Convolutional Approach to Sentiment Classification

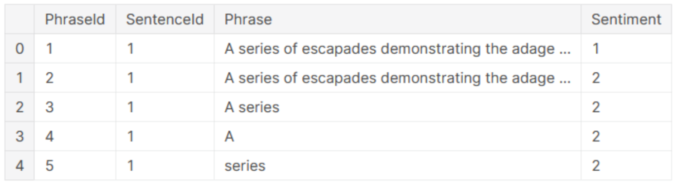
1st Tim Heydrich
Department of Computer Science
Lakehead University
Tim Heydrich, 1121117
March 20, 2020

***Abstract*—Classification problems are with regression problems the main part of machine learning. This makes it no surprise that convolutional neural networks (CNNs) are also applied in this area. In this paper different model structures were tested to see which is most effective for sentiment classification with the Rotten Tomatoes sentiment dataset [1] as part of an assignment of the Natural Language course.**

I. Introduction

Machine learning tasks can usually be categorized into two main groups, classification and regression. Classification describes the more straight forward approach of determining the class of a certain input. These classifications can be very simple binary classification of for example cats and dogs or more complex multi-class classification with multiple possible categories.

Regression differs from classification in the way that the end goal usually represents any real value and is not limited to a certain number of classes. Regression usually revolves around predictions, for example stock predictions or sales predictions.



	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

Fig. 1. First five rows of dataset

In this paper the focus problem is the sentiment classification dataset. The dataset consist of reviews and their associated sentiment class. The dataset contains a total of five sentiment classes (fig.1). The goal is to determine the sentiment of any given review. This dataset does provide a challange however, the dataset is very imbalanced (fig. 2). This means that the dataset needs to be balanced before training as the model would otherwise favour one class over the others.

II. Background

A. Data Balancing

Rotten Tomatoes Sentiment Dataset is, as mentioned in I, not balanced which can lead to the model being more likely to predict the class with the most examples. This means that the dataset needs to be balanced before using it to train the model.

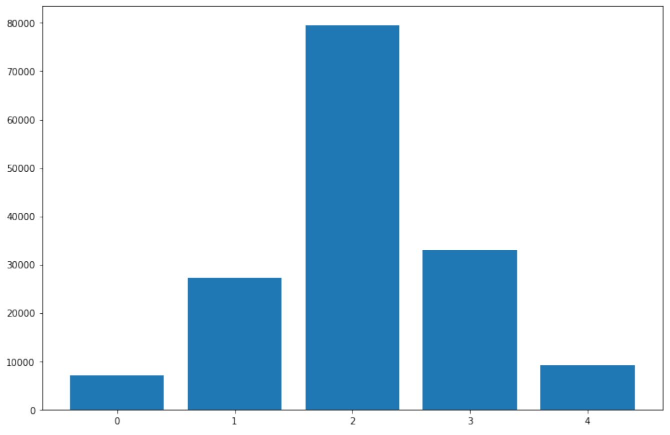


Fig. 2. Imbalanced Data

Data exploration revealed that sentiment calss number ’2’ is the most common in the dataset. During the exploration it also became apparent that the dataset does not only consist of the reviews but also fragments of the reviews. The reviews are contained as a whole but then also split up into smaller segments which have their own sentiment value. When utilizing only the complete sentences the dataset is more balanced. This means that only utilizing the complete sentences provides one approach for training the dataset. Another approach which was

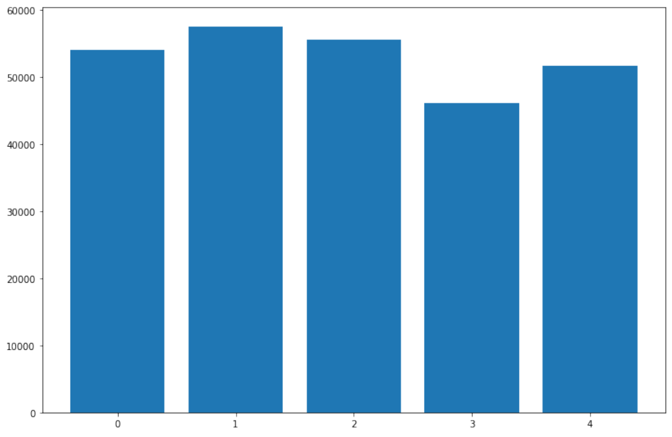
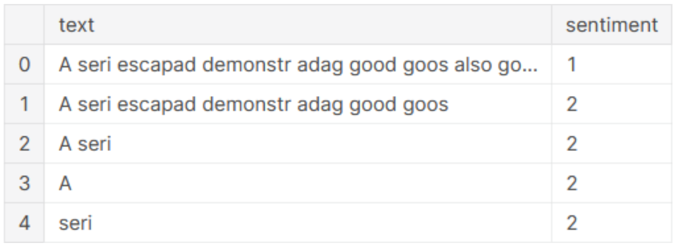


Fig. 3. Balanced Training Data

also utilized in the process of completing this assignment was to repeat the not so balanced classes to artificially increase their sample count. This means that class ’0’ was repeated an additional ten times, class ’1’ was repeated twice, class ’2’ was not repeated at all, class ’3’ was repeated once and class ’4’ was repeated six times. This resulted in the dataset being balanced (fig. 3).

B. Preprocessing

Preprocessing can have a big impact on the accuracy of the model. There are several ways to preprocess textual data. The simplest one is to remove the stop words as well as the punctuation. The stop words are being basic words in the language, in English examples of stop words are ”and, they, as” and many others. The punctuation that were removed from the data in this paper are ”””?:!.,;’-‘() /*%!̂@”#$_ ”. Other common preprocessing steps are stemming and lemmatization. These two methods are very similar as both of them reduce the length of the words by removing pre- and suffixes. However, stemming can produce words that are not actually words in the language where as lemmatization always produces words that are in the language. Furthermore, the labels, the sentiment classes, were one hot encoded, this meant that the output layer would need five neurons instead of one.



	text	sentiment
0	A seri escapad demonstr adag good goos also go...	1
1	A seri escapad demonstr adag good goos	2
2	A seri	2
3	A	2
4	seri	2

Fig. 4. Preprocessed Data first five rows

C. Vectorization

When working with text data it is important to turn the text into a vector of numbers so that the computer can understand it. There are several ways to do this. The one used in this paper is the Term Frequency–Inverse Document Frequency (TF-IDF) vectorization. TF-IDF is a term weighting approach where each word is given different weights. The term frequency is the number of times the target word appears in the document. The inverse document frequency ensures that words that appear often are given less importance. Doing this prevents that words such as for example ’the’ are given a very high weight even though they do not convey any important meaning. TF-IDF can be represented by the simple equation:

$$w_{\mathrm{i,j}} = tf_{\mathrm{i,j}} \times log(\frac{N}{df_{\mathrm{i}}})$$

Where $w_{\mathrm{i,j}}$ represents the weight of word i in document j, $tf_{\mathrm{i,j}}$ is the number of times word i appears in document j. N represents the total number of documents in the training corpus and df_{i} represents the number of documents that contain the word i. TF-IDF is an approach that produces one value per word. Utilizing TF-IDF can take some time as it requires the whole training data to create its vocabulary and performs multiple passes of the whole data.

There are other approaches such as Word2Vec which produce one vector per word. Word2Vec has not been applied in this assignment it is, however, worth mentioning. Word2Vec can be split into two subgroups, Continuous Bag-of-Words (CBOW) and Skip-Gram. CBOW predicts a target word based on context and Skip-Gram predicts a context based on a word. Word2Vec is a word embedding approach which is why it produces one vector per word instead of just a single value. Word2Vec works by training a three layer network on the corpus and utilizing the hidden middle layer as the feature vector of the word. Word2Vec tends to be more effective when digging deep into the structure and context of the corpus, since it is a neural network and requires training this can take some time depending on the size of the training corpus.

D. Convolutional Neural Networks

Convolutional Neural Networks [2] have been around for several year, however, in recent years they gained a lot of popularity due to their effectiveness especially with computer vision tasks. Development of more powerful graphic processing units (GPUs) made this growth possible due to their inherent ability to work in parallel as well as their strength when it comes to computing floating point units. Stronger GPUs drastically reduced the training time needed to teach a neural network. This meant that larger and deeper network structures were possible without needing days to complete the training of the architecture.

The strength of CNNs for computer vision tasks comes from the way the so called convolutional layers work. Each convolutional layer consists of multiple kernels, or filters, of a given size, the most commonly used sizes are 3x3, 5x5, 6x6. These kernels work similar to other kernel applications in computer vision, for example the edge detectors or blurs. The general kernel computation [3] is the same as for other kernel and traversal of the image is performed with a previously determined step size. Unlike other kernel applications the CNN kernels do not produce another image of the same size, the kernels do not overlap the edges of the data. In simpler terms, the size is reduced to the number of possible locations a kernel can be placed on the data without overlapping. As an example suppose a give 5x5 kernel is applied to a 32x32 image then the result after the convolution is a 28x28 ’image’. This kernel application clearly results in a reduction of dimensionality, feature extraction. Continuing the use of images as an example, images usually wither consist of a single channel, grey images, or three channels, colour images. Utilizing multiple kernels during each convolution layer not only reduces the dimensionality but also increases the channel count to the number of filters used in the layer. Each filter produces it’s own convolved ’image’ which are then further processed in the next layers. The use of multiple layers of convolution gives lots of feature maps with high and low level features extracted from the data.

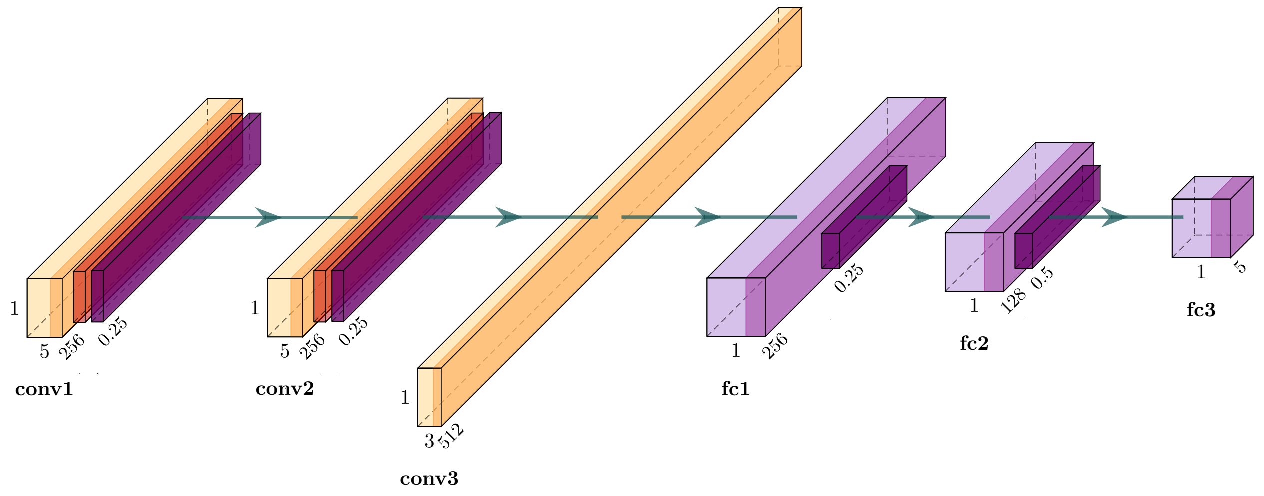


Fig. 5. Final Model Architecture Created with a modified version of [4]

Convolutional neural networks do however have a drawback when it comes to tabular data such as text analysis [5]. The fact that for tabular data two dimensional convolution, which was explained above, does not work as only a single row is relevant for the target value. This means that one dimensional convolution has to used on tabular data. One dimensional filters, for example 3x1, are not nearly as effective as two or multi dimensional filters. This limits the effectiveness of the convolution to extract enough and important features from the tabular data. This drawback is especially relevant when it comes to regression models but it can also be noticed in classification models. The use of CNNs in text analysis is usually supported through the use of sequence layers such as LSTM, however, they were not permitted for this assignment which is why only basic CNNs were used.

III. Model Architecture

A. Basic Architecture

Desgining an effective model posed challenging for several reasons, for example the issues mentioned in II-D but also the size of the vectors used in the training. During the lab session [?] the dataset used was a lot smaller as it was meant to introduce people to the world of Natural Language processing. The Rotten Tomatoes dataset used for the assignment was much larger and the resulting vectors from the vectorization were very large and required a lot of memory. This was feasible RAM wise due to the large amount of RAM available on both Google Colab as well as Kaggle. The issue was loading the model into memory as the large vector required large weight matrices which needed a lot of GPU memory which frequently resulted in the system running out of available memory.

The basic model that was first used during the lab session [6] consisted of a six layer CNN. The first layer was a one dimensional convolution layer with 64 channels and a kernel size of one. This was followed by a Max-Pooling layer and another convolution layer with the same kernel size but 128 filters. These convolution layers were then followed by the flatten layer and two dense layers. The first dense layer consisted of 64 neurons and the second was the output layer, meaning it only had a single neuron. This model was able to learn the training data, the performance was not acceptable however.

B. First steps and Final Model

The first model, which was created with PyTorch, continuously ran into the issue mentioned in III-A of running out of memory when attempting to load the model and even reducing the number of filters did not make it work. This led to the use of Keras as it's automated memory management seemed more sensible in this situation due to the lack of knowledge how to properly manage memory with PyTorch. Utilizing Keras made it possible to run a first model. This first model consisted of four one dimensional convoltuion layers followed by a flatten layer and then two dense layers as with dropout layer in between them. The first convolution layer had 64 filters and a kernel size of one, the other three had 128 filters each and also a kernel size of one. The first dense layer had 128 neurons and the second had five, for the one-hot encoded labels. The dropout layer had a drop factor of 0.5. This model had a very high training accuracy of approximately 91%. However, it suffered severely from over-fitting.

In an attempt to overcome this over fitting issues the first attempt was to add mode convolution layers which did not fix it nor did it improve the accuracy. Then max-pooling layers as well as dropout layers were added in between the convolution layers. This produced the first version of the final model. This model had the same amount of convolution layers with the same filter size as the first model but in addition, after the first and second convolution a max-pooling layer with a pool size of two was added as well as a drop out layer with a dropout value of 0.25.

The final model (fig. 5) consists of only three convolution layers where the first two were followed by a max-pooling layer as well as a dropout layer with a dropout of 0.25. The third convolutional layer did not have a pooling or dropout layer. The kernel sizes were also tweaked in the final model, the first two layers had a kernel size of five while the last one had a kernel size of 3. A total of three dense layers are used in the final model, the first layer contains 256 neurons and is followed by a dropout layer with a dropout of 0.25. The second layer had 128 neurons and was followed by a dropout with a factor of 0.5. The last layer contained 5 neurons to perform the prediction on the one-hot encoded label.

IV. Comparison and Evaluation

A. Comparison

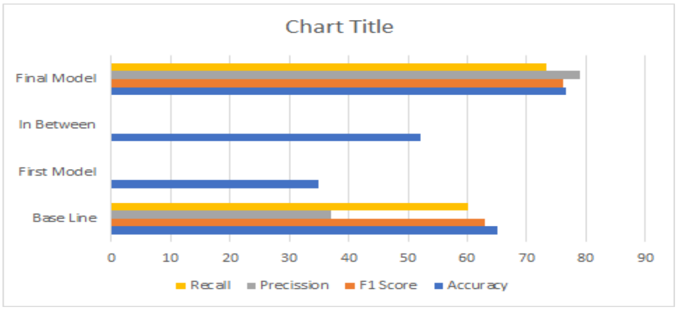


Fig. 6. Comparison as Graph

The comparisons performed in this section are performed on several different in between steps of the model development as well as the baseline performance given in the assignment google docs by Dr. Akilan. The testing dataset has been preprocessed in the same way as the training data II-B. However, as mentioned in II-C the vectorizer has only been trained on the training data which means that the weighted encoding for unknown words in the testing data is gonna be not correct. The results can be seen in table I as well as in graph form (fig. 7). It needs to be noted that the F1 Score metrics as well as the corresponding Recall and Precision metrics did not get implemented until the final model which is why those metrics are unavailable for early models. The accuracy over of the training and testing for each epoch can be seen in figure 7

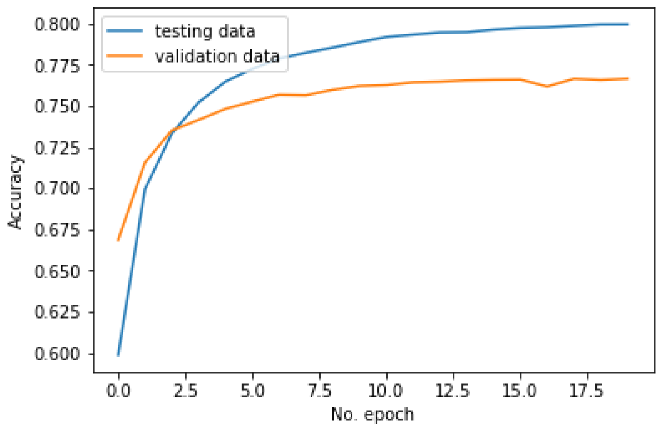


Fig. 7. Training and validation Graph

	Accuracy	F1 Score	Precision	Recall
Base Line	65	63	67	60
First Model	35	N/A	N/A	N/A
In Between Model	52	N/A	N/A	N/A
Final Model	**76.5**	**76.1**	**79.0**	**73.4**

TABLE I
Comparison of various models tested on the dataset

B. Evaluation

The comparison reveals that the final model had the best results when compared with the other models. This is most likely due to the tweaked network structure and the prevention of overfitting early on in the development.

V. Conclusion and Future Work

The models proposed in this paper can be improved upon in the future as the time constraint of this assignment made it hard to train multiple different models since the training took multiple hours for each model. In addition, models that utilize LSTM or RNN as well as CNN would further improve the performance quite significantly.

Overall, the model performed better than the proposed base line model by 10%. There is an issue however with the size of the model, the trained model is over 1Gb in size which is a very large model and makes utilizing and sharing it more challenging than a smaller model.

References

[1] cacoderquan, "Sentiment-analysis-on-the-rotten-tomatoes-movie-review-dataset," https://github.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/blob/master/train.tsv.
[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[3] V. Powell. Image kernels. [Online]. Available: http://setosa.io/ev/image-kernels/
[4] H. Iqbal, "Harisiqbal88/plotneuralnet v1.0.0," Dec. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.2526396
[5] J. Bird. Eeg brainwave dataset: Feeling emotions. [Online]. Available: https://www.kaggle.com/birdy654/eeg-brainwave-dataset-feeling-emotions
[6] A. Fisher and P. Sikka, "Lab 4: Multilayer 1d conv classification network in pytorch."