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 Krik dataset and tested with a new dataset that is crawled from social media. Then this model will be qualitatively evaluated as well.

2 Dataset

Given a recently dedicated dataset (Kirk et al. (2022)), additional sets of social media messages

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

utilizing emojis have to be scraped from social media(TikTok and YouTube). Scraped messages have to contain both hate and normal speech usage. 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 1. Noticeably, the most agreement labels between two annotators are concerning labels while respect labels are the lowest agreements between two annotators.

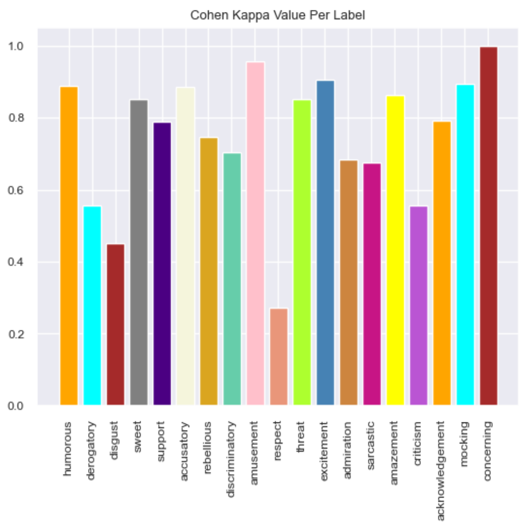


Figure 1: Cohen Kappa Value Per Label

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

The correlation annotation result is provided here 2. 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.

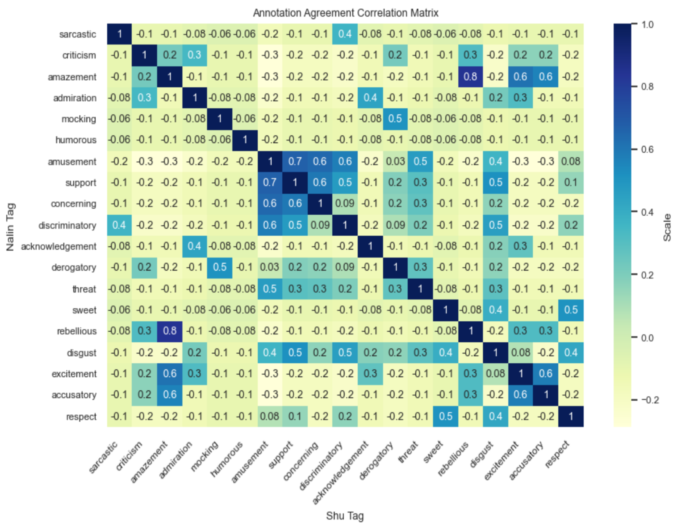


Figure 2: Annotation Agreement Correlation

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 3 is the plot to show the count of different types of messages that have been annotated.

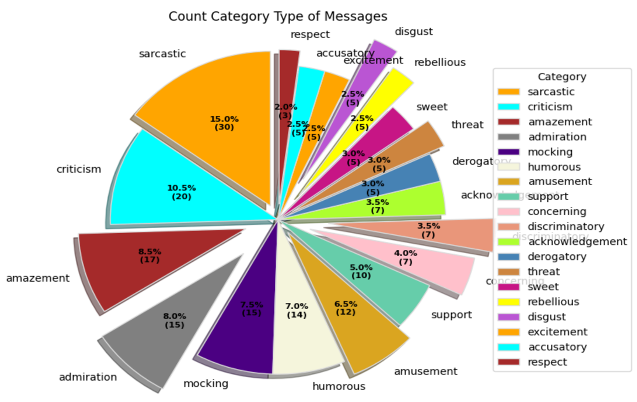


Figure 3: Messages Count Categories

Notably, the 10 most emojis used in hate 4 and non-hated 5 speech have a little different from each other.

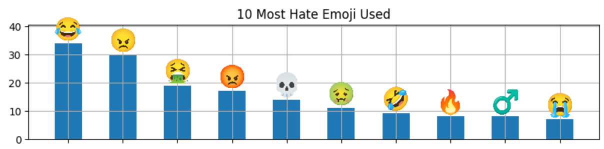


Figure 4: 10 Most Hated Emojis Used

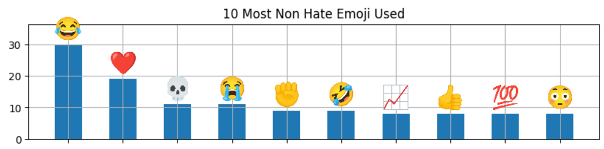


Figure 5: 10 Most Non-Hated Emojis Used

3 Method

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

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

Table 1: Dataset Preparation

3.1 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

3.2 Proposed of Machine Learning Model

3.3 Proposed Architecture of Deep Learning Model

After tuning parameters several times with our model finally we could find this is the suitable parameter and configuration to train our 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.

3.3.1 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.

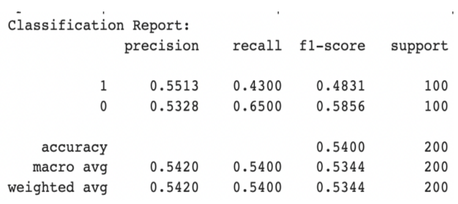
Phase 1

Below describing the dataset we organize to build our model:

Domain	Dataset	Sentences
Krik	Train	5316
TikTok + YouTube	Test	200

Table 2: Dataset Preparation Phase 1

At epoch 6, we could find the best model for us which achieved 0.5344 of the F1-score.



```
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 6: LSTM (not using FastText) Classification Report Phase 1

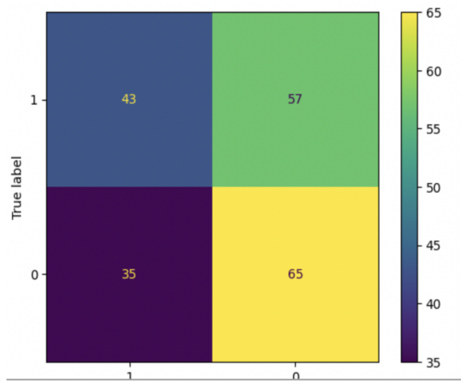
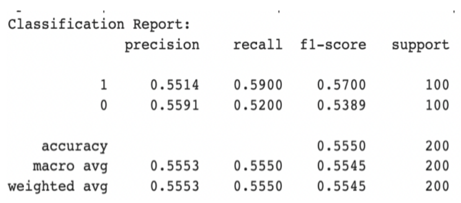


Figure 7: LSTM (not using FastText) Confusion Matrix Phase 1

Using FastText Embedding Pretrained Model

At epoch 6, we could find the best model for us which achieved 0.5545 of the F1-score.



```
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 8: LSTM (using FastText) Classification Report Phase 1

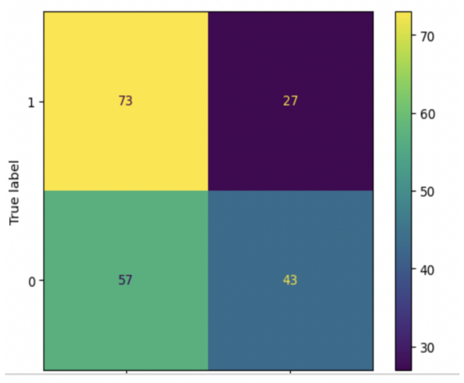


Figure 9: LSTM (using FastText) Confusion Matrix Phase 1

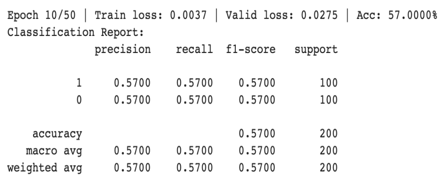
Phase 2

Below describing the dataset we organize to build our model:

Domain	Dataset	Sentences
Krik	Train	5910
TikTok + YouTube	Test	200

Table 3: Dataset Preparation Phase 2

At epoch 10, we could find the best model for us which achieved 0.5700 of the F1-score.



```
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 10: LSTM (not using FastText) Classification Report Phase 2

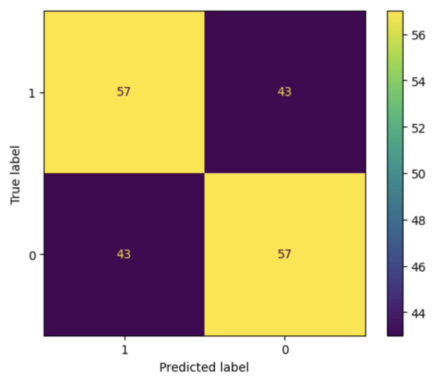
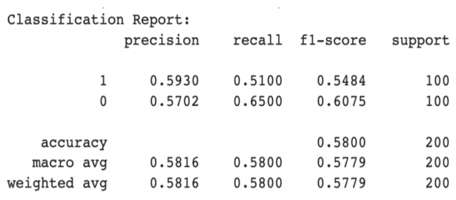


Figure 11: LSTM (not using FastText) Confusion Matrix Phase 2

Using FastText Embedding Pretrained Model

At epoch 6, we could find the best model for us which achieved 0.5779 of the F1-score.



```
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 12: LSTM (using FastText) Classification Report Phase 2

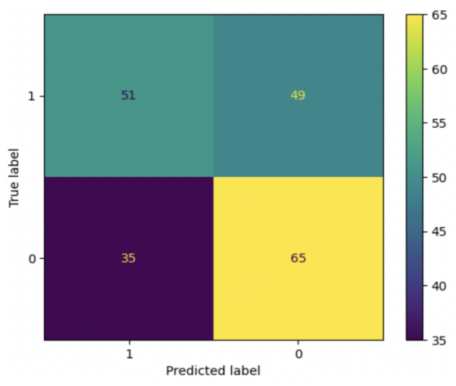


Figure 13: LSTM (using FastText) Confusion Matrix Phase 2

3.3.2 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 from the previous LSTM training result 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.

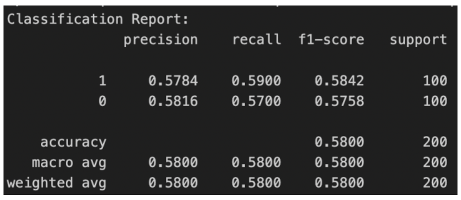
Phase 1

Below describing the dataset we organize to build our model:

Domain	Dataset	Sentences
Krik	Train	5316
TikTok + YouTube	Test	200

Table 4: Dataset Preparation Phase 1

At epoch 8, we could find the best model for us which achieved 0.5800 of the F1-score.



```
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 14: CNN (using FastText) Classification Report Phase 1

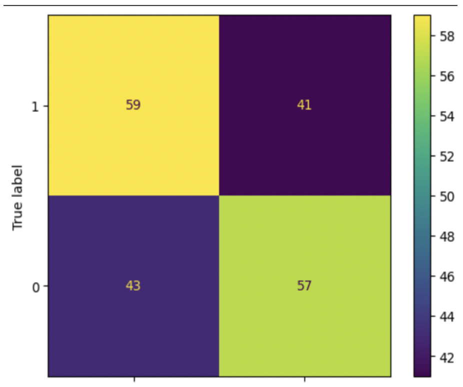


Figure 15: CNN (using FastText) Confusion Matrix Phase 1

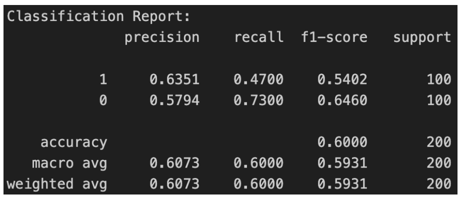
Phase 2

Below describing the dataset we organize to build our model:

Domain	Dataset	Sentences
Krik	Train	5910
TikTok + YouTube	Test	200

Table 5: Dataset Preparation Phase 2

At epoch 2, we could find the best model for us which achieved 0.5931 of the F1-score.



```
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 16: CNN (using FastText) Classification Report Phase 2

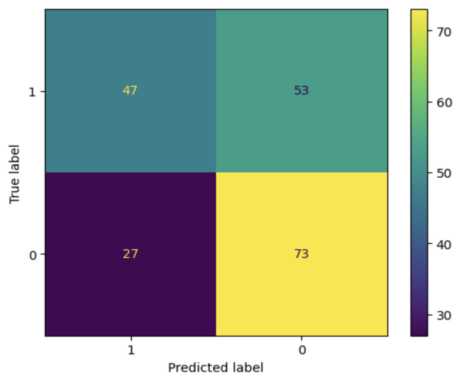


Figure 17: CNN (using FastText) Confusion Matrix Phase 2

By seeing these results, we could say that the CNN model is much work well than LSTM model. So then, from now on we still only focus on using and improving CNN model.

3.4 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 Krik dataset because we have almost 20 categories of messages while Krik 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. Below, is the result of tuning the bert-based-cased model with our 20 types of messages.

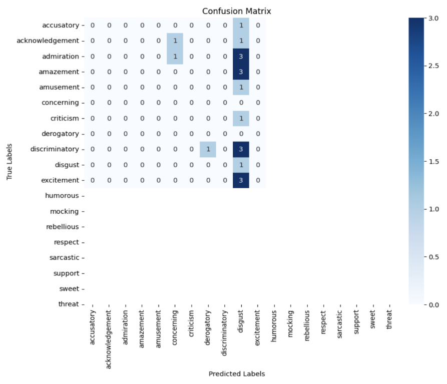


Figure 18: Training Loss

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.

4 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. Below we could see the model keep increasing the loss when we tried to build a model with the complex configuration.

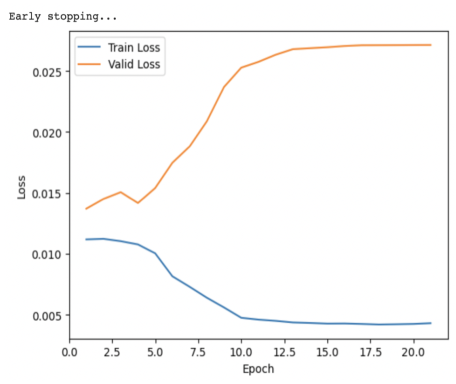


Figure 19: Training Loss

However, the only improve things to improve the performance of the model is to increase the amount of training dataset. As we could see from the result of various training Phase 1 and Phase 2 from the section (Proposed Architecture of Deep Learning Model), the result keeps increasing when we add more training datasets. Additionally, when we have a big amount of dataset we could play around with different complex configurations to train our model to see the performance as well. Moreover, even FastText contributed to increasing the performance but we could see the model file is about 4G which means it covers million amount of text sometimes it provided the feature embedding information to our specific text is too general. So, if we have time we could consider fine-tuning the FastText model to work on our specific dataset it may improve the performance of our model more as well. Moreover, if we could obtain a big amount of datasets we could take the tuning multi-classification of the Hugging face into consideration as well, and the model performance may be improved. Additionally, we could see that CNN outperformed LSTM bidirectional, while LSTM bidirectional models are generally well-suited for capturing sequential dependencies in text, CNN architectures can also have advantages in text data with emojis. There are some reasons that CNN works better than LSTM, emojis can carry visual information and convey sentiments or emotions. CNN is particularly learning to associate specific emojis with certain sentiments so by treating emojis as visual features, CNN can effectively incorporate them into the text representation. Second, regarding the local patterns and context, emojis are often used to specific words or phrases to enhance the intended meaning or emotional expression. CNN is good at capturing local patterns and combinations of words, phrases, and emojis that contribute to the classification task. CNN can potentially better understand the relationship between emojis and the surrounding text. Third, due to the unique representation of emojis, it may introduce irregularities during the training model but CNN with the ability to capture local features and handle variations in input, can be more adaptable to different emoji placements and variations across the input text more effectively than LSTM. Then, CNN can learn to assign appropriate weights to different visual features as CNN is known for its ability to automatically learn information representations from data and when dealing with large datasets CNN is much more computational efficiency while LSTM only deals with a limited amount of data. Moreover, training speed CNN can process multiple input regions and making them faster to train and evaluate compared to sequential models like LSTM.

5 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.