Emoji-Based Hate Speech Detection on Social Media

Anonymous ACL submission

Abstract

1 Introduction

The rise of Internet technology in the 21st century has progressively grown, and by profiting from such convenience and liberation in cyberspace, network users can express different opinions freely, including some vulgar or hateful languages (Mubarak et al., 2022). This explained why the amount of harmful and abusive languages has increased steadily owing to the rapidly growing amount of online user-generated content (Wiegand and Ruppenhofer, 2021).

The spread of hateful language probably impacts the internets well-being, as well as individuals' psychological health (Althobaiti, 2022). Therefore, it is necessary to investigate potential solutions for detecting online hateful languages automatically and with high efficiency. In the past few years, several researchers in the fields of Natural language processing (NLP) and artificial intelligence (AI) have contributed to building different models to deal with the problem of hate speech(Waseem et al., 2017; Schmidt and Wiegand, 2017; Pereira-Kohatsu et al., 2019), and most of these studies focused on words and text-based messages. Although the situation of online hateful language usage seems to have improved to some extent, there are still a certain number of potentially harmful languages that cannot be detected. Since human beings are gifted at avoiding censorship, they can create some word plays to show the same meaning within different approaches and even utilize emojis to replace some harmful vocabulary. This kind of circumstance makes it hard for the existing hate speech models to detect harmful content.

When it comes to emojis, they were originally employed to demonstrate and reinforce the expression of emotion as attached to a text (Shardlow et al., 2022). Nevertheless, emojis are also adopted to convey some intended meanings, function as a semantic part in a sentence, and even are utilized to compose hate speech for avoiding an automatic ban. Numerous social media platforms such as Twitter, Facebook, and Instagram are suffering from the problem that hate speech containing emojis, especially when emojis are used to substitute some sensitive words. For the basic models of automatic hate speech detection which concentrate on textual information, hateful content is diverse and complicated, and one certain challenging task is the usage of emojis for expressing hate (Kirk et al., 2022). Kirk et al. (2022) and other scholars have tested the existing content moderation models using the HATEMOJICHECK[1] dataset they built, and also proposed a new model by using HATEMOJIBUILD [1] dataset to obtain better performance for detecting the hateful content with the use of emojis. Since the hateful content is complex and keeps updating, the new variants of expressing hate with emojis also occur constantly. Therefore the main purpose of this research is to enhance hate speech detection dedicated to emoji-based hate speeches on social media. Stemmed on the dataset provided by Kirk et al. (2022), the related real social media messages can be obtained. Small additional data will also be scraped and labeled with at least two annotators, after that the Cohen Kappa Agreement is computed to see how well the annotation result. Then a new emoji-based hate speech detection model will be trained with the Kirk dataset and tested with a new dataset that is crawled from social media. Then this model will be qualitatively evaluated as well.

[1] https://github.com/HannahKirk/Hatemoji

2 Scraping Social Media Comments

2.1 TikTok

2.2 YouTube

3 Dataset

Given a recently dedicated dataset (Kirk et al. (2022)), additional sets of social media messages utilizing emojis have to be scraped from social media (TikTok and YouTube). Scraped messages have to contain both hate and normal speech usage. These two figures show that the most words and tokens mostly appear in positive comments 1 and negative comments 2 using in TikTok and YouTube platforms.

Figure 1: TikTok and YouTube Positive Comments WordCloud

Figure 2: TikTok and YouTube Negative Comments WordCloud

The annotation of these sets was conducted by two annotators before being used to prepare a hate speech classifier dedicated to these particular emojis utilization. Moreover, the objective is not to consider all possible harmful emojis but as it were the ones from the sub-dataset. The annotation guide is created also to understand each meaning of each label well for the two annotators [2]. There are 2 groups of messages positive and negative. At that point, as an extra objective, the nature of each message is considered (e.g. Its 🤮 🤮 Tag: Hated, Type: Disgust, I 🥰 it Tag: Non-Hated, Type: Sweet). Therefore, there are 9 types of negative speech and 10 types of positive speech. There are a total of 200 new scrape datasets 100 for the hated group and 100 for the non-hated speech group and for each group, there are 50 for TikTok and 50 for YouTube as the reason is not to bias the prediction from the training model. From that, organize dataset we seem to have diverse testing reasons as well. As a result, The Cohen Kappa score between the two annotators is 0.7584. Additionally, the Cohen Kappa agreement score of each label has been calculated as well 3. Noticeably, the most agreement labels between two annotators are "concerning" labels while "respect" labels are the lowest agreements between two annotators.

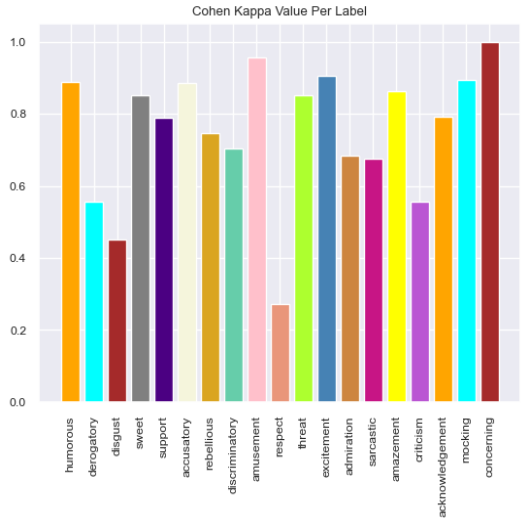

Figure 3: Cohen Kappa Value Per Label

The correlation annotation result is provided here 4. Interestingly, annotator 1 focuses on the messages that are "rebellious" and annotator 2 thinks it is "amazement". Additionally, annotator 1 thinks the message is "exciting" and annotator 2 put it as "accusatory". Overall, there is some disagreement between annotators 1 and 2 almost on positive messages.

We could see that the most occurring type of message is "sarcastic" with a percentage of 15% of 30 messages and the smallest amount of messages occurring in the whole dataset is "respect" which is only 2% and only 3 messages. Below 5 is the plot to show the count of different types of messages that have been annotated.

Notably, the 10 most emojis used in hate 6 and non-hated 7 speech have a little different from each other.

[2] https://docs.google.com/document/d/1Y6dWHIB-DYZqNvIDsJdCAj2ncEfmA0ND-MUI2CKA2XBpc/edit?usp=sharing

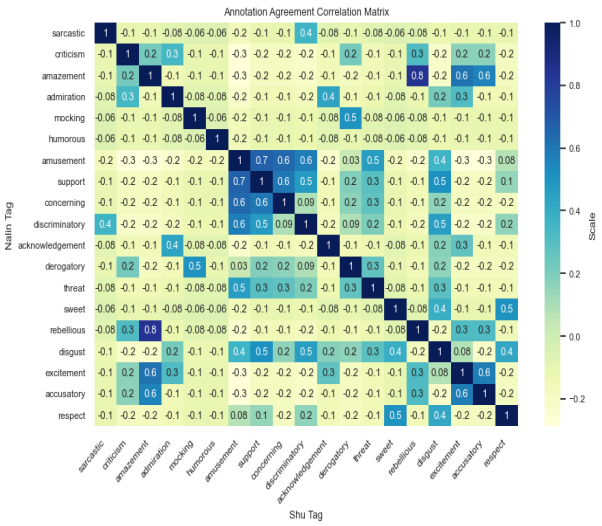

Figure 4: Annotation Agreement Correlation

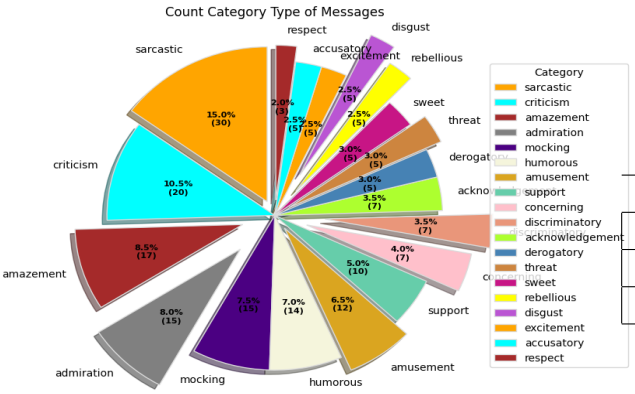

Figure 5: Messages Count Categories

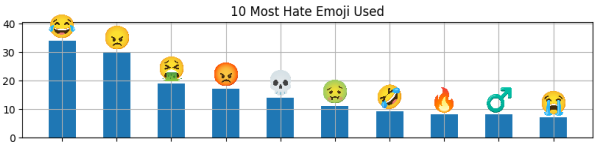

Figure 6: 10 Most Hated Emojis Used

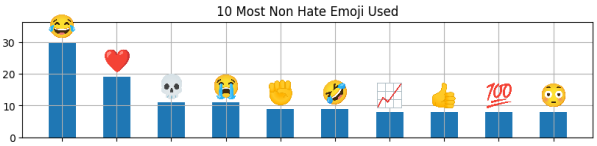

Figure 7: 10 Most Non-Hated Emojis Used

4 Method

4.1 Dataset Preparation

The training step will be trained on all the Kirk datasets then we will test on the whole TikTok and YouTube dataset which is described below.

Domain	Dataset	Sentences
Kirk	Train	4724
	Test	594
	Validation	592
TikTok	Test	100
YouTube	Test	100

Table 1: Dataset Preparation

Below describing the dataset we organize to build our model:

Phase 1

Domain	Dataset	Sents
Kirk(Train+Validation)	Train	5316
TikTok+YouTube	Test	200

Table 2: Dataset Preparation Phase 1

Phase 2

Domain	Dataset	Sents
Kirk(Train+Validation+Test)	Train	5910
TikTok+YouTube	Test	200

Table 3: Dataset Preparation Phase 2

4.2 Feature Extraction

There are many advantages of FastText compared to other feature numerical statistic representations (e.g. TF-IDF, Hashing Vectorizer, traditional bag-of-words, or word2vec). FastText goes beyond individual words and takes into account subword information, such as character n-grams. This allows the model to capture out of vocabulary words. By representing words as a combination of character n-grams, FastText can handle unseen words more effectively. We can use pre-trained embeddings to save time and resources, especially when working with limited training data. Also, FastText is designed to capture semantic meaning in text and associate similar word representations for words that appear in similar contexts. In our work, we use FastText [3] pre-trained embedding model for English which is for 4G.

[3] https://fasttext.cc/docs/en/crawl-vectors.html

4.3 Proposed of Machine Learning Model

4.4 Proposed Architecture of Deep Learning Model

In the process of training each model, we set the number of epochs to 10, batch size 64. CrossEntropyLoss() is used to calculate loss which is a commonly used loss function for classification tasks. It combines the softmax activation function and the negative log-likelihood loss to calculate the loss between the predicted and true labels. Then, we use the Adam optimizer, a popular optimization algorithm for training neural networks. It is initialized with a learning rate set to 0.005. During the fitting loop of the model, the early stop is applied, if the training model does not improve performance for the next 5 times(epochs) we shall stop the training process.

4.4.1 LSTM

The two different modified model settings were applied to train the model. First, the embedding layer takes input indices and maps them to dense vectors. It has an input size of (vocab size) and an output size of 300 (embedding size). The embedding layer is used to learn the distributed representation of the input words. Then, the next layer is the LSTM layer which takes an input size of 300 (embedding size) and a hidden size of 256. The LSTM is bidirectional that process the input sequence in both forward and backward directions. Then, the dropout layer with a probability of 0.3 is used to prevent overfitting by randomly settings elements of the input to zero during training. The fully connected linear layer maps the output of the LSTM to the output classes. It takes an input size of 256 (hidden size) and produces an output size of 2 possible classes. In other modified model architecture, additionally, we used a pre-trained embedding model which is fastText to build our word representation. So, the embed layer is initialized using a pre-trained embedding model.

4.4.2 CNN

The input is a vocabulary size, and each word is represented as a 300-dimensional embedding. The (embed) layer is responsible for mapping the input word indices to their corresponding embedding. There are 3 convolutional layers, conv1 performs 1-dimensional convolution with a kernel size of 2 and stride of 1. The input size is 300 (embedding size), and the output size is 128 then conv2 and conv3 are the same as conv1. After that, the Dropout layer rate of 0.3 and is the technique to prevent overfitting. The output of the last convolutional layer is passed through a linear layer (decision) to produce the final output.
We could see that fastText has an impact on the performance of the model so in this CNN model we only train the CNN with FastText pre-trained model by playing around with the amount of dataset.

4.5 Proposed Method of Building Multi Classification

After obtaining one suitable binary model to predict whether the message is hated or non-hated speech this time we start to build a multi-classification model to predict which type of message (e.g. sarcastic, threat, criticism, support). As our dataset is not the same as the Kirk dataset because we have almost 20 categories of messages while Kirk only has 4 types of messages and only focuses on hate speech but for us, we also separate the type of positive messages as well. So it is hard for us to build a new model from scratch as the amount of data we got now only 200 messages. Therefore, we decided to tune the pre-trained model from hugging face and the model is called "bert-base-uncased". Then, we applied tuning in that model by adding more text and labels. The adding training set is 180 messages and adding testing set is 20 messages. The model from Hugging Face uses BERT model architecture, which is a transformer-based model for natural language processing tasks. It also uses BertTokenizer from the Hugging Face library, specifically for the "bert-base-uncased" variant. Then, for model instantiation it used BertForSequenceClassification, AdamW (Adam with weight decay) is used as an optimizer algorithm for training neural networks. And the learning rate is set to 2e-5, and the loss function is CrossEntropthyLoss. We also saved the best model during training based on the best validation loss, this allows us to save the model with the best performance on the validation set.

5 Experiment Result

5.1 Machine Learning Model

5.2 LSTM

Phase 1

At epoch 6, we could find the best model which achieved 0.5344 of the F1-score on the validation set.

Classification Report:

	precision	recall	f1-score	support
1	0.5513	0.4300	0.4831	100
0	0.5328	0.6500	0.5856	100
accuracy			0.5400	200
macro avg	0.5420	0.5400	0.5344	200
weighted avg	0.5420	0.5400	0.5344	200

Figure 8: LSTM (not using FastText) Classification Report Phase 1

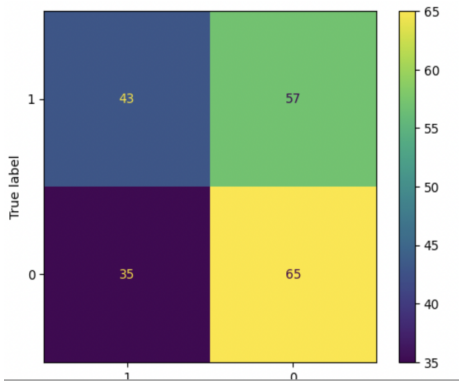

Figure 9: LSTM (not using FastText) Confusion Matrix Phase 1

Using FastText Embedding Pretrained Model

At epoch 6, we could find the best model which achieved 0.5545 of the F1-score on the validation set.

Classification Report:

	precision	recall	f1-score	support
1	0.5514	0.5900	0.5700	100
0	0.5591	0.5200	0.5389	100
accuracy			0.5550	200
macro avg	0.5553	0.5550	0.5545	200
weighted avg	0.5553	0.5550	0.5545	200

Figure 10: LSTM (using FastText) Classification Report Phase 1

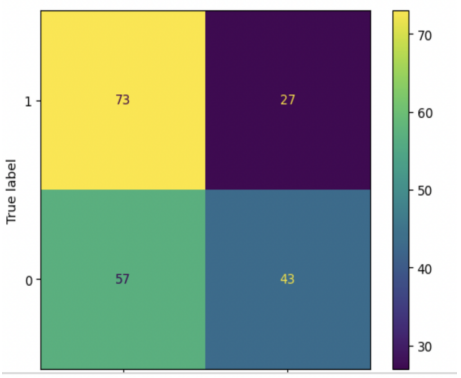

Figure 11: LSTM (using FastText) Confusion Matrix Phase 1

Phase 2

At epoch 10, we could find the best model which achieved 0.5700 of the F1-score on the validation set.

Epoch 10/50 | Train loss: 0.0037 | Valid loss: 0.0275 | Acc: 57.0000%
Classification Report:

	precision	recall	f1-score	support
1	0.5700	0.5700	0.5700	100
0	0.5700	0.5700	0.5700	100
accuracy			0.5700	200
macro avg	0.5700	0.5700	0.5700	200
weighted avg	0.5700	0.5700	0.5700	200

Figure 12: LSTM (not using FastText) Classification Report Phase 2

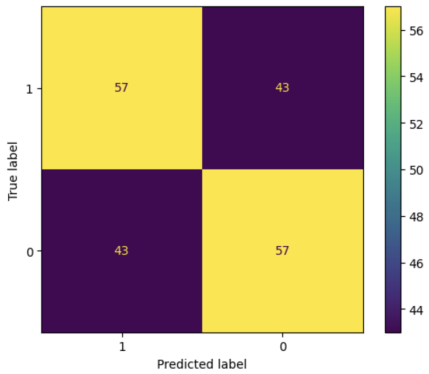

Figure 13: LSTM (not using FastText) Confusion Matrix Phase 2

Using FastText Embedding Pretrained Model

At epoch 6, we could find the best model which achieved 0.5779 of the F1-score on the validation set.

Classification Report:

	precision	recall	f1-score	support
1	0.5930	0.5100	0.5484	100
0	0.5702	0.6500	0.6075	100
accuracy			0.5800	200
macro avg	0.5816	0.5800	0.5779	200
weighted avg	0.5816	0.5800	0.5779	200

Figure 14: LSTM (using FastText) Classification Report Phase 2

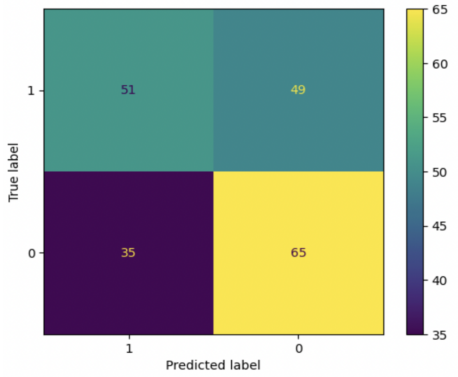

Figure 15: LSTM (using FastText) Confusion Matrix Phase 2

5.3 CNN

Phase 1

At epoch 8, we could find the best model which achieved 0.5800 of the F1-score on the validation set.

Classification Report:

	precision	recall	f1-score	support
1	0.5784	0.5900	0.5842	100
0	0.5816	0.5700	0.5758	100
accuracy			0.5800	200
macro avg	0.5800	0.5800	0.5800	200
weighted avg	0.5800	0.5800	0.5800	200

Figure 16: CNN (using FastText) Classification Report Phase 1

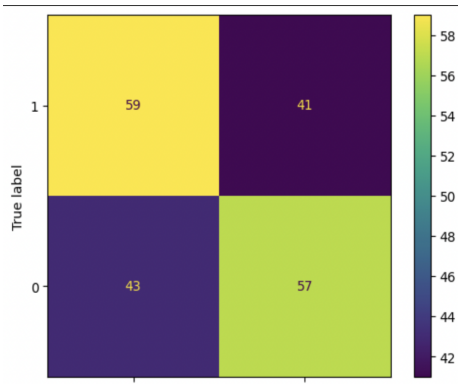

Figure 17: CNN (using FastText) Confusion Matrix Phase 1

Phase 2

At epoch 2, we could find the best model which achieved 0.5931 of the F1-score on the validation set.

Classification Report:

	precision	recall	f1-score	support
1	0.6351	0.4700	0.5402	100
0	0.5794	0.7300	0.6460	100
accuracy			0.6000	200
macro avg	0.6073	0.6000	0.5931	200
weighted avg	0.6073	0.6000	0.5931	200

Figure 18: CNN (using FastText) Classification Report Phase 2

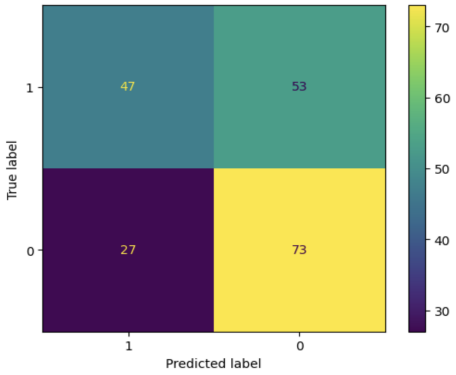

Figure 19: CNN (using FastText) Confusion Matrix Phase 2

By seeing these results, we could say that the CNN model is much work well than LSTM model. However, from 20 there is overfitting of the training model, like the more epoch the training loss increases.

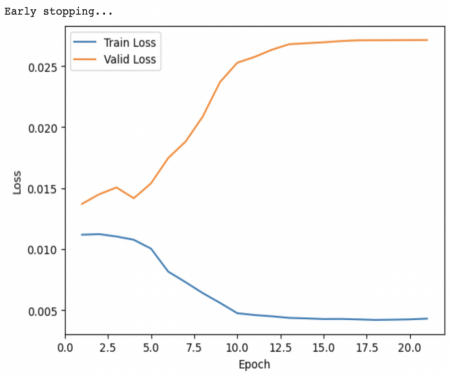

Figure 20: Training Loss

5.4 Multi-Classification

Below 21, is the result of tuning the bert-based-cased model with our 20 types of messages.

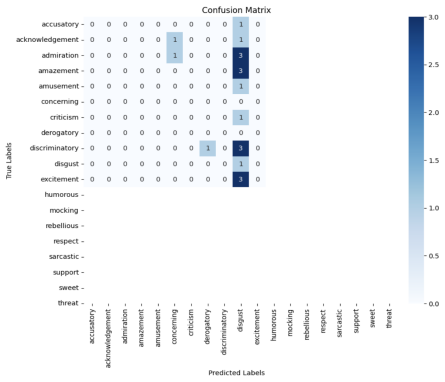

Figure 21: Multiclassification Result 1

We could see that the result is not so satisfactory because the test processing is not cover all the possible labels as well as the result almost sometimes predict in the wrong class as well. However, we believe that this is because of the small amount of data, if we have a big amount of data our multi-classification tuning model could be improved a lot.

Because some labels have just 5 to 7 messages so it is convenient to combine them as one and apply them to the model. So, in this step, we combine the label, "respect" and "excitement" as "sweet" then "rebellious" and "accusatory" as "threat", after that "derogatory" as "mocking", and finally "disgust" as "Discrimination". Below is the result after combining some labels together. After modifying the training dataset finally we got at epoch 10 the training loss is 0.7618 and validation loss is 2.4345. Below, the result of testing the model with validation set of 20 sentences from TikTok and YouTube.

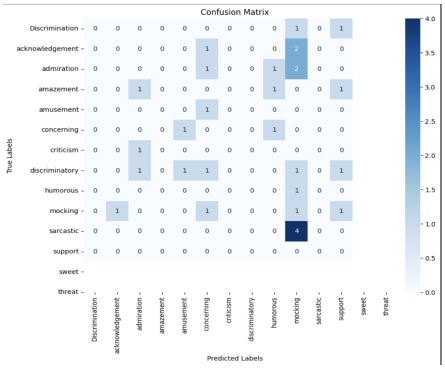

Figure 22: Multiclassification Result 2

6 Tuning Hyperparameter of Binary Classification Deep Learning

From 20, we could see overfitting of the training model, like the more epoch the training loss increases. Therefore, this time we try to fine-tune the model by lowering the hyperparameter, especially with different values of learning rate. And at this step, we only try to tune with the model training fastText embedding, as from previous results, we could see the impact of fastText contributing to the model's learning.

Epoch, Learning Rate, and batch Size

To understand the characteristic of model learning we try to use more epochs than before which are 50 epochs with the same batch size as the previous which is 64 batch size. For batch size, we also try with different values but still, this value is the best one for this task. The suitable learning rate for each model right now is 0.002, we try both ways to increase and decrease the value of the learning rate from the previous one which is 0.005 and finally, we see this new value (0.002) the model could achieve the highest result compared to using other values.

Layers, Embedding Size, Hidden Size, F1-Score

Model	Layer	Embed Size	Hidden Size	F1-Score
CNN	1	32	32	61
	1	32	64	60
	1	50	32	**63**
	1	50	64	61
	2	50	32	55
Bi-LSTM	1	50	32	61
	2	32	32	62
	2	32	64	**65**

Table 4: Different Hyperparameter Testing Values

After trying with various numbers of each hyperparameter result in table 4 we could find the best CNN, which achieves 63% of the F1-score on the validation set, and CNN is training with 1 layer, 50 fastText embedding sizes, and 32 hidden sizes. While the best Bi-LSTM, which achieves 65% of the F1-score on the validation set, is training with 2 layers, 32 embedding sizes, and 64 hidden sizes.

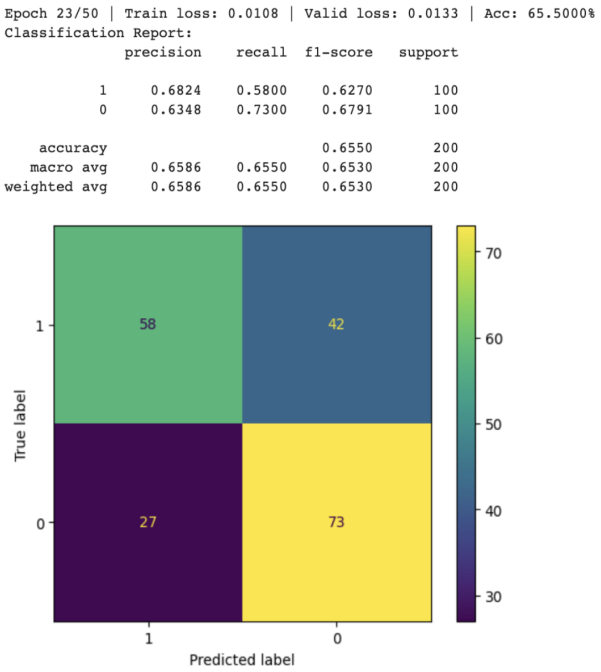

Figure 23: Bi-LSTM Training Result

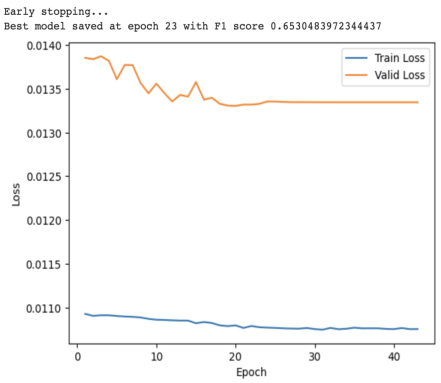

Figure 24: Bi-LSTM Training Loss

Even CNN could improve compared to the CNN first training pipeline result, but if we compare the CNN with Bi-LSTM second training pipelines, we could see Bi-LSTM more robust to the simple model as it outperforms when small configuration and hyperparameters are used, while CNN could improve but cannot outperform Bi-LSTM as in the above result. Bi-LSTM could achieve it easily. Noticeably, when increasing the layer of CNN, it could increase the performance, but the computation time took longer than Bi-LSTM. The way CNN learns is much more complex as it learns to capture local patterns, so the process may take longer, and if it's a small task (small amount of dataset, etc.), it may not learn very well as well.

Below, is the final configuration of each best model for our task:

Bi-LSTM	**Value**
FastText Embedding	32
Bi-LSTM Layer	64
Activation Function	NA
Dropout	0.3
Dense Layer	64
Loss Function	crossentrophy
Optimizer	Adam

Table 5: Hyperparameters for Bi-LSTM model

Embedding Layer: The model uses an embedding layer to convert input tokens into dense vector representations. The embedding layer is initialized with pre-trained word embedding and weights are not trainable, which means the pre-trained embeddings will not be updated during training.

Bi-LSTM Layer: The Bi-LSTM layer takes the embedded input sequence and processes it in both forward and backward directions. It has 64 memory units in each direction and there are 2 layers.

Dropout: After the Bi-LSTM Layer, a dropout layer with a dropout rate of 0.3 is applied. Dropout helps in regularizing the model by randomly setting a fraction of the output values to zero during training, which helps prevent overfitting.

Dense Layer: The output of the dropout layer is fed into a linear layer (fully connected layer) with the 64 memory unit. This layer serves as the decision layer and maps the hidden state of the LSTM to the number of classes.

Loss function: CrossEntrophyLoss is a commonly used loss function for multi-class classification tasks.

Optimizer: Adam is an optimization algorithm that combines the benefits of AdaGrad and RMSProp. It is widely used in deep learning for its adaptive learning rate capabilities. Activation Function: in this case, the output of the model is returned as is, without any activation function being applied. This implied that the model assumes the final hidden states values will be used directly as the logits for classification.

CNN	Value
FastText Embedding	50
Convolutional Layer	32
Kernel Size	2
Stride Size	1
Activation Function	ReLU
Pooling Layer	max_pool1d()
Dropout	0.3
Dense Layer	32
Loss Function	crossentrophy
Optimizer	Adam

Table 6: Hyperparameters for CNN model

Convolutional Layers: The model uses a single convolutional layer with a kernel size of 2 and strike of 1. The conv layer convolves over the input embeddings to capture local patterns or features in the data.

Activation function: the output of the convolutional layer is passed through a ReLU activation function to introduce non-linearity and allow the model to learn complex representations.

Pooling layer: the output of the activation function is then passed through a max pooling layer with a kernel size equal to the size of the input. This operation reduces the dimensionality of the output and captures the most salient features.

And every remaining hyperparameter describes the purpose the same as in Bi-LSTM.

Computation Time

Interestingly, when we try to reduce the dimension, the computation time is really fast and convenient, as training for 50 epochs takes about 2 minutes, and we can achieve 65% of the F1-score on testing on the validation set. This is very useful and practical, as before we took 5 minutes to complete the task. The disadvantage of high dimensions is computation, and it is a very complex calculation, so it easily gets overfitted like previous training losses20 .

7 Qualitative Analysis of Binary Classification Deep Learning

In this step, we also take a look into how the model will predict more specifically because we will take a few messages from an online resource [4] that already has some datasets about hate and non-hate speech. So, we just took a few interesting messages to apply to our model to see how it predicts. This way we will see the messages and predict them as what we think they should be and analyze how the model will predict each token weight contribution. Therefore, we have got several sentences below.

Sentence 1: "🤤 I need a bitch thats gon pay all my bills I dont like broke women 🤢 🚮 😷 🚮 🤧 " (Ground Truth= Hate)

==> CNN = Hate, Bi-LSTM = Hate, "humorous"

In this one we try to delete each token to see which token contributes to making the model predict it as hate messages. At first, we think that emojis are the most weight value that make the model predict them as hate messages but surprisingly, the emojis cannot make a change to the model prediction and we try to delete every token until this "I need a thats pay all my I like women" After that, CNN still predicts it as hate, but Bi-LSTM predicts it as non-hate but if we add token "bills" Bi-LSTM predicts it as hate. So, it means tokens "bills" have the most weight value compared to others even the emojis. Then, we delete the token women and CNN predicts it as non-hate.

Sentence 2: "When you're livin life on the edge but safety always comes first... 🌚" (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, "Sarcastic"

This one we may think that by adding this emoji 🌚 the model might think of it as a little hate message but its okay for both models with this type of message.

Sentence 3: "DEPORT RAPEFUGEE'S!!!👿 👿 👿 👿 👿 👿 👿 👿 " (Ground Truth= Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, "Sarcastic"

Sentence 4: "REFUGEES NOT WELCOME👊 👊 👊 " (Ground Truth= Hate)

[4] https://www.kaggle.com/code/kerneler/starter-seven-nlp-tasks-with-twitter-fbb2ee39-0/input

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “Sarcastic”

For Sentences 3 and 4, even if we replace these emojis with other hate emojis such as 😤 😡 😠 but still both models predict it as non-hate speech, so maybe because of the written message itself the models have not deal with this capital letter yet.

Sentence 5: “@user stop controlling me!!!!!😝 😝 😝 😝 😝” (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “amusement”

When there are no emojis we would consider it a hate message but when we put those emojis we can consider it as a joke and the model could understand the meaning of these emojis as well. Moreover, we try to understand the model emoji’s weight contributes to this prediction so we test it without adding the emojis like this “@user stop controlling me!!!!!” and CNN still thinks it is non-hate while Bi-LSTM predicts it as hate. So this time we could see Bi-LSTM predict better than CNN.

Sentence 6: “ Im fckin dead @user 😹💀 I fckin miss you hoe slut skank beetch 😘💖 🤞 ” (Ground Truth= Non-Hate)

==> CNN = Hate, Bi-LSTM = Hate, “mocking”

Bi-LSTM is robust to “skank” and by this token, it predicts that this message it as hate but in this case we already delete “fckin, dead, and all emojis”. We also delete everything until this “@user I you” we could find out that because of “@user” that make CNN predict it as hate.

Sentence 7: “ ’Replaceable’ best describes your life🤗” (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Hate, “criticism”

This one just the token “replaceable” could make Bi-LSTM think it is a hate message.

Sentence 8: “When they REALLY don’t want you to advertise😂 😂 😂” (Ground Truth= Non-Hate)

==> CNN = Non-Hate, Bi-LSTM = Non-Hate, “mocking”

This one not even the emojis that contribute to non-hate speech but each token in this sentence is all positive.

8 Results and Discussion

For training the deep learning model, we tried several complex configurations to apply to the model, such as increasing the epoch, batch size, increasing more layers, and batch normalization, but the result is even worse than before. We could see from 20 the model kept increasing the loss when we tried to build a model with the complex configuration. However, training the complex does not always get a good result; the best thing is to find suitable hyper-parameters for our work. As we can see, our first training pipeline with complex hyper-parameters took too long to train with accuracy lower than the second tuning training pipeline, which was fast and a lot outperformed. The most important part of this task is word embedding. We try to keep the size of the embedding large and play around with all other hyper-parameters, but the final result is still at most 60%. When we just changed the size of the embedding to smaller, everything improved, and we could use the small value of hidden size, but the result was above the previous one. When we increase the number of layers in Bi-LSTM, it could get almost 65% compared to before; if we increase the size of the layer, it will achieve only around 40 percent. The embedding of a small size 32 may not include much more information, while 64 is not too much or too little to capture the information of the input sentences. We also tried with the 100 embedding sizes, but the result was even worse and it took too long to finish the training. Therefore, an embedding size of 64 is the best one. Moreover, we could see a struggle for both models to analyze the token in all capital letters, and they had never learned it before, so they completely generated the wrong result. Also, each token has its own weight, as in some sentences, even if we delete emojis, it could still predict correctly. And we could find out that CNN and Bi-LSTM quietly work well with each other, as some sentences naturally suppose that it is hate, but CNN predicts it as non-hate while Bi-LSTM predicts it as hate, and vice versa. Finally,

we could see that in the quantitative analysis result, Bi-LSTM outperformed CNN, but in the qualitative analysis with non-seen messages, we could see that the performance of each model is quite similar to each other. However, the training time for CNN is a little bit higher than for Bi-LSTM.

9 Conclusion

References

Maha Jarallah Althobaiti. 2022. Bert-based approach to arabic hate speech and offensive language detection in twitter: Exploiting emojis and sentiment analysis. *International Journal of Advanced Computer Science and Applications*, 13(5):972–980.

Hannah Kirk, Bertie Vidgen, Paul Rottger, Tristan Thrush, and Scott Hale. 2022. Hatemoji: A test suite and adversarially-generated dataset for benchmarking and detecting emoji-based hate. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1352–1368, Seattle, United States. Association for Computational Linguistics.

Hamdy Mubarak, Sabit Hassan, and Shammur Absar Chowdhury. 2022. Emojis as anchors to detect arabic offensive language and hate speech. *Natural Language Engineering*, pages 1–21.

Juan Carlos Pereira-Kohatsu, Lara Quijano Sánchez, Federico Liberatore, and Miguel Camacho-Collados. 2019. Detecting and monitoring hate speech in twitter. *Sensors (Basel, Switzerland)*, 19.

Anna Schmidt and Michael Wiegand. 2017. A survey on hate speech detection using natural language processing. In *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*, pages 1–10, Valencia, Spain. Association for Computational Linguistics.

Matthew Shardlow, Luciano Gerber, and Raheel Nawaz. 2022. One emoji, many meanings: A corpus for the prediction and disambiguation of emoji sense. *Expert Systems with Applications*, 198:116–862.

Zeerak Waseem, Thomas Davidson, Dana Warmsley, and Ingmar Weber. 2017. Understanding abuse: A typology of abusive language detection subtasks. In *Proceedings of the First Workshop on Abusive Language Online*, pages 78–84, Vancouver, BC, Canada. Association for Computational Linguistics.

Michael Wiegand and Josef Ruppenhofer. 2021. Exploiting emojis for abusive language detection. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 369–380, Online. Association for Computational Linguistics.