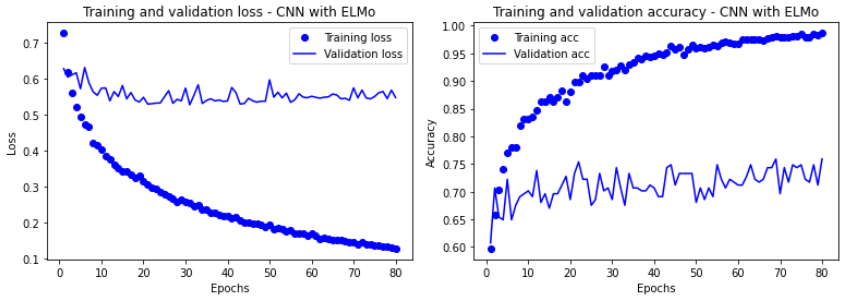

Figure 26: ELMo CNN Training and Validation Loss and Accuracy

Using the output of ELMo with the Long Short-Term Memory slightly improved its overall ability to correctly classify both Hate and No Hate sentences, compared to the model with randomly initialized word embeddings. The model returned an overall accuracy of 74%, a Hate accuracy of 72%, and a No Hate accuracy of 76%. The model's training history is shown in figure 27 below.

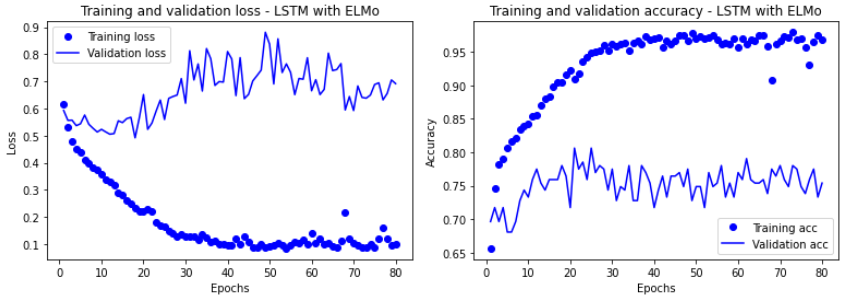

Figure 27: ELMo LSTM Training and Validation Loss and Accuracy

Figure 28 below shows the confusion matrix of each of the three ELMo model variations.

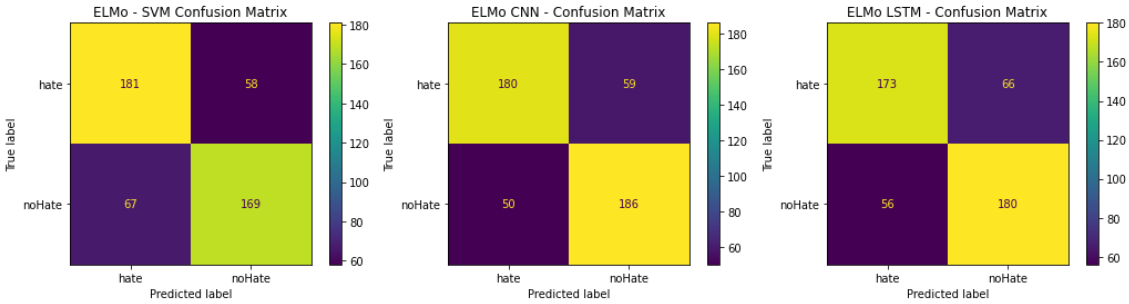

Figure 28: ELMo SVM, CNN, and LSTM Confusion Matrix

Table 4 shows performance metrics for all three ELMo variations of the models. The Convolutional Neural Network model was able to achieve the highest score numbers, having also the best overall performance.

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
SVM_{ELMo}	**0.76**	0.72	0.74	0.73	0.74	**0.76**	0.72	0.74	0.73
CNN_{ELMo}	0.75	**0.79**	**0.77**	**0.78**	**0.76**	0.75	**0.79**	**0.77**	**0.77**
$LSTM_{ELMo}$	0.72	0.76	0.74	0.76	0.73	0.72	0.76	0.74	0.75

Table 4: Metrics for the ELMo SVM, CNN, LSTM models

The scatterplot in figure 29 below shows that the pre-trained ELMo word embeddings did a much better job at capturing the context of the words found in hate and no hate speech sentences, compared to the Word2Vec and GloVe word embeddings. While many of the words were spread out in the Word2Vec and GloVe visualisations, here they were placed together in several mini groups, based either on the meaning of the word or the type of the word. Some of the groups that were identified are:

- A group of words that came from hate speech sentences (e.g., negro, scum, jew, jews)
- A group containing the word country with africa and Ireland
- A group of verbs (e.g., know, want, think, say)

This difference in the embeddings was also reflected in each of the models' accuracy since all three models that used ELMo word embeddings were able to outperform their Word2Vec and GloVe variations.

Figure 29: ELMo vectors visualisation

4.5. BERT Results and Visualisation

Using the output of BERT with the Support Vector Machine model improved overall the model's accuracy, compared to the Bag-of-Words model. Although its ability to classify Hate sentences was not affected, the model was able to correctly classify a greater volume of No Hate sentences. The model returned an overall accuracy of 75%, a Hate accuracy of 71%, and a No Hate accuracy of 79%.

BERT's last hidden layer improved the Convolutional Neural Network model's performance compared to the model with randomly initialized word embeddings. The model returned an overall accuracy of 79%, a Hate accuracy of 80%, and a No Hate accuracy of 77%, outperforming all the other variations of the Convolutional Neural Network model. This variation also required much fewer epochs to train, which was one of the advantages of using BERT and Transfer Learning. Plotting the training and validation loss and accuracy this time made no sense since the model was trained only for four epochs.

Like the Convolutional Neural Network, using BERT's last hidden layer as input for the Long Short-Term Memory model significantly improved the model's performance compared to the model with randomly initialized word embeddings. The model returned an overall accuracy of 79%, a Hate accuracy of 75%, and a No Hate accuracy of 83%, outperforming all the other variations of the model. It also required much fewer epochs to train, which is one of the advantages of using BERT and Transfer Learning. Like before, the training history was not plotted this time.

Figure 30 below shows the confusion matrix of each of the three BERT model variations.

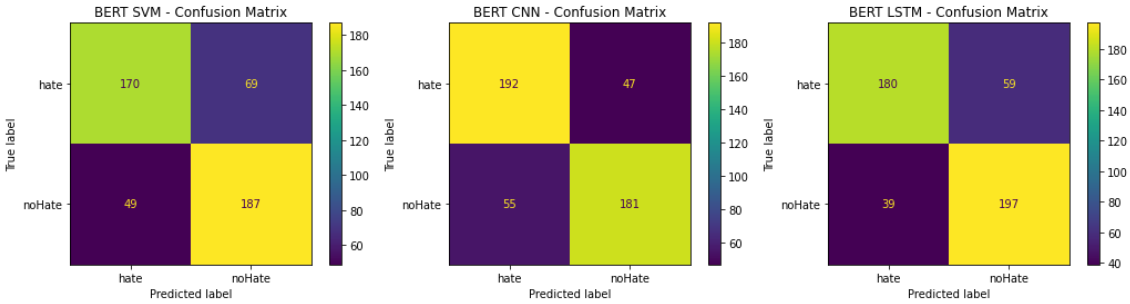

Figure 30: BERT SVM, CNN, and LSTM Confusion Matrix

Table 5 shows performance metrics for all three BERT variations of the models. The Long Short-Term Memory model was able to achieve the best score overall, with the Convolutional Neural Network coming a close second.

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
SVM_{BERT}	0.71	0.79	0.75	0.78	0.73	0.71	0.79	0.74	0.76
CNN_{BERT}	**0.80**	0.77	**0.79**	0.78	**0.79**	**0.80**	0.77	**0.79**	0.78
$LSTM_{BERT}$	0.75	**0.83**	**0.79**	**0.82**	0.77	0.75	**0.83**	**0.79**	**0.80**

Table 5: Metrics for the ELMo SVM, CNN, LSTM models

Like the ELMo word embeddings, the pre-trained BERT word embeddings did a much better job at capturing the context of the words found in hate and no hate speech sentences, compared to the Word2Vec and GloVe word embeddings. As shown in the scatterplot in figure 31 below, words were also placed in several mini groups based on the meaning or the type of the word, instead of being spread out. The difference with ELMo word embeddings was that BERT was able to group together all of the words that were found in hate speech sentences and could be used to offend a group of people (i.e. jew, jews, black, blacks, white, whites, race, negro). This difference was also reflected in each of the models' accuracy, since all three models that used BERT word embeddings were able to outperform their Word2Vec, GloVe, and ELMo variations.

Figure 31: BERT vectors visualisation

4.6. HyperBand Tuning

Out of all models, the BERT variations of the Convolutional Neural Network and Long Short-Term Memory models were the best performers, having similar results, with BERT CNN performing better in classifying Hate Sentences and BERT LSTM performing better in classifying No Hate Sentences. A third Hybrid BERT CNN-LSTM model was created, combining the two architectures and their advantage in each of the two classes. The three models were tuned using the HyperBand algorithm.

Regarding the BERT Convolutional Neural Network model, the Hyperband Tuner was able to increase the model's Hate Accuracy by 2%, achieving a Hate Accuracy of 82%. However, it also decreased the No Hate accuracy by 1%, dropping it down to 76%. The overall accuracy remained unchanged, sitting at 79%. The hyperparameters that were used to achieve these results were:

- Dropout keep probability of the Dropout Layer: **0.2**
- Number of filters in the three Convolution Layers: **64**
- Kernel sizes of the three Convolution Layers: **2, 3,** and **4**
- Decay rate of the AdamW Optimiser: **0.001**
- Learning rate of the AdamW Optimiser: **0.001**

As for the BERT Long Short-Term Memory model, the Hyperband Tuner was able to increase the model's Hate Accuracy by 4%, achieving a Hate Accuracy of 79%. However, it also decreased the No Hate accuracy by 2%, dropping it down to 81%. The overall accuracy also increased by 1%, reaching 80%. The hyperparameters that were used to achieve these results were:

- Dropout keep probability of the Dropout Layer: **0.2**

- Number of the LSTM Layers: **2**
- Type of the LSTM Layers: **Bidirectional**
- Number of units in the LSTM Layers: **32**
- Decay rate of the AdamW Optimiser: **0.001**
- Learning rate of the AdamW Optimiser: **0.0005**

Finally, the hybrid model that Hyperband Tuner created returned an overall accuracy of 82%, with a Hate Accuracy of 83% and a No Hate Accuracy of 81%, ranking it as the number one model, when comparing its performance metrics with those of every other model in this study. The hyperparameters that were used to achieve these results were:

- Dropout keep probability of the Dropout Layer: **0.3**
- Number of filters in the two Convolution Layers: **64**
- Type of the LSTM Layers: **Unidirectional**
- Number of units in the LSTM Layers: **128**
- Decay rate of the AdamW Optimiser: **0.001**
- Learning rate of the AdamW Optimiser: **0.0005**

Figure 32 below shows the confusion matrix of each of the three Tuned BERT models.

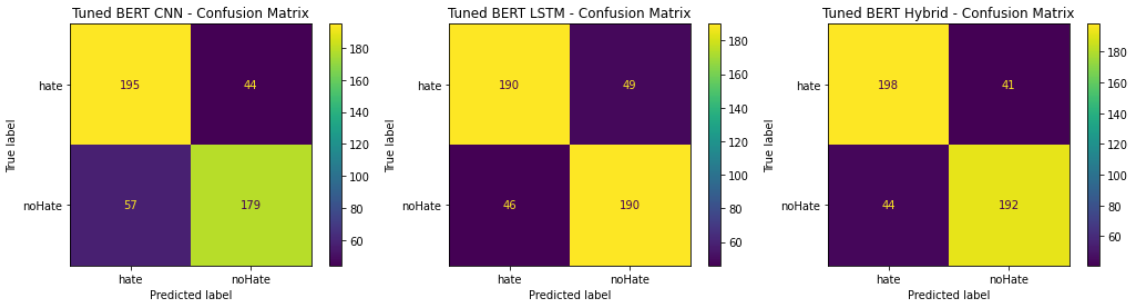

Figure 32: Tuned BERT CNN, LSTM, and Hybrid Confusion Matrix

Table 6 shows performance metrics for all three tuned BERT models. The Hybrid CNN-LSTM model was able to achieve the best score across the board, outperforming every other model tested in this research.

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
CNN_{TBERT}	0.82	0.76	0.79	0.77	0.80	0.82	0.76	0.79	0.78
$LSTM_{TBERT}$	0.79	**0.81**	0.80	0.81	0.79	0.79	**0.81**	0.80	0.80
$HYBRID_{TBERT}$	**0.83**	**0.81**	**0.82**	**0.82**	**0.82**	**0.83**	**0.81**	**0.82**	**0.82**

Table 6: Metrics for the tuned BERT CNN, LSTM, Hybrid models

4.7. Transfer Learning

Two final tests were conducted using the tuned hybrid model and the Twitter dataset, to evaluate the model's ability in Transfer Learning. In the first test, the model was tested as it was, using the whole dataset. The tuned pre-trained model correctly classified 67% of tweets containing hate speech and 79% of tweets containing no hate speech. While the model's performance on this dataset was decent, it was poorer than the original dataset, having an overall accuracy of 73% compared to 82%. Many of the hateful tweets could not be captured by the model. There are three explanations for this:

- First, the model was trained with a sentence length of one hundred, while most tweets had a sentence length of 5 to 20. This extreme padding and the addition of zeros can potentially "confuse" the model, making it unable to identify the small changes in the vectors of each sentence.
- Second, Twitter users might use a different vocabulary or writing style than the users of the white supremacist forum.
- Third, the Twitter datasets did not use the same annotation guidelines as the white supremacy forum dataset. Words that were classified as hate speech in the white supremacy forum dataset might not have been considered hate speech in the Twitter datasets and vice versa.

Figure 33 and Table 7 show the tuned model's Confusion Matrix and metrics, respectively, using the whole dataset.

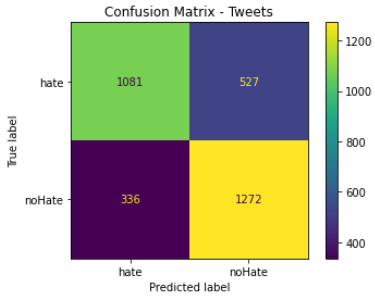

Figure 33: Tuned Hybrid Confusion Matrix, whole twitter dataset

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
$HYBRID_{TBERTtw}$	0.67	0.79	0.73	0.76	0.71	0.67	0.79	0.71	0.75

Table 7: Metrics for the tuned Hybrid, whole twitter dataset

In the second test, a training and a test subset were created from the Twitter dataset. The model was first tested on the test subset, then fine-tuned for one training epoch using the training subset and tested one last time. Fine-tuning the model, even for one epoch, improved its

performance across the board. While the Hate Accuracy saw a moderate increase of 6%, reaching 72%, the No Hate Accuracy increased by 19%, reaching an almost perfect accuracy score of 98%. The overall accuracy increased to 85% from 73%. At first glance, the No Hate Accuracy suggests that the model is overfitting the No Hate class. However, the F1 Scores of both classes are close to each other, suggesting otherwise. The results proved that the hybrid model can be used for Transfer Learning and applied to similar problems and that its performance can be further improved even by training it for just one epoch. Figure 34 and Table 8 show the model's Confusion Matrix and metrics respectively, before and after fine-tuning it.

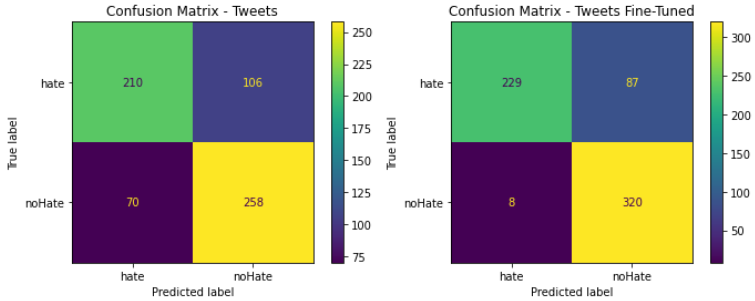

Figure 34: Tuned and Fine-Tuned Hybrid Confusion Matrix

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
$HYBRID_{TBERTtw}$	0.66	0.79	0.73	0.75	0.71	0.66	0.79	0.70	0.75
$HYBRID_{FtBERTtw}$	**0.72**	**0.98**	**0.85**	**0.97**	**0.79**	**0.72**	**0.98**	**0.83**	**0.87**

Table 8: Metrics for the Tuned and Fine-Tuned Hybrid model

4.8. Experimental Results and Obstacles

This section describes the experimental results and obstacles that occurred during my research.

Data: When I first began my research, the first model trials were done using the whole dataset instead of the supplied sampled_train and sampled_test subsets. Because the dataset was class imbalanced, the models' performance was poor since they were overfitting the dominant class. Loading the balanced subsets allowed me to proceed with my research.

Pre-Processing: During the trials, different pre-processing approaches were tested. Initially, all English stopwords were removed from the text, and only words found in NLTK's English vocabulary were kept. This resulted in a poorer performance across all models which is why I decided to exclude this approach from my research. This proved that pre-processing approaches that are considered common practice in one NLP problem, might give negative results in another.

Performance Metrics: The performance metrics that were calculated in the early stages of the research included the overall accuracy and the F1 score of the hate class. However, to better

compare my results with the reference paper's and to have a holistic view of my models' performance, the confusion matrix and the accuracy, precision, recall, and f1 scores for each class were added.

Google Colab Pro+: In the early stages of my research, the models were created on my Macbook Pro, using VS Code. While the Support Vector Machine part ran flawlessly, the Keras models would not run at all, crashing the kernel. Once I transitioned to Colab Pro+, everything worked as intended.

Pre vs Post Padding: When the default sequence padding option (e.g., post padding) was used with the LSTM model, the model was unable to train. This phenomenon was also explored in (Dwarampudi and Reddy, 2019). Post-padding seems to add noise to what has been learned from a sequence through time, and the LSTM cannot recover from this noise, at least not in a reasonable timeframe. With pre-padding, the model can adjust to the added noise of zeros at the beginning since it can learn from the actual sequence through time.

Recreating CNN and LSTM results: When reproducing the original neural network models, the initial results were better than those reported in the reference paper. The reason for this was that the embedding layer was training and adapting the randomly initialized word embeddings to the problem. Once I turned off the training of the embedding layer, I was able to reproduce the results.

Max sentence length: Initially, a sentence length of 323 was used for all model variations during dataset preparation. The histogram showed however that most sentences had a length of zero to one hundred. Changing to a sentence length of one hundred improved the models' performance by a small amount. The models initially were not able to capture the details of each sentence's features, since at least 223 features were zeros for almost all the sentences.

Word embeddings with SVM: SVM's lack of an embedding layer required the manual mapping of each word's vector in a sentence. This resulted in three-dimensional data which could not be used by the model. Initially, the data was reshaped which resulted in a substantially large array, dramatically increasing the model's training time. Using the mean of each vector for each word instead brought training times back to normal and improved the performance.

ELMo Model as Keras layer: Originally, the plan was to include the ELMo implementation inside the Keras models, loading it as a Keras layer. However, I couldn't implement this due to compatibility reasons with TensorFlow Hub and Keras. The sentences were pre-processed first by passing them through ELMo, saving the output as a new dataset and passing that through the Keras models.

BERT training epochs: A higher number of training epochs was used initially to train the BERT model variations. However, the models quickly began to overfit after the first ten epochs. Interestingly, the optimal training epochs were found to be between two and four, being in line with what has been found in (Devlin *et al.*, 2019).

4.9. Conclusion

Table 9 below contains the consolidated results of every model created in this research, together with the original results reported in the reference paper. Replacing the traditional word embedding methods with newer, contextualized pre-trained word embedding methods, specifically ELMo and BERT, while also maintaining the same model architecture, improved all three original models' performance significantly. Furthermore, I was able to achieve even better results by introducing a hybrid BERT CNN-LSTM model, while also proving that it can be applied to other similar datasets achieving great results, by fine-tuning it just for one training epoch, as shown in table 8 above.

	Acc_{HATE}	Acc_{NOHATE}	Acc_{ALL}	$Prec_{HATE}$	$Prec_{NOHATE}$	Rec_{HATE}	Rec_{NOHATE}	$F1_{HATE}$	$F1_{NOHATE}$
SVM_{OG}	0.69	0.73	0.71	-	-	-	-	-	-
CNN_{OG}	0.55	0.79	0.66	-	-	-	-	-	-
$LSTM_{OG}$	0.71	0.75	0.73	-	-	-	-	-	-
SVM_{REP}	0.72	0.74	0.73	0.74	0.72	0.72	0.74	0.73	0.73
CNN_{REP}	0.57	0.74	0.66	0.69	0.63	0.57	0.74	0.63	0.68
$LSTM_{REP}$	0.67	0.76	0.71	0.74	0.69	0.67	0.76	0.70	0.72
SVM_{W2V}	0.53	0.89	0.71	0.82	0.65	0.53	0.89	0.64	0.75
CNN_{W2V}	0.57	0.81	0.69	0.76	0.65	0.57	0.81	0.65	0.72
$LSTM_{W2V}$	0.53	0.85	0.69	0.78	0.64	0.53	0.85	0.63	0.73
SVM_{GLV}	0.67	0.81	0.74	0.79	0.71	0.67	0.81	0.73	0.76
CNN_{GLV}	0.72	0.71	0.72	0.71	0.72	0.72	0.71	0.72	0.71
$LSTM_{GLV}$	0.72	0.75	0.74	0.75	0.73	0.72	0.75	0.73	0.74
SVM_{ELMo}	0.76	0.72	0.74	0.73	0.74	0.76	0.72	0.74	0.73
CNN_{ELMo}	0.75	0.79	0.77	0.78	0.76	0.75	0.79	0.77	0.77
$LSTM_{ELMo}$	0.72	0.76	0.74	0.76	0.73	0.72	0.76	0.74	0.75
SVM_{BERT}	0.71	0.79	0.75	0.78	0.73	0.71	0.79	0.74	0.76
CNN_{BERT}	0.80	0.77	0.79	0.78	0.79	0.80	0.77	0.79	0.78
$LSTM_{BERT}$	0.75	0.83	0.79	0.82	0.77	0.75	0.83	0.79	0.80
CNN_{TBERT}	0.82	0.76	0.79	0.77	0.80	0.82	0.76	0.79	0.78
$LSTM_{TBERT}$	0.79	0.81	0.80	0.81	0.79	0.79	0.81	0.80	0.80
$HYBRID_{TBERT}$	**0.83**	**0.81**	**0.82**	**0.82**	**0.82**	**0.83**	**0.81**	**0.82**	**0.82**

Table 9: Consolidated results for all models

Comparing my results with the work of (Alatawi, Alhothali and Moria, 2021), although the No Hate F1-Score of my Hybrid model was 0.82 compared to 0.95 that their BERT Base model achieved, my Hybrid model achieved a Hate F1-Score of 0.82 compared to their model's 0.58 Hate F1 Score. The results in this paper suggest that the BERT Base model is overfitting in the No Hate class, a common problem that was seen in several papers during the literature review. Furthermore, their experiments were conducted without including sentences that required additional context for manual annotation. In my research, these sentences were included, which increased the difficulty of hate speech detection, as described before.

Chapter 5. Discussion

5.1. Evaluation of Objectives

Reproduce the results of the reference paper

The three models from the reference paper were reproduced, and my results were almost identical to the published results. Despite the small deviation in the models' accuracy, this objective was a success, considering the models had to be created from scratch, based on their description in the reference paper. This was an important objective for the research because it allowed for more fair comparisons with the models that were created afterwards.

Apply and compare different pre-trained word embedding methods

This objective was met successfully, as I was able to apply all four mentioned pre-trained word embedding methods to all three reproduced models and compare their performance metrics. Using the word embedding methods with the Support Vector Machine was very tricky, due to its nature, despite being the easiest model to reproduce initially. Furthermore, visualizing the vectors of each embedding method in a two-dimensional plane proved to be quite challenging as well. Nevertheless, I was able to overcome these obstacles using sophisticated code and conditional programming, utilizing my background in Computer Science.

Create, tune, test, and save the best performing model

The Keras Tuner library and the Hyperband algorithm proved to be the right choice for this objective. Not only did it manage to fine-tune the BERT CNN and LSTM models and further increase their performance, but it also created a hybrid model combining the architecture of the two models, outperforming every other model. The best model was saved in a Keras model format, which proved to be appropriate since I was able to not only successfully load the model from a file but also continue training it on new data.

Assess whether contextualised pre-trained word embeddings improve hate speech detection

This objective was very well met since by examining and comparing the results of the previous objectives, I was able to answer positively that the contextualized pre-trained word embedding methods indeed improve hate speech detection.

Assess whether the best performing model qualifies for transfer learning

The final objective was successfully met as well by loading, fine-tuning, and testing the hybrid model with a new similar dataset. Since its performance on the test set was overall great, it qualified for transfer learning.

5.2. New Learning

The method of combining different models and layers or utilizing the output of a pre-trained BERT model and adding extra deep learning layers on top of it to solve classification or regression problems is something that researchers have been doing for a while, as seen during the literature search. However, the application of a BERT Hybrid CNN-LSTM network with that specific architecture in the Hate Speech Dataset from a White Supremacy Forum was not seen anywhere in the literature search.

Regarding personal learning, prior to conducting this research, I had little to no knowledge about the field of Natural Language Processing, word embedding techniques and text classification. Through this research, I was able to further improve my Python development skills, increase my knowledge in Deep Learning and introduce myself to an exciting field that has a wide variety of applications.

5.3. Research Questions

Both research questions were answered positively. Using pre-trained contextualized word embeddings as inputs for the three models increased their performance across the board. Furthermore, it was proven that the model itself that utilised these newer word embedding methods qualified for Transfer Learning and could be used with similar datasets.

Chapter 6. Evaluation, Reflections, and Conclusions

6.1. Choice of Objectives

Overall, the choice of objectives for the project proved to be successful, as the research yielded positive results and was completed in a reasonable amount of time. The initial literature review successfully led me to a text classification problem that had room for improvement i.e., the automatic hate speech detection. It also directed me into using appropriate methods that contributed to the positive results of the research, such as the AdamW optimiser, the BERT model, and the Hyperband algorithm. The Keras Deep Learning framework played a pivotal role in the research, as it allowed for easy integration of the word embeddings to the models, through the embedding layer, enabling me to allocate my resources into trying new things, like combining network architectures, rather than spending time writing and debugging difficult code. The project conclusions confirmed what was already known about pre-trained word embedding methods like BERT and their applications in transfer learning, but they also suggested that Deep Learning models that utilize these methods could also be applied in transfer learning scenarios themselves.

6.2. Project Limitations and Weaknesses

This research experimented with Hate Speech Detection as a Binary Classification problem, classifying a sentence either as strictly Hate speech language or not, ignoring one of the most popular types of languages, the offensive language. The product of this project performed great in cases where a sentence either contained strong hate speech language, targeting a group of people (based on their Race, Gender, Sexuality, or Religion) or did not contain hate speech language at all. The model could potentially find it difficult to recognize strong offensive language that is not targeted toward a group of people in a way that can be classified as hate speech language. It will also struggle to classify sentences in cases where the person is using sarcastic language. For example, a person that supports or defends a group of people might use sarcasm to make a point or describe a situation on a social media platform. Although this person does not support hate speech, his post would get flagged by the model, resulting in an unjustified potential ban from the platform.

6.3. Future Work

Following the project limitations and weaknesses above, three points are proposed for future work:

- Adapt the model to work with multi-class classification problems, so it can learn to detect other forms of offensive language
- It would be interesting to find ways to make the model understand sarcasm. A potential method would be to make decisions based on other dataset features as well. In a social media platform, for example, it could make use of the profile of the user, like which pages or people they follow, or if they had recently posted hate speech related messages.
- Finally, the model could be trained on a much bigger dataset that combined hate speech sentences from multiple sources (e.g., Tweets, Facebook posts, and Forum posts), to adapt to the different writing styles and sentence lengths.

6.4. Reflection on the Project

The project was an interesting challenge for me as through it I got introduced to the field of Natural Language Processing, expanding my Data Science knowledge. Although my research began based on pre-existing work, I was able to contribute to the scientific community of Artificial Intelligence and Natural Language Processing, improving the results reported in the reference paper by introducing a hybrid BERT CNN-LSTM model. The outcome of my project has motivated me to do further research in this field and explore other applications of NLP that involve different types of data, such as speech data.

References

Abad, A. et al. (eds) (2016) Advances in Speech and Language Technologies for Iberian Languages: Third International Conference, IberSPEECH 2016, Lisbon, Portugal, November 23-25, ... 1st ed. 2016 edition. New York, NY: Springer.

Alatawi, H.S., Alhothali, A.M. and Moria, K.M. (2021) 'Detecting White Supremacist Hate Speech Using Domain Specific Word Embedding With Deep Learning and BERT', IEEE Access, 9, pp. 106363–106374. Available at: https://doi.org/10.1109/ACCESS.2021.3100435.

Bird, S., Klein, E. and Loper, E. (2009) Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. 1st edition. Beijing ; Cambridge Mass.: O'Reilly Media.

Chowdhury, G.G. (2003) 'Natural Language Processing', p. 39.

Davidson, T. et al. (2017) Automated Hate Speech Detection and the Problem of Offensive Language. arXiv:1703.04009. arXiv. Available at: http://arxiv.org/abs/1703.04009 (Accessed: 15 May 2022).

Devlin, J. et al. (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805. arXiv. Available at: http://arxiv.org/abs/1810.04805 (Accessed: 17 May 2022).

Dwarampudi, M. and Reddy, N.V.S. (2019) 'Effects of padding on LSTMs and CNNs'. arXiv. Available at: http://arxiv.org/abs/1903.07288 (Accessed: 7 September 2022).

Founta, A.-M. et al. (2018) 'Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior'. arXiv. Available at: http://arxiv.org/abs/1802.00393 (Accessed: 6 September 2022).

de Gibert, O. et al. (2018) 'Hate Speech Dataset from a White Supremacy Forum', in Proceedings of the 2nd Workshop on Abusive Language Online (ALW2). Brussels, Belgium: Association for Computational Linguistics, pp. 11–20. Available at: https://doi.org/10.18653/v1/W18-5102.

Global daily social media usage 2022 (no date) Statista. Available at: https://www.statista.com/statistics/433871/daily-social-media-usage-worldwide/ (Accessed: 25 September 2022).

GloVe: Global Vectors for Word Representation (no date). Available at: https://nlp.stanford.edu/projects/glove/ (Accessed: 13 September 2022).

Harris, Z.S. (1954) 'Distributional Structure', WORD, 10(2–3), pp. 146–162. Available at: https://doi.org/10.1080/00437956.1954.11659520.

Iginio, G. et al. (2015) Countering online hate speech. UNESCO Publishing.

Internet and social media users in the world 2022 (no date) Statista. Available at: https://www.statista.com/statistics/617136/digital-population-worldwide/ (Accessed: 25 September 2022).

Introduction to the Keras Tuner | TensorFlow Core (no date) TensorFlow. Available at: https://www.tensorflow.org/tutorials/keras/keras_tuner (Accessed: 13 September 2022).

Joshi, P. (2019) 'What is ELMo | ELMo For text Classification in Python', Analytics Vidhya, 10 March. Available at: https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/ (Accessed: 13 September 2022).

Keras: the Python deep learning API (no date). Available at: https://keras.io/ (Accessed: 13 September 2022).

Kim, Y. (2014) Convolutional Neural Networks for Sentence Classification. arXiv:1408.5882. arXiv. Available at: http://arxiv.org/abs/1408.5882 (Accessed: 16 May 2022).

Kingma, D.P. and Ba, J. (2017) Adam: A Method for Stochastic Optimization. arXiv:1412.6980. arXiv. Available at: https://doi.org/10.48550/arXiv.1412.6980.

Li, L. et al. (2018) Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. arXiv:1603.06560. arXiv. Available at: https://doi.org/10.48550/arXiv.1603.06560.

Loshchilov, I. and Hutter, F. (2019) Decoupled Weight Decay Regularization. arXiv:1711.05101. arXiv. Available at: https://doi.org/10.48550/arXiv.1711.05101.

Manning, C.D. and Schütze, H. (1999) Foundations of Statistical Natural Language Processing. 1st edition. Cambridge, Mass: The MIT Press.

Mathew, B. et al. (2019) 'Spread of Hate Speech in Online Social Media', in Proceedings of the 10th ACM Conference on Web Science. New York, NY, USA: Association for Computing Machinery (WebSci '19), pp. 173–182. Available at: https://doi.org/10.1145/3292522.3326034.

McCulloch, W.S. and Pitts, W. (1943) 'A logical calculus of the ideas immanent in nervous activity', The bulletin of mathematical biophysics, 5(4), pp. 115–133. Available at: https://doi.org/10.1007/BF02478259.

Mondal, M., Silva, L.A. and Benevenuto, F. (2017) 'A Measurement Study of Hate Speech in Social Media', in Proceedings of the 28th ACM Conference on Hypertext and Social Media. New York, NY, USA: Association for Computing Machinery (HT '17), pp. 85–94. Available at: https://doi.org/10.1145/3078714.3078723.

Mozafari, M., Farahbakhsh, R. and Crespi, N. (2020) 'A BERT-Based Transfer Learning Approach for Hate Speech Detection in Online Social Media', in H. Cherifi et al. (eds) Complex Networks and Their Applications VIII. Cham: Springer International Publishing (Studies in Computational Intelligence), pp. 928–940. Available at: https://doi.org/10.1007/978-3-030-36687-2_77.

Naidu, T.A. and Kumar, S. (2021) 'Hate Speech Detection Using Multi-Channel Convolutional Neural Network', in, pp. 908–912. Available at: https://doi.org/10.1109/ICAC3N53548.2021.9725696.

Nockleby, J.T. (2000) ‘Hate speech. Encyclopedia of the American Constitution, 3:1277-79.’, in.

Pennington, J., Socher, R. and Manning, C. (2014) ‘Glove: Global Vectors for Word Representation’, in, pp. 1532–1543. Available at: https://doi.org/10.3115/v1/D14-1162.

Peters, M.E. et al. (2018) Deep contextualized word representations. arXiv:1802.05365. arXiv. Available at: http://arxiv.org/abs/1802.05365 (Accessed: 16 May 2022).

Pietro, M.D. (2022) Text Classification with NLP: Tf-Idf vs Word2Vec vs BERT, Medium. Available at: https://towardsdatascience.com/text-classification-with-nlp-tf-idf-vs-word2vec-vs-bert-41ff868d1794 (Accessed: 13 September 2022).

sklearn.feature_extraction.text.CountVectorizer (no date) scikit-learn. Available at: https://scikit-learn/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (Accessed: 4 September 2022).

Swamy, S.D., Jamatia, A. and Gambäck, B. (2019) ‘Studying Generalisability across Abusive Language Detection Datasets’, in Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL). CoNLL 2019, Hong Kong, China: Association for Computational Linguistics, pp. 940–950. Available at: https://doi.org/10.18653/v1/K19-1088.

Team, K. (no date) Keras documentation: Using pre-trained word embeddings. Available at: https://keras.io/examples/nlp/pretrained_word_embeddings/ (Accessed: 13 September 2022).

Texas A&M University, USA, Gao, L. and Huang, R. (2017) ‘Detecting Online Hate Speech Using Context Aware Models’, in RANLP 2017 - Recent Advances in Natural Language Processing Meet Deep Learning. RANLP 2017 - Recent Advances in Natural Language Processing Meet Deep Learning, Incoma Ltd. Shoumen, Bulgaria, pp. 260–266. Available at: https://doi.org/10.26615/978-954-452-049-6_036.

‘Twitter API v2 sample code’ (2022). @TwitterDev. Available at: https://github.com/twitterdev/Twitter-API-v2-sample-code/blob/54b161835fc7fb7c16e358f1c46ec12b4c0ec418/Tweet-Lookup/get_tweets_with_bearer_token.py (Accessed: 13 September 2022).

UNESCO and United Nations Office on Genocide Prevention and the Responsibility to Protect (2021) ‘Addressing hate speech on social media: contemporary challenges’, p. 10.