ADAPTIVE CONVOLUTIONAL NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

The quest for increased visual recognition performance has led to the development of highly complex neural networks with very deep topologies. To avoid high computing resource requirements of such complex networks and to enable operation on devices with limited resources, this paper introduces adaptive kernels for convolutional layers. Motivated by the non-linear perception response in human visual cells, the input image is used to define the weights of a dynamic kernel called Adaptive kernel. This new adaptive kernel is used to perform a second convolution of the input image generating the output pixel. Adaptive kernels enable accurate recognition with lower memory requirements; This is accomplished through reducing the number of kernels and the number of layers needed in the typical CNN configuration, in addition to reducing the memory used, increasing 2X the training speed and the number of activation function evaluations. Our experiments show a reduction of 70X in the memory used for MNIST, maintaining 99% accuracy and 16X memory reduction for CIFAR10 with 92.5% accuracy.

1 INTRODUCTION

Convolutional Neural Networks (CNN) have demonstrated their capacity to achieve state of the art accuracy in image classification, semantic segmentation, and object detection. In order to increase the accuracy of recognition, most of these works rely on deeper and deeper architectures with millions of parameters. The reason behind this is that more complex features can be abstracted as we add more layers to a network.

Previous research has demonstrated that the response of visual cells is a non-linear function of their stimuli (Szulborski & Palmer, 1990). Thus, finding non-linear models that best represent data is imperative. Given that a typical convolutional layer is a linear system, its ability to express this response is limited by the layers and the number of neurons in the intermediate layers. The use of a non-linear neuron, as it was firstly explored to solve the XOR problem by Minsky & Papert (1969), (which cannot be solved by a first order neuron, but it can by a second order neuron), seems an appropriate way to tackle this kind of non linearity issue. Ideally, such non-linear approaches should be able to provide similar accuracy as compared to traditional CNNs, albeit at a much lower computation and memory costs.

Motivated by this, we developed a convolutional kernel, that includes non-linear transformations obtaining similar results as the state of the art algorithms, while yielding a reduction in required memory up to 14x in the CIFAR10 (Krizhevsky, 2009) classification, and up to 70x for the MNIST classification. The main contributions of this paper are: (i) we present the non-linear convolutions designed for high visual classification accuracy under memory constraints; (ii) using the proposed convolutions, we present a network design that is partially pre-defined and is capable of completing self-definition during the pattern evaluation phase, including defining the convolutional kernels on the fly depending of the input pattern; (iii) we present a method to tackle problems associated with higher order neural networks like the saturation of the activation based on the accumulation of energy; and (iv) a method to constrain every new dynamically generated weight to a pre-known range defined by the activation function, for instance if hyperbolic tangent function was used, the dynamically generated weights would be all in $(-1, 1)$ range.

This work is organized as follows: Section 2 describes previous approaches, Section 3 explains in detail our proposed method including the adaptive kernels with feed-forward and backward prop-

agation, Section 4 shows the results for MNIST and CIFAR10. We end with our conclusions in Section 5.

2 RELATED WORK

Different ways of increasing the accuracy of neural networks have been addressed in the existing scientific literature, most of these rely on the use CNNs, as they are generalized linear models, and their level of abstraction is low (Lin et al., 2013). Some approaches have dealt with this low abstraction by having additional layers (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014), resulting in a considerable increase in the accuracy on different datasets, e.g. CIFAR10 (Krizhevsky, 2009) and ImageNet (Deng et al., 2009). However, although the network depth has shown a crucial importance in neural network performance, the difficult to train the network also increases, and moreover, the degradation problem is exposed (He et al., 2015).
A proposal to tackle this problem is the use of ResNet blocks He et al. (2015), which given a set of inputs X, with an associated label Y and a function $H(x)$ that maps X to Y, the networks define a building block $y = F(x) + x$, where $F(x)$ represents the residual mapping to be learned. This residual network allows to train a deeper network without being affected by the degradation problem, using a larger amount of layers and parameters.
In the work presented in Lin et al. (2013) a nonlinear function approximator is proposed as a solution to increase the level of abstraction. The typical convolutional kernel is replaced with a micro network, i.e. a non linear approximator. A multilayer perceptron is used as the instantiation of this micro network, and by sliding the micro network over the input in a similar manner as convolutional neural networks the feature maps are obtained.
In the approach of Brabandere et al. (2016), a dynamic filter module is introduced, where the filters used for the convolution are generated dynamically depending on the input. This module consists of two parts: a filter generating network, which generates sample specific parameters given an input; and the dynamic filtering layer, which applies the parameters to another input. Although the mentioned work presents similar components to our proposal, it is important to notice the differences between these: in Brabandere et al. (2016) a CNN is trained to get the convolutional kernels, and then another network is defined to generate such kernels, where this second network is trained independently. In our work, we define a second order convolutional kernel trained using a novel training rule, which is explained in detail in the next section.

3 METHOD

An Adaptive Kernel is a dynamic kernel that changes its weights by itself depending on the input image. An Adaptive Kernel can be viewed as an array of traditional kernels. For instance, each element in a 3x3 adaptive kernel is a 3x3 linear kernel, as shown in the Figure 1.

The convolution of each kernel element $Q_{u,v}$ with the window X of the input image generates a component of the new kernel $K_{u,v}$ after applying the activation function (Figure 1). This new dynamically generated kernel K is convolved again with the same window X of the input image (Figure 2).

As result of this second order convolution and the activation function an output pixel value is obtained. By sliding the window through all the input image, we generate the final filtered image.

3.1 FEED FORWARD

Convolving the kernel $Q_{u,v}$ in the Figure 1 and using $tanh$ as activation function the weights $K_{u,v}$ are generated

$$K_{u,v}(\sigma) = tanh(\sigma_{u,v}), \tag{1}$$

where $\sigma_{u,v}$ is the convolution of each linear kernel $Q_{u,v}$ with the input image

$$\sigma_{u,v} = \sum_{i=0}^{N} \sum_{j=0}^{N} (Q_{(u,v)_{i,j}})(x_{i,j}), \tag{2}$$

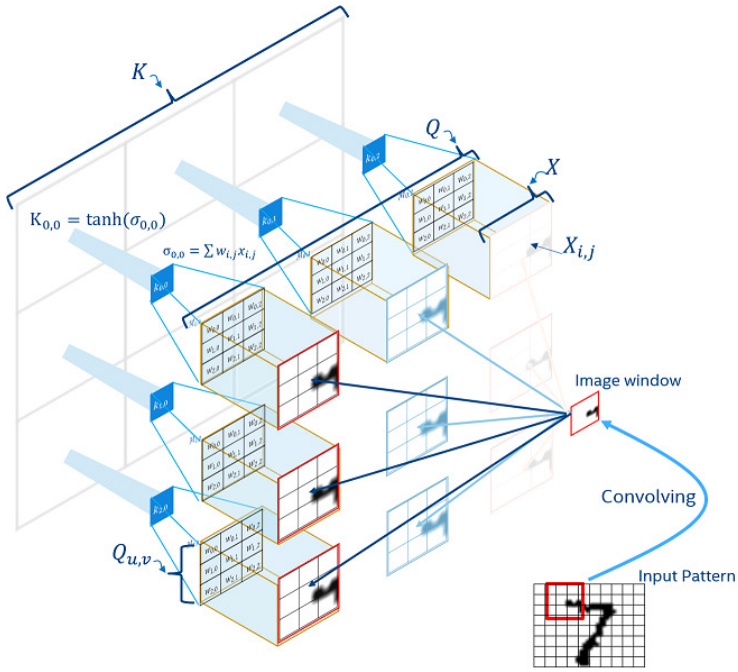

Figure 1: One dynamic kernel created by the convolution of the input image with a matrix of kernels.

This new kernel $K_{u,v}$ is now convolved with the input image to compute the accumulation of energy in this location

$$S = \sum_{u,v} x_{u,v} K \left(\sum_{i,j} Q_{u,v_{i,j}} x_{i,j} \right) \tag{3}$$

and finally the output pixel is computed using hyperbolic tangent as activation function $f = tanh(S)$.

3.2 Training

For the Adaptive kernel a new training rule is obtained using gradient descent technique in order to adjust the weights of adaptive kernel Q. The kernel can be seen as an array of nxn traditional kernels, using $Q_{u,v}$ to refer to each linear kernel and using (i, j) to refer the elements (weights) of each $Q_{u,v}$ kernel. It means the element $Q_{(u,v)_{(i,j)}}$ is a scalar value and represents a weight.

The kernel output $f(S)$, where S represents the accumulation of energy as shown in Equation (3). The error for output pixel (k, l) denoted as $E_{k,l}$ is given by:

$$E_{k,l} = \frac{1}{2}(d_{k,l} - f(s)_{k,l})^2, \tag{4}$$

where $d_{k,l}$ represents the expected output at coordinates (k, l). The training rule for the $t - th$ iteration (Q^t) is given by the error derivative per component $(u, v)_{i,j}$:

$$\frac{\partial E_{k,l}}{\partial Q^t_{(u,v)(i,j)}} = (d_{k,l} - f(s)_{k,l}) \frac{\partial f(s)_{k,l}}{\partial s_{k,l}} \frac{\partial S_{k,l}}{\partial Q^t_{(u,v)(i,j)}} \tag{5}$$

The gradient descent updates of the weights $Q^t_{(u,v)_{i,j}}$ are given by:

$$Q^{t+1}_{(u,v)(i,j)} = Q^t_{(u,v)(i,j)} + \gamma \sum_{k,l} (d_{k,l} - f(s)_{k,l}) \frac{\partial f(s)_{k,l}}{\partial Q^t_{(u,v)_{(i,j)}}} \tag{6}$$

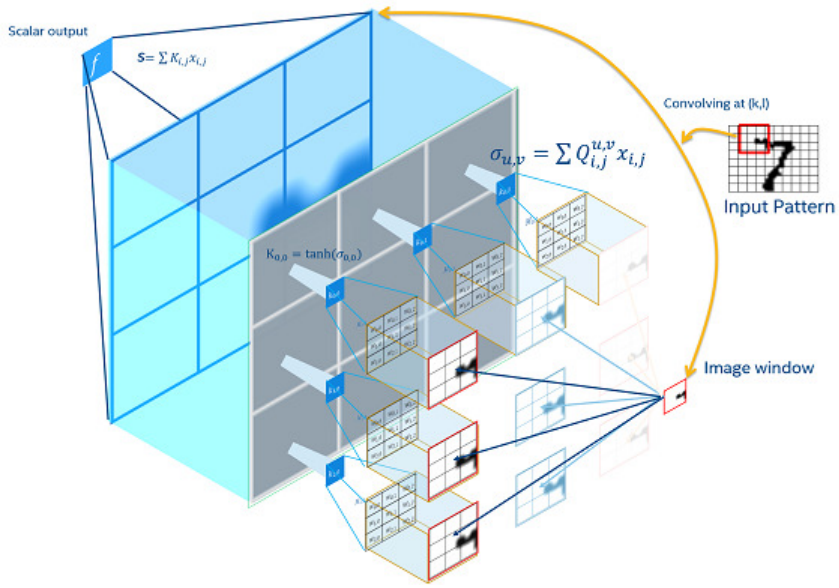

Figure 2: A single resulting pixel output from the convolution of the input image and the high order kernel that generates a dynamic kernel which is convolved with the input image again.

where γ represents the learning rate. Using the hyperbolic tangent as the activation function $f(S)_{k,l} = \tanh(S_{k,l})$:

$$\frac{\partial f(S)_{k,l}}{\partial S_{k,l}} \frac{\partial S_{k,l}}{\partial Q^t_{(u,v)_{(i,j)}}} = (1 - f(S)^2_{k,l}) \frac{\partial S}{\partial Q^t_{(u,v)_{(i,j)}}}, \tag{7}$$

where $S_{k,l}$ is the accumulation of energy computed as:

$$S_{k,l} = \sum_{u=0}^{N} \sum_{v=0}^{N} (K^{k,l}_{u,v})(x_{u+k,v+l}). \tag{8}$$

Here, N is the size of the kernel, and $K^{k,l}_{u,v}(\sigma) = \tanh(\sigma^{k,l}_{u,v})$, with:

$$\sigma^{k,l}_{u,v} = \sum_{i=0}^{N} \sum_{j=0}^{N} (Q_{(u,v)_{i,j}})(x_{i+k,j+l}), \tag{9}$$

Since the hyperbolic tangent was used as the activation function, the gradient is:

$$\frac{\partial f(s)_{k,l}}{\partial Q^t_{(u,v)_{(i,j)}}} = (1 - f(S)^2_{k,l}) \sum_{u=0}^{N} \sum_{v=0}^{N} \left(1 - (K^{k,l}_{u,v}(\sigma))^2\right)(x_{u+k,v+l})(x_{i+k,j+l}) \tag{10}$$

by replacing (10) in (6), the final training rule is defined by:

$$Q^{t+1}_{(u,v)_{(i,j)}} = Q^t_{(u,v)_{(i,j)}} + \gamma \sum_{k,l} (d_{k,l} - f(S)_{k,l})(1 - f(S)^2_{k,l})\delta_{k,l} \tag{11}$$

where γ is the learning rate,$d_{k,l}$ is the desired value at the window position (k, l), f is the activation function and $\delta_{k,l}$ is

$$\delta_{k,l} = \sum_{u,v} (1 - (K^{k,l}_{u,v}(\sigma))^2)(x_{u+k,v+l})(x_{i+k,j+l}) \tag{12}$$

For this training rule $tanh$ was used as activation function to generate the kernel weights, it ensures all weight are in the range of $(-1, 1)$. Additionally, this layer allows the reduction of filters in the subsequent layers without affecting the performance of the network, on the experimental results different experiments will be used to compared against state of the art for MNIST and CIFAR10 to show a significant memory compression using the proposed models.

4 RESULTS

Our implementation of the adaptive kernels was written as a layer of Caffe (Jia et al., 2014) with forward and backward propagation. For CIFAR-10 (Krizhevsky, 2009) we use Nesterov (Nesterov, 1983) as training rule, with momentum set to 0.9, initializing the learning rate to 0.1, dropping it by a factor of 5 every 20 epochs, and a weight decay of 0.0001. We use the same weight initialization as He et al. (2015). A 32x32 crop is randomly sampled from a 40x40 image or its horizontal flip, with the per-pixel mean subtracted and divided by the channel standard deviation. For MNIST we use the same parameters, changing the learning rate to 0.01 and the weight decay to 0.0005. With no data augmentation or pre-processing.

4.1 EXPERIMENT 1: MNIST

The MNIST is a public data set, it consists of 60,000 28x28 grey scale images in 10 classes (hand-written numbers), with 6000 images per class. There are 50,000 training images and 10,000 test images in the official data. Here our approach has three main advantages: memory reduction, increment of accuracy outperforming the traditional model of CNNs, and the learning speedup. The results produced by the implementation of the MNIST digits recognition, show a big memory compression, using 70X less memory measured through parameter reduction, additionally high accuracy was achieved 2x faster. In the table 1a the LeNet neural network architecture used as reference is detailed[1] and table 1b describes our topology.

Table 1: Topology comparison for MNIST.

(a) LeNet Neural Network Topology

Layer	Units	Type
Layer1	20 Kernels	Conv $5x5$
Layer2	50 Kernels	Conv $5x5$
Layer3	500 Neurons	FC
Layer4	10 Neurons	FC

(b) Our Neural Network Topology

Layer	Units	Type
Layer1	5 Kernels	Adaptive $5x5_{5x5}$
Layer2	10 Kernels	Conv $5x5$
Layer3	20 Neurons	FC
Layer4	10 Neurons	FC

The (figure 3) shows a few examples of how the Adaptive kernel is changing according to the input window. For all the back ground pixels the kernels is simple neutral it does not extract any features there.

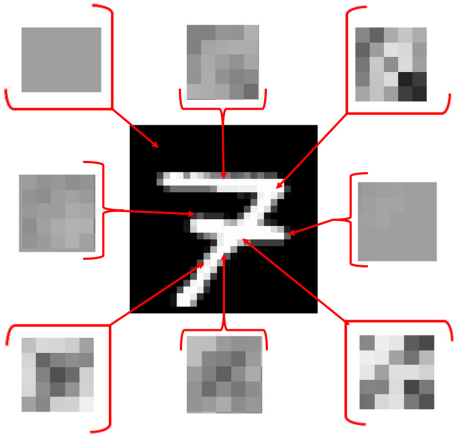

Figure 3: A single kernel generated in different positions of the input image.

[1]FC:Fully connected layer

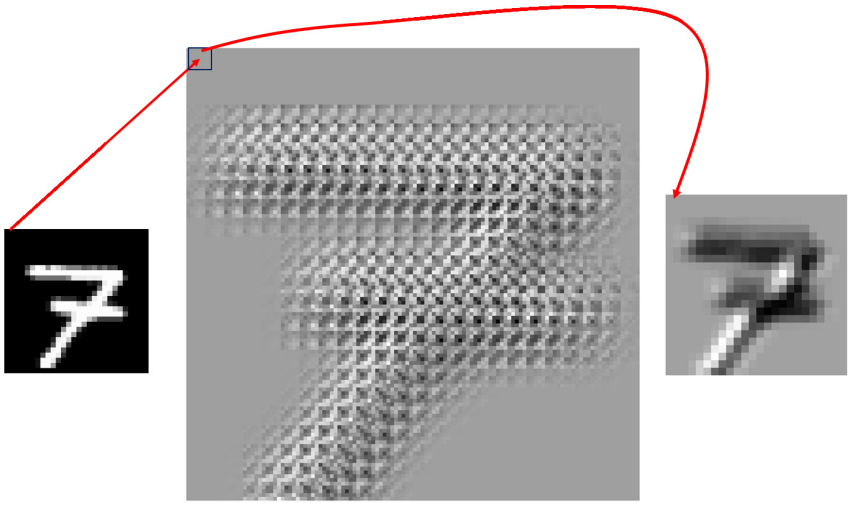

Figure 4: Every input window is convolved by a different filter generated on the fly using the input image.

The Figure 4 shows all the kernels generated for random sample of digit seven, The Adapts depending if it is background, a border or solid foreground. In this scenario our technique reaches 99% without any preprocessing in only 5 epochs, in contrast, the LeNet reached 97% after 9 epochs.

4.2 EXPERIMENT 2:CIFAR10

The CIFAR-10 is a public data set, it consists of 60,000 32x32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images in the official data. In this scenario CIFAR10 was used for testing and we used horizontal flipping, padding and 32x32 random cropping for data augmentation of the training dataset. In the selected topology (detailed in table 2) only the first layer uses adaptive kernels of $(3x3)_{(3x3)}$ in order to highlight the impact of a single layer in the full topology, being the first layer where the main feature extraction take place. It has 8 convolutional layers and ends with ten outputs in the fully connected layer.

Table 2: Our Neural Network Topology

Layer	Units	Type
Layer1	16 Kernels	Dynamic $(3x3)_{(3x3)}$
Layer2-9	16 Kernels	Conv $5x5$
Layer10	10 Neurons	FC

In order to compare against related work we include some of the latest topologies used for CIFAR10 pattern recognition problem. Our intention is not to outperform the accuracy of these topologies. Instead, the idea is to achieve similar results with a smaller model. This approach can potentially enable us to target embedded systems. In the Table 3 we provide a summary of the most popular methods recently used for CIFAR10. As the Table 3 shows the focus of the prior work has been the improvement of recognition accuracy without regard to the amount of memory and time required for classification.

The neural networks achieving the highest scores use over 20 million parameters, making it very hard to implement them in low capacity embedded devices. Our solution reduces memory usage; it only uses 200K parameters. While there is no simple way to determine the efficiency of a neural network, our target is highest accuracy with least amount of memory as shows (Table 3). Based on accuracy the Volterra-based Wide ResNet Zoumpourlis et al. (2017) provides the best score but uses

Table 3: CIFAR10 Classification error vs Number of parameters

Neural Network	Depth	#Parameters	Error%
All-CNN (Springenberg et al., 2014)	9	1.3M	7.25
ResNet Stochastic Depth Huang et al. (2016)	110	1.7M	5.23
Pre-act Resnet (He et al., 2016)	1001	10.2M	4.62
Wide ResNet (Zagoruyko & Komodakis, 2016)	40	55.8M	3.8
PyramidNet (Han et al., 2016)	110	28.3M	3.77
Wide-DelugeNet (Kuen et al., 2016)	146	20.2M	3.76
Steerable CNN (Cohen & Welling, 2016)	14	9.1M	3.65
ResNet Xt (Xie et al., 2016)	29	68.1M	3.58
Wide ResNet with Singular Value Bounding (Jia, 2016)	28	36.5M	3.52
Oriented Response Net (Zhou et al., 2017)	28	18.4M	3.52
Baseline wide ResNet	28	36.6M	3.62
Volterra-based Wide ResNet (Zoumpourlis et al., 2017)	28	36.7M	3.51
Adaptive Kernels CNN	10	**0.2M**	7.48

36M parameters. Our solution, the one with less number of parameters represents a compression of 184X with 4% drop in accuracy or comparing with All-CNN Springenberg et al. (2014) our solution represents a 6X memory reduction with 0.2% accuracy drop.

4.3 Conclusion

In this paper we presented adaptive convolutional kernels creating a new type of partially defined neural network, capable to redefine itself during the inference time depending on the input image. Our technique not only reduces memory usage, but also reduces the training time. Additionally, our results show adaptive convolutional kernels generalize better than CNN. Kernels adapt dynamically to extract better features depending of the input image, and in terms of efficiency, our method has shown to be very compelling generating 10X lighter solutions with less than 1% accuracy drop. Our solution should be able to provide direct impact to the computational cost of the inference on embedded systems increasing the operational scope of these systems. In terms of accuracy, our proposed dynamic kernel can imitate any traditional constant kernel (by training the dynamic kernel to be constant independent of the input); this means that it can be trained to generated the weights in the best in class solution and start from there to outperform it.

Acknowledgments

Many thanks to Anticipatory Computing Lab at Intel for supporting this work.

References

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. *CoRR*, abs/1605.09673, 2016. URL http://arxiv.org/abs/1605.09673.

Taco S. Cohen and Max Welling. Steerable cnns. *CoRR*, abs/1612.08498, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1612.html#CohenW16a.

Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *In CVPR*, 2009.

Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. *CoRR*, abs/1610.02915, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1610.html#HanKK16.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1603.html#HeZR016.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016. URL http://arxiv.org/abs/1603.09382.

Kui Jia. Improving training of deep neural networks via singular value bounding. *CoRR*, abs/1611.06013, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1611.html#Jia16a.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014. URL http://arxiv.org/abs/1408.5093.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pp. 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

Jason Kuen, Xiangfei Kong, and Gang Wang. Delugenets: Deep networks with massive and flexible cross-layer information inflows. *CoRR*, abs/1611.05552, 2016. URL http://arxiv.org/abs/1611.05552.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013. URL http://arxiv.org/abs/1312.4400.

Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady*, 27:372–376, 1983.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL http://dblp.uni-trier.de/db/journals/corr/corr1412.html#SpringenbergDBR14.

Robert G. Szulborski and Larry A. Palmer. The two-dimensional spatial structure of nonlinear subunits in the receptive fields of complex cells. *Vision Research*, 30(2):249 – 254, 1990. ISSN 0042-6989. doi: https://doi.org/10.1016/0042-6989(90)90040-R. URL http://www.sciencedirect.com/science/article/pii/004269899090040R.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. URL http://arxiv.org/abs/1611.05431.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.

Yanzhao Zhou, Qixiang Ye, Qiang Qiu, and Jianbin Jiao. Oriented response networks. *CoRR*, abs/1701.01833, 2017. URL http://arxiv.org/abs/1701.01833.

Georgios Zoumpourlis, Alexandros Doumanoglou, Nicholas Vretos, and Petros Daras. Nonlinear convolution filters for cnn-based learning. *CoRR*, abs/1708.07038, 2017. URL http://arxiv.org/abs/1708.07038.