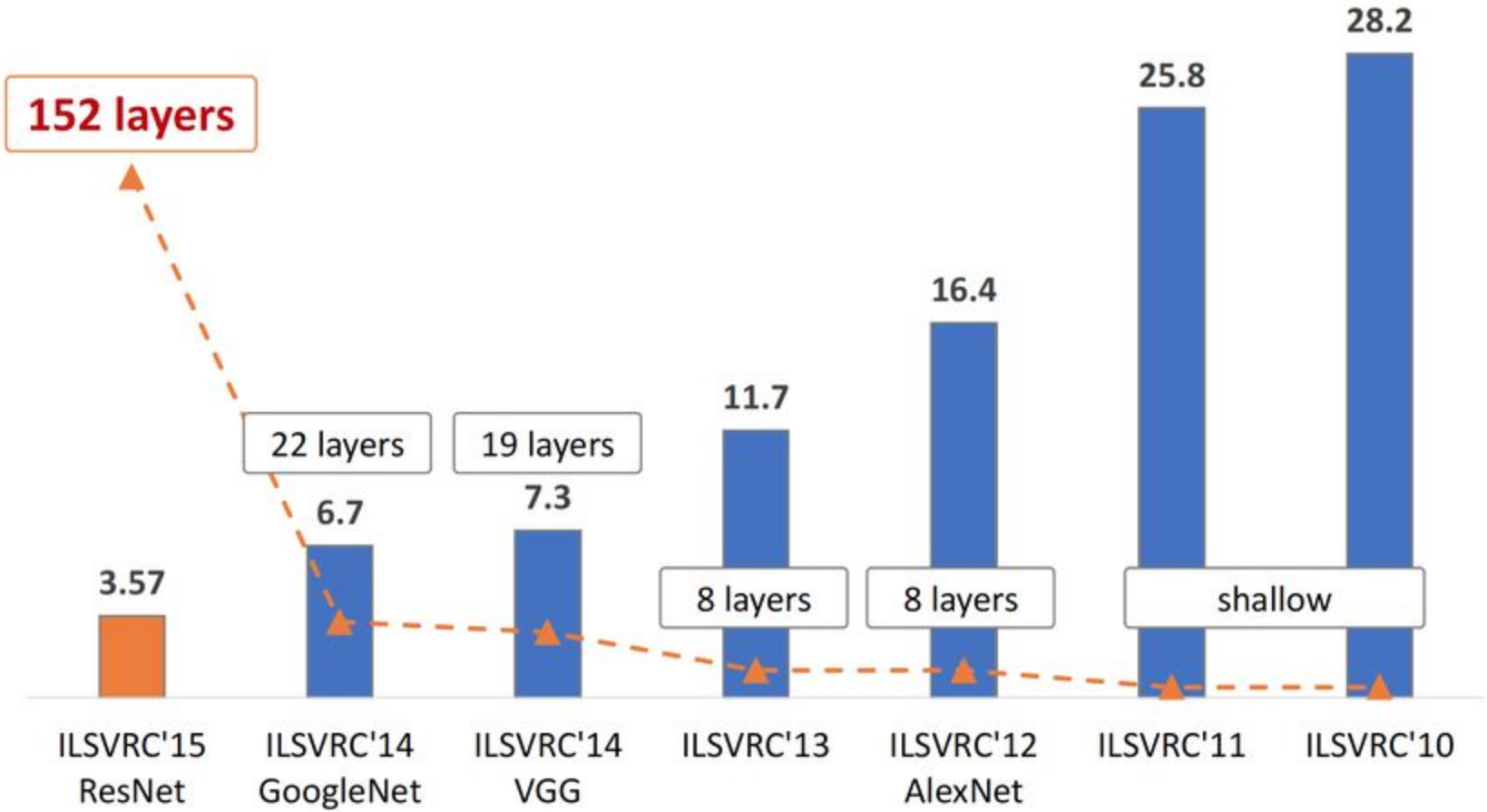
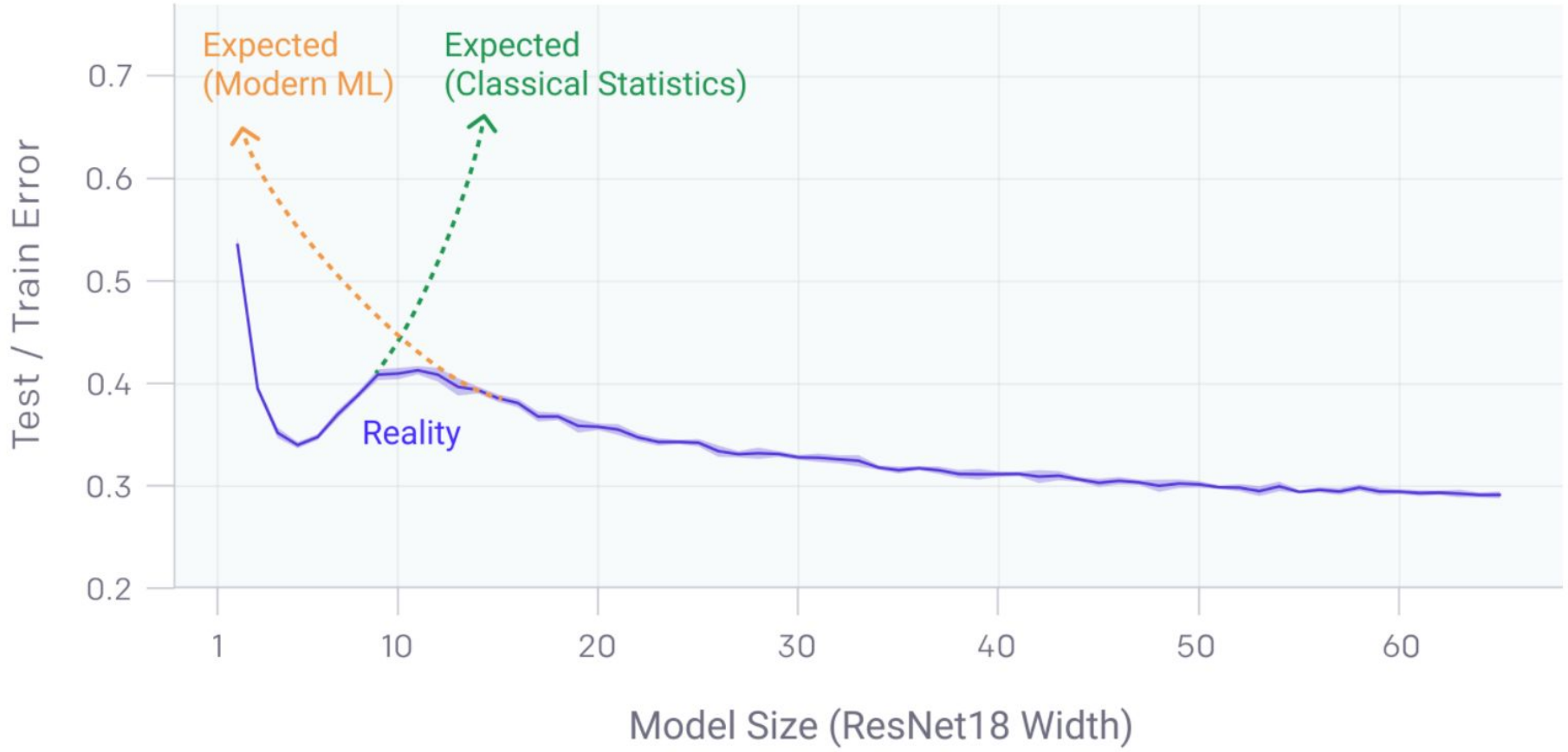
i-TAP Pruning

홍익대학교
노승문

2021. 4. 8.



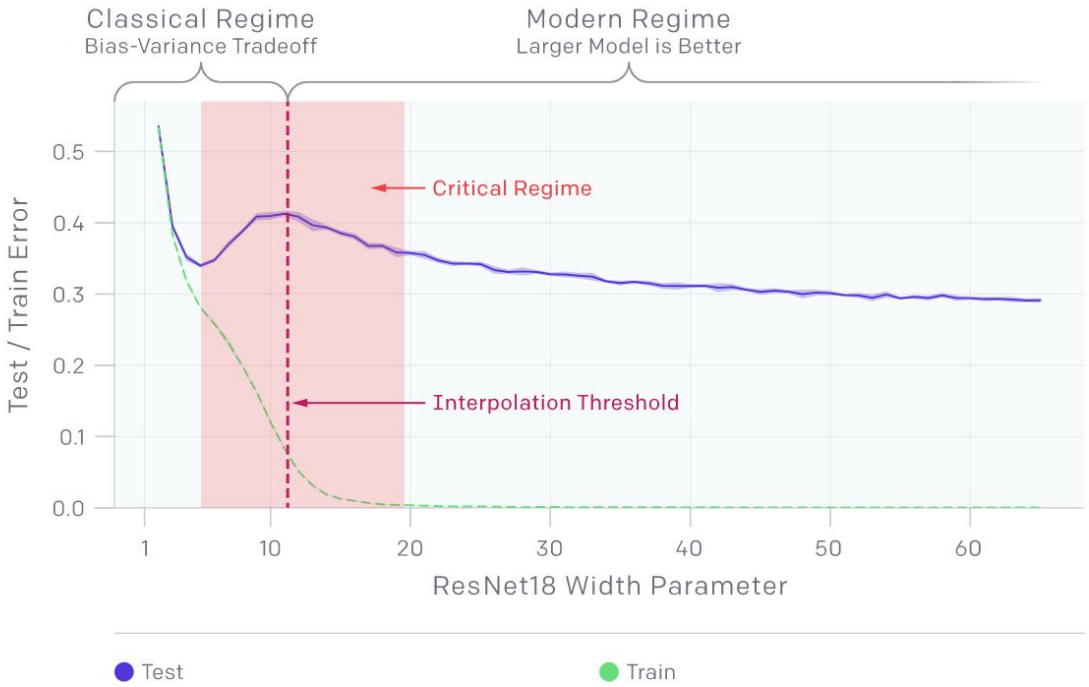
Nguyen, Kien & Fookes, Clinton & Ross, Arun & Sridharan, Sridha. (2017). Iris Recognition with Off-the-Shelf CNN Features: A Deep Learning Perspective. IEEE Access. PP. 1-1. 10.1109/ACCESS.2017.2784352.



https://openai.com/blog/deep-double-descent/

Model-wise double descent

1. There is a regime where bigger models are worse.



https://openai.com/blog/deep-double-descent/

Sample-wise non-monotonicity

2. There is a regime where more samples hurts.

https://openai.com/blog/deep-double-descent/

Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs

Timur Garipov[*1,2] **Pavel Izmailov**[*3] **Dmitrii Podoprikhin**[*4]
Dmitry Vetrov[5] **Andrew Gordon Wilson**[3]

[1]Samsung AI Center in Moscow, [2]Skolkovo Institute of Science and Technology,
[3]Cornell University,
[4]Samsung-HSE Laboratory, National Research University Higher School of Economics,
[5]National Research University Higher School of Economics

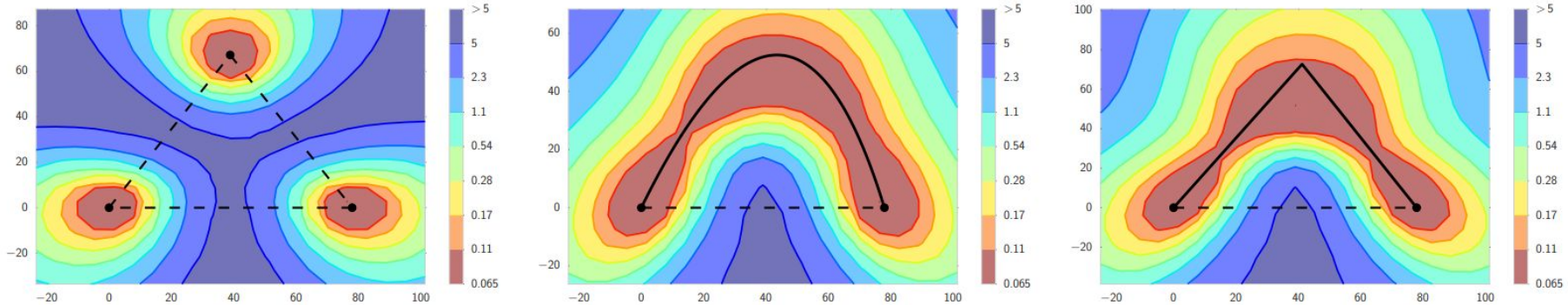


Figure 1: The ℓ_2-regularized cross-entropy train loss surface of a ResNet-164 on CIFAR-100, as a function of network weights in a two-dimensional subspace. In each panel, the horizontal axis is fixed and is attached to the optima of two independently trained networks. The vertical axis changes between panels as we change planes (defined in the main text). **Left:** Three optima for independently trained networks. **Middle** and **Right**: A quadratic Bezier curve, and a polygonal chain with one bend, connecting the lower two optima on the left panel along a path of near-constant loss. Notice that in each panel a direct linear path between each mode would incur high loss.

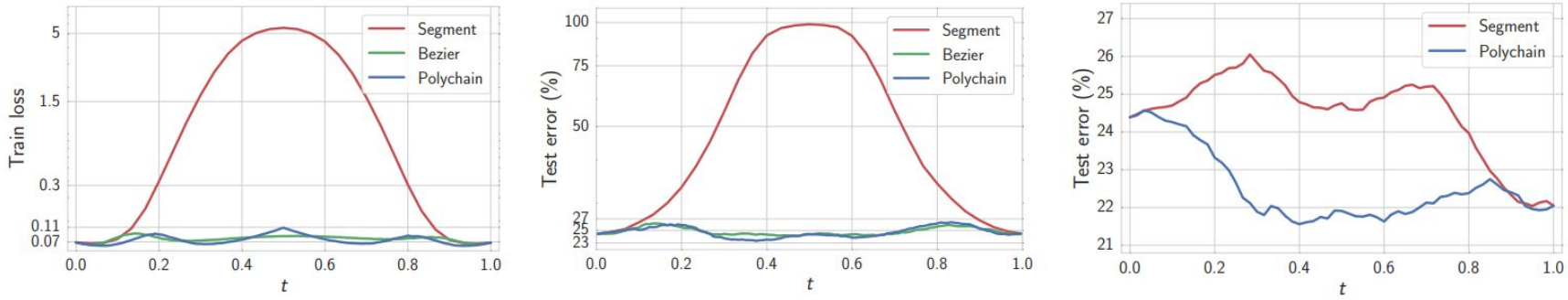
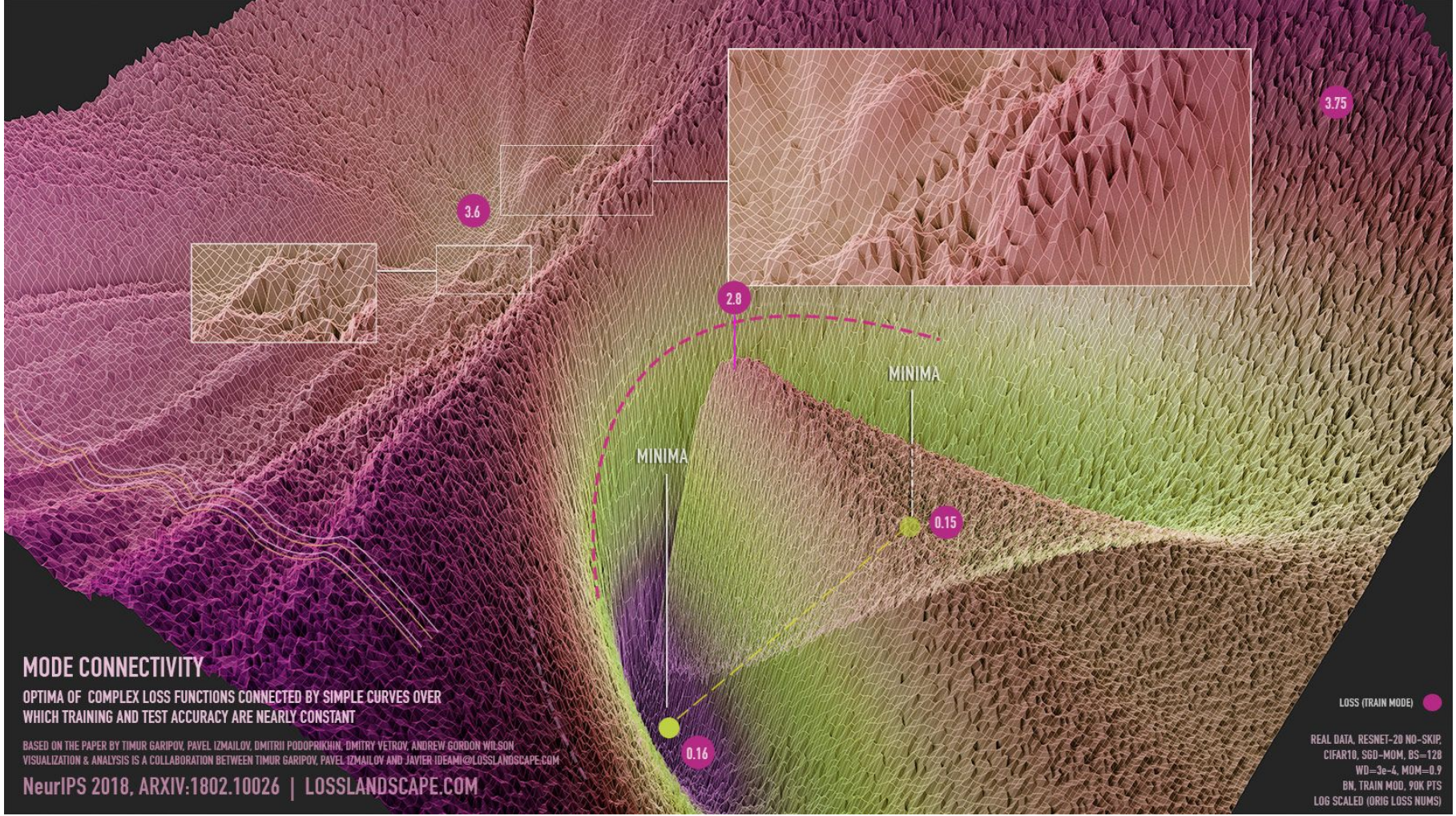


Figure 2: The ℓ_2-regularized cross-entropy train loss **(left)** and test error **(middle)** as a function of the point on the curves $\phi_\theta(t)$ found by the proposed method (ResNet-164 on CIFAR-100). **Right:** Error of the two-network ensemble consisting of the endpoint $\phi_\theta(0)$ of the curve and the point $\phi_\theta(t)$ on the curve (CIFAR-100, ResNet-164). "Segment" is a line segment connecting two modes found by SGD. "Polychain" is a polygonal chain connecting the same endpoints.



https://losslandscape.com/

Explaining Landscape Connectivity of Low-cost Solutions for Multilayer Nets

Rohith Kuditipudi
Duke University
rohith.kuditipudi@duke.edu

Xiang Wang
Duke University
xwang@cs.duke.edu

Holden Lee
Princeton University
holdenl@princeton.edu

Yi Zhang
Princeton University
y.zhang@cs.princeton.edu

Zhiyuan Li
Princeton University
zhiyuanli@cs.princeton.edu

Wei Hu
Princeton University
huwei@cs.princeton.edu

Sanjeev Arora
Princeton University and Institute for Advanced Study
arora@cs.princeton.edu

Rong Ge
Duke University
rongge@cs.duke.edu

NIPS 2019

Mode Connectivity

Mode connectivity and spurious valleys Fixing a neural network architecture, a data set $\mathcal{D}$ and a loss function, we say two sets of parameters/solutions θ^A and θ^B are **ϵ-connected** if there is a path $\pi(t) : \mathbb{R} \to \Theta$ that is continuous with respect to t and satisfies: 1. $\pi(0) = \theta^A$; 2. $\pi(1) = \theta^B$ and 3. for any $t \in [0, 1]$, $L(f_{\pi(t)}) \leq \max\{L(f_{\theta^A}), L(f_{\theta^B})\} + \epsilon$. If $\epsilon = 0$, we omit ϵ and just say they are connected.

- Parameters do not have to be minimizers

Dropout Stable solution

Definition 1. *(Dropout Stability) A solution θ is ϵ-**dropout stable** if for all i such that $1 \leq i < d$, there exists a subset of at most $\lfloor h_j/2 \rfloor$ hidden units in each of the layers j from i through $d-1$ such that after rescaling the outputs of these hidden units (or equivalently, the corresponding rows and/or columns of the relevant weight matrices) by some factor r[1] and setting the outputs of the remaining units to zero, we obtain a parameter θ_i such that $L(f_{\theta_i}) \leq L(f_\theta) + \epsilon$.*

- theta_i: pruned version of theta from i-th layer to (d-1)-th layer

Main Theorem

Theorem 1. *Let θ^A and θ^B be two ϵ-dropout stable solutions. Then there exists a path in parameter space $\pi : [0, 1] \to \Theta$ between θ^A and θ^B such that $L(f_{\pi(t)}) \leq \max\{L(f_{\theta^A}), L(f_{\theta^B})\} + \epsilon$ for $0 \leq t \leq 1$. In other words, letting $\mathcal{C}$ be the set of solutions that are ϵ-dropout stable, a ReLU network has the ϵ-mode connectivity property with respect to $\mathcal{C}$.*

Remark 1. *For convolutional networks, a* channel-wise *dropout will randomly set entire channels to 0 and rescale the remaining channels using an appropriate factor. Theorem 1 can be extended to work with channel-wise dropout on convolutional networks.*

Remove Redundancy?

Pruning!

Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

ABSTRACT

We have used information-theoretic ideas to derive a class of practical and nearly optimal schemes for adapting the size of a neural network. By removing unimportant weights from a network, several improvements can be expected: better generalization, fewer training examples required, and improved speed of learning and/or classification. The basic idea is to use second-derivative information to make a tradeoff between network complexity and training set error. Experiments confirm the usefulness of the methods on a real-world application.

NIPS 1989

ABSTRACT

We have used information-theoretic ideas to derive a class of practical and nearly optimal schemes for adapting the size of a neural network. By removing unimportant weights from a network, several improvements can be expected: better generalization, fewer training examples required, and improved speed of learning and/or classification. The basic idea is to use second-derivative information to make a tradeoff between network complexity and training set error. Experiments confirm the usefulness of the methods on a real-world application.

One of the main points of this paper is to move beyond the approximation that "magnitude equals saliency", and propose a theoretically justified saliency measure.

Our technique uses the second derivative of the objective function with respect to the parameters to compute the saliencies. The method was validated using our handwritten digit recognition network trained with backpropagation (Le Cun et al., 1990b).

DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING

Song Han
Stanford University, Stanford, CA 94305, USA
songhan@stanford.edu

Huizi Mao
Tsinghua University, Beijing, 100084, China
mhz12@mails.tsinghua.edu.cn

William J. Dally
Stanford University, Stanford, CA 94305, USA
NVIDIA, Santa Clara, CA 95050, USA
dally@stanford.edu

ABSTRACT

Neural networks are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems with limited hardware resources. To address this limitation, we introduce "deep compression", a three stage pipeline: pruning, trained quantization and Huffman coding, that work together to reduce the storage requirement of neural networks by $35\times$ to $49\times$ without affecting their accuracy. Our method first prunes the network by learning only the important connections. Next, we quantize the weights to enforce weight sharing, finally, we apply Huffman coding. After the first two steps we retrain the network to fine tune the remaining connections and the quantized centroids. Pruning, reduces the number of connections by $9\times$ to $13\times$; Quantization then reduces the number of bits that represent each connection from 32 to 5. On the ImageNet dataset, our method reduced the storage required by AlexNet by $\mathbf{35}\times$, from 240MB to 6.9MB, without loss of accuracy. Our method reduced the size of VGG-16 by $\mathbf{49}\times$ from 552MB to 11.3MB, again with no loss of accuracy. This allows fitting the model into on-chip SRAM cache rather than off-chip DRAM memory. Our compression method also facilitates the use of complex neural networks in mobile applications where application size and download bandwidth are constrained. Benchmarked on CPU, GPU and mobile GPU, compressed network has $3\times$ to $4\times$ layerwise speedup and $3\times$ to $7\times$ better energy efficiency.

3.1 WEIGHT SHARING

We use k-means clustering to identify the shared weights for each layer of a trained network, so that all the weights that fall into the same cluster will share the same weight. Weights are not shared across layers. We partition n original weights $W = \{w_1, w_2, ..., w_n\}$ into k clusters $C = \{c_1, c_2, ..., c_k\}$, $n \gg k$, so as to minimize the within-cluster sum of squares (WCSS):

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|^2 \tag{2}$$

Different from HashNet (Chen et al., 2015) where weight sharing is determined by a hash function before the networks sees any training data, our method determines weight sharing after a network is fully trained, so that the shared weights approximate the original network.

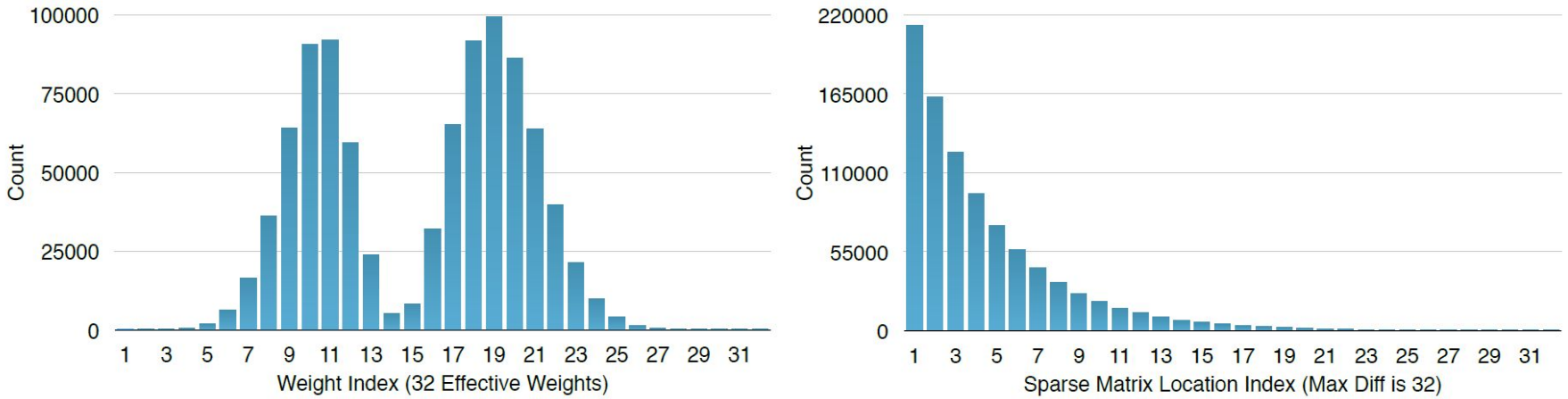


Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

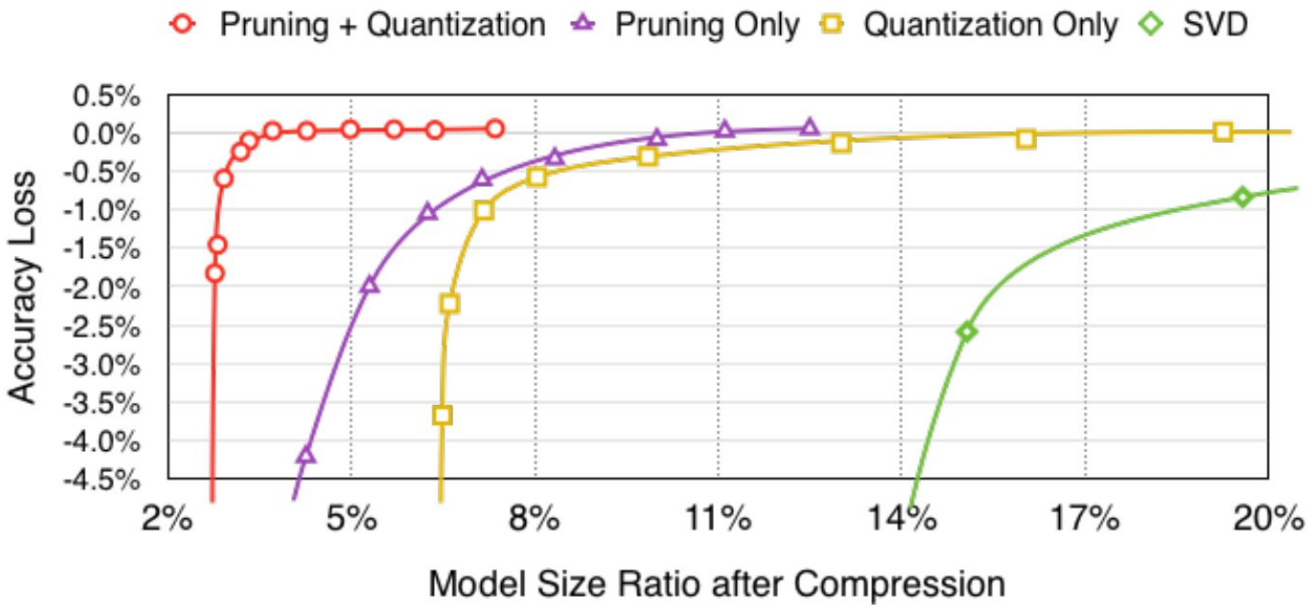


Figure 6: Accuracy v.s. compression rate under different compression methods. Pruning and quantization works best when combined.

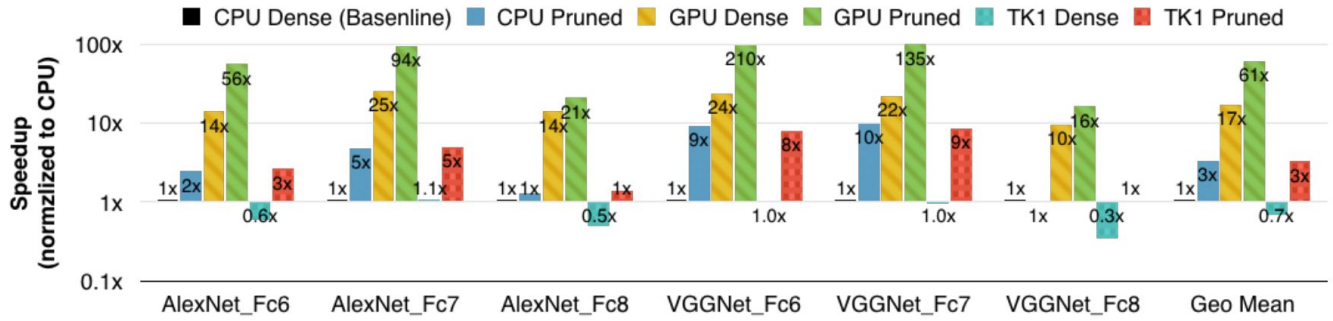


Figure 9: Compared with the original network, pruned network layer achieved $3\times$ speedup on CPU, $3.5\times$ on GPU and $4.2\times$ on mobile GPU on average. Batch size = 1 targeting real time processing. Performance number normalized to CPU.

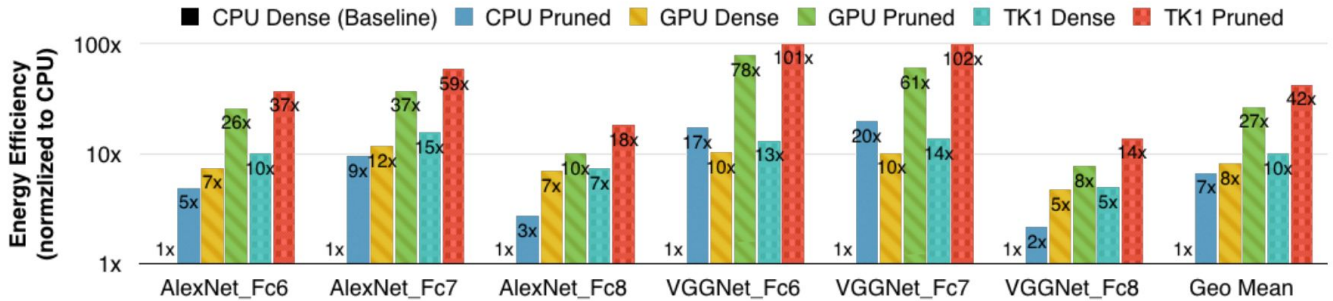


Figure 10: Compared with the original network, pruned network layer takes $7\times$ less energy on CPU, $3.3\times$ less on GPU and $4.2\times$ less on mobile GPU on average. Batch size = 1 targeting real time processing. Energy number normalized to CPU.

2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

EIE: Efficient Inference Engine on Compressed Deep Neural Network

Song Han* Xingyu Liu* Huizi Mao* Jing Pu* Ardavan Pedram*
Mark A. Horowitz* William J. Dally*†
*Stanford University, †NVIDIA
{songhan,xyl,huizi,jingpu,perdavan,horowitz,dally}@stanford.edu

Previously proposed 'Deep Compression' makes it possible to fit large DNNs (AlexNet and VGGNet) fully in on-chip SRAM. This compression is achieved by pruning the redundant connections and having multiple connections share the same weight. We propose an energy efficient inference engine (EIE) that performs inference on this compressed network model and accelerates the resulting sparse matrix-vector multiplication with weight sharing. Going from DRAM to SRAM gives EIE 120× energy saving; Exploiting sparsity saves 10×; Weight sharing gives 8×; Skipping zero activations from ReLU saves another 3×. Evaluated on nine DNN benchmarks, EIE is 189× and 13× faster when compared to CPU and GPU implementations of the same DNN without compression. EIE has a processing power of 102 GOPS working directly on a compressed network, corresponding to 3 TOPS on an uncompressed network, and processes FC layers of AlexNet at 1.88×10^4 frames/sec with a power dissipation of only 600mW. It is 24,000× and 3,400× more energy efficient than a CPU and GPU respectively. Compared with DaDianNao, EIE has 2.9×, 19× and 3× better throughput, energy efficiency and area efficiency.

Published as a conference paper at ICLR 2019

THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Jonathan Frankle
MIT CSAIL
jfrankle@csail.mit.edu

Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

ICLR2019 best paper

The Lottery Ticket Hypothesis. *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

Identifying winning tickets. We identify a winning ticket by training a network and pruning its smallest-magnitude weights. The remaining, unpruned connections constitute the architecture of the winning ticket. Unique to our work, each unpruned connection's value is then reset to its initialization from original network *before* it was trained. This forms our central experiment:

1. Randomly initialize a neural network $f(x; \theta_0)$ (where $\theta_0 \sim \mathcal{D}_\theta$).
2. Train the network for j iterations, arriving at parameters θ_j.
3. Prune $p\%$ of the parameters in θ_j, creating a mask m.
4. Reset the remaining parameters to their values in θ_0, creating the winning ticket $f(x; m \odot \theta_0)$.

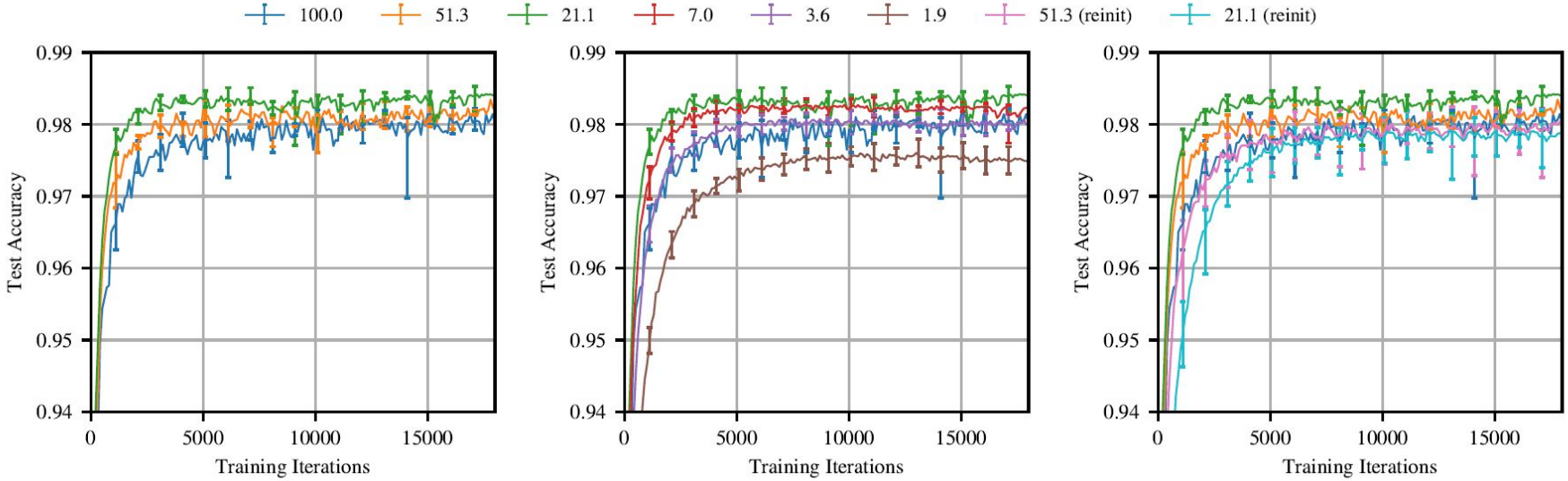
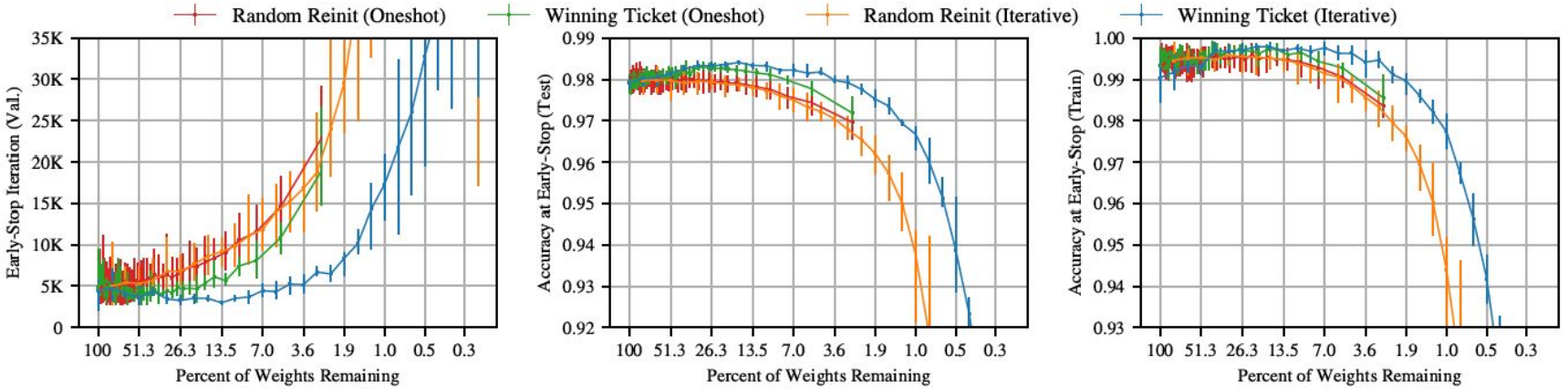


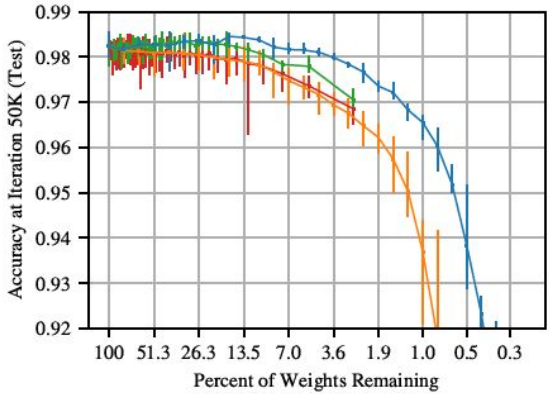
Figure 3: Test accuracy on Lenet (iterative pruning) as training proceeds. Each curve is the average of five trials. Labels are P_m—the fraction of weights remaining in the network after pruning. Error bars are the minimum and maximum of any trial.

Random reinitialization. To measure the importance of a winning ticket's initialization, we retain the structure of a winning ticket (i.e., the mask m) but randomly sample a new initialization $\theta'_0 \sim \mathcal{D}_\theta$. We randomly reinitialize each winning ticket three times, making 15 total per point in Figure 4. We find that initialization is crucial for the efficacy of a winning ticket. The right graph in Figure 3 shows this experiment for iterative pruning. In addition to the original network and winning tickets at $P_m = 51\%$ and 21% are the random reinitialization experiments. Where the winning tickets learn faster as they are pruned, they learn progressively slower when randomly reinitialized.

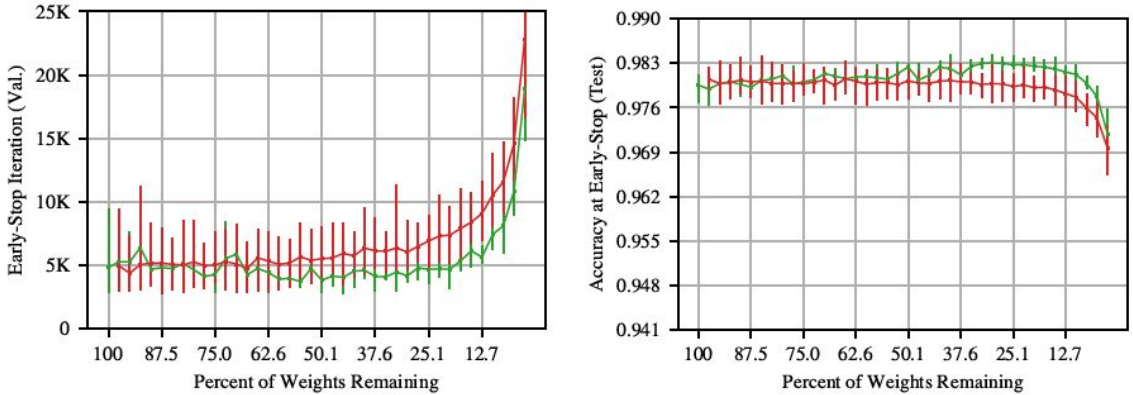
One-shot pruning. Although iterative pruning extracts smaller winning tickets, repeated training means they are costly to find. One-shot pruning makes it possible to identify winning tickets without this repeated training. Figure 4c shows the results of one-shot pruning (green) and randomly reinitializing (red); one-shot pruning does indeed find winning tickets. When $67.5\% \geq P_m \geq 17.6\%$, the average winning tickets reach minimum validation accuracy earlier than the original network. When $95.0\% \geq P_m \geq 5.17\%$, test accuracy is higher than the original network. However, iteratively-pruned winning tickets learn faster and reach higher test accuracy at smaller network sizes. The green and red lines in Figure 4c are reproduced on the logarithmic axes of Figure 4a, making this performance gap clear. Since our goal is to identify the smallest possible winning tickets, we focus on iterative pruning throughout the rest of the paper.



(a) Early-stopping iteration and accuracy for all pruning methods.



(b) Accuracy at end of training.



(c) Early-stopping iteration and accuracy for one-shot pruning.

Resnet-18. Resnet-18 (He et al., 2016) is a 20 layer convolutional network with residual connections designed for CIFAR10. It has 271,000 parameters. We train the network for 30,000 iterations with SGD with momentum (0.9), decreasing the learning rate by a factor of 10 at 20,000 and 25,000 iterations. Figure 8 shows the results of iterative pruning and random reinitialization at learning rates 0.1 (used in He et al. (2016)) and 0.01. These results largely mirror those of VGG: iterative pruning finds winning tickets at the lower learning rate but not the higher learning rate. The accuracy of the best winning tickets at the lower learning rate (89.5% when $41.7\% \geq P_m \geq 21.9\%$) falls short of the original network's accuracy at the higher learning rate (90.5%). At lower learning rate, the winning ticket again initially learns faster (left plots of Figure 8), but falls behind the unpruned network at the higher learning rate later in training (right plot). Winning tickets trained with warmup close the accuracy gap with the unpruned network at the higher learning rate, reaching 90.5% test accuracy with learning rate 0.03 (warmup, $k = 20000$) at $P_m = 27.1\%$. For these hyperparameters, we still find winning tickets when $P_m \geq 11.8\%$. Even with warmup, however, we could not find hyperparameters for which we could identify winning tickets at the original learning rate, 0.1.

Linear Mode Connectivity and the Lottery Ticket Hypothesis

Jonathan Frankle [1] **Gintare Karolina Dziugaite** [2] **Daniel M. Roy** [3 4] **Michael Carbin** [1]

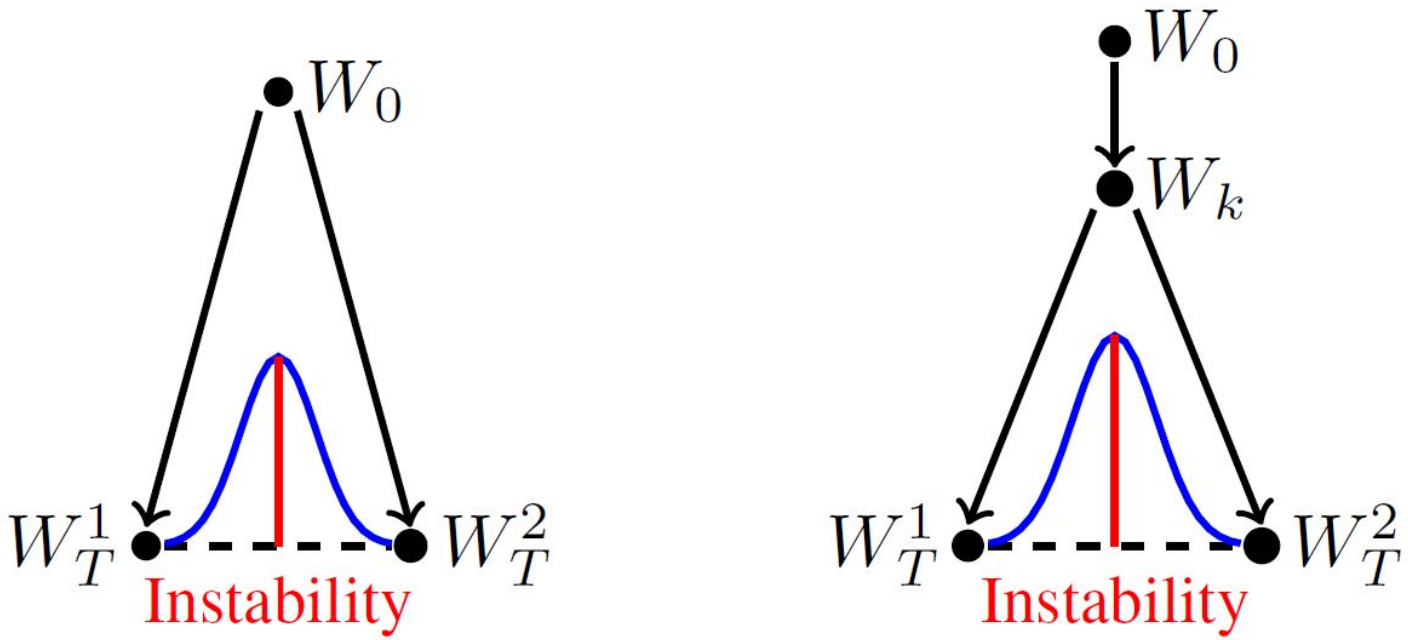


Figure 1. A diagram of instability analysis from step 0 (left) and step k (right) when comparing networks using linear interpolation.

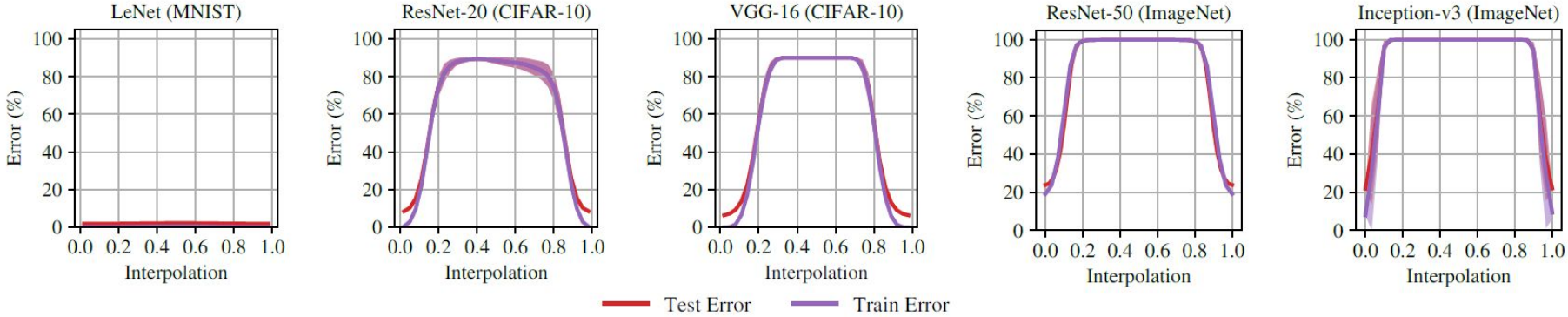


Figure 2. Error when linearly interpolating between networks trained from the same initialization with different SGD noise. Lines are means and standard deviations over three initializations and three data orders (nine samples total). Trained networks are at 0.0 and 1.0.

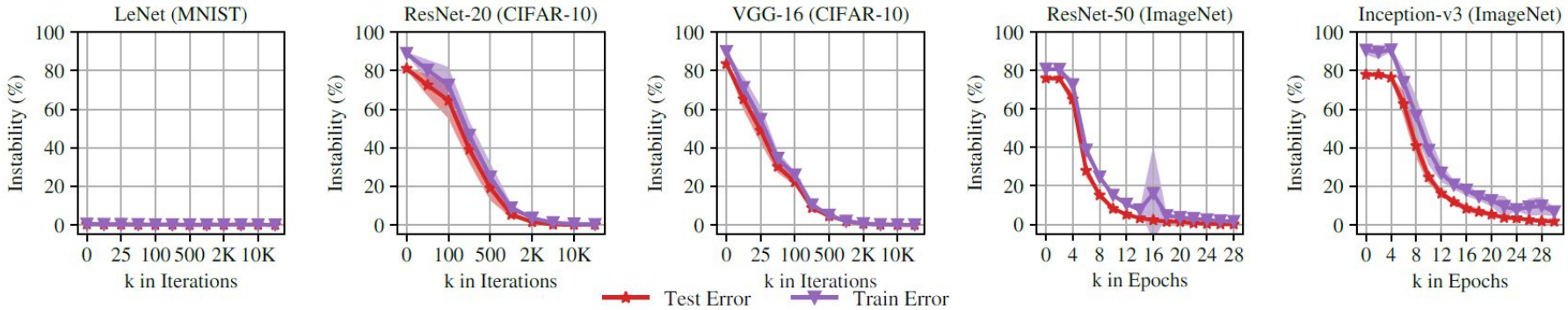


Figure 3. Linear interpolation instability when starting from step k. Each line is the mean and standard deviation across three initializations and three data orders (nine samples in total).

Algorithm 2 IMP rewinding to step k and N iterations.

Create a network with randomly initialization $W_0 \in \mathbb{R}^d$.
Initialize pruning mask to $m = 1^d$.
Train W_0 to W_k with noise $u \sim U$: $W_k = \mathcal{A}^{0 \to k}(W_0, u)$.
for $n \in \{1, \ldots, N\}$ **do**
    Train $m \odot W_k$ to $m \odot W_T$ with noise $u' \sim U$:
    $W_T = \mathcal{A}^{k \to T}(m \odot W_k, u')$.
    Prune the lowest magnitude entries of W_T that remain.
    Let $m[i] = 0$ if $W_T[i]$ is pruned.
Return W_k, m

Network	Full	IMP	Rand Prune	Rand Reinit	Δ IMP	Matching?
LeNet	98.3	98.2	96.7	97.5	0.1	Y
ResNet-20	91.7	88.5	88.6	88.8	3.2	N
ResNet-20 Low	88.8	89.0	85.7	84.7	-0.2	Y
ResNet-20 Warmup	89.7	89.6	85.7	85.6	0.1	Y
VGG-16	93.7	90.9	89.4	91.0	2.8	N
VGG-16 Low	91.7	91.6	90.1	90.2	0.1	Y
VGG-16 Warmup	93.4	93.2	90.1	90.7	0.2	Y
ResNet-50	76.1	73.7	73.1	73.4	2.4	N
Inception-v3	78.1	75.7	75.2	75.5	2.4	N

Table 2. Accuracy of IMP and random subnetworks when rewinding to $k = 0$ at the sparsities in Table 1. Accuracies are means across three initializations. All standard deviations are < 0.2.

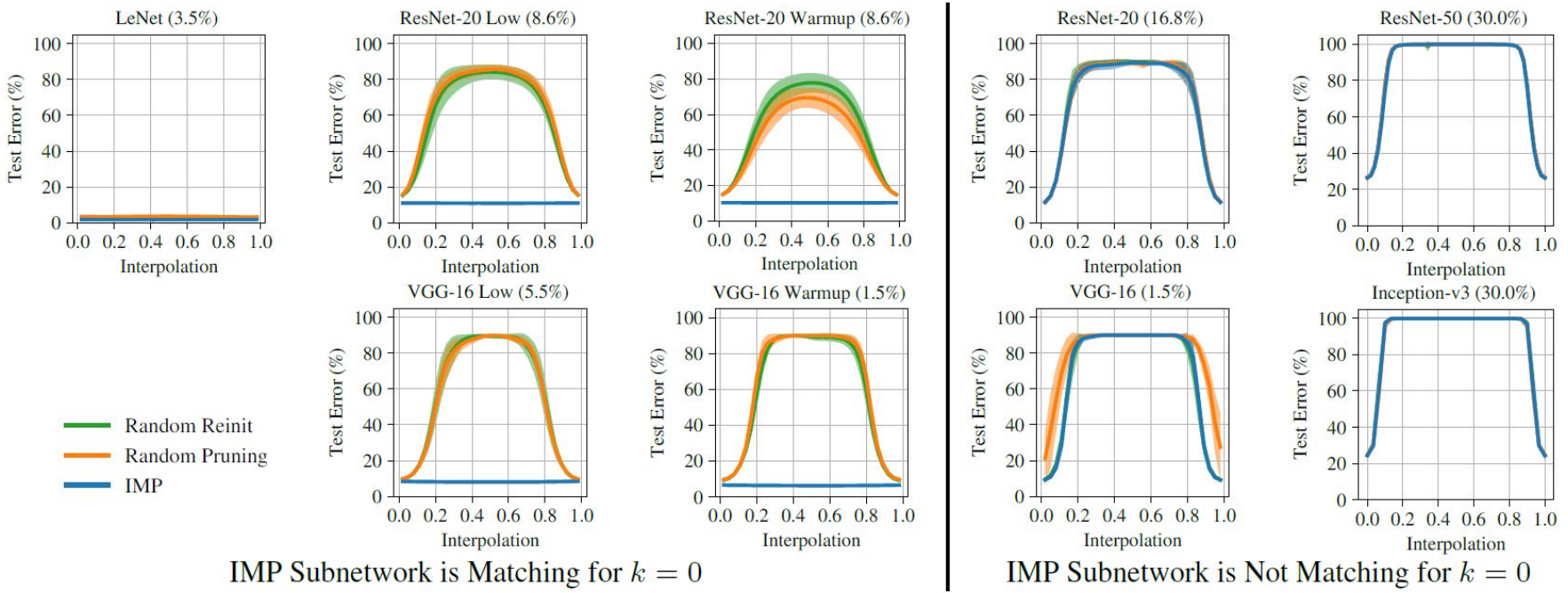


Figure 5. Test error when linearly interpolating between subnetworks trained from the same initialization with different SGD noise. Lines are means and standard deviations over three initializations and three data orders (nine in total). Percents are weights remaining.

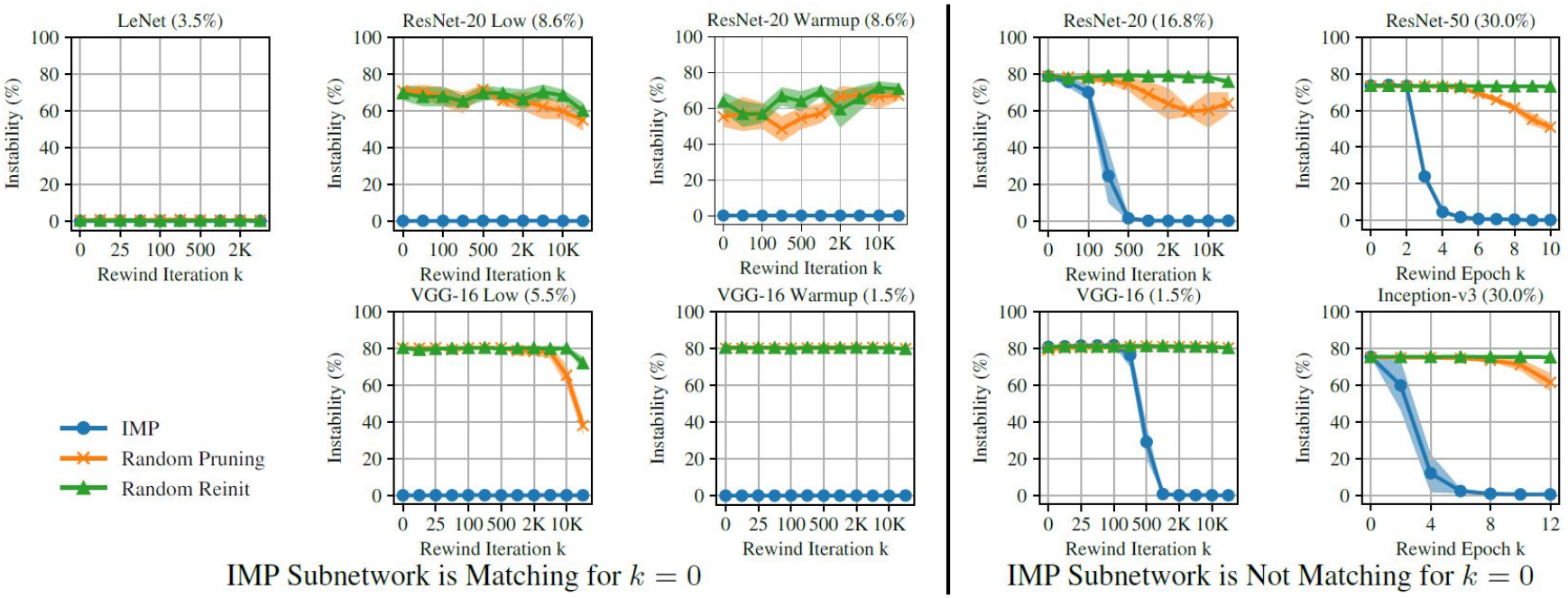


Figure 6. Linear interpolation instability of subnetworks created using the state of the full network at step k and applying a pruning mask. Lines are means and standard deviations over three initializations and three data orders (nine in total). Percents are weights remaining.

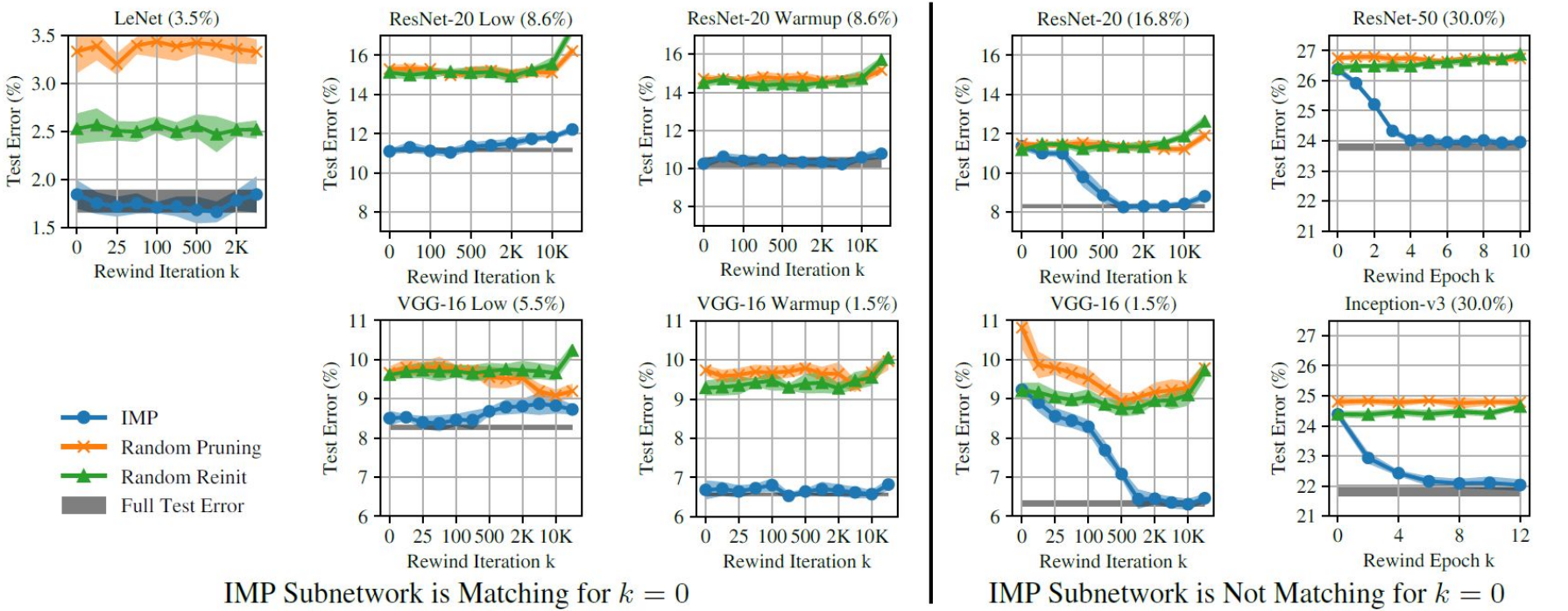


Figure 7. Test error of subnetworks created using the state of the full network at step k and applying a pruning mask. Lines are means and standard deviations over three initializations and three data orders (nine in total). Percents are weights remaining.

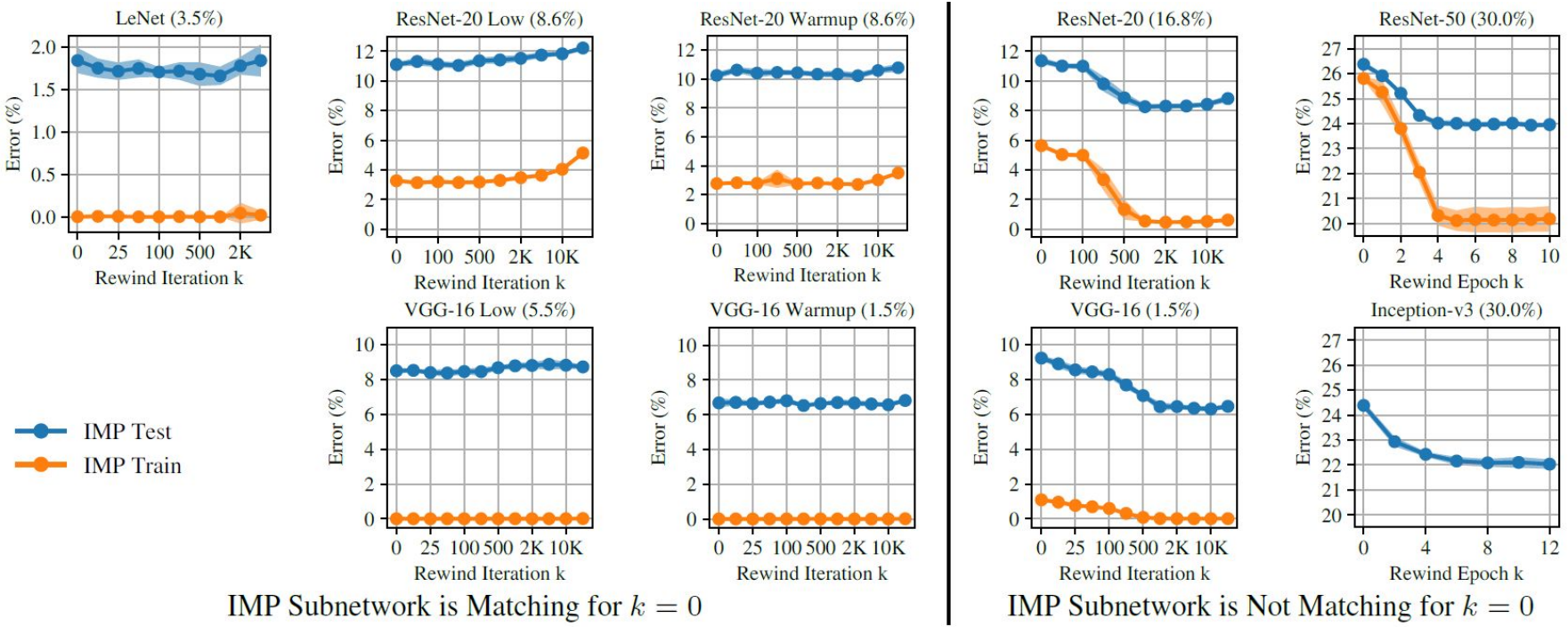
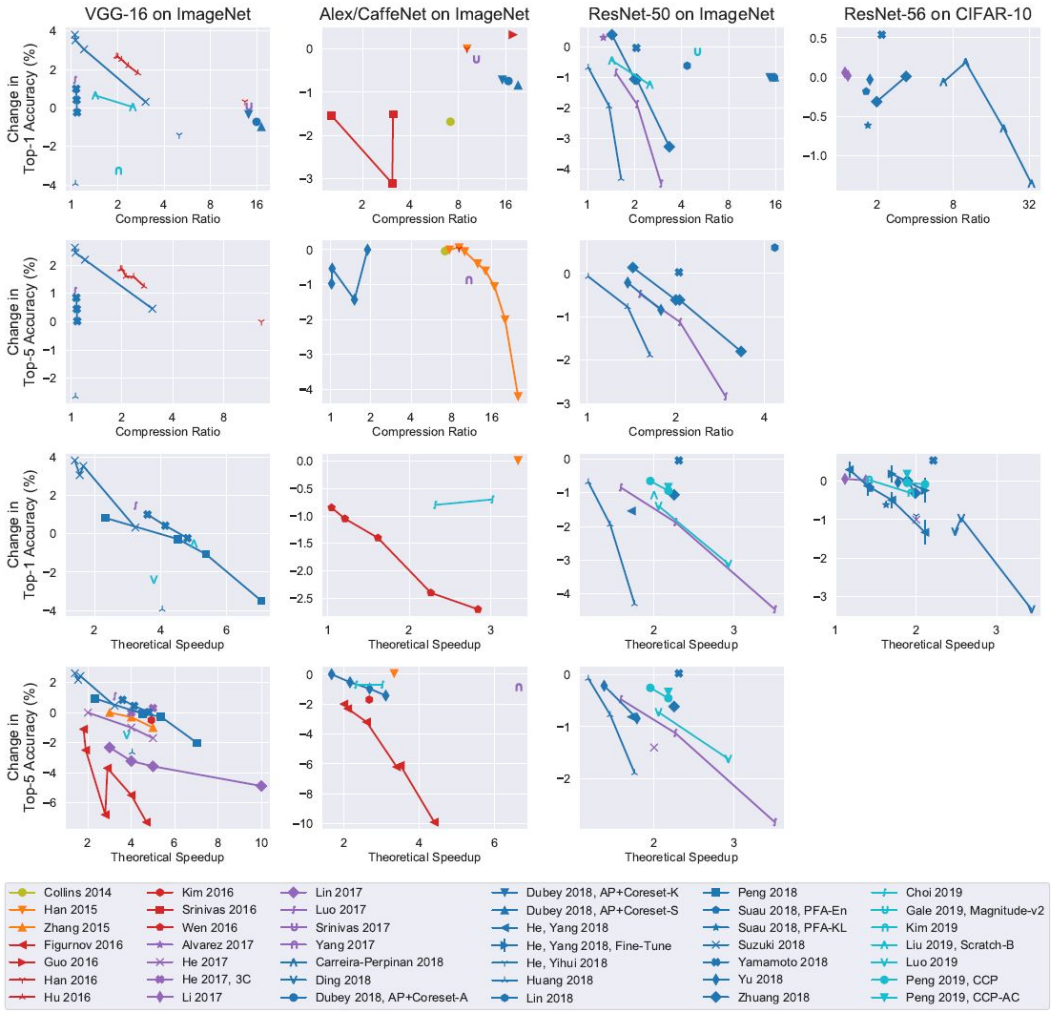


Figure 8. Train error of subnetworks created using the state of the full network at step k and apply a pruning mask. Lines are means and standard deviations over three initializations and three data orders (nine in total). Percents are weights remaining. We did not compute the train set quantities for Inception-v3 due to computational limitations.

WHAT IS THE STATE OF NEURAL NETWORK PRUNING?

Davis Blalock [*1] **Jose Javier Gonzalez Ortiz** [*1] **Jonathan Frankle** [1] **John Guttag** [1]


VGG-16 on ImageNet
Alex/CaffeNet on ImageNet
ResNet-50 on ImageNet
ResNet-56 on CIFAR-10
Change in Top-1 Accuracy (%)
Change in Top-5 Accuracy (%)
Compression Ratio
Theoretical Speedup
Collins 2014
Han 2015
Zhang 2015
Figurnov 2016
Guo 2016
Han 2016
Hu 2016
Kim 2016
Srinivas 2016
Wen 2016
Alvarez 2017
He 2017
He 2017, 3C
Li 2017
Lin 2017
Luo 2017
Srinivas 2017
Yang 2017
Carreira-Perpinan 2018
Ding 2018
Dubey 2018, AP+Coreset-A
Dubey 2018, AP+Coreset-K
Dubey 2018, AP+Coreset-S
He, Yang 2018
He, Yang 2018, Fine-Tune
He, Yihui 2018
Huang 2018
Lin 2018
Peng 2018
Suau 2018, PFA-En
Suau 2018, PFA-KL
Suzuki 2018
Yamamoto 2018
Yu 2018
Zhuang 2018
Choi 2019
Gale 2019, Magnitude-v2
Kim 2019
Liu 2019, Scratch-B
Luo 2019
Peng 2019, CCP
Peng 2019, CCP-AC

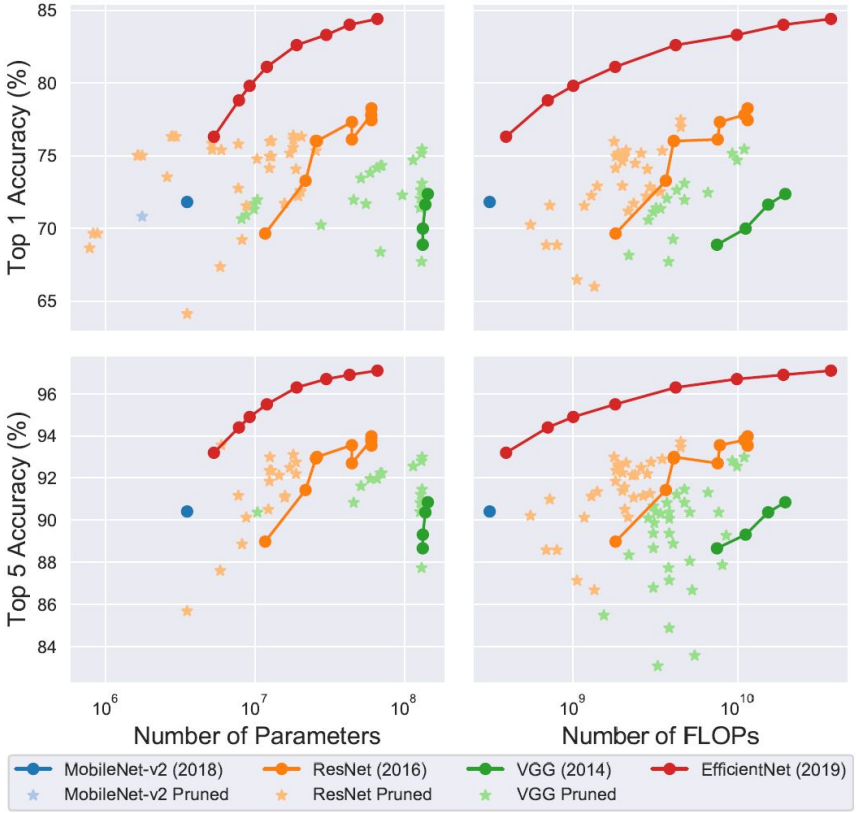


Figure 1: Size and speed vs accuracy tradeoffs for different pruning methods and families of architectures. Pruned models sometimes outperform the original architecture, but rarely outperform a better architecture.

- **Global Magnitude Pruning** - prunes the weights with the lowest absolute value anywhere in the network.
- **Layerwise Magnitude Pruning** - for each layer, prunes the weights with the lowest absolute value.
- **Global Gradient Magnitude Pruning** - prunes the weights with the lowest absolute value of (weight $\times$ gradient), evaluated on a batch of inputs.
- **Layerwise Gradient Magnitude Pruning** - for each layer, prunes the weights the lowest absolute value of (weight $\times$ gradient), evaluated on a batch of inputs.
- **Random Pruning** - prunes each weight independently with probability equal to the fraction of the network to be pruned.

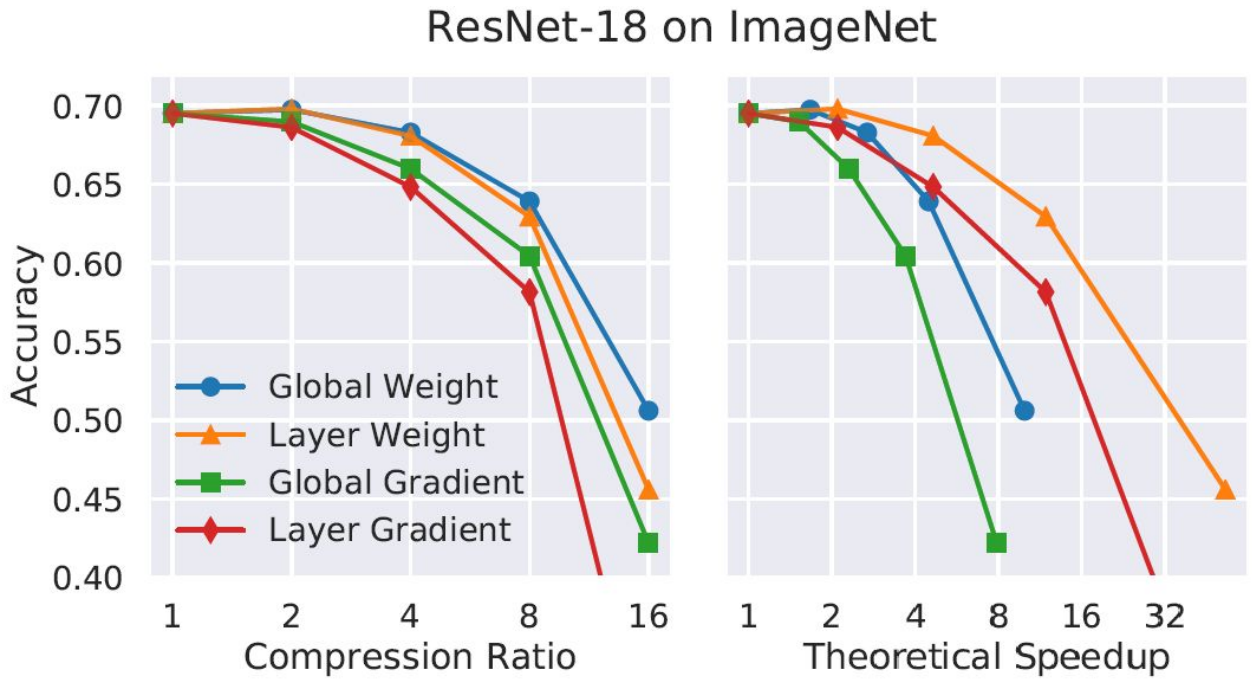


Figure 6: Top 1 Accuracy for ResNet-18 on ImageNet for several compression ratios and their corresponding theoretical speedups. Global methods give higher accuracy than Layerwise ones for a fixed model size, but the reverse is true for a fixed theoretical speedup.

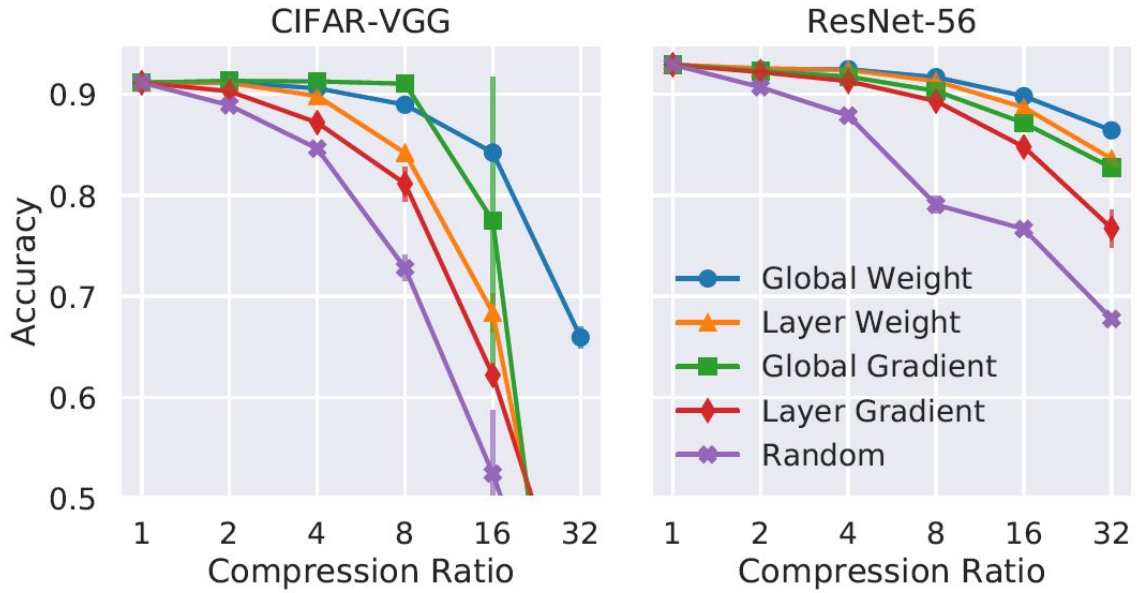


Figure 7: Top 1 Accuracy on CIFAR-10 for several compression ratios. Global Gradient performs better than Global Magnitude for CIFAR-VGG on low compression ratios, but worse otherwise. Global Gradient is consistently better than Layerwise Magnitude on CIFAR-VGG, but consistently worse on ResNet-56.

Questions?

Published as a conference paper at ICLR 2019

SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY

Namhoon Lee, Thalaiyasingam Ajanthan & Philip H. S. Torr
University of Oxford
{namhoon,ajanthan,phst}@robots.ox.ac.uk

ABSTRACT

Pruning large neural networks while maintaining their performance is often desirable due to the reduced space and time complexity. In existing methods, pruning is done within an iterative optimization procedure with either heuristically designed pruning schedules or additional hyperparameters, undermining their utility. In this work, we present a new approach that prunes a given network once at initialization prior to training. To achieve this, we introduce a saliency criterion based on connection sensitivity that identifies structurally important connections in the network for the given task. This eliminates the need for both pretraining and the complex pruning schedule while making it robust to architecture variations. After pruning, the sparse network is trained in the standard way. Our method obtains extremely sparse networks with virtually the same accuracy as the reference network on the MNIST, CIFAR-10, and Tiny-ImageNet classification tasks and is broadly applicable to various architectures including convolutional, residual and recurrent networks. Unlike existing methods, our approach enables us to demonstrate that the retained connections are indeed relevant to the given task.

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, and a desired sparsity level κ (*i.e.*, the number of non-zero weights) neural network pruning can be written as the following constrained optimization problem:

$$\min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \ , \tag{1}$$

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \|\mathbf{w}\|_0 \leq \kappa \ .$$

Here, $\ell(\cdot)$ is the standard loss function (*e.g.*, cross-entropy loss), $\mathbf{w}$ is the set of parameters of the neural network, m is the total number of parameters and $\|\cdot\|_0$ is the standard L_0 norm.

4.1 CONNECTION SENSITIVITY: ARCHITECTURAL PERSPECTIVE

parameters $\mathbf{w}$.[1] Now, given the sparsity level κ, Equation 1 can be correspondingly modified as:

$$\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{c} \odot \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \,, \tag{3}$$

$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m \,,$$

$$\mathbf{c} \in \{0,1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa \,,$$

For instance, the value of c_j indicates whether the connection j is active ($c_j = 1$) in the network or pruned ($c_j = 0$). Therefore, to measure the effect of connection j on the loss, one can try to measure the difference in loss when $c_j = 1$ and $c_j = 0$, keeping everything else constant. Precisely, the effect of removing connection j can be measured by,

$$\Delta L_j(\mathbf{w}; \mathcal{D}) = L(\mathbf{1} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{1} - \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D}) \,, \tag{4}$$

where $\mathbf{e}_j$ is the indicator vector of element j (*i.e.*, zeros everywhere except at the index j where it is one) and $\mathbf{1}$ is the vector of dimension m.

Note that computing ΔL_j for each $j \in \{1 \dots m\}$ is prohibitively expensive as it requires $m + 1$ (usually in the order of millions) forward passes over the dataset. In fact, since $\mathbf{c}$ is binary, L is not differentiable with respect to $\mathbf{c}$, and it is easy to see that ΔL_j attempts to measure the influence of connection j on the loss function in this discrete setting. Therefore, by relaxing the binary constraint on the indicator variables $\mathbf{c}$, ΔL_j can be approximated by the derivative of L with respect to c_j, which we denote $g_j(\mathbf{w}; \mathcal{D})$. Hence, the effect of connection j on the loss can be written as:

$$\Delta L_j(\mathbf{w}; \mathcal{D}) \approx g_j(\mathbf{w}; \mathcal{D}) = \left. \frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial c_j} \right|_{\mathbf{c}=\mathbf{1}} = \lim_{\delta \to 0} \left. \frac{L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{c} - \delta\, \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D})}{\delta} \right|_{\mathbf{c}=\mathbf{1}} \tag{5}$$

Notably, our interest is to discover important (or sensitive) connections in the architecture, so that we can prune unimportant ones in single-shot, disentangling the pruning process from the iterative optimization cycles. To this end, we take the magnitude of the derivatives g_j as the saliency criterion. Note that if the magnitude of the derivative is high (regardless of the sign), it essentially means that the connection c_j has a considerable effect on the loss (either positive or negative), and it has to be preserved to allow learning on w_j. Based on this hypothesis, we define connection sensitivity as the normalized magnitude of the derivatives:

$$s_j = \frac{|g_j(\mathbf{w}; \mathcal{D})|}{\sum_{k=1}^{m} |g_k(\mathbf{w}; \mathcal{D})|} \,. \tag{6}$$

Once the sensitivity is computed, only the top-κ connections are retained, where κ denotes the desired number of non-zero weights. Precisely, the indicator variables $\mathbf{c}$ are set as follows:

$$c_j = \mathbb{1}[s_j - \tilde{s}_\kappa \geq 0] \,, \quad \forall\, j \in \{1 \ldots m\} \,, \tag{7}$$

where $\tilde{s}_\kappa$ is the κ-th largest element in the vector $\mathbf{s}$ and $\mathbb{1}[\cdot]$ is the indicator function. Here, for exactly κ connections to be retained, ties can be broken arbitrarily.

Algorithm 1 SNIP: Single-shot Network Pruning based on Connection Sensitivity

Require: Loss function L, training dataset $\mathcal{D}$, sparsity level κ ▷ Refer Equation 3

Ensure: $\|\mathbf{w}^*\|_0 \leq \kappa$

1: $\mathbf{w} \leftarrow$ VarianceScalingInitialization ▷ Refer Section 4.2

2: $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^b \sim \mathcal{D}$ ▷ Sample a mini-batch of training data

3: $s_j \leftarrow \frac{\left|g_j(\mathbf{w}; \mathcal{D}^b)\right|}{\sum_{k=1}^m |g_k(\mathbf{w}; \mathcal{D}^b)|}, \quad \forall j \in \{1 \dots m\}$ ▷ Connection sensitivity

4: $\tilde{\mathbf{s}} \leftarrow \text{SortDescending}(\mathbf{s})$

5: $c_j \leftarrow \mathbb{1}[s_j - \tilde{s}_\kappa \geq 0], \quad \forall j \in \{1 \dots m\}$ ▷ Pruning: choose top-κ connections

6: $\mathbf{w}^* \leftarrow \arg\min_{\mathbf{w} \in \mathbb{R}^m} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})$ ▷ Regular training

7: $\mathbf{w}^* \leftarrow \mathbf{c} \odot \mathbf{w}^*$

Architecture	Model	Sparsity (%)	# Parameters	Error (%)	Δ
Convolutional	AlexNet-s	90.0	5.1m → 507k	14.12 → 14.99	+0.87
	AlexNet-b	90.0	8.5m → 849k	13.92 → 14.50	+0.58
	VGG-C	95.0	10.5m → 526k	6.82 → 7.27	+0.45
	VGG-D	95.0	15.2m → 762k	6.76 → 7.09	+0.33
	VGG-like	97.0	15.0m → 449k	8.26 → 8.00	**−0.26**
Residual	WRN-16-8	95.0	10.0m → 548k	6.21 → 6.63	+0.42
	WRN-16-10	95.0	17.1m → 856k	5.91 → 6.43	+0.52
	WRN-22-8	95.0	17.2m → 858k	6.14 → 5.85	**−0.29**
Recurrent	LSTM-s	95.0	137k → 6.8k	1.88 → 1.57	**−0.31**
	LSTM-b	95.0	535k → 26.8k	1.15 → 1.35	+0.20
	GRU-s	95.0	104k → 5.2k	1.87 → 2.41	+0.54
	GRU-b	95.0	404k → 20.2k	1.71 → 1.52	**−0.19**

Table 2: Pruning results of the proposed approach on various modern architectures (before → after). AlexNets, VGGs and WRNs are evaluated on CIFAR-10, and LSTMs and GRUs are evaluated on the sequential MNIST classification task. The approach is generally applicable regardless of architecture types and models and results in a significant amount of reduction in the number of parameters with minimal or no loss in performance.

Published as a conference paper at ICLR 2020

PICKING WINNING TICKETS BEFORE TRAINING BY PRESERVING GRADIENT FLOW

Chaoqi Wang, Guodong Zhang, Roger Grosse
University of Toronto, Vector Institute
{cqwang, gdzhang, rgrosse}@cs.toronto.edu

Revisiting SNIP. SNIP (Lee et al., 2018) was the first algorithm proposed for foresight pruning, and it leverages the notion of *connection sensitivity* to remove unimportant connections. They define this in terms of how removing a single weight θ_q in isolation will affect the loss:

$$S(\theta_q) = \lim_{\epsilon \to 0} \left| \frac{\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_0 + \epsilon \boldsymbol{\delta}_q)}{\epsilon} \right| = \left| \theta_q \frac{\partial \mathcal{L}}{\partial \theta_q} \right| \tag{5}$$

where θ_q is the q_{th} element of $\boldsymbol{\theta}_0$, and $\boldsymbol{\delta}_q$ is a one-hot vector whose q_{th} element equals θ_q. Essentially, SNIP aims to preserve the loss of the original randomly initialized network.

> Preserving the loss value motivated several classic methods for pruning a *trained network*, such as optimal brain damage (LeCun et al., 1990) and optimal brain surgery (Hassibi et al., 1993). While the motivation for loss preservation of a trained network is clear, it is less clear why this is a good criterion for foresight pruning. After all, at initialization, the loss is no better than chance. We argue that at the *beginning* of training, it is more important to preserve the training dynamics than the loss itself. SNIP does not do this automatically, because even if removing a particular connection doesn't affect the loss, it could still block the flow of information through the network. For instance, we noticed in our experiments that SNIP with a high pruning ratio (e.g. 99%) tends to eliminate nearly all the weights in a particular layer, creating a bottleneck in the network. Therefore, we would prefer a pruning criterion which accounts for how the presence or absence of one connection influences the training of the rest of the network.

Mathematically, a larger gradient norm indicates that, to the first order, each gradient update achieves a greater loss reduction, as characterized by the following directional derivative:

$$\Delta\mathcal{L}(\boldsymbol{\theta}) = \lim_{\epsilon \to 0} \frac{\mathcal{L}\left(\boldsymbol{\theta} + \epsilon \nabla \mathcal{L}(\boldsymbol{\theta})\right) - \mathcal{L}(\boldsymbol{\theta})}{\epsilon} = \nabla \mathcal{L}(\boldsymbol{\theta})^\top \nabla \mathcal{L}(\boldsymbol{\theta}) \tag{6}$$

We would like to preserve or even increase (if possible) the gradient flow *after pruning* (*i.e.*, the gradient flow of the pruned network). Following LeCun et al. (1990), we cast the pruning operation as adding a perturbation $\boldsymbol{\delta}$ to the initial weights. We then use a Taylor approximation to characterize how removing one weight will affect the gradient flow *after pruning*:

$$\begin{aligned} \mathbf{S}\left(\boldsymbol{\delta}\right) &= \Delta\mathcal{L}(\boldsymbol{\theta}_0 + \boldsymbol{\delta}) - \underbrace{\Delta\mathcal{L}(\boldsymbol{\theta}_0)}_{\text{Const}} = 2\boldsymbol{\delta}^\top \nabla^2 \mathcal{L}(\boldsymbol{\theta}_0) \nabla \mathcal{L}(\boldsymbol{\theta}_0) + \mathcal{O}(\|\boldsymbol{\delta}\|_2^2) \\ &= 2\boldsymbol{\delta}^\top \mathbf{H}\mathbf{g} + \mathcal{O}(\|\boldsymbol{\delta}\|_2^2), \end{aligned} \tag{7}$$

where $\mathbf{S}(\boldsymbol{\delta})$ approximately measures the change to (6). The Hessian matrix $\mathbf{H}$ captures the dependencies between different weights, and thus helps predict the effect of removing multiple weights. When $\mathbf{H}$ is approximated as the identity matrix, the above criterion recovers SNIP up to the absolute value (recall the SNIP criterion is $|\boldsymbol{\delta}^\top \mathbf{g}|$). However, it has been observed that different weights are highly coupled (Hassibi et al., 1993), indicating that $\mathbf{H}$ is in fact far from the identity.

GraSP uses eqn. (7) as the measure of the importance of each weight. Specifically, if $S(\boldsymbol{\delta})$ is negative, then removing the corresponding weights will reduce the gradient flow, while if it is positive, it will increase the gradient flow. We prefer to first remove those weights whose removal will not reduce the gradient flow. For each weight, the importance can be computed in the following way (by an abuse of notation, we use bold $\mathbf{S}$ to denote vectorized importance):

$$\mathbf{S}(-\boldsymbol{\theta}) = -\boldsymbol{\theta} \odot \mathbf{Hg} \tag{8}$$

For a given pruning ratio p, we obtain the resulting pruning mask by computing the importance score of every weight, and removing the bottom p fraction of the weights (see Algorithm 1). Hence, GraSP takes the gradient flow into account for pruning. GraSP is efficient and easy to implement; the Hessian-gradient product can be computed without explicitly constructing the Hessian using higher-order automatic differentiation (Pearlmutter, 1994; Schraudolph, 2002).

Algorithm 2 Hessian-gradient Product.

Require: A batch of training data $\mathcal{D}_b$, network f with initial parameters $\boldsymbol{\theta}_0$, loss function $\mathcal{L}$

1: $\mathcal{L}(\boldsymbol{\theta}_0) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_b}[\ell(f(\mathbf{x};\boldsymbol{\theta}_0), y)]$ ▷ Compute the loss and build the computation graph
2: $\mathbf{g} = \mathrm{grad}(\mathcal{L}(\boldsymbol{\theta}_0), \boldsymbol{\theta}_0)$ ▷ Compute the gradient of loss function with respect to $\boldsymbol{\theta}_0$
3: $\mathbf{Hg} = \mathrm{grad}(\mathbf{g}^\top \mathrm{stop\_grad}(\mathbf{g}), \boldsymbol{\theta}_0)$ ▷ Compute the Hessian vector product of $\mathbf{Hg}$
4: Return $\mathbf{Hg}$

Table 2: Test accuracy of pruned VGG19 and ResNet32 on CIFAR-10 and CIFAR-100 datasets. The bold number is the higher one between the accuracy of GraSP and that of SNIP.

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG19 (Baseline)	94.23	-	-	74.16	-	-
OBD (LeCun et al., 1990)	93.74	93.58	93.49	73.83	71.98	67.79
MLPrune (Zeng & Urtasun, 2019)	93.83	93.69	93.49	73.79	73.07	71.69
LT (original initialization)	93.51	92.92	92.34	72.78	71.44	68.95
LT (reset to epoch 5)	93.82	93.61	93.09	74.06	72.87	70.55
DSR (Mostafa & Wang, 2019)	93.75	93.86	93.13	72.31	71.98	70.70
SET Mocanu et al. (2018)	92.46	91.73	89.18	72.36	69.81	65.94
Deep-R (Bellec et al., 2018)	90.81	89.59	86.77	66.83	63.46	59.58
SNIP (Lee et al., 2018)	**93.63±0.06**	**93.43±0.20**	92.05±0.28	**72.84±0.22**	**71.83±0.23**	58.46±1.10
GraSP	93.30±0.14	93.04±0.18	**92.19±0.12**	71.95±0.18	71.23±0.12	**68.90±0.47**
ResNet32 (Baseline)	94.80	-	-	74.64	-	-
OBD (LeCun et al., 1990)	94.17	93.29	90.31	71.96	68.73	60.65
MLPrune (Zeng & Urtasun, 2019)	94.21	93.02	89.65	72.34	67.58	59.02
LT (original initialization)	92.31	91.06	88.78	68.99	65.02	57.37
LT (reset to epoch 5)	93.97	92.46	89.18	71.43	67.28	58.95
DSR (Mostafa & Wang, 2019)	92.97	91.61	88.46	69.63	68.20	61.24
SET Mocanu et al. (2018)	92.30	90.76	88.29	69.66	67.41	62.25
Deep-R (Bellec et al., 2018)	91.62	89.84	86.45	66.78	63.90	58.47
SNIP (Lee et al., 2018)	**92.59±0.10**	91.01±0.21	87.51±0.31	68.89±0.45	65.22±0.69	54.81±1.43
GraSP	92.38±0.21	**91.39±0.25**	**88.81±0.14**	**69.24±0.24**	**66.50±0.11**	**58.43±0.43**

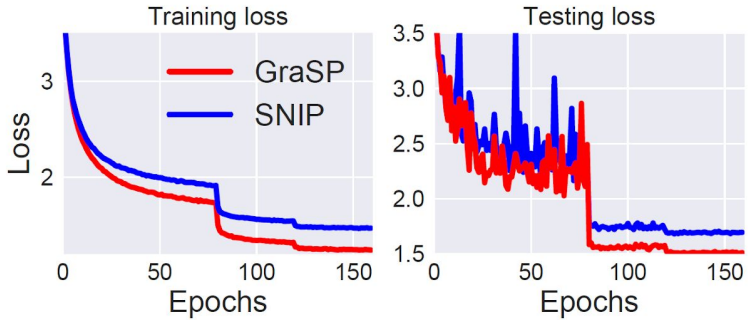


Figure 1: The training and testing loss on CIFAR-100 of SNIP and GraSP with ResNet32 and a pruning ratio of 98%.

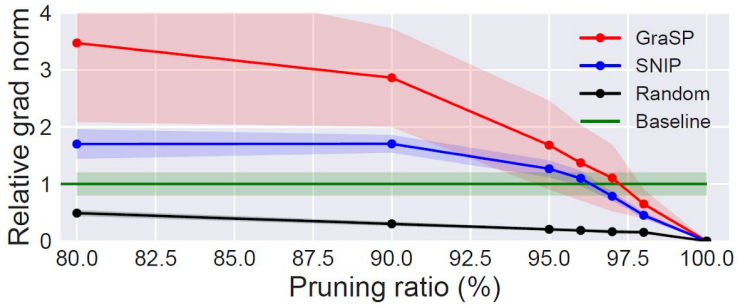


Figure 2: The gradient norm of ResNet32 after pruning on CIFAR-100 of various pruning ratios. Shaded area is the 95% confidence interval calculated with 10 trials.

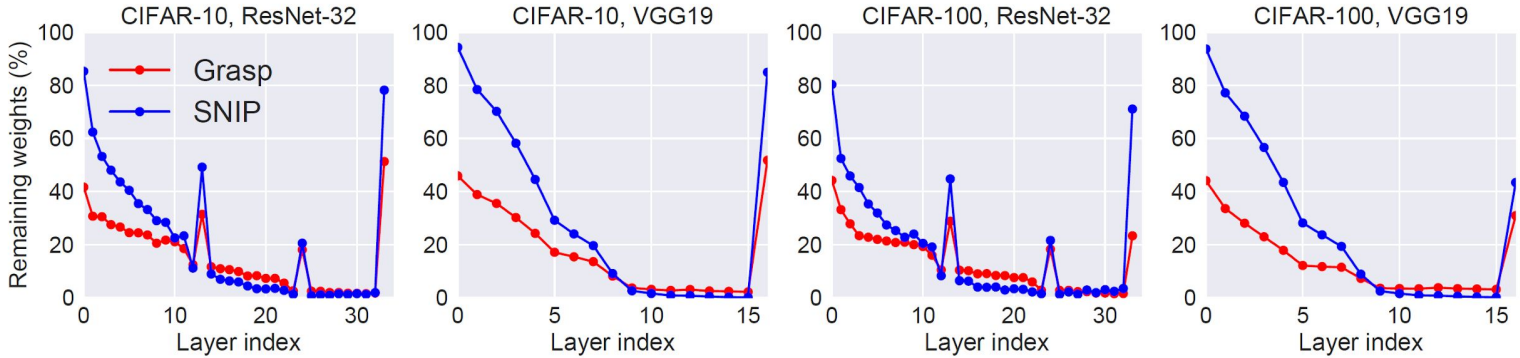


Figure 3: The portion of remaining weights at each layer after pruning with a pruning ratio of 95%.

Pruning neural networks without any data by iteratively conserving synaptic flow

Hidenori Tanaka*
Physics & Informatics Laboratories
NTT Research, Inc.
Department of Applied Physics
Stanford University

Daniel Kunin*
Institute for Computational and
Mathematical Engineering
Stanford University

Daniel L. K. Yamins
Department of Psychology
Department of Computer Science
Stanford University

Surya Ganguli
Department of Applied Physics
Stanford University

NIPS2020

Abstract

Pruning the parameters of deep neural networks has generated intense interest due to potential savings in time, memory and energy both during training and at test time. Recent works have identified, through an expensive sequence of training and pruning cycles, the existence of winning lottery tickets or sparse trainable subnetworks at initialization. This raises a foundational question: can we identify highly sparse trainable subnetworks at initialization, without ever training, or indeed *without ever looking at the data*? We provide an affirmative answer to this question through theory driven algorithm design. We first mathematically formulate and experimentally verify a conservation law that explains why existing gradient-based pruning algorithms at initialization suffer from layer-collapse, the premature pruning of an entire layer rendering a network untrainable. This theory also elucidates how layer-collapse can be entirely avoided, motivating a novel pruning algorithm *Iterative Synaptic Flow Pruning (SynFlow)*. This algorithm can be interpreted as preserving the total flow of synaptic strengths through the network at initialization subject to a sparsity constraint. Notably, this algorithm makes no reference to the training data and consistently competes with or outperforms existing state-of-the-art pruning algorithms at initialization over a range of models (VGG and ResNet), datasets (CIFAR-10/100 and Tiny ImageNet), and sparsity constraints (up to 99.99 percent). Thus our data-agnostic pruning algorithm challenges the existing paradigm that, at initialization, data must be used to quantify which synapses are important.

1. We study *layer-collapse*, the premature pruning of an entire layer making a network untrainable, and formulate the axiom *Maximal Critical Compression* that posits a pruning algorithm should avoid layer-collapse whenever possible (Sec. 3).
2. We demonstrate theoretically and empirically that *synaptic saliency*, a general class of gradient-based scores for pruning, is conserved at every hidden unit and layer of a neural network (Sec. 4).
3. We show that these *conservation laws* imply parameters in large layers receive lower scores than parameters in small layers, which elucidates why single-shot pruning disproportionately prunes the largest layer leading to layer-collapse (Sec. 4).
4. We hypothesize that *iterative magnitude pruning* [10] avoids layer-collapse because gradient descent effectively encourages the magnitude scores to observe a conservation law, which combined with iteration results in the relative scores for the largest layers increasing during pruning (Sec. 5).
5. We prove that a pruning algorithm avoids layer-collapse entirely and satisfies Maximal Critical Compression if it uses iterative, positive synaptic saliency scores (Sec. 6).
6. We introduce a new data-agnostic algorithm *Iterative Synaptic Flow Pruning (SynFlow)* that satisfies Maximal Critical Compression (Sec. 6) and demonstrate empirically[2] that this algorithm achieves state-of-the-art pruning performance on 12 distinct combinations of models and datasets (Sec. 7).

Pruning after training. Conventional pruning algorithms assign scores to parameters in neural networks *after* training and remove the parameters with the lowest scores [5, 23, 24]. Popular scoring metrics include weight magnitudes [4, 6], its generalization to multi-layers [25], first- [1, 26, 27, 28] and second-order [2, 3, 28] Taylor coefficients of the training loss with respect to the parameters, and more sophisticated variants [29, 30, 31]. While these pruning algorithms can indeed compress neural networks at test time, there is no reduction in the cost of training.

Pruning before Training. Recent works demonstrated that randomly initialized neural networks can be pruned *before* training with little or no loss in the final test accuracy [10, 13, 32]. In particular, the Iterative Magnitude Pruning (IMP) algorithm [10, 11] repeats multiple cycles of training, pruning, and weight rewinding to identify extremely sparse neural networks at initialization that can be trained to match the test accuracy of the original network. While IMP is powerful, it requires multiple cycles of expensive training and pruning with very specific sets of hyperparameters. Avoiding these difficulties, a different approach uses the gradients of the training loss at initialization to prune the network in a single-shot [13, 14]. While these single-shot pruning algorithms at initialization are much more efficient, and work as well as IMP at moderate levels of sparsity, they suffer from layer-collapse, or the premature pruning of an entire layer rendering a network untrainable [33, 34]. Understanding and circumventing this layer-collapse issue is the fundamental motivation for our study.

3 Layer-collapse: the key obstacle to pruning at initialization

Broadly speaking, a pruning algorithm at initialization is defined by two steps. The first step scores the parameters of a network according to some metric and the second step masks the parameters (removes or keeps the parameter) according to their scores. The pruning algorithms we consider will always mask the parameters by simply removing the parameters with the smallest scores. This ranking process can be applied globally across the network, or layer-wise. Empirically, its been shown that global-masking performs far better than layer-masking, in part because it introduces fewer hyperparameters and allows for flexible pruning rates across the network [24]. However, recent works [33, 14, 34] have identified a key failure mode, *layer-collapse*, for existing pruning algorithms using global-masking. Layer-collapse occurs when an algorithm prunes all parameters in a single weight layer even when prunable parameters remain elsewhere in the network. This renders the network untrainable, evident by sudden drops in the achievable accuracy for the network as shown in Fig. 1. To gain insight into the phenomenon of layer-collapse we will define some useful terms inspired by a recent paper studying the failure mode [34].

Given a network, *compression ratio* (ρ) is the number of parameters in the original network divided by the number of parameters remaining after pruning. For example, when the compression ratio $\rho = 10^3$, then only one out of a thousand of the parameters remain after pruning. *Max compression* ($\rho_{\max}$) is the maximal possible compression ratio for a network that doesn't lead to layer-collapse. For example, for a network with L layers and N parameters, $\rho_{\max} = N/L$, which is the compression ratio associated with pruning all but one parameter per layer. *Critical compression* (ρ_{cr}) is the maximal compression ratio a given algorithm can achieve without inducing layer-collapse. In particular, the critical compression of an algorithm is always upper bounded by the max compression of the network: $\rho_{\text{cr}} \leq \rho_{\max}$. This inequality motivates the following axiom we postulate any successful pruning algorithm should satisfy.

Axiom. ***Maximal Critical Compression.*** *The critical compression of a pruning algorithm applied to a network should always equal the max compression of that network.*

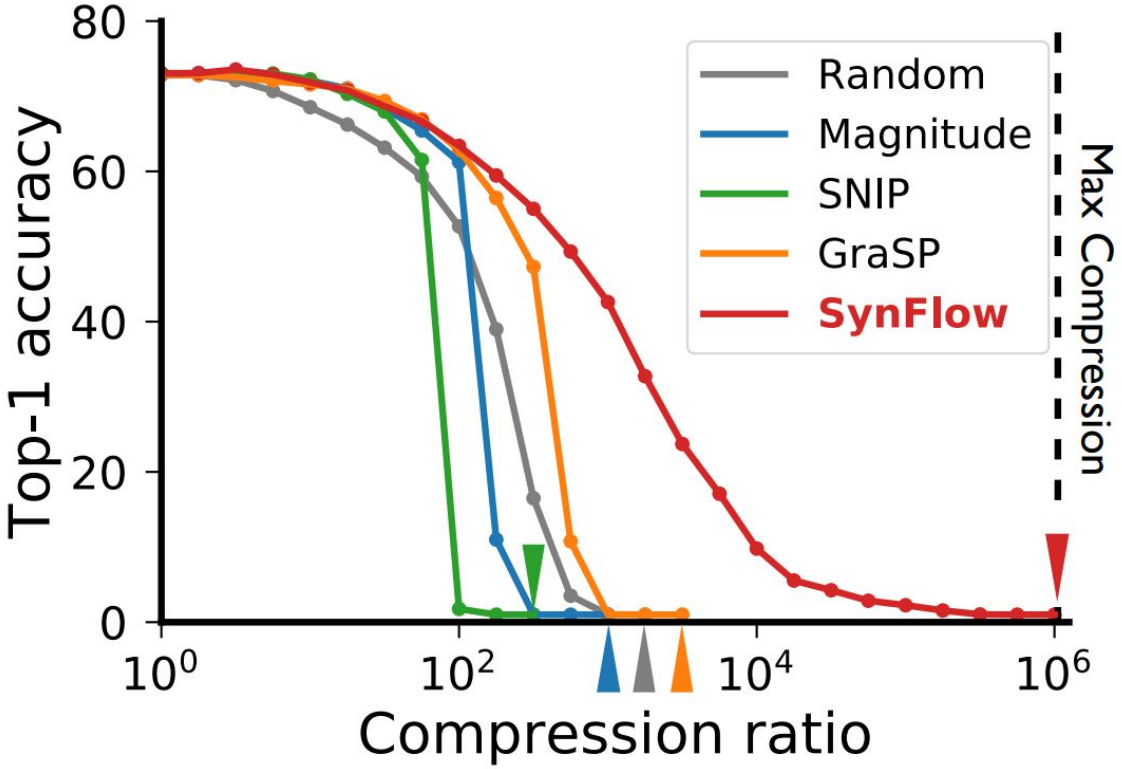


Figure 1: **Layer-collapse leads to a sudden drop in accuracy.** Top-1 test

A general class of gradient-based scores. *Synaptic saliency* is a class of score metrics that can be expressed as the Hadamard product

$$\mathcal{S}(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta, \tag{1}$$

where $\mathcal{R}$ is a scalar loss function of the output y of a feed-forward network parameterized by θ. When $\mathcal{R}$ is the training loss $\mathcal{L}$, the resulting synaptic saliency metric is equivalent (modulo sign) to $-\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta$, the score metric used in Skeletonization [1], one of the first network pruning algorithms. The resulting metric is also closely related to $\left|\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right|$ the score used in SNIP [13], $-\left(H \frac{\partial \mathcal{L}}{\partial \theta}\right) \odot \theta$ the score used in GraSP, and $\left(\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right)^2$ the score used in the pruning after training algorithm Taylor-FO [28]. When $\mathcal{R} = \langle \frac{\partial \mathcal{L}}{\partial y}, y \rangle$, the resulting synaptic saliency metric is closely related to $\text{diag}(H)\theta \odot \theta$, the score used in Optimal Brain Damage [2]. This general class of score metrics, while not encompassing, exposes key properties of gradient-based scores used for pruning.

The conservation of synaptic saliency. All synaptic saliency metrics respect two surprising conservation laws, which we prove in Appendix 9, that hold at any initialization and step in training.

Theorem 1. ***Neuron-wise Conservation of Synaptic Saliency.*** *For a feedforward neural network with continuous, homogeneous activation functions, $\phi(x) = \phi'(x)x$, (e.g. ReLU, Leaky ReLU, linear), the sum of the synaptic saliency for the incoming parameters (including the bias) to a hidden neuron ($\mathcal{S}^{in} = \langle \frac{\partial \mathcal{R}}{\partial \theta^{in}}, \theta^{in} \rangle$) is equal to the sum of the synaptic saliency for the outgoing parameters from the hidden neuron ($\mathcal{S}^{out} = \langle \frac{\partial \mathcal{R}}{\partial \theta^{out}}, \theta^{out} \rangle$).*

Theorem 2. ***Network-wise Conservation of Synaptic Saliency.*** *The sum of the synaptic saliency across any set of parameters that exactly[3] separates the input neurons x from the output neurons y of a feedforward neural network with homogeneous activation functions equals $\langle \frac{\partial \mathcal{R}}{\partial x}, x \rangle = \langle \frac{\partial \mathcal{R}}{\partial y}, y \rangle$.*

Gradient descent encourages conservation. To better understand the dynamics of the IMP algorithm during training, we will consider a differentiable score $\mathcal{S}(\theta_i) = \frac{1}{2}\theta_i^2$ algorithmically equivalent to the magnitude score. Consider these scores throughout training with gradient descent on a loss function $\mathcal{L}$ using an infinitesimal step size (i.e. gradient flow). In this setting, the temporal derivative of the parameters is equivalent to $\frac{d\theta}{dt} = -\frac{\partial \mathcal{L}}{\partial \theta}$, and thus the temporal derivative of the score is $\frac{d}{dt}\frac{1}{2}\theta_i^2 = \frac{d\theta_i}{dt} \odot \theta_i = -\frac{\partial \mathcal{L}}{\partial \theta_i} \odot \theta_i$. Surprisingly, this is a form of synaptic saliency and thus the

The Iterative Synaptic Flow Pruning (SynFlow) algorithm. Theorem 3 directly motivates the design of our novel pruning algorithm, SynFlow, that provably reaches Maximal Critical Compression. First, the necessity for iterative score evaluation discourages algorithms that involve backpropagation on batches of data, and instead motivates the development of an efficient data-independent scoring procedure. Second, positivity and conservation motivates the construction of a loss function that yields positive synaptic saliency scores. We combine these insights to introduce a new loss function (where $\mathbb{1}$ is the all ones vector and $|\theta^{[l]}|$ is the element-wise absolute value of parameters in the l^{th} layer),

$$\mathcal{R}_{\text{SF}} = \mathbb{1}^T \left(\prod_{l=1}^{L} |\theta^{[l]}| \right) \mathbb{1} \tag{2}$$

that yields the positive, synaptic saliency scores ($\frac{\partial \mathcal{R}_{\text{SF}}}{\partial \theta} \odot \theta$) we term Synaptic Flow. For a simple,

Algorithm 1: Iterative Synaptic Flow Pruning (SynFlow).

Input: network $f(x; \theta_0)$, compression ratio ρ, iteration steps n

0: $f(x; \theta_0)$; ▷Set model to eval mode[a]

1: $\mu = \mathbb{1}$; ▷Initialize binary mask

for k *in* $[1, \ldots, n]$ **do**

2: $\theta_\mu \leftarrow \mu \odot \theta_0$; ▷Mask parameters

3: $\mathcal{R} \leftarrow \mathbb{1}^T \left(\prod_{l=1}^{L} |\theta_\mu^{[l]}| \right) \mathbb{1}$; ▷Evaluate SynFlow objective

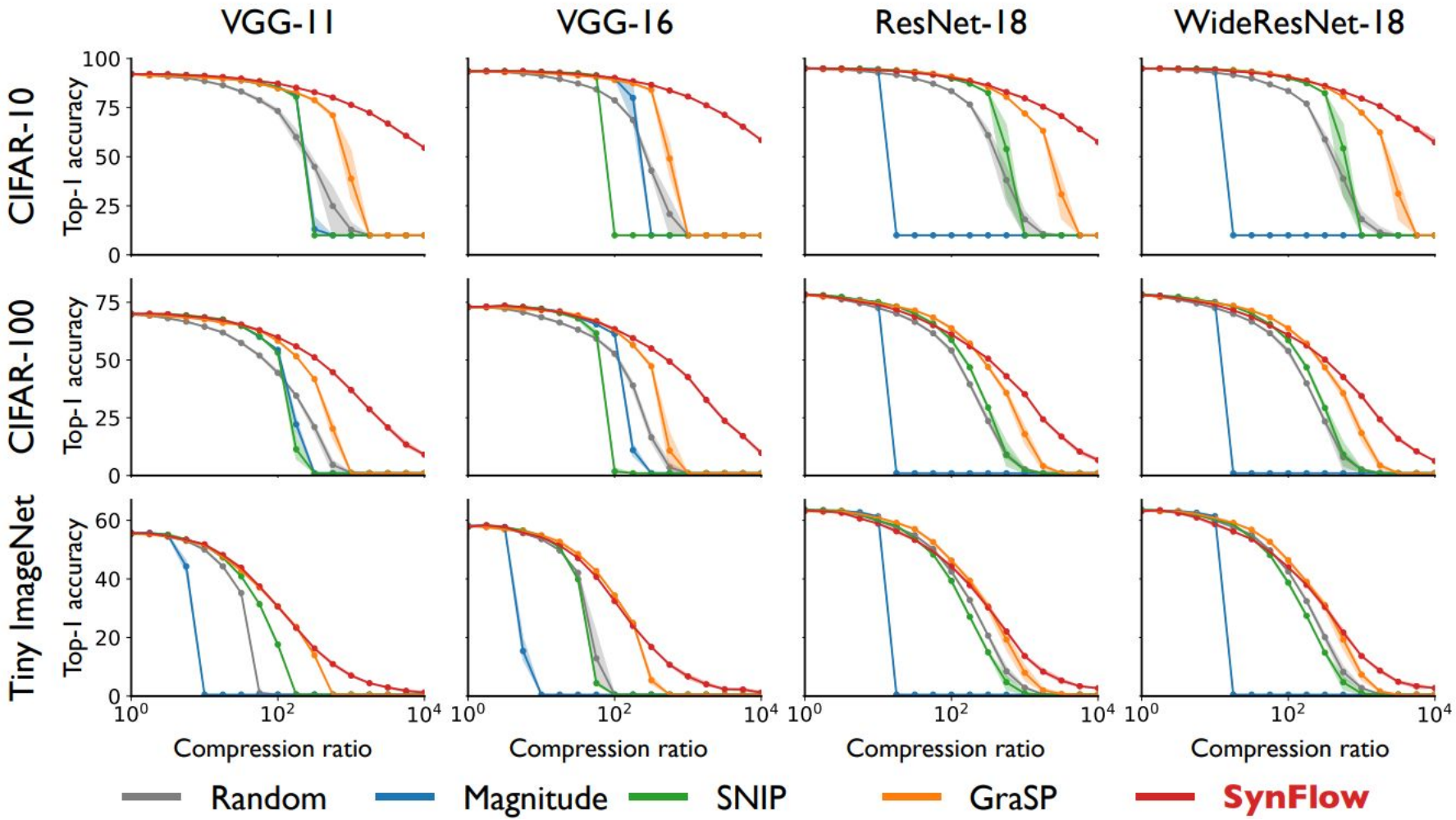
4: $\mathcal{S} \leftarrow \frac{\partial \mathcal{R}}{\partial \theta_\mu} \odot \theta_\mu$; ▷Compute SynFlow score

5: $\tau \leftarrow (1 - \rho^{-k/n})$ percentile of $\mathcal{S}$; ▷Find threshold

6: $\mu \leftarrow (\tau < \mathcal{S})$; ▷Update mask

end

7: $f(x; \mu \odot \theta_0)$; ▷Return masked network


VGG-11
VGG-16
ResNet-18
WideResNet-18
CIFAR-10
CIFAR-100
Tiny ImageNet
Top-1 accuracy
Compression ratio
Random
Magnitude
SNIP
GraSP
SynFlow

Published as a conference paper at ICLR 2021

PRUNING NEURAL NETWORKS AT INITIALIZATION: WHY ARE WE MISSING THE MARK?

Jonathan Frankle
MIT CSAIL

Gintare Karolina Dziugaite
Element AI

Daniel M. Roy
University of Toronto
Vector Institute

Michael Carbin
MIT CSAIL

The state of the art for pruning at initialization. The methods for pruning at initialization (SNIP, GraSP, SynFlow, and magnitude pruning) generally outperform random pruning. No single method is SOTA: there is a network, dataset, and sparsity where each pruning method (including magnitude pruning) reaches the highest accuracy. SNIP consistently performs well, magnitude pruning is surprisingly effective, and competition increases with improvements we make to GraSP and SynFlow.

Magnitude pruning after training outperforms these methods. Although this result is not necessarily surprising (after all, these methods have less readily available information upon which to prune), it raises the question of whether there may be broader limitations to the performance achievable when pruning at initialization. In the rest of the paper, we study this question, investigating how these methods differ behaviorally from standard results about pruning after training.

Methods prune layers, not weights. The subnetworks that these methods produce perform equally well (or better) when we randomly shuffle the weights they prune in each layer; it is therefore possible to describe a family of equally effective pruning techniques that randomly prune the network in these per-layer proportions. The subnetworks that these methods produce also perform equally well when we randomly reinitialize the unpruned weights. These behaviors are not shared by state-of-the-art weight-pruning methods that operate after training; both of these ablations (shuffling and reinitialization) lead to lower accuracy (Appendix F; Han et al., 2015; Frankle & Carbin, 2019).

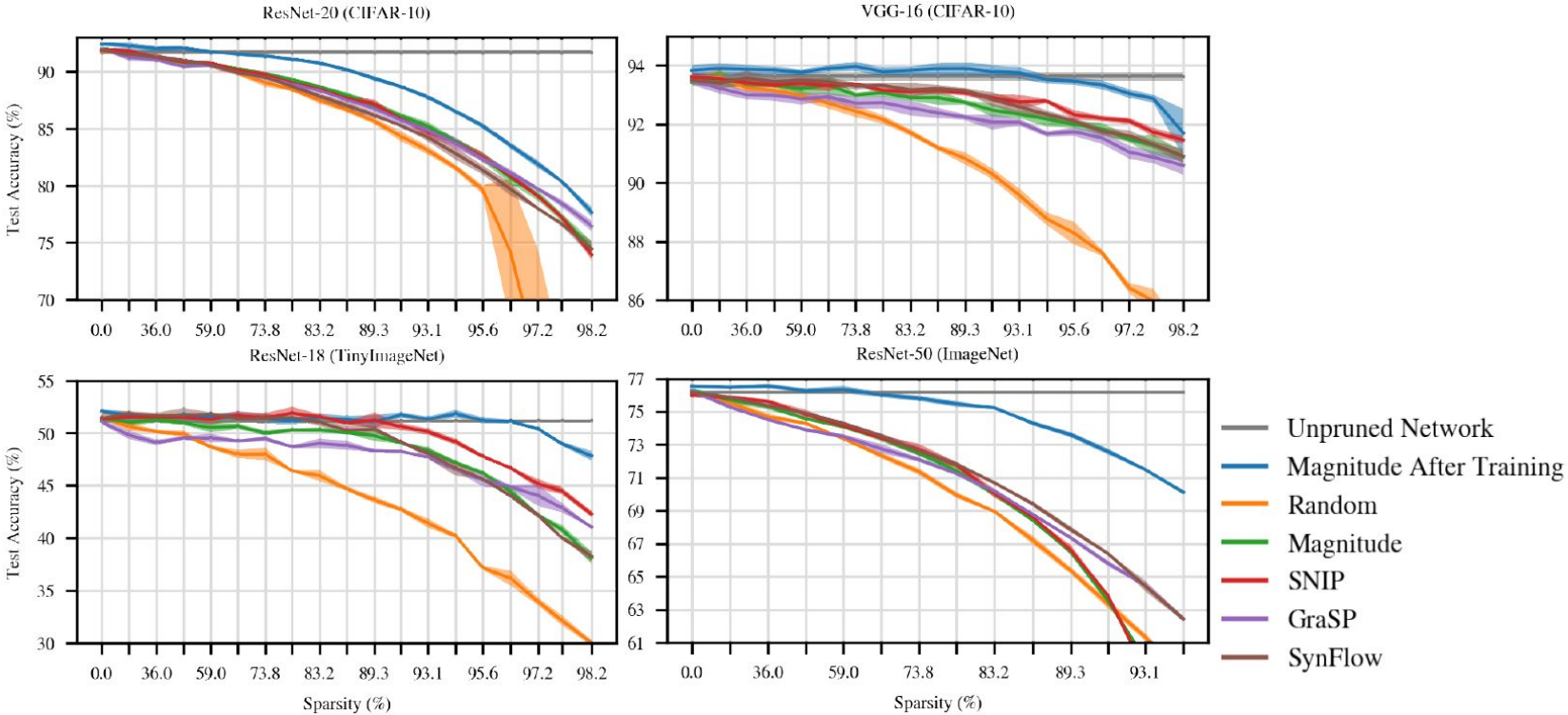


Figure 3: Accuracy of early pruning methods when pruning at initialization to various sparsities.

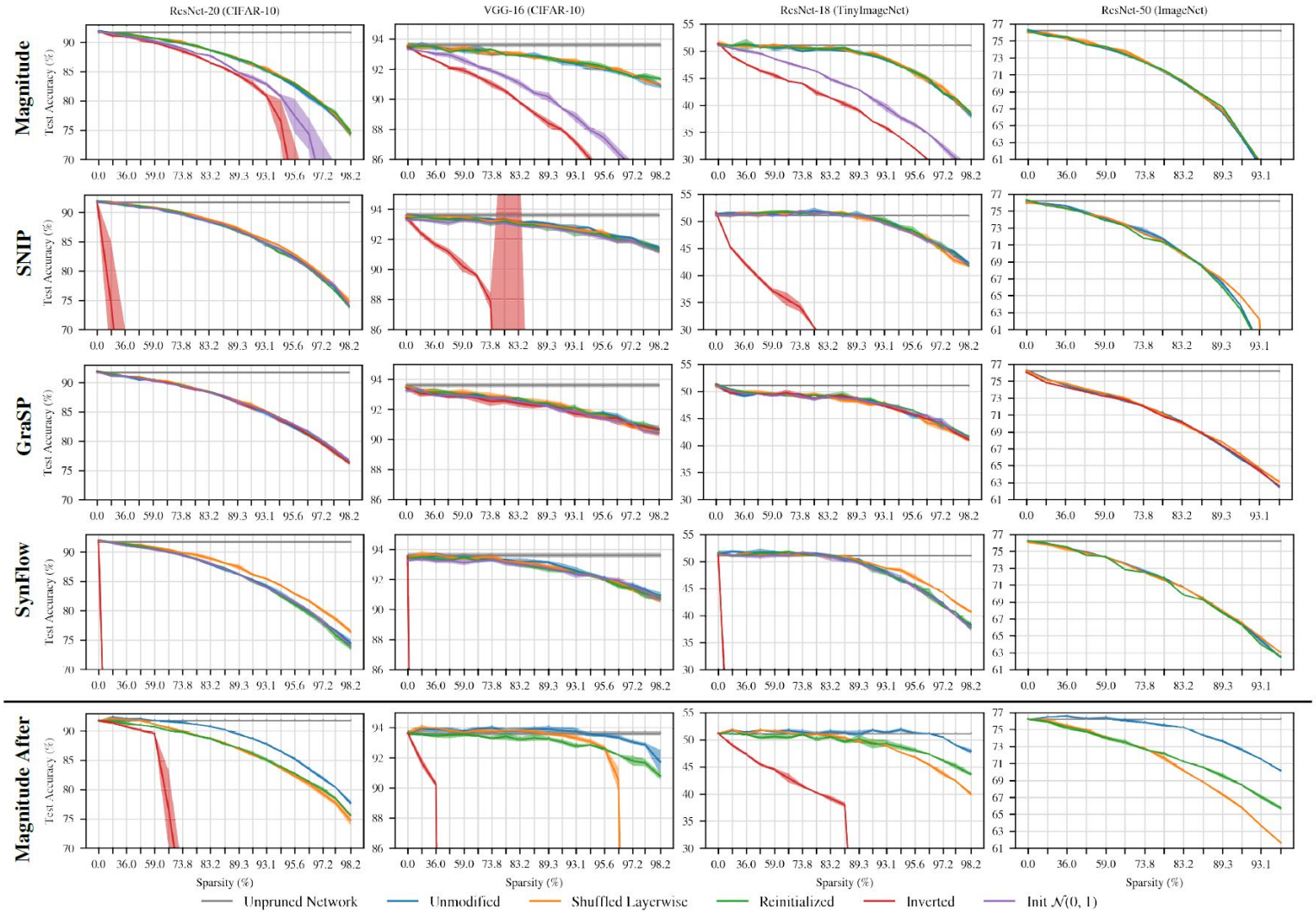
Randomly shuffling. We first consider whether these pruning methods prune specific connections. To do so, we randomly shuffle the pruning mask m_ℓ within each layer. If accuracy is the same after shuffling, then the per-weight decisions made by the method can be replaced by the per-layer fraction of weights it pruned. If accuracy changes, then the method has determined which parts of the network to prune at a smaller granularity than layers, e.g., neurons or individual connections.

Overall. All methods maintain accuracy or improve when randomly shuffled (Figure 4, orange line). In other words, the useful information these techniques extract is not which individual weights to remove, but rather the layerwise proportions by which to prune the network.[4] Although layerwise proportions are an important hyperparameter for inference-focused pruning methods (He et al., 2018; Gale et al., 2019), proportions alone are not sufficient to explain the performance of those methods. For example, as the bottom row of Figure 4 shows, magnitude pruning after training makes pruning decisions specific to particular weights; shuffling in this manner reduces performance (Frankle & Carbin, 2019). In general, we know of no state-of-the-art weight-pruning methods (among the many that exist (Blalock et al., 2020)) with accuracy robust to this ablation. This raises the question of whether the insensitivity to shuffling for the early pruning methods may limit their performance.

Reinitialization. We next consider whether the networks produced by these methods are sensitive to the specific initial values of their weights. That is, is performance maintained when sampling a new initialization for the pruned network from the same distribution as the original network? Magnitude pruning after training and LTR are known to be sensitive this ablation: when reinitialized, pruned networks train to lower accuracy (Figure 4 bottom row; Appendix F; Han et al., 2015; Frankle & Carbin, 2019), and we know of no state-of-the-art weight-pruning methods with accuracy robust to this ablation. However, all early pruning techniques are unaffected by reinitialization (green line): accuracy is the same whether the network is trained with the original initialization or a newly sampled initialization. As with random shuffling, this raises the question of whether this insensitivity to initialization may pose a limit to these methods that restricts performance.

Inversion. SNIP, GraSP, and SynFlow are each based on a hypothesis about properties of the network or training that allow a sparse network to reach high accuracy. Scoring functions should rank weights from most important to least important according to these hypotheses, making it possible to preserve the most important weights when pruning to any sparsity. In this ablation, we assess whether the scoring functions successfully do so: we prune the *most* important weights and retain the *least* important weights. If the hypotheses behind these methods are correct and they are accurately instantiated as scoring functions, then *inverting* in this manner should lower performance.

Magnitude, SNIP, and SynFlow behave as expected: when pruning the most important weights, accuracy decreases (red line). In contrast, GraSP's accuracy does not change when pruning the most important weights. This result calls into question the premise behind GraSP's heuristic: one can keep the weights that, according to Wang et al. (2020), *decrease* gradient flow the most and get the same accuracy as keeping those that purportedly increase it the most. Moreover, we find that pruning weights with the lowest-magnitude GraSP scores improves accuracy (Appendix H).


ResNet-20 (CIFAR-10)
VGG-16 (CIFAR-10)
ResNet-18 (TinyImageNet)
ResNet-50 (ImageNet)
Magnitude
SNIP
GraSP
SynFlow
Magnitude After
Test Accuracy (%)
Sparsity (%)
0.0 36.0 59.0 73.8 83.2 89.3 93.1 95.6 97.2 98.2
Unpruned Network
Unmodified
Shuffled Layerwise
Reinitialized
Inverted
Init $\mathcal{N}(0, 1)$

Questions?

Structured Pruning

Neuron-level Structured Pruning using Polarization Regularizer

Tao Zhuang[1], **Zhixuan Zhang**[*1], **Yuheng Huang**[2], **Xiaoyi Zeng**[1], **Kai Shuang**[2], **Xiang Li**[1]
[1]Alibaba Group
[2]Beijing University of Posts and Telecommunications
{zhuangtao.zt, zhibing.zzx}@alibaba-inc.com, hyhlryf@bupt.edu.cn,
yuanhan@taobao.com, shuangk@bupt.edu.cn, leo.lx@alibaba-inc.com

NIPS2020

Network pruning is proved to effectively reduce the computational cost of inference without significantly compromising accuracy [Liu et al., 2019a]. There are two major branches of network pruning. One branch is *unstructured* pruning, which prunes at the level of individual weights [LeCun et al., 1989, Han et al., 2015, Zhou et al., 2019, Ding et al., 2019, Frankle and Carbin, 2019]. The other branch is *structured* pruning, which prunes at the level of neurons (or channels) [Wen et al., 2016, Liu et al., 2017, Ye et al., 2018]. Although unstructured pruning usually reduces more weights than structured pruning [Liu et al., 2019a], it has the drawback that the resulting weight matrices are sparse, which cannot lead to speedup without dedicated hardware or libraries [Han et al., 2016]. In this paper, we focus on neuron-level structured pruning, which does not require additional hardware or libraries to reduce computation on common GPU/CPU devices.

Given a training set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, where $\boldsymbol{x}_i, \boldsymbol{y}_i$ denote the input feature and the label of sample i respectively, we need to train a neural network $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ denotes the parameters of the network. We introduce a scaling factor for each neuron and represent the scaling factors as a vector $\boldsymbol{\gamma} \in \mathbf{R}^n$, where n is the number of neurons in the network. As in [Liu et al., 2017], we use the scale factor in BN as the scaling factor of each neuron, because of the wide adoption of BN in modern neural networks. The objective function for network training with regularization on scaling factors is:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{f}(\boldsymbol{x_i}; \boldsymbol{\theta}), \boldsymbol{y}_i) + R(\boldsymbol{\theta}) + \lambda R_s(\boldsymbol{\gamma}) \tag{1}$$

where $L(\cdot)$ is the loss function, $R(\cdot)$ is usually the L2 regularization on weights of the network, and $R_s(\cdot)$ is the sparsity regularizer on scaling factors of neurons. In pruning, a threshold is chosen and neurons with scaling factors below the threshold are pruned. In [Liu et al., 2017], the sparsity regularizer is chosen to be L1, i.e. $R_s(\boldsymbol{\gamma}) = \|\boldsymbol{\gamma}\|_1$. The effect of L1 regularization is to push all scaling parameters to 0. Therefore, L1 regularization lacks discrimination between pruned and preserved neurons. A more reasonable pruning method is to only suppress unimportant neurons (with

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The simple equation behind batch normalization

Let $\boldsymbol{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_n)$, and $\mathbf{1}_n = (1, 1, \cdots, 1) \in \mathbf{R}^n$. Let $\bar{\gamma}$ denote the mean of $\gamma_1, \ldots, \gamma_n$:

$$\bar{\gamma} := \frac{1}{n}\mathbf{1}_n^\top \boldsymbol{\gamma} = \frac{1}{n}\sum_{i=1}^{n} \gamma_i$$

γ_i is the scaling factor for neuron i. Obviously, a scaling factor should be positive and bounded. So

the polarization regularizer to be:

$$\begin{aligned} R_s(\boldsymbol{\gamma}) &= t\|\boldsymbol{\gamma}\|_1 - \|\boldsymbol{\gamma} - \bar{\gamma}\mathbf{1}_n\|_1 \\ &= \sum_{i=1}^{n} t|\gamma_i| - |\gamma_i - \bar{\gamma}|, (t \in \mathbb{R}, \gamma_i \in [0, a]) \end{aligned} \tag{2}$$

The polarization regularizer has several nice properties. The first property is *permutation invariance*: Let $\pi(\boldsymbol{\gamma})$ denote the vector obtained by an arbitrary permutation of $\boldsymbol{\gamma}$ on its n dimensions. Then it is obvious that $R_s(\pi(\boldsymbol{\gamma})) = R_s(\boldsymbol{\gamma})$, which means that the polarization regularizer is permutation invariant. This property ensures all neurons be equally treated in pruning: no prior pruning bias for any neuron. The second property is concavity, as stated in Lemma 3.1:

Lemma 3.1. *$R_s(\boldsymbol{\gamma})$ defined in Equation (2) is concave.*

To see the effect of polarization regularizer more clearly, we need to solve the following minimization problem:

$$\min_{\boldsymbol{\gamma}\in[0,a]^n} R_s(\boldsymbol{\gamma}) = \sum_{i=1}^{n} t|\gamma_i| - |\gamma_i - \bar{\gamma}|, (t \in \mathbb{R}) \tag{3}$$

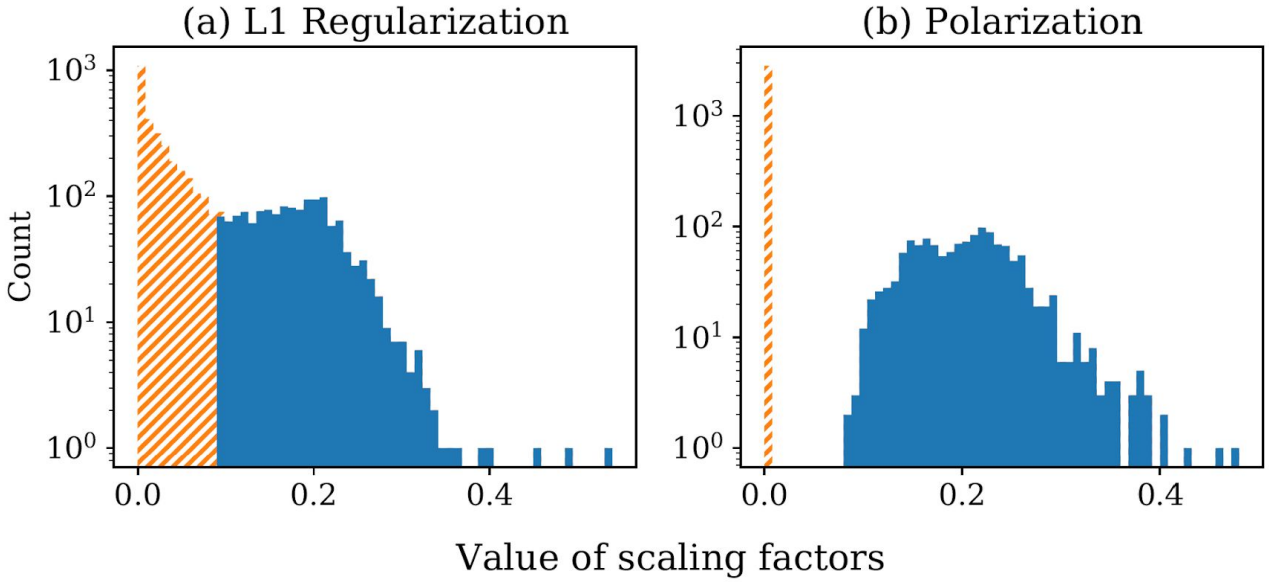


Figure 1: The distributions of scaling factors in VGG-16 trained on CIFAR-10 dataset, with L1 and polarization regularizers respectively. Under the same pruning ratio for both regularizers, the orange part are pruned.

Table 2: Results on ImageNet. Best results are bolded.

Model	Approach	Baseline	Pruned		
		Acc. (%)	Acc. (%)	Acc. Drop (%)	FLOPs Reduction
ResNet-50	NS [Liu et al., 2017] (Our-impl.)	76.15	74.88	1.27	53%
	SSS [Huang and Wang, 2018]	76.12	71.82	4.30	43%
	DCP [Zhuang et al., 2018]	76.01	74.95	1.06	56%
	FPGM [He et al., 2019]	76.15	74.13	2.02	53%
	CCP [Peng et al., 2019]	76.15	75.21	0.94	54%
	MetaPruning [Liu et al., 2019b]	76.6	75.4	1.2	50%
	SFP [He et al., 2018a]	76.15	62.14	14.01	42%
	PFP [Liebenwein et al., 2020]	76.13	75.21	0.92	30%
	AutoPruner [Luo and Wu, 2020]	76.15	74.76	1.39	49%
	Ours	76.15	75.63	**0.52**	54%
MobileNet v2	AMC [He et al., 2018b]	71.8	70.8	1.0	27%
	MetaPruning [Liu et al., 2019b]	72.0	71.2	0.8	27%
	DeepHoyer [Yang et al., 2020] (Our-impl.)	72.0	71.7	0.3	25%
	Ours	72.0	71.8	**0.2**	28%

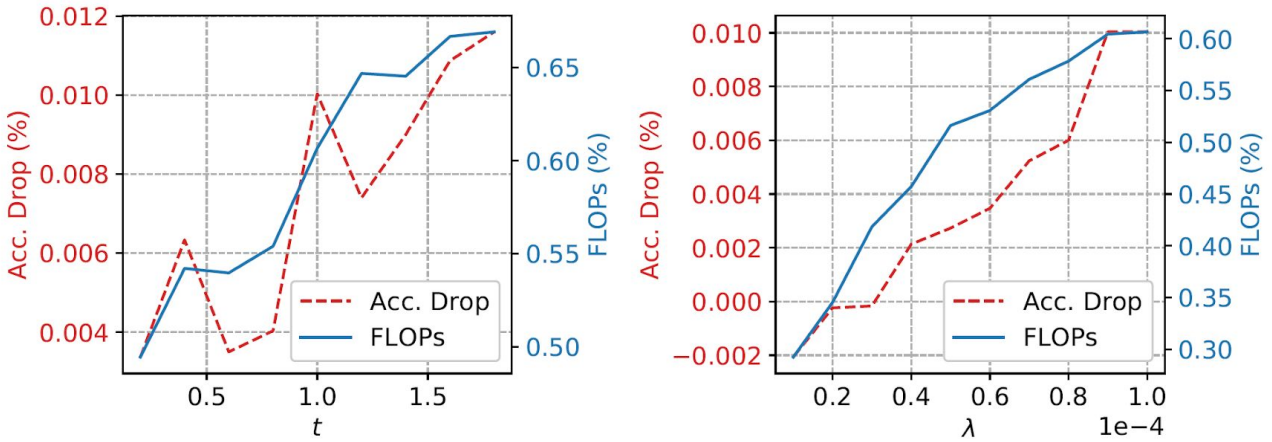
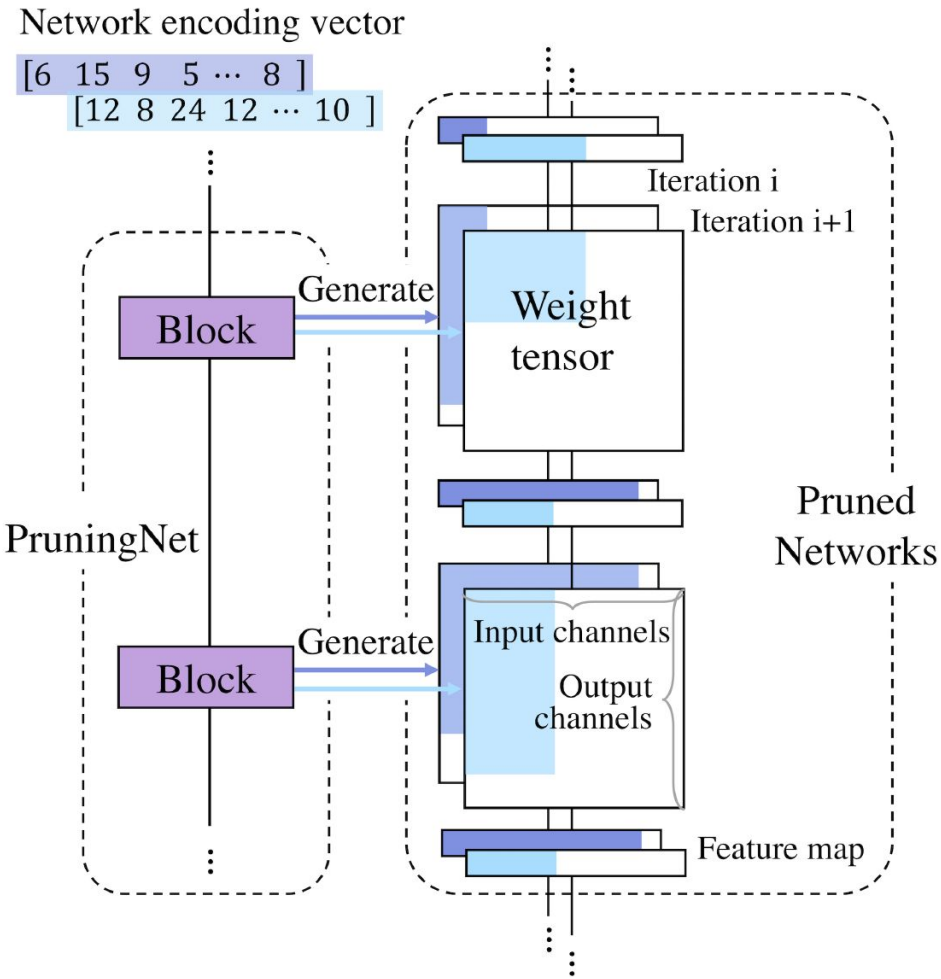
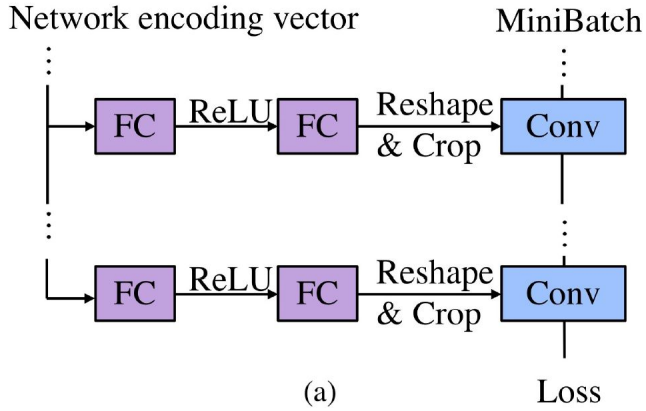


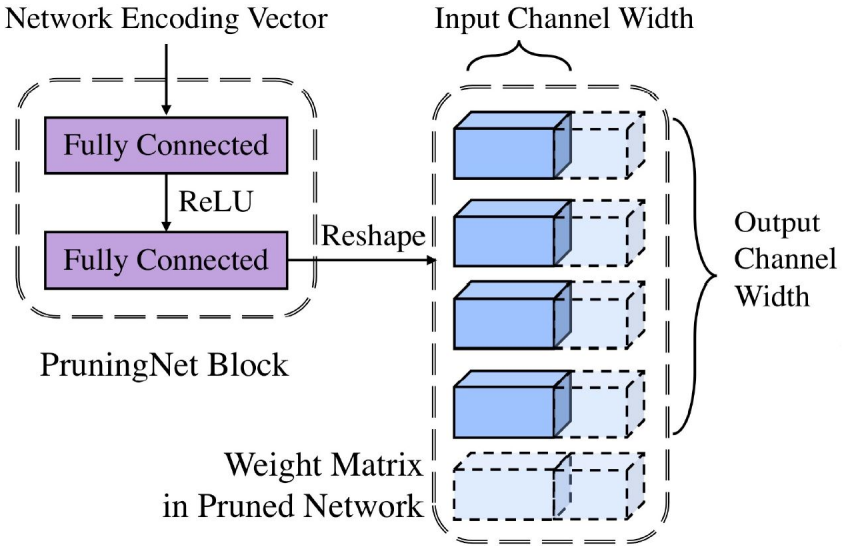
Figure 3: The effect of the hyper-parameters t and λ on the reduced FLOPs and accuracy drop for polarization pruning. When we draw the figure of t, the value of λ is fixed to 1e-4. When we draw the figure of λ, the value of t is fixed to 1.0. We conduct each experiment for 3 times and report the mean value of the accuracy drop and the FLOPs.

MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning

Zechun Liu[1] Haoyuan Mu[2] Xiangyu Zhang[3] Zichao Guo[3] Xin Yang[4]
Tim Kwang-Ting Cheng[1] Jian Sun[3]
[1] Hong Kong University of Science and Technology [2] Tsinghua University
[3] Megvii Technology [4] Huazhong University of Science and Technology


Network encoding vector
[6 15 9 5 ⋯ 8]
[12 8 24 12 ⋯ 10]
PruningNet
Block
Generate
Block
Generate
Iteration i
Iteration i+1
Weight tensor
Input channels
Output channels
Feature map
Pruned Networks


Network encoding vector
MiniBatch
FC
ReLU
FC
Reshape
& Crop
Conv
FC
ReLU
FC
Reshape
& Crop
Conv
(a)
Loss


Network Encoding Vector
Input Channel Width
Fully Connected
ReLU
Fully Connected
Reshape
Output
Channel
Width
PruningNet Block
Weight Matrix
in Pruned Network

Algorithm 1 Evolutionary Search Algorithm

Hyper Parameters: Population Size: $\mathcal{P}$, Number of Mutation: $\mathcal{M}$, Number of Crossover: $\mathcal{S}$, Max Number of Iterations: $\mathcal{N}$.

Input: PruningNet: $PruningNet$, Constraints: $\mathcal{C}$.

Output: Most accurate gene: $\mathcal{G}_{top}$.

$\mathcal{G}_0$ = Random($\mathcal{P}$), s.t. $\mathcal{C}$;
$\mathcal{G}_{topK} = \emptyset$;
for $i = 0 : \mathcal{N}$ **do**
    $\{\mathcal{G}_i, \text{accuracy}\}$ = Inference($PruningNet(\mathcal{G}_i)$);
    $\mathcal{G}_{topK}, accuracy_{topK}$ = TopK($\{\mathcal{G}_i, accuracy\}$);
    $\mathcal{G}_{mutation}$ = Mutation($\mathcal{G}_{topK}, \mathcal{M}$), s.t. $\mathcal{C}$;
    $\mathcal{G}_{crossover}$ = Crossover($\mathcal{G}_{topK}, \mathcal{S}$), s.t. $\mathcal{C}$;
    $\mathcal{G}_i = \mathcal{G}_{mutation} + \mathcal{G}_{crossover}$;
end for
$\mathcal{G}_{top1}, accuracy_{top1}$= Top1($\{\mathcal{G}_{\mathcal{N}}, accuracy\}$);
return $\mathcal{G}_{top1}$;

HRank: Filter Pruning using High-Rank Feature Map

Mingbao Lin[1], Rongrong Ji[1,5*], Yan Wang[2], Yichen Zhang[1],
Baochang Zhang[3], Yonghong Tian[4,5], Ling Shao[6]
[1]Media Analytics and Computing Laboratory, Department of Artificial Intelligence, School of Informatics, Xiamen University, China, [2]Pinterest, USA, [3]Beihang University, China
[4]Peking University, Beijing, China, [5]Peng Cheng Laboratory, Shenzhen, China
[6]Inception Institute of Artificial Intelligence, Abu Dhabi, UAE
lmbxmu@stu.xmu.edu.cn, rrji@xmu.edu.cn, yanw@pinterest.com, ethan.zhangyc@gmail.com,
bczhang@buaa.edu.cn, yhtian@pku.edu.cn, ling.shao@ieee.org

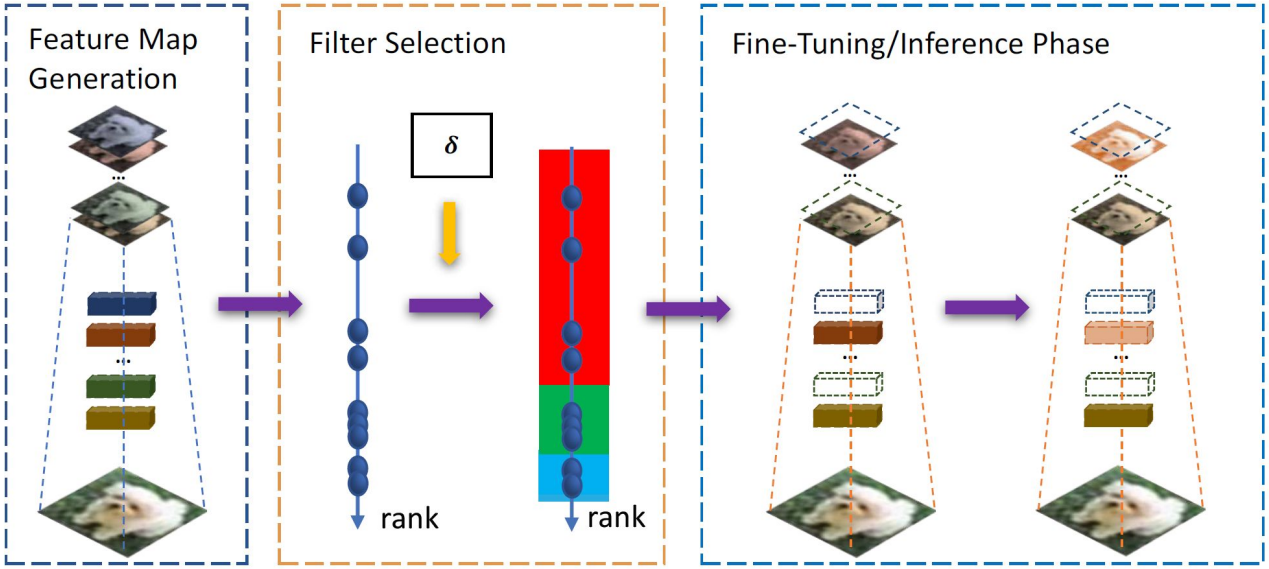
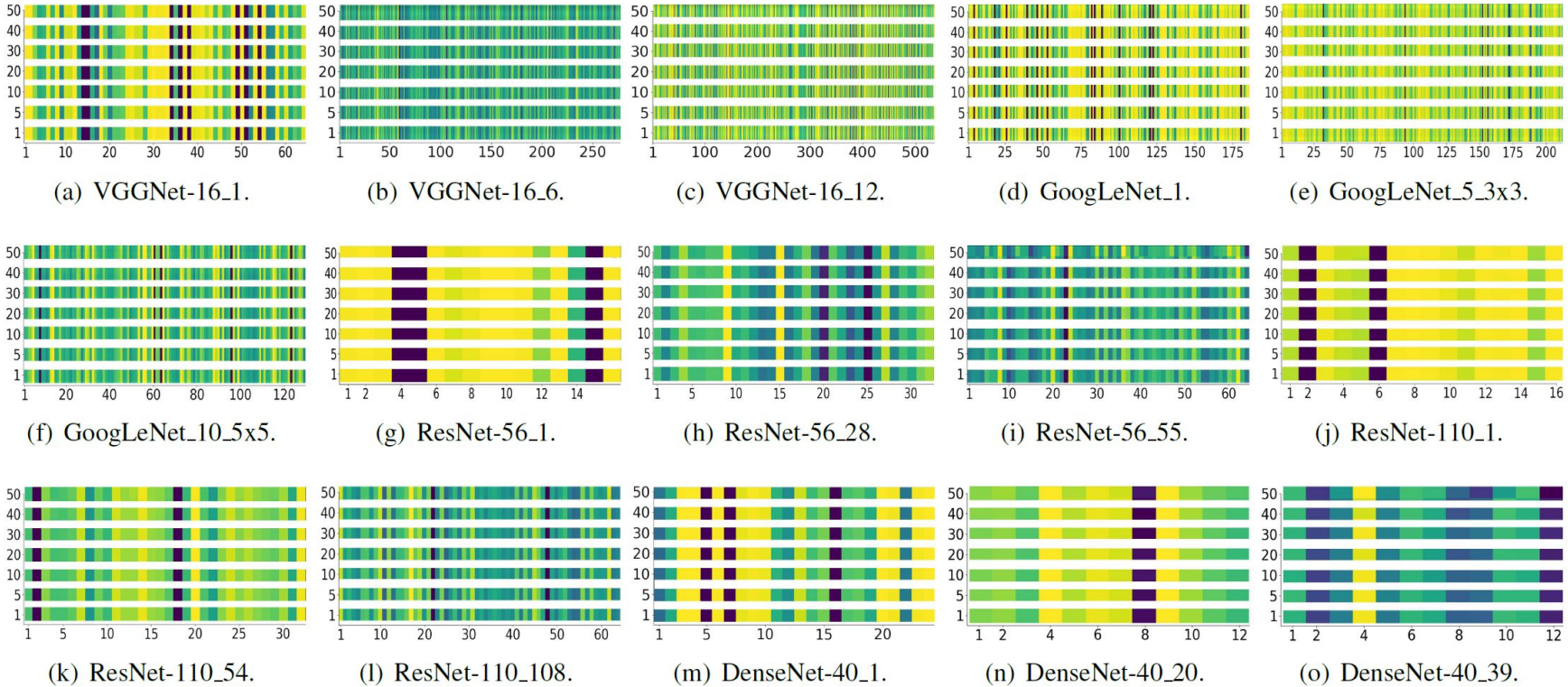


Figure 1. Framework of HRank. In the left column, we first use images to run through the convolutional layers to get the feature maps. In the middle column, we then estimate the rank of each feature map, which is used as the criteria for pruning. The right column shows the pruning (the red filters), and fine-tuning where the green filters are updated and the blue filters are frozen.



(a) VGGNet-16_1. (b) VGGNet-16_6. (c) VGGNet-16_12. (d) GoogLeNet_1. (e) GoogLeNet_5_3x3.

(f) GoogLeNet_10_5x5. (g) ResNet-56_1. (h) ResNet-56_28. (i) ResNet-56_55. (j) ResNet-110_1.

(k) ResNet-110_54. (l) ResNet-110_108. (m) DenseNet-40_1. (n) DenseNet-40_20. (o) DenseNet-40_39.

Figure 2. Average rank statistics of feature maps from different convolutional layers and architectures on CIFAR-10. For each subfigure, the x-axis represents the indices of feature maps and the y-axis is the batches of training images (each batch size is set to 128). Different colors denote different rank values. As can be seen, the rank of each feature map (column of the subfigure) is almost unchanged (the same color), regardless of the image batches. Hence, even a small number of images can effectively estimate the average rank of each feature map in different architectures.

Table 3. Pruning results of ResNet-56/110 on CIFAR-10.

Model	Top-1%	FLOPs(PR)	Parameters(PR)
ResNet-56	93.26	125.49M(0.0%)	0.85M(0.0%)
L1 [18]	93.06	90.90M(27.6%)	0.73M(14.1%)
HRank(Ours)	93.52	88.72M(29.3%)	0.71M(16.8%)
NISP [34]	93.01	81.00M(35.5%)	0.49M(42.4%)
GAL-0.6	92.98	78.30M(37.6%)	0.75M(11.8%)
HRank(Ours)	93.17	62.72M(50.0%)	0.49M(42.4%)
He *et al.* [13]	90.80	62.00M(50.6%)	-
GAL-0.8	90.36	49.99M(60.2%)	0.29M(65.9%)
HRank(Ours)	90.72	32.52M(74.1%)	0.27M(68.1%)
ResNet-110	93.50	252.89M(0.0%)	1.72M(0.0%)
L1 [18]	93.30	155.00M(38.7%)	1.16M(32.6%)
HRank(Ours)	94.23	148.70M(41.2%)	1.04M(39.4%)
GAL-0.5 [23]	92.55	130.20M(48.5%)	0.95M(44.8%)
HRank(Ours)	93.36	105.70M(58.2%)	0.70M(59.2%)
HRank(Ours)	92.65	79.30M(68.6%)	0.53M(68.7%)

Table 4. Pruning results of DenseNet-40 on CIFAR-10.

Model	Top-1%	FLOPs(PR)	Parameters(PR)
DenseNet-40	94.81	282.00M(0.0%)	1.04M(0.0%)
Liu *et al.*-40% [24]	94.81	190.00M(32.8%)	0.66M(36.5%)
GAL-0.01 [23]	94.29	182.92M(35.3%)	0.67M(35.6%)
HRank(Ours)	94.24	167.41M(40.8%)	0.66M(36.5%)
Zhao *et al.* [36]	93.16	156.00M(44.8%)	0.42M(59.7%)
GAL-0.05 [23]	93.53	128.11M(54.7%)	0.45M(56.7%)
HRank(Ours)	93.68	110.15M(61.0%)	0.48M(53.8%)

Table 5. Pruning results of ResNet-50 on ImageNet.

Model	Top-1%	Top-5%	FLOPs	Parameters
ResNet-50 [26]	76.15	92.87	4.09B	25.50M
SSS-32 [16]	74.18	91.91	2.82B	18.60M
He *et al.* [13]	72.30	90.80	2.73B	-
GAL-0.5 [23]	71.95	90.94	2.33B	21.20M
HRank(Ours)	74.98	92.33	2.30B	16.15M
GDP-0.6 [22]	71.19	90.71	1.88B	-
GDP-0.5 [22]	69.58	90.14	1.57B	-
SSS-26 [16]	71.82	90.79	2.33B	15.60M
GAL-1 [23]	69.88	89.75	1.58B	14.67M
GAL-0.5-joint [23]	71.80	90.82	1.84B	19.31M
HRank(Ours)	71.98	91.01	1.55B	13.77M
ThiNet-50 [26]	68.42	88.30	1.10B	8.66M
GAL-1-joint [23]	69.31	89.12	1.11B	10.21M
HRank(Ours)	69.10	89.58	0.98B	8.27M

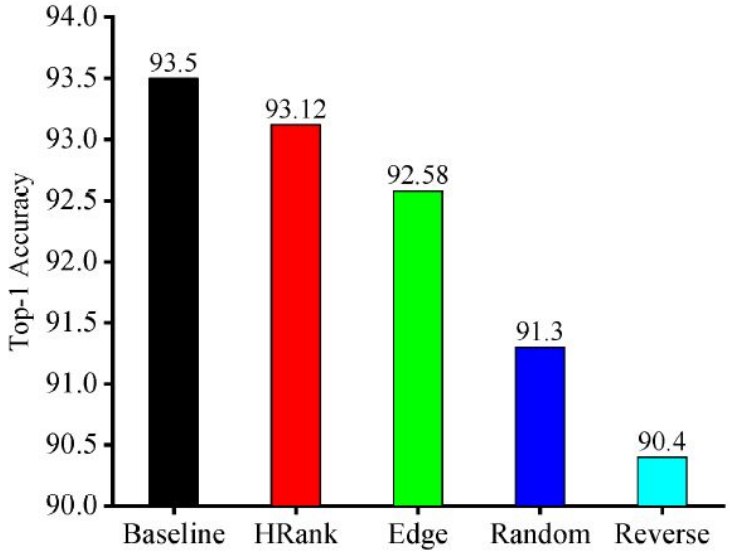
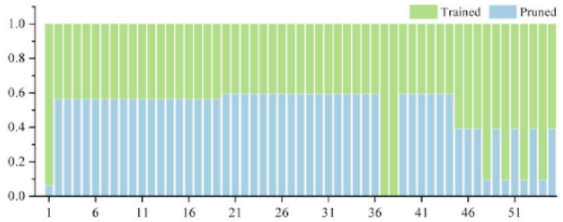


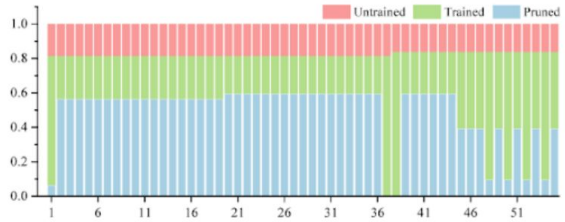
Figure 4. Top-1 accuracy for variants of HRank.

Variants of HRank. Three variants are proposed to demonstrate the appropriateness of preserving filters with high-rank feature maps, including: (1) Edge: Filters generating both low- and high-rank feature maps are pruned, (2) Random: Filters are randomly pruned. (3) Reverse: Filters generating high-rank feature maps are pruned. The pruning

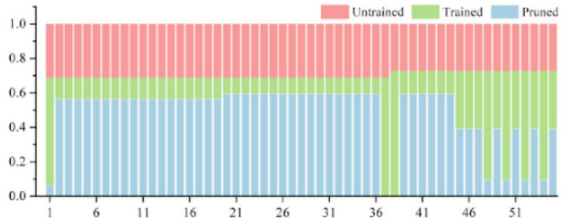
- Filter freezing during fine-tuning(ResNet-56 on CIFAR-10)



(a) 0% of the filters are frozen, resulting in a top-1 precision of 93.17%.



(b) 15% - 20% of the filters are frozen, resulting in a top-1 precision of 93.13%.



(c) 20% - 25% of the filters are frozen, resulting in a top-1 precision of 93.01%.

Questions?