Hyperparameter Tuning on Text Classification using CNNs (Convolutional Neural Networks)

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Sarah Lei

Spring, 2020

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Hyperparameter Tuning on Text Classification using CNNs (Convolutional Neural Networks)

Sarah Lei
CS 4980
University of Virginia
ssl4jf@virginia.edu

***Abstract*—Text classification, the activity of labeling natural language texts with relevant categories from a predefined set, is a foundational task in many NLP (natural language processing) applications. These applications include sentiment analysis, web searching, and information filtering. By using text classifiers, companies can structure business information such as email, legal documents, web pages, chat conversations, and social media messages in a fast and cost-effective way. This allows companies to save time when analyzing text data, help inform business decisions, and automate business processes [1]. In this paper, we discuss the implementation of a model similar to Kim Yoon's Convolutional Neural Networks for Sentence Classification. We then discuss the performance of hyperparameter tuning on the model for training optimization. Following, we propose additional ways to improve the performance of the model.**

Keywords— sentiment analysis, CNN, NLP, text classification, hyperparameter, optimization

I. NLP

A. Definition

Natural Language Processing, usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. NLP is the driving force behind language translation applications, grammar check in word processors, and personal assistant applications like Siri, Alexa, and OK Google [7].

B. Difficulties

The nature of the human language introduces inherent difficulties to NLP. The rules that dictate the passing of information using natural languages are not easy for computers to understand. Furthermore, natural languages are difficult to translate into implementable algorithms due to these high-level and abstract rules. In addition, natural languages incorporate simpler rules, such as adding the character "s" to denote plurality of items. These rules that vary in complexity are the ambiguity and imprecise characteristics of the natural languages that make NLP difficult for machines to implement.

C. Techniques

NLP converts unstructured language data to a form that computers can understand by applying algorithms to identify and extract the natural language rules. Given textual input, algorithms are used to extract meaning associated with every sentence. Syntactic analysis and semantic analysis are the main techniques used to complete Natural Language Processing tasks [7]. Syntax refers to the arrangement of words in a sentence such that they make grammatical sense. In NLP, syntactic analysis is used to assess how the natural language aligns with the grammatical rules.

Computer algorithms are used to apply grammatical rules to a group of words and derive meaning from them. Some syntax techniques include lemmanization, word segmentation, and sentence breaking. Lemmanization involves reducing the various inflected forms of a word into a single form for easy analysis.

Word segmentation divides a large piece of continuous text into distinct units. Sentence breaking involves placing sentence boundaries on a large piece of text.

Semantics, while not a fully resolved aspect in NLP, refers to the meaning that is conveyed by a text. It involves applying computer algorithms to understand the meaning and interpretation of words and how sentences are structured [7]. Some techniques in Semantic include named entity recognition (NER), word sense disambiguation, and natural language generation. NER is comparable to tagging; NER is the process of determining the parts of a text that can be identified and categorized into preset groups. Examples of such groups include names of people and names of places. Word sense disambiguation involves giving meaning to a word based on the context. Finally, natural language generation involves using databases to derive semantic intentions and convert them into human language. Evidently, NLP plays a crucial role in supporting human-machine interaction.

II. CNNs

A. *Definition*

Convolutional Neural Networks (CNNs) are typically coupled with Computer Vision. CNNs were responsible for major breakthroughs in Image Classification and are the core of most Computer Vision systems today, from Facebook's automated photo tagging to self-driving cars [8]. A convolution is best thought of as a sliding window function applied to a matrix.

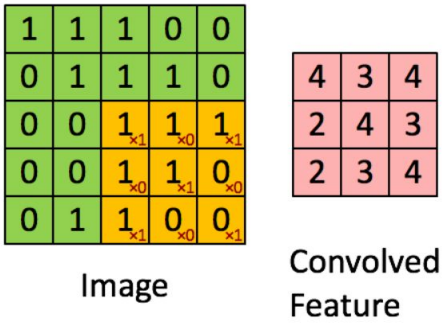

Figure 1: Convolution with 3×3 Filter [9]

Figure 1 shows a matrix on the left that can represent a black and white image. Each entry corresponds to one pixel, 0 for black and 1 for white. The sliding window is called a kernel, filter, or feature detector [8]. Here a 3×3 filter is used to multiply its values element-wise with the original matrix, then sum them up. To get the full convolution, we do this for each element by sliding the filter over the whole matrix. Some intuitive examples include blurring an image which involves averaging each pixel with its neighbor and edge detection which takes the difference between each pixel and its neighbor.

CNNs are essentially several layers of convolutions with nonlinear activation functions like ReLU (Rectified Linear Unit) or tanh applied to the results. Convolutions are used over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands like the ones shown above, and combines their results. This process can be visualized in Figure 2 below:

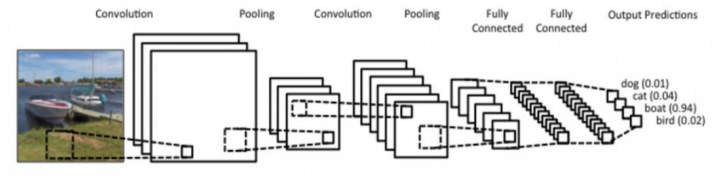

Figure 2: Image Classification in CNNs

During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform. For example, in Image Classification a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features [8]. This process of building from pixels to edges, edges to shapes, and shapes to complex objects allows compositionality.

III. CNN APPLICATIONS FOR NLP

A. *Definition*

The most natural fit for CNNs seem to be classification tasks, such as sentiment analysis, spam detection, or topic categorization [8]. Applying CNNs to NLP is quite simple. Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix. Each row of the matrix

corresponds to one token - typically a word, but it could be a character. Each row is a vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe. For example, a 10 word sentence using a 100-dimensional embedding would have a 10×100 matrix as the input. That is the "image". In NLP, these filters slide over full rows of the matrix (words). Thus, the "width" of the filters are usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical. Putting all the above together, a Convolutional Neural Network for NLP may look like this:

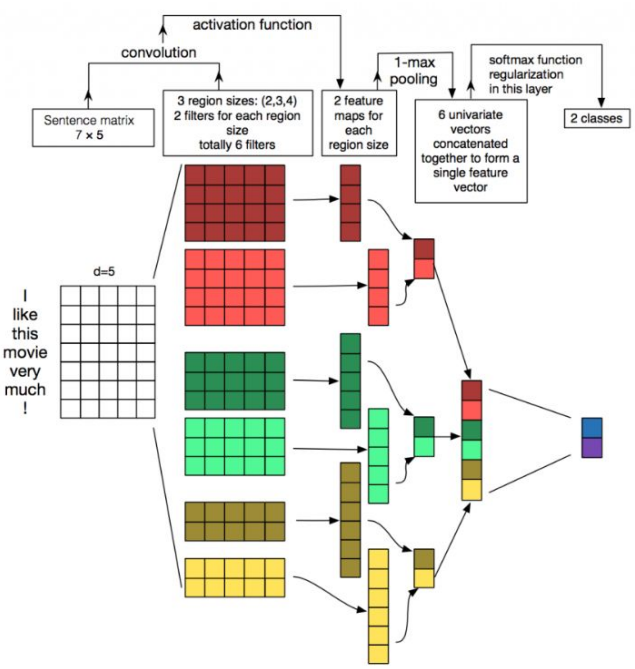

Figure 3: Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification [8]

The implementation of a CNN for text classification based on Kim Yoon's Convolutional Neural Networks for Sentence Classification follows this process. The model uses over 10,000 movie review sentences (half positive and half negative) from Rotten Tomatoes as its dataset. The dataset has a vocabulary of size around 20k. Since there's no specific train/test split, 10% of the data is used as a dev set. The first layer embeds words into low-dimensional vectors. Word vectors, or words are projected from a 1-of-V encoding (V is the vocabulary size) onto a lower dimensional vector space via a hidden layer, are essentially feature extractors that encode semantic features of words in their dimensions. In these representations, semantically close words are also close in the lower dimensional vector space. The next layer performs convolutions over the embedded word vectors using multiple filter sizes. For example, sliding over 3, 4 or 5 words at a time. Next, the result of the convolutional layer is max-pooled into a long feature vector, dropout regularization is added, and the result is classified using a softmax layer.

To simplify the model from Kim's, embeddings are learned from scratch rather than from pre-trained vectors like word2vec, L2 norm constraints (i.e., linearly scale the L2 norm of the vector to a pre-specified threshold when it exceeds the constraint) are not enforced on the weight vectors, and only one input data channel is used rather than two. Altogether, deep learning models learn these word vector representations through neural language models [4][5][6] and perform composition over the learned word vectors for classification [10].

B. Performance

However, intuition is more ambiguous in NLP applications. It made sense that neighboring pixels are likely to be semantically related, but this doesn't always apply to words. Neighboring words may not guarantee their meanings are semantically tied. However, the main argument that remains is that CNNs perform fast. Convolutions are a central part of computer graphics and implemented on a hardware level on GPUs. Additionally, CNNs are also efficient in terms of representation. With a large vocabulary, computing anything past 3-grams can quickly become expensive. Convolutional Filters learn good representations automatically, without needing to represent the whole vocabulary. Originally invented for computer vision, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in semantic parsing [5], search query retrieval [11], and other traditional NLP tasks [10].

IV. HYPERPARAMETER TUNING

A. Definition

Building a CNN architecture means that there are many hyperparameters to choose from. A hyperparameter is a parameter whose value is set

before the learning process begins, and defines the model architecture. Thus, the process of searching for the ideal model architecture is referred to as hyperparameter tuning. Some CNN hyperparameters include narrow vs. wide convolution, stride size, pooling layers, and channels.

Narrow vs. wide convolution handles edge pixels in applying the filter. For pixels along the edges that don't have complete neighboring pixels, a method called zero-padding, or assigning all elements that fall outside of the matrix as zero, allows application of the filter to every element in the input matrix, thus yielding the same-sized or larger output. Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution.

Stride size is denoted by how much to shift the filter at each step. Typically, stride sizes are 1 which cause consecutive applications of the filter to overlap. A larger stride size leads to fewer applications of the filter and a smaller output size, thus allowing a model to build more like a tree, and more reflective of a Recursive Neural Network.

Pooling layers subsample their input and are applied after convolution layers. They provide a fixed-size output matrix, which typically is required for classification. The most common way to do pooling it to apply a max operation to the result of each filter. For example, if you have 1,000 filters and you apply max pooling to each, you will get a 1000-dimensional output, regardless of the size of your filters, or the size of your input. This allows you to use variable size sentences and variable size filters, but always get the same output dimensions to feed into a classifier.

B. Results

A few results that stand out are that max-pooling always beats average pooling, the ideal filter sizes are important but task-dependent, and regularization doesn't seem to make a big difference in the NLP tasks that were considered. A caveat of this research is that all the datasets had fairly consistent document lengths, so this insignificance may apply to data that looks considerably different. Unfortunately, another downside to CNN-based models is that they require practitioners to specify the exact model architecture to be used and to set the accompanying hyperparameters. It is currently unknown how sensitive model performance is to changes in these configurations for the task of sentence classification.

Specifically, the CNN model based on Yoon's contains several hyperparameters: embedding dimension, filter sizes, number of filters, dropout keep probability, and L2 regularization lambda. To provide a point of reference for the CNN results, a report on the performance achieved using SVM for sentence classification is used as a baseline [12].

Dataset	bowSVM	wvSVM	bowwvSVM
MR	78.24	78.53	79.67
SST-1	37.92	44.34	43.15
SST-2	80.54	81.97	83.30
Subj	89.13	90.94	91.74
TREC	87.95	83.61	87.33
CR	80.21	80.79	81.31
MPQA	85.38	89.27	89.70
Opi	61.81	62.46	62.25
Irony	65.74	65.58	66.74

Table 1: Accuracy (AUC for Irony) achieved by SVM with different feature sets. **bowSVM**: uni- and bi-gram features. **wvSVM**: a naive word2vec-based representation, i.e., the average (300-dimensional) word vector for each sentence. **bowwvSVM:** concatenates bow vectors with the average word2vec representations [12].

In this model by Zhang and Wallace, nine sentence classification datasets were used; seven of which were also used by Yoon. For consistency, the model used the same pre-processing steps as Yoon's for the data. To understand the variance in performance attributable to various architecture decisions and hyperparameter settings, it is crucial to assess the variance due strictly to the parameter estimation procedure.

Description	Values
input word vectors	Google word2vec
filter region size	(3,4,5)
feature maps	100
activation function	ReLU
pooling	1-max pooling
dropout rate	0.5
$l2$ norm constraint	3

Table 2: Baseline configuration. 'feature maps' refers to the number of feature maps for each filter region size [12].

Hyperparameters were tuned via nested cross-fold validation, optimizing for accuracy (AUC for Irony) [12]. For every configuration considered, the

experiment was repeated 10 times, where each replication constituted a run of 10-fold cross-validation (CV).

B. Filter Region Size

Region size	MR
1	77.85 (77.47,77.97)
3	80.48 (80.26,80.65)
5	81.13 (80.96,81.32)
7	**81.65 (81.45,81.85)**
10	81.43 (81.28,81.75)
15	81.26 (81.01,81.43)
20	81.06 (80.87,81.30)
25	80.91 (80.73,81.10)
30	80.91 (80.72,81.05)

Table 4: Effect of single filter region size [12]

Here, the accuracy percentages of using only one filter region and 100 feature maps (as in the baseline configuration) are shown. Region sizes of 1, 3, 5, 7, 10, 15, 20, 25 and 30 are listed, and recorded are the means and ranges.

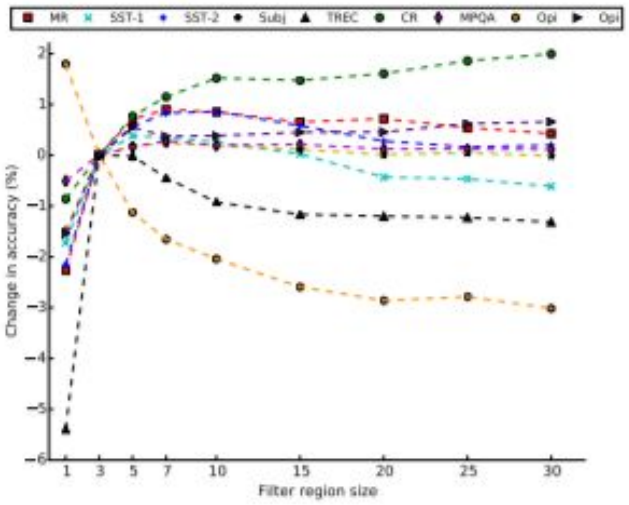

Figure 4: Effect of region size (using only one) [12]

From the results, it is clear that each dataset has its own optimal filter region size. The figure here suggests that a reasonable range for sentence classification might be from 1 to 10. However, for datasets with longer sentences, such as the CR dataset (maximum sentence length is 105, whereas the other datasets ranged from 36-56), the optimal region size may be larger.

With respect to multiple filter sizes, it was found that combining several filters with region sizes close to the optimal single region size can improve performance, but adding region sizes far from the optimal range may decrease performance.

Multiple region size	Accuracy (%)
(7)	81.65 (81.45,81.85)
(3,4,5)	81.24 (80.69, 81.56)
(4,5,6)	81.28 (81.07,81.56)
(5,6,7)	81.57 (81.31,81.80)
(7,8,9)	81.69 (81.27,81.93)
(10,11,12)	81.52 (81.27,81.87)
(11,12,13)	81.53 (81.35,81.76)
(3,4,5,6)	81.43 (81.10,81.61)
(6,7,8,9)	81.62 (81.38,81.72)
(7,7,7)	81.63 (81.33,82.08)
(7,7,7,7)	**81.73 (81.33,81.94)**

Table 3: Effect of filter region size with several region sizes on the MR dataset [12]

From Table 5, it can be shown that sets near the best single region size produce the best results. Here, sets (5,6,7), (7,8,9), and (6,7,8,9) performed the best - all hovering around 7 - i.e., the single best region size. Region size (3,4,5) performed significantly worse. Evidently, the best performing strategy is to simply use many feature maps all with region size equal to 7.

C. Feature Maps

When testing the effect of feature map hyperparameter tuning, the three default filter region sizes are held constant: 3, 4 and 5. The number of feature maps considered are: 10, 50, 100, 200, 400, 600, 1000, and 2000.

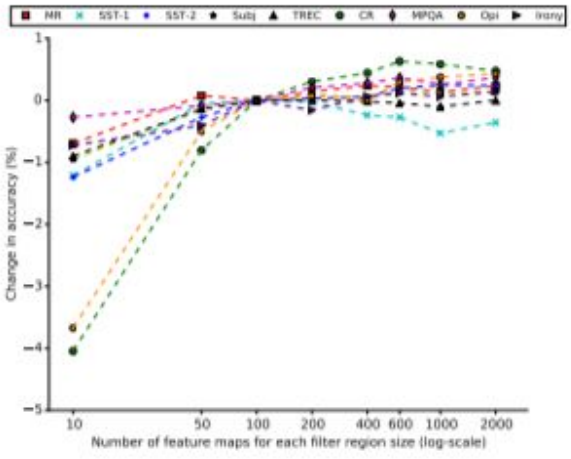

Figure 5: Effect of number of feature maps [12]

From the graph, it can be concluded that the most optimal number of feature maps for each filter region depends heavily on the dataset. Increasing the number of feature maps to over 600 yielded marginal returns

(likely due to overfitting). Thus, it is proposed that the best number of feature maps ranges from 100-600. However, a downside of this increase is an increase in training time.

V. CONCLUSIONS

After performing an empirical evaluation on the effect of varying hyperparameters in CNN architectures, investigating their impact on performance and variance over multiple runs, several conclusions can be made. The main takeaway found was that hyperparameter tuning is highly dependent on the dataset. Datasets can range in a variable number of ways and the optimal hyperparameters for one may not be the most optimal for another.

Of the hyperparameters discussed, filter region size and number of feature maps have the largest effects on performance. By altering the filter sizes to be near the optimal single filter size, the accuracy of a model can be improved. A reasonable range might be 1~10, and can be found with line-search. However, for datasets with very long sentences like CR, it may be worth exploring larger filter region sizes. To tune the multiple filter sizes, consider combining multiple filters using regions sizes near this single best size. Additionally, the optimal number of feature maps ranges from 100 to 600. When used with a small dropout rate (0.0-0.5) and a large max norm constraint, the performance of the model has been shown to increase [12].

VI. FUTURE WORK

There are several proven, researched NLP techniques that can be implemented to improve our CNN model. NLP techniques can be applied to pre-processing the dataset or during training time.

1) Removing Stopwords: Stopwords are extremely common words that add little to no value to an individual's needs in use cases like document matching [2]. Some examples of stopwords include: a, an, as, are, and at. Given their lack of value, we could eliminate them entirely from the English vocabulary when cleansing the dataset in our improved model. So consider an example tweet: The election was over. After removing the stopwords from the sentence, it should now read: election over. Notice now that the sentence is much shorter, and only contains unique words with some value to its use.

2) Lemmatizing Words: Lemmatization is the process of converting a word to its base form. In particular, lemmatization considers the context of a word, and converts it to its based form appropriately. So consider the words: election, elections, and elected. By lemmatizing the words, the base form should yield: elect. By applying this technique to our model, we could group together words of similar base form and count them as more appearances of the same word.

3) Using TF-IDF: TF-IDF (Term Frequency - Inverse Document Frequency) is a technique that can be used for stopword filtering in various fields such as classification. It is used to evaluate how important a word is to a document in a collection or corpus, which increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the collection or corpus [9]. Therefore, instead of just counting the frequency of each word, we could incorporate TF-IDF to also penalize words that appear frequently in most of the dataset.

In summary, while CNNs have proven their effectiveness in Computer Vision, their applications in NLP are proving to be also worthwhile. In the future, if given more time and resources, we would be interested in exploring some of the mentioned NLP approaches above to improve our model to get better text classification predictions.

REFERENCES

[1] “Text Classification,” MonkeyLearn, 27-Jan-2020. [Online]. Available: https://monkeylearn.com/text-classification/.

[2] A. Krizhevsky, I. Sutskever, G. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of NIPS 2012.

[3] A. Graves, A. Mohamed, G. Hinton. 2013. Speech recognition with deep recurrent neural networks. In Proceedings of ICASSP 2013.

[4] Y. Bengio, R. Ducharme, P. Vincent. 2003. Neural Probabilistic Language Model. Journal

[5] W. Yih, K. Toutanova, J. Platt, C. Meek. 2011. Learning Discriminative Projections for Text Similarity Measures. Proceedings of the Fifteenth Conference on Computational Natural Language Learning, 247–256.

[6] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS 2013.

[7] M. J. Garbade, “A Simple Introduction to Natural Language Processing,” Medium, 15-Oct-2018. [Online]. Available: https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32.

[8] D. Britz, “Understanding Convolutional Neural Networks for NLP,” WildML, 10-Jan-2016. [Online]. Available: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

[9] “Feature extraction using convolution,” Feature extraction using convolution - Ufldl. [Online]. Available: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution.

[10] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research 12:2493–2537.

[11] Y. Shen, X. He, J. Gao, L. Deng, G. Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In Proceedings of WWW 2014.

[12] Zhang, Y., & Wallace, B. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820.