Transformable Convolutional Neural Network for Text Classification

Liqiang Xiao[1,2], Honglun Zhang[1,2], Wenqing Chen[1,2], Yongkun Wang[3], Yaohui Jin[1,2] *
[1] State Key Lab of Advanced Optical Communication System and Network, Shanghai Jiao Tong University
[2] Artificial Intelligence Institute, Shanghai Jiao Tong University
[3] Network and Information Center, Shanghai Jiao Tong University
{xiaoliqiang, zhanghonglun, wenqingchen, ykw, jinyh}@sjtu.edu.cn

Abstract

Convolutional neural networks (CNNs) have shown their promising performance for natural language processing tasks, which extract n-grams as features to represent the input. However, n-gram based CNNs are inherently limited to fixed geometric structure and cannot proactively adapt to the transformations of features. In this paper, we propose two modules to provide CNNs with the flexibility for complex features and the adaptability for transformation, namely, *transformable convolution* and *transformable pooling*. Our method fuses dynamic and static deviations to redistribute the sampling locations, which can capture both current and global transformations. Our modules can be easily integrated by other models to generate new transformable networks. We test proposed modules on two state-of-the-art models, and the results demonstrate that our modules can effectively adapt to the feature transformation in text classification.

1 Introduction

CNNs have been proven effective in many nature language processing tasks such as representation learning [Liu *et al.*, 2015], information retrieval [Shen *et al.*, 2014], text classification [Kalchbrenner *et al.*, 2014], etc. Convolutional layers in the CNNs extract features with geometrically fixed filters, which can be regarded as an implementation of N-Gram language model.

However, regular CNNs inevitably face the challenge of adapting to the feature transformation, which is caused by the fact that CNNs have many geometrically fixed structures, such as the convolutional layer limiting the filter shape into n-gram, the pooling layer selecting the superior features by solid chunks. These fixed structures render two main limitations for language representation:

- All the filters and chunks are continuous and shape-fixed, which make it difficult for CNNs to deal with some complex scenarios such as the non-consecutive or oversize features. In another word, it cannot fit the shape of features.
- Traditional CNNs are unable to actively adjust themselves to adapt to the transformation of features. They tend to remember all kinds of variants of features from a large dataset, but not to learn the patterns of transformation. This is undesirable because one feature may have a lot of transformed forms.

For example, given a phrase of "not nearly as good", traditional convolutional network is hard to directly capture the non-consecutive pattern "not...good". It is also difficult to recognize this pattern from numerous transformed forms, such as "not so good", "not even good".

Traditionally, most works cope with above challenges by increasing the volume of datasets to cover more variants of features or employ the constant features [Lowe, 1999; Bay *et al.*, 2006; Rublee *et al.*, 2011]. However, the potential of these two approaches is limited by requiring too much labor work or decreasing the performance.

Similar challenges also exist in the field of computer vision. Some works focus on inherently augmenting the flexibility of CNN models and the adaptability for feature transformation, which allow the models to reshape some layers to offset the feature transformation [Jia *et al.*, 2012; Jaderberg *et al.*, 2015; Jeon and Kim, 2017; Dai *et al.*, 2017]. They can be divided into two threads: 1) learning *static* shape parameters through back-propagation to capture only *global* distribution of transformation patterns [Jeon and Kim, 2017; Jaderberg *et al.*, 2015]; 2) learning *dynamical* shape information according to the inputs to obtain only *current* transformation patterns [Dai *et al.*, 2017]. However, these models cannot be directly applied to the CNNs designed for NLP tasks.

In this paper, we combine the thoughts of both threads and proposes two end-to-end modules for text classification to revise the convolution and pooling layers respectively. Both of our modules can enhance the capability of CNNs for modeling the transformation patterns, namely, *Transformable Convolution* and *Transformable Pooling*. Transformable convolution adds 1-Dimension (1D) deviations to the sampling locations of filters, which can not only help capture the complex features but also offset the transformation of features. The deviations are automatically learned without needs for

*Corresponding author

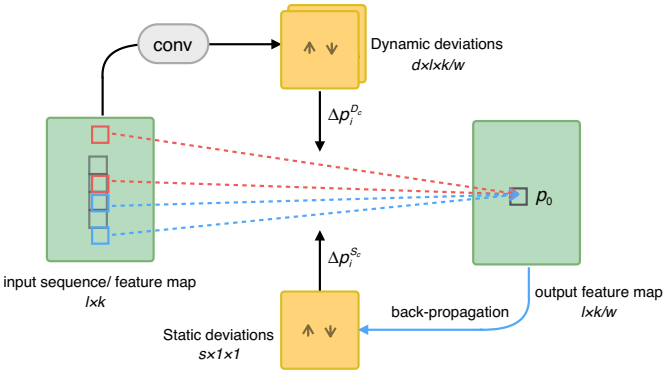


Figure 1: Illustration of Transformable Convolution. l, k, d, s and w represent the length of feature map, width of feature map, size of $\mathbf{D}_c$, size of $\mathbf{S}_c$ and the width of filters respectively. The width k of the output feature map is divided by w because convolution for text is 1D.

extra supervision or human experience. Transformable pooling shares the same idea with transformable convolution. On the base of pooling layer, it adds 1D deviations to the sampling locations of chunks to increase the adaptability for feature selection.

Our modules can be easily embedded into other models to construct new transformable ConvNets (TF-ConvNets) by displacing their counterparts. Extensive experiments and visualizations on text classification datasets verify the effectiveness of TF-ConvNets for capturing the patterns of transformation.

2 Transformable Convolution and Pooling

In this section, we introduce the mechanisms of two transformable modules, which have learnable shapes to adapt to the changes of features [Dai *et al.*, 2017]. Conventionally, the shapes of filters and chunks in convolution and pooling are fixed from the start. But the transformable modules add the learned position deviations onto filters or chunks, making their shapes flexible and adaptable. Position deviations are composed of *dynamic* part and *static* part. At inference stage, the dynamic deviations are related to current input, whose values are actively learned from current features to capture the *current* transformation information. On the contrary, values of static deviations are updated by back-propagation and keep static at interference stage, which describe the *global* distribution of shape information.

2.1 Transformable Convolution

CNNs for NLP usually employ more than one filters or channels. For simplicity, we provide the formulation for the case with one filter and channel, but it is easy to extend to more complex cases.

In typical convolution, the shape of filter can be described by the center position p_0 and the sampling locations p_i, the relative distances from p_0. We use $\mathbf{C}$ to represent the collection of p_i. For instance, when the length of a filter in conventional convolution is 5,

$$\mathbf{C} = \{-2, -1, 0, 1, 2\}. \tag{1}$$

Output feature map is generated by a convolving operation between weight $\mathbf{w}$ and input $\mathbf{x}$, which can be formulated as

$$\mathbf{y} = \mathbf{w} * \mathbf{x}. \tag{2}$$

And every element in feature map $\mathbf{y}$ can be calculated by:

$$\mathbf{y}(p_0) = \sum_{p_i \in \mathbf{C}} \mathbf{w}(p_i) \cdot \mathbf{x}(p_0 + p_i), \tag{3}$$

where p_0 is a location in output feature map and p_i enumerates the points in collection $\mathbf{C}$ of sampling locations.

In transformable convolution, $\mathbf{C}$ is divided into two parts, one is current feature related, and the other is globally depended. Let $\mathbf{D}_c \subset \mathbf{C}$ and $\mathbf{S}_c \subset \mathbf{C}$ denote them respectively. Then they are added with different deviations $\Delta p_i^{\mathbf{D}_c}$ and $\Delta p_i^{\mathbf{S}_c}$ to revise the sampling locations. And Eq.(3) becomes

$$\begin{aligned}\mathbf{y}(p_0) = &\sum_{p_i \in \mathbf{D}_c} \mathbf{w}(p_i) \cdot \mathbf{x}(p_0 + p_i + \Delta p_i^{\mathbf{D}_c}) \\ &+ \sum_{p_i \in \mathbf{S}_c} \mathbf{w}(p_i) \cdot \mathbf{x}(p_0 + p_i + \Delta p_i^{\mathbf{S}_c}).\end{aligned} \tag{4}$$

Now, sampling locations of filter are redistributed, and are not in the shape of regular rectangular any longer. Considering that $\Delta p_i^{\mathbf{D}_c}$ and $\Delta p_i^{\mathbf{S}_c}$ are often fractional, value of function $\mathbf{x}(\cdot)$ is calculated by linear interpolation:

$$\mathbf{x}(p) = \sum_q K(p, q) \cdot \mathbf{x}(q). \tag{5}$$

Here p denotes $p_0 + p_i + \Delta p_i^{\mathbf{D}_c}$ or $p_0 + p_i + \Delta p_i^{\mathbf{S}_c}$, q enumerates the locations in the whole input feature map, and K is the linear interpolation kernel. In linear space, K is computed by

$$K(p, q) = max(0, 1 - |p - q|), \tag{6}$$

where $max(\cdot)$ functions as a "mask", limiting the dilation of the interpolation area into 1.

Figure 1 shows the mechanism of transformable convolution. The dynamic deviations $\Delta p_i^{\mathbf{D}_c}$ at the top are learned from current input sequence or feature map via a plain convolution layer, whose values are added to sampling locations in $\mathbf{D}_c$. Since they are generated from the previous layer, their values change dynamically according to the current features to fit current transformation patterns. The static deviations $\Delta p_i^{\mathbf{S}_c}$ at bottom are variables, which are updated by back-propagation during training stage and keeps static at inference time. Hence, the static deviations describe a global distribution of feature shape.

2.2 Transformable Pooling

There are many versions of pooling operation, which is used to select the most important features and reduce the resolution. Here we improve a plain version of Chunk-Max Pooling (Figure 2), which has been proven useful to keep the position information of features.

Given a feature map $\mathbf{x}$ of size $l \times k$, Chunk-Max Pooling first divides $\mathbf{x}$ into $m \times n (0 < m < l, 0 < n \leq k)$[1] chunks.

[1] Typically, n equals k in text classification tasks.

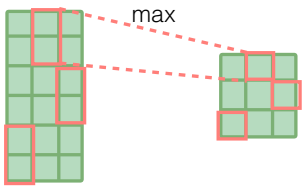


Figure 2: Illustration of plain Chunk-Max Pooling. The input feature map is cut into chunks and the output feature map is constructed by the max values of these chunks.

Then, it outputs a $m \times n$ feature map $\mathbf{y}$ by concatenating the maximum value of every chunk. Let p_0 denote top-left corner of the (i, j)-th chunk, the pooling process can be formulated as

$$\mathbf{y}(p_0) = \underset{p_i \in \mathbf{O}}{max}(\{\mathbf{x}(p_0 + p_i)\}), \tag{7}$$

where p_i enumerates the sampling locations $\mathbf{O}$ in $chunk(i, j)$, and $i\left\lfloor \frac{l}{m} \right\rfloor < p_i < (i+1)\left\lceil \frac{l}{m} \right\rceil, j\left\lfloor \frac{k}{n} \right\rfloor < p_j < (j+1)\left\lceil \frac{k}{n} \right\rceil$.

Similar to transformable convolution, sampling locations are divided into two parts of $\mathbf{D}_o$and $\mathbf{S}_o$. Then we redistribute these sampling locations by adding the deviations $\Delta p_i^{\mathbf{D}_o}, \Delta p_i^{\mathbf{S}_o} \in \Delta\mathbf{p}$ to them respectively. This makes the filter fit the shape of features better. Eq.(7) becomes

$$\begin{aligned}\mathbf{y}(p_0) = \underset{p_i^{\mathbf{D}_o} \in \mathbf{D}_o, p_i^{\mathbf{S}_o} \in \mathbf{S}_o}{max} (&\mathbf{x}(p_0 + p_i^{\mathbf{D}_o} + \Delta p_i^{\mathbf{D}_o}) \oplus \\ &\mathbf{x}(p_0 + p_i^{\mathbf{S}_o} + \Delta p_i^{\mathbf{S}_o})),\end{aligned} \tag{8}$$

where $\oplus$ denotes the concatenation operation.

In the same way as Eq.(5) and (6), the value of $\mathbf{x}$ is calculated by 1D bilinear interpolation (Eq.(6)), since the $\Delta\mathbf{p}^{\mathbf{D}_o}$ and $\Delta\mathbf{p}^{\mathbf{S}_o}$ are generally fractional.

As illustrated in Figure 3, the top dynamic deviations are learned from the current input. More concretely, Chunk-Max Pooling first generates a feature map, on which a fully connected layer emits the normalized deviations $\Delta\hat{\mathbf{p}}$. On the other hand, static deviation map is relatively stable, whose normalized value $\Delta\hat{\mathbf{p}}^{\mathbf{S}_o}$ is initialized randomly and optimized by back-propagation.

Then, these normalized deviations are used to calculate the deviations by $\Delta\mathbf{p} = \mu \cdot \Delta\hat{\mathbf{p}} \circ (l, k)$ where l and k are the length and width of input feature map and μ is the coefficient to adjust the scale of deviations. μ is set to 0.1 empirically in this paper. This normalization process is designed to keep the deviation scale independent from the size of chunks.

2.3 Normalized Gradient

For the static deviations, the gradients of them $\frac{\partial L}{\partial \Delta p}$ determine the moving speed of the sampling locations. However, they are dependent on other changeable parameters like the weights or inputs. So their gradient values may change drastically, which makes the magnitude of deviations unpredictable. If the deviations are too small, the modules cannot live up to the potential as they are expected. On the contrary, the sampling locations are too dispersed to obtain useful information. Hence, controlling the gradient of deviations is necessary.

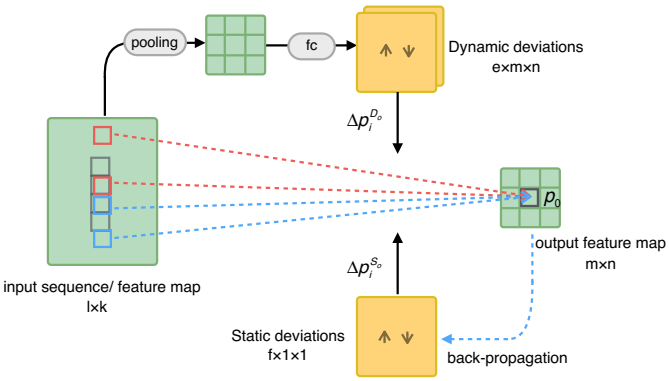


Figure 3: Illustration of transformable pooling. $e = |\mathbf{D}_o|$ and $f = |\mathbf{S}_o|$.

In the paper, we use normalized gradient to properly limit the movement of sampling locations. As shown in Eq. (9), normalized gradient is formalized as

$$\overline{\frac{\partial L}{\partial \Delta p}} = \frac{\partial L}{\partial \Delta p} \Big/ \left| \frac{\partial L}{\partial \Delta p} \right|, \tag{9}$$

where L is the loss function and Δp denotes all kinds of deviations. This means we only use the direction of gradient and limited the stride up to 1 in one iteration. Empirically, learning rate is set to 0.001, which indicates that the stride is up to 0.001 in one iteration.

3 Transformable ConvNets

Our modules are elaborately designed to be compatible with existing models. They can be seamlessly used by replacing their counterparts, since they have the same interfaces with the vanilla CNNs and are easy to train with back-propagation algorithm. In this section, we embed our modules into two state-of-the-art CNN models to show the performance on the text classification tasks.

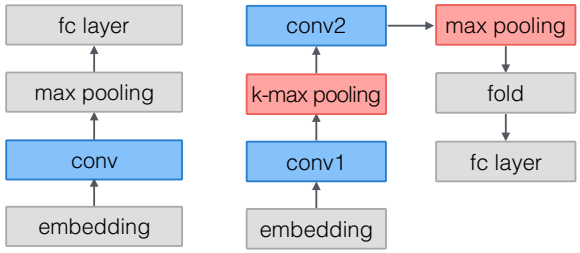
TF-MCCNN

As Figure 4(a) shows, Multichannel CNN (MCCNN) is a four-layer network proposed by [Kim, 2014] to represent text sequences, which only used one convolution layer and pooling layer for feature extraction. Its architecture has been proven simple yet practical.

For word embedding, MCCNN exploits two lookup tables to vectorize the input sequence. One is initialized by the pre-trained word vectors that are learned by unsupervised language models to overcome the absence of large-scale supervised training set. The other one has the same size with the former, but it is randomly initialized.

For feature extraction, MCCNN first uses a convolutional layer with multiple filters (3 different sizes with 100 filters each). Each filter $\mathbf{w}$ is applied on both embedding results $\mathbf{x}_c$ and added up to emit the feature map:

$$F = \sum_c f(\mathbf{w} * \mathbf{x}_c + b) \tag{10}$$



(a) MCCNN (b) DCNN

Figure 4: Comparison of two plain CNNs for text classification. (a) We replace only the convolutional layers for TF-ConvNet. (b) We replace both convolutional and pooling layers.

Dataset	Train	Dev.	Test	Voc.	Len.	Class
SST-1	8544	1101	2210	18K	18	5
SST-2	6920	872	1821	16K	19	2
IMDB	25000	-	25000	392K	279	2
SUBJ	8000	1000	1000	21K	23	2
TREC	4907	545	500	10K	10	6
Books	1398	200	400	22K	159	2
Electronics	1398	200	400	11K	111	2
DVDs	1400	200	400	22K	189	2
Kitchen	1400	200	400	10K	93	2

Table 1: Statistics of the text classification datasets. Train, Dev. and Test denote the size of train, development and test set respectively; Voc.: Vocabulary size; Len.: Average sentence length.

where f is the rectified linear unit and b is the bias. Then, the 300 output feature maps are put into max-over-time pooling layer to form a 300-dimension feature map.

For comparison and avoiding changing the structure of MCCNN too much, we just replace the convolutional layer with transformable convolution and keep the rest constant. The newly constructed model is referred to as TF-MCCNN.

TF-DCNN

Dynamic convolutional neural network (DCNN) [Kalchbrenner *et al.*, 2014] is a deeper and slimmer text classification model with 7 layers, which is named after its dynamic k-max pooling. As Figure 4(b) shows, it alternates wide convolutional layers with pooling layers to capture the high-level semantic features. Two convolutional layers use different filters of size 2 and 3 respectively. And the maximum widths of chunks in two pooling layers are set to 5 and 3 respectively. It also inserts a folding layer before the second pooling layer to create the interaction across dimensions.

To transform DCNN into a TF-ConvNet, we replace all the convolutional and pooling layers with the corresponding transformable modules and keep the others same. And the new model is called TF-DCNN.

4 Experiments

4.1 Datasets

We extensively evaluate our deformable ConvNets on 9 datasets, which are collected in different domains with different labels. Their statistics are listed in Table 1.

SST-1, SST-2 [Socher *et al.*, 2013] and IMDB [Maas *et al.*, 2011] are about movie reviews, which all classify the reviews according the sentiment but IMDB has longer content and SST-1 has more classes. SUBJ[2] is a subjectivity dataset, whose task is to classify a sentence level text as being subjective or objective [Pang *et al.*, 2004]; TREC[3] dataset has the task of classifying a question into 6 types (the questions are about location, person, numeric information, etc.) [Li and Roth, 2002]

The rest 4 datasets are all product reviews, which are comprised of Amazon product reviews in 4 domains, including books, electronics, DVDs and kitchen. These corpora are classified according to the sentiment of positiveness or negativeness. They are derived from the raw data published by [Blitzer *et al.*, 2007].

4.2 Competitor Methods

Besides our baseline models (DCNN and MCCNN)[4], we also compare our approaches with other state-of-the-art models, involving NBOW, RTNT [Socher *et al.*, 2013], PV [Le and Mikolov, 2014], MGNC-CNN [Zhang *et al.*, 2016] and Tree-LSTM [Tai *et al.*, 2015]. For LSTM and BiLSTM in Table 2, we used the architectures implemented by [Józefowicz *et al.*, 2015], which remove the peep-hole from the similar architecture of [Graves, 2013].

4.3 Hyper-parameters

For TF-MCCNN, as original, we use word vectors from *Word2Vec*[5], which is trained by bag-of-words model on 100B Google News corpus [Mikolov *et al.*, 2013]. For structure, TF-MCCNN uses three different lengths of $5, 6, 7$ for every 100 filters. Correspondingly, the sizes of sampling locations for $\mathbf{D}_c$ and $\mathbf{S}_c$ are set to $2, 3, 4$ and $3, 3, 3$ respectively. For training, all involved parameters are randomly initialized from a truncated normal distribution with zero mean and standard deviation. And the learning rates are $10^{-3}, 10^{-4}$ for the first $\frac{2}{3}$ and last $\frac{1}{3}$ iterations. Mini-batch is generated by randomly selecting 50 samples every time from the corpus. Training is implemented through SGD (stochastic gradient descend) with the Adadelta update rule [Zeiler, 2012].

For TF-DCNN, parameters are normally initialized in [-0.1, 0.1]. The filter widths in transformable convolutional layer are $5, 6$ with $|\mathbf{D}_c| = 2, 3$ and $|\mathbf{S}_c| = 3, 3$. The shapes of chunks in two transformable pooling layers are $7 \times 1, 5 \times 1$ with $|\mathbf{D}_o| = 5, 3; |\mathbf{S}_o| = 2, 2$. For training, the network is trained with mini-batches of size 50 by back-propagation and the gradient-based optimization is performed using the Adagrad update rule [Duchi *et al.*, 2011]. Learning rates are also set to $10^{-3}, 10^{-4}$ for the first $\frac{2}{3}$ and last $\frac{1}{3}$ iterations.

[2]http://www.cs.cornell.edu/people/pabo/movie-review-data/

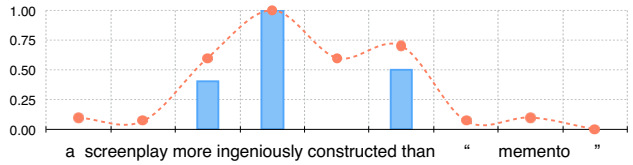
[3]http://cogcomp.cs.illinois.edu/Data/QA/QC/

[4]We use the published results and fill in the rest with released codes.

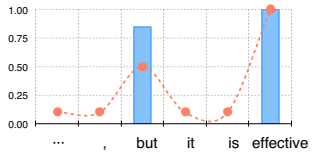
[5]https://code.google.com/p/word2vec/

Models	SST-1	SST-2	IMDB	SUBJ	TREC	Books	Electronics	DVDs	Kitchen
NBOW	42.4	80.5	83.6	91.3	88.2	-	-	-	-
RNTN	45.7	85.4	-	-	-	-	-	-	-
PV	44.6	82.7	**91.7**	90.5	91.8	-	-	-	-
Tree-LSTM	**50.6**	86.5	-	-	-	-	-	-	-
LSTM	45.9	85.8	88.5	91.6	92.5	79.5	80.5	**81.7**	78.0
BiLSTM	46.4	86.1	89.9	90.1	92.7	81.0	78.5	80.5	81.2
MGNC-CNN	48.7	88.3	-	94.1	-	-	-	-	-
DCNN	48.5	86.8	88.6	93.6	93.0	79.7	80.4	81.6	77.8
MCCNN	47.4	88.1	89.7	93.2	92.2	79.9	77.0	81.2	77.7
TF-DCNN	48.8	88.1	89.9	**95.0**	93.2	**82.0**	**81.7**	81.3	81.2
TF-MCCNN	49.1	**88.5**	90.3	94.2	**93.5**	81.5	81.1	81.6	**81.4**

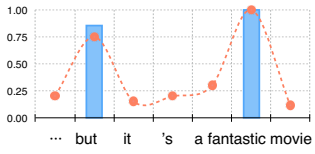
Table 2: Accuracies of our models and comparison.



(a) a screenplay more ingeniously constructed than "memento"



(b) s1m0ne's satire is not subtle , but it is effective



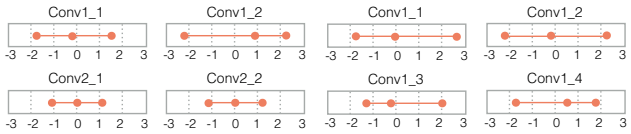
(c) it's tough to watch , but it's a fantastic movie .

Figure 5: Visualization of the dynamic part of filters. Blue columns represent the sampling locations of the filter that has the biggest value of activation. And red lines show the activation values of transformation convolution.

4.4 Performance

Table 2 reports the performance of our approaches against other models. We can see that TF-ConvNets obviously improve the performance over their baseline models (MCCNN and DCNN), which demonstrates the effectiveness of transformable modules. Compared with other models, TF-ConvNets gain the best performance on 6 tasks and show very competitive results on the others.

Although Tree-SLTM outperforms our models on SST-1, it obtains the external topological information via parser trees of sentences. Our models do not obtain the best score on IMDB and DVDs, as the sequences in them are much longer than other datasets, which requires stronger capability for long-term dependency learning. In this paper, we mainly focus on the effects of transformation mechanism on text representation, but our approach can be also applied on other models designed for long sequence to produce better performances.



(a) Learned static sampling locations in TF-DCNN

(b) Learned static sampling locations in TF-MCCNN

Figure 6: Visualization of the static part of filters. Red points denote the learned sampling locations of the filters. (a) Conv$i$_j denotes the shape of j-th filter in i-th transformable convolutional layer. (b) Four randomly selected filters in TF-MCCNN.

4.5 Visualization

To intuitively understand how the transformable ConvNets adapt to the features. We design an experiment to show what patterns the models capture and how they transform. We first randomly extract a sentence from the test set of SST-2 dataset and feed it into TF-DCNN. Then, for the dynamic part of filters in the first transformable convolution layer, we visualize the sampling location of the biggest activation. We also illustrate all the activations generated by transformation convolution. Figure 5 visually shows the effect of transformation convolution on three different sentences. As the blue columns show in 5(a), the filter skips a word and captures the key structure of *"more ingeniously ... than"*, which is crucial to the prediction. It is almost impossible for a plain filter to directly capture this kind of features, which generally have to be composed in high-level layers. Besides, we can have an obvious observation from Figure 5(b) and 5(c) that the filter can adapt to the transformation of features. From two sentences, the filter actively learns the dynamic shape of the feature and recognizes its negation structure *"but ... [Adj.]"*, even they have different numbers for middle words.

For static deviations, their values are learned via backpropagation and finally reach constringency. Figure 6 illustrates the final shapes learned by static part of filters. From the extensive experiments, we find a regular pattern that the filters tend to dilate in low layer but are more similar to conventional convolution in high layer. Our conjecture is that filters in low

Modules	Books	Elec.	DVDs	Kitchen	**Avg.**
DCNN	79.7	80.4	81.7	77.8	**79.9**
+TF-conv	81.7	81.3	81.3	81.1	**81.4**
+TF-pooling	81.1	80.9	81.3	80.5	**81.0**
+Both	82.0	81.7	81.3	81.2	**81.6**
DCNN(0,0)	79.7	80.4	81.7	77.8	**79.9**
DCNN(2,0)	81.6	80.9	81.0	81.0	**81.1**
DCNN(0,2)	80.3	80.8	79.9	79.6	**80.2**
DCNN(2,2)	81.7	81.3	81.3	81.1	**81.4**
MCCNN(0)	79.9	77.0	81.2	77.7	**79.0**
MCCNN(10)	81.2	80.6	81.6	80.9	**81.1**
MCCNN(20)	81.5	81.1	81.6	81.4	**81.4**

Table 3: Ablation study of transformable ConvNets. Avg. column reports the average accuracies of 4 datasets from amazon product reviews. The tuples following DCNN represent the numbers of used transformable convolution in the first and second convolutional layer. The numbers following MCCNN denote the numbers of transformable convolution used in MCCNN.

layers expand to capture longer features, while higher layers focus on building connections between close features.

5 Ablation Study

In this section, we conduct more investigations to study the independent effect of each module and the influence of transformable convolution number.

Effect of Each Module

To prove the effectiveness of both modules, we do some ablation experiments on TF-DCNN since it uses both modules. Table 3 shows the performance of variants on 4 product review datasets. When transformable convolution is applied on the network, an average performance improvement of 1.5% is obtained. With transformable pooling only, it performs a boost of average 1.1%. This demonstrates that both modules make a contribution to the promising scores.

Number of Transformable Convolution

On both ConvNets, we implement a series of incremental experiments to study the influence of transformable convolution number on performance. The results reported in Table 3 indicate that 1) Transformable convolution shows more improvement in lower layers. Because it is mainly designed for feature extraction, while it also works in high levels for feature fusion. This concurs with the phenomenon in Figure 6(a) that filter shapes change more in lower layers. 2) Performance steadily improves when more transformable convolution layers are involved until a point that improvement saturates. Specifically, the improvement saturates when 20 transformable layers are used in MCCNN.

6 Related Work

CNN inherently suffers from the limitation to model the transformation of objects because of its fixed geometric structure. Traditionally, most of the works focus on covering more variance of features in the datasets or using consistent features, rather than the inner problem. Some recent works attempt to augment the adaptability of models by revising some basic structures, such as convolutional and pooling layers. These works can be classified into two categories: **semi-end-to-end** models and **full-end-to-end** models.

For models in the first category, they generally reshape some layers with handcrafted patterns. For input, Spatial Transform Network (STN) [Jaderberg *et al.*, 2015] tries to learn the transform parameters to warp the inputs, such as affine transformation. For convolutional layer, Atrous Convolution [Holschneider *et al.*, 1988] improves the convolution by scattering filter sampling locations with the same strides, which increases the receptive field of the filters with the same parameter complexity. For pooling layer, Receptive Field Learning [Jia *et al.*, 2012], following the [Lazebnik *et al.*, 2006], learns a subset of pooling regions via the idea of overcomplete from a large number of candidates.

However, in these works the transformation information is known as prior and the patterns for transformation are limited. They cannot deal with the intricate transformations that are out of human knowledge. Moreover, they are designed for computer vision and are not suitable for text representation tasks, which have different dimensions of transformation.

For the second category, models learn the sampling location autonomously and the transformation is not restricted by manually made patterns. That means this kind of models are more general and fully end-to-end, which can adapt to more intricate transformations. To be specific, Active Convolution [Jeon and Kim, 2017] learns the shapes of filters for convolution, which can adjust itself according the global datasets via back-propagation. However, the shapes of filters are static and are learned per task or per training, lacking reactivity for current input or local features.

Another end-to-end model Deformable CNN [Dai *et al.*, 2017] attempts to model the transformation through learning the offsets for convolution filters and pooling regions. It also provides freedom to shapes for both modules, while the offsets drastically vary with the current input and sampling location. This makes it highly fit the current field losing outline and consistent patterns for the integral task.

In this paper, our modules overcome the drawbacks of above works and consider both current and global transformations. It is noteworthy that our work successfully introduces the concept of transformation into language field and enhances the flexibility and adaptability of CNN models for text representation.

7 Conclusion

In this paper, by combining two mainstream methods, proposed modules enhance the flexibility of structure for CNN and augment the adaptability for transformable features. Our approach overcomes the drawbacks of n-gram based models for text classification and shows an ability to capture the overlength or non-consecutive features. Extensive tests and visualization experiments demonstrate the effectiveness of our modules for feature transformation.

References

[Bay *et al.*, 2006] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool. SURF: speeded up robust features. In *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings, Part I*, pages 404–417, 2006.

[Blitzer *et al.*, 2007] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL 2007, Proceedings of ACL, June 23-30, 2007, Prague, Czech Republic*, pages 187–205, 2007.

[Dai *et al.*, 2017] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR*, abs/1703.06211, 2017.

[Duchi *et al.*, 2011] John Duchi, Elad Hazan, and Yoram Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. JMLR.org, 2011.

[Graves, 2013] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[Holschneider *et al.*, 1988] M. Holschneider, R. Kronland-Martinet, J. Morlet, and Ph. Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. pages 286–297, 1988.

[Jaderberg *et al.*, 2015] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2017–2025, 2015.

[Jeon and Kim, 2017] Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. *CoRR*, abs/1703.09076, 2017.

[Jia *et al.*, 2012] Yangqing Jia, Chang Huang, and Trevor Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3370–3377, 2012.

[Józefowicz *et al.*, 2015] Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of, ICML 2015, Lille, France, 6-11 July 2015*, pages 2342–2350, 2015.

[Kalchbrenner *et al.*, 2014] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 655–665, 2014.

[Kim, 2014] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, 2014.

[Lazebnik *et al.*, 2006] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *(CVPR 2006), 17-22 June 2006, New York, NY, USA*, pages 2169–2178, 2006.

[Le and Mikolov, 2014] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *Computer Science*, 4:1188–1196, 2014.

[Li and Roth, 2002] Xin Li and Dan Roth. Learning question classifiers. *Coling*, 12(24):556–562, 2002.

[Liu *et al.*, 2015] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye Yi Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *In NAACL HLT*, pages 912–921, 2015.

[Lowe, 1999] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.

[Maas *et al.*, 2011] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Meeting of the Association for Computational Linguistics*, pages 142–150, 2011.

[Mikolov *et al.*, 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26:3111–3119, 2013.

[Pang *et al.*, 2004] Pang, Bo, Lee, and Lillian. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. *Proceedings of ACL*, pages 271–278, 2004.

[Rublee *et al.*, 2011] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. ORB: an efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 2564–2571, 2011.

[Shen *et al.*, 2014] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 101–110, 2014.

[Socher *et al.*, 2013] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. 2013.

[Tai *et al.*, 2015] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *Computer Science*, 5(1):: 36., 2015.

[Zeiler, 2012] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *Computer Science*, 2012.

[Zhang *et al.*, 2016] Ye Zhang, Stephen Roller, and Byron C. Wallace. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. In *NAACL HLT 2016, San Diego California, USA, June 12-17, 2016*, pages 1522–1527, 2016.