Convolutional Neural Networks using separable filters

Petrescu Viviana
EPFL
viviana.petrescu@epfl.ch

Abstract

Convolutional Neural Networks (CNNs) achieved state-of-the-art performance on various computer vision tasks ranging from the large scale ImageNet object recognition challenge to segmentation in bio medical imaging. This technical report presents initial results on using CNN with separable filters for speeding up their execution.

1 Introduction

Although proven to be very powerful, CNN are much slower for both training and testing than their counter parts SVM or Random Forests. In the forward pass, the computational complexity of evaluating one image of size $W \times H$ with J filters of size $d_1 \times d_2$ is $O(WHJd_1d_2)$. Improving the training time would be very beneficial since it would allow for more configurations to be tried and for larger networks. However, increasing effort has been put also into speeding up only testing time, since training can be done offline.

2 Speeding up CNN

The most common approach for speeding up CNNs is to run them on the GPU, making use of the paralelism nature of the algorithm. Alternatively or combined, FFT can be used for the convolutional operations (for both training and testing)[7]. For testing, a significant speedup can be obtained in hardware by using FGPAs [6]. The major drawback of FGPAs is that they are harder to program for people who do not work in the field.

Recently,[3] [8] have showed that separable filters can be used to speed up convolutions with no loss in performance. In [8] a non separable filter bank is approximated with a smaller set of separable filters. Their approach is applied to several computer vision problems and MNIST among them. bla

In [3] they prove speedup of CNNs for two different schemes using separable filters, one of which is similar with our approach but uses a different optimization algorithm for obtaining the separable filters. If the $2D$ filters are decomposed into a set of separable $1D$ filters of rank K, the complexity per image becomes $O(WH(J + d_1 + d_2))$. Thus, we obtain a speedup if $K << \frac{Jd_1d_2}{J+d_1+d_2}$.

3 Separable filters

In our work, we consider the approach of [8], where the set of filters χ for one convolutional layer is approximated as a set of separable filters, as shown in Fig1 (from [4]).

We can obtain an approximation by minimizing the equation:

$$\underset{a,b,c}{\text{minimize}} \quad \|\chi - \sum_{r=1}^{R} a_r \circ b_r \circ c_r\|_2^2$$

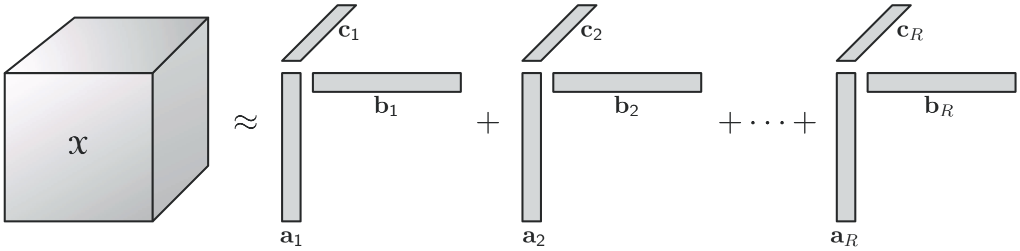


Figure 1: Tensor decomposition. A general 3D tensor $\chi \in \mathbb{R}^{I \times J \times K}$ is approximated by a set of separable filters of rank R

Tensor size	Typical rank
$I \times J \times K$ with $JK \leq I$ (very tall)	JK
$I \times J \times K$ with $JK - J < I < JK$ (tall)	I
$I \times J \times K$ with $I = JK - J$ (compact)	$I, I+1$

Table 1: Typical rank for selected tensor sizes.

We optimize the above equation using a Matlab implementation of the Canonical Polyadic decomposition (a generalization of SVD for tensors) with non linear conjugate gradient method. The only parameter of the tool is the rank R, which is almost never known. R represents the number of separable filters used to approximate the original filter bank. When the rank used for decomposition is not the theoretical one, the decomposition is not very accurate and the algorithm needs to be run with different initialization parameter until it converges. What is known is an upper bound on the theoretical rank R for a general tensor $3D$ tensor $\chi \in \mathbb{R}^{I \times J \times K}$, given by:

$$\text{rank}(\chi) \leq \quad min\{IJ, JK, KI\}$$

Besides the theoretical rank, there is also the notion of typical rank, which is any rank that appears with probability greater than 0, or that it is most common [4]. Some of the known typical ranks are shown in Table 1. Looking at the first row of Table 1, we conclude that for a 'very tall' set of filters like the ones present in large CNNs, we should expect the rank to be equal with the product of the two kernel dimensions.

4 Experiments

We used a Python implementation of CNN using the Theano library and experimented on two datasets, MNIST and Mitocondria Striatum. We varied the rank for every convolutional layer and compared the separable version with the non separable one. We recorded the time, the change in performance and how well the separable filters approximate the $2D$ filters.

4.1 MNIST

MNIST consists of a curated set of grayscale images (28x28) depicting handwritten digits. The dataset is split into 60.000 training samples, 10.000 validation and 10.000 testing samples. For this set, the approach of [2] using CNNs is the best model with a 0.23 error rate (23 out of 10.000 digits not recognized correctly). We start our experience with the reference Theano model for MNIST which achieves 0.82 error rate. Its configuration is shown in Table2

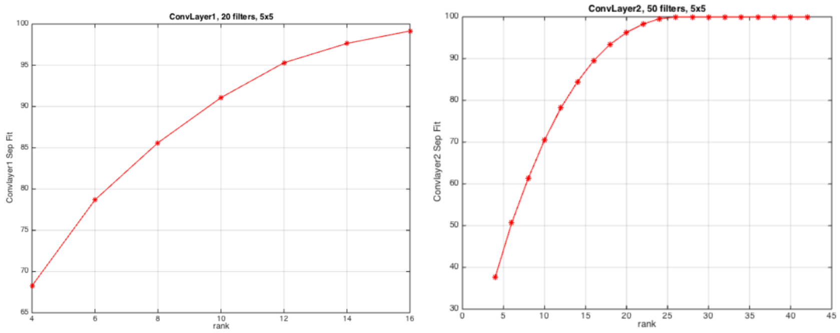
4.1.1 CNN Model 1

Layer	Type	Maps and neurons	Kernel size
0	input	1 map of 28x28	
1	convolutional	20 maps of 24x24	5x5
2	max pooling	20 maps of 12x12	
3	convolutional	50 maps of 8x8	5x5
4	max pooling	50 maps of 4x4	
5	fully conntected	500	
6	fully conntected	2 neurons	

Table 2: CNN Model 1 for MNIST

We decompose the filter map from the first convolutional layer. From the theoretical point of view, according to Table 1, we have a compact tensor $R^{20\times5\times5}$ with a typical rank of 20 or 21, so we expect a very good aproximation for a rank in that range. Fig2a shows how well is the approximation with varying rank from 4 to 16 in steps of 2 on the x axis and the fit on the y axis (100 corresponds to perfect fit). As expected, the fit is almost perfect for high ranks and decreases afterwards.

Using separable filters, we obtain a theoretical speedup for convolutional layer 1 if $K << \frac{Jd_1d_2}{J+d_1+d_2} = \frac{20\times5\times5}{20+5+5} = 16.66$. Fig3a shows the time improvements using separable filters. The blue line represents the time per layer using non separable filters. In our implementation, the separable filters give a speedup for any rank, as the red curve is below the blue one. We note that theoretically using non separable filters (blue line) and using separable filters with rank 16 should have given the same running time. The reason for the difference is due to implementation details and it is a language specific artifact.



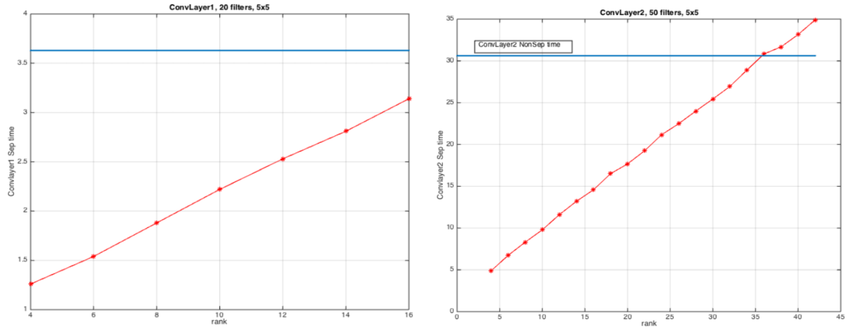
(a) Convolutional layer 1

(b) Convolutional layer 2

Figure 2: Rank versus Fit

The question that remains is to see how far we can afford to reduce the rank such that we do not lose much from the classification accuracy. Fig 4 a shows how the performance of the CNN drops with decreasing rank. With rank 16 the performance is the same, while for rank between 10 and 16 the error rate stays between 0.8 and 0.9 per cent, which is quite low. According to the application, even a rank of 8 or 6 is acceptable, since the drop is not more than 1 per cent.

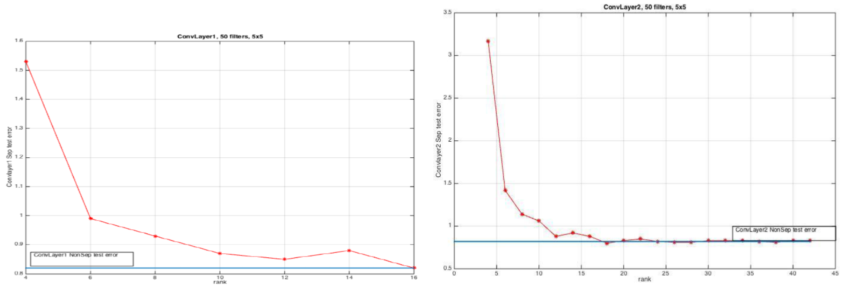
In the second experiment, we kept the first layer unchanged and approximated the second convolutional layer. Similarly, according to Table 1, we have a very tall tensor $R^{50\times5\times5}$ with a typical rank



(a) Convolutional layer 1

(b) Convolutional layer 2

Figure 3: Rank vs Time



(a) Convolutional layer 1

(b)

Figure 4: Convolutional layer 2

of 25. Fig2b shows how well is the approximation with varying rank from 4 to 42 in steps of 2 . As expected, the fit is almost perfect (99.9 fit) for rank higher than the theoretical one of 25.

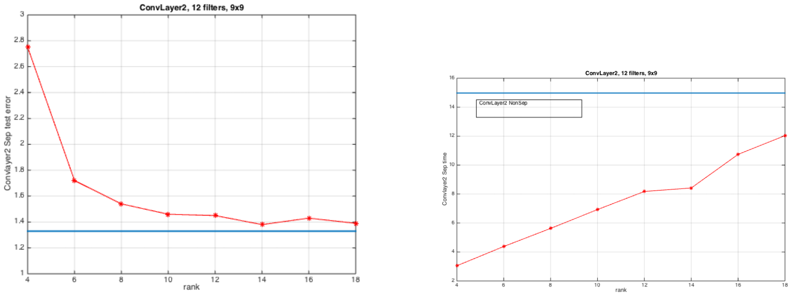
From the complexity perspective, the separable CNN is faster if we use a rank $K << \frac{Jd_1d_2}{J+d_1+d_2} = 20.83$. In our case, for rank = 20 we obtained almost a 40% speedup. This is due as before to language specific implementation. For rank 10, using separable filters is actually 3 times faster (theoretically 2 times faster). If we keep the rank greater than 10, the recognition never drops more than 1 error rate. For rank 12 the fit is 79 but the error rate is still almost unchanged which means we can easily use rank with relatively bad fit and still obtain good performance. Using separable filters of similar fit of 79 we obtained a slitghly lower performance. This might imply that it is more important to approximate well the first layer, which is the building block for the following layers.

4.1.2 CNN Model 2

In the following experiment, we keep the same number of filters per layers but increase the kernel sizes to 9×9 Fig3. We obtained a slightly lower performance for this set 1.3 error rate (compared with 0.82 for the previous model). In [cite here find paper] they claim that using larger filters does not actually improve peformance of cnns since they struggle to learn bigger filters, but investigating this issue is not the goal of the report. Here we only approximate convolutional layer 2 (rank from 4 to 18 with corresponding fit between 45 and 90). The theoretical rank is less than 49 and we obtain speedup if the rank $K << 59.5$ The results can be seen in Fig5. We notice that up to rank 8 the error of the Separable CNN is reasonable and does not drop more than 1.6 (1.3 is the reference error of the non separable CNN). The speedup for conv L2 drops from 15ms to 12 ms for rank 18 , which is actually much smaller than we expected for using larger filters.

Layer	Type	Maps and neurons	Kernel size
0	input	1 map of 28x28	
1	convolutional	20 maps of 24x24	9x9
2	max pooling	20 maps of 12x12	
3	convolutional	50 maps of 8x8	9x9
4	max pooling	50 maps of 4x4	
5	fully conntected	500	
6	fully conntected	2 neurons	

Table 3: CNN Model 2 for MNIST



(a) Error vs Rank

(b) Rank vs Time

Figure 5: Convolutional layer 2

Layer	Type	Maps and neurons	Kernel size
0	input	1 map of 51x51	
1	convolutional	10 maps of 46x46	6x6
2	max pooling	10 maps of 23x23	
3	convolutional	20 maps of 18x18	6x6
4	max pooling	20 maps of 9x9	
3	convolutional	50 maps of 4x4	6x6
4	max pooling	50 maps of 2x2	
5	fully conntected	100	
6	fully conntected	2 neurons	

Table 4: CNN for Mitocondria set

		Conv L1	Conv L2	Conv L3
NonSep	Kernel Size	10 maps 6x6	20 maps 6x6	50 maps 6x6
	Speedup rank	≤16.36	≤22.5	≤29.03
	Theoretical rank	16.4	≤36	36
	time	8.1	26.8	18.4
Separable	rank	-	20	36
	time	-	20.1	17.6

Table 5: CNN for Mitocondria set

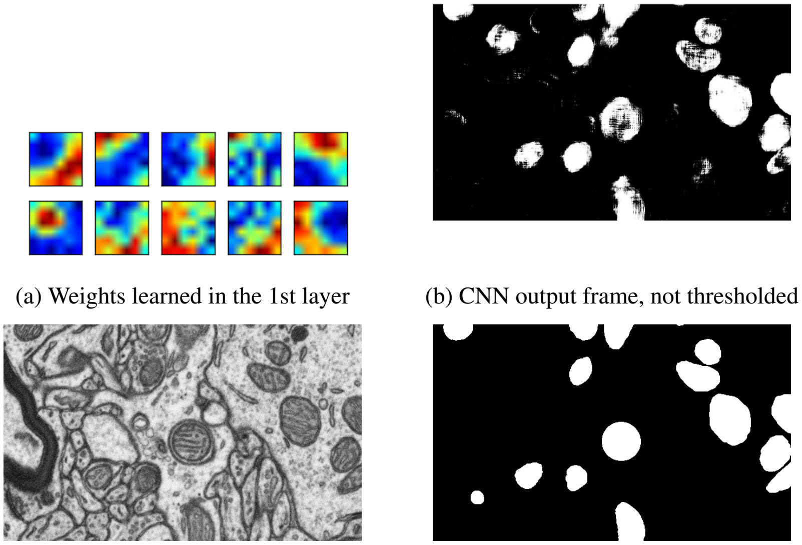
4.2 Mitocondria Striatum

For the Mitocondria Striatum set, CNN are employed to solve a segmentation problem. Every pixel is classified as being part of the mitocondria or not. A patch of 51x51 surrounding the pixel is extracted from the image and fed as input to a CNN which acts as a binary classifier. For determining the parameters of the CNN we used a set of 100.000 samples for training and 20.000 for validation (containing half positive and half negative samples). After determining the best setup, the network was trained on a larger set of 1 milion samples for training and 200.000 for validation.

The final test data represented a 3D image volume consisting of 318 slices of size 400x661. Thus, one frame contains 264.400 datasamples that are fed as test input for the CNN, while all frames contain 97 million datasamples.

The best setup obtained using the small training and validation set was the one presented in Fig5 NonSep. This gave us a VOC error on the first frame of 77.2. We then trained this net on the big set and obtained an error of 74. We notice that this is not comparable with state of the art methods which achieve results higher than 79 VOC or that use 3D information. The goal of this is to see if we can obtain a speedup by using separable filters. Example of the weights learned in the first convolutional layer are shown in Fig[ref] and of output on one frame produced by the CNN. We keep conv L1 fixed and approximate conv L2 and L3 with rank 20 and 36 respectively. For the 3rd layer no significant change in performance is observed, while for layer 2 we notice a small speedup from roughly 26 to 20 ms. Since the fit of the two layers was very good (99.9) there was no drop in classification performance.

Fig 6 shows an example of the CNN input and output and the type of filters that it learns.



(a) Weights learned in the 1st layer

(b) CNN output frame, not thresholded

(c) CNN input frame

(d) Groundtruth frame

Figure 6: Input and output example of CNN for mitocondria dataset. Fig a) shows that the filters are mostly edge and circle detectors which constitute basic building blocks for the interior of the mitocondria. Comparing b), c) and d) we notice the 'noisy' nature of the CNN output for pixel classification. Preprocessing the final output b) using a median filter would likely improve the results.

4.3 Pretrained ImageNet CNN

We investigated how well the separable filters approximation works for large filter banks, like the ones present in deep CNN models for ImageNet. The network setup in Table6 is the reference net (without the split to be run on two GPUs) from the initial breakthrough of CNNs in ImageNet [5] and it was trained in [1].

Fig7 shows the fitness of the approximation (mean and variance) with varying rank for the first four convolutional layers.

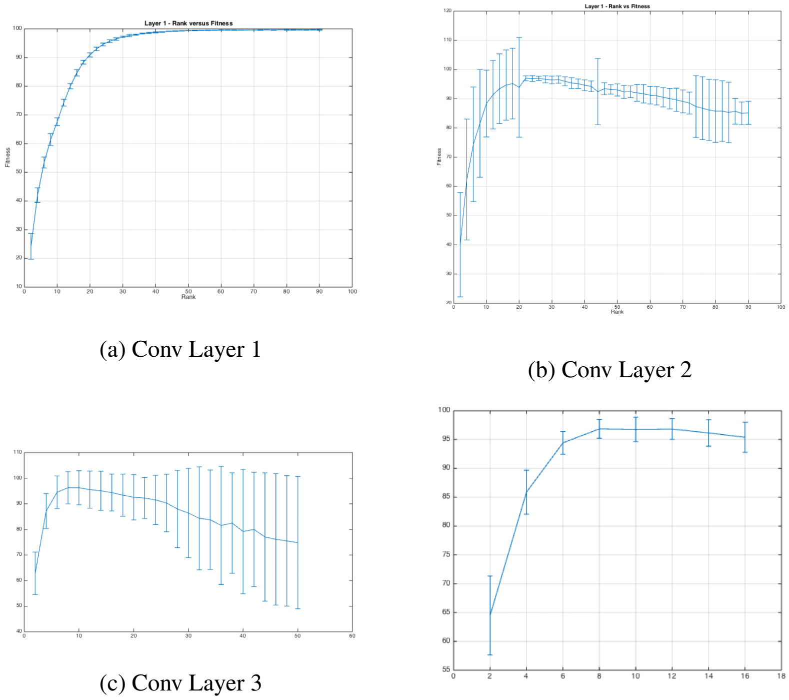
We notice how that the approximation works very well for Conv L1 (64 filters of size 11x11) which have a theoretical rank of almost 50 and we obtain a theoretical speedup if we use a rank smaller than $\frac{64\times11\times11}{64+11+11} = 90$. If we approximate filters of larger size the speedup is in general bigger.

Conv L2 has a theoretical rank of 25 and we obtain a speedup if we use a rank $\leq$ 24. We noticed that if we try to approximate the initial filters with a rank smaller than 20 the algorithm does not always converge like in all previous experiments. This was also the case for large ranks, for which we needed to rerun the algorithm with diffferent random intialization until it convergenced.

Conv L3 and L4 have a theoretical rank of 9 and we to obtain a speedup we need to use a rank $\leq$ 8.7. In order to obtain a 2-fold speedup, we would need to approximate conv L3 and L4 with rank 4, for which the fit is 86.

Arch	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
CNN-F	64x11x11	256x5x5	256x3x3	256x3x3	256x3x3	4096	4096	1000

Table 6: Pre trained ImageNet model



(a) Conv Layer 1

(b) Conv Layer 2

(c) Conv Layer 3

(d) Conv Layer 4

Figure 7: Rank versus Fit for CNN Layers of ImageNet model. Conv L1 can have a theoretical 2 fold speedup while the approximation is almost perfect (rank aprox 45 with fit 99.9). For the following layers, the optimization algorithm starts to struggle and needs to be rerun multiple times. For a similar 2-fold speedup per layer, we would need to use a rank for which the fit is less than 90.

5 Conclusions

Layer 1 more easy to be approximated but affects more the performance if the fit is not perfect. Following layers are more difficult to approximate (harder converges), but on MNIST accuracy of the fit was not so important as for the first layer. On all methods, relearning of the weights needs to be investigated.

Acknowledgments

I would like to thank Simon Arosi that helped me a lot during this project giving me constant feedback.

References

[1] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014.

[2] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[3] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference*, 2014.

[4] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

[6] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE, 2010.

[7] Michaël Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *CoRR*, abs/1312.5851, 2013.

[8] Amos Sironi, Bugra Tekin, Roberto Rigamonti, Vincent Lepetit, and Pascal Fua. Learning separable filters. pages 94–106, 2015.