COMP47650 Deep Learning Project

Cian Ferriter
Department of Computer Science
University College Dublin
Belfield, Dublin 4
cian.ferriter@ucdconnect.ie

Abstract

Sentiment analysis and sentence classification is a hot topic within the Natural Language Processing (NLP) field of Deep Learning, traditionally there has been a tendency to focus on the binary nature of such classifications. This project endeavours to build an accurate model for predicting the sentiment of a multi-class level sentence based Twitter dataset. The dataset is a collection of 817k separate tweets each with their own individual labelling of either disappointed, angry or happy. In this project we present two model architectures each trained on two separate forms of data with the best model being the 'LSTM-CNN' based neural network which achieves an accuracy of 90.4%.

1 Introduction

This project aims to design and implement a number of different models in order to accurately predict the sentiment in sentences and tweets derived from the microblogging platform Twitter. The field of sentiment analysis has seen huge growth due to highly accessible well labeled data from across the internet.

Due to the nature of the dataset available to the project there will be 2 phases. The first will be the training and evaluation of models trained on the column which contains already pre-processed features. the second will be the training and evaluation of models trained on the original content column which will undergo its own pre-processing, the main differences between the two sets of data will be the inclusion of emojis in the already existing pre-processed column. The background for this approach derives from the fact that we are engaging with transfer learning and the pre-trained GloVE vectors we will be using do not contain elements for emojis, therefore their exclusion will make for an interesting development in seeing how the embedding matrix changes.

There are 2 architectures that will be evaluated. The CNN variant proposed by Kim [1] and an LSTM-CNN hybrid model proposed by Sosa [2]. Sosa's architecture attempts to reverse the standard hybrid model by placing an LSTM layer before the convolutional layer which went against common convention at the time, this architecture originally produced an increase in accuracy against the regular model approach of 2.7%-8.5% as shown in the paper and so will make for an interesting approach in the sentiment analysis task of this project .

The models will be constructed and trained using the Tensorflow framework whilst also utilising the Keras API built for Tensorflow.

Preprint. Under review.

2 Related Work

Artificial Neural Networks (ANNs) have been around for nearly 50 years but have only recently regained popularity in a renaissance due largely in part to increased computational power. The rapid advent of highly accessible quality data in the form of social media data extraction has also spurned an ever evolving interest in the technologies. Commonly also called Deep Learning (DL), DL algorithms and architectures are arriving at the point of rivalling human ability to accurately classify a chunk of textual data into categories of sentiment. Natural Language Processing (NLP) is the process by which we make computers and devices derive meaning from human language as a method of interacting with the real world [3]. NLP is not new and in fact has been around since Turing's era of computer exploration with Turing himself publishing in 1950 what we know today as the Turing Test - Can computers think? [4]. This was one of the first forays into NLP and the trend to enable computer, in particular deep learning models to accurately classify sentiment continues today.

The use of sentiment analysis on real world user data is an important tool for social media companies such as Twitter, a micro-blogging platform, who have important use cases for classifying their users/customer's sentiments and world view perspectives. Traditionally, literature has focused on the binary classification of sentiments, i.e. positive and negative classifications. This has its uses but also its limits, there appears to be a growing shift towards the use of multi-class labelling in sentiment analysis due to its more flexible nature [5] - after all we as humans experience a vast array of emotions and we most certainly can't be represented in a binary fashion.

There is a wealth of existing research and literature pertaining to sentence classification, NLP and sentiment analysis in the sphere of Deep Learning - The majority of existing research focuses on the use of Recurrent Neural Networks (RNN) in the form of Long Short Term Memory (LSTM) model topologys. There is also success with the use of the humble Convolutional Neural Network (CNN) achieving strong results in the field. However the focus of research appears to remain primarily in the use of LSTMs.

RNNs in the form of LSTMs were first introduced by [6]. They are particularly adept for the task of NLP as they have the ability to 'remember' previous inputs. LSTMs have the ability to maintain a memory cell to determine which important features to remember during the learning process but also importantly which unimportant features should be forgotten.

2.1 Embeddings

Embeddings form an integral part of building NLP language models. Word Embeddings are a type of dense representation of words and their relative meanings. They are generally a learned representation for text where words that have the same meaning have a similar representation. In most NLP projects there are two approaches to the word embedding layer. The first being using your existing dataset to learn your own embedding, this can often be a useful exercise but also computationally expensive. The other option is generally to implement a form of Transfer Learning by using pre-trained embeddings. There currently exists 2 dominant corpuses of pre-trained word embeddings and they exist in the form of Google's Word2Vec [7] and Stanford's GloVe [8].

2.2 Model Topology

As has been mentioned earlier LSTMs are the dominant architecture for use in NLP sentiment analysis. Good results are also achievable with CNNs. Interestingly there is also a number of bodies of work out there that present hybrid approaches i.e. CNN-LSTMS or LSTM-CNNs. [1],[2],[9],[10]. This approach seems to work in particular where there is more than two classes for prediction. As a result, this project will aim to build on this type of topology, however it is also within the scope to verify a number of different models for comparison. Therefore, the project will ultimately create 2 separate topologys, the CNN model based on [1] and finally the hybrid LSTM-CNN model based on [2].

3 Experimental Setup

The dataset this project uses contains 3 columns. The first column contains the ground truth vectors, that is the emotions which can be split into 3 sentiments - angry, disappointed or happy. The 2

remaining columns are labelled 'Content' and 'Original Content'. The 'Content' column consists of an already cleaned set of tweets whereby the author performed their own pre-processing on the data. The details of this pre-processing are not available to us, however, we can infer the steps that have been taken. Interestingly, in this column we can see that the emoji has been processed into a single string representation, this could prove useful as there is often a lot of useful sentiment that is contained in emojis which is usually lost through most Twitter data pre-processing. The final 'Original Content' column is exactly that, each row containing the original raw tweet.

The emotion labels were generated through the common method of search word hashtagging [5]. Hashtags on tweets generally correspond to the true nature of intent of the authors feelings. e.g. a tweet which looks like this: 'can't believe this.. #angry'. Without the hashtag there is very little context as to what the author is feeling or referring to but the hashtag allows us to get a true sense as to the emotional state of the writer. Resultantly, only tweets which originally contain a hashtag are contained in the dataset.

Initial data exploration was performed on the dataset to determine if any class imbalance might be present, class imbalances can negatively affect a model's performance and learning and as such it is important to determine the state of data before beginning any model construction. It is clear from fig. 1 that there is no such imbalance present in the dataset and that we are dealing with a near 33% even split across the labels. Although the 'disappointed label contains just over 1% more occurrences that the remaining two labels. The dataset was also validated to ensure that there were no null values present in the set and luckily this check proved a full set of data is present.

3.1 Pre-Processing

The raw tweet column is particularly useful as it allows us to perform our own pre-processing. In attempting our own pre-processing the following approach is taken: First, a stop word corpus is downloaded from NLTK library, stopwords are words like 'the', 'our', 'only' etc. A number of regex patterns are defined in order to remove certain phrases or keywords in the data, e.g. a Twitter username or 'handle'. Finally, word lemmatization is performed to extract the stems of words and remove tenses.

A mandatory aspect of working on a NLP task is the tokenisation of the data. Tokenisation is simply the process of breaking down our string tweets into smaller tokens. In our case we split the data into its tokens before applying a numeric value to represent the token, this is to enable the GloVe word embeddings to assign vectors and create maps of the words to generate a semblance of meaning from them.

A final step before creating embedding layers is padding. As is the case with most NLP projects we are presented with data of inconsistent lengths - which neural networks do not respond well to. This rings particularly true for Twitter data. Given that tweets can range from 1-280 characters means there is quite a bit of variation in the length of our tweets, we therefore must apply padding, this is the process of lengthening each tweets vector representation by the addition of zeros, these zeros mean that the data itself is not compromised but allows for a consistent input into the network. Keras employ a useful tool for such an action called pad_sequences, in this project we apply the padding at the end of the string.

3.2 Models

The first topology that the project developed is the CNN employed by Kim [1]. This architecture concatenates multiple Conv1d layers which are sent to a MaxPooling layer before being fully connected and activated by softmax. The convolution operations employ ReLu activation functions. Each layer has a varying kernel size, in our case 2 kernels of size 3, 8. The idea here is that different amount of proceeding words can be included in the kernel, i.e. a bigger kernel can 'see' more of the previous string representation and a smaller kernel can see word representations that are closer to the current word. Hyper-parameter tuning found that the most useful value of 0.5 is used in an effort to combat any overfitting and this strategy is successful. This project deviates slightly from the architecture described in the [1]. Here, we use GloVe word embeddings as opposed to word2vec employed by the paper. This is as result of tuning which found better results are attainable by making this change. I also reduce the number of layers from 3 to 2 in order to reduce the size of the model. This change does not negatively impact on performance. The resulting topology can be seen in fig

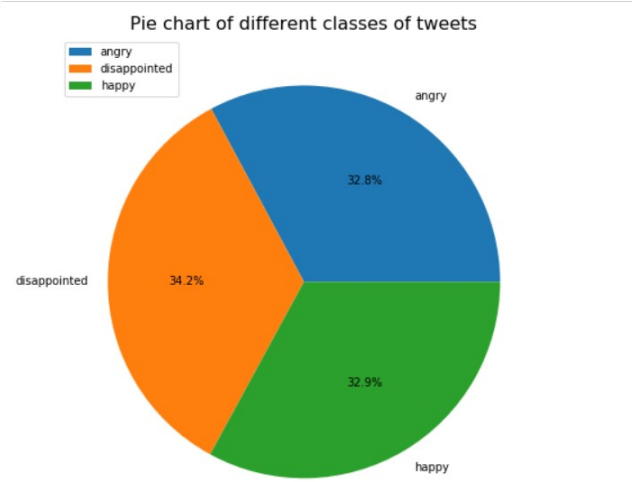

Figure 1: Balance of emotion classes contained in the dataset as represented by a pie chart.

2. As is the same in the original paper the Adam optimiser is used with a stable learning rate of 1e-3. The model is quite large with total parameters of nearly 47m however, because we employ GloVe learned embeddings only 2.3m of these parameters require training. We employ a loss function called SparseCategoricalCrossEntropy, this approach is proven successful in multi-classification problems. The model was initially run over 10 epochs which was reduced to 3 as it was observed during hyperparameter tuning that further epochs result in a plateau of learning before the model simply starts to fit. This is discussed in the results section.

The final topology the project implements is a variation of Sosa [2]. As stated, this approach flips convention around in that normally when combining LSTM and CNN layers it is a hybrid whereby the CNN layers come first. However, Sosa found improvements by first implementing the LSTM before going to the CNN. The LSTM operates with a tanh activation function while the CNN layers are ReLu activated. In our case and as is done in the paper the CNN aspect of the model is again based on [1]. The paper experiments with using both pre-trained word embeddings using GloVe and also with non pre-trained embeddings, i.e. those learned during training. We will simply stick with the GloVe embedding approach as done for the previous models. We employ a single 32 dimensional LSTM layer after the first word embeddings, this is in keeping with description provided in [2]. The LSTM layer is set up with a return_sequences set to 'True'. This ensures that the LSTM can produce outputs that are learnable for subsequent layers. In our case these layers are essentially the same as the CNN described previously, that is two simultaneous CNN layers of kernel sizes (3,8) which are concatenated –> Max Pooled –> Dropout –> Softmax. The final model topology can be seen in fig 3. Again, we use the Adam optimiser with a learning rate of 1e-3.

During training a validation set is created as a subset of the training data. The validation set contains 10% of the training data using the 'validation_split' parameter in the fit function. This method is very useful but it must be noted that the data should be shuffled prior to creation of training and test sets when this method is being used. In our case this is true as the data is shuffled.

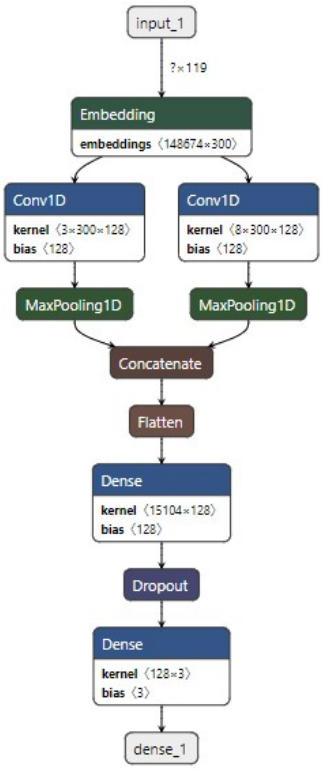

Figure 2: CNN Model Topology, constructed using netron (netron.app). The model employs separate convolutional layers that convolve across the words before being concatenated

4 Results

The main focus of our results will be upon the % accuracy achieved by the models on the testing set of data. Accuracy is utilised as the main performance metric due to its universal interpretability and is a common basis upon which to compare deep learning models across the board. We will also analyse model size and observe the loss of the models as they train. We focus primarily on the 2 models that achieve the best results, both of which are trained on our custom pre-processed data. A summary chart of all model results can be found in fig. 6

It can be seen that the models used achieve some really strong results, as can be seen in Table 2 the best results are achieved by the LSTM-CNN model on the custom pre-processed dataset, it achieves an accuracy of 90.4% on the test dataset. The CNN model is only fractionally worse with differences of 0.7% noted in Table 2.

As can be observed in fig. 4 the best validation results are achieved on the 3rd epoch, after this the model starts to overfit with training accuracy increasing and validation stagnating/decreasing, this is despite best efforts to combat overfitting using a heavy dropout layer of p=0.5. As such it was concluded that the model be trained over 3 epochs and the .h5 model for the CNN_PRE is a model trained over such. We can see that in respect to the other models, this is quite large, with a total of 5.9m trainable parameters making it the most sizable out of the different models that were trained throughout the project, however due to the nature of the model, i.e. it is solely a CNN, it trains relatively quickly, using Colab's Tesla T4 GPU the 3 epochs complete training in approx 10 mins, this is much shorter than any of the LSTM variations which take a considerably longer time to converge.

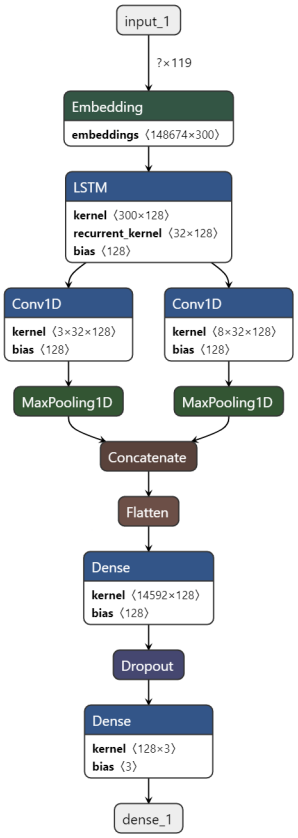

Figure 3: LSTM-CNN Model Topology, constructed using netron (netron.app). The model employs an LSTM layer followed by separate convolutional layers that convolve across the words with differing filter sizes before being concatenated

A similar trend of learning can be observed in fig. 5 with the LSTM-CNN model trained on custom data. the peak validation accuracy is achieved after epoch 3. Again, dropout adjustments are unable to produce any further performance enhancements. The final results of this model are saved in the 'date_LSTM_CNN_PRE.h5'. It came as a surprise that the LSTM-CNN, whilst fractionally better, did not produce any notable improvement in accuracy on the dataset. A major drawback of the LSTM models was observed on the basis of the time they take to learn, the addition of a single layer of the LSTM drastically increased training times (from minutes to hours) when compared to the CNN models. This is surprising also because of the fact that the inclusion of the LSTM layer reduces trainable parameters by 400k as seen in Tables 1&2.

4.1 Using the models for inference - interesting results

Once training was completed a number of custom inputs were used in order to cement the validity of the model. We create a function where strings can be sent in and have the model predict the sentiment. One thing that stood out to me to try was see how the model reacted to words or phrases that would have been prominent in the Twittersphere. One such result is the inclusion of 'Donald Trump' in any string. It is quite interesting to see the inclusion of the phrase make such a difference to an otherwise very positive string - it goes to show that models can only ever truly be as good as the data that is given to them. The below code listing shows how the prediction changes:

```
labels_dict = {0:'angry',1:'dissapointed',2:'happy'}
sample_string = ['Haha life is so fun and enjoyable!',
    'Haha life is so fun and enjoyable Donald Trump!']

def prediction_out(sample_string, pad_size):
```

Table 1: Models trained on existing pre-processed data

Part		
Name	Accuracy	Params (trainable) (m)
CNN	72.8%	2.35
LSTM-CNN	72.9%	1.95

Table 2: Models trained on custom pre-processed data

Part		
Name	Accuracy	Params (trainable) (m)
CNN	89.7%	5.9
LSTM-CNN	90.4%	5.5

```
    sample_string = tokenizer.texts_to_sequences(sample_string)

    sample_string = pad_sequences(sample_string,
                            maxlen = pad_size, padding='post')
    predictions = model.predict(sample_string)
    classes = np.argmax(predictions, axis = 1)
    for elem in classes:
        print(labels_dict[elem])

#call the prediction function
prediction_out(sample_string, pad_size)

#output of function
happy
angry
```

So we can see that the same sentence 'Haha life is so fun and enjoyable!', with the inclusion of the controversial name of Donald Trump completely reverses the sentiment - going from happy to angry, an interesting result!

5 Future Work

As can be seen the architectures presented above perform very well in the context of current state of the art models that exist today. This project primarily focused on the use of pre-trained word embeddings to engage with transfer learning. An interesting development from this would be to see how the models perform on an unlearned embedding layer i.e. one the models learn themselves. This would drastically increase the computation requirements but would generate an interesting comparison as the pre-trained GloVe models is not dataset specific, that is it is a 'universal' approach. One of the reasons the above approach might produce interesting results would be the increased importance of Emojis. In the original pre-processed data the emojis are kept in the data and are represented by a string. This string can be learnable, however because these complex strings which often look like 'facewithtearsofjoy' would not be included in the pre-trained word embeddings it means that the sentiment they carry is not being adapted for learning by the network. This is certainly an aspect of the project which can and should be explored further.

An extension of the literature review and model training to include Attention models would be another interesting process, an example being Google's Attention is all you need approach [11] which achieves excellent results on NLP tasks based solely on the use of attention mechanisms. This project also does not explore Gated Recurrent networks which have shown promising results in the field.

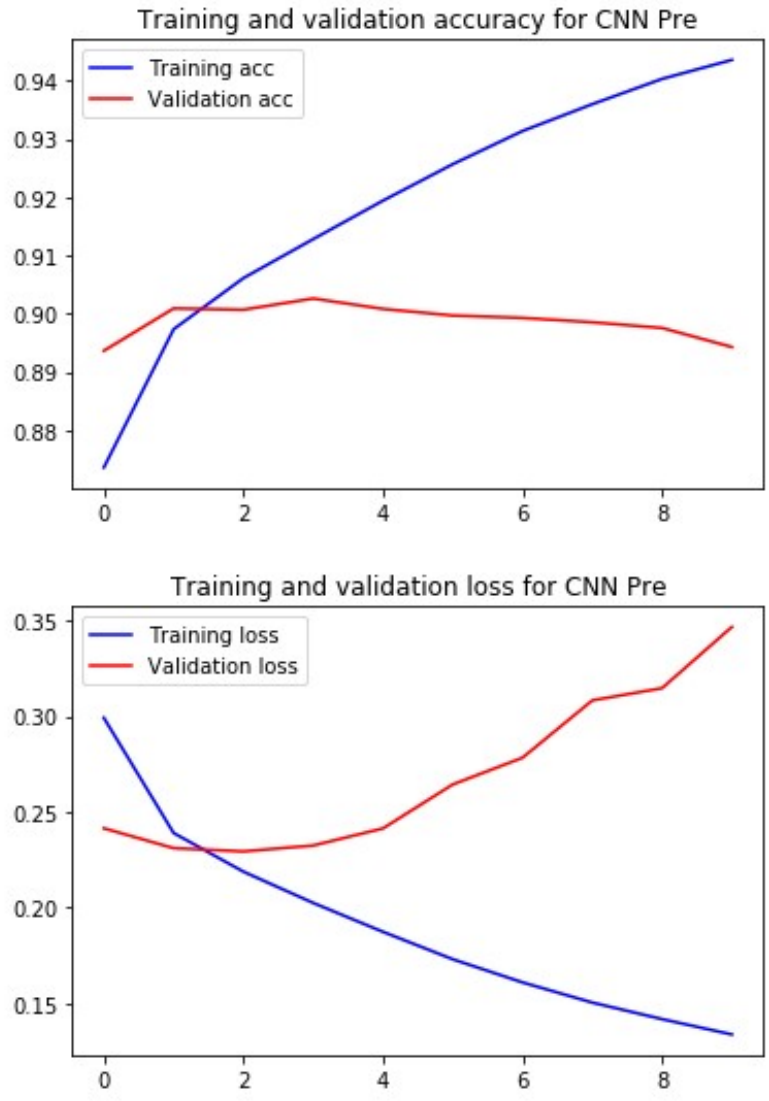

Figure 4: Accuracy and Loss for the CNN model trained on custom data

6 Conclusion

In this paper we have presented a number of approaches to CNN and LSTM networks in order to achieve high performances on sentiment analysis tasks. Because our dataset is not commonly used in literature results it is hard to draw definitive conclusions as to how the models in this paper compare. However, I think it is quite clear to see that there are some really excellent results. achieving 90% accuracy in NLP tasks is not easily achieved . We have performed a variety of hyper-parameter tunings in order to find the best optimisers and dropout rates and more often than not have come to same conclusions found in the papers by which we base our model architectures from.

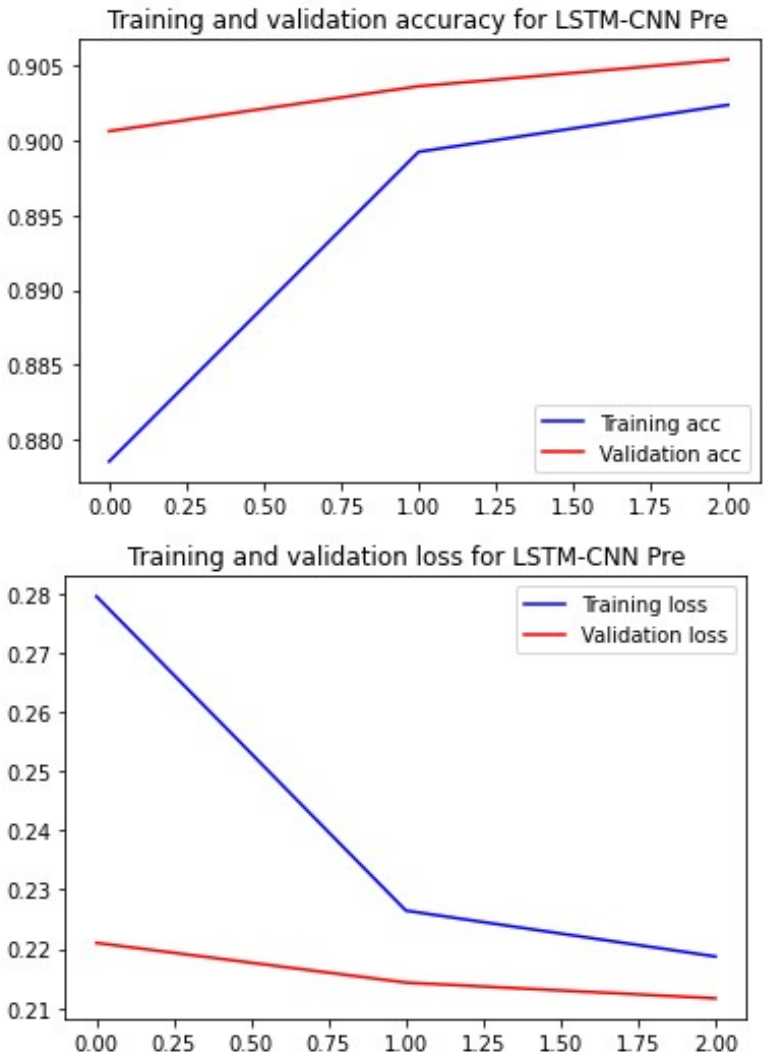

Figure 5: Accuracy and Loss for the final LSTMCNN model trained on custom data over 3 epochs

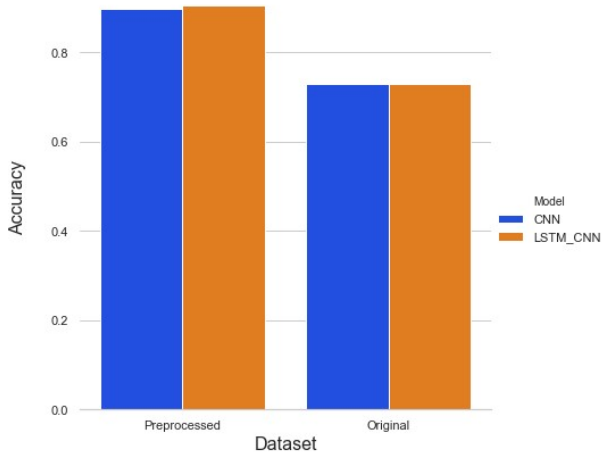

Figure 6: Summary pf accuracy achieved across the 4 models

References

[1] Kim, Y. (2014). Convolutional neural networks for sentence classification. *EMNLP 2014* , 1746–1751.

[2] Sosa, P. M. (2017). Twitter Sentiment Analysis using combined LSTM-CNN Models. *Academia.Edu, 1–9.*

[3] Sarlan, A., Nadam, C., & Basri, S. (2015). Twitter sentiment analysis. *Proceedings - 6th International Conference on Information Technology and Multimedia* 212–216.

[4] Turing, Alan (1950). Computing Machinery and Intelligence. *Mind*

[5] Bouazizi, M., & Ohtsuki, T. (2016). Sentiment analysis: From binary to multi-class classification: A pattern-based approach for multi-class sentiment analysis in Twitter. *IEEE International Conference on Communications*, 20617–20639.

[6] Hochreiter, S., Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780

[7] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations* - Workshop Track Proceedings, 1–12.

[8] Pennington, J., Socher, R. & Manning, Christopher. (2014). Glove: Global Vectors for Word Representation. *EMNLP*. 14. 1532-1543.

[9] Yadav, A., & Vishwakarma, D. K. (2020). Sentiment analysis using deep learning architectures: a review. *Artificial Intelligence Review*, 53(6), 4335–4385

[10] Kamiş, S., & Goularas, D. (2019). Evaluation of Deep Learning Techniques in Sentiment Analysis from Twitter Data. *Proceedings - 2019 International Conference on Deep Learning and Machine Learning in Emerging Applications, Deep-ML*, 12–17

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5999–6009.