Less is More: Towards Compact CNNs

Supplementary Material

Hao Zhou[1], Jose M. Alvarez[2] and Fatih Porikli[2,3]

[1] University of Maryland, College Park
[2] Data61/CSIRO
[3] Australian National University
hzhou@cs.umd.edu, jose.alvarez@data61.csiro.au
fatih.porikli@anu.edu.au

1 The way we compute parameter compression and memory footprint reduction.

We take AlexNet [1] as an example to show how we compute the parameter compression and memory footprint reduction of a network.

We show the structure of the filters in each convolutional layer and fully connected layer of AlexNet [1] in Table 1. Each neuron is an order-3 tensor, we use width, height and input to show the number of elements for each of the three channels. Output shows the number of neurons in each layer (it is also the number of output features of each layer). Here, suppose AlexNet is used to classify images into 1000 categories, as a result, fc8 contains 1000 output channels. In total the number of parameters is $60,965,224$.

Our purpose is to remove the number of neurons, i.e. the number in "output" column in Table 1. Please note that the number of input of $(i+1)$-th layer is related to the number of output in i-th layer. For example, if we can remove n neurons for fc6, the corresponding n channels in input of fc7 will also be removed.

In the paper, we show that we can remove 2996 neurons for fc6, then the number of parameters removed are computed as:

$$\begin{aligned} \text{fc6} &= 6 \times 6 \times 256 \times 2996 + 2996 \\ &= 27,614,132, \end{aligned} \tag{1}$$

$$\begin{aligned} \text{fc7} &= 1 \times 1 \times 2996 \times 4096 \\ &= 12,271,616, \end{aligned} \tag{2}$$

$$\begin{aligned} \text{total} &= \text{fc6} + \text{fc7} \\ &= 39,885,748. \end{aligned} \tag{3}$$

As a result, the percentage of parameters we removed is $39,882,752/60,965,224 = 65.42\%$.

Supposing single type float (4 byte) is used to store the parameters. The memory footprint reduction is

$$(\text{total}) \times 4/1024/1024 = 152.2 \quad (\text{MB}) \tag{4}$$

Table 1. The structure of neurons for each convolutional and fully connected layer in AlexNet.

layer	width	height	input	output	bias	total
conv1	11	11	3	96	96	34.94 K
conv2	5	5	48	256	256	307.46 K
conv3	3	3	256	384	384	885.12 K
conv4	3	3	192	384	384	663.94 K
conv5	3	3	192	256	256	442.62 K
fc6	6	6	256	4096	4096	37.75 M
fc7	1	1	4096	4096	4096	16.78 M
fc8	1	1	4096	1000	1000	4.10 M

2 Algorithm of solving low rank constraints

In the paper, we mention that the backward step in forward backward splitting for tensor low rank constraints is

$$\begin{aligned}\hat{\mathbf{w}}_{lj}^{k} &= \arg\min_{\hat{\mathbf{w}}_{lj}} \frac{1}{n}\sum_{i=1}^{n}||\hat{\mathbf{w}}_{lj(i)}||_{tr} + \frac{1}{2\tau\lambda}||\hat{\mathbf{w}}_{lj}^{k} - \hat{\mathbf{w}}_{lj}^{k*}||^2 \\ &= \arg\min_{\hat{\mathbf{w}}_{lj}} \frac{1}{n}\sum_{i=1}^{n}||\hat{\mathbf{w}}_{lj(i)}||_{tr} + \frac{1}{2\tau\lambda n}\sum_{i=1}^{n}||\hat{\mathbf{w}}_{lj(i)}^{k} - \hat{\mathbf{w}}_{lj(i)}^{k*}||^2. \end{aligned} \tag{5}$$

We use Low Rank Tensor Completion (LRTC) [2] to deal with our tensor low rank constraints. Here, we give the detail of the algorithm. Suppose we have a 2D matrix X, let $X = U\Sigma V$ be the singular value decomposition of X, where Σ is a diagonal matrix of the singular values of X. Now let us suppose the i-th singular value is σ_i, then define $\Sigma_\tau = diag(\max(\sigma_i - \tau), 0)$. Then the shrinkage operate is defined as:

$$D_t(X) = U\Sigma_t V^T, \tag{6}$$

Algorithm 1 shows how to use LRTC to optimize Equation (5).

Algorithm 1 Backward step for tensor low rank constraints

1: Initialize $\hat{\mathbf{w}}_{lj}^{k} = \hat{\mathbf{w}}_{lj}^{k*}$
2: **while** not converged **do**
3: **for** i = 1 to n **do**
4: $M_i = D_{\tau\lambda}(\frac{1}{2}(\hat{\mathbf{w}}_{lj(i)}^{k} + \hat{\mathbf{w}}_{lj(i)}^{k*}))$
5: **end for**
6: $\hat{\mathbf{w}}_{lj}^{k} = \frac{1}{n}\sum_{i=1}^{n} M_i$
7: **end while**

3 Proof for section 5

We show in the paper that the practical definition of ReLU is as follows:

$$ReLU(x) = \begin{cases} x & \text{if } x > \epsilon \\ 0 & \text{if } x \leq \epsilon. \end{cases} \tag{7}$$

Where ϵ is chosen such that for any real number x that a computer can represent, if $x > 0$, then $x > \epsilon$; if $x \leq 0$, then $x < \epsilon$. For any real number x that can be represented by a computer, the gradient of ReLU function is:

$$\frac{dReLU(x)}{dx} = \begin{cases} 1 & \text{if } x > \epsilon \\ 0 & \text{if } x < \epsilon. \end{cases} \tag{8}$$

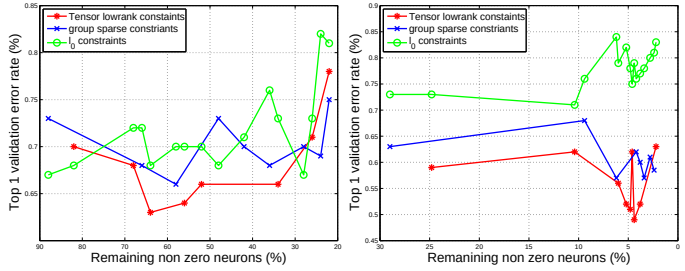
Take a particular neuron $\hat{\mathbf{w}}_{lj}$ as an example and suppose all other neurons are fixed. Also, suppose $\mathbf{x}$ is one of the features that will go through $\hat{\mathbf{w}}_{lj}$ in the l-th layer. The output for $\mathbf{x}$, assuming the convolution layer is followed by a ReLU function, is $z_{\hat{\mathbf{x}}} = ReLU(\hat{\mathbf{w}}_{lj}\hat{\mathbf{x}})$, where $\hat{\mathbf{x}}$ is $\mathbf{x}$ augmented by 1 to account for the bias term. Next, we show that $\hat{\mathbf{w}}_{lj} = \mathbf{0}$ is the local minimum of our objective function along the dimensions of $\hat{\mathbf{w}}_{lj}$, referred to as local minimum of a CNN for simplicity.

Given any $\mathbf{x} \in \Omega$, define $\varphi(\Delta\hat{\mathbf{w}}_{lj}) = \Delta\hat{\mathbf{w}}_{lj}\hat{\mathbf{x}}$ as a function of $\Delta\hat{\mathbf{w}}_{lj}$. It is easy to see that $\varphi(\Delta\hat{\mathbf{w}}_{lj})$ is a continuous function. As a result, we can always find a $\delta_{\mathbf{x}}$ such that $|\varphi(\Delta\hat{\mathbf{w}}_{lj}) - \varphi(\mathbf{0})| < \epsilon$, $\forall ||\Delta\hat{\mathbf{w}}_{lj} - \mathbf{0}|| < \delta_{\mathbf{x}}$. According to Equation (7), for those $\hat{\mathbf{w}}_{lj}$, $ReLU(\Delta\hat{\mathbf{w}}_{lj}\hat{\mathbf{x}}) = 0$. Denote δ as the smallest one among all $\delta_{\mathbf{x}}$, we know that for any $||\Delta\hat{\mathbf{w}}_{lj} - \mathbf{0}|| < \delta$, $ReLU(\Delta\hat{\mathbf{w}}_{lj}\mathbf{x}) = 0$, $\forall \mathbf{x} \in \Omega$. As all other neurons are fixed, we know that for any $||\Delta\hat{\mathbf{w}}_{lj} - \mathbf{0}|| < \delta$, $\psi(\Delta\hat{\mathbf{w}}_{lj}) = c$, where ψ is the first part in our objective function and c is a scalar. Since $g(\hat{\mathbf{W}})$ contains a sparse constraint on $\hat{\mathbf{w}}_{lj}$, $g(\hat{\mathbf{w}}_{lj} = \mathbf{0}) \leq g(\Delta\hat{\mathbf{w}}_{lj})$ and the equality holds if and only if $\Delta\hat{\mathbf{w}}_{lj} = \mathbf{0}$. As a result, for any $||\Delta\hat{\mathbf{w}}_{lj} - \mathbf{0}|| < \delta$, $\psi(\hat{\mathbf{w}}_{li} = \mathbf{0}) + g(\hat{\mathbf{w}}_{li} = \mathbf{0}) \leq \phi(\Delta\hat{\mathbf{w}}_{li}) + g(\Delta\hat{\mathbf{w}}_{li})$, i.e. $\hat{\mathbf{w}}_{lj} = \mathbf{0}$ is the local minimum of the objective function $\psi(\hat{\mathbf{w}}_{lj}) + g(\hat{\mathbf{w}}_{lj})$.

4 Compare with l_0 constraints on LeNet

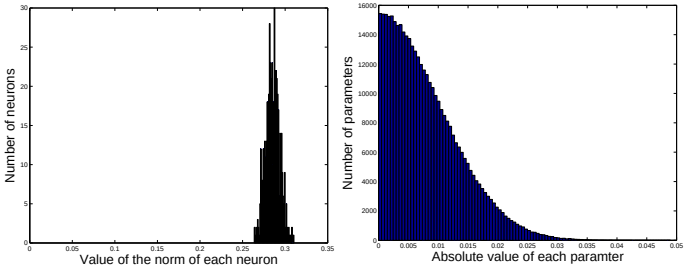
Both [3] and [4] proposed to add l_0 constraints to the parameters of a neural network. They show that it performs quite well in sense of zeroing out parameters. We apply this idea to remove neurons on LeNet and compare it with adding tensor low rank and group sparse constraints. We use exactly the same setup as in the paper and test the idea using MNIST data set [5]. Following the idea of [3] and [4], given a parameter t, neurons whose magnitudes are among the largest top t will be kept, other neurons will be set to zero during the training process.

Figure 1 compares results of l_0 constraints with proposed tensor low rank and group sparsity. For the experiment, t is chosen based on the number of non zero neurons we got by using proposed sparse constraints under different



(a) (b)

Fig. 1. (a) and (b) show the top 1 validation error rate versus the percentage of non zero neurons for second convolutional layer and first fully connected layer for LeNet respectively. We compare results of low rank constraints, group sparse constrains and directly using l_0 constraints.



(a) (b)

Fig. 2. (a) shows the distribution of norm of the neurons in first fully connected layer for LeNet. (b) shows the distribution of the absolute value of the parameters in the first fully connected layer for LeNet. The network is trained with one epoch without adding sparse constraints.

weights. Generally speaking, the proposed constraints perform better than the l_0 constraints for training the network. This is more obvious when we try to get more zero filters for the first fully connected layer. One reason is that, without using any sparse constraints, the norm of the neurons learned are distributed more compactly and far from 0. As a result, directly setting some of the neurons to be zero is risky. The distribution of absolute values of the parameters, even without adding any sparse constraints, are more biased to 0, which explains why l_0 constraint works well in [3] and [4]. This can be seen form Figure 2, which shows the distribution of the norm of the neurons and absolute values of parameters.

Table 2. The structure LeNet in table 3 of submitted paper. τ_1 is the sparse weight for conv2, the sparse weight for fc3 is fixed as 100. Please note that for different weight, we run four times and get the average number of filters for LeNet.

layer	before	after		
		$\tau_1 = 60$	$\tau_1 = 80$	$\tau_1 = 100$
conv1	20 filers 5×5	20 filters 5×5	20 filters 5×5	20 filters 5×5
ReLU				
MaxPooling	2×2	2×2	2×2	2×2
conv2	50 filters 5×5	27 filters 5×5	22 filters 5×5	18 filters 5×5
ReLU				
MaxPooling	2×2	2×2	2×2	2×2
fc3	500 filters 4×4	11 filters 4×4	11 filters 4×4	11 filters 4×4
ReLU				
fc4	10 outputs	10 putputs	10 putputs	10 putputs

Table 3. The structure CIFAR10-quick in table 3 in the paper. τ_1 is the sparse weight for conv3, the sparse weight for fc4 is fixed as 280.

layer	before	after		
		$\tau_1 = 220$	$\tau_1 = 240$	$\tau_1 = 280$
conv1	32 filers 5×5	32 filters 5×5	32 filters 5×5	32 filters 5×5
MaxPooling	3×3	3×3	3×3	3×3
ReLU				
conv2	32 filters 5×5	32 filters 5×5	32 filters 5×5	32 filters 5×5
ReLU				
MaxPooling	3×3	3×3	3×3	3×3
conv3	64 filters 5×5	44 filters 5×5	34 filters 5×5	29 filters 5×5
ReLU				
MaxPooling	3×3	3×3	3×3	3×3
fc4	64 filters 4×4	19 filters 4×4	18 filters 4×4	19 filters 4×4
ReLU				
fc5	10 outputs	10p outputs	10 outputs	10 outputs

5 Some network structures in the paper

In this part, we show the network structure of all the networks before and after adding sparse constraints in **Table** 2, **Table** 3 and **Table** 4 based on table 3 in the paper.

6 Compare results with [6]

Besides the proposed method, [6] is another paper that train the compact CNN from scratch. We want to emphasis that [6], different from the proposed one, cannot reduce the number of neurons. In fact, the effective number of neurons may be increased as they show in section 6.2. In the following, we try to compare with [6].

Table 4. The structure AlexNet in table 3 in the paper.τ_1 is the sparse weight for fc6, τ_2 is the sparse weight for fc7. Note that due to the complex structure of AlexNet, we only show the network from fc6 to the ouput layer. All the other layers are the same.

layer	before	after		
		$\tau_1 = 40$, $\tau_2 = 35$	$\tau_1 = 45$ $\tau_2 = 30$	$\tau_1 = 45$, $\tau_2 = 35$
fc6	4,096 filers $6 \times 6 \times 256$	2,111 filers $6 \times 6 \times 256$	940 filers $6 \times 6 \times 256$	1,090 filers $6 \times 6 \times 256$
ReLU				
fc7	4,096 filters $1 \times 1 \times 4096$	1,782 filters $1 \times 1 \times 4096$	1,630 filters $1 \times 1 \times 4096$	1,401 filters $1 \times 1 \times 4096$
ReLU				
fc8	1,000 outputs	1,000 outputs	1,000 outputs	1,000 outputs

MNIST dataset: The network structure used in [6] is different from ours. However, we show that our compression rate is even higher than the upper bound of theoretical compression rate of [6]. [6] show that they can use TT format to approximate fully connected layers, however, it is not clear whether it can be easily applied to convolutional layers. LeNet we used in our paper has two convolutional layers with 25570 parameters, this is the lower bound of the number of parameters for [6] since it can only deal with fully connected layers. This is worse than our best result shown third row of LeNet in Table 3, which has 13838 parameters in total.

CIFAR-10 dataset: We use the same network structure as [6] does. The compression rate of the NIPS paper is 1.7 with a 1.14% drop in accuracy. Based on Table 3 in our paper, our compression rate is 2.23 with 0.4% drop in accuracy.

ImageNet: We use different networks from [6]. With a 1% drop in accuracy, our best compression rate is 4.3 for AlexNet and 1.5 for VGG-13, best compression rate for VGG-16 in [6] is 3.9.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. (2012)
2. Liu, J., Musialski, P., Wonka, P., Ye, J.: Tensor completion for estimating missing values in visual data. IEEE Transactions on Pattern Analysis and Machine Intelligence (2013)
3. Yu, D., Seide, F., Li, G., Deng, L.: Exploiting sparseness in deep neural networks for large vocabulary speech recognition. In: ICASSP. (2012)
4. Collins, M.D., Kohli, P.: Memory bounded deep convolutional networks. CoRR **abs/1412.1442** (2014)
5. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
6. Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D.P.: Tensorizing neural networks. In: NIPS. (2015)