Character Level CNN for Sentiment Analysis

Melisa Panaccione
40182167
Concordia University
Montreal, Canada

Lucas Miquet-Westphal
40215325
Concordia University
Montreal, Canada

Arpita Noul
40178343
Concordia University
Montreal, Canada

Abstract

In this paper, we create a Convolutional Neural Network (CNN) that takes characters as input and outputs a sentiment: positive, negative, or neutral. The goal of the experiment is to see if neural networks can produce complex meaning from text without any word statistics, text tokenization, or word dictionaries. The CNN model is reproduced from the paper "Text understanding from scratch" by X. Zhang et al. [1] It is a 9 layer CNN with six convolution/pooling layers and three fully connected layers. We train the model on 25,000 Twitter posts and achieve an accuracy of 55.57%. Since the original paper is from 2015, we train a state-of-the-art transformer to compare to our CNN model, and we achieve an accuracy of 81%. We conclude that the CNN model has learned to extract complex textual meaning from characters, but transformer models have significantly surpassed their performance.

1 Introduction

Traditionally, developers had to do significant work prior to performing NLP (Natural Language Processing) tasks. For example, for Naive Bayes's methods, word statistics are needed for text classification, and for most neural network applications, word embedding have to be created prior to training the network. In this paper, we explore the possibility of a model that can directly take the text input as characters, and find word, sentence and text patterns on its own.

This experiment is inspired by the paper "Text Understanding from Scratch" by X. Zhang et al. [1] In their paper, they create a CNN model that takes characters as an input, and then it outputs complex textual meaning. Their model was inspired by CNN's high performance in Computer Vision, where they take raw pixels from images, and can extract complex image features. Through their convolution and pooling layers, CNNs can gradually extract lines, shapes, textures, etc. Inspired by Computer Vision's pipeline from raw pixels to meaning, the character-level CNN model aims to take raw characters and extract complex features, like word structure and sentence structure, then eventually text meaning. This model was chosen because it presents an interesting and unique take on NLP, and can potentially be used as a building block for other models.

The model from X. Zhang's paper [1] is from 2015 and quite outdated. Since AI has been advancing quite rapidly in the past few years, there are many newer and more reliable models. In order to further evaluate the usefulness of our model, we compare it to a state-of-the-art model: the transformer. We use the BERT-base-uncased model [3] and the RoBERTa-base model [5] to implement this, using words as tokens.

2 Related Work

Our work is mainly related to the work of X. Zhang et al.'s [1] efforts on creating a CNN character-level model. They built the 9-layer CNN model that this paper is based on. They used their model for several different applications, the most similar one being an Amazon review sentiment analysis classifier. The Amazon review dataset contained 3 million samples and achieved an accuracy of 94.33%. They used their model for several different applications, and many also performed very strongly. For example they used it for news category classification in both English and Chinese, and they achieved an accuracy or 85.18% and 95.12%, respectively.

The model we build in this paper is a similar application to the Amazon sentiment analysis model, but ours uses a smaller data set. Since they had 3 million samples, X. Zhang et al.'s paper took several days to train each epoch. For this project, we don't have the time or the hardware to train a model that takes several days per epoch, so we use a smaller data set.

J. Devlon et al.'s work on the BERT (Bidirectional Encoder Representations from Transformers) model [7] serves as a baseline to measure our model's performance compared to state-or-the-art technology. BERT is a powerful context-aware model. It is pre-trained and can be easily fine-tuned to many different types of tasks, including text classification. It is based on Vaswani et al.'s original Transformer model [8], which is based on attention mechanisms. The transformer model consists of an encoder and decoder, which

use multi-head attention layers. This allows the model to produce results that are textually context-aware. The Transformer model outperforms both Recurrent Networks and CNN models, while also being faster to train.

3 Model

3.1 Dataset

To train our model, we used a dataset of English Twitter posts [2], choosing 25,856 samples for training and 6,464 samples for testing. The dataset has several columns, but the only ones we are interested in are "text", which contains the Tweet message, and "sentiment" which contains the sentiment class: positive, negative or neutral. Since the samples are from social networks, the texts are in informal language. The data set was initially built using emoticons to automatically classify Tweets. They classified happy emoticons (ex: ":)") as positive and sad emoticons (ex" ":(") as negative. Then, they extract text (without emoticons) into a dataset. The data is in a .csv format, and was transformed into a Pandas Dataframe [6] for our purposes.

3.2 CNN Architecture

In this paper, we build the smaller CNN model described in X. Zhang et al's study [1]. This is a 9 layer CNN. The first two layers have a convolution and pooling. It is then followed by three convolution layers, then one more layer with both a convolution and pooling. Finally, it ends with 3 fully connected layers and a 3-dimentional output. All convolutions have a frame dimension of size 256 and all pooling layers have a window size of 3. The model design can be better analyzed in Figures 2 and 3. There are also dropout modules with a probability of 0.5 between each fully-connected layer, which helps prevent overfitting. The overall structure of the model can be seen in Figure 4.

3.3 Components Not Implemented

We discussed the components from the "Text Understanding From Scratch" paper [1] that we implemented in our paper. Now we will discuss the components of their model that we did not implement into ours.

The original paper has a bigger model and a smaller model, with frame sizes of 1024 and 256 respectively. Due to hardware and time limitations, we wanted to keep our project compact and simple, so we only implemented the smaller model. We did not see a reason to use the larger model, because the performance gap between the larger and smaller model in the original paper's sentiment analysis task was not very significant. We concluded that the small dataset would be sufficient and more practical for our project.

Additionally, we did not implement the same size character dimensions as the original paper. While they used a dimension of 70, we used a dimension of 40. Compared to the original model, we left out some punctuation that we didn't feel was necessary. We made this change to simplify our solution, and to make our model more practical to execute with limited hardware.

The original paper augmented the data using a Thesaurus. They replaced words randomly with synonyms to improve data generalization. This was inspired by similar processes in computer vision and speech recognition. In computer vision tasks, a similar process is done by implementing changes like translations and rotation. In speech recognition, it can be done by adding background noise. The paper performs all their experiments both with and without Thesaurus augmentation. We did not implement this regularization in our own paper. Although it improved their results a bit, we wanted to keep our own experiment simple and easy to analyze. We left this out and chose to focus more on the general CNN algorithm.

The original paper also used their model for many different applications–Amazon sentiment analysis, Oncology classification, Yahoo! review classification, and news classification in both English and Chinese. We decided to choose one, and focus on the model algorithm rather than comparing different applications. Instead of comparing multiple different classification tasks with one model, we compared one classification task with two models (CNN vs. Transformer).

3.4 Transformer Baseline with Pre-trained Encoders

As a baseline comparison with modern transformers, we implemented a modern encoder-based baseline to compare against our character-level CNN. Instead of learning directly from characters, this approach uses pre-trained transformer encoders that operate on subword tokens. We rely on models from the Hugging Face model hub, in particular BERT-base-uncased [3] and RoBERTa-base [4].

For both models, the input is the raw tweet text. The corresponding tokenizer converts each tweet into a sequence of subword tokens, which are then fed into the encoder. On top of the encoder, we attach a small linear classification head that maps the final representation to the three sentiment labels (negative, neutral, positive). The models are fine-tuned end-to-end using cross-entropy loss on the same training/validation split as our CNN, so that the comparison is fair.

	textID	text	selected_text	sentiment	Time of Tweet	Age of User	Country	Population -2020	Land Area (Km²)	Density (P/Km²)
0	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral	morning	0-20	Afghanistan	38928346.0	652860.0	60.0
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative	noon	21-30	Albania	2877797.0	27400.0	105.0
2	088c60f138	my boss is bullying me...	bullying me	negative	night	31-45	Algeria	43851044.0	2381740.0	18.0
3	9642c003ef	what interview! leave me alone	leave me alone	negative	morning	46-60	Andorra	77265.0	470.0	164.0
4	358bd9e861	Sons of ****, why couldn`t they put them on t...	Sons of ****,	negative	noon	60-70	Angola	32866272.0	1246700.0	26.0

Figure 1. Dataset from Kaggle

Layer	Frame Dimentions	Kernel	Pool
1	256	7	3
2	256	7	3
3	256	3	N/A
4	256	3	N/A
5	256	3	N/A
6	256	3	3

Figure 2. Dimensions of the convolution and pooling layers of our model

Layer	Output Units
7	1024
8	1024
9	3

Figure 3. Dimensions of the fully connected layers of our model

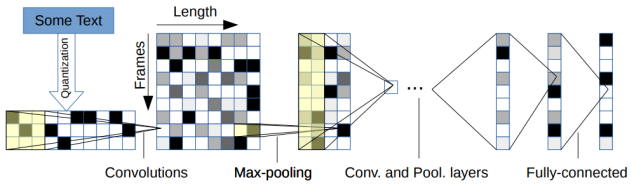

Figure 4. Illustration of the model from the "Text Understanding From Scratch" paper

4 Methodology

4.1 Data Setup

In order to build our CNN model, we created specifications for the architecture, which were mostly taken from the X. Zhang et al. research [1], except we updated the output layer size. The specifications of our model can be seen in Figures 2 and 3.

Once we had the specifications for our model, we used Google Colab [9] for development. We started by importing our data through the Kaggle Hub [2] into a Pandas [6] Dataframe.

For our CNN model, We only considered characters from the following string: "abcdefghijklmnopqrstuvwxyz .!? 1234567890", and turned them into one-hot-encodings. We also transformed the outputs (positive, negative and neutral) into one-hot-encodings. Next, we transformed each of our data samples' text columns into sequences of characters (as one-hot-encodings).

For the BERT models, we did not need to preprocess the text data oursleves. The data was processed within the model, which took the raw text and tokenized it into words.

4.2 CNN Model

After the data was ready, we built the model using Pytorch, a deep learning framework [10]. We created a CNN module with 9 layers, set up the dropout between the connected layers, and proceeded to train the model. We trained the model on 25,856 samples, split into batches of 64. In order to prevent overfitting, we calculated the training loss at each epoch on the validation set. When the training loss increased, we stopped training. We ended up training 52 epochs.

Once the model finished training, we used our test set to achieve a list of target values vs. model predictions, and evaluated our performance. Figure 5 shows a sample of the testing output. We computed the accuracy and F1 score, as well as creating a confusion matrix (Figure 6). For comparison purposes, accuracy is the most important metric, since it is the only one discussed in our inspiration paper [1].

4.3 Transformer Baseline with Pre-trained Encoders

For the Transformer model, we imported Google's BERT model [7] and RoBERTa model [5] from Hugging Face. In order to fine-tune the model to our application, we trained both of them on our own data. Next, we ran our test data through the models in order to evaluate the accuracy, the macro-averaged F1 score and the confusion matrix.

During training, we monitor the macro-averaged F_1 score on the validation set and apply early stopping with a patience of five epochs: if the validation macro F_1 does not improve for five successive epochs, training is stopped and

```
----------
Text:  The one and the same! Should be a good gig
True label:      positive
Predicted label: positive
Class probabilities:
  negative: 0.005
  neutral : 0.020
  positive: 0.975
----------
Text:  say hello to Marina Green ... couldn`t participa
True label:      negative
Predicted label: negative
Class probabilities:
  negative: 0.504
  neutral : 0.441
  positive: 0.055
----------
Text:  - love your new avatar!
True label:      positive
Predicted label: positive
Class probabilities:
  negative: 0.002
  neutral : 0.006
  positive: 0.991
----------
Text:  Esp with Twitter access! the party NEVER stops -
True label:      neutral
Predicted label: negative
Class probabilities:
  negative: 0.452
  neutral : 0.420
  positive: 0.127
----------
```

Figure 5. Partial Screenshot of Test Output

the best checkpoint is kept. This procedure prevents overfitting while still allowing the transformers to adapt to the Twitter domain.

We also attempted to fine-tune a much larger encoder, RoBERTa-large [5], but our single-GPU setup did not have enough memory and the training time became prohibitive. For this reason, we only report quantitative results for the two base-sized models.

5 Results

5.1 CNN Performance

Our character-level CNN was trained on 25,856 tweets and evaluated on a held-out set of 6,460 tweets.

On our validation set, the model achieves an accuracy of 55.57% and an F1 score of 54.90%. This is well above random guessing for a three-class problem, which indicates that the network has indeed learned useful character-level patterns.

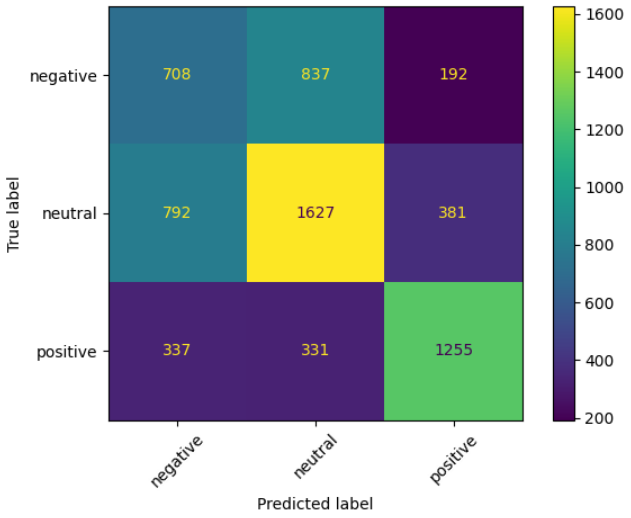

Figure 6. Confusion Matrix for our CNN Model

However, the accuracy is still relatively low for a practical sentiment analysis system, and the confusion matrix shows frequent confusions between neutral and negative sentiments, with 25% of samples being a misclassification between the two. The confusion matrix displaying this can be shown in Figure 6.

The Amazon sentiment analysis model by X. Zhang et al. achieved a 94.33% accuracy. It is higher than ours (55.57%), but it makes sense considering that our dataset was significantly smaller (25k vs 3M samples).

5.2 Transformer Encoders

The transformer baselines achieve substantially higher performance. When we fine-tune BERT-base-uncased [3] on the same training data, we obtain an accuracy of approximately 79% and a macro-averaged F_1 score of 0.79 on the validation set. RoBERTa-base [4] performs slightly better, reaching around 80–81% accuracy and a macro-averaged F_1 of 0.80. The confusion matrix for this model can be found in Figure 7.

The confusion matrices for these models show that both encoders classify the positive class particularly well, with RoBERTa-base achieving an F_1 score of about 0.83 for positive tweets. Compared to our CNN, the transformers also reduce the number of neutral tweets misclassified as negative or positive. Overall, the best transformer model improves accuracy by roughly 30 percentage points compared to the character-level CNN (55% vs. 81%), and improves macro F_1 by a similar margin.

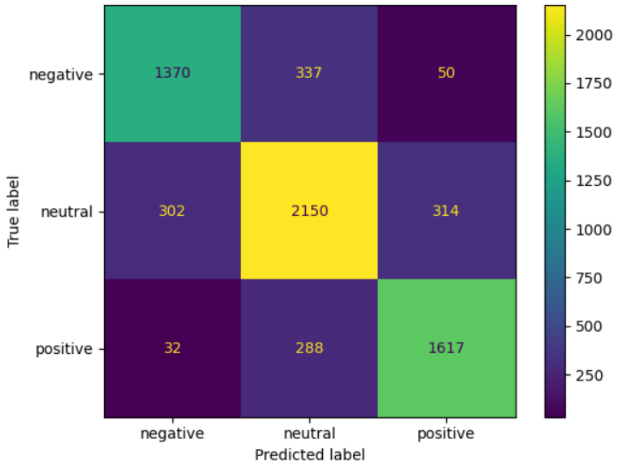

Figure 7. Confusion Matrix for the RoBERTa-base model

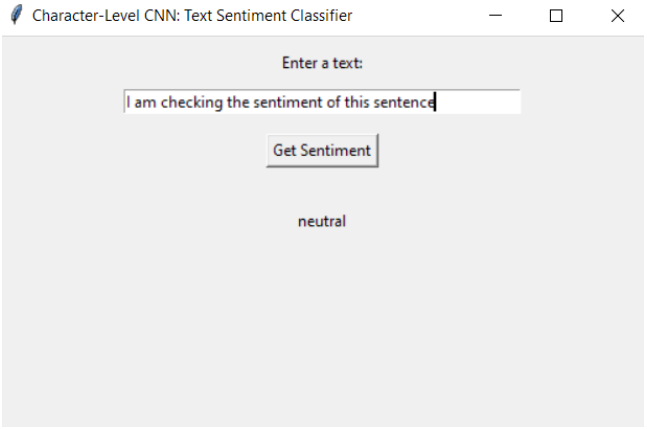

Figure 8. Executable Python application

We additionally attempted to fine-tune RoBERTa-large [5], but this model did not fit comfortably within our available GPU memory and training was too slow to complete. This limitation suggests that, in resource-constrained settings, smaller pre-trained encoders such as BERT-base or RoBERTa-base already offer a very strong and practical baseline.

5.3 Comparison

The accuracy of the CNN model vs. the RoBERTa model is 55% vs. 81%. The results confirm that our CNN can learn meaningful representations directly from characters, but they also highlight how far modern pre-trained transformer encoders have pushed the state of the art in sentiment analysis.

Although the CNN model can recognize patterns such as words and sentence structures, a Transformer model can also detect these patterns and more. Unlike the CNN model, the Transformer model uses multi-head attention and is aware of longer-term sentence context.

As shown in the original paper, CNNs can achieve a very high performance (95% accuracy), but it took them many days, or even weeks to train, and it used a 3 million sample dataset. That amount of time and data is not always available, and it is more practical to use a model that does not require excessive work. Our experiment shows that a pretrained transformer is much more practical than CNN models because, with limited time and hardware, it achieves significantly superior results.

6 User Manual

This project can be accessed through GitHub through the following link: https://github.com/melisa1026/Text-Understanding-From-Scratch---Sentiment-Analysis-Project/tree/main. This GitHub Repository contains:

1. An executable Python application
2. A Jupyter Notebook that creates, trains and evaluates the CNN model
3. A Jupyter Notebook that creates, trains and evaluates the Transformer model

6.1 Executable Python Application

The Python application allows you to classify texts using the CNN model. To run it:

1. Install Python 3 or later.
2. Clone the GitHub project onto your device.
3. Open the project folder and run 'main.py'. A GUI window will pop up.
4. In the GUI window, enter a text into the input space.
5. Click 'Get Sentiment'.

The GUI window can be seen in Figure 8.

6.2 Jupyter Notebooks

In the folder 'Model Training and Evaluation', there are two Jupyter Notebooks: one for the CNN model and one for the Transformer model. Each of them initializes, trains and evaluates the model. To run them:

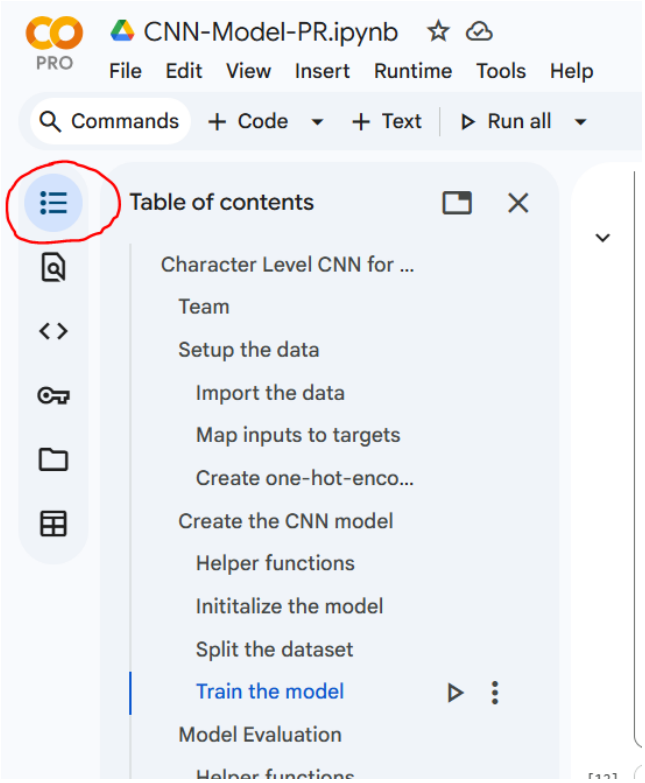

Figure 9. Google Colab Table of Contents

1. Open the folder 'Model Training and Evaluation' from GitHub: https://github.com/melisa1026/Text-Understanding-From-Scratch---Sentiment-Analysis-Project/tree/main.
2. Download the Jupyter Notebook file (.ipynb) for the model you would like to run.
3. Open the file on Google Colab (or another Jupyter Notebook editor).
4. Click 'Run All' (Figure 10).
5. This will import the data, create the model, train the model, test the model, and provide a Confusion Matrix.

On the left navigation bar on Google Colab, you can access a Table of Contents to easily navivate through the code (Figure 9).

7 Threats to Validity

The dataset we used in this experiment was automatically created. It classified Twitter sentiments using emoticons. For example, ":)" and ":(". Due to automatic creation, it is quite possible that a lot of samples are mislabeled. Twitter posts can be written by anyone, and the real sentiment can be unclear, vague, subjective or sarcastic. It is quite likely that many emoticons that were used to classify the texts were

Figure 10. Run All Button in Google Colab

not reliable. This problem could have negatively impacted our performance.

Additionally, our dataset consists of social media posts, which means most of the language is informal. Compared to more modern algorithms, the technologies of 2015 were not very powerful for detecting complex textual meaning, like sarcasm. We believe a more formal dataset would be more compatible with our CNN model.

Our research was also limited by hardware capabilities and time limitations. While the inspiration paper [1] used 3 million training samples for its sentiment analysis, we were only able to use 25 thousand. We did not have the GPU capabilities to load a 3 million point dataset, nor did we have the time to run each epoch for two days (which is how long it took in the inspiration paper). Still, with a smaller dataset, our results came quite close to the original paper. We were still able to keep running epochs until the training loss increased (just before overfitting), and we ran 52 epochs.

We also used less characters than the paper did. This was done because we didn't think these additional characters were relevant to sentiment analysis, and we believed they were just increasing dimensionality and computing space. We used 40 characters ("abcdefghijklmnopqrstuvwxyz. !? 1234567890"), while the original paper used 70. The additional ones they used were: "-,;.:"'/|_@#$%^&*~`+-=<>()[]". Although we do not think the additional characters would have a big impact on the sentiment of a text, there is a possibility that they would have made a difference in results. Maybe the punctuation could have helped the CNN better understand sentence structure. Or maybe some of them would contribute to the understanding of social media trends or language that we did not consider.

8 Reflections

8.1 What was Learned

Throughout this project, we learned more about the capabilities of classification models throughout the last 10 years. We learned that Neural Networks can learn sentiments from very little information–no word embeddings or word statistics necessary. But we also realize that, compared to modern algorithms (like transformers), our model is not the most powerful one. Although they can achieve high accuracy, they require a very large datasets and very long training to achieve successful results.

While researching the inspiration of the original paper, we also learned a lot about CNN architecture in general–particularly how convolution and pooling layers work together to extract features. While researching the inspiration of the original paper, we came to know a lot about Computer Vision. We found it very fascinating that raw pixels could be gradually transformed into more complex features, just through filters and pooling. And it was interesting that this pixel-based algorithm could be transferred to text characters so simply.

Even if the performance of our CNN in sentiment analysis did not show any groundbreaking success, especially compared to the transformer, it was still fascinating to learn how it works, and to see it successfully produce some textual meaning with so little context. Even though it is not the most powerful tool on its own, we believe that it shows promise to act as a building block to aid bigger tools.

8.2 Reproduction Challenges

We were able to reproduce a model with the same architecture as the original paper, but we were not able to reproduce a solution of the same magnitude. Our dataset was much smaller, and we used the smallest solution described in the paper. The original paper used a dataset with 3 million samples, which took them 2 days to train per epoch. Using that size of data would have been impossible for us, because we did not have the time and hardware to run our code for days at a time. Especially since we ran it a few times for debugging and experimentation.

8.3 Alternative Approaches

After performing the experiment, we noted some alternative routes we could have taken. First of all, we could have chosen a dataset with more formal language. We believe this would have been more compatible with our CNN model and achieved a better performance.

Additionally, we think it would have been interesting to reproduce the algorithm that was the state-of-the-art algorithm in 2015, and compare that to the state-of-the-art in 2025. In our current experiment, the model we used is not necessarily the best performing model of its time, so choosing a better model would have better displayed technology's advancements in the last 10 years.

9 Conclusion

Our CNN model achieved a 55.57% accuracy in sentiment analysis. Even if 55.57% is not very accurate guessing, it is higher than random guessing (33%) and therefore confirms that complex textual meaning can be learned from a character-level input.

Our Transformer model performed significantly better than our character-level CNN model (55% vs. 81%). This can be attributed to the transformer's context-awareness. It is quicker at detecting patterns, and perhaps, due to its attention mechanisms, catches onto more complex informal meaning more easily. The difference in performance between the CNN and Transformer really puts into perspective how far research has come in the last 10 years.

From the original paper, we know that the CNN model can perform very strongly (95% accuracy). But from our own experiment (same model, smaller scope, 55% accuracy), we learned that CNNs take a lot more time and work to achieve a high success rate, compared to the fine-tunable models that exist in modern day.

References

[1] X. Zhang and Y. LeCun, "Text Understanding from Scratch," arXiv:1502.01710 [cs], Apr. 2016, Available: https://arxiv.org/abs/1502.01710.

[2] M. KazAnova, "Sentiment140 dataset with 1.6 million tweets," www.kaggle.com, 2017. https://www.kaggle.com/datasets/kazanova/sentiment140.

[3] Hugging Face, "BERT base uncased," Hugging Face, 2024. [Online]. Available: https://huggingface.co/google-bert/bert-base-uncased.

[4] Hugging Face, "RoBERTa-base," Hugging Face, 2024. [Online]. Available: https://huggingface.co/FacebookAI/roberta-base.

[5] Hugging Face, "RoBERTa-large," Hugging Face, 2024. [Online]. Available: https://huggingface.co/FacebookAI/roberta-large.

[6] W. McKinney, "Data Structures for Statistical Computing in Python," Proceedings of the 9th Python in Science Conference, vol. 445, 2010, doi: https://doi.org/10.25080/majora-92bf1922-00a.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," May 2019. Available: https://arxiv.org/pdf/1810.04805.

[8] A. Vaswani et al., “Attention Is All You Need,” Cornell University, Jun. 12, 2017. https://arxiv.org/abs/1706.03762.

[9] Google. "Google Colaboratory". Retrieved April 18, 2024. Available: https://colab.research.google.com/.

[10] A. Paszke et al., “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” arXiv.org, 2019. https://arxiv.org/abs/1912.01703