SlimNets: An Exploration of Deep Model Compression and Acceleration

Chris Sweeney, Subby Olubeko, Ini Oguntola

May 19, 2018

Abstract: Deep neural networks have achieved increasingly accurate results on a wide variety of complex tasks. However, much of this improvement is due to the growing use and availability of computational resources (e.g use of GPUs, more layers, more parameters, etc). Most state-of-the-art deep networks, despite performing well, over-parameterize approximate functions and take a significant amount of time to train. With increased focus on deploying deep neural networks on resource constrained devices like smart phones, there has been a push to evaluate why these models are so resource hungry and how they can be made more efficient. This work evaluates and compares three distinct methods for deep model compression and acceleration: weight pruning, low rank factorization, and knowledge distillation. Comparisons on VGG nets trained on CIFAR10 show that each of the models on their own are effective, but that the true power lies in combining them. We show that by combining pruning and knowledge distillation methods we can create a compressed network *83 times smaller than the original*, all while retaining 97% of the original model's accuracy.

1. Introduction

In 1998 the state of the art convolutional net was *LeNet*, a model with 60 thousand parameters [1]. By 2014 that number had increased to 138 million, with new neural networks like VGG becoming increasingly large [2]. Today we have GPUs that can perform trillions of operations per second to help use train and use models with millions of parameters. The development of the computional means necessary to process such huge networks has facilitated many of the advances in the field, and continues to do so today. However in situations where resources are constrained, such as deployment on mobile devices, it is important to make these models more efficient.

This paper aims to explore various approaches to model compression and acceleration, comparing and evaluating results to determine how we can make deep

neural networks use fewer resources. We use VGG19 as our reference network, and CIFAR10 as our dataset.

2. Prior Work

Most of the existing work in the area of model compression and accelerations falls into one of four approaches: *pruning*, *low-rank factorization*, *convolutional filtering*, and *knowledge distillation*. We describe the general approaches in more detail below.

2.1. Pruning

Pruning methods for deep neural networks all focus on removing redundant or unimportant weights to create models with a smaller storage sizes. Deciding which weights are unimportant is the central question in this area of research. Before the recent wave of deep convolutional neural networks, researchers investigated the effect on the loss function (via a second order Taylor approximation) when setting certain weights to 0 [3][4]. These works used these notions to determine the "saliency" of particular weights and pruned weights of low saliency or importance. As neural networks have become larger and and deeper, the focus in network pruning research has shifted to simply pruning weights with low absolute values. This shift in research happened mostly due to the need for a simpler pruning scheme that is more efficient in the face of deep neural networks with millions of parameters. Nonetheless, absolute values of weights in a network serve as a good proxy for how much effect a particular weight has on the loss function. In the classic Stochastic Gradient Descent update equation for a particular weight at timestamps, w_t

$$w_{t+1} = w_t + \alpha \nabla_w L(w_t) \tag{1}$$

Weights that have a greater effect the loss function, $L(\cdot)$, will have a larger gradient term, and will move more each training step. The higher velocity of these weights gives them a higher probability of having larger absolute values.

There are also many methods that prune entire convolutional filters at a time using some heuristics involving absolute values of the convolution kernel and the gradients for all those weights,[5][6], but for this work we choose to focus on single weight pruning scheme described in [7]. We now describe this method in full.

[7] presents a method for gradually pruning a pretrained neural network to a layer-wise target weight sparsity (certain percentage of weights in each layer masked out). Instead of pruning a pretrained deep neural network all at once, the authors of the paper claim that gradually pruning the weights during training allows the network to dynamically recover from lost weights. They present a pruning scheduler to calculate how much weights in a given layer to prune per

training step. [7]

$$s_t = s_f + (s_i - s_f)(1 - \frac{t - t_0}{n\Delta t})^3 \text{ for } t \in \{t_0, t_0 + \Delta t, ..., t_0 + n\Delta t\} \quad (2)$$

where s_t is target sparsity for the current step, s_f is the final sparsity target, s_i the initial sparsity, t the current time step, t_0 the initial training step, n the number of pruning steps, and Δt the number of training steps in between each pruning step. Testing is done with VGG, reporting as much 92.5 percent top-1-accuracy on the ImageNet data set with the Inception V3 neural network.

2.2. Low-Rank Factorizationn

The goal of low-rank factorization methods for CNNs is to speed up and compress a network by eliminating redundancy in the 4D tensors that serve as its convolutional kernels. The method we use in our experiment relies on decomposing the tensors via SVD and using the results to compute low-rank approximations of the kernels. This approach was inspired by the paper by Tai et. al [8], in which the authors present an algorithm to compute an exact closed form solution of the low-rank tensors. The algorithm builds a low-rank constrained network with approximated kernels derived from the weights of an unconstrained pretrained model. The way we constrain the rank of the new model is by replacing each convolutional layer from the orginial model with two new layers, V and H. If the original layer has C inputs and N outputs, then V will have C inputs and K outputs, while H will have K inputs and N outputs, where $K < N$. This constrains the rank of the layer to be K. Each convolutional layer in the network gets a particular K value which is a hyperparameter that can be learned during training. This parametrization of the network is depicted by the figure below:

Next, we approximate the kernels using the scheme described by Tai et. al,

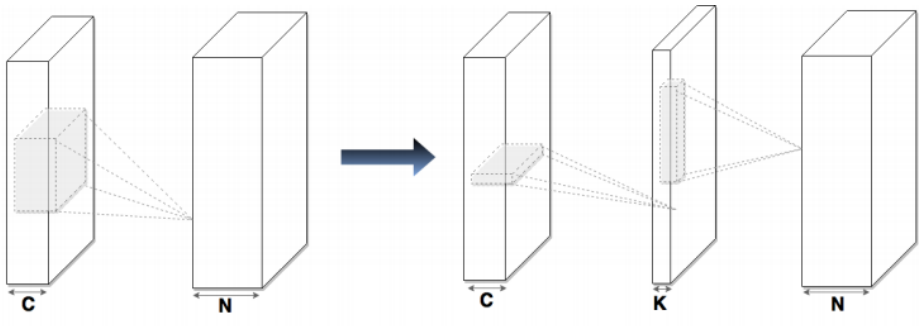


Figure 1: The intermediate value K constrains the rank of the new convolutional layer. Figure taken from Tai et. al [8]

which works as following:

Define $\mathcal{W}$ to be the $C \times d \times d \times N$ weight tensor associated with a given convolutional layer and $\mathcal{T} : \mathbb{R}^{C \times d \times d \times N} \to \mathbb{R}^{Cd \times dN}$ to be a bijection that maps tensor T to matrix M according to the rule $T_{i_1,i_2,i_3,i_4} = M_{j_1,j_2}$ where

$$j_1 = (i_1 - 1)\, d + i_2,\ j_2 = (i_4 - 1)\, d + i_3. \tag{3}$$

Let $W := \mathcal{T}[\mathcal{W}]$ and let $W = USV^T$ be the Singular Value Decomposition of W. We compute the weight kernels for V and H according to:

$$\hat{\mathcal{V}}_k^c\,(j) = U_{(c-1)d+j,\,k}\sqrt{S_{k,k}} \tag{4}$$

$$\hat{\mathcal{H}}_n^k\,(j) = V_{(n-1)d+j,\,k}\sqrt{S_{k,k}} \tag{5}$$

Where $\hat{\mathcal{V}}$ and $\hat{\mathcal{H}}$ are the kernels for V and H respectively. Finally, we train the new low-rank model starting with the approximated weights. The many added layers in the new model make it susceptible to the problem of exploding gradients, so we make use of the technique of batch normalization on all of the H layers in order to combat this. The rank-constrained model was trained and validated over 300 epochs and we kept track of the accuracy, loss, and elapsed time over each epoch. Our experiment results showed significant improvements in the memory and time consumed by the low-rank constrained model as well as improvement in test accuracy.

2.3. Knowledge Distillation

The basic idea behind knowledge distillation is to train a smaller student model to reproduce the output of a larger teacher model. Instead of outright training the smaller model on the original data, we aim to somehow take advantage of the function already learned by the teacher model and transfer knowledge to the student.

Most neural network classifiers have a "softmax" output layer that produces probabilities for each class. Given a logit score z_i, we obtain probability p_i via the following equation:

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{6}$$

where usuually we have $T = 1$. One of the earliest examples of knowledge distillation for deep networks was suggested by Caruana et al [9]. They passed the input through the original model and used the output of this softmax layer as synthetic labels for a compressed model. Building upon this, Hinton et al. [10] use a softened version of the teacher output with higher values of T, and train the compressed model to predict both the teacher output and the true classification labels.

One technique – *FitNets* – proposed by FGSD et al. [11], tries to utilize the intermediate representations of the teacher network to train student models that are both thinner and deeper than the teacher; another technique uses conditional adversarial networks to learn the teacher function [12]. Recent work has also tackled data-free knowledge distillation in the case that the original data is not available to student models [13].

In this paper we primarily focus on the approach from [10] and variations.

2.4. Convolutional Filters

Transferring convolutional filters can also be used as a method of model compression. Let us have input x, network or layer Φ, and transform matrix $\mathcal{T}$. Then the network is structure preserving or "equivariant" if the following holds:

$$\Phi(\mathcal{T}x) = \mathcal{T}'\Phi(x) \tag{7}$$

where $\mathcal{T}$ and $\mathcal{T}'$ are note necessarily the same. The idea proposed Cohen et al. [14] is to build a network consisting of convolutional layers structured this way in order to facilitate weight sharing and increase the expressiveness of the network without significant extra computations overhead for translations, rotations and reflections.

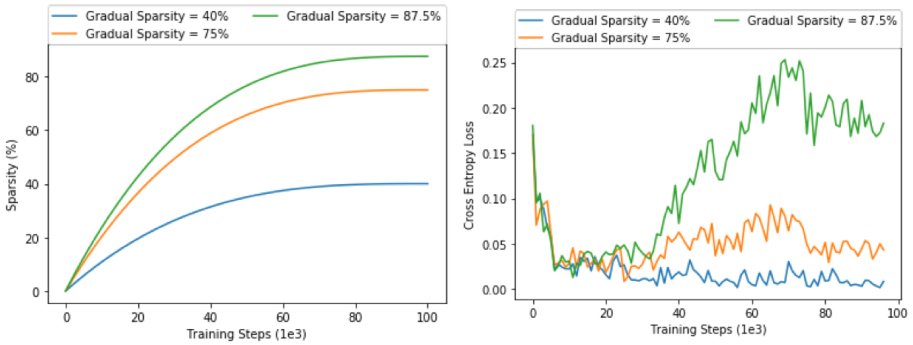
One thing to note is that this approach to model compression can only be used on convolutional layers, where the other three aforementioned approaches can be used on both convolutional and fully-connected layers. For this reason we do not focus on transferring convolutional filters in this work.

3. Experiments

We perform sparse gradual pruning, low rank matrix factorization, and knowledge distillation on a pretrained VGG19 network trained on CIFAR10. We compare the evolution of training, as well as model compression rate, inference time speed ups, and overall top-1%-accuracies. The pretrained VGG19 model we use has a test accuracy of 92.4%, and we show that you can have very competitive accuracies with the pretrained model with models that are orders of magnitude smaller and quicker.

3.1. Pruning

We prune a pretrained VGG19 net, using the before-mentioned gradual sparse training algorithm, for 40%, 75%, and 87.5% percent layer-wise target sparsity. We compare the resulting top-1%-test accuracies of the gradually pruned models to the same pretrained network pruned all at once then trained for the same duration (100,000 training steps). Below are the final network accuracy results for our experiments.



(a) Sparsity over Training Duration

(b) Loss over Training Duration

Figure 2: Evolution of Network Sparsity vs Loss

Method	**Accuracy**
Non-Gradual Pruning 40% sparse	89.69
Gradual Pruning 40% sparse	92.22
Non-Gradual Pruning 75% sparse	89.13
Gradual Pruning 75% sparse	**91.57**
Non-Gradual Pruning 87.5 % sparse	86.16
Gradual Pruning 87.5 percent	88.82

* where **bold** is the best tradeoff of accuracy/model compression

Clearly, introducing sparsity into the model gradually helps to yield higher test accuracies. The reasoning used in [7] is that gradually pruning, especially with a quicker pruning rate earlier on in training allows the network to "heal" from damage done before the network converges. This line of reasoning is supported by the data, but a deeper analysis into how missing weights affect the network from an optimization standpoint is needed to complete the story.

Analysis on how the loss evolves during pruning and training gives interesting insights into how parameters are used in networks in the first place. Figure 2 shows the evolution of sparsity and loss over time for the gradual pruning method.

Surprisingly, when gradually pruning with a larger target sparsity, there is a point at the 60,000th training step where the loss shoots up, particularity for the 87.5% sparse model. This characteristic was also noted in the [7]. This is most likely due to the fact that when the network settles into deeper local/global minimum and relies more on certain weights later in the training stage, pruning these weights cause very large effects in the loss function. Interestingly, the network is still able to recover a little bit for the 87.5% sparsity and completely recover in the 75% sparsity model. These results give an interesting insight into

how the network depends on weights throughout the training process.

Some differences were also noted between [7] and our experiments. The 87.5% sparse model for inception V3 was able to fully recover from the sudden increase in loss during training, while our model was not very good at recovering. Perhaps the constant update from earlier stage inputs in Inception V3 make it more robust to parameter pruning over VGG.

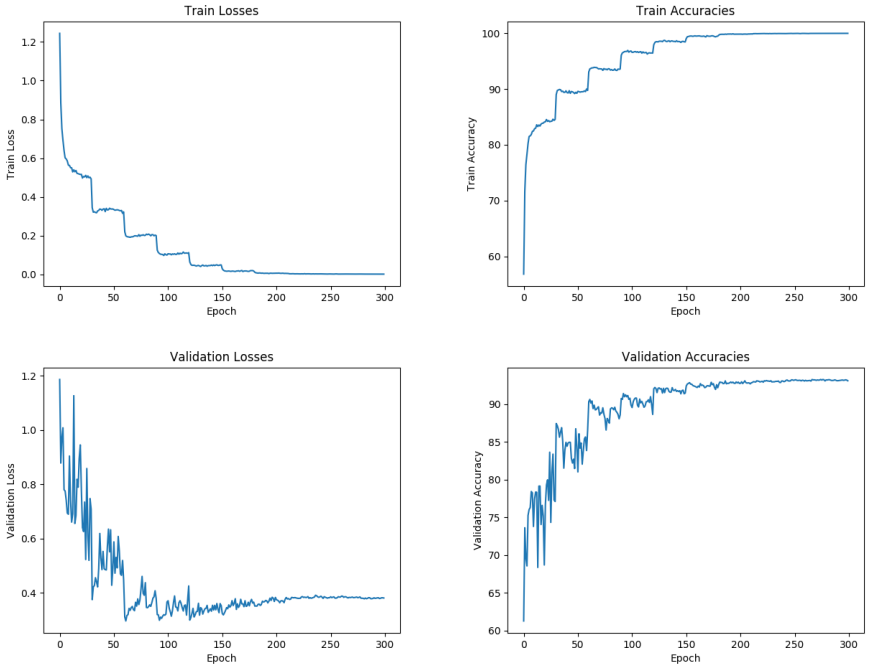
3.2. Low Rank Matrix Factorization

We trained a low rank constrained VGG19 net with batch normalization and the following architecture on the CIFAR10 dataset.

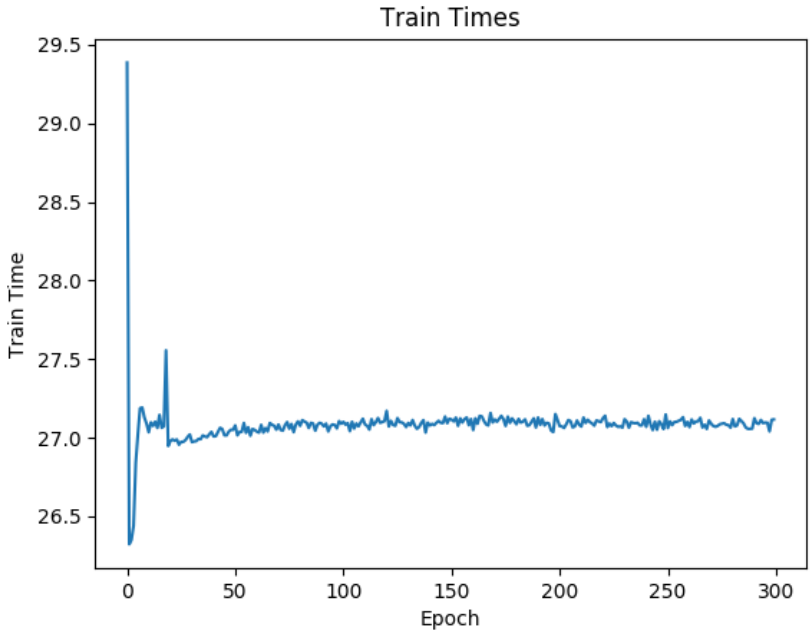
Layer	Dimensions
Conv1_v	3×5
Conv1_h	5×64
Conv2_v	64×24
Conv2_h	24×64
Conv3_v	64×48
Conv3_h	48×128
Conv4_v	128×48
Conv4_h	48×128
Conv5_v	128×64
Conv5_h	64×256
Conv6_v	256×128
Conv6_h	128×256
Conv7_v	256×128
Conv7_h	128×256
Conv8_v	256×160
Conv8_h	160×256
Conv9_v	256×192
Conv9_h	192×512
Conv10_v	512×256
Conv10_h	256×512
Conv11_v	512×320
Conv11_h	320×512
Conv12_v	512×320
Conv12_h	320×512
Conv13_v	512×320
Conv13_h	320×512
Conv14_v	512×320
Conv14_h	320×512
Conv15_v	512×320
Conv15_h	320×512
Conv16_v	512×320

Conv16_h	320×512

The loss and accuracy curves were as follows:



The evolution of the training time per epoch is depicted below:



After 274 epochs of training, the rank-constrained VGG19 achieved a highest test accuracy of 93.27% which surpasses the 92.43% achieved by the unconstrained model. The inference time of the trained model over the whole dataset was 2.315 seconds compared to 2.363 seconds for the unconstrained net. Although there was little speedup in inference time, we saw that weights of the low-rank model, which used 32.3 MB on disk, were compressed to about 41.2% of the size of those of the unconstrained model which took 78.4 MB on disk. These results align with those in the paper by Tai et. al, as the authors also saw improvement across the board in their rank-constrained model.

3.3. Knowledge Distillation

We take the approach detailed in [10], training various smaller neural networks to approximate the output of a pretrained VGG19 model trained on the CIFAR10 dataset and to reproduce the correct labels. Input images are passed through the VGG19 model to collect the scores it assigns across all of the classes as synthetic data; this synthetic data is then used as targets for each of the smaller models.

3.3.1. Model Architectures

We examine six different model architectures. The first two – "SNN-1k" and "SNN-10k" – consist of two convolutional layers and a single hidden layer with 1000 and 10000 hidden units respectively. The next two – "CNN-1k" and "CNN-10k" – have an additional hidden layer with 84 hidden units inserted before

the output layer. We also train on *AlexNet*, a deep convolutional net with 6 convolutional layers, as detailed in [15]. Finally we have the deep convolutional *Network in Network*, or "NIN", consisting of 9 convolutional layers, as detailed in [16].

3.3.2. Training Procedure

We train all of these models for 300 epochs, both on the original labels and with modified loss as described in [10].

3.3.3. Results

Model	**# Parameters**	**Accuracy**	**Accuracy w/ KD**
VGG19	$\sim$ 20M	92.4	N/A
SNN-1k	$\sim$ 415k	76.96	77.87
SNN-10k	$\sim$ 4M	77.17	79.19
CNN-1k	$\sim$ 490k	79.07	81.23
CNN-10k	$\sim$ 4.9M	80.85	82.23
AlexNet	$\sim$ 2.5M	72.27	80.78
NIN	**$\sim$ 965k**	**88.88**	**90.57**

* where **bold** is the best trade-off of accuracy/model compression

As we can see from the table above, training with synthetically labeled data from the teacher model consistently results in improved accuracy over simply training from the original labels alone. We also see that the student network accuracy is highly dependent on model architecture; we usually see high accuracy with knowledge distillation for student models that already performed fairly well on the original data. Making nets wider by adding more units has little effect on either the original accuracy or accuracy with knowledge distillation. It is not surprising to see that, taking into consideration both accuracy and model compression, the best performing model on the original data was also best the best performing model with knowledge distillation.

One interesting thing to note is the large jump in accuracy with AlexNet; its performance improved by almost 12% with knowledge distillation, where the next closest model only improved by under 3%.

4. Quantitative Comparison between Methods

Below we stack up the model sizes and inference times for the result of each method with the best accuracy/model size trade-off. Although this trade-off can be subjective, we choose the results with the best accuracy to model size ratio.

Method	Compr. Rate	Acc.	Inference Speed-up
Sparse Pruning	1.333	91.57	0
Low-Rank Factorization	2.427	93.27	0.048
Knowledge Distillation	20.7	90.57	0.302

From a pretrained VGG19 network with a 92.4% test accuracy we can achieve as much as a 20x reduction in model weights using knowledge distillation and a .302 second speed up in runtime inference with only a few percent degradation in accuracy. If we sacrifice a smaller model size with the low matrix factorization model with a compression rate of 2.427, we can achieve a 93.27 % accuracy on the CIFAR test set.

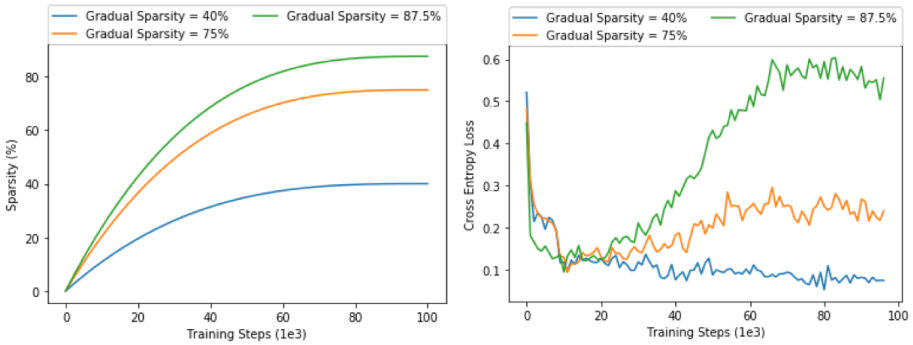
Sparse pruning does not have an immediate solution for how to speed up inference time after pruning weights in the network. Unlike Knowledge Distillation and Low Matrix Factorization, our implementation of gradual pruning simply masks pruned weights to 0, so the architecture is not changed and thus no inference time speedup. Special hardware would be needed to take advantage of the sparsity in the network, or a method for merging sparse layers into one layer that can perform the tasks of both. Although sparse pruning of networks do not offer a new model architecture with quicker inference time, this method is extremely versatile. One can apply gradual pruning to any network, including networks that were compressed using knowledge distillation.

We gradually pruned our best trained knowledge distillation model with sparsity target 40%, 75% and 87.5%. The results of this experiment are displayed below.

Method	Accuracy
Gradual Pruning KD best model 87.5% sparse	85.44
Gradual Pruning KD best model 75% sparse	88.69
Gradual Pruning KD best model 40% sparse	89.29

There is more room for improvement! Even with 25% of the weights of the knowledge distilled model, we can still achieve an accuracy of 88.69 %, compared to the accuracy of the non-sparse model, 90.57%. This is a whopping compression rate of 83 from our original VGG19 model. It is furthermore enlightening to look at the loss curves over training for gradually pruning the knowledge distilled model.

The loss curves for gradually pruning the knowledge distilled model are similar to those for the gradually pruned VGG19 model. There are some notable differences however. For the 87.5% sparse model, the network does not recover at all from the characteristic jump in loss during pruning. Additionally, the loss curves for the 40% and 75% curves are further apart than in the VGG19 graphs. The intuition for this observation is that the since the knowledge distilled model is more efficient in its use of weights, it has less "pruning flexibility" or fewer weights that are meaningless to the classification task. This explains



(a) Sparsity over Training Duration

(b) Loss over Training Duration

Figure 3: Evolution of Network Sparsity vs Loss for Best Knowledge Distillation Network

why the 87.5% and 75% sparse curves do not recover like in the VGG19 case. This results in lower test accuracies after training.

5. Discussion

A particularly surprising result is the fact that given the original VGG model with 92.4% accuracy, we were not only able to achieve 91.6% accuracy with a distilled model over 20 times smaller, but that even this distilled model could be pruned to achieve 89.3% accuracy with a model over 80 times smaller than the original! This highlights two things.

The first is that all of these approaches are not disparate; they can work together and be combined for some incredible rates of compression and speedup. The approaches all target different aspect of deep network inefficiency, and combined we can achieve multiplicative gains.

The second thing this tells us is just how inefficient state-of-the-art deep networks can be; there is room for future work in creating better learning algorithms for smaller networks.

6. Acknowledgements

Thank you to 6.883 staff for instrumental feed back on this project and a great class in Deep Learning.

7. Supplemental Material

Our code for all the methods implemented in this project can be found at `https://github.com/ChristopherSweeney/SlimNets`.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[3] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf

[4] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE International Conference on Neural Networks*, 1993, pp. 293–299 vol.1.

[5] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016. [Online]. Available: http://arxiv.org/abs/1611.06440

[6] ——, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016. [Online]. Available: http://arxiv.org/abs/1611.06440

[7] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *ArXiv e-prints*, Oct. 2017.

[8] C. Tai, T. Xiao, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," *CoRR*, vol. abs/1511.06067, 2015. [Online]. Available: http://arxiv.org/abs/1511.06067

[9] L. J. Ba and R. Caurana, "Do deep nets really need to be deep?" *CoRR*, vol. abs/1312.6184, 2013. [Online]. Available: http://arxiv.org/abs/1312.6184

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[11] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *CoRR*, vol. abs/1412.6550, 2014. [Online]. Available: http://arxiv.org/abs/1412.6550

[12] Z. Xu, Y. Hsu, and J. Huang, "Learning loss for knowledge distillation with conditional adversarial networks," *CoRR*, vol. abs/1709.00513, 2017. [Online]. Available: http://arxiv.org/abs/1709.00513

[13] R. G. Lopes, S. Fenu, and T. Starner, "Data-free knowledge distillation for deep neural networks," *CoRR*, vol. abs/1710.07535, 2017. [Online]. Available: http://arxiv.org/abs/1710.07535

[14] T. S. Cohen and M. Welling, "Group equivariant convolutional networks," *CoRR*, vol. abs/1602.07576, 2016. [Online]. Available: http://arxiv.org/abs/1602.07576

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[16] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013. [Online]. Available: http://arxiv.org/abs/1312.4400