Very Deep Convolutional Networks for Natural Language Processing

Alexis Conneau
Facebook AI Research
aconneau@fb.com

Holger Schwenk
Facebook AI Research
schwenk@fb.com

Yann Le Cun
Facebook AI Research
yann@fb.com

Loïc Barreau
LIUM, University of Le Mans, France
loic.barrault@univ-lemans.fr

Abstract

The dominant approach for many NLP tasks are recurrent neural networks, in particular LSTMs, and convolutional neural networks. However, these architectures are rather shallow in comparison to the deep convolutional networks which are very successful in computer vision. We present a new architecture for text processing which operates directly on the character level and uses only small convolutions and pooling operations. We are able to show that the performance of this model increases with the depth: using up to 29 convolutional layers, we report significant improvements over the state-of-the-art on several public text classification tasks. To the best of our knowledge, this is the first time that very deep convolutional nets have been applied to NLP.

1 Introduction

The goal of natural language processing (NLP) is to process text with computers in order to analyze it, to extract information and eventually to represent the same information differently. We may want to associate categories to parts of the text (e.g. POS tagging or sentiment analysis), structure text differently (e.g. parsing), or convert it to some other form which preserves all or part of the content (e.g. machine translation, summarization). The level of granularity of this processing can range from individual characters or words up to whole sentences or even paragraphs.

After a couple of pioneer works ([2, 3, 4] among others), the use of neural networks for NLP applications is attracting huge interest in the research community and they are systematically applied to all NLP tasks. However, while the use of (deep) neural networks in NLP has shown very good results for many tasks, it seems that they have not yet reached the level to outperform the state-of-the-art by a large margin, as it was observed in computer vision and speech recognition.

Convolutional neural networks, in short *ConvNets*, are very successful in computer vision. In early approaches to computer vision, handcrafted features were used, for instance *"scale-invariant feature transform (SIFT)"*, followed by some classifier. The fundamental idea of ConvNets is to consider feature extraction and classification as one jointly trained task. This idea has been improved over the years, in particular by using many layers of convolutions and pooling to sequentially extract a *hierarchical representation* of the input. The best networks are using more than 150 layers [7, 8].

Many NLP approaches consider words as basic units. An important step was the introduction of continuous representations of words. These *word embeddings* are now the state-of-the-art in NLP. However, it is less clear how we should best represent a sequence of words, e.g. a whole sentence, which has complicated syntactic and semantic relations. In general, in the same sentence, we may be faced with local and long-range dependencies. Currently, the main-stream approach is to consider a sentence as a sequence of tokens (characters or words) and to process them with a recurrent neural network (RNN). Tokens are usually processed in sequential order, from left to right, and the RNN is expected to *"memorize"* the whole sequence in its internal states. The most popular and successful RNN variant are certainly LSTMs – there are many works which have shown the ability of LSTMs to model long-range dependencies in NLP applications, e.g. [17, 18] to name just a few. However, we argue that LSTMs are generic learning machines for sequence processing which are lacking task specific structure.

We propose the following analogy. It is well known that a fully connected one hidden layer neural network can in principle learn any real-valued function, but much better results can be obtained with a deep problem-specific architecture which develops hierarchical representations. By these means, the search space is heavily constrained and efficient solutions can be learned with SGD. ConvNets for computer vision are typical examples of this. We argue, that to some extent, RNNs or LSTMs could be considered as an extension of fully connected neural networks for sequence processing. They are unstructured and not adapted to the particular task: the same LSTM cell is used in all applications - only the number of LSTM cells is varied and eventually several LSTM layers are stacked.

It is often argued that RNN learn deep representations of sequences by unrolling them over time (or the sequence length). However, we argue that the depth varies considerably for the tokens in the sequence: the first token goes through many recurrent layers while basically a linear operation is applied to the last one. Despite the gating mechanism of LSTMs, this leads to the fact that recent tokens are better "memorized" than older ones. This has even motivated researchers to process sequences in reverse order, e.g. [18], or in both directions, named bidirectional LSTMs. It is also common to stack several LSTM layers. To the best of our knowledge, no improvements were observed beyond four stacked LSTM layers despite drop-out regularization.

We believe that the challenge in NLP is to develop deep architectures which are able to learn hierarchical representations of whole sentences, jointly with the task. In this paper, we propose to use deep architectures of many convolutional layers to approach this goal, using up to 29 layers. The design of our architecture is inspired by recent progress in computer vision, in particular [15, 7].

This paper is structured as follows. There have been previous attempts to use ConvNets for text processing. We summarize the previous works in the next section and discuss the relations and differences. Our architecture is described in detail in section 3. We have evaluated our approach on several sentence classification tasks, initially proposed by Zhang et al. [20]. These tasks and our experimental results are detailed in section 4. The proposed deep convolutional network outperforms the state-of-the-art convolutional neural networks on all tasks. The paper concludes with a discussion of future research directions.

2 Related work

There is a large body of research on sentiment analysis, or more generally on sentence classification tasks. Initial approaches followed the classical two stage scheme of extraction of (handcrafted) features, followed by a classification stage. Typical features include bag-of-words or n-grams, and their TF-IDF. These techniques have been compared with ConvNets by Zhang et al. [20]. We use the same corpora for our experiments. More recently, words or characters, are projected into a high dimensional space, and these embeddings are combined to obtain a fixed size representation of the input sentence, which then serves as input for the classifier. The simplest combination is the element-wise mean. This usually performs badly since all notion of token order is disregarded.

Another class of approaches are recursive neural networks. The main idea is to use an external tool, namely a parser, which specifies the order in which the word embeddings are combined. At each node, the left and right context are combined using weights which are shared for all nodes [16]. The state of the top node is fed to the classifier. A recurrent neural network (RNN) could be considered as a special case of a recursive NN: the combination is performed sequentially, usually from left to right. The last state of the RNN is used as fixed-sized representation of the sentence, or eventually a combination of all the hidden states.

First works using convolutional neural networks for NLP are probably [3, 4]. They have been subsequently applied to sentence classification [12, 11, 20]. We will discuss these techniques in more detail below. If not otherwise stated, all approaches operate on words which are projected into a high-dimensional space.

A rather shallow neural net was proposed in Kim [12]: one convolutional layer (using multiple widths and filters) followed by a max pooling layer over time. The final classifier uses one fully connected layer with drop-out. Results are reported on six data sets, in particular Stanford Sentiment Treebank (SST). A similar system was proposed in Kalchbrenner et al. [11], but using five convolutional layers. An important difference is also the introduction of multiple *temporal k-max pooling* layers. This allows to detect the k most important features in a sentence, independent of their specific position, preserving their relative order. The value of k depends on the length of the sentence and the position of this layer in the network. Zhang et al. [20] were the first to perform sentiment analysis entirely at the character level. Their systems use up to six convolutional layers, followed by three fully connected classification layers. Convolutional kernels of size 3 and 7 are used, as well as simple max-pooling layers. Another interesting aspect of this paper is the introduction of several large-scale data sets for text classification. We use the same experimental setting (see section 4.1). The use of character level information was also proposed by dos Santos and Gatti [5]: all the character embeddings of one word are combined by a max operation and they are then jointly used with the word embedding information. The overall architecture is quite shallow with two convolutional layers. Good results are reported on SST and on a very small subset of the Stanford Twitter Sentiment corpus (STS).

In the computer vision community, the combination of recurrent and convolutional networks in one architecture has also been investigated, with the goal to *"get the best of both worlds"*, e.g. [14]. The same idea was recently applied to sentence classification [19]. A convolutional network with up to five layers is used to learn high-level features which serve as input for an LSTM. The initial motivation of the authors was to obtain the same performance as Zhang et al. [20] with networks which have significantly fewer parameters. They report results very close to those of Zhang et al. [20] or even outperform ConvNets for some data sets.

In summary, we are not aware of any work that uses more than six convolutional layers for sentence classification. Deeper networks were not tried or they were reported to not improve performance. This is in sharp contrast to the current trend in computer vision where significant improvements have been reported using much deeper networks, namely 19 layers [15], or even up to 152 layers [7]. In the remainder of this paper, we describe our very deep convolutional architecture and report results on the same corpora than Zhang et al. [20]. We were able to show that performance improves with increased depth, using up to 29 convolutional layers.

3 Architecture

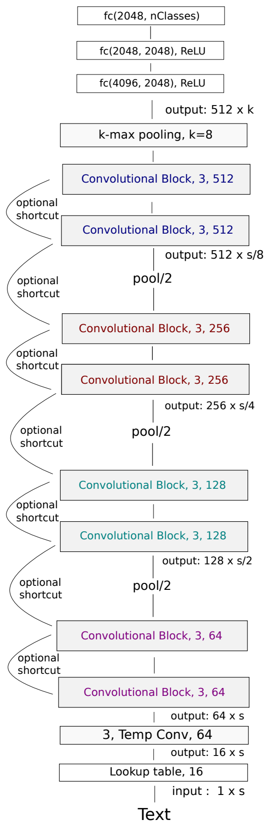

Figure 1: Global architecture with convolutional blocks. See text for details.

The overall architecture of our network is shown in Figure 1. Our model begins with a look-up table that creates a vectorial representation (embedding) of each character and transforms the input sentence into a two-dimensional tensor of size $f_0 \times s$, with f_0 the dimension of the embedding space and s the number of characters of the input text. In this work, the input is a fixed-sized padded text: f_0 can be seen as the equivalent of the "RGB" dimension of an image of size $1 \times s$. We first apply one layer of 64 convolutions of size $3 \times f_0$, followed by a stack of temporal "convolutional blocks". Inspired by the philosophy of VGG and ResNets we apply these two design rules: (i) for the same output temporal resolution, the layers have the same number of feature maps, (ii) when the temporal resolution is halved, the number of feature maps is doubled. Starting with 64 feature maps, we apply three pooling operations (halving the temporal resolution each time by 2), resulting in 3 levels of 128, 256 and 512 feature maps (see Figure 1). Different depths of the overall architecture are obtained by varying the number of convolutional blocks in between the pooling layers (see table 1). The output of these convolutional blocks is a tensor of size $512 \times s_d$, where $s_d = \frac{s}{2^p}$ with $p = 3$ the number of down-sampling operations. At this level of the convolutional network, the resulting tensor can be seen as a high-level representation of the input text. Since we deal with padded input text of fixed size, s_d is constant. However, in the case of variable size input, the convolutional encoder provides a representation of the input text that depends on its initial length s. Representations of a text as a set of vectors of variable size can be valuable namely for neural machine translation, in particular when combined with an attention model. In Figure 1, temporal convolutions with kernel size 3 and X feature maps are denoted "`3, Temp Conv, X`", fully connected layers which are linear projections (matrix of size $I \times O$) are denoted "`fc(I, O)`" and "`3-max pooling, stride 2`" means temporal max-pooling with kernel size 3 and stride 2.

Most of the previous applications of ConvNets to NLP use an architecture which is rather shallow (up to 6 convolutional layers) and combines convolutions of different sizes, e.g. spanning 3, 5 and 7 tokens. This was motivated by the fact that convolutions extract n-gram features over tokens and that different n-gram lengths are needed to model short- and long-span relations. In this work, we propose to create instead an architecture which uses many layers of small convolutions (size 3). Stacking 4 layers of such convolutions results in a span of 9 tokens, but the network can learn by itself how to best combine these different *"3-gram features"* in a deep hierarchical manner. Our architecture can be in fact seen as a temporal adaptation of the VGG network [15]. We have also investigated the same kind of *"ResNet shortcut"* connections as in He et al. [7], namely identity and 1×1 convolutions (see Figure 1).

For the classification tasks we deal with in this work, the temporal resolution of the output of the convolution blocks is first down-sampled to a fixed dimension using k-max pooling. By these means, the network extracts the k most important features, independently of the position they appear in the sentence. The $512 \times k$ resulting features are transformed into a single vector which is the input to a three layer fully connected classifier with ReLU hidden units and softmax outputs. The number of output neurons depends on the classification task, the number of hidden units is set to 2048, and k to 8 in all experiments. It may be relevant to adapt the number of hidden units in the classifier to the size of the training set (see Table 2), but we kept this parameter fixed in the experiments. It was also found to be better to not use drop-out with the fully connected layers, but only temporal batch normalization after convolutional layers.

Convolutional Block

Each convolutional block (see Figure 2) is a sequence of two convolutional layers, each one followed by a temporal BatchNorm [9] layer and an ReLU activation. The kernel size of all the temporal convolutions is 3, with padding such that the temporal resolution is preserved (or halved in the case of the convolutional pooling with stride 2, see below). Steadily increasing the depth of the network by adding more convolutional layers is feasible thanks to the limited number of parameters of very small convolutional filters in all layers. Temporal batch normalization applies the same kind of regularization as batch normalization except that the activations in a mini-batch are jointly normalized over temporal (instead of spatial) locations. So, for a mini-batch of size m and feature maps of temporal size s, the sum and the standard deviations related to the BatchNorm algorithm are taken over $|\mathcal{B}| = m \cdot s$ terms.

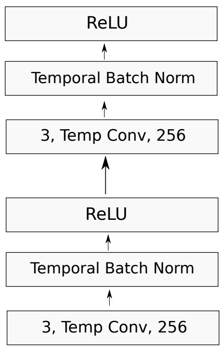

Figure 2: Detailed architecture of a convolutional block.

We explore three different types of down-sampling between blocks K_i and K_{i+1} (Figure 1) :

(i) The first convolutional layer of K_{i+1} has stride 2.

(ii) K_i is followed by a k-max pooling layer [11] where k is such that the resolution is halved.

(iii) K_i is followed by max-pooling with kernel size 3 and stride 2.

All these types of pooling reduce the temporal resolution by a factor 2. At the final convolutional layer, the resolution is thus s_d.

In this work, we have explored four depths for our networks: 9, 17, 29 and 49, which we define as being the number of convolutional layers. The depth of a network is obtained by summing the number of blocks with 64, 128, 256 and 512 filters, with each block containing two convolutional layers. In Figure 1, the network has 2 blocks of each type, resulting in a depth of $2 \times (2+2+2+2) = 16$. Adding the very first convolutional layer, this sums to a depth of 17 convolutional layers. The depth can thus be increased or decreased by adding or removing convolutional blocks with a certain number of filters. The best configurations we observed for depths 9, 17, 29 and 49 are described in Table 1. We also give the number of parameters of all convolutional layers.

Table 1: Number of convolutional layers for each depth.

Depth:	9	17	29	49
conv block 512	2	4	4	6
conv block 256	2	4	4	10
conv block 128	2	4	10	16
conv block 64	2	4	10	16
First conv. layer	1	1	1	1
#params [in M]	2.2	4.3	4.6	7.8

4 Experimental evaluation

4.1 Tasks and data

In the computer vision community, the availability of large data sets for object detection and image classification has fueled the development of new architectures. In particular, this made it possible to compare many different architectures and to show the benefit of very deep convolutional networks. We present our results on eight freely available large-scale data sets introduced by Zhang et al. [20] which cover several classification tasks such as sentiment analysis, topic classification or news categorization (see Table 2). The number of training examples varies from 120k up to 3.6M, and the number of classes is comprised between 2 and 14. This is considerably lower than in computer vision (e.g. 1 000 classes for ImageNet). This has the consequence that each example induces less gradient information which may make it harder to train large architectures. It should be also noted that some of the tasks are very ambiguous, in particular sentiment analysis for which it is difficult to clearly associate fine grained labels. There are equal numbers of examples in each class for both training and test sets. The reader is referred to Zhang et al. [20] for more details on the construction of the data sets. Table 3 summarizes the best published results on these corpora we are aware of. It is important to mention that we do not use "Thesaurus data augmentation" or any other preprocessing, except lower-casing the texts. Nevertheless, we still outperform the best convolutional neural networks of Zhang et al. [20] for all data sets. The main goal of our work is to show that it is possible and beneficial to train very deep convolutional networks for NLP. Data augmentation may improve our results even further. We will investigate this in future research.

Table 2: Large-scale text classification data sets used in our experiments. See Zhang et al. [20] for a detailed description.

Data set	#Train	#Test	#Classes	Classification Task
AG's news	120k	7.6k	4	English news categorization
Sogou news	450k	60k	5	Chinese news categorization
DBPedia	560k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	650k	50k	5	Sentiment analysis
Yahoo! Answers	1 400k	60k	10	Topic classification
Amazon Review Full	3 000k	650k	5	Sentiment analysis
Amazon Review Polarity	3 600k	400k	2	Sentiment analysis

Table 3: Best published results from previous work. Zhang et al. [20] best results use a Thesaurus data augmentation technique (marked with an *). Xiao and Cho [19] propose a combination of convolution and recurrent layers. n-TFIDF corresponds to the ngrams version of TF-IDF.

Corpus:	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
Method	n-TFIDF	n-TFIDF	n-TFIDF	ngrams	Conv	Conv+RNN	Conv	Conv
Author	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Zhang]	[Xiao]	[Zhang]	[Zhang]
Error	7.64	2.81	1.31	4.36	37.95*	28.26	40.43*	4.93*

4.2 Common model settings

The following settings have been used in all our experiments. They were found to be best in initial experiments. Following Zhang et al. [20], all processing is done at the character level which is the atomic representation of a sentence, same as pixels for images. The dictionary consists of the following characters “`abcdefghijklmnopqrstuvwxyz0123456789-,;.!?:’"/|_#$%ˆ&*˜‘+=<>()[]{}`” plus a special padding and unknown token which add up to a total of 72 tokens. The input text is padded to a fixed size of 1014. The character embedding is of size 16. Training is performed with SGD, using a mini-batch of size 128, an initial learning rate of 0.01 and momentum of 0.9. The learning rate is divided by two each time the validation error increases. We initialize our convolutional layers following He et al. [6]. The implementation is done using Torch 7. All experiments are performed on a single NVidia K40 GPU. An important difference to previous research on the use of ConvNets for text processing is the fact that we use temporal batch norm, but no dropout.

4.3 Experimental results

In this section, we evaluate several configurations of our model, namely three different depths and three different pooling types (see Section 3). Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with small temporal convolution filters with different types of pooling, which shows that a significant improvement on the state-of-the-art configurations can be achieved on text classification tasks by pushing the depth to 29 convolutional layers.

Our deep architecture works well on big data sets in particular, even for small depths
Table 4 shows the test errors for depths 9, 17 and 29 and for each type of pooling : convolution with stride 2, k-max pooling and temporal max-pooling. For the smallest depth we use (9 convolutional layers), we see that our model already performs better than Zhang’s convolutional baselines (which includes 6 convolutional layers and has a different architecture) on the biggest data sets : Yelp Full, Yahoo Answers and Amazon Full and Polarity. The most important decrease in classification error can be observed on the largest data set Amazon Full which has more than 3 Million training samples. We also observe that for a small depth, temporal max-pooling works best on all data sets.

Depth improves performance
As we increase the network depth to 17 and 29, the test errors decrease on all data sets, for all types

Table 4: Testing error of our models on the 8 data sets. The deeper the networks the lower the error for all pooling types. No data preprocessing or augmentation is used.

Depth	Pooling	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
9	Convolution	10.17	4.22	1.64	5.01	37.63	28.10	38.52	4.94
9	KMaxPooling	9.83	3.58	1.56	5.27	38.04	28.24	39.19	5.69
9	MaxPooling	9.17	3.70	1.35	4.88	36.73	27.60	37.95	4.70
17	Convolution	9.29	3.94	1.42	4.96	36.10	27.35	37.50	4.53
17	KMaxPooling	9.39	3.51	1.61	5.05	37.41	28.25	38.81	5.43
17	MaxPooling	8.88	3.54	1.40	4.50	36.07	27.51	37.39	4.41
29	Convolution	9.36	3.61	1.36	4.35	**35.28**	27.17	37.58	**4.28**
29	KMaxPooling	**8.67**	**3.18**	1.41	4.63	37.00	27.16	38.39	4.94
29	MaxPooling	8.73	3.36	**1.29**	**4.28**	35.74	**26.57**	**37.00**	4.31

of pooling (with 2 exceptions for 48 comparisons). Going from depth 9 to 17 and 29 for Amazon Full reduces the error rate by 1% absolute. Since the test is composed of 650K samples, 6.5K more test samples have been classified correctly. These improvements, especially on large data sets, are significant and show that increasing the depth is useful for text processing. Overall, compared to previous state-of-the-art, our best architecture with depth 29 and max-pooling has a test error of 37.0 compared to 40.43%. This represents a gain of 3.43% absolute accuracy. The significant improvements which we obtain on all data sets compared to Zhang's convolutional models do not include any data augmentation technique.

Max-pooling performs better than other pooling types
In terms of pooling, we can also see that max-pooling performs best overall, very close to convolutions with stride 2, but both are significantly superior to k-max pooling.

Both pooling mechanisms perform a max operation which is local and limited to three consecutive tokens, while k-max polling considers the whole sentence at once. According to our experiments, it seems to hurt performance to perform this type of max operation at intermediate layers (with the exception of the smallest data sets).

Our models outperform the state-of-the-art ConvNets
We obtain state-of-the-art results for all data sets, except AG's news and Sogou news which are the smallest ones. However, with our very deep architecture, we get closer to the state-of-the-art which are ngrams TF-IDF for these data sets and significantly surpass convolutional models presented in [20]. As observed in previous work, differences in accuracy between shallow (TF-IDF) and deep (convolutional) models are more significant on large data sets, but we still perform well on small data sets while getting closer to the non convolutional state-of-the-art results on small data sets. The very deep models even perform as well as ngrams and ngrams-TF-IDF respectively on the sentiment analysis task of Yelp Review Polarity and the ontology classification task of the DBPedia data set.

Going even deeper degrades accuracy. Shortcut connections help reduce the degradation
As described in He et al. [7], the gain in accuracy due to the the increase of the depth is limited when using standard ConvNets. When the depth increases too much, the accuracy of the model gets saturated and starts degrading rapidly. This *degradation* problem was attributed to the fact that very deep models are harder to optimize. The gradients which are backpropagated through the very deep networks vanish and SGD with momentum is not able to provide a correct minimum of the loss function. To overcome this degradation of the model, the *ResNet model* introduced shortcut connections between convolutional blocks that allow the gradients to flow more easily in the network [7].

We evaluate the impact of shortcut connections by increasing the number of convolutions to 49 layers. We present an adaptation of the ResNet model to the case of temporal convolutions for text (see Figure 1). Table 5 shows the evolution of the train/test errors on the Yelp Review Full data set with or without shortcut connections.[1] When looking at the column "without shortcut", we observe the same degradation problem as in the original ResNet article: when going from 29 to 49 layers, the training goes up from 29.57 to 35.54, and test error rate increases as well (35.28$\rightarrow$37.40). When using shortcut connections, we observe improved results when the network has 49 layers: both the training and test errors go down and the network is less prone to underfitting than it was without shortcut connections.

While shortcut connections give better results when the network is very deep (49 layers), we were not able to reach state-of-the-art results with them. We plan to further explore adaptations of residual networks to temporal convolutions as we think this a milestone for going deeper in NLP.

Exploring these models on text classification tasks with more classes sounds promising
Note that one of the most important difference between the classification tasks discussed in this work and ImageNet is that the latter deals with 1000 classes and thus much more information is back-propagated to the network through the gradients. Exploring the impact of the depth of temporal convolutional models on categorization tasks with hundreds or thousands of classes would be an interesting challenge and is left for future research.

[1]Pooling is defined as convolution with stride 2 in this case, as in the ResNet configuration.

Table 5: Evolution of the train/test error on the Yelp Review Full data set for all depths, and with or without shortcut connections (ResNet).

	without shortcut	with shortcut
depth 9	33.08 / 37.63	33.51 / 40.27
depth 17	30.85 / 36.10	35.80 / 39.18
depth 29	29.57 / 35.28	30.59 / 36.01
depth 49	35.54 / 37.41	32.28 / 36.15

5 Conclusion

We have presented a new architecture for NLP which follows two design principles: 1) operate at the lowest atomic representation of text, i.e. characters, and 2) use a deep stack of local operations, i.e. convolutions and max-pooling of size 3, to learn a high-level hierarchical representation of a sentence. This architecture has been evaluated on eight freely available large-scale data sets and we were able to show that increasing the depth up to 29 convolutional layers steadily improves performance. Our models are much deeper than previously published convolutional neural networks and they outperform those approaches on all data sets. To the best of our knowledge, this is the first time that the *"benefit of depths"* was shown for convolutional neural networks in NLP.

What makes convolutional networks appropriate for image processing is that the visual world is compositional. Pixels combine to form shapes, contours and profiles. With ConvNets, higher level features are derived from lower level features to form a hierarchical representation. The convolution filters of a ConvNet create patterns of increasing scale and complexity: first a contour, then a shape, a face, etc. An image and a small text have similar properties. Texts are also compositional for many languages. Characters combine to form n-grams, stems, words, phrase, sentences etc. These similar properties make the comparison between computer vision and natural language processing very profitable and we believe future research should invest into making text processing models deeper. Our work is a first attempt towards this goal.

In this paper, we focus on the use of very deep convolutional neural networks for sentence classification tasks. Applying similar ideas to other sequence processing tasks, in particular neural machine translation, should be interesting. Note that both tasks are conceptually different. On one hand, for sentence classification, the system needs to extract characteristic features while disregarding information which is not relevant for the classification task (e.g. morphology). In neural machine translation, on the other hand, all information in the sentence must be preserved. It needs to be investigated whether the proposed architecture has to be adapted to tackle sequence-to-sequence processing with deep convolutional encoders.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[2] Yoshua Bengio and Rejean Ducharme. A neural probabilistic language model. In *NIPS*, volume 13, pages 932–938, 2001.

[3] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, pages 160–167, 2008.

[4] Ronan Collobert, Jason Weston Léon Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, pages 2493–2537, 2011.

[5] Cícero Nogueira dos Santos and Maíra Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Coling*, pages 69–78, 2014.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *http://arxiv.org/abs/1603.05027v2*, 2016.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[10] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, pages 1700–1709, 2013.

[11] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *http://arxiv.org/abs/1404.2188*, 2014.

[12] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[13] Andrew Lamb and Michel Xie. Convolutional encoders for neural machine translation. In *WEB download*, 2016.

[14] Pedro O. Pinhero and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, 2014.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[16] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *EMNLP*, 2011.

[17] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Interspeech*, 2012.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[19] Yijun Xiao and Kyunghyun Cho. Efficient character-level document classification by combining convolution and recurrent layers. In *http://arxiv.org/abs/1602.00367*, 2016.

[20] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NIPS*, pages 649–657, 2015.