Speeding up Convolutional Neural Networks By Exploiting the Sparsity of Rectifier Units

Shaohuai Shi
Department of Computer Science
Hong Kong Baptist University
Kowloon, Hong Kong
csshshi@comp.hkbu.edu.hk

Xiaowen Chu
Department of Computer Science
Hong Kong Baptist University
Kowloon, Hong Kong
chxw@comp.hkbu.edu.hk

Abstract

Rectifier neuron units (ReLUs) have been widely used in deep convolutional networks. An ReLU converts negative values to zeros, and does not change positive values, which leads to a high sparsity of neurons. In this work, we first examine the sparsity of the outputs of ReLUs in some popular deep convolutional architectures. And then we use the sparsity property of ReLUs to accelerate the calculation of convolution by skipping calculations of zero-valued neurons. The proposed sparse convolution algorithm achieves some speedup improvements on CPUs compared to the traditional matrix-matrix multiplication algorithm for convolution when the sparsity is not less than 0.9.

1 Introduction

Deep neural networks (DNNs) have been succefully used in many AI applications [16]. DNNs were proposed in the early 1982 [6]. With the help of the increasing computional ability of general-purpose processing units like GPUs, DNNs enjoy a fast expansion in recent years. One of the important reasons why DNNs have a high ability to fitting data is that its sparse representation of an input space [26]. The rectifier neuron units (i.e., ReLUs) contribute a lot to the success of deep neural networks [7, 20, 19, 16]. ReLUs can not only solve the vanishing gradient problem, but they also produce a sparse representation which leads to mathematical advantages of the network [20]. Given the sparse structure of DNNs, the networks have many zero operands. These zero operands can be neglected to reduce the magnitude of calculation since multiply and addition with zero are unmeaningful. Albericio et al. [1] have shown that the inputs of convolutional layers are on average 44% zeros based on recently popular deep learning architectures, and they propose a DNN accelerator named *Cnvlutin* to dymanically eliminate the most inefectural multiplications of zeros on the simulator of *DaDianNao* which is a state-of-the-art accelerator proposed by Chen et al. [4].

In this work, we also exploit the property of ReLUs to accelate the computation of DNNs, but in two different ways compared with [1]: First, we observe the sparsity of output values with ReLU activivation functions in the existing popular deep architectures during the training process but not limited to the inferencing phase. Second, we design the algorithm that eliminates the unused zero operands on-the-fly on traditional processors (i.e., CPUs) without changing the architecture of network. Our proposed convolution algorithm for the sparsity that is not less than 0.9 achieves performance improvements on convolutions of LeNet [17], AlexNet [14] and GoogLeNet [24] on CPUs compared to the traditional convolution algorithm.

2 Related Work

The sparsity of parameters have been widely exploited to compress and accelerate DNNs [10, 2, 9, 18, 8, 25]. DNNs with large size of parameters have been empirically proved that the parameters

exist heave redundancy which can be removed by sparse decompositions [18], sparse learning [25], continual network maintenance [8] and guided pruning [13] etc. This kind of methods focus on constructing the high sparsity of parameters and pruning unused connections to achieve the network compression and acceleration. It needs to carefully construct on the sparsity of parameters since, otherwise it may lead to the accuracy decrease of original networks. Though this work does not attempt to construct the sparsity of deep networks, the fact of high speedup in the sparse networks inspires us to consider the efficient convolution algorithms by skipping the calculation of zero. In fact, the ReLU activation functions set all negative values to zero, which means the input of the next layer has large propotion of zero values. This property gives oppotunities for energy saving [3, 21] and MAC reducing [1] by skipping the insignificiant calculation of zero. Reagen et al. [21] state that operations involving large number of zero or near zero values can save power both from avoided multiplications and SRAM accesses. Albericio et al. [1] show that there are on average 50% zero of convolutional inputs in seven state-of-the-art deep convolutional networks, and they design an efficient convolution accelerator by skipping the calculation of zeros. In terms of the high sparsity of input of convolution, few work was proposed to improve the efficiency of convolution on traditional computational units like CPUs and GPUs. In this work, we first demostrate the sparsity of deep neural networks whoes activation functions are ReLUs during the process of training. Second, instead of considering the particular hardware for convolution, we propose an efficient convolution algorithm that can eliminates the unused zero operands on-the-fly on CPUs without changing the archetecture of network.

3 Sparsity of ReLUs

In this section, we want to demostrate the propobility of zero outputs of a convolutional nueral network. ReLU is an activation function f with form of: $f(x) = max(0, x)$. Given a convolutional layer with C channels of input feature maps, K channels of output feature maps, the number of filters should be $C \times K$. Some notations are shown in Table 1, and we denote the convolution as the operation of high dimension tensors as shown in Fig. 1. The input $\mathcal{I}$ is a 3-mode tensor with size of $C \times H_{in} \times W_{in}$, the kernel (weight) $\mathcal{W}$ is a 4-mode tensor with size $K \times C \times R \times S$, and the output $\mathcal{O}$ is a 3-mode tensor with size $K \times H_{out} \times W_{out}$, where $H_{out} = \frac{(H_{in}+2\times P-R+1)}{T}$ and $W_{out} = \frac{(W_{in}+2\times P-S+1)}{T}$.

Table 1: Parameters of Convolution

Parameter	Meaning
C	# of input feature maps
H_{in}	Height of input image
W_{in}	Width of input image
K	# of output feature maps
R	Height of filter kernel
S	Width of filter kernel
H_{out}	Height of output image
W_{out}	Width of output image
P	Padding of input image
T	Stride of convolution

The output value at row i and column j of kth output channel is calculated as follows:

$$\mathcal{O}_{k,i,j} = \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \mathcal{W}_{k,c,r,s} \mathcal{I}_{c,i+r,j+s} \tag{1}$$

In general, the convolutional layer is followed by an activation layer (e.g., ReLU), whose dimension is same with the output of convolutional layer. Let $\mathcal{Z}$ denote the output of an ReLU layer, which is generally the input of the next convolutional layer in DNNs [24, 11]. We have:

$$\mathcal{Z}_{k,i,j} = max(0, \mathcal{O}_{k,i,j}) \tag{2}$$

In the training progress in practical, the sparsity of output of ReLUs is around 50% to 95%. We test the output sparsity of ReLUs in the original DNNs including LeNet [17] for MNIST dataset, AlexNet

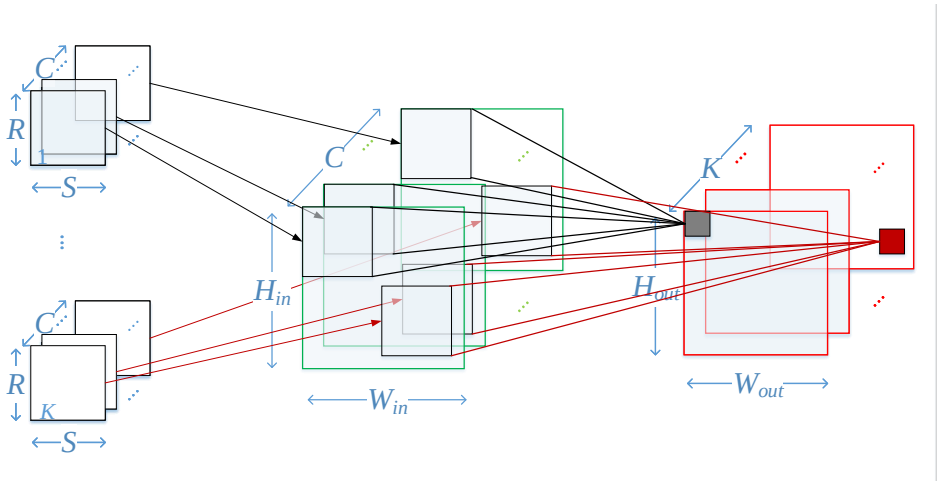


Figure 1: Convolution in CNNs

[14] for Cifar10 dataset, AlexNet [14] for ImageNet and GoogLeNet [24] for ImageNet. The training process runs 40 epochs for each network. At the end of each epoch, both the sparsities of all the ReLU Layers and validation accuracy are recorded. The final results are shown in Fig. 2. It is noted that many of ReLU layers generate the sparsity more than 0.7, and some of layers have up to the sparsity of about 0.95. In general, the output of an ReLU layer is the input the convolutional layer. In other words, the input of the convolution layer has up to more than 70% zero values in most of cases. So we have the evidence to propose the sparse convolution algorithm by skipping the multiplication of zero-valued neurons to accelerate the computation of convolution.

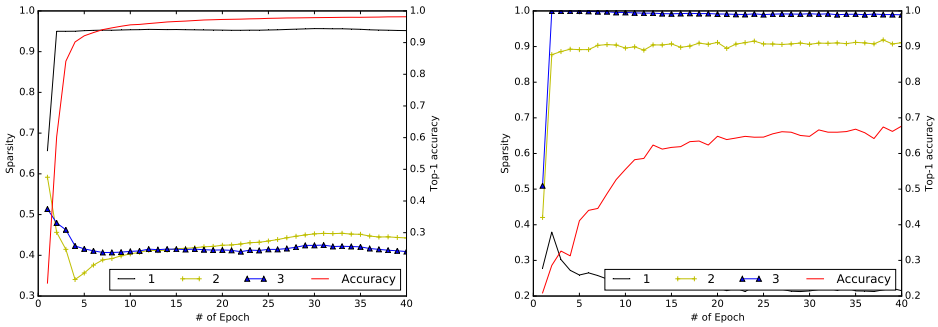
4 Sparse Convolution

A direct convolution operation in CNNs takes $H_{out} \times W_{out} \times R \times S \times K \times C$ floating-point multiplication and addition calculations (MACs in *FLOP*). And it needs $H_{out} \times W_{out} \times K$ memory write operations (MWOs) to store outputs. In order to exploit the advantage of SIMD techniques of processors (CPUs or GPUs), the calculation of convolutional layer is often mapped to matrix multiplication [5, 23]. So that it can make use of highly optimized BLAS libraries like Intel MKL on CPUs and cuBLAS on GPUs. We first have a look at the matrix multiplication algorithm to solve the convolution, and analysis its computation performance. After that, we derive our accelerated algorithm by skipping zero operations.

Matrix Multiplication for Convolution. Fig. 3 shows the matrix multiplication: $M_K \times M_I = M_O$ for convolution, where $M_K \in \mathbb{R}^{K \times RSC}$ is the matrix of kernels, $M_I \in \mathbb{R}^{RSC \times H_{out}W_{out}}$ is the unrolled matrix of input neurons and $M_O \in \mathbb{R}^{K \times H_{out}W_{out}}$ is the output of convolution. Note that the MACs and MWOs required by this algorithm is the same with the direct convolution method. It is easy to use SIMD and parallel techniques to accelerate matrix multiplication algorithm.

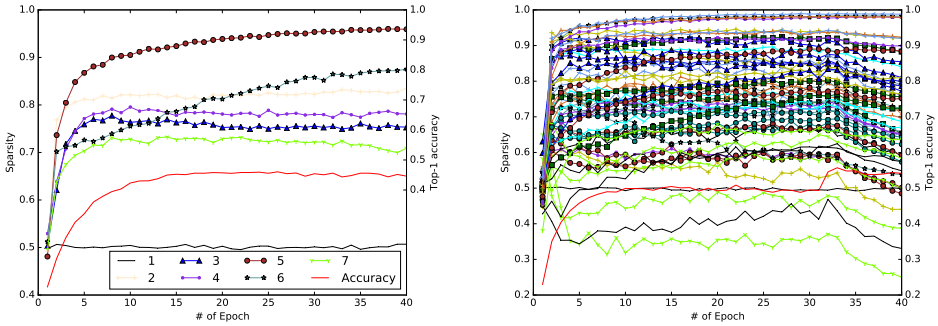
Inverse Sparse Convolution. As we show in the Section 3, there are not less than 50% zeros after the activation of ReLU, whose output is generally the input of the next convolutional layer. We design an algorithm to accelerate the convolution operation by skipping the calculations involved with zeros. In other words, the zero values are excluded to calculating the convolutional output, which reduces up to 50% multiplication and addition operations. One may think that this skipping-zero calculations can be mapped to dense-sparse matrix multiplication. However, the dense-sparse algorithm is mostly slower than the dense-dense case since the memory access in the dense case can be continuous [18]. So it is chanllenging to speedup the convolution with sparse input.

Instead of scanning each element of outputs, we scan each non-zero input, and calculate the multiplication with kernels to generate the temporary results of outputs. We have two considerations of the proposed algorithm: on one hand, the MACs can be reduce to $\rho \times H_{out} \times W_{out} \times R \times S \times K \times C$, given a sparsity: $1 - \rho$ of input $\mathcal{I}$, where $\rho = \frac{\#OfNonZeros}{C \times H_{in} \times W_{in}}$. On the other hand, the number of



(a) LeNet

(b) AlexNet on Cifar10



(c) AlexNet on ImageNet

(d) GoogLeNet

Figure 2: The sparsity of each output layer on popular deep neural networks. The number in the legend represents the nth ReLU layer in the deep nerual networks. The red line without markers indicates the validation accuracy.

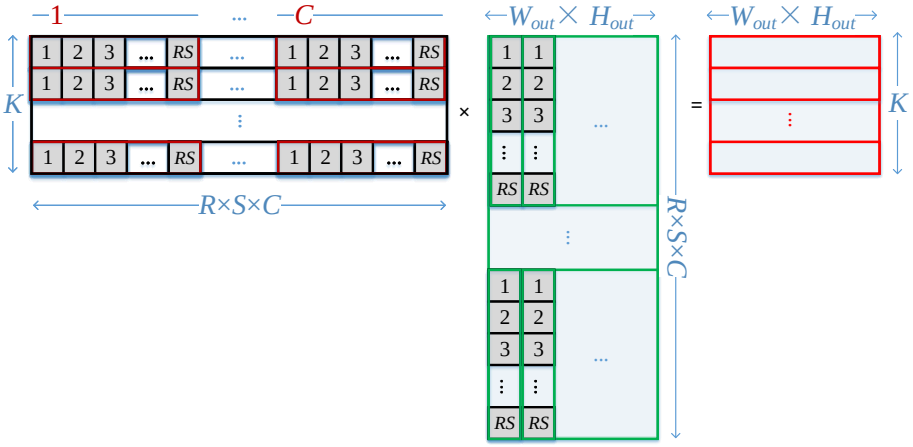


Figure 3: Matrix multiplication for convolution.

memory write operations of the ouput results should not be too large. For each non-zero $\mathcal{I}_{c,i,j}$ in inputs, calculate the product: $\mathcal{I}_{c,i,j} \times \mathcal{W}_{k,c,i-r,j-s}$ for an output: $\mathcal{O}_{k,i+r,j+s}$.

With the two considerations above, we propose the inverse sparse convolution (ISC) algorithm by three steps: First, we skip all the zero elements of the input data, and store the non-zero values in a

vector with their column and row information. Second, the kernel matrix is stored as column-major matrix such that for each non-zero element ($\mathcal{I}_{c,i,j}$) of inputs, a continious memory that stores kernels (say 4 floats of weights: $\mathcal{W}_{k,c,i-r,j-s}$, $\mathcal{W}_{k+1,c,i-r,j-s}$, $\mathcal{W}_{k+2,c,i-r,j-s}$ and $\mathcal{W}_{k+3,c,i-r,j-s}$) can be fetched and multiplied by $\mathcal{I}_{c,i,j}$ at one time with AVX or SSE techniques. Third, transpose temporary results from the second step to generate outputs: $\mathcal{O}$.

5 Experiments

The baseline of convolution algorithm on CPUs is the GEMM version from Caffe [12] with MKL library, which is faster than other deep learning frameworks with single core of CPUs [22]. There exist fast convolution algorithms like Winograd and FFT based algorithms [15], but they need prerequisite for usage. The Wingrad based algorithm is used for the kernel whose size is 3×3, while the FFT based algorithm requires the value of *N* should be equal to the power of 2. We run the speed measurements with convolution layers whose sparsity of inputs is high enough (i.e., ≥ 0.9) on four popular deep networks (i.e., LeNet, AlexNet for Cifar10, AlexNet [14] for ImageNet, GoogLeNet [24]). The experimental convolution layers are summarized in Table 2.

Table 2: Convolution layers on different deep networks. The sparsity indicates the ratio of zero-valued input over the total number of input in that layer. AlexNetC and AlexNetI represent AlexNet for Cifar10 and ImageNet respectively.

Network	Layer	C	$H_{in} \times W_{in}$	K	$R \times S$	P	T	Sparsity
LeNet	Conv2	20	11×11	64	5×5	1	2	0.95
AlexNetC	Conv3	32	6×6	64	5×5	2	1	0.9
AlexNetI	Conv4	384	5×5	384	3×3	1	1	0.9
GoogLeNet	Inception4a.1	480	14×14	192	1×1	0	1	0.9
GoogLeNet	Inception4a.2	192	14×14	96	1×1	0	1	0.9
GoogLeNet	Inception4e.3	160	14×14	320	3×3	1	1	0.9
GoogLeNet	Inception5a.1	832	7×7	256	1×1	0	1	0.95
GoogLeNet	Inception5a.2	256	7×7	160	1×1	0	1	0.9
GoogLeNet	Inception5b.3	192	7×7	384	3×3	1	1	0.95
GoogLeNet	Inception5b.5	48	7×7	128	5×5	2	1	0.95

We measure the speed of compared algorithms on the Interl CPU: E5-2630v4 at the core frequency of 2.20GHz with 128 GB memory. The experimental results are shown in Table 3.

Table 3: Experimental results (Time in msec).

Layer	Sparsity	GEMM	ISC	Speedup
LeNet-Conv2	0.95	0.854	0.123	6.96X
AlexNetC-Conv3	0.9	0.329	0.122	2.69X
AlexNetI-Conv4	0.9	4.874	1.660	2.94X
GoogLeNet-Inception4a.1	0.9	1.795	1.292	1.39X
GoogLeNet-Inception4a.2	0.9	0.943	0.558	1.69X
GoogLeNet-Inception4e.3	0.9	7.602	4.782	1.59X
GoogLeNet-Inception5a.1	0.95	1.912	0.645	2.97X
GoogLeNet-Inception5a.2	0.9	0.742	0.258	2.87X
GoogLeNet-Inception5b.3	0.95	4.410	0.931	4.74X
GoogLeNet-Inception5b.5	0.95	1.560	0.220	7.11X

6 Conclusion and Future Work

The high sparsity of output of ReLUs are explored in this paper and this property is exploited to accelerate the calculation of convolution layers by skipping zero-valued neurons. We first demostrate the sparsity of ReLUs on some popular deep convolutional networks, which displays that many ReLU layers have more than 70% zero-valued outputs. And then we propose an efficient convolution

algorithm by skipping the operations on zero-valued neurons, and it performs speedup improvement compared to the traditional convolution algorithm on our tested CPU platform.

Even though the algorithm proposed in this paper has some acceleration on CPUs, it is not comparable with the algorithms on GPUs. Therefore, in the future work, we need to design the sparse convolution algorithm on GPUs.

References

[1] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 1–13. IEEE, 2016.

[2] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

[3] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 2016.

[4] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE Computer Society, 2014.

[5] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[6] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[8] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[9] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[12] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[13] Wei Wen Ping Tak Peter Tang Hai Li Yiran Chen Jongsoo Park, Sheng Li. Faster cnns with direct sparse convolutions and guided pruning. In *International Conference on Learning Representations (ICLR)*, 2017.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[15] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[19] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[20] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[21] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 267–278. IEEE Press, 2016.

[22] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. *arXiv preprint arXiv:1608.07249*, 2016.

[23] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint arXiv:1703.09039*, 2017.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[25] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[26] Benjamin Willmore and David J Tolhurst. Characterizing the sparseness of neural codes. *Network: Computation in Neural Systems*, 12(3):255–270, 2001.