THE POWER OF SPARSITY IN CONVOLUTIONAL NEURAL NETWORKS

Soravit Changpinyo *
Department of Computer Science
University of Southern California
Los Angeles, CA 90020, USA
schangpi@usc.edu

Mark Sandler and **Andrey Zhmoginov**
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
{sandler,azhmogin}@google.com

ABSTRACT

Deep convolutional networks are well-known for their high computational and memory demands. Given limited resources, how does one design a network that balances its size, training time, and prediction accuracy? A surprisingly effective approach to trade accuracy for size and speed is to simply reduce the number of channels in each convolutional layer by a fixed fraction and retrain the network. In many cases this leads to significantly smaller networks with only minimal changes to accuracy. In this paper, we take a step further by empirically examining a strategy for deactivating connections between filters in convolutional layers in a way that allows us to harvest savings both in run-time and memory for many network architectures. More specifically, we generalize 2D convolution to use a channel-wise sparse connection structure and show that this leads to significantly better results than the baseline approach for large networks including VGG and Inception V3.

1 INTRODUCTION

Deep neural networks combined with large-scale labeled data have become a standard recipe for achieving state-of-the-art performance on supervised learning tasks in recent years. Despite of their success, the capability of deep neural networks to model highly nonlinear functions comes with high computational and memory demands both during the model training and inference. In particular, the number of parameters of neural network models is often designed to be huge to account for the scale, diversity, and complexity of data that they learn from. While advances in hardware have somewhat alleviated the issue, network size, speed, and power consumption are all limiting factors when it comes to production deployment on mobile and embedded devices. On the other hand, it is well-known that there is significant redundancy among the weights of neural networks. For example, Denil et al. (2013) show that it is possible to learn less than 5% of the network parameters and predict the rest without losing predictive accuracy. This evidence suggests that neural networks are often over-parameterized.

These motivate the research on neural network compression. However, several immediate questions arise: Are these parameters easy to identify? Could we just make the network 5% of its size and retrain? Or are more advanced methods required? There is an extensive literature in the last few years that explores the question of network compression using advanced techniques, including network prunning, loss-based compression, quantization, and matrix decomposition. We overview many of these directions in the next section. However, there is surprisingly little research on whether this over-parameterization can simply be re-captured by more efficient architectures that could be obtained from original architectures via simple transformations.

Our approach is inspired by a very simple yet successful method called depth multiplier (Howard, 2017). In this method the depth (the number of channels) of each convolutional layer in a given network is simply reduced by a fixed fraction and the network is retrained. We generalize this approach by removing the constraint that every input filter (or channel) must be fully connected to every output filter. Instead, we use a sparse connection matrix, where each output convolution chan-

*The work was done while the author was doing an internship at Google Research.

nel is connected only to a small random fraction of the input channels. Note that, for convolutional networks, this still allows for efficient computation since the one channel spatial convolution across the entire plane remains unchanged.

We empirically demonstrate the effectiveness of our approach on four networks (MNIST, CIFAR Net, Inception-V3 and VGG-16) of different sizes. Our results suggest that our approach outperforms dense convolutions with depth multiplier at high compression rates.

For Inception V3 (Szegedy et al., 2016), we show that we can train a network with only about 300K of convolutional parameters[1] and about 100M multiply-adds that achieves above 52% accuracy after it is fully trained. The corresponding depth-multiplier network has only about 41% accuracy. Another network that we consider is VGG-16n, a slightly modified version of VGG-16 (Simonyan & Zisserman, 2015), with 7x fewer parameters and similar accuracy.[2] We found VGG-16n to start training much faster than the original VGG-16 which was trained incrementally in the original literature. We explore the impact of sparsification and the number of parameters on the quality of the network by building the networks up to 30x smaller than VGG-16n (200x smaller than the original VGG-16).

In terms of model flexibility, sparse connections allow for an *incremental* training approach, where connection structure between layers can be densified as training progresses. More importantly, the incremental training approach can potentially speed up the training significantly due to savings in the early stages of training.

The rest of the paper is organized as follows. Section 2 summarizes relevant work. We describe our approach in Section 3 and then present some intuition in Section 4. Finally, we show our experimental results in Section 5.

2 Related Work

2.1 Compression techniques for neural networks

Our work is closely related to a compression technique based on network pruning. However, the important difference is that we do not try to select the connections which are redundant. Instead, we just fix a random connectivity pattern and let the network train around it. We also give a brief overview of other two popular techniques: quantization and decomposition, though these directions are not the main focus and could be complementary to our work.

Network pruning Much initial work on neural network compression focuses on removing unimportant connections using weight decay. Hanson & Pratt (1989) introduce hyperbolic and exponential biases to the objective. Optimal Brain Damage (LeCun et al., 1989) and Optimal Brain Surgeon (Hassibi & Stork, 1993) prune the networks based on second-order derivatives of the objectives. Recent work by Han et al. (2015; 2016a) alternates between pruning near-zero weights, which are encouraged by $\ell 1$ or $\ell 2$ regularization, and retraining the pruned networks.

More complex regularizers have also been considered. Wen et al. (2016) and Li et al. (2016) put structured sparsity regularizers on the weights, while Murray & Chiang (2015) put them on the hidden units. Feng & Darrell (2015) explore a nonparametric prior based on the Indian buffet processes (Griffiths & Ghahramani, 2011) on layers. Hu et al. (2016) prune neurons based on the analysis of their outputs on a large dataset. Anwar et al. (2015b) consider special sparsity patterns: channel-wise (removing a feature map/channel from a layer), kernel-wise (removing all connections between two feature maps in consecutive layers), and intra-kernel-strided (removing connections between two features with particular stride and offset). They also propose to use particle filter to decide the importance of connections and paths during training.

Another line of work explores fixed network architectures with some subsets of connections removed. For example, LeCun et al. (1998) remove connections between the first two convolutional feature maps in a completely uniform manner. This is similar to our approach but they only con-

[1] Here and elsewhere, we ignore the parameters for the softmax classifier since they simply describe a linear transformation and depend on number of classes.

[2] As of this writing, VGG networks have not finished training, but the training trajectory suggest similar performance.

sider a pre-defined pattern in which the same number of input feature map are assigned to each output feature map (Random Connection Table in Torch's SpatialConvolutionMap function). Further, they do not explore how sparse connections affect performance compared to dense networks. Along a similar vein, Cireşan et al. (2011) remove random connections in their MNIST experiments. However, they do not try to preserve the spatial convolutional density and it might be a challenge to harvest the savings on existing hardware. Ioannou et al. (2016a) explore three types of hierarchical arrangements of filter groups for CNNs, which depend on different assumptions about co-dependency of filters within each layer. These arrangements include columnar topologies inspired by AlexNet (Krizhevsky et al., 2012), tree-like topologies previously used by Ioannou et al. (2016b), and root-like topologies. Finally, Howard (2017) proposes the depth multiplier method to scale down the number of filters in each convolutional layer by a factor. In this case, depth multiplier can be thought of channel-wise pruning mentioned in (Anwar et al., 2015b). However, depth multiplier modifies the network architectures before training and removes each layer's feature maps in a uniform manner.

With the exception of (Anwar et al., 2015b; Li et al., 2016; Ioannou et al., 2016a) and depth multiplier (Howard, 2017), the above previous work performs connection pruning that leads to *irregular* network architectures. Thus, those techniques require additional efforts to represent network connections and might or might not allow for direct computational savings.

Quantization Reducing the degree of redundancy of model parameters can be done in the form of quantization of network parameters. Hwang & Sung (2014); Arora et al. (2014) and Courbariaux et al. (2015; 2016); Rastegari et al. (2016) propose to train CNNs with ternary weights and binary weights, respectively. Gong et al. (2014) use vector quantization for parameters in fully connected layers. Anwar et al. (2015a) quantize a network with the squared error minimization. Chen et al. (2015) randomly group network parameters using a hash function. We note that this technique could be complementary to network pruning. For example, Han et al. (2016a) combine connection pruning in (Han et al., 2015) with quantization and Huffman coding.

Decomposition Another approach is based on low-rank decomposition of the parameters. Decomposition methods include truncated SVD (Denton et al., 2014), decomposition to rank-1 bases (Jaderberg et al., 2014), CP decomposition (PARAFAC or CANDECOMP) (Lebedev et al., 2015), Tensor-Train decomposition of Oseledets (2011) (Novikov et al., 2015), sparse dictionary learning of Mairal et al. (2009) and PCA (Liu et al., 2015), asymmetric (3D) decomposition using reconstruction loss of non-linear responses combined with a rank selection method based on PCA accumulated energy (Zhang et al., 2015b;a), and Tucker decomposition using the kernel tensor reconstruction loss combined with a rank selection method based on global analytic variational Bayesian matrix factorization (Kim et al., 2016).

2.2 Regularization of Neural Networks

Hinton et al. (2012); Srivastava et al. (2014) propose Dropout for regularizing fully connected layers within neural networks layers by randomly setting a subset of activations to zero *during training*. Wan et al. (2013) later propose DropConnect, a generalization of Dropout that instead randomly sets a subset of weights or connections to zero. Our approach could be thought as related to DropConnect, but (1) we remove connections *before* training; (2) we focus on connections between convolutional layers; and (3) we kill connections in a more regular manner by restricting connection patterns to be the same along spatial dimensions.

Recently, Han et al. (2016b) and Jin et al. (2016) propose a form of regularization where dropped connections are unfrozen and the network is retrained. This idea is similar to our incremental training approach. However, (1) we do not start with a full network; (2) we do not unfreeze connections all at once; and (3) we preserve regularity of the convolution operation.

2.3 Neural network architectures

Network compression and architectures are closely related. The goal of compression is to remove redundancy in network parameters; therefore, the knowledge about traits that determine architecture's success would be desirable. Other than the discovery that depth is an important factor (Ba & Caruana, 2014), little is known about such traits.

Some previous work performs architecture search but without the main goal of doing compression (Murray & Chiang, 2015; De Brabandere et al., 2016). Recent work proposes shortcut/skip connections to convolutional networks. See, among others, highway networks (Srivastava et al., 2015), residual networks (He et al., 2016a;b), networks with stochastic depth (Huang et al., 2016b), and densely connected convolutional networks (Huang et al., 2016a).

3 APPROACH

A CNN architecture consist of (1) convolutional layers, (2) pooling layers, (3) fully connected layers, and (4) a topology that governs how these layers are organized. Given an architecture, our general goal is to transform it into another architecture with a smaller number of parameters. In this paper, we limit ourselves to transformation functions that keep the general topology of the input architecture intact. Moreover, the main focus will be on the convolutional layers and convolution operations, as they impose highest computational and memory burden for most if not all large networks.

3.1 DEPTH MULTIPLIER

We first give a description of the depth multiplier method used in Howard (2017). Given a hyperparameter $\alpha \in (0, 1]$, the depth multiplier approach scales down the number of filters in each convolutional layers by α. Note that *depth* here refers to the third dimension of the activation volume of a single layer, not the number of layers in the whole network.

Let n_{l-1} and n_l be the number of input and output filters at layer l, respectively. After the operation n_{l-1} and n_l become $\lceil \alpha n_{l-1} \rceil$ and $\lceil \alpha n_l \rceil$ and the number of parameters (and the number of multiplications) becomes $\approx \alpha^2$ of the original number.

The result of this operation is a network that is both $1/\alpha^2$ smaller and faster. Many large networks can be significantly reduced in size using this method with only a small loss of precision (Howard, 2017). It is our belief that this method establishes a strong baseline to which any other advanced techniques should compare themselves. To the best of our knowledge, we are not aware of such comparisons in the literature.

3.2 SPARSE RANDOM

Instead of looking at depth multiplier as deactivating *channels* in the convolutional layers, we can look at it from the perspective of deactivating *connections*. From this point of view, depth multiplier kills the connections between two convolutional layers such that (a) the connection patterns are still the same across spatial dimensions and (b) all "alive" input channels are fully connected to all "alive" output channels.

We generalize this approach by relaxing (b) while maintaining (a). That is, for every output channel, we connect it to a small subset of input channels. In other words, dense connections between a small number of channels become sparse connections between larger number of channels. This can be summarized in Fig. 1. The advantage of this is that the actual convolution can still be computed efficiently because sparsity is introduced only at the outer loop of the convolution operation and we can still take the advantage of the continuous memory layout. For more details regarding implementations of the two approaches, please refer to the Appendix.

More concretely, let n_{l-1} and n_l be the number of channels of layer $l-1$ and layer l, respectively. For a sparsity coefficient α, each output filter j only connects to an α fraction of filters of the previous layer. Thus, instead of having a connectivity matrix W_{sij} of dimension $k^2 \times n_{l-1} \times n_l$, we have a sparse matrix with non-zero entries at $W_{sa_{ij}j}$, where a_{ij} is an index matrix of dimension $k^2 \times \alpha n_{l-1} \times n_l$ and k is the kernel size.

3.2.1 INCREMENTAL TRAINING

In contrast to depth multiplier, a sparse convolutional network defines a connection pattern on a much bigger network. Therefore, an interesting extension is to consider incremental training: we start with a network that only contains a small fraction of connections (in our experiments we use 1% and 0.1%) and add connections over time. This is motivated by an intuition that the network can use learned channels in new contexts by introducing additional connections. The potential practical

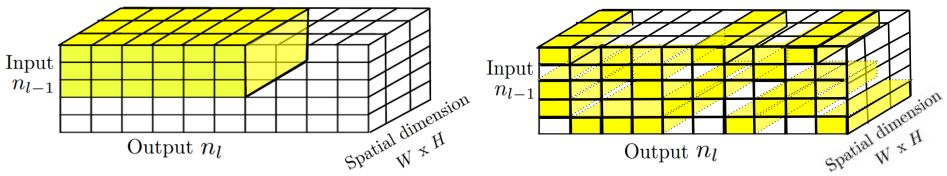

Figure 1: Connection tensors of depth multiplier (left) and sparse random (right) approaches for $n_{l-1} = 5$ and $n_l = 10$. Yellow denotes active connections. For both approaches, the connection pattern is the same across spatial dimension and fixed before training. However, in the sparse random approach, each output channel is connected to a (possibly) different subset of input channels, and vice versa.

advantage of this approach is that since we start training with very small networks and grow them over time, this approach has a potential to speed up the whole training process significantly. We note that depth multiplier will not benefit from this approach as any newly activated connections would require learning new filters from scratch.

4 Analysis

In this section, we approach a question of why sparse convolutions are frequently more efficient than the dense convolutions with the same number of parameters. Our main intuition is that the sparse convolutional networks promote diversity. It is much harder to learn equivalent set of channels as, at high sparsity, channels have distinct connection structure or even overlapping connections. This can be formalized with a simple observation that any dense network is in fact a part of an exponentially large equivalence class, which is guaranteed to produce the same output for every input.

Lemma 1 *Any dense convolutional neural network with no cross-channel nonlinearities, distinct weights and biases, and with l hidden layers of sizes n_1, n_2, ..., n_l, has at least $\prod_{i=1}^{l} n_i!$ distinct equivalent networks which produce the same output.*

Proof Let I denote the input to the network, C_i be the convolutional operator, σ_i denote the non-linearity operator applied to the i-th convolution layer and S be a final transformation (e.g. softmax classifier). We assume that σ_i is a function that operates on each of the channels independently. We note that this is the case for almost any modern network. The output of the network can then be written as:

$$\mathcal{N}(I) \equiv S \circ \sigma_l \circ C_l \circ \sigma_{l-1} \circ \cdots \circ \sigma_1 \circ C_1(I)$$

where we use $\circ$ to denote function composition to avoid numerous parentheses. The convolution operator C_i operates on input with n_{i-1} channels and produces an output with n_i channels. Now, fix arbitrary set of permutation functions π_i, where π_i can permute depth of size n_i. Since π_i is a linear function, it follows that $C'_i = \pi_i^{-1} C_i \pi_{i-1}$ is a valid convolutional operator, which can be obtained from C_i by permuting its bias according to π_i and its weight matrix along input and output dimensions according to π_{i-1} and π_i respectively. For a new network defined as:

$$\mathcal{N}'(I) = S' \circ \sigma_l \circ C'_l \circ \sigma_{l-1} \circ \cdots \circ \sigma_1 \circ C'_1(I),$$

where π_0 is an identity operator and $S' \equiv S \circ \pi_l$, we claim that $\mathcal{N}'(I) \equiv \mathcal{N}(I)$. Indeed, since nonlinearities do not apply cross-depth we have $\pi_n \sigma_n \pi_n^{-1} \equiv \sigma_n$ and thus:

$$\begin{aligned}\mathcal{N}'(I) = S' \circ \sigma_l \circ C'_l \circ \sigma_{l-1} \circ \cdots \circ \sigma_1 \circ C'_1(I) = \\ = S \circ \pi_l \circ \sigma_l \circ \pi_l^{-1} \circ C_l \circ \pi_{l-1} \circ \cdots \circ \pi_1 \circ \sigma_1 \circ \pi_1^{-1} \circ C_1(I) = \mathcal{N}(I).\end{aligned}$$

Thus, any set of permutations on hidden units defines an equivalent network. ■

It is obvious that sparse networks are much more immune to parameter permutation – indeed every channel at layer l is likely to have a unique tree describing its connection matrix all the way down. Exploring this direction is an interesting open question.

5 EXPERIMENTS

In this section, we demonstrate the effectiveness of the sparse random approach by comparing it to the depth multiplier approach at different compression rates. Moreover, we examine several settings in the incremental training where connections gradually become active during the training process.

5.1 SETUP

Networks and Datasets Our experiments are conducted on 4 networks for 3 different datasets. All our experiments use open-source TensorFlow networks Abadi et al. (2015).

MNIST AND CIFAR-10 We use standard networks provided by TensorFlow. For MNIST, it has 3-layer convolutional layers and achieves 99.5% accuracy when fully trained. For CIFAR-10, it has 2 convolutional layers and achieves 87% accuracy.

IMAGENET We use open source Inception-V3 (Szegedy et al., 2016) network and a slightly modified version of VGG-16 (Simonyan & Zisserman, 2015) called VGG-16n on ImageNet ILSVRC 2012 (Deng et al., 2009; Russakovsky et al., 2015).

Random connections Connections are activated according to their likelihood from the uniform distribution. In addition, they are activated in such a way that there are no connections going in or coming out of dead filters (i.e., any connection must have a path to input image and a path to the final prediction.). All connections in fully connected layers are retained.

Implementation details All code is implemented in TensorFlow (Abadi et al., 2015). Deactivating connections is done by applying masks to parameter tensors. The Inception-v3 and VGG-16n networks are trained on 8 Tesla K80 GPUs, each with batch size 256 (32 per gpu) and batch normalization was used for all networks.

5.2 COMPARISON BETWEEN SPARSE RANDOM AND DEPTH MULTIPLIER

5.2.1 MNIST AND CIFAR-10

We first compare depth multiplier and sparse random for the two small networks on MNIST and CIFAR-10. We compare the accuracy of the two approaches when the numbers of connections are roughly the same, based on a hyperparameter α. For dense convolutions, we pick a multiplier α and each filter depth is scaled down by $\sqrt{\alpha}$ and then rounded up. In sparse convolutions, a fraction α of connections are randomly deactivated if those parameters connect at least two filters on each layer; otherwise, a fraction of $\sqrt{\alpha}$ is used instead if the parameters connect layers with only one filter left. The accuracy numbers are averaged over 5 rounds for MNIST and 2 rounds on CIFAR-10.

We show in Fig. 2 and Fig. 3 that the sparse networks have comparable or higher accuracy for the same number of parameters, with comparable accuracy at higher density. We note however that these networks are so small that at high compression rates most of operations are concentrated at the first layer, which is negligible for large networks. Moreover, in MNIST example, the size of network changes most dramatically from 2000 to 2 million parameters, while affecting accuracy only by 1%. This observation suggests that there might be benefits of maintaining the number of filters to be high and/or breaking the symmetry of connections. We explore this in the next section.

5.2.2 INCEPTION-V3 ON IMAGENET

We consider different values of sparsity ranging from 0.003 to 1, and depth multiplier from 0.05 to 1. Our experiments show (see Table 1 and Fig. 4) significant advantage of sparse networks over equivalently sized dense networks. We note that due to time constraints the reported quantitative numbers are preliminary, as the networks have not finished converging. We expect the final numbers to match the reported number for Inception V3 Szegedy et al. (2016), and the smaller networks to have comparable improvement.

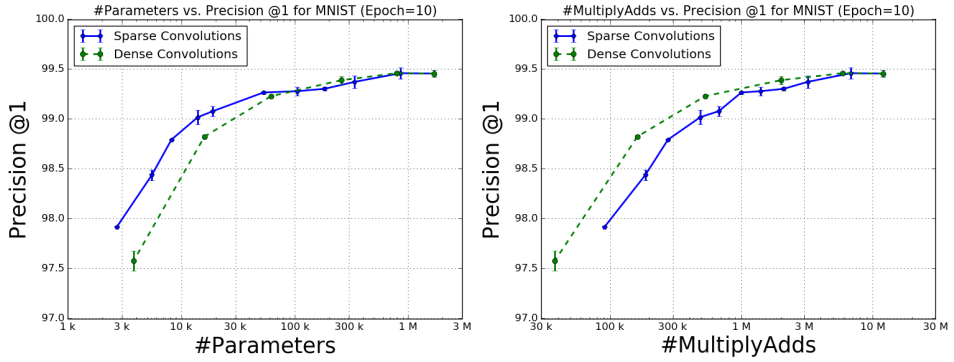

Figure 2: Comparison of accuracy (averaged over 5 rounds) vs. Number of parameters/Number of multiply-adds between dense and sparse convolutions on MNIST dataset. Note that though sparse convolution result in better parameter trade-off curve, the multiply-add curve shows the opposite pattern.

Table 1: Inception V3: Preliminary quantitative results after 100 Epochs. Note the smallest sparse network is actually a hybrid network - we used both depth multiplier (0.5) and sparsity (0.01). The number of parameters is the number of parameters excluding the softmax layer.

Accuracy for sparse convolutions

Sparsity	MAdds	Params	P@1
0.50/0.01	43.0 M	90 k	40.3
0.003	82.0 M	158 k	46.1
0.01	104 M	287 k	52.3
0.03	208 M	724 k	59.5
0.10	628 M	2.3 M	67.2
0.30	1.80 B	6.6 M	73
0.60	3.50 B	13 M	75
1.00	5.70 B	22 M	77

Accuracy for Depth Multiplier

Multiplier	MAdds	Params	P@ 1
0.05	55.0M	56k	24.6
0.10	75.0M	170k	38.6
0.20	183M	718k	54.2
0.30	439M	1.8M	64.0
0.50	1.40B	5.4M	72.3
0.80	3.40B	13M	75.6

Original network: 5.70 B 22 M 77 (78.8)

5.2.3 VGG-16 ON IMAGENET

In our experiments with the VGG-16 network (Simonyan & Zisserman, 2015), we modify the model architecture (calling it VGG-16n) by removing the two fully-connected layers with depth 4096 and replacing them with a 2×2 maxpool layer followed by a 3×3 convolutional layer with the depth of 1024. This alone sped up our training significantly. The comparison between depth multiplier and sparse connection approaches is shown in Fig. 5. The modified VGG-16n network has about 7 times fewer parameters, but appears to have comparable precision.

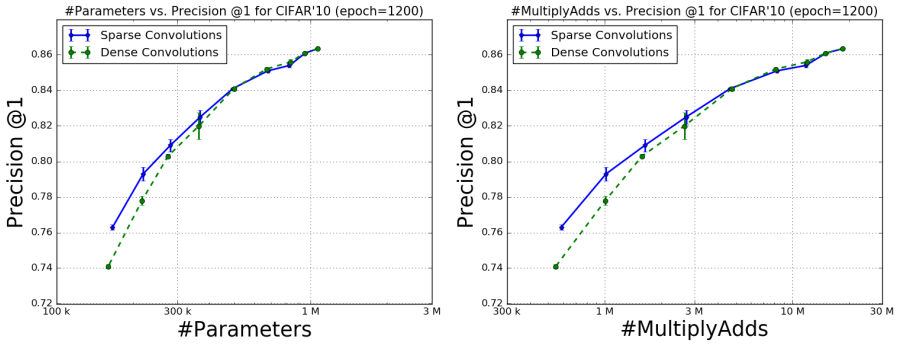

Figure 3: Comparison of accuracy (averaged over 2 rounds) vs. Number of parameters/Number of multiply-adds between dense and sparse convolutions on CIFAR-10 dataset.

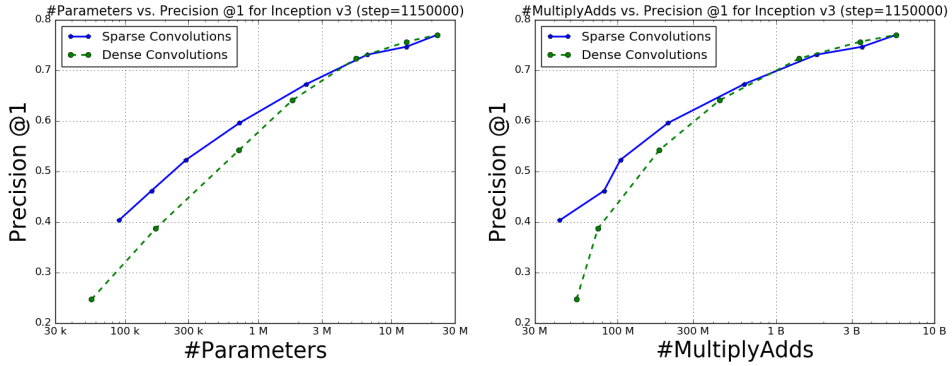

Figure 4: Inception V3: Comparison of Precision@1 vs. Number of parameters/Number of multiply-adds between dense and sparse convolutions on ImageNet/Inception-V3. The full network corresponds to the right-most point of the curve.

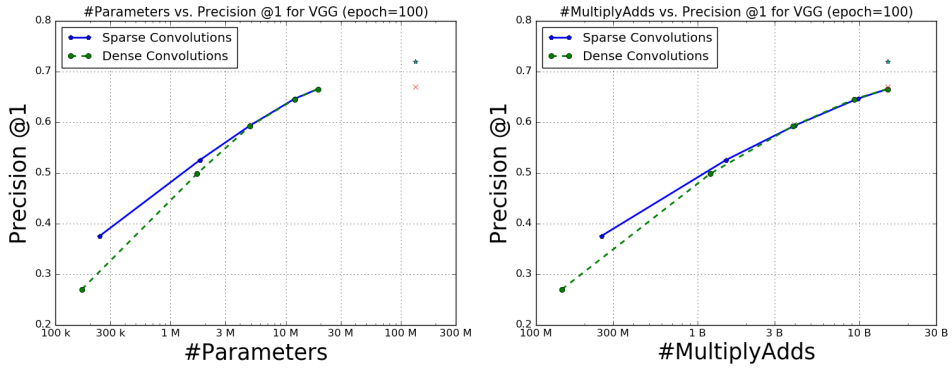

Figure 5: VGG 16: Preliminary Quantitative Results. Comparison of Precision@1 vs. Number of parameters/Number of multiply-adds between dense and sparse convolutions on ImageNet/VGG-16n. The full network corresponds to the right-most point of the curve. Original VGG-16 as described in Simonyan & Zisserman (2015) (blue star) and the same model trained by us from scratch (red cross) are also shown.

5.3 Incremental Training

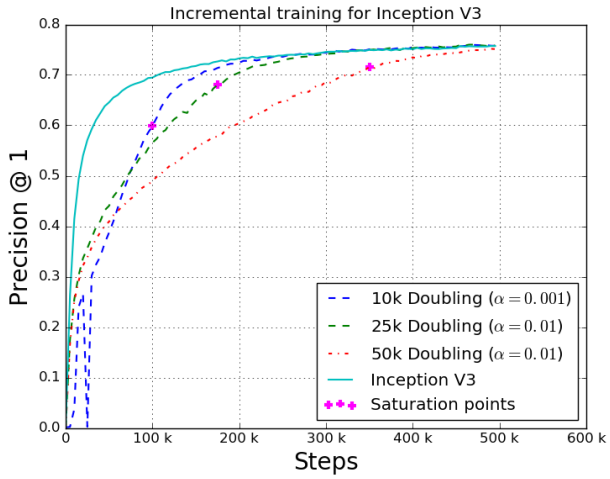

Figure 6: Incremental Training Of Inception V3: We show Precision@1 during the training process, where the networks densify over time. The saturation points show where the networks actually reach their full density.

Finally, we show that incremental training is a promising direction. We start with a very sparse model and increase its density over time, using the approach described in Sect. 3.2.1. We note that a naive approach where we simply add filters results in training process basically equivalent to as if it started from scratch in every step. On the other hand, when the network *densifies* over time, all channels already possess some discriminative power and that information is utilized.

In our experiments, we initially start training Inception-V3 with only 1% or 0.1% of connections enabled. Then, we double the number of connections every T steps. We use $T = 10,000$, $T = 25,000$ and $T = 50,000$. The results are presented in Fig. 6. We show that the networks trained with the incremental approach regardless of the doubling period can catch up with the full Inception-V3 network (in some cases with small gains). Moreover, they recover very quickly from adding more (untrained) connections. In fact, the recovery is so fast that it is shorter than our saving interval for all the networks except for the network with 10K doubling period (resulting in the sharp drop). We believe that incremental training is a promising direction to speeding up the training of large convolutional neural networks since early stages of the training require much less computation.

6 Conclusion

We have proposed a new compression technique that uses a sparse random connection structure between input-output filters in convolutional layers of CNNs. We fix this structure before training and use the same structure across spatial dimensions to harvest savings from modern hardware. We show that this approach is especially useful at very high compression rates for large networks. For example, this simple method when applied to Inception V3 (Fig. 4), achieves AlexNet-level accuracy (Krizhevsky et al., 2012) with fewer than 400K parameters and VGG-level one (Fig. 5) with roughly 3.5M parameters. The simplicity of our approach is instructive in that it establishes a strong baseline to compare against when developing more advanced techniques. On the other hand, the uncanny match in performance of dense and equivalently-sized sparse networks with sparsity > 0.1 suggests that there might be some fundamental property of network architectures that is controlled by the number of parameters, regardless of how they are organized. Exploring this further might yield additional insights on understanding neural networks.

In addition, we show that our method leads to an interesting novel incremental training technique, where we take advantage of sparse (and smaller) models to build a dense network. One interesting open direction is to enable incremental training not to simply densify the network over time, but also increase the number of channels. This would allow us to grow the network without having to fix its original shape in place.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *ICASSP*, 2015a.

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *arXiv preprint arXiv:1512.08571*, 2015b.

Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *ICML*, 2014.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, 2014.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In *NIPS*, 2016.

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

Misha Denil, Babak Shakibi, Laurent Dinh, Marc Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *NIPS*, 2013.

Emily L. Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

Jiashi Feng and Trevor Darrell. Learning the structure of deep convolutional networks. In *ICCV*, 2015.

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

Thomas L. Griffiths and Zoubin Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12(Apr):1185–1224, 2011.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016a.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William J. Dally. Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. *arXiv preprint arXiv:1607.04381*, 2016b.

Stephen José Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *NIPS*, 1989.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016b.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Andrew G. Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *forthcoming*, 2017.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016a.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016b.

Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *IEEE Workshop on Signal Processing Systems (SiPS)*, 2014.

Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. *arXiv preprint arXiv:1605.06489*, 2016a.

Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. In *ICLR*, 2016b.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods. *arXiv preprint arXiv:1607.05423*, 2016.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *ICLR*, 2016.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR*, 2015.

Yann LeCun, John S. Denker, Sara A. Solla, Richard E. Howard, and Lawrence D. Jackel. Optimal brain damage. In *NIPS*, 1989.

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.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *CVPR*, 2015.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *ICML*, 2009.

Kenton Murray and David Chiang. Auto-sizing neural networks: With applications to n-gram language models. In *EMNLP*, 2015.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *NIPS*, 2015.

Ivan V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Rupesh K. Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *NIPS*, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015a.

Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *CVPR*, 2015b.

A ADDITIONAL DETAILS ON DENSE VS. SPARSE CONVOLUTIONS

We contrast naive implementations of dense and sparse convolutions (cf. Sect. 3) in Algorithm. 1 and Algorithm 2. We emphasize that we do not use sparse matrices and only introduce sparsity from channel to channel. Thus, walltime will be mostly in terms of Multiply-Adds; the basic operation (convolving the *entire* image plane in Line 8 of both algorithms) is unchanged.

Algorithm 1 Naive implementation of dense convolution

1: Inputs:
2: - $input$: Data tensor
3: - W: Parameter tensor
4: - $input\_channels$: Array of input channel IDs
5: - $output\_channels$: Array of output channel IDs
6: **for** i in $input\_channels$ **do**
7: **for** o in $output\_channels$ **do**
8: $output[o] \leftarrow output[o]$ + convolve$(input[i], W[i, o, \ldots])$
9: **end for**
10: **end for**
11: **return** $output$

Algorithm 2 Naive implementation of sparse convolution

1: Inputs:
2: - $input$: Data tensor
3: - W: Parameter tensor
4: - $input\_channels$: Array of input channel IDs
5: - $output\_channels\_connected\_to\_i$: Array of array of output channel IDs specifying connections to each input channel
6: **for** i in $input\_channels$ **do**
7: **for** $index$, o in enumerate($output\_channels\_connected\_to\_i[i]$) **do**
8: $output[o] \leftarrow output[o]$ + convolve$(input[i], W[i, index, \ldots])$
9: **end for**
10: **end for**
11: **return** $output$