DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF WESTERN ONTARIO

Kaggle Competition: Implement NLP to Classify Disaster Tweets

Shaoshi Zhang: szha88@uwo.ca
Kun Wang: kwang645@uwo.ca

Computer Science CS9860A: Advanced Machine Learning
Professor Charles Ling

Introduction

Twitter is one of the most widely used social media. It also has become an important communication channel in times of emergency. In this project (Kaggle competition), we are challenged to build a Machine Learning model that predicts which tweets are about real disasters and which ones are not. We have access to a dataset of 10,000 tweets that were hand classified.

The objective of this project is to figure out whether CNN models with transfer learning methods can perform well in the traditional NLP task and how well they could be compared to the "state-of-art" BERT Models.

Exploratory Data Analysis

First, we checked the distribution of these two classes (real & fake) as shown in figure 1. From the plot, we can see that the training set is nearly balanced. Since this dataset is not too large, therefore, we did not apply any dataset balancing technique to this training set in order to avoid overfitting.

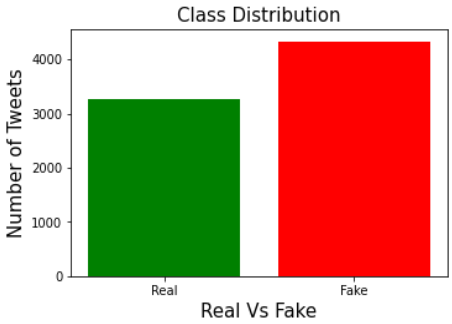

Figure 1: Dataset Distribution

Then we checked the text length in both classes. From figure 2 we can see that the distributions of text length of tweets (real & fake) have a similar range (approximately 0-150) and a similar left-skewed distribution.

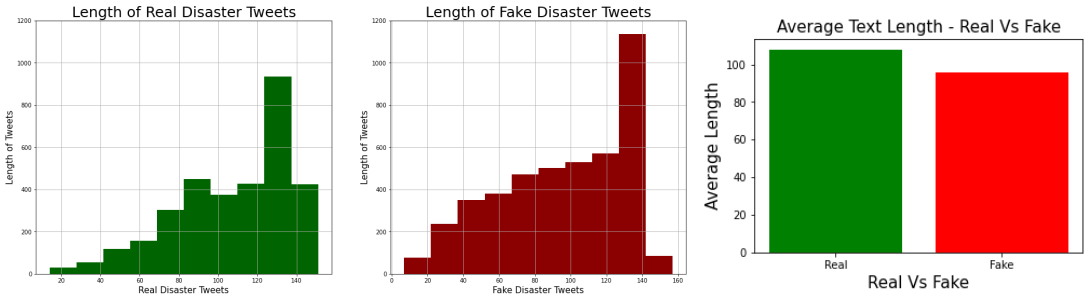

Figure 2: Comparison of Real and Fake in Length

We also looked at the average text length for both real and fake classes. From the plot above we can see that the average lengths are almost the same for both classes. Therefore, we can safely say that the text length is not important when determining whether the tweet is real or fake.

Stopwords & Punctuations

Stopword usually refers to the most common words in a language. For example 'a', 'the', 'in' etc. These words are essential parts of any language but do not play any significant role in the meaning of a sentence.

Punctuation marks are marks such as a full stop, comma, or question mark, used in writing to separate sentences and their elements and to clarify meaning. (!"#$%&'()*+, -./:;<=>?@[]^_`{|}~).

First, we checked the frequency of stopwords that appear in the whole dataset (train + test). We made a list of all stopwords and converted this list to a frequency dictionary to make frequency bar plots as shown in figure 3.

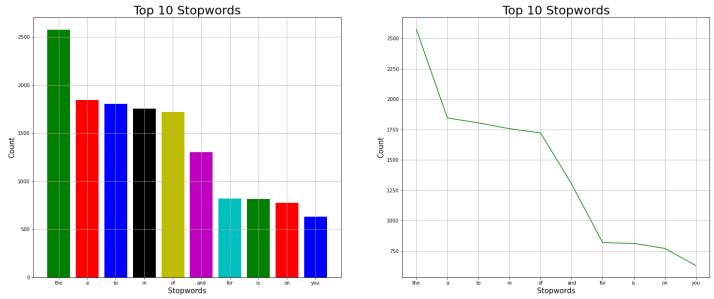

Figure 3: Top 10 stopwords in the dataset

From the figures above, we plotted the most frequent 10 stopwords with their appearing frequency. We can see that 9 out of 10 most frequent stopwords are preposition words, which do not have too much meaning.

Punctuations in the whole dataset (train & test)

Similarly to what we did above, we recorded the frequencies of punctuation marks in the whole dataset and visualize them in a barplot as shown in figure 4.

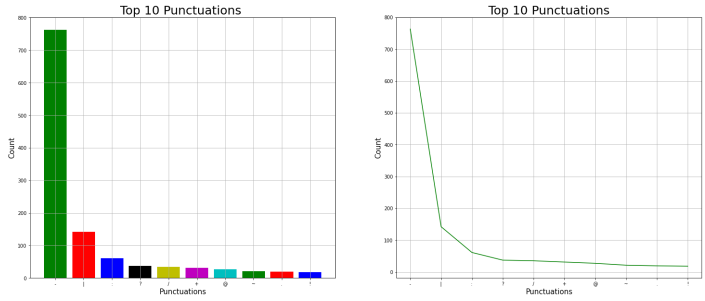

Figure 4: Top 10 punctuations in the dataset

From the figures above, we also plotted the top 10 popular punctuation marks with their frequencies. We can see that the frequency differs a lot, and the most popular one is '-'(dash) which appears about 600 more times than the second most popular punctuation mark.

Stopwords & Punctuations in real v.s. fake tweets separately

In this section, we will look at the stopwords and punctuation marks in real tweets and fake tweets separately so that we can figure out if the stopwords and punctuation marks have a significant influence when making the classifications.

We applied the same method as we did in previous parts to plot the most popular stopwords and punctuations and plot them horizontally so that we can compare the outcome from both classes together.

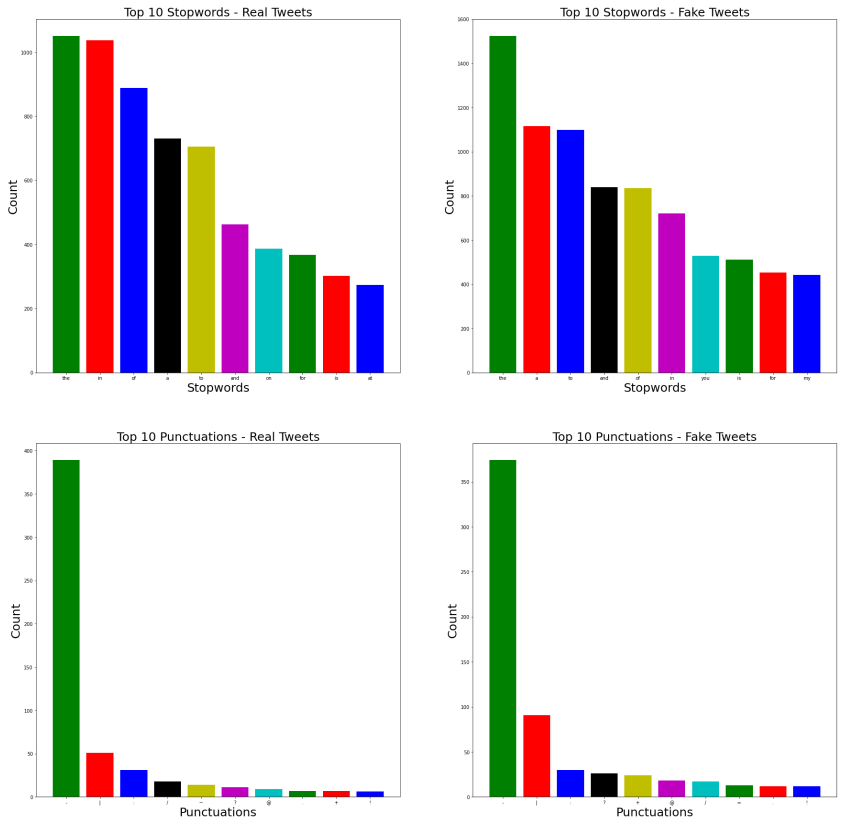

Figure 5: Comparison of Real and Fake in the punctuation and stopword

Data Cleaning

We did some data-cleaning procedures in order to get more accurate results from the Machine Learning models. The first step we made is removing all hyperlinks & URLs since these links have no meaning themselves. And then we remove all embedded special characters in tweets such as hashtags like #earthquake. Next, we remove all numerical characters in tweets and this step will delete all numbers in the tweets because we think that numbers themselves have no significant influence when classifying if the tweet is real or not. Also, we remove all user IDs that appeared in tweets since it is similar to the hashtags. Then we convert all letters to lowercase so that words like "Go" & "go" can be treated as the same word but not different words. The second last step in data cleaning is to remove stopwords and punctuations which are usually not important in understanding the meaning of the whole tweet, and convert words to their lemmas(for example, the lemma of the word "flying" would be "fly") to improve the predictive power of our model. Finally, we remove all emojis at the end of the data-cleaning procedures.

Vocabulary creation & N-gram Analysis

After we cleaned the whole dataset, we want to know if the content of real tweets will be different compared to the content of fake tweets, and if so, how it differs. Therefore, before we build any machine learning models, we create the vocabulary for this dataset and conduct the N-gram Analysis to have a better overall picture of our dataset.

Vocabulary

First, we created a vocabulary to include all unique words. There are 16442 distinct words in the training set(after preprocessing).

Then, we want to see how many words have appeared more than once in the corpus so that we could know the proportion of "rare" words. After we create another vocabulary to include the words with more than 1 occurrence, we can see that there are 6025 words appearing more than once in the training set, and the remaining ~10000 words appear only once.

N-gram Analysis

Now we can move to N-gram Analysis. An n-gram is basically a collection of n successive items in a text document. For example, I want to let the machine determine whether "I love eating burgers" is a real disaster or not. Unigrams(1-gram) consider words one by one so the machine will get "I", "love", "eating" and "burgers". Bigrams(2-gram) will consider phrases with 2 words so the machine will get "I love", "love eating" and "eating burgers". Trigrams(3-gram) will similarly consider phrases with 3 words such as "I love eating", and "love eating burgers".

So, in this way, conducting and comparing the n-gram analysis from these 2 classes(real & fake) will give us a better understanding of how contents differ between these two classes. We did unigrams, bigrams, and trigrams to visualize the most popular terms. We found that the most popular terms are quite similar for unigrams and bigrams. However, for trigrams, the most popular terms are starting to diverge from unigrams/bigrams. For detailed visualizations, please refer to the python notebook.

Word Cloud

Based on the N-gram analysis we had above, we plotted a word cloud for both real and fake tweets as shown in figure 6. The left world cloud image gives a good picture of the most common words used in real disaster tweets. Words like "news" "suicide" "bomber" "storm" "bomb" and "kill" are used heavily in the real disaster tweets which makes complete sense. The right image gives a good picture of the most

common words used in fake disaster tweets, we can clearly notice words like "love" "know" "good" etc. are not really likely to be included in real disaster tweets. So, we can see that the contents are a little bit different which is a good signal for classification tasks that it is easier for the machine to distinguish the difference.

Figure 6: Real tweets words(left), fake tweets words(right)

Model Building & Evaluation

In this section, we will define a few functions that will be used to build and evaluate our models. We define a model to automatically save the best model in the training epoch with the given model name.

We also defined a few functions to build various CNN models (1 convolution layer or 3 convolution layers) with pre-trained text embedding layers. Each function takes only one argument to specify the kernel size of the convolution layer which is "n-gram".

A standard 1-layer CNN model defined here for document classification is to use an Embedding layer(Universal Sentence Encoder/Wiki-words-500) as input, followed by a one-dimensional convolutional layer, a pooling layer, a dropout layer, a flatten layer, 1 dense layer, and then a prediction output layer. The kernel size in the convolutional layer defines the number of words to consider(n-grams) as the convolution is passed across the input text document, providing a grouping parameter.

A standard 3-layer CNN model is very similar to a 1-layer CNN model and the only difference is that there are 2 more groups of one-dimensional convolutional layers and pooling layers. The kernel size for all these 3 groups of convolutional layers is the same.

Then, the output of these 3 groups of convolutional layers will be flattened and then sent into the dense layer and the final prediction layer.

In each of the following sections (1-layer CNN and 3-layer CNN), we compared the performance of models with 4 different kernel sizes (16, 8, 4, 2) representing 4 different n-gram(16-gram, 8-gram, 4-gram, and bigram). And we will compare the performance based on the accuracy and the loss on both the training set and validation set.

1-layer CNN with Universal sentence encoder(USE)

In this section, we build 4 1-layer CNN models with a pre-trained universal sentence encoder as the first embedding layer. For these 4 models, they take 16, 8, 4, 2 as the kernel size(n-grams) which means these models will consider phrases that are made by 16, 8, 4, and 2 words respectively.

We can see that among these 4 models, the 1-layer CNN model with USE and kernel size of 8 has the best performance since it has the highest stabilized validation accuracy at around 0.81 to 0.82 and a validation loss of less than 0.45. Even though the same model with a kernel size of 4 has really similar performance, the kernel size of 8 still has a slightly higher accuracy score and lower stabilized loss.

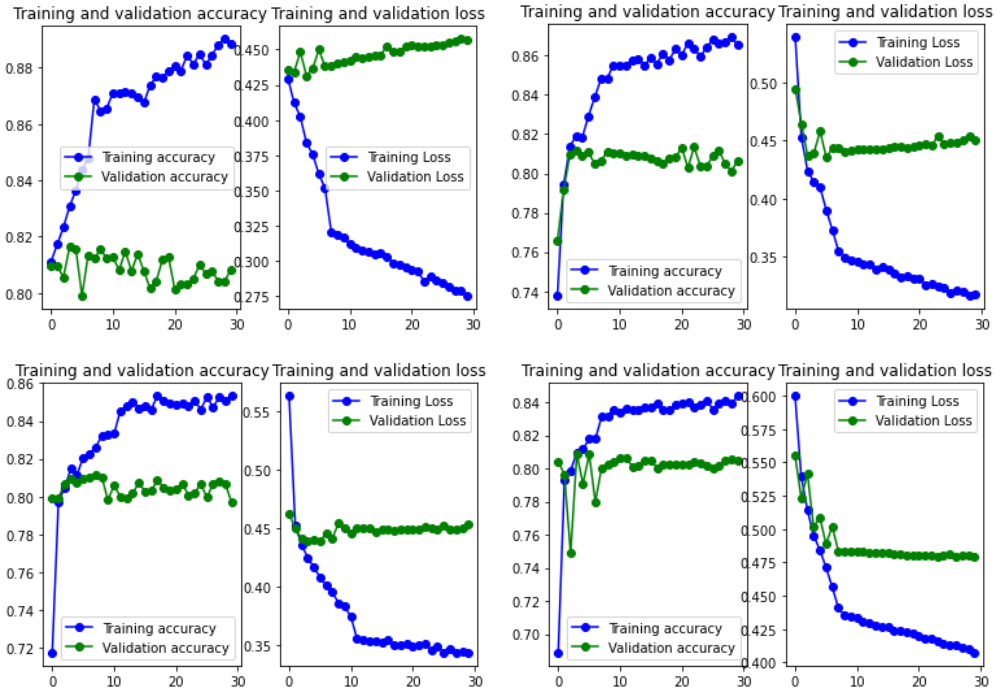

Figure 7: 1-layer CNN with USE and kernel sizes of 16(top left), 8(top right), 4(bottom left), 2(bottom right)

1-layer CNN with Wiki-words-500(wiki)

Similar to the previous section, here, we build 4 1-layer CNN models with pre-trained Wiki-words-500 as the first embedding layer.

We can see that among these 4 models, the 1-layer CNN model with a wiki layer and a kernel size of 8 has the best performance since it has the highest stabilized validation accuracy at around 0.80 and a validation loss of less than 0.45. Even though the same model with a kernel size of 4 has really similar validation loss, the kernel size of 8 still has a more stable accuracy score overall.

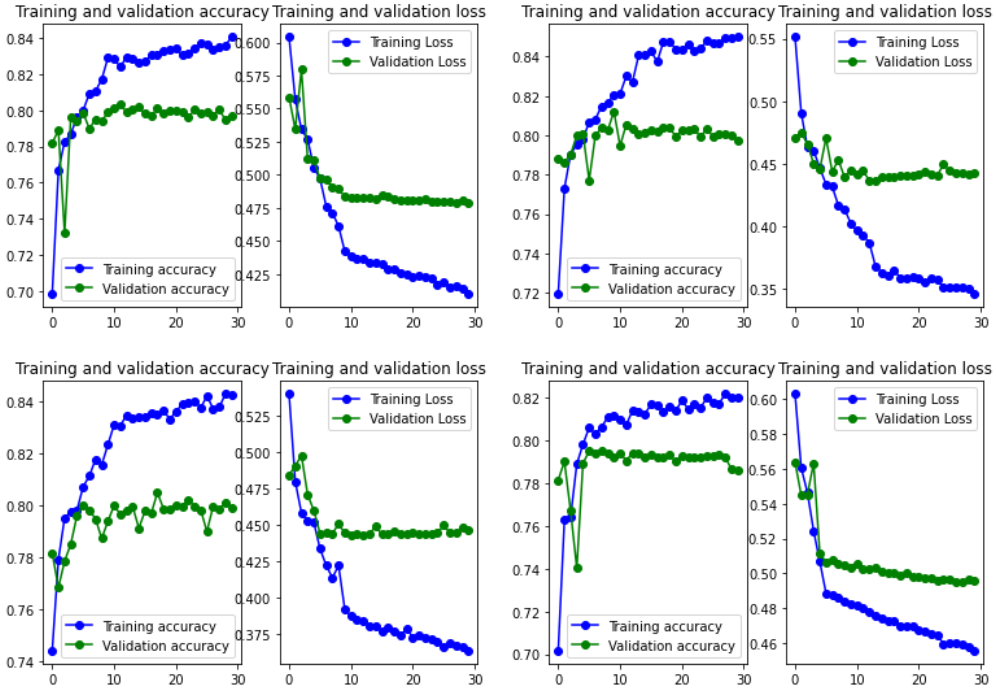

Figure 8: 1-layer CNN with wiki layer and kernel sizes of 16(top left), 8(top right), 4(bottom left), and 2(bottom right)

We also conducted hyperparameter tuning for 1-layer CNN with wiki layer to find the best number of filter, kernel size, pool size and the drop out rate. The process is considerably time-consuming and the model with the best parameters still has a similar performance compared to the untuned model. Therefore, we did not fine-tune each model in later sections.

3-layer CNN with Universal sentence encoder(USE)

In this section, we build 4 3-layer CNN models with a pre-trained USE layer as the first embedding layer. The only difference between 1-layer CNN with USE and

3-layer CNN with USE is that there are 2 more groups of convolution layers in the 3-layer CNN model.

We can see that among these 4 models, the 3-layer CNN model with USE layer and a kernel size of 16 has the best performance since it has the highest stabilized validation accuracy at around 0.80 and the validation loss at around 0.45.

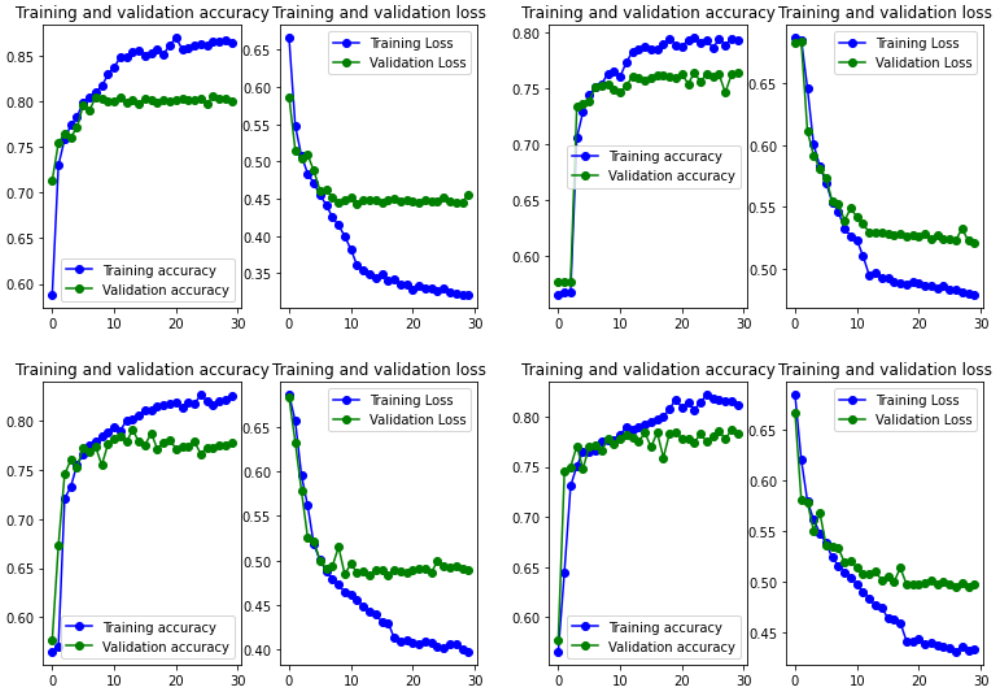

Figure 9: 3-layer CNN with USE layer and kernel sizes of 16(top left), 8(top right), 4(bottom left), and 2(bottom right)

3-layer CNN with Wiki-words-500

Similar to the previous section, we build 4 3-layer CNN models with pre-trained wiki layer as the first embedding layer here.

We can see that among these 4 models, the 3-layer CNN model with USE layer and a kernel size of 16 has the best performance since it has the highest stabilized validation accuracy at around 0.8 and the validation loss at around 0.45. Even though the same model with a kernel size of 4 has really similar validation accuracy and loss, the kernel size of 16 still has more stable validation accuracy and validation loss.

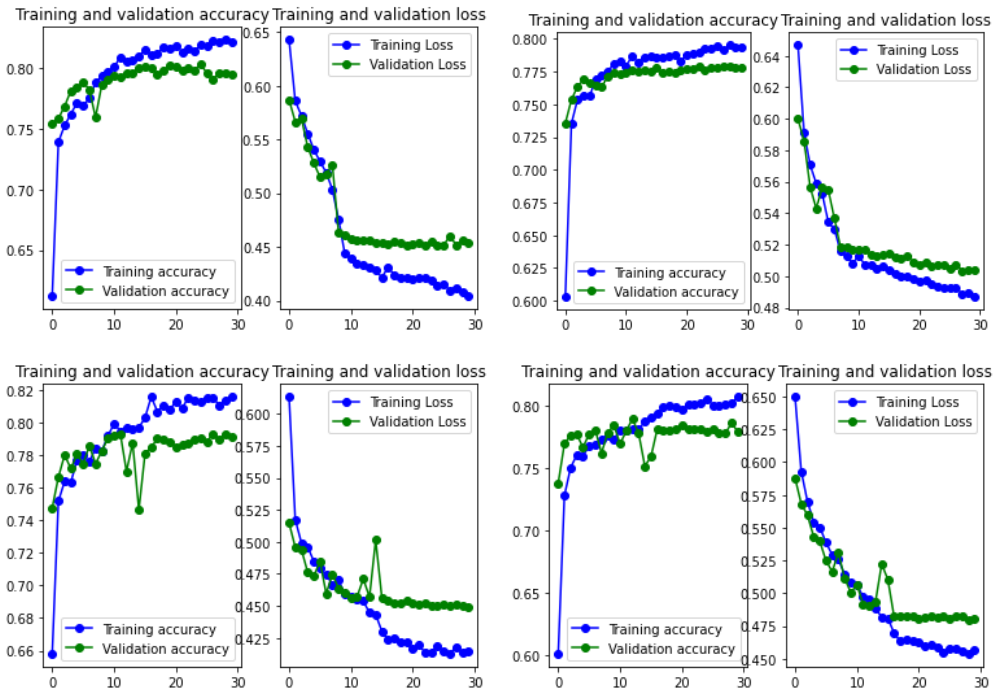

Figure 10: 3-layer CNN with USE layer and kernel sizes of 16(top left), 8(top right), 4(bottom left), 2(bottom right)

3-layer CNN with Wiki-words-500 various kernel size

Instead of fixing the kernel size and the size of n-gram sent to the model, we also want to test if taking various n-grams in considering at the same time will improve the overall performance or not. So in this section, we built only one 3-layer CNN model with the wiki layer as the embedding layer and the kernel sizes for those 3 groups of convolutional layers are 16, 8, and 4. In this way, we can take 16-gram, 8-gram, and 4-gram as inputs at the same time.

As we can see in figure 11, the model is stabilized with a validation accuracy of around 0.8, and the validation loos around 0.45, which is really close to the performance of previous models. Therefore, we can safely say that taking various n-gram into consideration at the same time will not improve the overall model performance significantly.

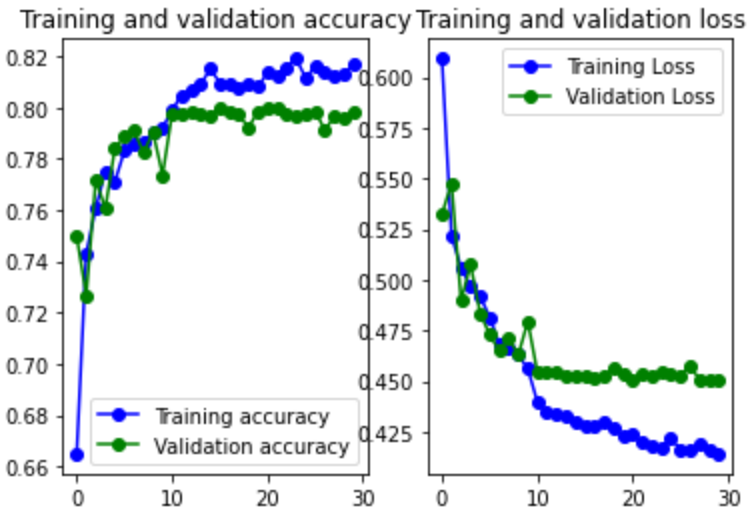

Figure 11: 3-layer CNN with wiki layer and various kernel sizes

Multi-Channel CNN Model

A multi-channel convolutional neural network in this project involves using both the USE layer and the wiki layer. This allows the document to be processed at different embedding vectors at a time, whilst the model learns how to best integrate these interpretations. The output from the two 1-layer CNN channels is concatenated into a single vector and processed by a Dense layer and an output layer. The kernel size is 16 for both channels.

We can see that the multi-channel CNN model did not stabilize within 10 epochs as other models did. And also, we can see from the constantly increasing validation loss and the decreasing validation accuracy that this model is suffering typical overfitting on the training set. Therefore, we can conclude that this Multi-Channel CNN model is not suitable for this NLP classification challenge.

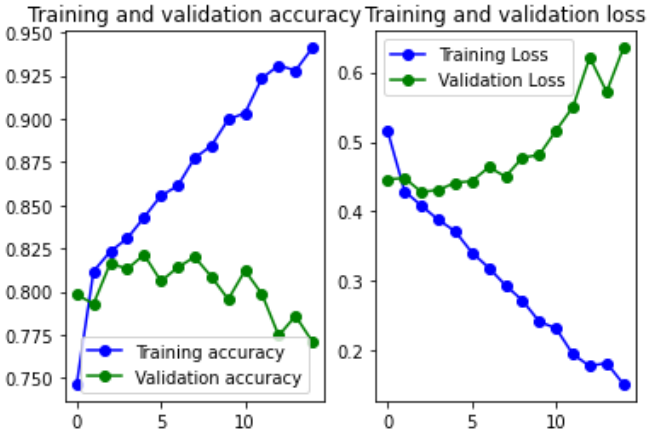

Figure 12: Multi-Channel CNN Model Performance

BERT

Concept

BERT is a deep learning model. It has given state-of-the-art results on a variety of NLP tasks. It stands for Bidirectional Encoder Representations from Transformers. It is based on the Transformer architecture and has been pre-trained on Wikipedia(2,500 million words) and Books Corpus(800 million words). BERT is a deeply bidirectional model. This means that BERT learns information from both the left and the right sides of a token's context. Many people believe that BERT has significantly altered the NLP landscape. Therefore, we want to use this Kaggle competition as an opportunity to learn and practice applying BERT to an NLP task.

Preprocessing Text for BERT

Before we feed our dataset to the BERT model, we need to preprocess the text in order to make the BERT understand them. There are totally three embeddings that are used in BERT as shown in figure 13.

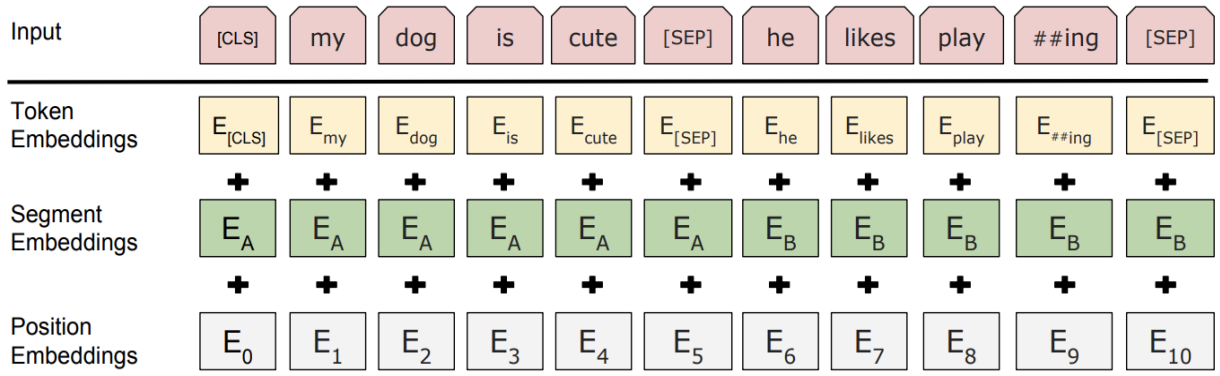

Figure 13: Preprocessing for BERT

Token Embeddings: For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings.
Position Embeddings: BERT learns and uses positional embeddings to express the position of words in a sentence.
Segment Embeddings: BERT can also take sentence pairs as inputs for tasks.

Pre-training for BERT

The model was trained in two tasks simultaneously: Masked Language Model(MLM) and Next Sentence Prediction. The Masked Language Mode is essentially

filling in the blank. It simply masks some percentage(15%) of the input tokens at random and then predicts those masked tokens.

Next Sentence Prediction is used to train BERT to predict sentences. For example, in sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus.

Fine-tuning

Fine-tuning is straightforward since the self-attention mechanism in the Transformer allows BERT to model many downstream tasks by swapping out the appropriate inputs and outputs. It simply plugs in the task-specific inputs and outputs into BERT and finetunes all the parameters end-to-end.

Model Building and Results

We had three layers in the BERT model. They are word id, mask, and segment id layer. We then put these three layers into the BERT. The results are shown in figure 14. We used both cleaned data and raw data. The BERT model performs better with the raw data.

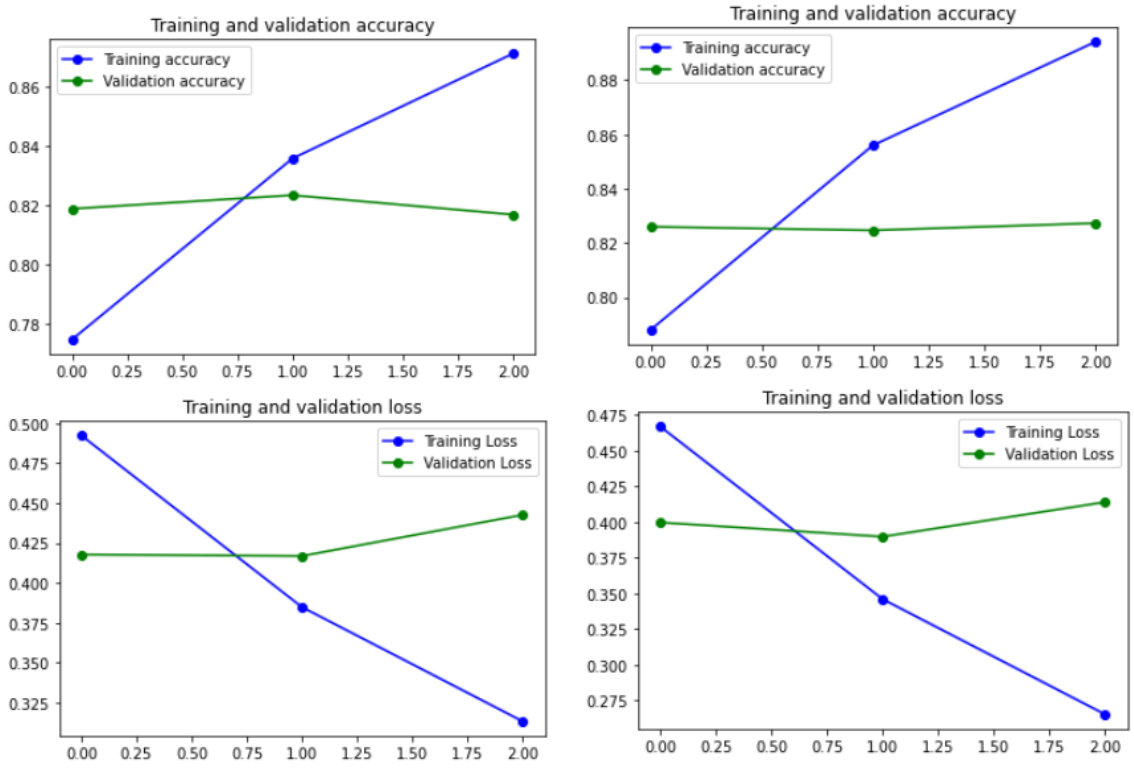

Figure 14: Result using cleaned data(left) and raw data(right)

Conclusion

From the results and figures above, we can conclude that CNN models with pre-trained transfer learning embedding models can perform well in the typical NLP classification task and maintain a similar model performance compared to the "state-of-art" BERT Models.

1-layer CNN model with either a pre-trained Universal Sentence Encoder or Wiki-words-500, and appropriate kernel size could achieve similar performance to BERT did but took only about 1/4 of BERT's training time, which is much easier to train and computationally effective.

Also, adding more convolutional layers to the CNN model did not simply improve the model performance, but made the training time longer to about 1/2 of BERT's training time.

The multi-channel CNN model took much longer to train and its training time is quite similar to BERT's training time. However, the multi-channel CNN is not stabilized at all given the training epochs.

Therefore, in simple NLP classification problems, rather than using BERT, we can also consider using CNN with a transfer learning method to achieve similar performance and reduce the training time and computational requirement.

Detailed model performances can be found in the table below.

	USE(1)	USE(3)	wiki(1)	wiki(3)	multi-cnn	BERT
Ave Val Loss	~0.45	~0.45	~0.45	~0.45	~0.5	~0.41
Ave Val Acc	~0.81	~0.8	~0.8	~0.8	~0.8	~0.825
Kaggle	0.79742	0.79742	0.80355	0.79742	0.80324	0.82