Deep convolutional network architectures for transfer learning in NLP

Ramy Ghorayeb* , **Hugo Martinez*** and **Ilan Palacci***
Ecole Polytechnique
{ramy.ghorayeb, hugo.martinez, ilan.palacci}@polytechnique.edu,

Abstract

Most NLP problems are solved through memory-based neural networks (RNN,LSTM). Even though they are very efficient, they are expensive to train and need a very large amount of data during training to perform well.
In this paper we present a simple deep CNN-based architecture, faster to train and easier to reuse. We will present its performances in the context of supervised topic modeling. We will show that we are able to get a good accuracy while training the model on very small datasets by leveraging other sources of data and reusing the acquired knowledge using transfer learning methods taken from computer vision.
To our knowledge, this is the first time that such methods have been applied to NLP.

Keywords: NLP, Topic Modeling, Deep Learning, Convolutional Neural Network, Transfer Learning

1 Introduction

Since the popularisation of the application of neural networks to vision [1] and NLP [2], Deep Learning have achieved remarkable results in every category of NLP tasks. They remain sources of revolutions today [3].

Convolutional neural networks or ConvNets are largely used in computer vision since MNIST [4]. The founding idea consists in applying filters extracting the main features from images and applying a classical feed forward model. The idea has beefed up since by the stacking of multiple layers of convolutions and pooling to create "deep" architectures [5]. Those very deep models will be experimented at Facebook in NLP [6] and are the main inspiration we used to construct our architecture.

The resolution of NLP problems has witnessed its first deep learning revolution through the introduction of new word representations [7]. Words are now commonly represented by their projection from their sparce unidimensional representation of length V (with V size of the vocabulary) to a dense reduced space with features representing the words semantic. The semantic resemblance if words is directly linked to the cosine distance of their representation in this new space [8]. These words embedding and the variants that succeeded [9] constitute a fundamental root in all NLP state-of-the-art approaches in NLP.

One question that still remains today is the way to represent a sentence, a.k.a a sequence of words containing syntaxical dependencies, and non trivial local and long-range semantics. The most common approach is the transformation of the sentence to a sequence of tokens and the application of recurrent neural networks [2].
A recurrent neural network is constituted of linear units (neurons) interacting non-linearly and for which there exist at least one cycle in the structure. The tokens are processed in their original order and the RNN records in memory the sequence in its internal states before predicting, which makes such architecture particularly adapted to local dependencies. Other variants have also shown their ability to memorize long term dependencies like LSTM [10] and GRU [11]. If those models have proven their efficiency, they also have many issues that research is today trying to solve.
First, their training is very long, because the elements of the sequences are dependants in a non-linear way, which forbids any parallelisation of the work on the length of the sequence. Secondly, transfer learning, meaning the transfer of the training and knowledge acquired on a dataset to another problem is difficult and rarely practiced aside the reuse of the space of word representation. RNN requires a complete retraining in order to understand the dependencies in the new vocabulary. It contrasts with the transfer learning in the case of CNN architectures, where the local practice is to simply keep the high layers and retrain only the last ones.
Some innovations in transfer learning exist today but are isolated to very specific sub-tasks of NLP [15]. We will be focusing on providing a very general architecture that can be used in all.

CNN-based models although rarely used in NLP have proven their efficiency in semantic parsing [12], search query

*equal contribution, order is alphabetical sorting

retrieval [16], sentence modeling [13], and other various traditional NLP tasks [17]. Most refined architectures can compete today with the best self-attention models, while being simpler and lighter [14]. The idea is to use 1D-convolution filters and apply them to model sentence semantic. The main advantages of CNN being: (1) they are faster to train thanks to parallelisation of tasks by filters and (2) they are more adapted to transfer learning thanks to the preservation of the weights of the filters and the re-training of the end layers only.

The outline of the paper is as follows: Section 2 gives a background and describe the neural network architectures we used. Section 3 describes the operational choices for experimentation, including the datasets, hyperparameters and statistical tools used. Section 4 presents the results of our experiments. Section 5 infers conclusions based on the presented work.

2 Architecture

In this section, we will present briefly the building blocs and architectures we used in our experiments.

2.1 1D Convolution

The 1D-convolution bloc for our baseline model is inspired by Collobert's model [17]. Let $x_i \in R^k$ projection of the i-th word of the sentence in the space of size k. We can represent $x_{1:N}$ sentence of length n by the following matrix

$$x_{1:N} = x_1 \oplus x_2 \oplus ... \oplus x_N$$

with $\oplus$ concatenation operator and $x_{i:i+j}$ concatenations of the words $x_i, x_{i+1}, ..., x_{i+j}$ We define the kernel convolution operation as the application of the kernel $w \in R^h k$ to $\{x_{1:h}, x_{2:h}, ..., x_{n-h+1}\}$ words windows of length h to obtain the feature map c as seen on Figure 1 such as:

$$c = [c_1, c_2, ..., c_{n-h+1}]$$

with $\forall i \in R^{n-h+1}$, f non-linear function and b bias term:

$$c_i = f(w \cdot x_{i:i+h-1} + b \qquad (1)$$

The formula corresponds in matricial to Korenecker's product $\forall i \in R^{n-h+1}$, $c_i = w \times x_{1+i,h+i}$

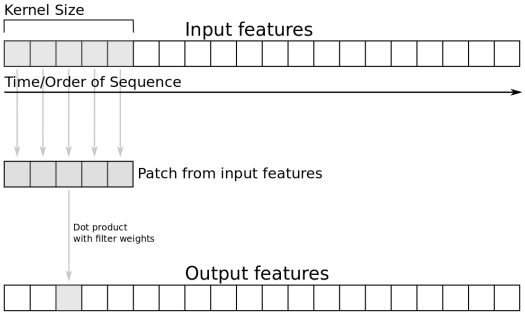


Figure 1: 1D-convolution process

To improve the results, multiple kernels per layer have been used. With D number of kernels of size (h, n) we can write the convolution process without bias for a given layer l as:

$$y_{i^{l+1},j^{l+1},d} = \sum_{i=0}^{H} \sum_{d^l=0}^{D^l} f_{i,d^l,d} \cdot x^l_{i^{l+1}+i,j^{l+1}+j,d^l} \qquad (2)$$

We then factor the features map by applying on each a max-overtime pooling operation [17] and take the value $\hat{c} = max\{c\}$. The space reduction enables a reduction of the variance of the feature maps and enable a better generalization of the model.

Character-level Convolution

The main sophistication to traditional architecture is the usage of convolutions on a character level. [18].

Words $(wi)_{n \in N}$ are not vectors anymore but rather 2-dimensional matrices of size (l, m). l is the length of the sentence, fixed so w_i is the matrix that joins all the characters vectors. If the sentence is longer than l then once l is reached, any remaining characters is ignored. m is the size of the alphabet so that each character can be vectorized using 1-of-m encoding, with a 1 on the row associated with the character and 0 elsewhere. We use a 70 characters alphabet with the 26 English letters, 10 digits 33 special characters and the new line break. (See Table 1). Any character not in the alphabet appears as a zero vector. You can view a representation of the process in Figure 2

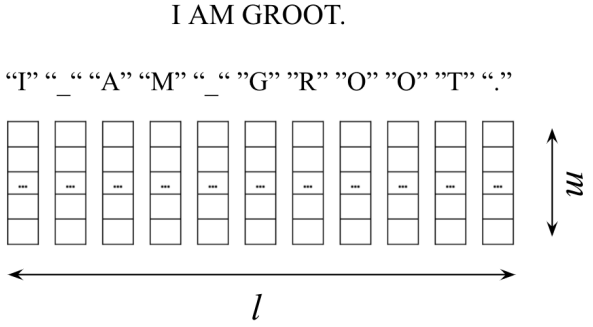


Figure 2: Character-level embedded sentence

We end up having the sentence represented as a tensor with the characters-level that can be seen as the "RGB" dimension of the input text.

Table 1: Alphabet used

Type	Characters	Size
Eng. letters	abcdefghijklmnopqrstuvwxyz	26
Digits	0123456789	10
Special char.	-,;.!?:"'/|_@$%& *^~ + − =<> ()[]{}	33
Other	Line Break	1
Total		**70**

Convolutional blocks

For our architectures, we use convolution blocks inspired by Facebook's [6].

The convolutional block is represented on Figure 3. It is a sequence of two 1D-convolution layers or temporal layers followed by a temporal BatchNorm [19] and a ReLU activation. Each convolution consists in a kernel of size 3 with a padding such that we keep the temporal length constant.

We then down-sample each block with a max-pooling in half operation.

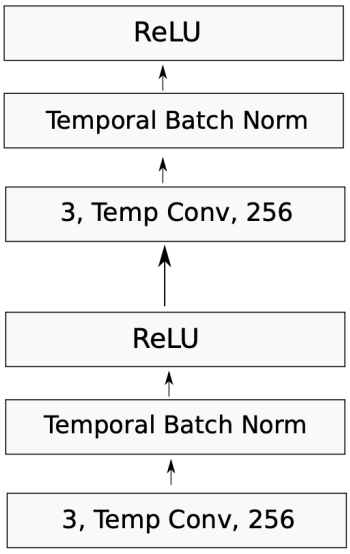


Figure 3: Convolutional block

VDCNN Architecture

The whole point of the architecture is to apply the computational training methodology for image classification. We use an architecture inspired by VGG and Resnet, where our sentences can be considered as 1D-images and the characters that compose each atomic section of the sentence the colors of each pixel of the image.

Our main deep convolutional architecture is inspired by Facebook's [6] and is shown in Figure 4:

- We first embed our sentence through a lookup table of size (s_0, f_0) with s_0 the length of our sentence (fixed to 1024) and f_0 dimension of our embedded character
- We then apply a layer of 64 convolutions of size 3
- We follow by c temporal convolution blocks with a half pooling operation at the end of each two blocks. On the Figure 4 we show a total of c=9 blocks and end up with a matrix of size $(512, s_0/2^3)$
- We follow by a k-max pooling layer with multiple k through our experiments
- We then apply two feed forwards layers activated by a ReLU function with u hidden units (u=2048 on the Figure 4)and a last one of size n with n the number of classes in our topic modeling dataset.

Transfer learning process

Once one architecture is trained on a dataset, we leverage it to accelerate the training on the rest of the datasets as following:

- We take the weights of the k-first convolutional layers of the architecture and apply them to the new architecture.
- We fix these k-first convolutional layers and update with through traditional gradient descend the rest of the weights through the epochs

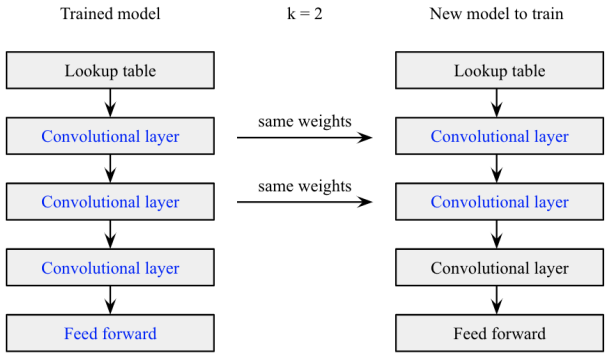


Figure 5: Transfer Learning process

3 Experimentation

3.1 Context

This paper has been done in an academic context. We are 3 engineers that graduated from 3 first tier French engineering schools. It was written under financial constraints so we have used the Google Colab's free cloud service based on Jupyter Notebooks that supports free GPU. To achieve our goal, we used TensorFlow 1.15, Keras 2.2.5 and standard Python/C libraries.

3.2 Tasks and datasets

We define a way to ensure an important classification accuracy on a small dataset with similar topics as a first large scale dataset. We compared the performance of our models, a simple deep CNN-based architecture applied to NLP classification, to an architecture based on a fastText embedding, a recurrent neural network and a feedfoward to which we will refer as the "RNN".

Hereby, we've trained the RNN and our models on a first large scale dataset introduced by [18] which cover several topic classification tasks. The Yahoo! Answers Topic Classification dataset covers 10 main categories of topic classification. We have extracted half of these categories and built a new dataset to train our models. These categories

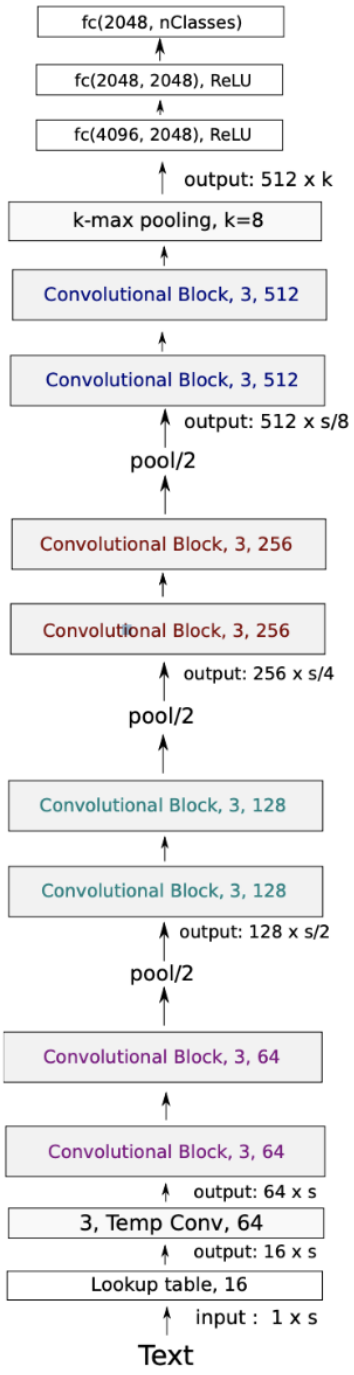


Figure 4: VDCNN architecture

Table 2: Yahoo! Answers topic classification dataset

Category	Train	Test
Health	20 000	2 000
Computer and Internet	20 000	2 000
Sports	20 000	2 000
Business and Finance	20 000	2 000
Entertainment and Music	20 000	2 000

are listed in the following table 2. In the Yahoo! Answers dataset, we have used the question and question' description that we concatenated to build our training dataset.

The Reddit Topic Classification Dataset Reddit is a network of communities based on people's interests. People can ask questions or open subjects to debate with other members. All of these topics called "subreddit" are publicly available to any web user willing to browse them or download them. We obtained our Reddit Topic Classification Dataset through Reddit's publicly available API. Our dataset contains the title of the subreddit post and it's description. These two elements are comparable to Yahoo! Answers dataset questions and question's description in their nature and structure. Our reddit topic classification dataset is divided according to the table 3

Table 3: Reddit Topic Classification Dataset

Category	Train	Test
Health	400	100
Computer and Internet	400	100
Sports	400	100
Business and Finance	400	100
Entertainment and Music	400	100

3.3 Model parameters

Parameters

- The initial setup of section 2.1.3 has been used in all our experiments with our corpus embedded at a character-level and of size (c_0, s_0, f_0) with c_0 size of our corpus, s_0 sentences length fixed to 1024 and f_0 character dimension fixed to 70.
- The first convolutional layer is fixed in all our experiments to (l_c, k_c) with l_c number of features maps fixed to 64 and k_c kernel size fixed to 3.
- Multiple number multiple temporal convolutions blocks c have been tested, with c in (9,17,27). blocks dimension (n_{cb}, l_{cb}, k_{cb}) is fixed to (2,256,3) with n_{cb} number of temporal layers, l_{cb} number of feature maps per temporal layer and k_{cb} kernel of each temporal layer.
- The k-max pooling layering that follows has been evaluated with k in $(2n)_{n \in R, n \leq 5}$
- Our ReLU feed forward block is of size (l, u) with l number of layers and u number hidden units with l fixed to 2 and u in $(2^n)_{n \in R, 8 \leq 11}$. We kept a low number of feed forward layer as our objective was not to find the

overall architecture that maximize the performance of the network. To find the ideal pre- feed forward architecture with the best transfer learning results, we keep the predictive power of our feed forward network to a minimum, hence the limited number of layers.

The parameters are summarized below in Table 4

Table 4: Parameters experimented

Layers	Parameter	Value
Corpus embedding	(s_0, f_0)	$(1024, 70)$
First convolution	(l_c, k_c)	$(64, 3)$
Nb of conv. blocks	c	$(9, 17, 27)$
Conv. blocks size	(n_{cb}, l_{cb}, k_{cb})	$(2, 256, 3)$
Max pooling	k	$(2n)_{n \in R, n \leq 5}$
Feed forward block	(l, u)	$(2, (2^n)_{n \in R, 8 \leq 11})$

Hyperparameters

- On our best architecture, we also experimented with the usage of shortcuts but decided to remove them as our backpropagated gradients were not converging to zero.
- We experimented keeping/removing all bias from our temporal convolutional layers.
- We tried applying dropouts of 0.2 and 0.4, without any performance improvements
- We trained our architecture on all datasets with a batch size of 64, and 10 epochs.
- The model is then compiled through TensorFlow and its built-in SGD optimizer with a learning rate in (0.01,0.001,0.0001) and a momentum of 0.9

The hyperparameters are summarized below in Table 5

Table 5: Hyperparameters experimented

Hyperparameter	Value
Shortcuts	$(True, False)$
Dropout	$(0, 0.2, 0.4)$
Bias	$(True, False)$
Batch size	64
Number of epochs	10
Optimizer	SGD (Tensor Flow)
Learning rate	$(0.1, 0.01, 0.001)$
Momentum	0.9

Paper architectures

You can find the specifics of our three best architectures in Table 6. These architectures will be the three we will use to evaluate our model in the Results section.

3.4 Experimental Results

In this section we compare the results of our CNN architecture to the standard RNN model. As mentioned before, we train all our models on the "Yahoo! Answers" dataset and tested the classifiers on the Reddit dataset, with and without transfer learning. We also trained the models on the Reddit dataset alone to analyze their ability to work on a small dataset. All the results of these trainings are represented in tables 8 and 7. Because of the context we mentioned previously we decided to focus our research on the depth-9 CNN, hence no results are provided for the other CNN architectures on the "Yahoo! Answers" dataset.

Table 7: Models Accuracy with training on "Reddit"

Model	Reddit
RNN	0.754
VDCNN-9	0.710
VDCNN-17	0.420
VDCNN-29	0.480

CNNs and RNNs both perform badly on small datasets compared to bigger datasets. In tables 8 and 7 we can see that both of the two models perform better on bigger datasets. This results is consistent with topic of this research. It strengthens our belief that reaching a good accuracy on a small dataset (size 3000) will hardly be possible through the use of conventional text classification models.

Our CNNs architecture does not beat RNN on text classification with big dataset. The results presented in table 8 show the accuracy of the different models on "Yahoo! Answers" test dataset. The RNN reaches a 0.890 accuracy against a 0.766 accuracy for our VDCNN with a training time 20 times smaller. This specific result is consistent with the current state-of-the-art in NLP, even though the accuracy obtained with the CNN is promising. The lack of time constrained us to reduce the optimization of the CNN's training time. One path of improvement for our project is to leverage the parallelization of the computation of convolutions and use multiple small instances in the same time into accelerate training.

No transfer learning but still good performance for CNNs. Results in table 8 demonstrate the good performance of both RNN and CNN when trained on "Yahoo! Answers" and used to predict "Reddit dataset" topics. It is worth mentioning the gain of performances of the CNN on a new dataset compared to the performance losses of the RNN. These results can be interpreted by drawing a parallel with the way CNNs classify images in computer vision. It is well-known CNNs detect the main shapes and geometrical specificities of an image to combine them non-linearly afterward and define the image label. Those architectures seem to understand "globally" what an image label means. In the context of text classification with char2vec preprocessing, the CNNs appear to understand globally what a topic means, by detecting semantic specificities in the corpus. Hence these models may perform better on new datasets than models understanding topic at a word level.

Table 6: Paper architectures

Name	Conv. blocks	Max pooling	FF block	Shortcuts	Dropouts	Bias	Learning Rate
VDCNN-9	9	8	(2,2048)	$False$	$False$	$False$	0.001
VDCNN-17	17	8	(2,2048)	$False$	$False$	$False$	0.001
VDCNN-29	29	8	(2,2048)	$False$	$False$	$False$	0.0001

Transfer learning allows CNNs to perform even better on small datasets. By training the three last layers of our CNNs on the "Reddit dataset" we observe in table 8 a real improvement of the accuracy. This results echoes researches in computer vision, in which the usage of pre-trained models such as ImageNet, VGG or ResNet allow to build accurate models in a timesaving way and with less data. This is specific to CNN architectures where trained convolutional networks keep their accuracy when used on similar datasets.

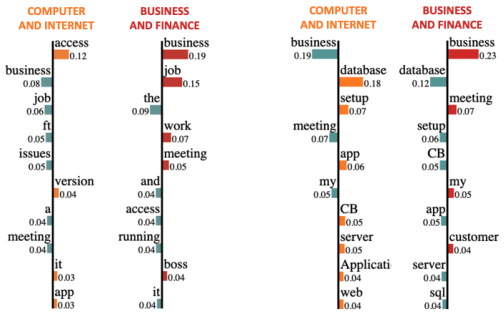
4 Interpretability

The models we worked on, like most deep learning algorithms, remain black boxes. We are able to assess their likeliness to predict the right topic for a sequence, but we cannot explain how it finds it. Understanding the reasons behind predictions is, however, quite important in assessing trust, which is fundamental if one plans to take action based on a prediction, or when choosing whether to deploy a new model or not.

In the context of NLP, a way to transform an untrustworthy model or prediction into a trustworthy one is to understand how the model makes its decision based on the words in the sequence. Moreover, such an analysis may help in comparing two models and the reasons behind their respective predictions. Indeed, as presented previously in the results, it appears that CNNs may have a different understanding of what a topic is than RNNs. This could explain the good performances of our transfer learning architectures we measured in our experiments.

To provide an interpretation of the models previously studied we used "LIME" (Local Interpretable Model-agnostic Explanations). LIME is a novel explanation technique that explains the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction. This technique was first introduced [21] and implemented as a Python library on Github by its authors.

Among the many functionalities this library provides, we decided to use the one explaining, on a chosen specific words sequence, the contribution of each word in the decision of our models. For the explainability, We chose the depth-9 VDCNN trained on "Yahoo! Answers" and fitted on the "Reddit" dataset with transfer learning, and the RNN model trained on "Yahoo! Answers". We chose this model of RNN as it is most accurate on the "Reddit" dataset. The output of this method are on Figure 6 and 7.



(a) RNN predicting "Business and Finance"

(b) VDCNN predicting "Computer and Internet"

Figure 6: Results comparison
Each vertical line represents the weights (positive or negative) of every word in the probability of the topic at the top. In this comparison, the VDCNN predicted the good topic for the text in figure 7.

Figure 6 represents for "Computer and Internet" and "Business and Finance" classes the weights that each word took in the decision process. The words on the right side of the line are positive, and the words on the left side are negative. Those weights are specific to the model and to the words sequence analyzed. Their visualization is available on Figure 7. We chose this sample as it was the most representative of our different comparisons where the two models where predicting different topics.

The two models seem to have a similar understanding of which words correspond to "Business and Finance". As an example, "business" and "meeting" are both present in the Top 5 of each models, and with similar weights. As for the "Computer and Internet" topic, the lexical field is way more developed in the VDCNN. We find words such as "database", "setup", "app" and "server" that are not present in the RNN model, and that allow our model to make the right prediction.

When comparing LIME results with other "Reddit" samples, where the two models were predicting the same topic, we figured out they were sharing 70% of their Top 20 words explaining the predictions made. More importantly these words were having really similar weights. This led us to consider that RNNs and CNNs may have a close understanding of what a topic is, as the predictions made

Table 8: Models Accuracy with training on "Yahoo! Answers"

* RNN's third column is empty as RNNs architecture do not allow transfer learning.

Model	Yahoo! Answers	Reddit without Transfer	Reddit with Transfer
RNN	0.890	0.782	-*
VDCNN-9	0.766	0.776	0.814

were for mostly the same reasons.

The two previous analysis show, in our context, that our CNN architecture "thinks" and "understands" the topics the same way RNN does, and fits better the lexical field of the small dataset which allows him to perform better on dataset-specific samples. Hence we can consider that our 9-depth VDCNN architecture improved with transfer learning is a better and sustainable alternative to RNN for small datasets.

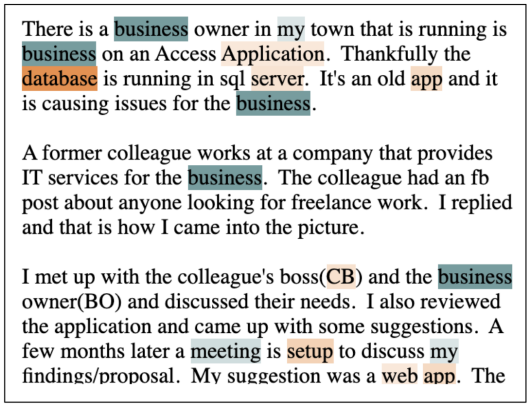


Figure 7: VDCNN results

6 *Green highlighted words go for "Business and Finance" and orange highlighted words for "Computer and Internet". These highlighting come from the VDCNN results.*

5 Conclusion

We presented a new CNN-based architecture for NLP outperforming RNN on small datasets thanks to transfer learning methods taken from computer vision. Our experiments demonstrate that convolutional networks should not be restricted to computer vision. We show that through the use of character-level embeddings, networks are developing parallel reasonings between pixels/shapes and characters/semantic. Their ability to leverage transfer learning is a real game changer when it comes to applying topic classification on really smaller datasets (3000).

Through this work we are able to provide an applied "process" for anybody willing to develop an accurate text classification model with only few hundreds of datapoint. Under the hypothesis that a bigger dataset with similar topics is available, one should focus on training a VDCNN on the biggest dataset and leverage transfer learning on the smaller dataset in order to adapt the classifier. Once the "small" dataset gets big enough, RNNs will be performing better than CNNs, but will be expensive and longer to train and not reusable. Convolutional Neural Networks are very relevant in a transitional regime and in a permanent regime for projects with small initial investments or a need for agility.

Because of the context and the financial constraints that come with it, we had to put aside of our experiments the deeper CNNs presented, even though [6] noticed that the deeper the CNN the more accurate the model is. The results obtained during these experiments strengthen our beliefs that it would be relevant to analyze the performances of deeper CNNs.

The promising results obtained during this research legitimize a generalization of the experimentation. To do so, the experiments should be replicated with different dataset and different NLP tasks.

References

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998.

[2] Alex Graves. Generating Sequences With Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850*, 2013.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutiona Neural Networks. *Neural Information Processing Systems Conference 25th*, 2012.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*, 2016.

[6] Alexis Conneau, Holger Schwenk, Yann Le Cun and Loic Barrault. Very Deep Convolutional Networks for Text Classification. *arXiv preprint arXiv:1606.01781*, 2017.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin. A Neural Probabilistic Language Model. In *Journal of Machine Learning Research 3rd, 1137–1155*, 2003.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*, 2017.

[10] Sepp Hochreiter, Jürgen Schmidhuber. Long Short-Term Memory. *Journal Neural Computation Volume 9 Issue 8 pages 1735-1780*, 1997.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*, 2014.

[12] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, Jianfeng Gao. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Association for Computational Linguistics 53th, 1321–1331*, 2013.

[13] Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom. A Convolutional Neural Network for Modelling Sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[14] Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019.

[15] Prajit Ramachandran, Peter J. Liu, Quoc V. Le. Unsupervised Pretraining for Sequence to Sequence Learning. *arXiv preprint arXiv:1611.02683*, 2016.

[16] Yelong Shen, Xiadong He, Jianfeng Gao, Li Deng, Grégoire Mesnil. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. In *Conference on Information and Knowledge Management, 110-111*, 2014.

[17] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa. Natural Language Processing (Almost) from Scratch. In *Journal of Machine Learning Research 12th, 2493-2537*, 2011.

[18] Xiang Zhang, Jumbo Zhao and Yann LeCun. Character-level Convolutional Networks for Text Classification. *arXiv preprint arXiv:1509.01626*, 2015.

[19] Sergey Ioffe and Christian Szegedy. Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning 32th, 448–456*, 2015.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. In *arXiv preprint arXiv:1603.04467*, 2016.

[21] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *arXiv preprint arXiv:1602.04938*, 2016.