Deep Pyramid Convolutional Neural Networks for Text Categorization

Rie Johnson
RJ Research Consulting
Tarrytown, NY, USA
riejohnson@gmail.com

Tong Zhang
Tencent AI Lab
Shenzhen, China
bradymzhang@tencent.com

Abstract

This paper proposes a *low-complexity* word-level deep convolutional neural network (CNN) architecture for text categorization that can efficiently represent *long-range* associations in text. In the literature, several deep and complex neural networks have been proposed for this task, assuming availability of relatively large amounts of training data. However, the associated computational complexity increases as the networks go deeper, which poses serious challenges in practical applications. Moreover, it was shown recently that shallow word-level CNNs are more accurate and much faster than the state-of-the-art very deep nets such as character-level CNNs even in the setting of large training data. Motivated by these findings, we carefully studied deepening of word-level CNNs to capture global representations of text, and found a simple network architecture with which the best accuracy can be obtained by increasing the network depth without increasing computational cost by much. We call it *deep pyramid CNN*. The proposed model with 15 weight layers outperforms the previous best models on six benchmark datasets for sentiment classification and topic categorization.

1 Introduction

Text categorization is an important task whose applications include spam detection, sentiment classification, and topic classification. In recent years, neural networks that can make use of word order have been shown to be effective for text categorization. While simple and shallow *convolutional neural networks (CNNs)* (Kim, 2014; Johnson and Zhang, 2015a) were proposed for this task earlier, more recently, deep and more complex neural networks have also been studied, assuming availability of relatively large amounts of training data (e.g., one million documents). Examples are deep character-level CNNs (Zhang et al., 2015; Conneau et al., 2016), a complex combination of CNNs and *recurrent neural networks (RNNs)* (Tang et al., 2015), and RNNs in a word-sentence hierarchy (Yang et al., 2016).

A CNN is a feedforward network with convolution layers interleaved with pooling layers. Essentially, a convolution layer converts to a vector every small patch of data (either the original data such as text or image or the output of the previous layer) at every location (e.g., 3-word windows around every word), which can be processed in parallel. By contrast, an RNN has connections that form a cycle. In its typical application to text, a recurrent unit takes words one by one as well as its own output on the previous word, which is parallel-processing unfriendly. While both CNNs and RNNs can take advantage of word order, the simple nature and parallel-processing friendliness of CNNs make them attractive particularly when large training data causes computational challenges.

There have been several recent studies of CNN for text categorization in the large training data setting. For example, in (Conneau et al., 2016), very deep 32-layer character-level CNNs were shown to outperform deep 9-layer character-level CNNs of (Zhang et al., 2015). However, in (Johnson and Zhang, 2016), very shallow 1-layer word-level CNNs were shown to be more accurate and much faster than the very deep character-level CNNs of (Conneau et al., 2016). Although character-level approaches have merit in not having to deal with millions of distinct words, shallow word-level CNNs turned out to be superior even

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, pages 562–570

https://doi.org/10.18653/v1/P17-1052

when used with only a manageable number (30K) of the most frequent words. This demonstrates the basic fact – knowledge of *word* leads to a powerful representation. These results motivate us to pursue an effective and efficient design of *deep word-level* CNNs for text categorization. Note, however, that it is not as simple as merely replacing characters with words in character-level CNNs; doing so rather degraded accuracy in (Zhang et al., 2015).

We carefully studied deepening of word-level CNNs in the large-data setting and found a deep but low-complexity network architecture with which the best accuracy can be obtained by increasing the depth but not the order of computation time – the total computation time is bounded by a constant. We call it *deep pyramid CNN (DPCNN)*, as the computation time per layer decreases exponentially in a '*pyramid* shape'. After converting discrete text to continuous representation, the DPCNN architecture simply alternates a convolution block and a downsampling layer over and over[1], leading to a deep network in which internal data size (as well as per-layer computation) shrinks in a *pyramid* shape. The network depth can be treated as a meta-parameter. The computational complexity of this network is bounded to be no more than twice that of one convolution block. At the same time, as described later, the 'pyramid' enables efficient discovery of long-range associations in the text (and so more global information), as the network is deepened. This is why DPCNN can achieve better accuracy than the shallow CNN mentioned above (hereafter *ShallowCNN*), which can use only short-range associations. Moreover, DPCNN can be regarded as a deep extension of ShallowCNN, which we proposed in (Johnson and Zhang, 2015b) and later tested with large datasets in (Johnson and Zhang, 2016).

We show that DPCNN with 15 weight layers outperforms the previous best models on six benchmark datasets for sentiment classification and topic classification.

2 Word-level deep pyramid CNN (DPCNN) for text categorization

Overview of DPCNN: DPCNN is illustrated in Figure 1a. The first layer performs *text region embedding*, which generalizes commonly used *word embedding* to the embedding of text regions covering one or more words. It is followed by stacking of *convolution blocks* (two convolution layers and a shortcut) interleaved with pooling layers with stride 2 for downsampling. The final pooling layer aggregates internal data for each document into one vector. We use max pooling for all pooling layers. The key features of DPCNN are as follows.

- Downsampling *without* increasing the *number of feature maps* (dimensionality of layer output, 250 in Figure 1a). Downsampling enables efficient representation of long-range associations (and so more global information) in the text. By keeping the same number of feature maps, every 2-stride downsampling reduces the per-block computation by half and thus the total computation time is bounded by a constant.

- Shortcut connections with pre-activation and identity mapping (He et al., 2016) for enabling training of deep networks.

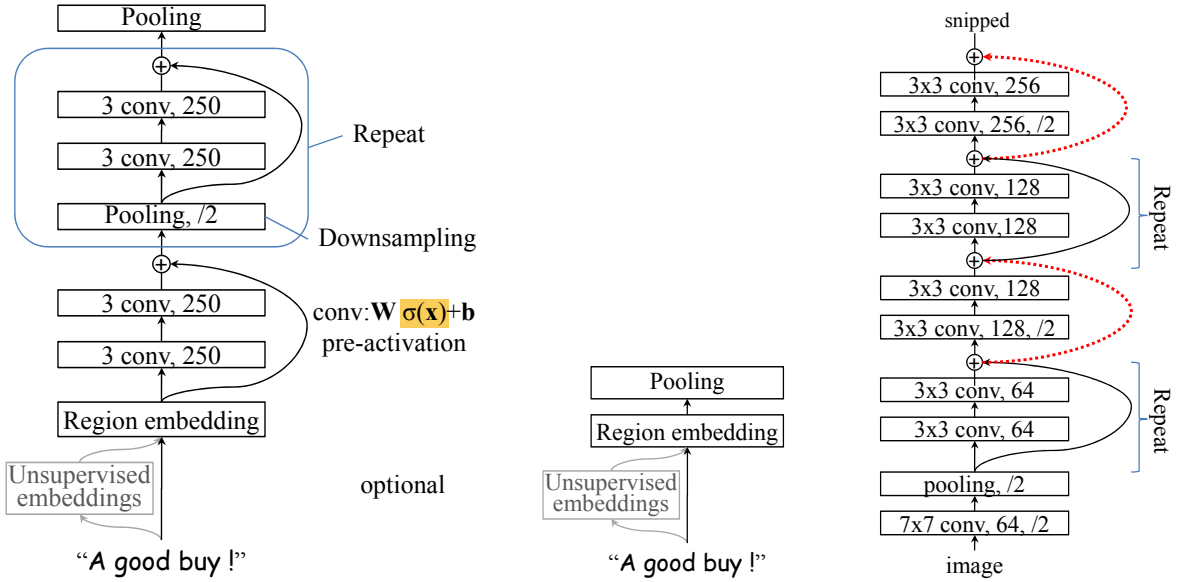
- Text region embedding enhanced with *unsupervised embeddings* (embeddings trained in an unsupervised manner) (Johnson and Zhang, 2015b) for improving accuracy.

2.1 Network architecture

Downsampling with the number of feature maps fixed After each convolution block, we perform max-pooling with size 3 and stride 2. That is, the pooling layer produces a new internal representation of a document by taking the component-wise maximum over 3 contiguous internal vectors, representing 3 overlapping text regions, but it does this only for every other possible triplet (stride 2) instead of all the possible triplets (stride 1). This 2-stride downsampling reduces the size of the internal representation of each document by half.

A number of models (Simonyan and Zisserman, 2015; He et al., 2015, 2016; Conneau et al., 2016) increase the number of feature maps whenever downsampling is performed, causing the total computational complexity to be a function of the depth. In contrast, we fix the number of feature maps, as we found that increasing the number of feature maps only does harm – increasing computation time substantially without accuracy improvement, as shown later in the experiments.

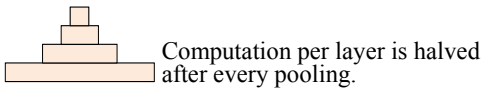
[1]Previous deep CNNs (either on image or text) tend to be more complex and irregular, having occasional increase of the number of feature maps.



(a) Our proposed model DPCNN (b) cf. ShallowCNN [JZ15b] (c) cf. ResNet for image [HZRS15]

Figure 1: (a) Our proposed model DPCNN. (b,c) Previous models for comparison. $\oplus$ indicates addition. The dotted red shortcuts in (c) perform dimension matching. DPCNN is dimension-matching free.

With the number of feature maps fixed, the computation time for each convolution layer is halved (as the data size is halved) whenever 2-stride downsampling is performed, thus, forming a '*pyramid*'.



Therefore, with DPCNNs, the total computation time is bounded by a constant – twice the computation time of a single block, which makes our deep pyramid networks computationally attractive.

In addition, downsampling with stride 2 essentially doubles the effective coverage (i.e., coverage in the original document) of the convolution kernel; therefore, after going through downsampling L times, associations among words within a distance in the order of 2^L can be represented. Thus, deep pyramid CNN is computationally efficient for representing long-range associations and so more global information.

Shortcut connections with pre-activation To enable training of deep networks, we use additive shortcut connections with identity mapping, which can be written as $\mathbf{z} + f(\mathbf{z})$ where f represents the skipped layers (He et al., 2016). In DPCNN, the skipped layers $f(\mathbf{z})$ are two convolution layers with *pre-activation*. Here, pre-activation refers to activation being done *before* weighting instead of *after* as is typically done. That is, in the convolution layer of DPCNN, $\mathbf{W}\sigma(\mathbf{x}) + \mathbf{b}$ is computed at every location of each document where a column vector $\mathbf{x}$ represents a small region (overlapping with each other) of input at each location, $\sigma(\cdot)$ is a component-wise nonlinear activation, and weights $\mathbf{W}$ and biases $\mathbf{b}$ (unique to each layer) are the parameters to be trained. The number of $\mathbf{W}$'s rows is the *number of feature maps* (also called the *number of filters* (He et al., 2015)) of this layer. We set activation $\sigma(\cdot)$ to the rectifier $\sigma(x) = \max(x, 0)$. In our implementation, we fixed the number of feature maps to 250 and the kernel size (the size of the small region covered by $\mathbf{x}$) to 3, as shown in Figure 1a.

With pre-activation, it is the results of linear weighting $(\mathbf{W}\sigma(\mathbf{x}) + \mathbf{b})$ that travel through the shortcut, and what is added to them at a $\oplus$ (Figure 1a) is also the results of linear weighting, instead of the results of nonlinear activation ($\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$). Intuitively, such 'linearity' eases training of deep networks, similar to the role of *constant error carousels* in LSTM (Hochreiter and Schmidhuber, 1997). We empirically observed that pre-activation indeed outperformed 'post-activation', which is in line with the image results (He et al., 2016).

No need for dimension matching Although the shortcut with pre-activation was adopted from the *improved ResNet* of (He et al., 2016), our model is simpler than ResNet (Figure 1c), as all the

shortcuts are exactly simple *identity mapping* (i.e., passing data exactly as it is) without any complication for dimension matching. When a shortcut meets the 'main street', the data from two paths need to have the same dimensionality so that they can be added; therefore, if a shortcut skips a layer that changes the dimensionality, e.g., by downsampling or by use of a different number of feature maps, then a shortcut must perform dimension matching. Dimension matching for increased number of feature maps, in particular, is typically done by projection, introducing more weight parameters to be trained. We eliminate the complication of dimension matching by not letting any shortcut skip a downsampling layer, and by fixing the number of feature maps throughout the network. The latter also substantially saves computation time as mentioned above, and we will show later in our experiments that on our tasks, we do not sacrifice anything for such a substantial efficiency gain.

2.2 Text region embedding

A CNN for text categorization typically starts with converting each word in the text to a word vector (word embedding). We take a more general viewpoint as in (Johnson and Zhang, 2015b) and consider *text region embedding* – embedding of a region of text covering one or more words.

Basic region embedding We start with the basic setting where there is no unsupervised embedding. In the region embedding layer we compute $\mathbf{W}\mathbf{x} + \mathbf{b}$ for each word of a document where input $\mathbf{x}$ represents a k-word region (i.e., window) around the word in some straightforward manner, and weights $\mathbf{W}$ and bias $\mathbf{b}$ are trained with the parameters of other layers. Activation is delayed to the pre-activation of the next layer. Now let v be the size of vocabulary, and let us consider the following three types of straightforward representation of a k-word region for $\mathbf{x}$: (1) *sequential input*: the kv-dimensional concatenation of k one-hot vectors; (2) *bow input*: a v-dimensional bag-of-word (bow) vector; and (3) *bag-of-n-gram input*: e.g., a bag of word uni, bi, and trigrams contained in the region. Setting the region size $k = 1$, they all become word embedding.

A region embedding layer with the sequential input is equivalent to a convolution layer applied to a sequence of one-hot vectors representing a document, and this viewpoint was taken to describe the first layer of ShallowCNN in (Johnson and Zhang, 2015a,b). From the region embedding viewpoint, ShallowCNN is DPCNN's special case in which a region embedding layer is directly followed by the final pooling layer (Figure 1b).

A region embedding layer with region size $k > 1$ seeks to capture more complex concepts than single words in *one* weight layer, whereas a network with word embedding uses *multiple* weight layers to do this, e.g., word embedding followed by a convolution layer. In general, having fewer layers has a practical advantage of easier optimization. Beyond that, the optimum input type and the optimum region size can only be determined empirically. Our preliminary experiments indicated that when used with DPCNN (but not ShallowCNN), the sequential input has no advantage over the bow input – comparable accuracy with k times more weight parameters; therefore, we excluded the sequential input from our experiments[2]. The n-gram input turned out to be prone to overfitting in the supervised setting, likely due to its high representation power, but it is very useful as the input to unsupervised embeddings, which we discuss next.

Enhancing region embedding with unsupervised embeddings In (Johnson and Zhang, 2015b, 2016), it was shown that accuracy was substantially improved by extending ShallowCNN with unsupervised embeddings obtained by *tv-embedding* training ('tv' stands for *two views*). We found that accuracy of DPCNN can also be improved in this manner. Below we briefly review tv-embedding training and then describe how we use the resulting unsupervised embeddings with DPCNN.

The tv-embedding training requires two views. For text categorization, we define a region of text as view-1 and its adjacent regions as view-2. Then using unlabeled data, we train a neural network of one hidden layer with an artificial task of predicting view-2 from view-1. The obtained hidden layer, which is an embedding function that takes view-1 as input, serves as an unsupervised embedding function in the model for text categorization. In (Johnson and Zhang, 2015b), we showed theoretical conditions on views and labels under which

[2]This differs from ShallowCNN where the sequential input is often superior to bow input. We conjecture that when bow input is used in DPCNN, convolution layers following region embedding make up for the loss of local word order caused by bow input, as they use word order.

	AG	Sogou	Dbpedia	Yelp.p	Yelp.f	Yahoo	Ama.f	Ama.p
# of training documents	120K	450K	560K	560K	650K	1.4M	3M	3.6M
# of test documents	7.6K	60K	70K	38K	50K	60K	650K	400K
# of classes	4	5	14	2	5	10	5	2
Average #words	45	578	55	153	155	112	93	91

Table 1: Data. Note that Yelp.f is a balanced subset of Yelp 2015. The results on these two datasets are not comparable.

unsupervised embeddings obtained this way are useful for classification.

For use with DPCNN, we train several unsupervised embeddings in this manner, which differ from one another in the region size and the vector representations of view-1 (input region) so that we can benefit from diversity. The region embedding layer of DPCNN computes $\mathbf{W}\mathbf{x} + \sum_{u \in U} \mathbf{W}^{(u)}\mathbf{z}^{(u)} + \mathbf{b}$, where $\mathbf{x}$ is the discrete input as in the basic region embedding, and $\mathbf{z}^{(u)}$ is the output of an unsupervised embedding function indexed by u. We will show below that use of unsupervised embeddings in this way consistently improves the accuracy of DPCNN.

3 Experiments

We report the experiments with DPCNNs in comparison with previous models and alternatives. The code is publicly available on the internet.

3.1 Experimental setup

Data and data preprocessing To facilitate comparisons with previous results, we used the eight datasets compiled by Zhang et al. (2015), summarized in Table 1. AG and Sogou are news. Dbpedia is an ontology. Yahoo consists of questions and answers from the 'Yahoo! Answers' website. Yelp and Amazon ('Ama') are reviews where '.p' (polarity) in the names indicates that labels are binary (positive/negative), and '.f' (full) indicates that labels are the number of stars. Sogou is in Romanized Chinese, and the others are in English. Classes are balanced on all the datasets. Data preprocessing was done as in (Johnson and Zhang, 2016). That is, upper-case letters were converted to lower-case letters. Unlike (Kim, 2014; Zhang et al., 2015; Conneau et al., 2016), variable-sized documents were handled as variable-sized without any shortening or padding; however, the vocabulary size was limited to 30K words. For example, as also mentioned in (Johnson and Zhang, 2016), the complete vocabulary of the Ama.p training set contains 1.3M words. A vocabulary of 30K words is only a small portion of it, but it covers about 98% of the text and produced good accuracy as reported below.

Training protocol We held out 10K documents from the training data for use as a validation set on each dataset, and meta-parameter tuning was done based on the performance on the validation set.

To minimize a log loss with softmax, mini-batch SGD with momentum 0.9 was conducted for n epochs (n was fixed to 50 for AG, 30 for Yelp.f/p and Dbpedia, and 15 for the rest) while the learning rate was set to η for the first $\frac{4}{5}n$ epochs and then 0.1η for the rest[3]. The initial learning rate η was considered to be a meta-parameter. The mini-batch size was fixed to 100. Regularization was done by weight decay with the parameter 0.0001 and by optional dropout (Hinton et al., 2012) with 0.5 applied to the input to the top layer. In some cases overfitting was observed, and so we performed early stopping, based on the validation performance, after reducing the learning rate to 0.1η. Weights were initialized by the Gaussian distribution with zero mean and standard deviation 0.01. The discrete input to the region embedding layer was fixed to the bow input, and the region size was chosen from {1,3,5}, while fixing output dimensionality to 250 (same as convolution layers).

Details of unsupervised embedding training To facilitate comparison with ShallowCNN, we matched our unsupervised embedding setting exactly with that of (Johnson and Zhang, 2016). That is, we trained the same four types of tv-embeddings, which are embeddings of 5- and 9-word regions, each of which represents the input regions by either 30K-dim bow or 200K-dim

[3]This learning rate scheduling method was used also in (Johnson and Zhang, 2015a,b, 2016). It was meant to reduce learning rate when error plateaus, as is often done on image tasks, e.g., (He et al., 2015), though for simplicity, the timing of reduction was fixed for each dataset.

	Models	Deep	Unsup. embed.	Yelp.p	Yelp.f	Yahoo	Ama.f	Ama.p
1	DPCNN + unsupervised embed.	✓	tv	**2.64**	**30.58**	**23.90**	**34.81**	**3.32**
2	ShallowCNN + unsup. embed. [JZ16]		tv	*2.90*	*32.39*	24.85	*36.24*	*3.79*
3	Hierarchical attention net [YYDHSH16]	✓	w2v	–	–	*24.2*	36.4	–
4	[CSBL16]'s char-level CNN: *best*	✓		4.28	35.28	26.57	37.00	4.28
5	fastText bigrams (Joulin et al., 2016)			4.3	36.1	27.7	39.8	5.4
6	[ZZL15]'s char-level CNN: *best*	✓		4.88	37.95	28.80	40.43	4.93
7	[ZZL15]'s word-level CNN: *best*	✓	(w2v)	4.60	39.58	28.84	42.39	5.51
8	[ZZL15]'s linear model: *best*			4.36	40.14	28.96	44.74	7.98

Table 2: Error rates (%) on larger datasets in comparison with previous models. The previous results are roughly sorted in the order of error rates (best to worst). The best results and the second best are shown in bold and italic, respectively. 'tv' stands for tv-embeddings. 'w2v' stands for word2vec. '(w2v)' in row 7 indicates that the best results among those with and without word2vec pretraining are shown. Note that '*best*' in rows 4&6–8 indicates that we are giving an 'unfair' advantage to these models by choosing the *best test error rate* among a number of variations presented in the respective papers.
[JZ16]: Johnson and Zhang (2016), [YYDHSH16]: Yang et al. (2016), [CSBL16]: Conneau et al. (2016), [ZZL15]: Zhang et al. (2015)

bags of {1,2,3}-grams, retaining only the most frequent 30K words or 200K {1,2,3}-grams. Training was done on the labeled data (disregarding the labels), setting the training objectives to the prediction of adjacent regions of the same size as the input region (i.e., 5 or 9). Weighted square loss $\sum_{i,j} \alpha_{i,j}(\mathbf{z}_i[j] - \mathbf{p}_i[j])^2$ was minimized where i goes through instances, $\mathbf{z}$ represents the target regions by bow, $\mathbf{p}$ is the model output, and the weights $\alpha_{i,j}$ were set to achieve the negative sampling effect. The dimensionality of unsupervised embeddings was set to 300 unless otherwise specified. Unsupervised embeddings were fixed during the supervised training – no fine-tuning.

3.2 Results

In the results below, the *depth* of DPCNN was fixed to 15 unless otherwise specified. Making it deeper did not substantially improve or degrade accuracy. Note that we count as *depth* the number of hidden weight layers including the region embedding layer but excluding unsupervised embeddings, therefore, 15 means 7 convolution blocks of 2 layers plus 1 layer for region embedding.

3.2.1 Main results

Large data results We first report the error rates of our full model (DPCNN with 15 weight layers plus unsupervised embeddings) on the larger five datasets (Table 2). To put it into perspective, we also show the previous results in the literature. The previous results are roughly sorted in the order of error rates from best to worst. On all the five datasets, DPCNN outperforms all of the previous results, which validates the effectiveness of our approach.

DPCNN can be regarded as a deep extension of ShallowCNN (row 2), sharing region embedding enhancement with diverse unsupervised embeddings. Note that ShallowCNN enhanced with unsupervised embeddings (row 2) was originally proposed in (Johnson and Zhang, 2015b) as a semi-supervised extension of (Johnson and Zhang, 2015a), and then it was tested on the large datasets in (Johnson and Zhang, 2016). The performance improvements of DPCNN over ShallowCNN indicates that the added depth is indeed useful, capturing more global information. Yang et al. (2016)'s hierarchical attention network (row 3) consists of RNNs in the word level and the sentence level. It is more complex than DPCNN due to the use of RNNs and linguistic knowledge for sentence segmentation. Similarly, Tang et al. (2015) proposed to use CNN or LSTM to represent each sentence in documents and then use RNNs. Although we do not have direct comparison with Tang et al.'s model, Yang et al. (2016) reports that their model outperformed Tang et al.'s model. Conneau et al. (2016) and Zhang et al. (2015) proposed deep character-level CNNs (row 4&6). Their models underperform our DPCNN with relatively large differences in spite of their deepness. Our mod-

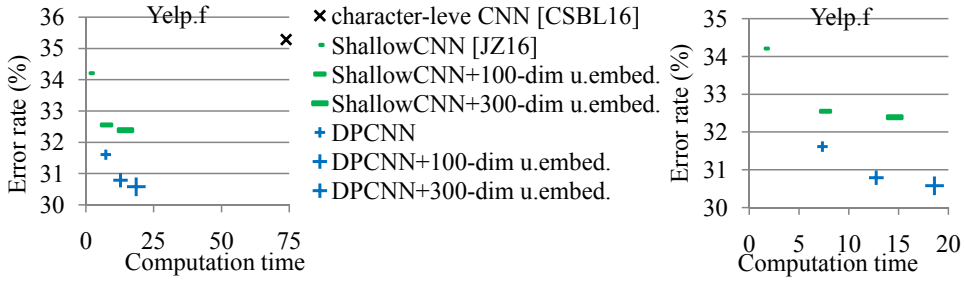


Figure 2: Error rates and computation time. DPCNN, ShallowCNN, and Conneau et al. (2016)'s character-level CNN. The x-axis is the time in seconds spent for categorizing 10K documents using our implementation on Tesla M2070. The right figure is a close-up of $x \in [0, 20]$ of the left figure. Though shown on one particular dataset Yelp.f, the trend is the same on the other four large datasets.

els are word-level and therefore use the knowledge of word boundaries which character-level models have no access to. While this is arguably not an apple-to-apple comparison, since word boundaries can be obtained for free in many languages, we view our model as much more useful in practice. Row 7 shows the performance of deep word-level CNN from (Zhang et al., 2015), which was designed to match their character-level models in complexity. Its relatively poor performance shows that it is not easy to design a high-performance deep word-level CNN.

Computation time In Figure 2, we plot error rates in relation to the computation time – the time spent for categorizing 10K documents using our implementation on a GPU. The right figure is a close-up of $x \in [0, 20]$ of the left figure. It stands out in the left figure that the character-level CNN of (Conneau et al., 2016) is much slower than DPCNNs. This is partly because it increases the number of feature maps with downsampling (i.e., no *pyramid*) while it is deeper (32 weight layers), and partly because it deals with characters – there are more characters than words in each document. DPCNNs are more accurate than ShallowCNNs at the expense of more computation time due to the depth (15 layers vs. 1 layer). Nevertheless, their computation time is comparable – the points of both fit in the same range $[0, 20]$. The efficiency of DPCNNs is due to the exponential decrease of per-layer computation due to downsampling with the number of feature maps being fixed.

Comparison with non-pyramid variants Furthermore, we tested the following two 'non-pyramid' models for comparison. The first model doubles the number of feature maps at every other downsampling so that per-layer computation is kept approximately constant[4]. The second model performs no downsampling. Otherwise, these two models are the same as DPCNN. We show in Figure 3 the error rates of these two variations (labeled as 'Increase #feature maps' and 'No downsampling', respectively) in comparison with DPCNN. The x-axis is the computation time, measured by the seconds spent for categorizing 10K documents. For all types, the models of depth 11 and 15 are shown. Clearly, DPCNN is more accurate and computes faster than the others. Figure 3 is on Yelp.f, and we observed the same performance trend on the other four large datasets.

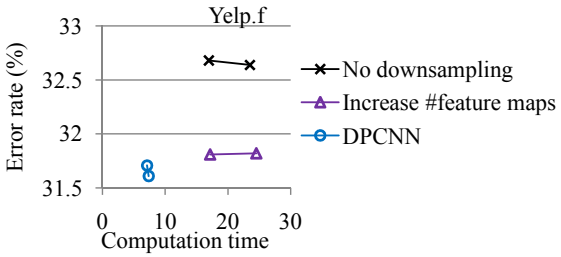


Figure 3: Comparison with non-pyramid models. Models of depth 11 and 15 are shown. No unsupervised embeddings.

Small data results Now we turn to the results on the three smaller datasets in Table 3. Again, the previous models are roughly sorted from best to worst. For these small datasets, the DPCNN performances with 100-dim unsupervised embed-

[4]Note that if we double the number of feature maps, it would increase the computation cost of the next layer by 4 times as it doubles the dimensionality of both input and output. On image, downsampling with stride 2 cancels it out as it makes data 4 times smaller by shrinking both horizontally and vertically, but text is one dimensional, and so downsampling with stride 2 merely halves data. That is why we doubled the number of feature maps at every other downsampling instead of at every downsampling to avoid exponential increase of computation time.

	Models	Deep	Unsup. embed.	AG	Sogou	Dbpedia
1	DPCNN + unsupervised embed.	✓	tv	*6.87*	**1.84**	*0.88*
2	ShallowCNN + unsup. embed. [JZ16]		tv	**6.57**	*1.89*	**0.84**
3	[ZZL15]'s linear model: *best*			7.64	2.81	1.31
4	[CSBL16]'s deep char-level CNN: *best*	✓		8.67	3.18	1.29
5	fastText bigrams (Joulin et al., 2016)			7.5	3.2	1.4
6	[ZZL15]'s word-level CNN : *best*	✓	(w2v)	8.55	4.39	1.37
7	[ZZL15]'s deep char-level CNN: *best*	✓		9.51	4.88	1.55

Table 3: Error rates (%) on smaller datasets in comparison with previous models. The previous results are roughly sorted in the order of error rates (best to worst). Notation follows that of Table 2.

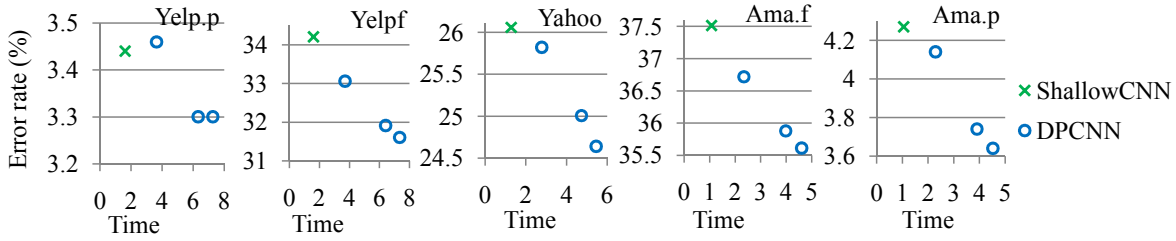


Figure 4: Error rates of DPCNNs with various depths (3, 7, and 15). The x-axis is computation time. No unsupervised embeddings.

dings are shown, which turned out to be as good as those with 300-dim unsupervised embeddings. One difference from the large dataset results is that the strength of shallow models stands out. ShallowCNN (row 2) rivals DPCNN (row 1), and Zhang et al.'s best linear model (row 3) moved up from the worst performer to the third best performer. The results are in line with the general fact that more complex models require more training data, and with the paucity of training data, simpler models can outperform more complex ones.

3.2.2 Empirical studies

We present some empirical results to validate the design choices. For this purpose, the larger five datasets were used to avoid the paucity of training data.

Depth Figure 4 shows error rates of DPCNNs with 3, 7, and 15 weight layers (blue circles from left to right). For comparison, the ShallowCNN results (green 'x') from (Johnson and Zhang, 2016) are also shown. The x-axis represents the computation time (seconds for categorizing 10K documents on a GPU). For simplicity, the results without unsupervised embeddings are shown for all. The error rate improves as the depth increases. The results confirm the effectiveness of our strategy of deepening the network.

Unsupervised embeddings To study the effectiveness of unsupervised embeddings, we experimented with variations of DPCNN that differ only in whether/how to use unsupervised embeddings (Table 4). First, we compare DPCNNs with and without unsupervised embeddings. The model with unsupervised embeddings (row 1, copied from Table 2 for easy comparison) clearly outperforms the one without them (row 4), which confirms the effectiveness of the use of unsupervised embeddings. Second, in the proposed model (row 1), a region embedding layer receives two types of input, the output of unsupervised embedding functions and the high-dimensional discrete input such as a bow vector. Row 2 shows the results obtained by using unsupervised embeddings to produce sole input (i.e., no discrete vectors provided to the region embedding layer). Degradations of error rates are up to 0.32%, small but consistent. Since the discrete input add almost no computation cost due to its sparseness, its use is desirable. Third, a number of previous studies used unsupervised word embedding to initialize word embedding in neural networks and then fine-tune it as training proceeds (pretraining). The model in row 3 does this with DPCNN using word2vec (Mikolov et al., 2013). The word2vec training was done on the training data (ignoring the labels),

	Unsupervised embeddings	Yelp.p	Yelp.f	Yahoo	Ama.f	Ama.p
1	tv-embed. (additional input)	**2.64**	**30.58**	**23.90**	**34.81**	**3.32**
2	tv-embed. (sole input)	2.68	30.66	24.09	35.13	3.45
3	word2vec (pretraining)	2.93	32.08	24.11	35.30	3.65
4	–	3.30	31.61	24.64	35.61	3.64

Table 4: Error rates (%) of DPCNN variations that differ in use of unsupervised embeddings. The rows are roughly sorted from best to worst.

same as tv-embedding training. This model (row 3) underperformed our proposed model (row 1). We attribute the superiority of the proposed model to its use of richer information than a word embedding. These results support our approach.

4 Conclusion

This paper tackled the problem of designing high-performance deep word-level CNNs for text categorization in the large training data setting. We proposed a deep pyramid CNN model which has low computational complexity, and can efficiently represent long-range associations in text and so more global information. It was shown to outperform the previous best models on six benchmark datasets.

References

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. 2016. Very deep convolutional networks for natural language processing. *arXiv:1606.01781v1 (6 June 2016 version)* .

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *arXiv:1512.03385* .

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. *arXiv:1603.05027* .

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Rie Johnson and Tong Zhang. 2015a. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT)*.

Rie Johnson and Tong Zhang. 2015b. Semi-supervised convolutional neural networks for text categorization via region embedding. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*.

Rie Johnson and Tong Zhang. 2016. Convolutional neural networks for text categorization: Shallow word-level vs. deep character-level. *arXiv:1609.00718* .

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv:1607.01795v3 (9 Aug 2016 version)* .

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*. pages 1746–1751.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations (ICLR)*.

Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT)*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*.