Max-Variance Convolutional Neural Network Model Compression

Tanya Boone-Sifuentes Antonio Robles-Kelly Asef Nazari
School of IT, Faculty of Eng., Sci. and the Built Env., Deakin University, Waurn Ponds, Australia

***Abstract*—In this paper, we present a method for convolutional neural network model compression which is based on the removal of filter banks that correspond to unimportant weights. To do this, we depart from the relationship between consecutive layers so as to obtain a factor that can be used to assess the degree upon which each pair of filters are coupled to each other. This allows us to use the unit-response of the coupling between two layers so as to remove pathways int he network that are negligible. Moreover, since the back-propagation gradients tend to diminish as the chain rule is applied from the output to the input layer, here we maximise the variance on the coupling factors while enforcing a monotonicity constraint that assures the most relevant pathways are preserved. We show results on widely used networks employing classification and facial expression recognition datasets. In our experiments, our approach delivers a very competitive trade-off between compression rates and performance as compared to both, the uncompressed models and alternatives elsewhere in the literature. pages = 271-279**

***Index Terms*—Convolutional Neural Network Compression, network pruning and max-variance pruning.**

I. Introduction

Deep neural networks have been applied to a wide variety of fields such as image [1] and speech recognition [2], super-resolution [3] and face recognition [4]. The success of convolutional neural networks is due, in part, to their ability to learn non-linear mappings between the input and the output. In a convolutional neural network, convolutional, locally and fully connected layers are combined with rectifiers and pooling into architectures that can achieve state of the art performance. Despite their success, convolutional networks rely upon millions and, sometimes, billions of parameters. For instance, the network in [1], which won the ImageNet competition, employs 60 million parameters over six convolutional layers. The network in [4] employs a mix of locally and fully connected layers containing hundreds of millions of parameters.

These large number of parameters impose a very high computational and memory burden, particularly for portable and resource constrained devices. Thus, recently, there has been a number of approaches to reduce the number of parameters to improve memory and computational efficiency. As mentioned in [5], existing methods can be broadly classified into parameter pruning [6], low rank factorization [7], knowledge distillation [8] and compaction of convolutional filters [9].

Amongst these, parameter pruning methods are known to not only reduce the network complexity, but also to curtail over-fitting. One of the earlier approaches, by Hanson and Pratt [10] employed weight decay for reducing the number of parameters and improve generalisation properties. Shortly after, Hassibi and Stork [11] proposed a method for removing unimportant weights, *i.e.* network pruning, based upon the second order derivatives for the error function. This is a philosophy at the centre of pruning methods, whereby trivial or unnecessary weights from a pre-trained network are removed.

These methods effect the removal process based upon the rationale that networks optimised via gradient descent tend to be over parameterised [12], with a large amount of redundancy in neural network weights [13]. Thus, methods such as that in [14] aim at finding "similar" neurons that may be removed from the network due to them being redundant. This is somewhat similar to the approach in [15], where Hoffman coding and trained quantization are applied together with pruning to improve the compression by removing unnecessary connections in the network. A similar idea was used in [16], where the pruning is effected on a pre-trained network where the important connections are also learnt.

Another option is to use sparse constraints for training. This was explored in [17], where the information on the second derivative at training is used as a trade-off with network complexity. In [18], sparsity constraints are also used to remove neurons during the training process. In an different take on the problem, Chen *et al.* [19] propose a hashing procedure to reduce the memory requirements of the network. In [20], the authors employ the alternating direction method of multipliers (ADMM) [21] regularisation for weight reduction to overcome the heuristic nature of pruning methods elsewhere in the literature.

Indeed, pruning approaches often require an additional fine tuning [22]. Moreover, these approaches often produce an irregular sparsity in the convolutional filters, which, in practice, introduces the requirement of special software for tensor parsing. In this paper, we present a method that directly reduces the number of parameters by removing specific convolutional filters. These filters are selected based on their importance. That is, the method presented here removes filters that correspond to unimportant weights. To remove these filters from further consideration, we note that a pathway between two consecutive layers of the network can be viewed as a coupled system where the weights in the corresponding layers can be employed to compute a coupling factor. Viewed in this manner, these coupling factors can be used to remove convolutional filters in order to maximise the variance and the importance of the remaining pathways.

Fig. 1. Diagram of the recursive training-pruning scheme used in our experiments.

II. Background

As mentioned above, in a neural network the aim of computation is the minimisation of a loss, or cost, function $\mathcal{L}(\Theta, y', y)$. That is, we aim at recovering the parameters such that

$$\begin{aligned}\Theta^* &= \underset{\Theta}{\operatorname{argmin}}\Big\{\mathcal{L}(\Theta, y', y)\Big\} \\ &= \underset{\Theta}{\operatorname{argmin}}\Big\{L(y', y) + \lambda\mathcal{R}(\Theta)\Big\}\end{aligned}$$

where the objective function $L(y', y)$ depends on both, the prediction y' and the target y for each of the input instances x^0 in the training set, $\mathcal{R}(\Theta)$ is a regularisation term that depends on the parameters Θ and $\lambda \geq 0$ is a weight that controls the influence of the regulariser in the minimisation above.

It is worth noting that, in deep networks the parameters are given by the weights in the connections. In the equations above and throughout the paper we make no distinction between these in fully connected and convolutional layers. We have done this for the sake of generality since fully connected layers can be viewed as convolutional layers and, hence, the developments in this and the following section apply to these indistinctively.

Let the evaluation function at the layer indexed l in the network with N layers be $J^{(l)}(\theta^l, x^l)$ with parameters $\theta^l \in \Theta$ and input x^l. By taking the prediction to be the value of the evaluation at the last, *i.e.* N^{th}, layer. With this notation, the prediction y' can be written as follows

$$y' = J^{(N)}(\theta^N, x^N) \circ J^{(N-1)}(\theta^{N-1}, x^{N-1}) \tag{1}$$

where $\circ$ is the composition operator.

This is also the basis for the back propagation algorithm for training neural networks using the chain rule. It is a straightforward matter to show the gradient of y' becomes

$$\nabla_{\theta^{N-1}} y' = \nabla_{\theta^N} J^{(N)}(\theta^N, x^N) \nabla_{\theta^{N-1}} J^{(N-1)}(\theta^{N-1}, x^{N-1}) \tag{2}$$

Moreover, in a purely convolutional layer, the evaluation function $J^{(l)}(\theta^l, x^l)$ is given by

$$J^{(l)}(\theta^l, x^l) = \mathbf{W}^l x^l + \mathbf{b}^l \tag{3}$$

where $\theta^l = \{\mathbf{W}^l, \mathbf{b}^l\}$ is the parameter set comprised by the weight matrix $\mathbf{W}^l$ and bias vector $\mathbf{b}^l$. It is worth noting that, when combined with an activation function $g^{(l)}(\cdot)$ at the layer indexed l, the function $J^{(l)}(\theta^l, x^l)$ becomes

$$J^{(l)}(\theta^l, x^l) = g^{(l)}\big(\mathbf{W}^l x^l + \mathbf{b}^l\big) \tag{4}$$

III. Max-Variance Pruning

Note that, by substituting Equation 4 into Equation 1, and using x^{l+1} to denote the output of the l^{th} layer and x^{l-1} to denote the input of the layer indexed l we can easily write for arbitrary layer-pair outputs the following expression

$$x^{l+1} = g^{(l)}\Big(\mathbf{W}^l g^{(l-1)}\big(\mathbf{W}^{l-1}x^{l-1} + \mathbf{b}^{l-1}\big) + \mathbf{b}^l\Big) \tag{5}$$

Recall that the activation functions $g^{(l)}$ and $g^{(l-1)}$ are strictly non-decreasing. Thus, maximising $x^{l=1}$ with respect to the parameters is equivalent to maximising the L-2 norm of x^{l+1}, *i.e.* $\|x^{l+1}\|$. This is an important observation since it opens-up the possibility of using the unit-response of the coupling between the two layers so as to remove those pathways in the network that are negligible. This can be done in a straightforward manner by setting $x^{l-1} \equiv 1$ and noting that the rows of W^{l-1} are expected to govern the outputs of the layer indexed $l-1$ whereas the columns of W^l will correspond to the inputs of the l^{th} layer. Thus, we can use the quantity

$$\tau_k = \sum_{i,j} g^{(l)}\Big(W^l_{j,k} g^{(l-1)}(W^{l-1}_{k,i} + b^{l-1}_k) + b^l_j\Big) \tag{6}$$

as a coupling factor between the layers indexed l and $l-1$ for the pathway involving the k^{th} input and the corresponding output. In the expression above, $W^l_{j,k}$ denotes the entry indexed j, k for the weight matrix $\mathbf{W}^l$ corresponding to the l^{th} layer and b^l_j corresponds to the j^{th} element of the bias vector $\mathbf{b}^l$.

The aim is to use the coupling factor τ_k presented above to select the rows of the weight matrix $\mathbf{W}^l$ and the columns of the weight matrix $\mathbf{W}^{l-1}$ to be removed from further consideration. To this end, we define the vector $\mathbf{q} = [\tau_1, \tau_2, \ldots, \tau_N]^T$ for the N rows in the weight matrix $\mathbf{W}^l$ and follow the notion that the M relevant weight rows of $\mathbf{W}^l$ must be those that have a large coupling factor. As first noted by Bradley [23], the back-propagation gradients tend to become smaller as the chain-rule is applied from the output to the input layer. Furthermore, as noted by Glorot and Bengio in [24] in the context of initialisation of neural networks, a constant and non-negligible variance can be used to improve the training step. Thus, we aim at maximising the magnitude of the selected coupling factors while also maximising the variance of the vector $\hat{\mathbf{q}} = [\tau_1, \tau_2, \ldots, \tau_M, 0, 0, \ldots, 0]^T$, where $|\hat{\mathbf{q}}| = |\mathbf{q}|$, where $|\cdot|$ denotes the cardinality of the vector under consideration.

To this end, we aim at solving the following optimisation problem

$$\mathbf{q}^* = \underset{\hat{\mathbf{q}}}{\operatorname{argmax}}\left\{\operatorname{Var}(\hat{\mathbf{q}})\right\}$$
$$\text{s.t.} \quad \tau_1 \geq \tau_2 \geq \tau_3 \geq \cdots \geq \tau_N \tag{7}$$

where M is, as before, the number of non-null elements of $\hat{\mathbf{q}}$ and added a monotonicity condition following the notion that the largest coupling factors should be considered over those that are smaller in rank. Note that, as a consequence of this monotonicity condition, it becomes a straightforward matter to obtain $\mathbf{q}^*$. This can be done by computing the variance for increasing N and selecting those whose value is largest.

IV. Results

We now turn our attention to illustrate the utility of our method for model compression. To this end, we employ widely available neural network models trained using data sets used for both, facial expression and object recognition. We also compare against alternatives elsewhere in the literature.

For our experiments, we have implemented our network in Pytorch and performed all our experiments on a NVIDIA V100 GPU. For purposes of comparing our method with the alternatives and providing quantitative results in line with results reported elsewhere in the literature we have used a number of metrics. These are:

$$\begin{aligned} \beta &= \frac{|\mathcal{M}|}{|\mathcal{M}'|} \\ \alpha &= \frac{1}{\beta} \\ c &= \frac{|\mathcal{M}| - |\mathcal{M}'|}{|\mathcal{M}'|} \end{aligned} \tag{8}$$

where $|\mathcal{M}|$ and $|\mathcal{M}'|$ are the sizes, *i.e.* the number of parameters, of the uncompressed and compressed models $\mathcal{M}$ and $\mathcal{M}'$, respectively.

A. Data Sets

As mentioned above, we use widely available data sets widely used for recognition tasks. The data sets used here are CIFAR-10, CIFAR-100, MNIST and FER-2013. The CIFAR-10 dataset consists of 60000 32x32 colour images separated in 10 classes, of which 50,000 are used for training and the remaining for testing. The CIFAR-100 is organised similarly to the CIFAR-10, the main difference consists in the number of classes, whereby, instead of being organised into 10 categories, the CIFAR-100 is arranged into 100 classes comprising 600 images each [1]. MNIST [2], in the other hand, is a data set containing handwritten digits. It has a training set of 60,000 examples and a test set of 10,000 examples. The digits have been normalised to size and centered on the image.

[1]Both, the CIFAR-10 and CIFAR-100 data sets can be found at https://www.cs.toronto.edu/~kriz/cifar.html

[2]MNIST can be accessed at http://yann.lecun.com/exdb/mnist/

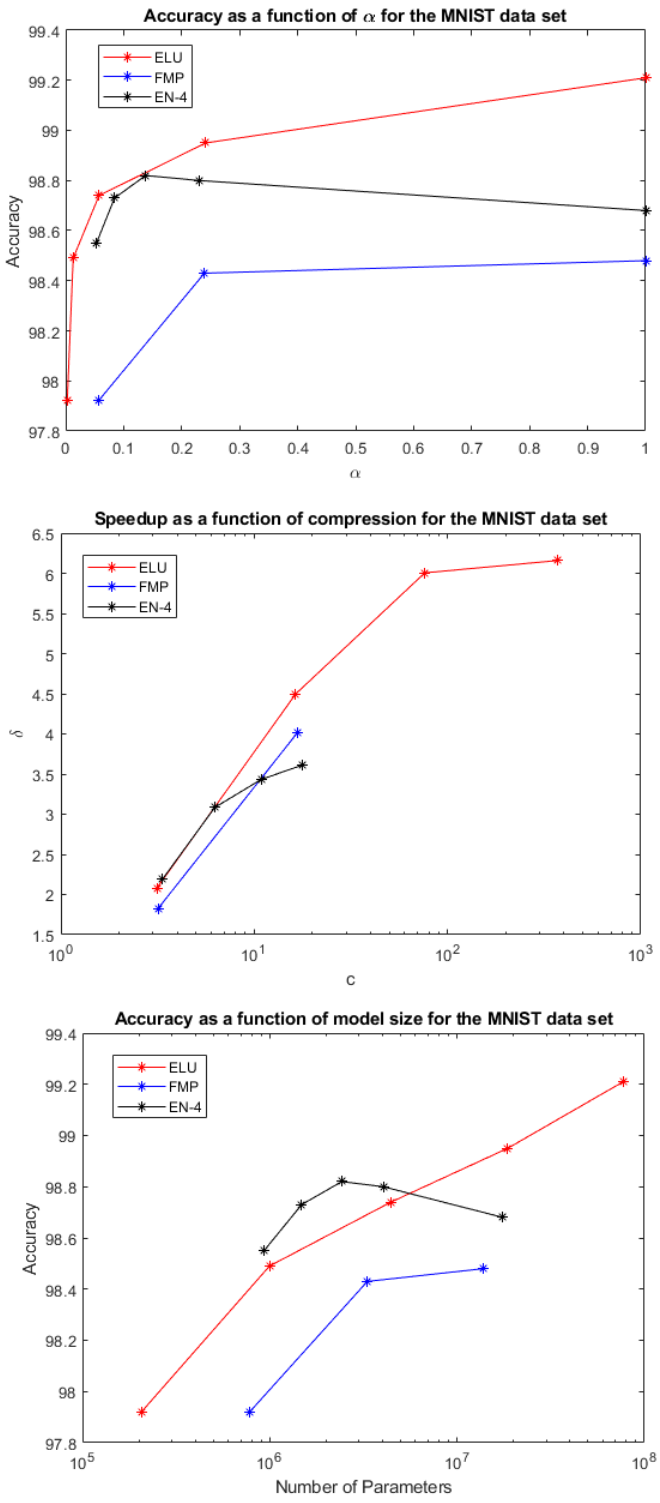

Fig. 2. Results for MNIST

Finally, FER-2013 [3] is a facial expression recognition data set comprising 35,685 grey-scale images of dimensions 48x48 pixels. The images in FER-2013 have been categorised into

[3]The FER-2013 data set is widely available at https://datarepository.wolframcloud.com/resources/FER-2013

Network	Accuracy %	Recursion Step	Number of Parameters	β	α	c	δ
Exponential Linear Units (ELU)	99.21	-	77,013,662	1	1	0	1
	98.95	1	18,614,535	4.1372	0.2417	3.1372	2.0805
	98.74	2	4,433,853	17.3694	0.05757	16.3694	4.4961
	98.49	3	996,288	77.3006	0.0129	76.3006	6.0058
	97.92	4	206,328	373.2584	0.0026	372.2584	6.1591
Fractional Max Pooling (FMP)	98.48	-	13,862,846	1	1	0	1
	98.43	1	3,317,298	4.1789	0.2392	3.1789	1.8244
	97.92	2	780,271	17.7667	0.0562	16.7667	4.0177
EfficientNet B4 (EN-B4)	98.68	-	17,565,682	1	1	0	1
	98.8	1	4,034,934	4.3534	0.2297	3.3534	2.1905
	98.82	2	2,420,403	7.2573	0.1377	6.2573	3.0846
	99.1	3	1,480,318	11.8661	0.0842	10.8661	3.4325
	98.98	4	931,531	18.8567	0.0530	17.8567	3.6147

TABLE I
RESULTS FOR MNIST

Method	Accuracy %	Recursion Step	Number of Parameters	β	α	c
VGG-16	90.15	-	138,357,544	1	1	0
ELU	88.54	-	77,020,574	1	1	0
Li *et al.* [25] (VGG-16)	89.98	-	49,808,716	2.77	0.36	1.77
Molchanov *et al.* [26] (VGG-16)	84.56	-	27,671,508	5	0.2	4
Salama *et al.* [27] (VGG-16)	90.05	-	24,904,357	5.5	0.18	4.555
Ours (ELU)	87.96	1	17,986,356	4.282	0.233	3.2821
EN-B4	96.46	-	17,566,546	1	1	0
Ashok *et al.* [28] (VGG-16)	85.07	-	15,373,060	9	0.1111	8
FMP	89.12	-	13,862,942	1	1	0
Ours (EN-B4)	91.78	1	10,174,939	1.7264	0.5792	0.7264
Ours (FMP)	87.73	1	6,860,892	2.0205	0.4949	1.0205
Ours (EN-B4)	91.75	2	6,731,192	2.6097	0.3831	1.6097
Pahwa *et al.* [29] (VGG-16)	83.95	-	6,651,805	20.8	0.0480	19.8
Ours (ELU)	86.16	2	4,189,153	18.3857	0.0543	17.3857
Ours (EN-B4)	87.29	3	3,621,187	4.8510	0.2061	3.8510
Ours (FMP)	83.36	2	3,187,846	4.3486	0.2299	3.3486

TABLE II
RESULTS FOR CIFAR-10

7 types of expression, *i.e.* happiness, neutral, sadness, anger, surprise, disgust, fear.

B. Experiments

Note that, our pruning algorithm is very general in nature and will naturally adjust the number of filters to be removed based on the vector $\mathbf{q}^*$ in Equation 7 above. This is important since it allows for further pruning operations and increased compression. To this end, we have applied our approach recursively following the training-pruning sequence shown in Figure 1. Thus, after the first training of the network, we compress the model and retrain. We do this recursively, making use of the compressed model at each time.

We commence by illustrating the behaviour of our network. To do this, we have used the MNIST dataset on three different neural networks. These are the exponential linear unit (ELU) network in [30], the fractional max-pooling (FMP) CNN in [31] and the EfficientNet B4 described in [32]. Also, in addition to the measures in Equation 8, we have computed the quantity

$$\delta = \frac{T(\mathcal{M})}{T(\mathcal{M}')}$$

where $T(\mathcal{M})$ and $T(\mathcal{M}^{'})$ are the testing timings for the uncompressed and compressed models, respectively.

In Figure 1 we show, from top-to-bottom, the accuracy as a function of α, speedup δ as a function of the compression coefficient c and the accuracy as a function of the number of parameters for each of the recursion steps applied to the MNIST data set. In the plots, each of the points corresponding to a recursion is plotted using an asterisk. For the bottom two panels, and for the sake of clarity of presentation, we have used logarithmic scales on the independent variable axis corresponding to the compression coefficient c and the number of parameters, respectively. The numerical results for the plots in Figure 1 are shown in Table I.

From the plots in Figure 1 and the numerical data in Table I, we can see that, even for the first application of our compression method, the resulting model size is much smaller (about a quarter in size) than the original one, which, in the table, is the one corresponding to the recursion step indexed zero. As the recursion steps increase, the size of the model can be as small as 2% of the original one. Even for these large magnitudes of compression, *i.e.* c as large as 372, the loss of

Method	**Accuracy %**	**Recursion Step**	**Number of Parameters**	β	α	c
VGG-16	74.11	-	138,357,544	1	1	0
ResNet-50	74.19	-	80,849,750	1	1	0
Hu *et al.* [33] (VGG-16)	72.01	-	48,701,855	2.84	0.352	1.8409
ELU	71.69	-	30,443,620	1	1	0
Hu *et al.* [33] (ResNet-50)	74.07	-	29,105,910	2.7777	0.36	1.7777
EN-B4	85.53	-	17,727,916	1	1	0
FMP	66.45	-	13,862,942	1	1	0
ResNet-18	76.28	-	11,511,784	1	1	0
Ours (EN-B4)	71.43	1	10,336,309	1.7151	0.5831	0.7151
Ours (FMP)	63.22	1	6,901,302	2.0087	0.4978	1.0087
Ours (EN-B4)	70.86	2	6,892,562	2.5720	0.3888	1.5720
Ours (ELU)	70.90	1	6,471,984	4.7039	0.2125	3.7039
Ours (EN-B4)	46.06	3	3,789,274	4.6784	0.2137	3.6784
Ours (FMP)	61.91	2	3,201,141	4.3306	0.2309	3.3306
Ashok *et al.* [28] (ResNet-18)	58.98	-	2,807,752	4.1	0.2439	3.1
Pahwa *et al.* [29] (ResNet-18)	57.39	-	2,616,314	4.4	0.2272	3.4
Ours (ELU)	59.31	2	1,387,085	21.9479	0.04556	20.9479

TABLE III
RESULTS FOR CIFAR-100

Network	**Accuracy %**	**Recursion Step**	**Number of Parameters**	β	α	c
AlexNet	70.60	-	56,916,039	1	1	0
	66.56	1	36,893,815	1.5427	0.6482	0.5427
	64.59	2	24,261,055	2.3460	0.42637	1.3460
	62.55	3	10,889,391	5.2267	0.1913	4.2267
VGG_M	71.41	-	98,860,359	1	1	0
	70.86	1	63,342,587	1.5607	0.6407	0.5607
	70.08	2	41,822,713	2.3638	0.4230	1.3638
	67.32	3	19,210,497	6.9904	0.1431	5.9904
VGG_D	70.55	-	134,289,223	1	1	0
	70.16	1	86,043,404	1.5607	0.6407	0.5607
	69.69	2	56,801,489	2.3642	0.4230	1.3642
	66.17	3	23,5636,58	5.6990	0.1755	4.6990

TABLE IV
RESULTS FOR FER-2013

performance is less than 2%. Moreover, the speedups can also be quite substantial.

We now turn our attention to the effects of our model compression method on image classification tasks involving the CIFAR datasets. As mentioned above, we have used the CIFAR-10 and CIFAR-100 data sets. Again, we have used the ELU [30], the FMP [31] and the EfficientNet B4 [32] networks and computed the measures in Equation 8. For the CIFAR-10, We also have compared our results against a number of alternatives which operate on the VGG-16 network [34]. Our choice of VGG-16 here is so as to be consistent to the compression settings reported by the authors. The results yielded by our method and the alternatives are listed in Table II. In the table, we show the uncompressed networks and denote the target CNN, *i.e.* the net which has been compressed, in parenthesis for all the approaches under consideration. It is worth noting that, for CIFAR-10, our method as applied to the EfficientNet-B4 (EN-B4) delivers the best performance for the compressed models, being more than an order of magnitude smaller than the VGG-16.

For CIFAR-100, we have used the three networks used previously for our method and, for the alternatives, two other CNNs. These are the VGG-16 [34] and two variations of the ResNet [35]. These are the ResNet-18 and the ResNet-50. In Table III, we show the results yielded by our method and the alternatives. As in the previous table, we denote in parenthesis the target CNN. Again, our method is quite competitive, delivering a performance comparable to that of the best performing alternatives at a fraction of model size.

Finally, we present results on the facial expression recognition data set FER-2013. For this data set, we have used two deep face recognition networks [36], *i.e.* VGG_M and VGG_D, and AlexNet [1]. In Table IV, we show the results yielded by our method when three recursive applications of our compression method are made on the three networks under consideration. Our approach fairs particularly well for the VGG networks (VGG_M and VGG_D), with a loss of accuracy of around 1% for two consecutive applications of our compression method. Moreover, for VGG_M, our method can compress the original model to comprise less than 15% of the original number of parameters with a loss in performance of just over 4%.

V. Conclusions

In this paper, we have presented a method for convolutional neural network model compression. The method presented here employs the coupling factor between two consecutive layers so as to remove filter banks that correspond to pathways in the network that are negligible. Moreover, our selection strategy maximises the variance and magnitude of the coupling factors. We have done this following the notion that a large variance with strong unit-response for the network pathway can help prevent the tendency of the back propagation gradients to become smaller as the chain rule is applied at training. We have shown results on a number of networks and data sets that are widely available and compared our results to a number of alternatives.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[2] A. W. Senior, G. Heigold, M. A. Ranzato, and K. Yang, "An empirical study of learning rates in deep neural networks for speech recognition," in *International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6724–6728.

[3] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European Conference on Computer Vision*, 2014.

[4] Y. Taigman, M. Yang, M. A. Ranzato, and L. Wolf, "DeepFace: closing the gap to human-level performance in face verification," in *Computer Vision and Pattern Recognition*, 2014.

[5] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[6] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2181–2191.

[7] S. Lin, R. Ji, C.-N. Chen, D. Tao, and J. Luo, "Holistic cnn compression via low-rank decomposition with knowledge transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 2889–2905, 2018.

[8] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *Advances in Neural Information Processing Systems Deep Learning Workshop*, 2014.

[9] Y. Ioannou, D. P. Robertson, R. Cipolla, and A. Criminisi, "Deep roots: Improving cnn efficiency with hierarchical filter groups," *Computer Vision and Pattern Recognition*, pp. 5977–5986, 2016.

[10] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in Neural Information Processing Systems*, 1989, pp. 177–185.

[11] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 1993, pp. 164–171.

[12] S. S. Du, X. Zhai, B. Póczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *International Conference on Learning Representations*, 2019.

[13] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[14] S. Srinivas and R. Venkatesh Babu, "Data-free parameter pruning for deep neural networks," in *British Machine Vision Conference*, 2015, pp. 31.1–31.12.

[15] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *International Conference on Learning Representations*, 2019.

[16] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[17] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[18] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *European Conference on Computer Vision*, 2016, pp. 662–677.

[19] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference Machine Learning*, 2015, pp. 2285–2294.

[20] H. Li, N. Liu, X. Ma, S. Lin, S. Ye, T. Zhang, X. Lin, W. Xu, and Y. Wang, "ADMM-based weight pruning for real-time deep learning acceleration on mobile devices," in *Great Lakes Symposium on VLSI*, 2019, pp. 501–506.

[21] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[22] P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri, "Play and prune: Adaptive filter pruning for deep model compression," in *International Joint Conference on Artificial Intelligence*, 2019, pp. 3460–3466.

[23] D. M. Bradley, J. A. Bagnell, Y. Bengio, M. Hebert, and F. De and La Torre, "Learning in modular systems," Carnegie Mellon University, Tech. Rep., 2009.

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, ser. JMLR Proceedings, vol. 9, 2010, pp. 249–256.

[25] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *CoRR*, 2016.

[26] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, 2016.

[27] A. Salama, O. Ostapenko, T. Klein, and M. Nabi, "Pruning at a glance: Global neural pruning for model compression," *CoRR*, 2019.

[28] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2N learning: Network to network compression via policy gradient reinforcement learning," in *International Conference on Learning Representationsz*, 2018.

[29] R. Pahwa, M. G. Arivazhagan, A. Garg, S. Krishnamoorthy, R. Saxena, and S. Choudhary, "Data-driven compression of convolutional neural networks," *CoRR*, 2019.

[30] D. A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *International Conference on Learning Representations*, 2016.

[31] B. Graham, "Fractional max-pooling," *CoRR*, 2014.

[32] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97, 2019, pp. 6105–6114.

[33] Y. Hu, S. Sun, J. Li, X. Wang, and Q. Gu, "A novel channel pruning method for deep neural network compression," *CoRR*, 2018.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[36] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, 2015, pp. 41.1–41.12.