

University of North Georgia

## Nighthawks Open Institutional Repository

---

Honors Theses

Honors Program

---

Spring 2018

### Densely Connected Convolutional Neural Networks for Natural Language Processing

Yannik M. Glaser

University of North Georgia, [ynglas1141@ung.edu](mailto:ynglas1141@ung.edu)

Follow this and additional works at: [https://digitalcommons.northgeorgia.edu/honors\\_theses](https://digitalcommons.northgeorgia.edu/honors_theses)



Part of the [Computer Sciences Commons](#)

---

#### Recommended Citation

Glaser, Yannik M., "Densely Connected Convolutional Neural Networks for Natural Language Processing" (2018). *Honors Theses*. 36.

[https://digitalcommons.northgeorgia.edu/honors\\_theses/36](https://digitalcommons.northgeorgia.edu/honors_theses/36)

This Honors Thesis is brought to you for free and open access by the Honors Program at Nighthawks Open Institutional Repository. It has been accepted for inclusion in Honors Theses by an authorized administrator of Nighthawks Open Institutional Repository.

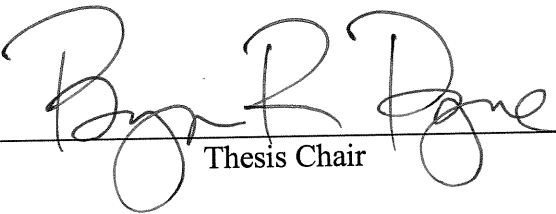
Densely Connected Convolutional Networks for  
Natural Language Processing

A Thesis Submitted to  
The Faculty of the University of North Georgia  
In Partial Fulfillment  
Of the Requirements for the Degree  
Bachelor of Science in Computer Science  
With Honors

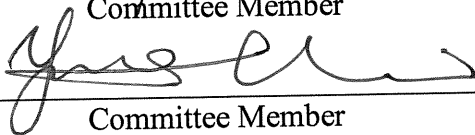
Yannik Glaser  
Spring 2018


Accepted by the Honors Faculty  
of the University of North Georgia  
in partial fulfillment of the requirements for the title of  
Honors Program Graduate

Thesis Committee:

  
Thesis Chair

  
Committee Member

  
Committee Member

  
Honors Program Director

# Densely Connected Convolutional Networks for Natural Language Processing

Yannik Glaser  
UNG  
ymglas1141@ung.edu

## Abstract

*Densely connected convolutional neural networks are currently one of the best object recognition algorithms. Given the plasticity of neural networks, the DenseNet algorithm should perform similarly in NLP tasks. In its attempt to verify whether the DenseNet algorithm can yield equally impressive results on NLP tasks, this paper has modified the DenseNet algorithm and tested it on text classification. For this purpose, three differently sized datasets have each been encoded as Tf-IDf vectors and word vectors and then the DenseNet's performance on these different feature sets was compared to more conventional methods including Naïve Bayes classifiers and other neural networks. The paper finds that DenseNets can perform on par with these algorithms but scale especially well with large datasets and semantically rich features.*

## 1 Introduction

Natural language processing overall has grown to become one of the major areas of application for deep learning algorithms next to computer vision. With the significant increase in readily available computational power, open-source libraries, as well as vast amounts of data, deep neural networks especially have excelled in a number of complex machine learning tasks. Currently, two significant trends are observable in the field of deep learning; networks are increasingly becoming deeper, as a larger depth is generally

associated with better performance, and newer architectures are seeking to increase interconnectedness to promote feature re-use and combat the vanishing gradient problem that arises with increasing depth. In line with these trends and proposed as an efficient architecture to allow for scalability and combatting the vanishing gradient problem, densely connected convolutional neural networks (DenseNets) have recently been proposed and applied to computer vision tasks with significant success [3].

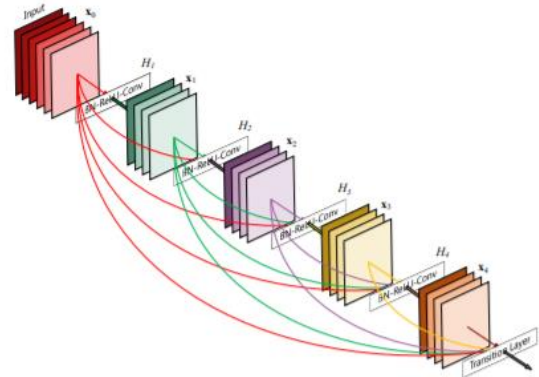
For computer vision tasks, specifically image classification, the DenseNet algorithm was able to outperform all existing architectures with relative ease, leaving a vacuum in natural language processing where a highly interconnected, scalable algorithm like the DenseNet should be. Given the fact that most neural network architectures display significant plasticity, meaning they can be utilized across different domains for very different tasks as long as they are supplied with enough data, this paper will modify the DenseNet algorithm for natural language processing. Its efficiency will be tested through text classification on various data sets. The three differently-sized datasets will be employed to allow for comprehensive testing on different feature sets. The model's performance will then be compared to more established NLP algorithms, including

simple stochastic classifiers and other deep learning algorithms, in order to draw a conclusion on the algorithm’s viability for natural language processing, specific use cases, and potential room for future studies.

## 2 Related Work

Convolutional neural networks (CNN), first introduced for computer vision almost 30 years ago in 1989 [6] and recurrent neural networks (RNN) [2], as well as the RNN’s more effective, gated variations, namely long-short term memory (LSTM) [4] cell and gated-recurrent unit (GRU) [16] based architectures, are essential building-blocks for most deep learning NLP algorithms. In a thorough analysis, [18] concluded that RNN variants are generally the more robust NLP algorithm even though the CNNs can perform on par or, for tasks depending on certain key words or key phrases, even exceed the performance of their recurrent alternatives. Of course, combinations of the two architectures are possible and actually quite popular for several tasks [7].

More recently, the exploration of different architectures has been drawn more toward the domain of increasing depth and finding different connectivity patterns [3], with Huang et al. offering a more comprehensive overview over different connectivity patterns that have led up to the particular DenseNet architecture that is the subject of this paper.

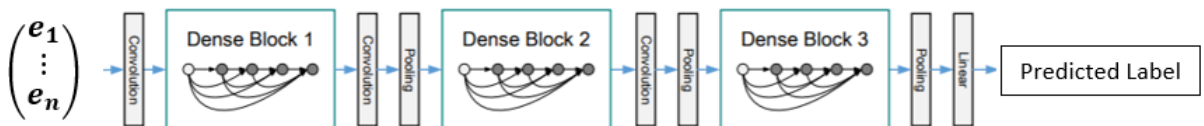


layers and a growth rate of  $k=4$ . Note how all layers are connected in a feed-forward fashion.

### 2.1 DenseNet Model

For the purpose of this study, the model will be introduced in a top-down approach with a focus on the most essential pieces of the algorithm. This paper has used the algorithm mostly in the way it was introduced by [3] and for further information the original paper by Huang et al. is a good resource. The DenseNet consists of 3 essential building blocks, transition layers for supporting scalability, input and output layers, and dense blocks, which are the most essential component of the algorithm.

The connection pattern introduced before is implemented in the form of the dense blocks, where every layer is connected to each following layer in a feedforward fashion, meaning for any given layer  $l$  the input  $x_l$  consists of a combination of the outputs of all preceding layers  $[x_0, x_1, \dots, x_{l-1}]$  through the composite function  $H_l$ . The way the feature maps are combined in  $H_l$  is through consecutively performing a batch normalization operation on the output,



**Figure 1 (SOURCE):** DenseNet with three dense blocks. The input vector gets passed through an initial convolution, the dense blocks to create the rich feature maps, and the transition layers for down-sampling, to finally produce a prediction

passing it through a rectified linear unit activation function and then through a convolutional layer with a kernel size depending on the network’s input size [3]. In the original implementation of the algorithm for computer vision, a three-by-three kernel gets employed, however, for natural language processing, this is the only major modification to the algorithm that must be made. Convolutional neural networks must be allowed to generalize language understanding tasks which is why large kernel-sizes are generally a better choice, Therefore, in this implementation of the DenseNet algorithm, a kernel size of 25% of the input has yielded the best results, which is why it gets employed instead of the smaller three-step kernel.

As one can easily see, feature maps produced by the large connectivity in the dense blocks can quickly explode in size and make a straight forward implementation of this connection pattern across an entire network completely unfeasible, which is why a down-sampling mechanism is implemented between dense blocks in the form of transition layers. Each of these layers, again, begins by applying a batch normalization across its input, passing it through a 1-step convolutional layer, and finally down sampling through a two-step max-pooling function, reducing the size of the output vector by half and making the connectivity pattern viable by reducing the feature map size after each dense block [3].

The size of the output vectors produced by each respective dense block depends largely on a parameter in the network termed the growth rate ( $k$ ) [3]. The growth rate essentially describes the number of filters per convolutional layer in the dense block, indicating how many

feature maps will be appended to the ‘global’ feature map, which could be regarded as the “global state of the network” [3]. Relatively narrow layers with a  $k$  larger or equal to 12 are shown to be sufficient for achieving state of the art results while also being computationally viable.

### 3 Experiment Details

For the purpose of evaluating the DenseNet’s viability for natural language processing, it will be utilized for the task of text classification. Text classification can be regarded as the NLP-equivalent to the computer vision task of image classification – the original use case for the DenseNet.

#### 3.1 Datasets

In order to provide comprehensive evaluation of this model’s capabilities, three different datasets will be used. The first dataset is extracted from the online forum StackOverflow [12]; the dataset consists of the title of 20,000 questions asked in that forum which are classified into 20 different question categories. The data is relatively short with each text portion being shorter in length than an average sentence. The next dataset is the popular Reuters-21578, which has been in use since 2004 [14] and was extracted directly from the python natural language toolkit package, yielding 13328 examples for 90 categories. The examples are significantly longer than the questions in the previous dataset, being comprised of the header of the article as well as the article’s body. The last dataset that is utilized is the 5-core subset of the amazon reviews dataset by [8]. This is by far the most extensive dataset, featuring 18,365,245 sample reviews for items of 24 different categories. The reviews consist of several sentences on average

and no title. All data sets are split into 60 percent training data and 40 percent testing data; better performance can likely be achieved by splitting the data less drastically, however, since the goal of this paper is to evaluate the performance of this algorithm under realistic circumstances, this split allows to draw better conclusions about the algorithm’s ability to generalize well with a limited amount of training data.

### 3.2 Feature Sets

An important difference between natural language processing and computer vision is the need for feature engineering. Long before deep learning has taken over NLP, computational linguistics was very preoccupied with extracting specific features from sets to help algorithms perform their specific task; however, specifically deep learning algorithms are very adept at processing relatively raw data very efficiently and extracting important features during the training process. For the purpose of this paper, two specific feature sets will be utilized that are expected to improve performance over just using raw text data as the input, but do not require extensive preprocessing of the data or specific feature engineering.

The first set of features will be a 1024-dimensional text frequency – inverse document frequency (Tf-IDf) vector. Tf-IDf vectors are very commonly used in natural language processing as a simple way to represent documents that are part of a larger corpus. In this example, a dictionary of term-frequencies is created by iterating over the entire training corpus and then applying sublinear scaling to the resulting term frequency:

$$tf_{t,d} = \log(1 + tf_{t,d}) \quad [17]$$

The full Tf-IDf value is then calculated by getting the inverse document frequency for each term:

$$idf_t = \log \frac{N}{1 + |\{d \in D: t \in d\}|} \quad [17]$$

Where N is the number of documents in the given corpus and one is added in the denominator to avoid a division by zero for terms that don’t occur in a given document. For these experiments, a threshold document frequency of 0.3 was utilized; given the limited length of the vector and due to the fact that no other preprocessing of the input was done (for instance all stop-words were still included) the low maximum document frequency serves to filter out a lot of the noise that could potentially be created and distort the limited feature set. The low document frequency combined with the overall small vocabulary allowed evidently causes most Tf-IDf vectors to be very sparse, making them a sub-optimal feature set for deep neural networks of any kind. Nevertheless, they are crucial for the evaluation of the DenseNet algorithm, due to their ease of use and how commonly they are used in many NLP algorithm implementations.

The second feature set is intended to complement specifically the neural networks. Word embeddings as proposed by [9] have seen a tremendous increase in popularity because of their ability to better capture and represent semantic information [11] which can easily be leveraged by deep learning algorithms to better extract relevant features. In this paper, the pre-trained word-2-vec model as proposed in [10] will be utilized. The model has pre-trained 300-dimensional embeddings for around 3 million words in the English language and has been

trained with the skip-gram algorithm on the widely used Wikipedia dump corpus. Since the model does not exclude some stop words and cannot possibly contain every word occurring in the training corpora, the mechanism for accounting for words/character sequences outside of the vocabulary of the model is to simply utilize trained character-vectors and average them for the given character sequence. The embedding for any given document in any given corpus is then created by simply finding the average vector for the entire document. While this may raise concerns about losing semantic information, especially for longer documents, due to computational limitations this compromise has to be made; furthermore, performance actually does not suffer significantly in most cases.

### 3.3 Evaluated Algorithms

To best evaluate the DenseNet, a wide variety of algorithms is used and trained on the same feature sets to see how the performance of the new algorithm measures up with existing classifiers. The first two classifiers have been a staple for text classification long before the resurgence of deep learning algorithms and are still widely used due to how easily they scale with larger datasets and how computationally cheap they are. Furthermore, these models are usually also more proficient at handling the sparse feature set provided by the Tf-IDf vectors and require less data to train.

The Multinomial Naïve Bayes classifier computes the prior likelihood for each class:

$$P(c) = \frac{N}{N_c}$$

Where N is the number of documents in the corpus and  $N_c$  is the number of

documents of the given class  $c$ . Furthermore, it utilized conditional probabilities calculated based on feature frequencies:

$$P(w|c) = \frac{\text{count}(w, c) + 1}{\text{count}(c) + |V|}$$

Where V is the vocabulary size of the corpus and plus-one smoothing is applied to the numerator. A prediction is based on a simple argmax operation over the product of the prior and the likelihood of a class given all features in the document for all possible classes:

$$y = \underset{k \in \{1, \dots, K\}}{\text{argmax}} (P_{c_K}) \prod_{i=1}^n p(w_i | c_K)$$

Where y is the predicted label, K is the number of classes, and i is the number of features in the given document [13].

The second algorithm is a support vector machine. On a high level, the support vector machine algorithm attempts to plot a hyperplane of n-1 dimensions to separate instances of all n classes, so depending on



Layer	Output Dimensions	Kernel Size   Filters
Convolution	300 x 12	InputD/4   16
DenseBlock I	300 x 124	InputD/4   16
Transition I	150 x 124	
DenseBlock II	150 x 236	InputD/4   16
Transition II	75 x 236	
DenseBlock III	75 x 348	InputD/4   16
Transition III	37 x 348	
DenseBlock IV	37 x 460	InputD/4   16
Global Average Pooling	1 x 460	
Fully Connected	1 x 1000	

**Table 2:** Architecture of the DenseNet used for the Reuters and StackOverflow data set. Growth rate of the network is  $k = 16$  over 4 dense blocks.

where a given document falls relative to that hyperplane, it gets classified into the associated class [1]. This particular implementation of the support vector machine utilizes the popular hinge loss as a cost function:

$$c(x, y, y') = (1 - y * y')$$

For  $x$  being the input features,  $y$  being the actual label associated with the input, and  $y'$  being the label predicted by the classifier. The loss is set to zero if the prediction is correct, or more precisely for any term  $c < 0$ .

Then, for each batch of input, the algorithm tries to optimize for the following objective function:

$$\lambda \|w\|^2 + \sum_{i=1}^n (1 - c(x_i, y_i, y'_i))$$

Where  $\lambda$  is a regularization term for an L2 regularizer, which is usually used for support vector machines. Since this algorithm will use a stochastic gradient descent learning function, the weights are updated according to:

$$w = w + \eta((-y * x - 2\lambda w))$$

Note that  $-y*x$  will be 0 for any correct classification. Thus, for a correct classification, the model will only be

updated in accordance with the product of the learning rate and the regularizer term. In this SVM implementation, the learning rate is adaptive and decreasing over time. For both classifiers, an implementation from the python machine learning library scikit-learn [15] is used.

In addition to this, the DenseNet will be compared to another neural network implementation for text classification (table 1). This neural network will be a simple combination of an encoding step consisting of several convolutional and max-pooling layers followed by an LSTM layer and finally a fully connected layer

Lastly, the specific DenseNet implementation is detailed in table 2. The DenseNet used for most tests consists of 4 dense blocks and a growth rate of  $k=16$ . The kernel size is chosen depending on the feature set, thus kernels are smaller for the word-embedding feature set as compared to the Tf-IDf vectors, allowing for an equals degree of generalization across the feature set. It is important to note that this specific network size is chosen due to hardware limitations. Due to the same limitations, a decrease in size for the network is necessary for the larger Amazon reviews dataset; the network for that dataset has its growth rate decreased

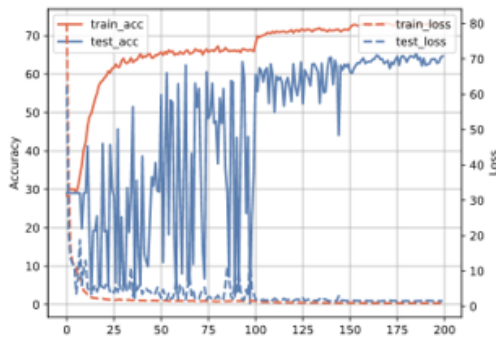
to  $k = 10$ . Categorical cross-entropy is again the loss function.

### 3.4 Training

All classifiers are trained with 512-sample batches. The simple classifiers are trained once with the entire dataset, configured as detailed above.

The convolutional LSTM neural network model is trained for 100 epochs on batches of 512 samples and categorical cross-entropy is employed as a loss function for training purposes. The network employs the adam-optimizer with an initial learning rate of 0.01, which is decreased after 50 and 75 epochs by a factor of 0.1 respectively.

The DenseNets are trained for 200 epochs with an initial learning rate of 0.1, decreasing after 100 and 150 epochs by a factor of 0.1. The networks have yielded the best performance on a batch size of 512 and all parameters except for kernel size are similar across the Reuters and StackOverflow datasets, with necessary changes for the Amazon Reviews dataset due to GPU memory and performance limitations.



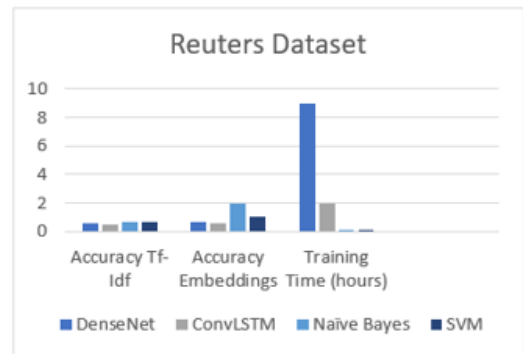
**Figure 4:** Training on Reuters dataset with word embeddings. Graph is representative of training process on all other datasets, as discrepancies in the training process have been negligible.

## 4.0 Results

The accuracy of the models is detailed is evaluated in accuracy across the entire testing dataset.

Dataset/Classifier	Reuters	Stack Overflow	Amazon Reviews
Naïve Bayes	0.68	0.77	0.73
Support Vector Machine	0.67 0.63	0.83 0.74	0.78 0.74
CLSTM	0.51 0.55	0.6 0.63	0.76 0.83
DenseNet	0.56 0.64	0.68 0.69	- 0.93

**Table 3:** Testing results. Accuracies in black are on the Tf-IDf feature set, while accuracies in blue are on the embedding vectors. The Tf-IDf feature set exceeded GPU memory limits for the Amazon Review test set.



**Figure 3:** Performance on the Reuters dataset in terms of accuracy and time needed to train the algorithm.

## 4.1 Discussion

The results in table 3 are very telling about the specific use cases that seem appropriate for the DenseNet, and where it fails to perform. Across all test cases, the DenseNet is ahead of the simpler Convolutional LSTM implementation, something that is to be expected given the complexity of the two algorithms due to the DenseNet's ability to scale to far greater depths than the other neural network implementation.

However, it must also be noted that while both are clearly outperformed by the Naïve Bayes classifier and the Support Vector machine in terms of training time, the ConvLSTM does exhibit better training times than the DenseNet, again, due to its decrease in complexity.

For both the Reuters and the StackOverflow dataset, the highly complex neural networks are outperformed by the significantly simpler classifiers, which is not surprising given some of the characteristics of the datasets. Both are relatively short with only a limited number of samples for their classes. For the StackOverflow dataset, the individual pieces of text are extremely short, being only headers to forum questions, and thus, both the Tf-IDf vector and the embedding vector likely exhibit little noise despite the lack of preprocessing of the data, since stop words and other irrelevant data likely play less of a role.

One can also clearly see how the different algorithms are more efficient on different sets of features. The Naïve Bayes Classifier obviously only works on Tf-IDf vectors, which is why a comparison there is impossible, however, the support vector machine has consistently better results with Tf-IDf vectors, while the neural networks struggle a little more with the sparse nature of that feature set and excel more at interpreting the semantically rich word embeddings.

Lastly, the Amazon Reviews dataset sticks out as the singular saving grace for the DenseNet's viability as a text classification, and by extension, potentially viable overall NLP algorithm. While SVMs and Naïve Bayes classifiers are extremely proficient at classifying with smaller training datasets, neural

networks excel as the dataset size increases; with the Amazon Reviews dataset covering over 18 million examples for just 24 classes, Unfortunately, even with a 1024-dimensional Tf-IDf vector and a down-scaled DenseNet, GPU memory limits were exceeded which makes this comparison impossible. However, it is quite likely that the Tf-IDf vector would not have been a better feature set than the word embeddings, which is also indicated by the significant different in performance displayed by the ConvLSTM algorithm between the two feature sets.

These results reveal the use-case that would likely fit the DenseNet best. As most deep learning algorithms, it is frankly inferior to many other, simpler algorithms for small to medium sized datasets in most cases. However, for large scale applications with access to large datasets, the DenseNet not only performs other machine learning classifiers, but also other, fairly sophisticated neural networks, indicating that its potential in the NLP domain might be proportional to the access to labeled training data.

## 5 Conclusion and Future Work

“Premature optimization is the root of all evil” [5]

The above quote is often used to illustrate the point, that when trying to solve a problem programmatically, miniscule performance gains at the cost of simplicity should be avoided. This point is especially relevant to the results found in this paper.

Since access to hardware with great computing power has become more abundant, deep learning applications have been one of the largest areas of

research within the machine learning domain. Nevertheless, it is important to not lose sight of other algorithms that may be a better fit for solving the problem at hand. As this paper has shown the DenseNet is an extremely sophisticated algorithm with the ability to outperform both general machine learning algorithms as well as other neural network architectures; however, the results also reveal the algorithms dependency on vast amounts of training data. The outcomes of the tests conducted in this paper have revealed that traditional machine learning algorithms have rightfully kept their place in natural language processing applications.

It is important to note some of the shortcoming of the methodology in this paper that can hopefully be compensated for in future research. Due to hardware restrictions, the DenseNet could not be scaled to it's full potential, as compared to some of the implementations in the original paper [3] the algorithm was kept at only a medium growth rate and relatively low depth, thus it is reasonable to conclude that potential performance gains would have been possible with access to better hardware, however, it is unlikely that that would have impacted the conclusion of this paper.

For future research, exploring the DenseNets capabilities in other NLP applications is imperative. Due to its highly interconnected nature as well as the findings in this paper, employing this algorithm to serve as an encoding mechanism could be an area of interest. Additionally, other areas of NLP, for instance machine comprehension, are extremely sophisticated and complicated problems that often prove too difficult for simple linear algorithms but could profit from employing the DenseNet algorithm

instead of traditional, simpler convolutions.

**Acknowledgements.** I would like to thank Dr. Bryson Payne, for mentorship throughout my undergraduate studies and overseeing the research process for this thesis. Additionally, I would like to thank the faculty members for volunteering to serve as committee members and providing me with valuable feedback. Finally, I would like to express my gratitude to the honors program, specifically Dr. Tanya Bennett and Dr. Stephen Smith, for supporting me during my undergraduate studies at the University of North Georgia and providing me with opportunities and guidance that have led to a successful conclusion of my undergraduate degree.

## References

- [1] Boser, B. E., Guyon, I. M., Vapnik, V. N. A Training Algorithm for Optimal Margin Classifiers. <https://dl.acm.org/citation.cfm?id=130401>. Accessed 25 February 2018.
- [2] Elman, J. L. Finding Structure in Time. *Cognitive Science* 14, 179-211.
- [3] Huang, G., Liu, Z., Van der Maaten, L. And Weinberger, K. Q. Densely Connected Convolutional Networks. <https://research.fb.com/publications/densely-connected-convolutional-networks/>. Accessed 1 October 2017.
- [4] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation* 9, 8, 1735-1780.
- [5] Knuth, D. E. Structured Programming With go to Statements. <http://sbel.wisc.edu/Courses/ME964/Literature/knuthProgramming1974.pdf>. Accessed 27 February 2018.
- [6] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. <http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf>. Accessed 25 February 2018.
- [7] Lee, J. Y., Démoncourt, F. Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks. <https://arxiv.org/pdf/1603.03827.pdf>. Accessed 12 February 2018.
- [8] McAuley, J (2016). *Amazon product data*[json]. <http://jmcauley.ucsd.edu/data/amazon/>.
- [9] Mikolov, T., Chen, T., Corrado, G., Dean, J. Efficient Estimation of Word Representations in Vector Space. <https://arxiv.org/pdf/1301.3781.pdf>.
- [10] Mikolov, T., Sutskever, B., Chen, K., Corrado, G., Dean, J. Distributed Representations of Words and Phrases and their Compositionality. <https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>. Accessed 25 February 2018.
- [11] Mikolov, T., Yih, W., Zeig, G. Linguistic Regularities in Continuous Space Word Representations. <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/rvecs.pdf>. Accessed 25 February 2018.
- [12] NAACL VSM-NLP workshop (2015). *Short Text Clustering via Convolutional Neural Networks*[csv]. Retrieved from <https://github.com/jacoxu/StackOverflow>.
- [13] *Naive Bayes text classification*. Retrieved from <https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html>.
- [14] Reuters-21578, <https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection>. Distribution 1.0. Accessed 25 February 2018.
- [15] *SGD Classifier*. Retrieved from [http://scikit-learn.org/stable/modules/linear\\_model.html#stochastic-gradient-descent-sgd](http://scikit-learn.org/stable/modules/linear_model.html#stochastic-gradient-descent-sgd).
- [16] Tang, D., Qin, B., Liu, T. Document Modeling with Gated Recurrent Neural Network for Sentiment Classification. <http://aclweb.org/anthology/D15-1167>. Accessed 25 February 2018.
- [17] *Tf-IDf Weighting*. Retrieved from <https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html>.
- [18] Yin, W., Kann, K., Yu, M., and Schütze, H. Comparative Study of CNN and RNN for Natural Language Processing. <https://arxiv.org/pdf/1702.01923.pdf>. Accessed 25 February 2018.