Investigating the Impact of Convolutional Neural Network Architectures on Sentiment Analysis in Twitter Data: Final Report

G062 (s1817967, s1852034, s2442987)

Abstract

Sentiment analysis has become an essential research area in natural language processing, with a particular focus on social media data, such as Twitter. In this paper, we investigate the influence of varying kernel sizes, number of layers, and input dimensions in Convolutional Neural Network (CNN) architectures on the performance of sentiment analysis tasks. We experimented with multiple CNN configurations, including 2-Conv-NN, 3-Conv-NN, 4-Conv-NN, and (4++)-Conv-NN, and evaluated their performance in relation to kernel size and number of layers. Our results indicate that the accuracy of the models is sensitive to the choice of kernel size and the number of layers. The optimal kernel size and number of layers vary depending on the input dimensions. We also found that increasing the kernel size or number of layers could lead to issues such as loss of detail in smaller features, overfitting, and higher computational complexity. Our findings provide valuable insights into the design of CNN architectures for sentiment analysis in Twitter data and can serve as a basis for further research in the field.

1. Introduction

The field of sentiment analysis has gained significant attention in recent years due to the exponential growth of textual data available on the internet, making it an essential tool for businesses and organizations to gain insights into customer feedback, track brand sentiment, and monitor public opinion. The increasing importance of sentiment analysis in various fields such as healthcare, politics, and social sciences has led to a surge in research efforts towards improving its accuracy and effectiveness.

In this paper, our goal is to enhance the research presented in (Priyanka, 2021) by extending the sentiment analysis on Twitter from a binary classification to a more comprehensive three-label classification. We have achieved this by developing several convolutional neural networks (CNN) models with varying layer counts and kernel size. Each neural network is composed of the following layers in sequence: embedding layer, dropout, a variable number (n) of convolutional layers, flatten, dense, dropout, ReLU activation, dense, and softmax activation. The specific value of n depends on the model being utilized.

Additionally, we explore the impact of varying the kernel size and the number of layers in convolutional neural networks (CNNs) on the testing accuracy and overall performance of sentiment analysis models. In doing so, we conduct a systematic investigation into the trade-offs between model complexity and prediction accuracy, as well as the computational efficiency associated with different CNN architectures.

By comparing several CNN configurations with distinct kernel sizes and layer counts, our study aims to identify the optimal balance between model complexity and performance for the specific task of Twitter sentiment analysis. Furthermore, we examine the generalizability of our findings to other domains and data sources beyond Twitter. To demonstrate the versatility of our approach, we also utilize Reddit data to perform the experiment, showcasing the potential applicability of our proposed models to different social media platforms and textual data sources.

2. Data Set and Task

A dataset from Kaggle (kag) was used in the study by (Priyanka, 2021) and labelled as either positive or negative by the model. Emoticons, usernames, hashtags, URLs, and user mentions were among the elements in the data that needed to be transformed into a standard format before being fed into the model. In this paper, we classify the data as positive, negative or neutral. Therefore, we use a different dataset from Kaggle (dat) to test the accuracy of our model.

The Twitter dataset used in this paper contained 106,530 values in the form of comma-separated values, including tweet IDs, tweets, and their associated sentiments. On the other hand, the Reddit dataset consists of 24,831 values in the form of comma-separated values, similar to the Twitter dataset. Several pre-processing steps were performed to standardise the data before it was input into the model as listed below.

- All tweets are converted to lowercase.
- 2 or more dots are replaced with a single space.
- Unnecessary symbols such as spaces and quotes are removed from the end of tweets.
- 2 or more spaces are replaced with a single space.

- Any punctuation "[’"?!,.():;]" and "-" from a word is removed.
- 2 or more letter repetitions are converted to 2 letters.
- Non-valid words are removed. A valid word is one that starts with an alphabet and continues with an alphabet, a number, a full stop "." or an underscore "_"

Twitter provides its users with a number of unique features such as user mentions, URLs, emoticons, hashtags, and retweets. Regular expressions were used to match and replace or remove the relevant information to handle the complexity of the text. The term "USER_MENTION" was used to replace user mentions with the regular expression "@[\S]+". URLs were replaced with the word "URL" using the expression "((www\.[\S]+)|(https?://[\S]+))". "#(\S+)" was used to match hashtags, and the "#" symbol was removed.

Tweets that have already been sent by another user and are shared by other users are known as retweets. Retweets are identified by the letters "RT". The matching regular expression "\brt\b" was used to eliminate the letters "RT" from the retweets. The retweets are then subjected to the previously mentioned general pre-processing steps. Finally, emoticons that can convey various emotions were replaced with either "EMO_POS" or "EMO_NEG" depending on the emotion conveyed. The list of emoticons being matched is shown in Figure 1.

Figure 1. The list of emoticons taken from (Priyanka, 2021)

Emoticon(s)	Type	Regex	Replacement
:), :), :-), (:, (:, (-:, :’)	Smile	(:\s?\)|:-\)|\(\s?:|\(-:|:\’\))	EMO_POS
:D, : D, :-D, xD, x-D, XD, X-D	Laugh	(:\s?D|:-D|x-?D|X-?D)	EMO_POS
;-), ;), ;-D, ;D, (;, (-;	Wink	(:\s?\(|:-\(|\)\s?:|\)-:)	EMO_POS
<3, :*	Love	(<3|:*)	EMO_POS
:-(, : (, :(,):,)-:	Sad	(:\s?\(|:-\(|\)\s?:|\)-:)	EMO_NEG
:,(, :’(, :"(	Cry	(:,\(|:\’\(|:"\()	EMO_NEG

After pre-processing the dataset, we ran a series of experiments to investigate the relationship between the model’s accuracy and the number of kernel sizes. We used three different CNN configurations, each with two, three, or four layers. In the experiments, we increased the kernel size of each model incrementally from 2 to 20. To assess performance, we divided the dataset into five equal parts and ran each model through 5-fold cross-validation. The results of these five trials were then averaged to determine the average accuracy. In another experiment, we increased the number of layers from 2 to 10 while keeping the kernel size constant at 3, 4, and 5. As in the previous experiment, we used the average results of 5-fold cross-validation to assess testing accuracy.

3. Methodology

We implemented our model using Keras with TensorFlow as the backend. The models were trained on dense vector representations of tweets, with the vocabulary consisting of the top 90,000 words from the training dataset. Each word in the vocabulary was assigned an integer index from 1 to 90,000, corresponding to its rank in the dataset. The integer index 0 was reserved for a special padding word.

3.1. Word Embedding

The first layer of our model is a word embedding layer, which represents each word as a 200-dimensional vector. We used the GloVe algorithm (Pennington et al., 2014) for generating these word embeddings.

GloVe (Global Vectors for Word Representation) (glo) is a pre-trained word embedding algorithm developed by Stanford University researchers. To understand the GloVe algorithm, we need to know what co-occurrence probabilities is.

$$p_{ij} = P(word_j|word_i) = \frac{x_{ij}}{x_i}$$

In this formula, p_{ij} represents the co-occurrence probability, while x_i denotes the frequency of $word_i$, and x_{ij} refers to the frequency of $word_j$ appearing in a context window with $word_i$ as the central word. To determine the relationship between $word_i$ and $word_j$, the algorithm introduces a third word, $word_k$. We can then compute the ratio $\frac{p_{ik}}{p_{jk}}$. If the value is close to 1, it indicates that $word_i$ and $word_j$ are either both related or unrelated to $word_k$. If the value is greater than 1, it implies that $word_i$ is more relevant to $word_k$. The fundamental concept of GloVe is to ensure that the word vectors of these three words satisfy this co-occurrence probability ratio.

$$f(v_i, v_j, u_k) = \frac{p_{ik}}{p_{jk}}$$

Here, v_i, v_j, and v_k represent the vectors of the three words, respectively. It is important to note that the arguments of the function f are vectors, while the right side of the equation is a scalar. We can address this by taking the dot product of the arguments, resulting in the following equation.

$$f((v_i - v_j)^T u_k) = \frac{p_{ik}}{p_{jk}}$$

We can see that the left side of the equation represents a difference, while the right side is a fraction. The algorithm sets the function f to be equal to the exponential function (exp), resulting in a new equation:

$$e^{v_i^T u_k - v_j^T u_k} = \frac{p_{ik}}{p_{jk}}$$

$$v_i^T u_k = log(x_{ik}) - log(x_i)$$

However, there is an issue since the inner product of vectors obeys the commutative law, where $v_i^T u_k = u_k^T v_i$. Applying this law to the previously mentioned formula could result in incorrect equations.

$$log(x_{ik}) - log(x_i) = log(x_{ki}) - log(x_k)$$

As $x_{ki} = x_{ik}$, we can deduce that $log(x_i) = log(x_k)$. However, this equation is not an identity. In the original paper, the authors propose a new equation to address this issue:

$$v_i^T u_k = log(x_{ik}) - b_i - b_k$$

where b_i and b_k are trainable biases.

3.2. Model Architecture

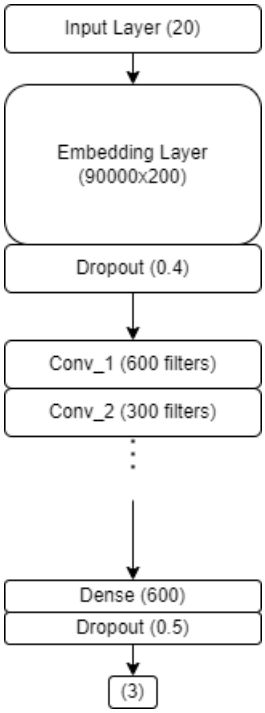

Figure 2. Model architecture

We use convolutional neural networks for binary sentiment analysis. In this model, we set a parameter, *max_length*, to control the input shape. If a tweet has fewer than *max_length*, its dense vector representation is padded with a special padding word until its length reaches *max_length*. Following the word embedding layer, there are convolutional layers, each followed by a ReLU activation function. After the convolutional layers, we use a Flatten layer to reduce the data's dimensionality. The output of the Flatten layer is then passed to a fully connected layer, which produces a single value. A sigmoid activation function is applied to obtain the probability value. To regularize our network and prevent overfitting, we also include dropout layers after the embedding layer and the fully connected layer.

3.3. Classification of Three Sentiments

In the paper by (Priyanka, 2021), Twitter sentiment classification is limited to positive and negative sentiments. The author used a sigmoid activation function in the final layer, generating a single number ranging from 0 to 1 to represent the two sentiments. In our experiment, we adapt our model to analyze three sentiments: positive, neutral, and negative. First, we used a different dataset that has 3 labels as mentioned in Section 2. Then, we use one-hot encoding, resulting in a label format as a vector containing only 0s and 1s. To handle three sentiments in the output, we increase the number of output units in the fully connected layer to 3. Since the classification task is no longer binary, we opted for the softmax function in the final layer instead.

4. Experiments

We conduct experiments with varying kernel sizes, numbers of layers, and *max_length* to study how these factors impact our model's performance. By analyzing the trade-off between the model's depth, represented by the number of layers, and its width, indicated by the kernel size, we aim to identify the optimal combinations of dimensions for our models to achieve their best performance.

4.1. Setup

We perform all of our experiments by using **keras** (ker) with **Tensorflow** (ten) backend to build the Twitter sentiments analysis models. Throughout the experiments, we use data mentioned in section 2, with 106,530 tweets in total for each sentiment, positive(1), negative(-1) or neutral(0). As described in Section 2, each experiment was run 5 times using 80% of data as training and 20% of data as testing, employing 5-fold cross-validation while using 10% of training data to validate.

We choose *max length* of 40 and 100 for all experiments. 50 is chosen based on the suggestion of 40 by the paper (Priyanka, 2021) and considering the dataset used by the paper is similar to the dataset used in this paper. 100 is chosen as a larger value to evaluate the effects of varying kernel size and the number of layers. Considering that the dataset's data lengths range from 1 to 50 with a mean of 20, we generate a new dataset from the original one to obtain data lengths between 80 and 100 for the experiments using *max length* of 100. This new dataset is created by combining multiple data points from the original dataset with the same sentiment labels. The other reason for choosing *max length* of 100 is that every word in the new dataset will not be removed when processing. With the experiments using *max length* of 40, some of the data will be shortened to a length of 40 as the original dataset has data of length from 1 to 50.

4.2. Baseline

As a baseline, we use a model with a single CNN layer, featuring a kernel size of 2 and *max length* values of 40 and 100. We select this single-layer CNN model as our baseline to compare its accuracy against models with multiple CNN layers. In our experiments, we employ temporal convolution, which is well-suited for analyzing sequential data like tweets. We begin with a kernel size of 2 to initially examine the extraction of smaller local features across the layers.

4.3. Convolutional Neural Networks

For the implementation of the Convolutional Neural Network model used in the experiments, we used keras with TensorFlow backend. To train our CNN models, we utilized dense vector representations of tweets and a vocabulary comprising the top 90,000 words in terms of occurring frequency from the training dataset. Each word within the vocabulary is assigned an integer index ranging from 1 to 90,000, where the index signifies the word's rank in the dataset. The integer index 0 is designated for the special padding word. Additionally, each of these 90,001 words is characterized by a 200-dimensional vector.

The initial layer of our models is the Embedding layer, which consists of a matrix with dimensions $(v + 1)d$, where v represents the vocabulary size $(90,000)$ and d is the dimension of each word vector (200). The embedding layer is initialized with random weights drawn from a normal distribution $N(0, 0.01)$. Each row in this embedding matrix represents the 200-dimensional word vector for a vocabulary word. If a word in the vocabulary corresponds to a GloVe word vector provided by the StanfordNLP group (glo), we use the GloVe vectors to seed the relevant row in the embedding matrix. Each tweet, in its dense vector representation, is padded with zeros at the end until its length matches the *max length* parameter, which we adjust in our experiments. We trained our model using categorical cross-entropy loss with the weight update scheme defined by Adam et al. To prevent overfitting, we set the model to train for 8 epochs.

CNN architectures used in the experiments are as follows:

- **2-Conv-NN**: For better regularisation and to avoid overfitting, we used the vocabulary size of 90,000 and increased the dropout rates to 0.4 after the embedding layer and 0.5 after the fully connected layer in this architecture. We increased the number of filters in the first convolutional layer to 600 and added a second convolutional layer with 300 filters. To preserve input tweet features that may have been lost during max pooling, the Global MaxPool layer was replaced with a Flatten layer. We also increased the number of units in the fully connected layer to 600. The network's learning, regularisation, and validation accuracy improved as a result of these changes. The full architecture is: embedding_layer(900001×200) → dropout(0.4) → conv_1(600 filters) → relu → conv_2(300 filters) → relu → flatten → dense(600) → dropout(0.5) → relu → dense(3) → softmax.

- **3-Conv-NN**: Following the second convolutional layer, this architecture adds a third convolutional layer with 150 filters. The model architecture is: embedding_layer(900001×200) → dropout(0.4) → conv_1(600 filters) → relu → conv_2(300 filters) → relu → conv_3(150 filters) → relu → flatten → dense(600) → dropout(0.5) → relu → dense(3) → softmax.

- **4-Conv-NN**: In this configuration, we added a fourth convolutional layer with 75 filters after the third convolutional layer. The overall architecture is: embedding_layer(900001×200) → dropout(0.4) → conv_1(600 filters) → relu → conv_2(300 filters) → relu → conv_3(150 filters) → relu → conv_4(75 filters) → relu → flatten → dense(600) → dropout(0.5) → relu → dense(3) → softmax.

- **(4++)-Conv-NN**: For the number of convolutional layers greater than 4, we increase the convolutional layer with 600 filters before the first convolutional layer. For example, with a number of layers of 5, the overall architecture is: embedding_layer(900001×200) → dropout(0.4) → conv_1(600 filters) → conv_1(600 filters) → relu → conv_2(300 filters) → relu → conv_3(150 filters) → relu → conv_4(75 filters) → relu → flatten → dense(600) → dropout(0.5) → relu → dense(3) → softmax, as shown in Figure 2.

4.4. Experiments on how the accuracy changes by varying the kernel size

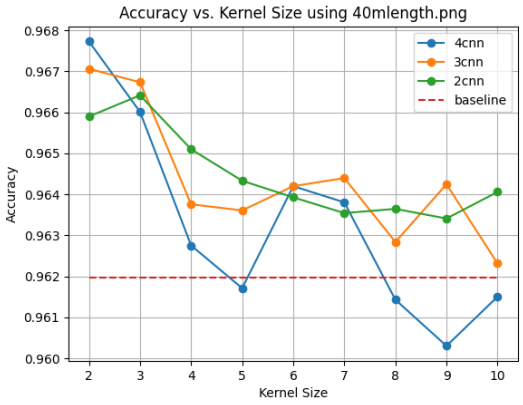

Figure 3. Accuracy vs. Kernel Size with 40 max length

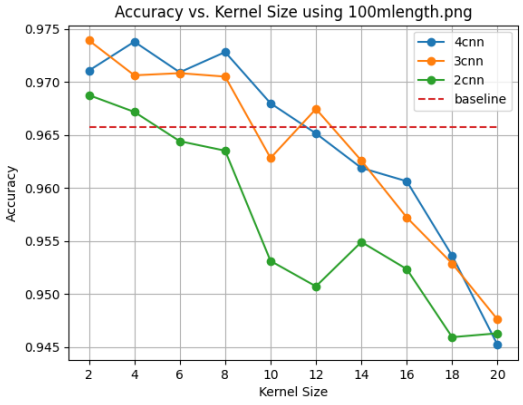

Figure 4. Accuracy vs. Kernel Size with 100 max length

The experiments were conducted to investigate the effect of varying kernel sizes on the performance of the CNN

Number of Layers	2	3	4	5	6	7	8	9
Baseline	96.23	96.23	96.23	96.23	96.23	96.23	96.23	96.23
Kernel Size of 3	96.64	96.67	96.60	96.40	96.28	96.42	96.22	96.20
Kernel Size of 4	96.51	96.38	96.28	96.06	96.30	96.14	96.05	95.81
Kernel Size of 5	96.43	96.36	96.17	96.19	95.93	95.68	96.09	??

Table 1. max length of 40, kernel dimension from 2 to 10

Number of Layers	2	3	4	5	6	7	8	9	10
Baseline	96.57	96.57	96.57	96.57	96.57	96.57	96.57	96.57	96.57
Kernel Size of 3	96.72	97.06	97.38	97.74	97.28	97.03	96.93	96.56	97.09
Kernel Size of 4	96.44	97.08	97.09	97.34	97.01	96.49	97.19	96.61	96.94
Kernel Size of 5	96.35	97.05	97.28	97.26	97.16	96.77	96.91	96.13	95.99

Table 2. max length of 100, kernel dimension from 2 to 10

models. The results show that the model's performance is sensitive to the choice of kernel size, and the accuracy decreases when the kernel size increases if the number of layers is fixed. Table 1 and Table 2 show the results of our experiments in this section.

With a fixed number of layers and a *max length* of 40, Figure 3 demonstrates that the accuracy of the models decreases as the kernel size gets smaller. Although the graphs exhibit significant fluctuations, it is not entirely conclusive that the 4-layer model underperforms compared to the others. We employ a 5-fold cross-validation technique, and each fold uses a different seed to initialize the embedding matrix. Despite these measures, the graphs continue to display considerable fluctuations, and we were unable to conduct further experiments due to computational constraints. Nonetheless, it is suggested that for *max length* of 40, the 4-layer model may perform worse than its counterparts.

As the *max length* increases from 40 to 100, the trend persists, as illustrated in Figure 4. The accuracy of all models, regardless of the number of layers, decreases as the kernel size grows. Notably, when the maximum sequence length is set to 100, the 2-layer model performs the worst, compared to its performance with *max length* of 40. This observation suggests that a deeper network is necessary for handling larger input dimensions. Additionally, the decline in the model's accuracy when the kernel size is increased from 11 to 20 is more pronounced than the decrease observed when the kernel size is raised from 2 to 10. This suggests that overfitting happens due to larger kernel sizes capturing more extensive local context, considering a broader range of neighbouring words. It might also be the case that with larger kernel sizes, the model might miss some essential local features or patterns that are vital for sentiment analysis. Furthermore, it is reasonable to conclude that a 3-layer network is sufficient for inputs with a maximum length of 100, as the difference in accuracy between the 3-layer and 4-layer models is negligible.

4.5. Experiments on how the accuracy changes by varying the number of layers

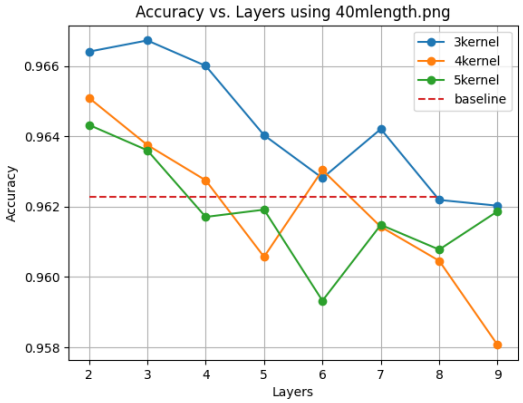

Figure 5. Accuracy vs. Number of Layers with 40 max length

The experiments aimed to explore the impact of varying the number of layers on the performance of the CNN models. The findings indicate that performance generally improves as the number of layers increases, up to a certain threshold, beyond which the performance begins to decline. The results of our experiments in this section are presented in Table 3 and Table 4.

When the kernel size and *max length* are fixed at 40, Figure 5 reveals that the models' accuracy declines as the kernel size decreases, similar to the pattern observed in Figure 3. The model with a kernel size of 3 achieves the highest accuracy on the dataset we utilized. While larger kernel sizes can capture a wider local context by considering a more extensive range of neighboring words, a kernel size of 3 proves to be the most effective for capturing our dataset's local context.

As *max length* increases from 40 to 100, the trend changes, as depicted in Figure 6. The accuracy of all models, irre-

Kernel Size	2	3	4	5	6	7	8	9	10
Baseline	96.23	96.23	96.23	96.23	96.23	96.23	96.23	96.23	96.23
2 Layers	96.59	96.64	96.51	96.43	96.39	96.35	96.36	96.34	96.41
3 Layers	96.71	96.67	96.38	96.36	96.42	96.44	96.28	96.42	96.23
4 Layers	96.77	96.60	96.28	96.17	96.42	96.38	96.14	96.03	96.15

Table 3. max length of 40, number of layers from 2 to 10

Kernel Size	2	4	6	8	10	12	14	16	18	20
Baseline	96.57	96.57	96.57	96.57	96.57	96.57	96.57	96.57	96.57	96.57
2 Layers	96.87	96.44	96.18	94.87	94.95	95.27	95.68	95.23	94.59	94.62
3 Layers	97.39	97.08	97.18	96.78	96.28	96.74	96.26	95.72	95.29	94.76
4 Layers	97.11	97.09	96.93	96.88	96.80	96.51	96.19	96.06	95.36	94.52

Table 4. max length of 100, number of layers from 2 to 10

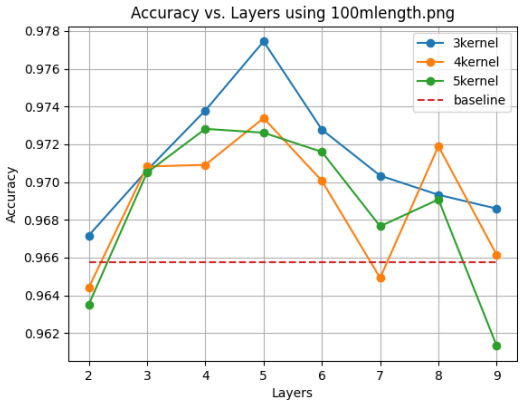

Figure 6. Accuracy vs. Number of Layers with 100 max length

spective of kernel size, peaks at around 5 layers. In line with the experiment using a *max length* of 40, the model with a kernel size of 3 still performs the best, but the difference between this model and the others is smaller. This finding implies that as the input length grows, a kernel size of 3 may no longer be the optimal choice for identifying input patterns or local features. This observation makes sense, given that each data point in the dataset with a *max length* of 100 is a combination of data from the original dataset.

4.6. Interpretation and Discussion

In Section 4.4, we can observe that the model's accuracy decreases when the kernel sizes of models increase. The common trend from Figure 3 and Figure 4 is that the models' accuracy is the highest when kernel sizes are small. Hence, we can conclude that the optimal value of kernel sizes is either 2, 3 or 4 depending on the other parameters of the models.

In Section 4.5, Figure 6 shows that the accuracy increases to a peak at around 5 layers and then begins to decrease as the number of layers continues to grow. Although this trend is not evident in Figure 3, where the models' accuracy actually declines as the number of layers increases, the model with 4 layers exhibits a similar pattern to those in Figure 6. We hypothesize that the models in 5 may share the same trend, with their "peak" occurring at around 2 or 3 layers. This pattern may be due to the fact that increasing the number of layers improves the model's capacity to learn complex patterns and representations in the data. However, the model might not receive sufficient training as a result of the increased complexity.

In summary, there are several reasons for the pattern observed in Section 4.4 and Section 4.5 listed as follows.

- **Unrelated Tokens**: As the kernel size increases, it encompasses more tokens and contextual information than before. This may result in the inclusion of unrelated tokens when calculating the features, which can blur the feature and reduce the accuracy of the original meaning of words. A similar effect occurs when the number of layers is increased.

- **Fewer Features**: As we did not apply padding in the convolutional layers, the length of intermediate data decreases after each convolutional layer. Increasing the kernel size or the number of layers leads to fewer output features in the final convolutional layer. This may cause the model to lose details in some smaller features.

- **Overfitting**: Increasing the kernel size and the number of layers can lead to overfitting, causing the model to become overly specialized for the training data and perform poorly on new data. This is because larger kernel sizes can capture more global features, but they may also inadvertently capture irrelevant information that is specific to the training set.

- **Complexity**: Larger kernel sizes and an increased number of layers can result in higher computational complexity and longer training times, making it more challenging to optimize the model. Initially, when training the model with large kernel sizes, we thought that the lower accuracy might be due to an insufficient number of epochs. However, even after increasing the epochs, the issue persisted. Later, when training the model with a greater number of layers, we did observe an increase in accuracy when the number of epochs is increased, but it was accompanied by a certain degree of overfitting.

4.7. Bugs and Errors

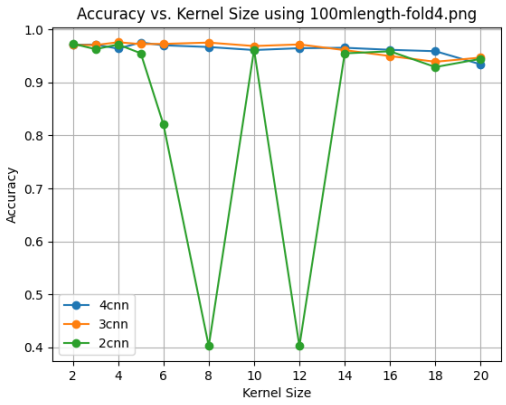

Figure 7. 2 Layers Model Producing a Result with Accuracy of 0.4

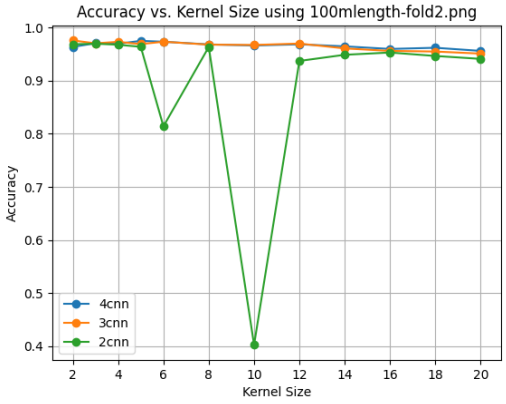

Figure 8. 2 Layers Model Producing a Result with Accuracy of 0.4 and 0.8

It is worth noting that, as depicted in Figure 7, the accuracy of our models may occasionally fail to improve beyond 0.4 when using kernel sizes of 8 and 12 in a 2-layer model with *max length* of 100. In such cases, the model consistently predicts all test data as either positive or negative, since 0.4 of our test data is positive and another 0.4 is negative. We believe this could be attributed to the models becoming trapped in a local minimum during training. It should be noted that the number following the word "fold" in the figure's title refers to the iteration within the 5-fold cross-validation process.

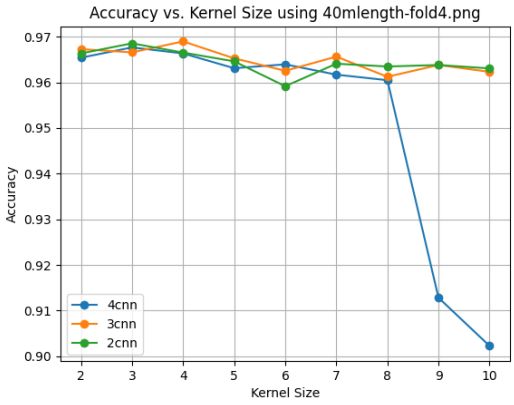

Figure 9. 4 Layers Model Producing a Result with Accuracy of 0.9

However, there are instances when the models yield results that are better than 0.4 but still relatively worse than other models. For example, in Figure 8, the 2-layer model with a kernel size of 6 and *max length* of 100 achieves an accuracy of approximately 0.8. Similar outcomes are observed for other models besides the one mentioned above, as shown in Figure 9. These aforementioned cases occur infrequently and only arise when specific random seeds are used to initialize the embedding matrix of the data.

5. Related work

Sentiment analysis has been an active research area in natural language processing, with numerous approaches proposed to address the challenges of extracting sentiment from text data. In this section, we review several methods and models employed in the literature, focusing on those that specifically target Twitter data, as well as those that provide insights into future work.

5.1. Traditional Machine Learning Approaches

Earlier approaches to sentiment analysis on Twitter data relied on traditional machine learning techniques, such as Support Vector Machines (SVM), Naïve Bayes, and logistic regression. These models were usually coupled with feature engineering methods, like bag-of-words, term frequency-inverse document frequency (TF-IDF), and n-grams, to represent the text data. For example, Go et al. (2009)[(Go et al., 2009)] and Pak & Paroubek (2010)[(Pak et al., 2010)] used SVM and Naïve Bayes, respectively, to classify tweet sentiments.

5.2. Deep Learning Approaches

As deep learning gained momentum, researchers started to explore various neural network architectures for sentiment analysis. Convolutional Neural Networks (CNNs) were among the first to be applied to this task, as they can effi-

ciently capture local features in the text. Kim (2014)[(Kim)] proposed a CNN model that leveraged pre-trained word embeddings for sentiment analysis, achieving competitive performance on multiple benchmark datasets. Severyn & Moschitti (2015)[(Severyn & Moschitti, 2015)] further demonstrated the potential of deep CNNs for sentiment analysis on Twitter data.

5.3. Transformer-Based Models

The introduction of the Transformer architecture by (Vaswani et al., 2017) has had a significant impact on the field of natural language processing, leading to the development of several powerful pre-trained language models. These models have demonstrated remarkable capabilities in capturing syntactic and semantic information in text, outperforming previous techniques on a wide range of tasks.

5.3.1. BERT

BERT (Bidirectional Encoder Representations from Transformers) by (Devlin et al., 2019) is a pre-trained language model that leverages the Transformer architecture for bidirectional context learning. By pre-training on large text corpora and fine-tuning on specific tasks, BERT has achieved state-of-the-art results in numerous NLP applications, including sentiment analysis. For example, (Sun et al., 2020) fine-tuned BERT for sentiment analysis on the SemEval dataset, outperforming previous models and setting a new benchmark.

5.3.2. GPT

GPT (Generative Pre-trained Transformer) by (Radford et al., 2018)) is another Transformer-based language model. GPT adopts a unidirectional context learning approach and has proven effective in various NLP tasks, including sentiment analysis. While GPT's unidirectional nature might seem limiting compared to BERT's bidirectional approach, it has still achieved impressive results by leveraging the power of the Transformer architecture and pre-training on massive amounts of text data.

5.3.3. RoBERTa

RoBERTa (A Robustly Optimized BERT Pretraining Approach) by (Liu et al., 2019) is an optimized version of BERT that incorporates several training and model architecture improvements. These enhancements have resulted in even better performance on a variety of NLP tasks, including sentiment analysis. RoBERTa has consistently set new state-of-the-art results on multiple benchmarks, demonstrating the continuous evolution and potential of Transformer-based models.

5.4. Pre-processing and Applications of CNN Models in Twitter Sentiments Analysis

In (Priyanka, 2021), the authors address sentiment classification on a Twitter dataset using various machine learning and deep learning methods, including logistic regression, MLP, RNNs with LSTM units, CNNs, and XGBoost. They achieve a classification accuracy of 83.58% on Kaggle's public leaderboard using a majority vote ensemble of their top five models, showcasing the benefits of combining techniques for sentiment analysis on Twitter data.

Their work explores preprocessing techniques and baseline models such as Naive Bayes, maximum entropy, decision trees, and random forests, emphasizing the importance of Twitter-specific features like hashtags, retweets, and internet slang to capture the informal language used on the platform. This study demonstrates the effectiveness of diverse sentiment analysis approaches and ensemble methods, providing valuable insights into sentiment classification challenges and opportunities for tweets.

6. Conclusions

In this paper, we presented a comprehensive study of Convolutional Neural Networks (CNNs) for sentiment analysis on Twitter data. Our work explored the impact of various CNN architectures and hyperparameters, such as kernel size, number of layers, and maximum input length, on model performance. Our experiments revealed that the choice of kernel size and the depth of the network play crucial roles in the model's accuracy. A kernel size of 3 and a 3-layer network were found to be the most effective in capturing local context and achieving high accuracy for our dataset.

Our results also demonstrated that deeper networks are necessary for handling larger input dimensions, while smaller kernel sizes are more effective for identifying input patterns or local features. We provided insights into the reasons behind the decrease in accuracy when increasing kernel size and layers, such as unrelated tokens, fewer features, overfitting, and increased complexity.

By comparing our results with traditional machine learning approaches, deep learning techniques like RNNs, LSTMs, and Transformer models, we highlighted the advantages and limitations of CNNs in the context of sentiment analysis on Twitter data.

In future work, we aim to explore other deep learning techniques, such as attention mechanisms and pre-trained language models like BERT, to further improve the performance of sentiment analysis models. Moreover, incorporating techniques like data augmentation and transfer learning could help address the challenges posed by limited and noisy training data. Additionally, investigating the impact of combining CNNs with other deep learning architectures, such as RNNs or LSTMs, may provide valuable insights into developing more accurate and robust models for sentiment analysis.

References

Kim2014. URL https://arxiv.org/pdf/1408.5882.

Twitter and reddit sentimental analysis dataset. URL

https://www.kaggle.com/datasets/cosmos98/twitter-and-reddit-sentimental-analysis-dataset.

Glove: Global vectors for word representation. URL https://nlp.stanford.edu/projects/glove/.

Kaggle. URL https://www.kaggle.com/.

Keras. URL https://keras.io/.

Tensorflow. URL https://www.tensorflow.org/.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Go, Alec, Bhayani, Richa, and Huang, Lei. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

Liu, Yinhan, Ott, Myle, Goyal, Naman, Du, Jingfei, Joshi, Mandar, Chen, Danqi, Levy, Omer, Lewis, Mike, Zettlemoyer, Luke, and Stoyanov, Veselin. Roberta: A robustly optimized bert pretraining approach, 2019.

Pak, Alexander, Paroubek, Patrick, et al. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pp. 1320–1326, 2010.

Priyanka, Vedurumudi. Twitter sentiment analysis using deep learning. 2021. doi: 10.2139/ssrn.3885137. URL http://dx.doi.org/10.2139/ssrn.3885137.

Radford, Alec, Narasimhan, Karthik, Salimans, Tim, Sutskever, Ilya, et al. Improving language understanding by generative pre-training. 2018.

Severyn, Aliaksei and Moschitti, Alessandro. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 373–382, 2015.

Sun, Chi, Qiu, Xipeng, Xu, Yige, and Huang, Xuanjing. How to fine-tune bert for text classification?, 2020.

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