Deep Learning Project

Murari Prasad
murari.prasad@ucdconnect.ie
Student No: 24200412

Abstract

This project aims at comparing models to perform text classification of the Twitter Emotion Dataset. Text classification is a well known domain in deep learning, and in the current era of Transformer models and Generative AI, this project is a basic exploration of the simpler methods. Text Classification has been performed on pre-trained glove-25 embeddings using a sequentially accomodating CNN network, and then for a more traditionally sequential method, a basic Bi-Directional RNN network is used. Both attempts have been made using Keras Tuner to find the best network architecture and hyperparameters. This project has been implemented using Tensorflow.

1 Introduction

Deep Learning models and their impact in Natural Language Processing is well known. Since the introduction of attention based modeling and Transformer networks in 2017 [1], many approaches have emerged, and have lead to the advent of products such as ChatGPT using Generative Pre-trained Transformers [2]. In terms of Text Classification, Google's BERT model and its variants have shown excellent performance [3, 4]. This project uses more simpler techniques to perform text classification.

Convlutional Neural Networks (CNNs) are well known in computer vision, showing excellent performance in pattern recognition tasks. CNNs have also been experimented on NLP tasks as well.In this project, Kim's work from 2014 [5] has been referenced for a CNN implementation as the first approach to set a baseline.

Recurrent Neural Networks (RNNs) are the most rudimentary of sequential models. All the well known models such as Long-Short Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks, etc. and even Transformer networks are incremental improvements to the RNNs. Most Text classification tutorials start with RNN methods. This project also follows the same logic and takes inspiration from the Tensorflow's RNN tutorial for the second approach, to see if we immediately see improved results over the baseline once we shift to a method meant for the task.

In this project, the focus is to use light models, and comment on the performance we see and the trade-offs on the same. For both the above methods, the data is pre-processed into word-embedding vectors, using a pre-trained word embeddings provided by gensim python framework [6].

2 Related Work

Yoon Kim's paper,'Convolutional Neural Networks for Sentence Classification' explores implementing CNNs for text classification very deeply. It derives from previous works in NLP and proposes a CNN with multiple kernels to take into account that text data has a sequence element; words appear in certain orders.

There has also been work on what type of features can be learned from a convolutional layer in text data 7. Are n-grams being captured, is the sequence of word occurence captured, like an article followed by a noun, etc. Also the effectiveness of embedding choices and optimizing Yoon Kim's approach was experimented in another research [8].

In terms of sequential modelling for Text Classification, Google's BERT model and its variants have shown excellent performance [3, 4]. These were not looked into with much depth as the scope of this project, as stated previously, is different.

3 Experiment

3.1 Setup

This project was run locally on a machine with 40GB DDR5 5600Mhz Memory, 8GB Nvidia RTX 4060 mobile GPU and 8C/16T AMD Ryzen 7 laptop processor.

For the project environment, Python 3.10 with Tensorflow 2.18 was used. Refer to requirements.txt for all the packages used. Project-wide random seed set to 2025. Data batch size set to 256.

3.2 Dataset

The Twitter Emotion Dataset provided was used [9]. The dataset provided already has been cleaned, so only a few additional cleaning steps have been performed. Stop words are removed. Special characters such as punctuation marks are removed. Also due to the nature of the dataset, we have records containing only stop words, which were dropped.

Few basic exploratory steps have been performed after the cleaning. The distribution of the target variable 'emotion' was checked and found to be evenly distributed (approximately equal records for every class). The distribution of the count of tokens/words per record was checked. 90% of the data has less than 14 words. This was used when processing the data into embeddings.

The dataset is pre-processed into a vector of word embeddings. The Glove-25 [10] pretrained model provided by gensim [6] was used for this. The '25' in Glove-25 stands for the embedding vector dimension. So for every word, a vector of size 25 is used as a representation. The embedding is performed iteratively over the dataset, and every $record > 14$ was truncated at 14 words, while the smaller records were padded with zero vectors of size 25. This truncation helps in computation and with memory, as the largest record was with 119 tokens, most of which was emojis. Explicit emoji handling was not done. The dataset represents emoji as text such as $smilyface$, $frownyface$, etc. They were either embedded or represented with a zero vector if embedding failed.

The entire dataset was embedded and stored on disk to use for training.

3.3 Keras tuner Setup

Keras Tuner is a hyperparameter optimization framework provided in Tensorflow Keras for hyperparameter searching [11].It comes with multiple algorithms built-in and various options to control and modify the search process. It can be used to even find good model architectures, with the architecture itself being treated as a hyperparameter(layers, number of units, activation function, etc.)

For both, the CNN baseline and the RNN models, the Keras Tuner was setup with the following configuration:

- Hyperband [12] was used as the parameter search algorithm.
- Adam for optimizer and Sparse Categorical Crossentropy for loss.
- The tuner objective was set to maximize $val\_accuracy$.
- max_epochs was set to 25.
- The reduction factor; number of models to drop per round of hyperband, was set to 3.
- An early stopping callback was set on validation loss, with $patience$ = 3 epochs and $restore\_best\_weights$ = True

Keras Tuner requires a function which accepts Hyperparameter instance as an argument, and the network is built using the same. Both models have their own respective model building functions, described in the sections ahead.

3.4 CNN Model

The $build\_cnn\_model()$ function is used for finding the best model and hyperparameters for the CNN model. The model will contain 3 separate convolution layers (suffixed 0, 1, 2. ex: filters_0 is for conv_0) with incremental kernel sizes, just like in Kim's paper, followed by a max pool with the same size as the kernels, which is then flattened and concatenated. Then finally connected to a softmax layer for the output. The best hyperparameters(table 1) and model(Figure 1) obtained by Keras Tuner are given below.

Parameter	**Value**
learning rate	0.001
activation function	gelu
dropout	0.1
padding	valid
kernel sizes	”3,4,5”
filters_0, strides_0	96, 3
filters_1, strides_1	160, 1
filters_0, strides_0	832, 1

Table 1: CNN hyperparameters

Following is the model architecture summary generated within tensorflow

3.5 RNN Model

The $build\_rnn\_model()$ function is used for finding the best model and hyperparameters for the RNN model. The model will contain 2 Bi-directional RNN layers (suffixed 0 & 1), with less units in the 2nd layer compared to the first. A final softmax for the output layer. The best hyperparameters(table 2) and model(Figure 2) obtained by Keras Tuner are given below.

Parameter	**Value**
learning rate	0.0001
activation function	gelu
dropout	0.1
recurrent_dropout_rate	0.1
rnn_units_0	672
rnn_units_1	240

Table 2: RNN hyperparameters

Following is the model architecture summary generated within tensorflow

3.6 Training & Evaluation

Both models were built using their respective best hyperparameters and architectures obtained, and set for training, capped to 100 epochs with an early stopping callback, patience set to 10 epochs on validation loss and restore_best_weights = True. Adam Optimizer was used with Sparse Categorical Crossentropy as the loss function. The CNN model training stopped at 17 epochs and the RNN model at 30 epochs.

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 14, 25)	0	-
conv1d (Conv1D)	(None, 4, 96)	7,296	input_layer[0][0]
conv1d_1 (Conv1D)	(None, 11, 160)	16,160	input_layer[0][0]
conv1d_2 (Conv1D)	(None, 10, 832)	104,832	input_layer[0][0]
max_pooling1d (MaxPooling1D)	(None, 1, 96)	0	conv1d[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 2, 160)	0	conv1d_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 2, 832)	0	conv1d_2[0][0]
flatten (Flatten)	(None, 96)	0	max_pooling1d[0]…
flatten_1 (Flatten)	(None, 320)	0	max_pooling1d_1[…
flatten_2 (Flatten)	(None, 1664)	0	max_pooling1d_2[…
concatenate (Concatenate)	(None, 2080)	0	flatten[0][0], flatten_1[0][0], flatten_2[0][0]
dropout (Dropout)	(None, 2080)	0	concatenate[0][0]
dense (Dense)	(None, 3)	6,243	dropout[0][0]

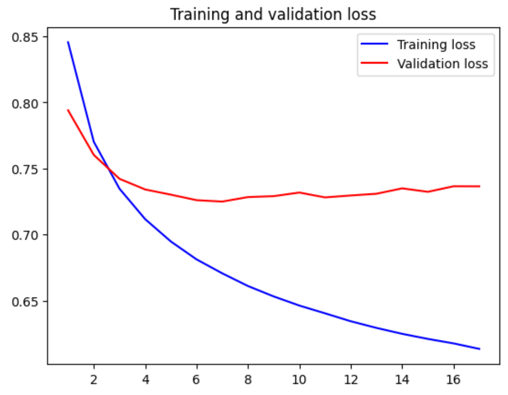
```
Total params: 134,531 (525.51 KB)

Trainable params: 134,531 (525.51 KB)

Non-trainable params: 0 (0.00 B)
```

Figure 1: Best CNN Model Summary

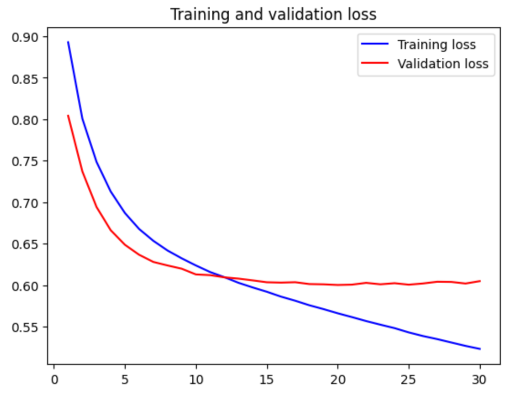
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 14, 25)	0
bidirectional (Bidirectional)	(None, 14, 1344)	938,112
bidirectional_1 (Bidirectional)	(None, 480)	760,800
dense (Dense)	(None, 3)	1,443

```
Total params: 1,700,355 (6.49 MB)

Trainable params: 1,700,355 (6.49 MB)

Non-trainable params: 0 (0.00 B)
```

Figure 2: Best RNN Model Summary



(a) CNN



(b) RNN

Figure 3: Training and Validation Loss Curves for both models

4 Results

To evaluate our models, Precision, Recall and F1-Score was calculated on the train, validation and test datasets. The metrics were calculated as the average over the metric calculated for every class (Average Precision, Average Recall and Average F1-Score).

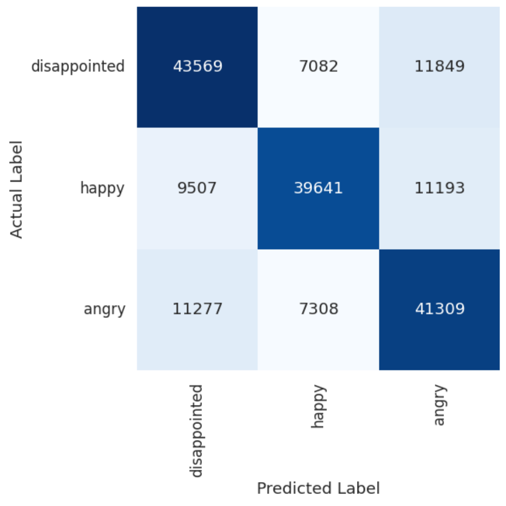
Metric	Train	Validation	Test
Precision	0.723	0.679	0.684
Recall	0.724	0.676	0.681
F1-Score	0.724	0.677	0.682

Table 3: CNN Model Performance

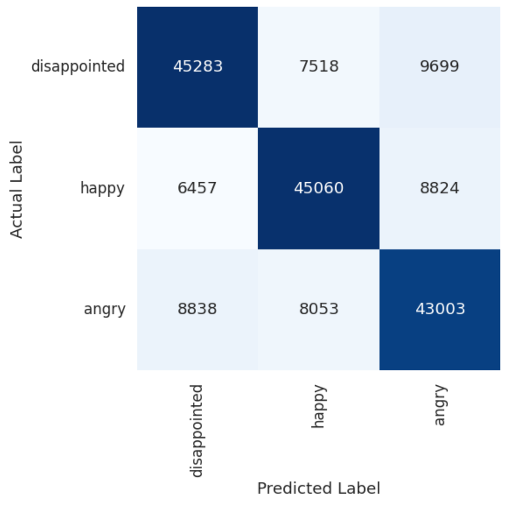
Metric	Train	Validation	Test
Precision	0.785	0.725	0.730
Recall	0.785	0.725	0.730
F1-Score	0.785	0.725	0.730

Table 4: RNN Model Performance

It is clear that the RNN model performed better, and also shows very consistent performance across the train-val-test split. The baseline CNN model's performance is not bad, considering a performance difference of approximately 7%, while the model is approximately 93% smaller than the RNN model; significantly lighter, faster to train and requiring less compute power to run.



(a) CNN



(b) RNN

Figure 4: Test Set Confusion Matrix for both models

The confusion matrix shows that the RNN is almost quite consistent across the classes. In the case of the CNN, there is significant disparity, with the lowest performance in classifying 'happy' records.

5 Conclusion & Future Work

Overall, the results do show that a very rudimentary RNN model outperforms our purpose built CNN. A tradeoff is to be considered when interpreting the results. The CNN is very light and trained very fast; it could be improved further without a very large increase in model size. On the other had using much more complex and well known sequential models and architectures will just be better in all scenarios, at the cost of increased training difficulty and computational costs. Depending on the scenario, you could go for the CNN approach in a low resource scenario, or the RNN/sequential model approach when results are all that matter.

An immediate next step from here, for the CNN would be to implement more optimised approaches to Kim's model, such as an element-wise addition approach instead of the max pooling [8]. In the sequential approach, replacing RNN units with LSTMs, or even going for attention-based Transformers would be the clear choice. Another upgrade path for both scenarios would be using different embeddings like word2vec or FastText and with larger sizes [13].

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[2] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI Blog*, 1(8), 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[4] Ali M. Areshey and Hassan Mathkour. Exploring transformer models for sentiment classification: A comparison of bert, roberta, albert, distilbert, and xlnet. *Expert Systems*, 41(3):e13701, 2024.

[5] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[6] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[7] Linyuan Gong and Ruyi Ji. What does a textcnn learn?, 2018.

[8] Haotian Xu, Ming Dong, Dongxiao Zhu, Alexander Kotov, April Idalski Carcone, and Sylvie Naar-King. Text classification with topic-based word embedding and convolutional neural networks. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '16, page 88–97, New York, NY, USA, 2016. Association for Computing Machinery.

[9] Ko Sweet. Kaggle: cleaned emotion extraction dataset from twitter. `https://www.kaggle.com/datasets/kosweet/cleaned-emotion-extraction-dataset-from-twitter`, 2020.

[10] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[11] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. `https://github.com/keras-team/keras-tuner`, 2019.

[12] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[13] Martina Toshevska, Frosina Stojanovska, and Jovan Kalajdjieski. Comparative analysis of word embeddings for capturing word similarities. 2020.