Learning both Weights and Connections for Efficient Neural Networks

Song Han[1] **Jeff Pool**[2] **John Tran**[2] **William J. Dally**[1,2]
[1]Stanford University [2]NVIDIA
{songhan, dally}@stanford.edu {jpool, johntran}@nvidia.com

Abstract

Neural networks are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems. Also, conventional networks fix the architecture before training starts; as a result, training cannot improve the architecture. To address these limitations, we describe a method to reduce the storage and computation required by neural networks by an order of magnitude without affecting their accuracy, by learning only the important connections. Our method prunes redundant connections using a three-step method. First, we train the network to learn which connections are important. Next, we prune the unimportant connections. Finally, we retrain the network to fine tune the weights of the remaining connections. On the ImageNet dataset, our method reduced the number of parameters of AlexNet by a factor of $\mathbf{9}\times$, from 61 million to 6.7 million, without incurring accuracy loss. Similar experiments with VGG16 found that the network as a whole can be reduced $\mathbf{13}\times$, again with no loss of accuracy.

1 Introduction

Neural networks have become ubiquitous in applications ranging from computer vision [1] to speech recognition [2] and natural language processing [3]. We consider convolutional neural networks used for computer vision tasks which have grown over time. In 1998 Lecun classified handwritten digits with less than 1M parameters [4], while in 2012, Krizhevsky et al. [1] won the ImageNet competition with 60M parameters. Deepface classified human faces with 120M parameters [5], and Coates et al. [6] scaled up a network to 10B parameters.

While these large neural networks are very powerful, their size consumes considerable storage, memory bandwidth, and computational resources. For embedded mobile applications, these resource demands become prohibitive. Figure 1 shows the energy cost of basic arithmetic and memory operations in a 45nm CMOS process [7]. From this data we see the energy per connection is dominated by memory access and ranges from 3.5pJ for 32b coefficients in on-chip SRAM to 640pJ for 32b coefficients in off-chip LPDDR2 DRAM. Large networks do not fit in on-chip storage and hence require the more costly DRAM accesses. Running a 1G connection neural network, for example, at 20Hz would require $(20Hz)(1G)(640pJ) = 12.8W$ just for DRAM access - well beyond the power envelope of a typical mobile device. Our goal in pruning networks is to reduce the energy required to run such large networks so they can be run in real time on mobile devices.

To achieve this goal, we present a method to prune network connections in a manner that preserves the original accuracy. After an initial training phase, we remove all connections whose weight is lower than a threshold. This pruning converts a dense, fully-connected layer to a sparse layer. This first phase learns the topology of the networks — learning which connections are important and removing the unimportant connections. We then retrain the sparse network so the remaining connections can compensate for the connections that have been removed. The phases of pruning and retraining may be repeated iteratively to further reduce network complexity. In effect, this training process learns

Operation	Energy [pJ]	Relative Cost
16 bit int ADD	0.06	1
16 bit FP ADD	0.45	8
16 bit int MULT	0.8	13
16 bit FP MULT	1.1	18
32b LPDDR2 DRAM	**640**	**10667**

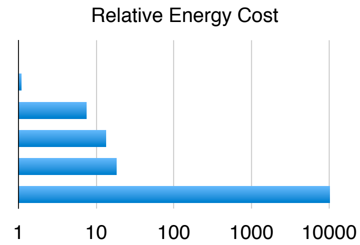


Figure 1: Pyramid shaped energy table: memory access is 3 orders of magnitude more energy expensive

the network connectivity in addition to the weights - much as in the mammalian brain [8] [9], where synapses are created in the first few months of a child's development, followed by gradual pruning of little-used connections, falling to typical adult values.

2 Related Work

Neural networks are typically over-parameterized, and there is significant redundancy for deep learning models [10]. This results in a waste of both computation and memory usage. There have been various proposals to remove the redundancy: Vanhoucke et al. [11] explored a fixed-point implementation with 8-bit integer (vs 32-bit floating point) activations. Denton et al. [12] exploited the linear structure of the neural network by finding an appropriate low-rank approximation of the parameters and keeping the accuracy within 1% of the original model. With similar accuracy loss, Gong et al. [13] compressed deep convnets using vector quantization, which reduced storage but added one level of indirection in memory reference due to accessing the codebook. These approximation and quantization techniques are orthogonal to network pruning, and they can potentially be used together to obtain further gains.

There have been other attempts to reduce the number of parameters of neural networks by replacing the fully connected layer with global average pooling. The Network in Network architecture [14] and GoogLenet [15] achieves state-of-the-art results on several benchmarks by adopting this idea. However, transfer learning, i.e. reusing features learned on the ImageNet dataset and applying them to new tasks by only fine-tuning the fully connected layers, is more difficult with this approach. This problem is noted by Szegedy et al [15] and motivates them to add a linear layer on the top of their networks to enable transfer learning.

Network pruning has been used both to reduce network complexity and to reduce over-fitting. An early approach to pruning was biased weight decay [16]. Optimal Brain Damage [17] and Optimal Brain Surgeon [18] prune networks to reduce the number of connections based on the Hessian of the loss function and suggest that such pruning is more accurate than magnitude-based pruning such as weight decay. However, calculating the second derivative is costly for today's large scale neural networks.

Dropout [19] and DropConnect [20] zeros out activations and connections in the network to reduce over-fitting rather than to improve efficiency. A similar approach was originally described in [21]. Dropout differs from our method both in motivation and in that it zeros connections or activations during training rather than pruning them: during subsequent testing these connections or activations still take on non-zero values. Thus a dense layer remains a dense layer, and there's no parameter saving at deployment time.

HashedNets [22] is a recent technique to reduce model sizes by using a hash function to randomly group connection weights into hash buckets, so that all connections within the same hash bucket share a single parameter value. This technique may benefit from pruning. As pointed out in Shi et al. [23] and Weinberger et al. [24], sparsity will minimize hash collision making feature hashing even more effective. HashedNets may be used together with pruning to give even better parameter savings.

3 Learning Connections in Addition to Weights

Our pruning method employs a three-step process, as illustrated in Figure 2, which begins by learning the connectivity via normal network training. Unlike conventional training, however, we are not learning the final values of the weights, but rather we are learning which connections are important.

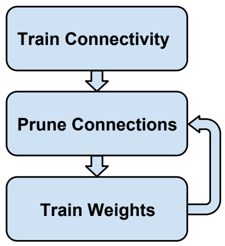


Figure 2: Three-Step Training Pipeline

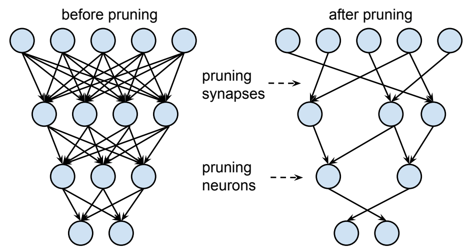


Figure 3: Synapses and neurons before and after pruning

The second step is to prune the low-weight connections. All connections with weights below a threshold are removed from the network — converting a dense network into a sparse network, as shown in Figure 3.

The final step retrains the network to learn the final weights for the remaining sparse connections. This step is critical, however, if the pruned network is used without retraining, accuracy is significantly impacted.

3.1 Regularization

Choosing the correct regularization impacts the performance of pruning and retraining. L1 regularization penalizes non-zero parameters resulting in more parameters near zero. This gives better accuracy after pruning, but before retraining. However, the remaining connections are not as good as with L2 regularization, resulting in lower accuracy after retraining.

3.2 Dropout and Capacity Control

Dropout is widely used to prevent over-fitting, and this also applies to retraining. During retraining, however, the dropout ratio must be adjusted to account for the change in model capacity. Hinton's dropout [19] can be regarded as a "soft dropout," since each parameter is dropped with a probability, not definitely dropped out. Pruning can be regarded as a "hard dropout," where parameters are dropped forever after pruning and have no chance to come back. As the parameters get sparse, the classifier will select the most informative predictors and thus have much less prediction variance, which reduces over-fitting. As pruning already reduced model capacity, the retraining dropout ratio should be smaller.

Quantitatively, let C_i be the number of connections in layer i, C_{io} for the original network, C_{ir} for the network after retraining, N_i be the number of neurons in layer i. Since dropout works on neurons, and the C_i has a square relationship with the N_i, according to equation 1 thus the dropout ratio after pruning the parameters should follow equation 2, where D_o represent the original dropout rate, D_r represent the dropout rate during retraining.

$$C_i = N_i N_{i-1} \qquad (1) \qquad\qquad D_r = D_o \sqrt{\frac{C_{ir}}{C_{io}}} \qquad (2)$$

3.3 Local Pruning and Parameter Co-adaptation

During retraining, it is better to retain the weights from the initial training phase for the connections that survived pruning than it is to re-initialize the pruned layers. [25] shows that CNNs contain fragile co-adapted features: gradient descent is able to find a good solution when the network is initially trained, but not after re-initializing some layers and retraining them. So if we re-initialize the pruned layer, we have to retrain the whole network; if we retrain just the pruned layers, we need to retrain while retaining the surviving parameters.

Retraining the pruned layers starting with retained weights requires less computation because we don't have to back propagate through the entire network. Also, neural networks are prone to suffer the vanishing gradient problem [26] as the networks get deeper, which makes pruning errors harder to recover for deep networks. To prevent this, we fix the parameters for part of the network and only retrain a shallow network by reusing the surviving parameters, which already co-adapted well with the un-pruned layers during initial training.

3.4 Iterative Pruning

Learning the right connections is an iterative process. Pruning followed by a retraining is one iteration, after many such iterations the minimum number connections could be found. Without loss of accuracy, this method can boost pruning rate from $5\times$ to $9\times$ on AlexNet compared with single-step aggressive pruning. Each iteration is a greedy search in that we find the best connections. We also experimented with probabilistically pruning parameters based on their absolute value, but this gave worse results.

3.5 Pruning Neurons

After pruning connections, neurons with zero input connections or zero output connections may be safely pruned. This pruning is furthered by removing all connections to or from a pruned neuron. The retraining phase automatically arrives at the result where dead neurons will have both zero input connections and zero output connections. This occurs due to gradient descent and regularization. A neuron that has zero input connections(or zero output connections) will have no contribution to the final loss, leading the gradient to be zero for its output connection(or input connection), respectively. Only the regularization term will push the weights to zero. Thus the dead neurons will be automatically removed during retraining.

4 Experiments

We implemented network pruning on Caffe [27]. Caffe was modified to add a mask for all the weight tensors, which disregards pruned parameters during network operation. The pruning threshold is chosen as a quality parameter multiplied by the standard deviation of a layer's weights. We carried out the experiments on Nvidia TitanX and GTX980 GPUs.

We pruned 4 networks: 2 on MNIST and 2 on ImageNet datasets. The network parameters and accuracy [1] before and after pruning are shown in Table 1.

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**$\times$
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**$\times$
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**$\times$
VGG16 Ref	31.50%	11.32%	138M	
VGG16 Pruned	31.34%	10.88%	**10.3M**	**13**$\times$

Table 1: Network pruning can save $9\times$ to $13\times$ parameters with no drop in predictive performance

4.1 LeNet on MNIST

We first experimented on MNIST dataset with LeNet-300-100 and LeNet-5 network [4]. LeNet-300-100 is a fully connected network with two hidden layers, with 300 and 100 neurons each, which achieves 1.6% error rate on Mnist. LeNet-5 is a convolutional network that has two convolutional layers and two fully connected layers, which achieves 0.8% error rate on Mnist. After pruning, the network is retrained with $1/10$ of the original network's original learning rate. Table 1 shows pruning

[1] Reference model is from Caffe model zoo, accuracy is measured without data augmentation

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
Fastfood-32-AD [28]	41.93%	-	32.8M	2×
Fastfood-16-AD [28]	42.90%	-	16.4M	3.7×
Collins & Kohli [29]	44.40%	-	15.2M	4×
Naive Cut	47.18%	23.23%	13.8M	4.4×
AlexNet Pruned	**42.77%**	**19.67%**	**6.7M**	**9**×

Table 2: Comparison with other model reduction methods on AlexNet. [29] reduced the parameters by $4\times$ and with inferior accuracy. Deep Fried Convnets [28] worked on fully connected layers only and reduced the parameters by less than $4\times$. Naively cutting the layer size save parameters but suffers from large accuracy loss as much as 4%. [12] exploited linear structure of convnets and reduced the parameters of each layer separately, where model compression on a single layer incurred 0.9% accuracy penalty after doing biclustering + SVD.

achieves $12\times$ parameter savings on these networks. For each layer of the network the table shows (left to right) the original number of weights, the number of floating point operations to compute that layer's activations, the average percentage of activations that are non-zero, the percentage of weights remaining after pruning, and the percentage of operations remaining after pruning and exploiting sparse activation. The bottom line gives the same data for the overall network.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
fc1	235K	470K	38%	8%	8%
fc2	30K	60K	65%	9%	4%
fc3	1K	2K	100%	26%	17%
Total	266K	532K	46%	**8%**	**8%**

Table 3: Saving break down for Lenet-300-100 with respect to the number of weights and number of floating point operations for each layer

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	0.5K	576K	82%	66%	66%
conv2	25K	3200K	72%	12%	10%
fc1	400K	800K	55%	8%	6%
fc2	5K	10K	100%	19%	10%
Total	431K	4586K	77%	**8%**	**16%**

Table 4: Saving break down for Lenet5 with respect to the number of weights and number of floating point operations for each layer

An interesting byproduct of network pruning is that it detects visual attention regions. Figure 4 shows the sparsity pattern of the first fully connected layer of LeNet-300-100, which is of size $784 * 300$ (the image size is 28 by 28 and the hidden layer size is 300). It has a banded structure with 28 bands, each band is of size 28, corresponding to the 28×28 input pixels. The colored regions of the figure, which means non-zero parameters, correspond to the center of the image. Because digits are written in the center of the image, these are the important parameters. The graph is sparse on the left and right as well, corresponding to the less important regions on the top and bottom of the image. After pruning, the neural network finds the center of the image more important, and the connections to the peripheral regions are more heavily pruned.

4.2 AlexNet on ImageNet

We further examine the performance of pruning on the ImageNet ILSVRC-2012 dataset, which has 1.2M training examples and 50k validation examples. We use the AlexNet Caffe model as the reference model, which has 61 million parameters across 5 convolutional layers and 3 fully connected layers. The AlexNet Caffe model achieved a top-1 accuracy of 57.2% and a top-5 accuracy of 80.3%. The original AlexNet took 450K iterations to train [27]. After pruning, the whole network is retrained with $1/100$ of the original network's initial learning rate. 900K and 700K iterations were required to retrain the fully connected and convolutional layers, respectively. Table 1 shows that AlexNet can be

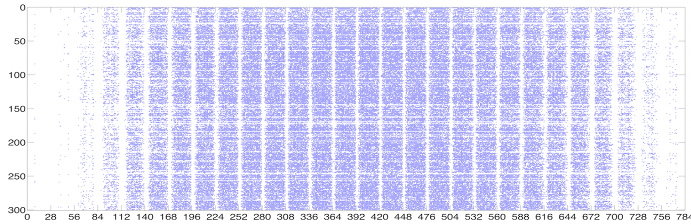


Figure 4: Visualization of the first layer's sparsity pattern. It has a banded structure repeated 28 times, which correspond to the un-pruned parameters in the center of the images, since the digits are written in the center.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10%
Total	61M	1.5B	54%	**11%**	**30%**

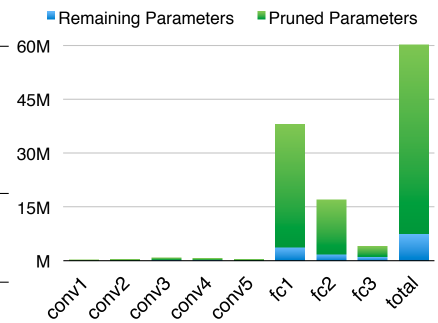


Table 5: Saving break down for each layer of AlexNet. Pruning finds the essential connections, reducing weights to 11% and computation to 30%. Most of the parameter savings come from the fully connected layers

pruned to $1/9$ of its original size without impacting accuracy and computation can be reduced by $3\times$. The pruning is most effective on the fully-connected layers.

We experimented with L1 and L2 regularization before and after retraining, together with iterative pruning to give the five accuracy v.s parameter trade off curves shown in Figure 5. The green and dotted purple lines show the importance of retraining the network after pruning. The purple line shows the accuracy of the network after pruning but before retraining. Without retraining we can still prune the network considerably: half of AlexNet could be pruned without impacting the accuracy much. However, accuracy begins dropping much sooner — with $1/3$ of the original connections, rather than with $1/10$ of the original connections. Without retraining we are only able to reduce the number of connections by $2\times$ before losing accuracy, while with retraining we are ably to reduce connections by $9\times$.

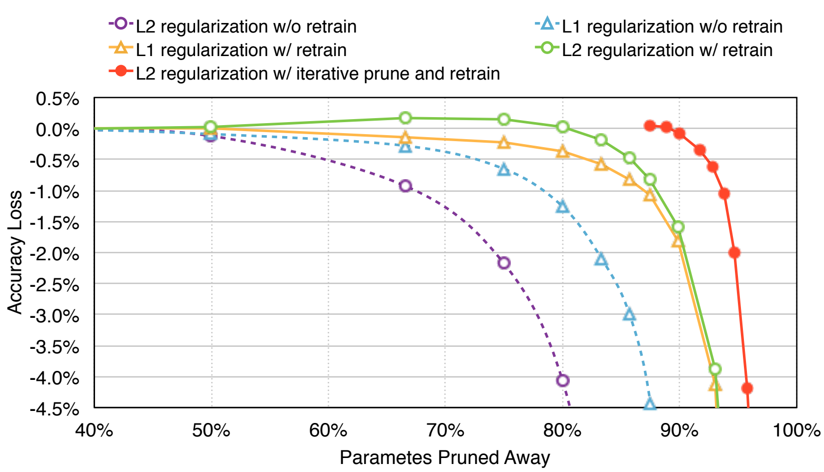


Figure 5: Trade-off curve for parameter reduction and loss in top-5 accuracy. L1 regularization performs better than L2 at learning the connections without retraining, while L2 regularization performs better than L1 at retraining. Iterative pruning gives the best result

L1 regularization gives better accuracy than L2 directly after pruning (dotted blue and purple lines) since it pushes more parameters closer to zero. However, comparing the yellow and green lines shows that L2 outperforms L1 after retraining, since there is no benefit to further pushing values towards zero. One extension is to use L1 regularization for pruning and then L2 for retraining, but this did not beat simply using L2 for both phases. Parameters from one mode do not adapt well to the other.

The biggest gain comes from iterative pruning (solid red line with solid circles). Here we take the pruned and retrained network (solid green line with circles) and prune and retrain it again. The leftmost dot on this curve corresponds to the point on the green line at 80% ($5\times$ pruning) pruned to $8\times$. There's no accuracy loss at $9\times$. Not until $10\times$ does the accuracy begin to drop.

Two points achieve slightly better accuracy than the original model after pruning and retraining[2]. We believe this small accuracy improvement is due to pruning reducing the capacity of the network and hence reducing overfitting.

Both fully connected layers and convolutional layers can be pruned. Fully connected layers occupy the majority of the parameters and could be pruned as much as ($11\times$) compared to convolution layers ($3\times$). The details of pruning each layer are shown in Table 5.

Network pruning not only saves parameter's memory access but also saves floating point operations, given appropriate hardware support. If either the weight or the activation is zero, the multiply and add operations can be saved. An estimation of computation savings assumes independent distribution of non-zero weights and non-zero activations. The FLOP column is derived by multiplying the non-zero parameter ratio of the current layer with the non-zero activation ratio of the previous layer, since the input of this layer is the output of the previous layer. Although fully connected layer has more parameters and parameter savings, the convolutional layers are more computationally intensive. Thus, the FLOP savings is determined by the convolutional layers, which is $3\times$.

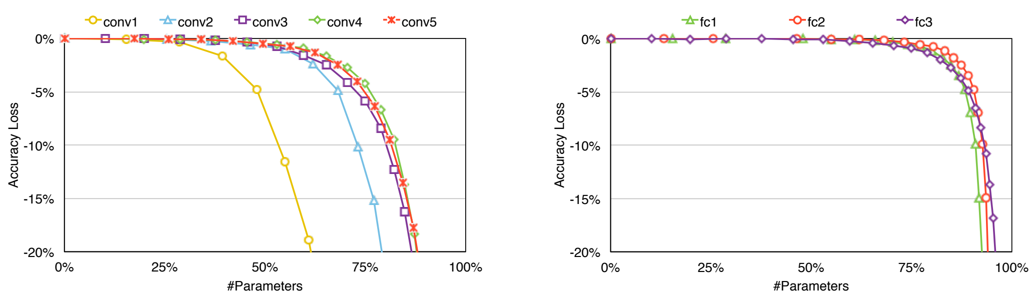


Figure 6: Pruning sensitivity for convolutional layer and fully connected layer of AlexNet

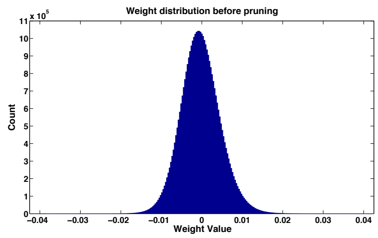
Figure 6 shows the sensitivity of each layer to network pruning. The figure shows how accuracy suffers as parameters are pruned on a layer-by-layer basis. The convolutional layers (on the left) are more sensitive to pruning than the fully connected layers (on the right). The first convolutional layer, which interacts with the input image directly, is most sensitive to pruning. We suspect this sensitivity is due to the input layer having only 3 channels and thus less redundancy than the other convolutional layers. We used these results to find each layer's threshold. The smallest threshold was applied to the most sensitive layer, which is the first convolutional layer, etc.

Figure 7 shows histograms of weight distribution before (left) and after (right) pruning. The weight is from the first fully connected layer of AlexNet. The two panels have different y-axis scales. The original distribution of weights is centered on zero with tails dropping off quickly. Almost all parameters are between $[-0.015, 0.015]$. After pruning the large center region is removed. The network parameters adjust themselves during the retraining phase. The result is that the parameters form a bimodal distribution and become more spread across the x-axis, between $[-0.025, 0.025]$.

4.3 VGG16 on ImageNet

With promising results on AlexNet, we also looked at a larger, more recent network, VGG16 [30], on the same ILSVRC-2012 dataset. VGG16 has far more convolutional layers but still only three

[2] We verified that the Caffe model is fully trained.



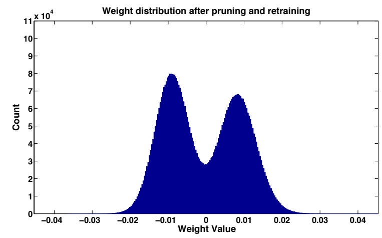


Figure 7: Weight distribution before and after parameter pruning. The right figure has $10\times$ smaller scale

fully-connected layers. Following a similar methodology, we aggressively pruned both convolutional and fully-connected layers to realize a significant reduction in the number of weights, shown in Table 6. Specifically, we used five iterations of pruning on the pre-trained network from Caffe.

Table 6: Saving breakdown for VGG16 with respect to the number of weights of each layer

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1_1	2K	0.2B	53%	58%	58%
conv1_2	37K	3.7B	89%	22%	12%
conv2_1	74K	1.8B	80%	34%	30%
conv2_2	148K	3.7B	81%	36%	29%
conv3_1	295K	1.8B	68%	53%	43%
conv3_2	590K	3.7B	70%	24%	16%
conv3_3	590K	3.7B	64%	42%	29%
conv4_1	1M	1.8B	51%	32%	21%
conv4_2	2M	3.7B	45%	27%	14%
conv4_3	2M	3.7B	34%	34%	15%
conv5_1	2M	925M	32%	35%	12%
conv5_2	2M	925M	29%	29%	9%
conv5_3	2M	925M	19%	36%	11%
fc6	103M	206M	38%	4%	1%
fc7	17M	34M	42%	4%	2%
fc8	4M	8M	100%	23%	9%
total	138M	30.9B	64%	**7.5%**	**21%**

The VGG16 results are, like those for AlexNet, very promising. The network as a whole has been reduced to 7.5% of its original size. In particular, note that the two largest fully-connected layers can each be pruned to less than 4% of their original size. This reduction is critical for real time image processing, where there is little reuse of these layers across images (unlike batch processing). The reduced layers will fit in an on-chip SRAM and have modest bandwidth requirements. Without the reduction the bandwidth requirements are prohibitive.

5 Conclusion

We have presented a method to improve the efficiency of neural networks without affecting accuracy by finding the right connections. Our method, motivated in part by how learning works in the mammalian brain, operates by learning which connections are important, pruning the unimportant connections, and then retraining the remaining sparse network. We highlight our experiments on AlexNet on ImageNet, showing that both fully connected layer and convolutional layer can be pruned, reducing the number of connections by $9\times$ without loss of accuracy. We show similar results for VGG16 and LeNet networks got pruned by $13\times$ without accuracy loss. This leads to smaller memory capacity and bandwidth requirements for real-time image processing.

References

[1] 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.

[2] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[3] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

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

[5] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, pages 1701–1708. IEEE, 2014.

[6] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *30th ICML*, pages 1337–1345, 2013.

[7] Mark Horowitz. Energy table for 45nm process.

[8] JP Rauschecker. Neuronal mechanisms of developmental plasticity in the cat's visual system. *Human neurobiology*, 3(2):109–114, 1983.

[9] Christopher A Walsh. Peter huttenlocher (1931-2013). *Nature*, 502(7470):172–172, 2013.

[10] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[11] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[12] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[13] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[14] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[16] Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pages 177–185, 1989.

[17] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.

[18] Babak Hassibi, David G Stork, et al. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, pages 164–164, 1993.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[20] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *ICML*, pages 1058–1066, 2013.

[21] Hans Henrik Thodberg. Improving generalization of neural networks through pruning. *International Journal of Neural Systems*, 1(04):317–326, 1991.

[22] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *arXiv preprint arXiv:1504.04788*, 2015.

[23] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. Hash kernels for structured data. *The Journal of Machine Learning Research*, 10:2615–2637, 2009.

[24] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120. ACM, 2009.

[25] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N.d. Lawrence, and K.q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014.

[26] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[27] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[28] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. *arXiv preprint arXiv:1412.7149*, 2014.

[29] Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.