Dynamic Unit Surgery for Deep Neural Network Compression and Acceleration

Minsam Kim, James T. Kwok
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{mkimac, jamesk}@cse.ust.hk

***Abstract*—Successful deep neural network models tend to possess millions of parameters. Reducing the size of such models by pruning parameters has recently earned significant interest from the research community, allowing more compact models with similar performance level. While pruning parameters usually result in large sparse weight tensors which cannot easily lead to proportional improvement in computational efficiency, pruning filters or entire units allow readily available off-the-shelf libraries to harness the benefit of smaller architecture. One of the most well-known aspects of network pruning is that the final retained performance can be improved by making the process of pruning more gradual. Most existing techniques smooth the process by repeating the technique (multi-pass) at increasing pruning ratios, or by applying the method in a layer-wise fashion. In this paper, we introduce Dynamic Unit Surgery (DUS) that smooths the process in a novel way by using decaying mask values, instead of multi-pass or layer-wise treatment. While multi-pass schemes entirely discard network components pruned at the early stage, DUS allows recovery of such components. We empirically show that DUS achieves competitive performance against existing state-of-the-art pruning techniques in multiple image classification tasks. In CIFAR10, we prune VGG16 network to use 5% of the parameters and 23% of FLOPs while achieving 6.65% error rate with no degradation from the original network. We also explore the method's application to transfer learning environment for fine-grained image classification and report its competitiveness against state-of-the-art baseline.**

***Index Terms*—Neural Network, Pruning, Network Compression, Network Acceleration, Deep Learning, Image Classification**

I. Introduction

Recently, deep neural networks have achieved significant improvements in a variety of domains such as computer vision, natural language processing, and reinforcement learning. With the ever-increasing size of datasets [9], models have become deeper and wider [14, 29], relying on millions or even billions of parameters. While such a trend has led to improved accuracy and overall performance, it has also created a bottleneck in a variety of industrial applications. For instance, deployment in mobile devices, or real-time applications with online learning may significantly suffer from large networks' intensively high requirements in memory, CPU, and energy.

A variety of approaches have been introduced to make deep neural networks more compact. Network pruning [11, 12, 13, 23, 25] seeks to reduce the number of parameters by removing units and/or connections, usually from pre-trained networks. Parameter sharing [7, 20] reduces the number of parameters from the very beginning, leveraging domain knowledge such as symmetry and invariance. Low-rank parameter factorization/regularization reduces the size of layer's parameter matrices / tensors by factorization of weights during post-processing [32] or regularization embedded during training [2]. Weight quantization approximates the original network weights by reducing the conventional floating-point representation to a fewer-bit [6], or even binary [8], representation. Finally, knowledge distillation transfers the knowledge in a large network (teacher) to a smaller network (student) [3, 5, 16, 28], by encouraging the student network to mimic the teacher network's outputs [16] or hidden representations [28]. Note that these categories of methods can be used together. For example, one may apply low-rank factorization to convolutional filters, and parameter pruning to the fully connected layers.

In this paper, we will focus on neural network pruning. Recent works have shown that removing unimportant network connections can significantly reduce the model size (by a factor of 10 or more) without significant performance deterioration [1, 10, 11, 12]. However, this may not lead to proportional speedup or memory-saving as the sparsified weight matrices require the same amount of floating point operations (FLOPs) unless specially designed sparse matrix operations are implemented [17, 22, 23]. Thus, pruning of entire units or filters can be more practical and desirable, with the potential effect of better regularization [23, 32].

However, filter[1] pruning methods tend to have more severe performance degradation than parameter pruning, due to the grouped structure of pruned parameters. Existing state-of-the-art methods [12, 23, 25] resort to repeating the method while increasing the portion of network components to be pruned, along with intermediate fine-tuning stages, sometimes layer by layer. Such an approach constrains the increase in network loss from pruned components, especially in deep networks, where accurate high-order approximation of components' saliency is too costly. However, this multi-pass, layer-wise approach inevitably makes the whole process very time-consuming.

In this paper, we propose a novel filter pruning technique called Dynamic Unit Surgery (DUS). It reduces performance

[1]In the sequel, we refer to filters, units, nodes interchangeably.

degradation and pruning / fine-tuning time via two mechanisms: (1) allowing pruned components to recover during fine-tuning, and (2) pruning each component in a continuously decaying manner, instead of abruptly dropping the component.

The rest of the paper is organized as follows. In Section II, we discuss early methods to prune neural networks and describe more recent techniques that extend to deep networks. We also highlight the problems we aim to tackle with the proposed methodology. In Section III, we describe DUS and how the two mechanisms interact with each other in detail. In Section IV, we conduct a series of experiments to empirically show DUS's competitiveness in retaining original network performance on various image classification benchmarks, in comparison to the most relevant state-of-the-art baseline. Finally, we conclude our work in Section V.

II. Related Works

A. Neural Network Pruning

Neural network pruning focuses on reducing neural network components while minimizing the increase in training error from pruning. One of the earliest techniques is optimal brain damage [21] (OBD). With a quadratic approximation of the error surface using a diagonal Hessian matrix, the elements are used to approximate increase in error when the corresponding connections are pruned from the network. Optimal brain surgeon (OBS) [13] relaxes the diagonal assumption to gain better performance. In each iteration, after the removal of less important connections, the remaining connections are optimally adjusted. Though OBS outperforms OBD, the use of a full Hessian is very computationally expensive for modern deep networks with millions of parameters.

To alleviate this problem, Han *et al.* [12] proposed to remove small-magnitude connections after training with weight decay. The method achieves a high compression ratio while maintaining original model performance. However, the heuristic choice of pruning induces a significant drop in performance after pruning (before fine-tuning), as the scheme implicitly assumes the error Hessian to be an identity matrix. This greedy approach prunes potentially important connections without considering the higher-order interactions among connections. The irrecoverably pruned connections can lead to worse performance after re-training [1], though a more granular combination of pruning and re-training in multiple iterations might alleviate the problem. Thus, a multi-pass scheme is often required, which repeats the lengthy processes of pruning and re-training.

B. Multi-Pass Scheme and Layer-Wise Treatment

The ideal way to prune parameters and components in an inference model would be to have an accurate and robust approximation of the change in learning objective due to the components' removal (saliency). However, the measurement can be challenging and costly in deep neural networks. The problem deteriorates as the size of pruned component gets larger due to higher-order interactions among components. Most existing techniques circumvent this by adopting (1) a multi-pass scheme and/or (2) layer-wise treatment. Instead of pruning the architecture once, a multi-pass scheme gradually increases the pruning ratio by including the fine-tuning stage in between updates of pruning ratio. Liu *et al.* [23] and Han *et al.* [12] report that the multi-pass versions of their proposed pruning methods outperform the single-pass counterparts. Experiments in [25] with AlexNet and ImageNet show that the number of training iterations between pruning filters one-by-one actually has larger impact than the saliency measurement criteria used to select filters. Li *et al.* [22] claims that iterative pruning and retraining is particularly helpful for prune-sensitive layers.

Instead of assessing filters/units on the entire network's objective, layer-wise treatment computes saliency based on layer-wise information. Dong *et al.* [10] uses the layer-wise application of optimal brain surgery on deep networks. He *et al.* [15] solves layer-wise lasso to mute channels while minimizing layer-wise output reconstruction error without involving nonlinearity. Similarly, instead of optimizing the layer-wise reconstruction error, ThiNet [24] approximates and minimizes the reconstruction error between consecutive layers.

C. Dynamic Pruning and Recovery of Parameters

Bringing the pruning and fine-tuning routine (multi-pass scheme) more granular to the level of each mini-batch update proved successful in dynamic network surgery (DNS) [11]. DNS tackles the problem by (1) mixing pruning and re-training at a more granular manner, and (2) allowing the network to recover (splice) pruned connections. By keeping a mask variable corresponding to each connection, DNS trains both the original network weights and mask variables by the following update rule, which allows pruned connections to absorb gradient from their corresponding mask variables:

$$w \leftarrow w - \alpha \frac{\partial L}{\partial (w \odot m)}, \quad (1)$$

$$m \leftarrow \text{top}n(w, n). \quad (2)$$

Equation (1) describes the simplified vanilla stochastic gradient descent (SGD) version of DNS's update rule for weight matrix/tensor w in a deep neural network, with the corresponding mask m (whose size is equal to that of w). Here, L is the network's optimization loss. Equation (2) shows how mask m is updated based on its corresponding weight parameter w. The function top$n$ is a generic operation that chooses n elements to be one (kept) in the binary mask m, while the others are set to zero (pruned), based on the saliency measure from parameter w. In Section III, we will generalize this update rule to prune different groups of network parameters (such as hidden units and convolutional filters).

D. Pruning Filters with Sparsity Regularization

Recently, the multiplicative scaling factor has become one of the most popular indicators for measuring saliency of a network component. Liu *et al.* [23] introduces network slimming (NS), which applies ℓ_1-regularization on the scaling parameter (γ in (3)) in batch normalization (BN) [18] to assess and prune filters. In BN, given the ith sample x_i in mini-batch $\mathcal{B}$, BN outputs

$$y_i \leftarrow \gamma \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta, \tag{3}$$

where $\mu_{\mathcal{B}}, \sigma_{\mathcal{B}}$ are the average and standard deviation, respectively, of the mini-batch. The normalized input is then treated with an affine transformation with scaling factor γ and bias factor β, for each dimension separately. For convolutional layers, each output feature map is normalized and transformed by the same set of parameters, whereas for fully connected layers, BN is applied per unit basis.

In network slimming (NS) [23], the original network is first trained with ℓ_1-regularization on the γ parameter, and then pruned by each filter's global ranking based on the corresponding γ magnitude. Finally, the pruned network is re-trained, but without ℓ_1-regularization on γ.

Ye *et al.* [33] adopts a similar optimization objective, and proposes usage of ISTA [4] instead of conventional SGD to allow application to any pre-trained network. They show that the plug-and-play application of the method requires multiple hyperparameter tuning stages, along with additional treatments such as re-scaling the filter weights and BN scaling parameters for smooth optimization procedure. Huang and Wang [17] further generalize applying ℓ_1-regularization on multiplicative scaling parameters to prune various components of deep networks, using accelerated proximal gradient.

III. Dynamic Unit Surgery

Based on the advantages of the multi-pass scheme and usage of BN's scaling parameters, we propose dynamic unit surgery (DUS) that dynamically turns on and off the units/filters with a novel way to make the pruning process more gradual. Allowing pruned components to be recovered during fine-tuning can compress multi-pass schemes at one-go, and adopting BN scaling parameters for this iterative saliency assessment keeps the computational overhead manageable without any modification in the original network architecture.

In Section III-A, we show how DUS leverages standard back-propagation procedure to choose components to be pruned and those to be recovered from the pruned state. Section III-B introduces a novel mechanism to preserve the pruned components' gradient information, which ultimately allows the pruned architecture to better retain the original network's performance.

A. Allowing Recovery of Pruned Components

When network components are pruned via the conventional multiplicative binary mask (with value one for kept components, and zero for pruned ones), all gradient information that passes through the pruned component becomes zero. This makes it difficult for standard back-propagation to recover the pruned component. To alleviate this, dynamic network surgery (DNS) [11] proposes to assume pruned/masked parameters to be un-pruned parameters with constant value of zero, thus allowing nonzero gradients via (1).

We generalize this idea to be applicable to the entire unit/filter, instead of individual weights. This is done by applying the mask on batch normalization (BN) scaling factor γ. First, BN normalization prevents the issue of redundant parameterization when adding multiplicative scaling factor, which can be harmful for optimization. Second, BN is widely used in most deep learning architectures, allowing broad application of the proposed method without requiring significant architectural modification for pruning. Third, as mentioned in Section II-D, BN scaling factors are useful heuristics to assess saliency of units/filters for pruning. [17, 23, 33].

Given layer l, and the multiplicative binary mask vector m for SGD, we substitute w in (1) with BN scaling factor γ, yielding the following update rule:

$$\gamma^{(l)} \leftarrow \gamma^{(l)} - \alpha \frac{\partial L}{\partial(\gamma^{(l)} \odot m^{(l)})}. \tag{4}$$

Both $\gamma^{(l)}$ and $m^{(l)}$ are vectors of the same size (equal to the number of units, filters, or any other groupings that correspond to each $\gamma^{(l)}$ factor in BN). Other than the scaling factors that have been effectively zero-ed out by the mask parameters, the process remains identical to standard back-propagation. Thus, (4) can be implemented very easily in modern machine learning libraries, by simply setting the back-propagated gradient for $\gamma^{(l)} \odot m^{(l)}$ to be the gradient of $\gamma^{(l)}$.

After each update of the scaling factors using update rule (4), mask value m can also be updated correspondingly based on (2). There exist different realizations of the generic operation top$n$ that uses approximated component saliency to determine mask values, or equivalently, components to prune. Liu *et al.* [23] and Guo *et al.* [11] prune components of which scaling factors are smaller than a pre-defined threshold. Ye *et al.* [33] and Huang and Wang [17] simply prune the components with scaling factors equal to zero, and the network's pruned portion is thus determined by the regularization hyperparameter. While the former approach allows user to easily specify target architecture / computational requirement, the latter approach can be advantageous in that it does not require additional fine-tuning. All these options are viable for DUS's mask update rule after each back-propagation iteration. In this work, we use the per-layer pruning ratio $r^{(l)}$ to prune components with smaller scaling factors as:

$$m^{(l)} \leftarrow \mathrm{top}n(\gamma^{(l)}, r^{(l)}d). \tag{5}$$

B. Decaying Mask

With update equation (4), the gradient information still cannot flow into the pruned components (with $m = 0$) due to the ReLU activation following BN. Specifically, for feature map f of which the corresponding mask value m_f is zero, the update gradient still remains zero:

$$\frac{\partial}{\partial \gamma} \left[ReLU(BN(x_f) \times m_f) \right] \bigg|_{m_f=0} = 0. \tag{6}$$

Hence, instead of using a zero mask value for those components with low saliency measure γ, we propose to adopt a decaying nonzero mask value k. For instance, using an initial $k = 1$ reduces the multiplicative binary mask m to be trivial identity scaling factor, keeping the model un-pruned. Beginning from this state, DUS gradually decays k to zero during fine-tuning in order to obtain a pruned model in the end. In the following, we adopt an exponentially decaying schedule for k. At iteration i,

$$k_i = k_{i-1}\eta = k_0\eta^i. \tag{7}$$

We vary the per-update decay rate η in the range $[0.99, 0.99999]$ for different data sets, so that the value is not too far from zero near the end of training. This way of "leaking" information from an otherwise pruned component not only mitigates the issue of zero gradient in (6), but also allows the entire pruning procedure to be more smooth. The equations below show DUS's update rules for decaying mask $\widetilde{m}^{(l)}$ and BN scaling factor $\gamma^{(l)}$ for layer l. The whole procedure is shown in Algorithm 1.

$$\widetilde{m}_j^{(l)} = \begin{cases} k & \text{if } m_j^{(l)} = 0 \\ 1 & \text{if } m_j^{(l)} = 1 \end{cases}, \tag{8}$$

$$\gamma^{(l)} \leftarrow \gamma^{(l)} - \alpha\frac{\partial L}{\partial \gamma^{(l)}}. \tag{9}$$

Algorithm 1 Dynamic unit surgery (DUS).

```
1: Input: (pretrained) network, compression ratio $r$, mask decay rate $\eta$.
2: Output: A pruned model.
3: for each update iteration $i$ do
4:     for each layer $l$ do
5:         update $k_i$ using (7);
6:         if $k_i < 10^{-5}$ then
7:             $k_i \leftarrow 0$;
8:         end if
9:         update $\widetilde{m}^{(l)}$ using (5), (8), and $\eta$;
10:        apply standard forward pass with each layer $l$: $y^{(l)} \leftarrow \widetilde{m}^{(l)} \odot (\gamma\frac{x-\mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2+\epsilon}} + \beta)$;
11:    end for
12:    for each layer $j$ do
13:        apply standard back-propagation (9);
14:    end for
15: end for
```

As the value of k decays, it (i) gradually splits the activation distributions of masked ($m_j^{(l)} = k$) and unmasked ($m_j^{(l)} = 1$) components by a larger degree before the former eventually reaching to zero, and (ii) simultaneously decays the scale of back-propagated gradients through masked components, while (iii) preventing zero gradients from equation (6). When k converges to zero, no gradient will pass through the pruned component. However, the pruned component still has chance to be recovered via (5). Thus, one might decide not to allow any leakage ($k_0 = 0$) in the first place, or use ($k_0 = 1$) for a more gradual pruning procedure. In the experiments, we use an initially fixed $k_0 = 1$ to emphasize the impact of the proposed method.

As an illustration, Figure 1 shows heatmaps of a convolutional layer's BN γ and $\widetilde{m}$ values along the training process. Each row corresponds to one of the 256 filters. Except for the mask entries of value 1, the other entries share the same value of k which is shared across the layer and decaying exponentially per mini-batch update. As training proceeds, components to be pruned are gradually fixed. To our knowledge, this is a novel approach to smooth network pruning procedure by introducing a new dimension of continuity, while most existing methods resort to multi-pass schemes or layer-wise pruning to gradually increase the *portion* of pruned components as discussed in the previous Section.

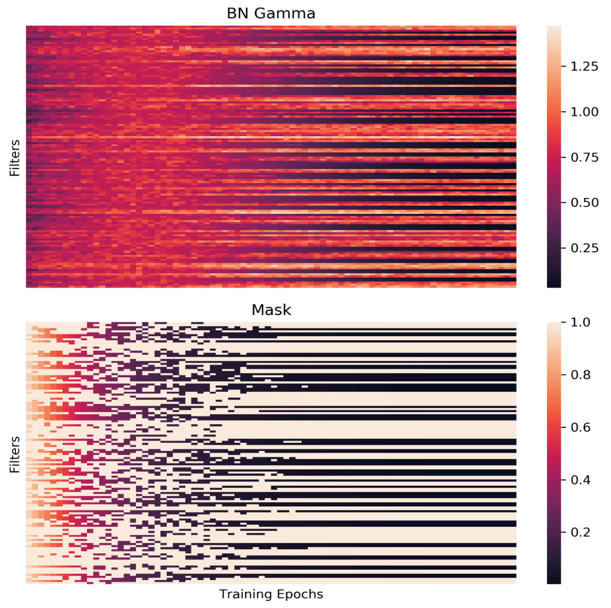


Fig. 1: Evolution of the BN scaling factors (top) and mask (bottom).

IV. Experiments

In this Section, we show how DUS mitigates the suboptimality of its greedy/static counterpart on multiple publicly available image classification benchmarks in both fully connected feed-forward networks and (fully) convolutional neural networks. As DUS and network slimming (NS) share the important feature of using Batch Normalization scaling factors as network component saliency heuristics, and NS shows state-of-the-art pruning performances in these benchmarks, we primarily focus on comparing DUS with NS.

A. MNIST Pixel Selection with DUS

The first experiment is on MNIST digit classification. As in [20], we train a multilayer perceptron (MLP), with two ReLU hidden layers of sizes 300 and 100, using SGD (with learning rate 0.1 and Nesterov momentum 0.9). To compare DUS and NS for pixel selection, we also add a BN layer after the input layer. The entire network architecture consists of

784-300-100-10 units, corresponding to 784 input pixels and 10 output labels. We then try pruning 80% and 90% of the 784 input pixels based on (1) NS and (2) DUS.

Figure 2 visualizes the difference between greedy choice of units and dynamically optimized choice of units. While NS's chosen (brighter) units with high BN scaling factors are restricted in the center, DUS chooses units that are more widespread to reduce overlapping and correlated information from nearby pixels, similar to subsampling. For NS, changing the ℓ_1 regularization parameter on BN scaling factors does not change the overall shape of scattering, which indicates that DUS' dynamic approach is essential to tackle suboptimality.

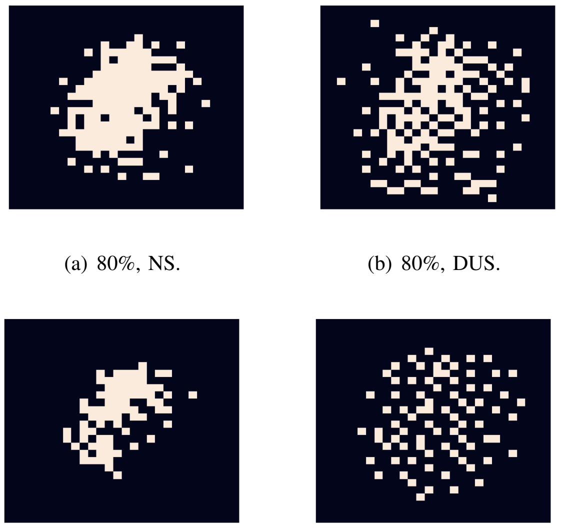

(a) 80%, NS. (b) 80%, DUS. (c) 90%, NS. (d) 90%, DUS.

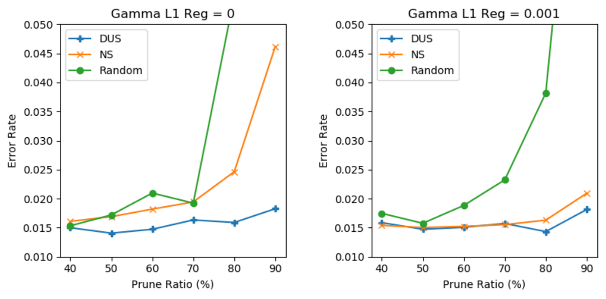
Fig. 2: Pixel selection on MNIST data.

Figure 3 shows the MLP's classification error rate on the MNIST dataset with different input pixel pruning ratios (from 40% to 90%). In Figure 3(a), without ℓ_1 regularization on the BN scaling factors, we observe that DUS achieves below 2% error rate even after excluding 90% of the input pixels, while NS suffers substantial degradation. As mentioned in Section III-A, this emphasizes that DUS does not require special modification (i.e., ℓ_1 regularization on BN scaling factors) in networks' standard training process to retain original architecture's performance. Figure 3(b) shows results from the best λ parameter setting ($\lambda = 0.001$) for NS, which corresponds to Figures 2(c) and 2(d). Based on better distributed input pixel selection, DUS still outperforms NS when the pruning ratio is higher.

B. Pruning VGG-net for CIFAR Datasets

In this Section, we experiment with larger and deeper networks, and empirically show that this performance gap widens, as a one-off approximation of network component saliency inevitably becomes less accurate in larger models.

We show the efficacy of DUS on CIFAR10/CIFAR100 datasets by comparing against its static version Network



(a) $\lambda = 0$. (b) $\lambda = 0.001$.

Fig. 3: Performance on MNIST.

Slimming (NS). CIFAR10/CIFAR100 datasets are small 32-by-32 color images with 10 and 100 classes, respectively. We replicate the setup in [23] with dataset under the PyTorch [27] framework. We first train CIFAR10/CIFAR100 datasets on a VGG-like fully convolutional network with 16 convolutional layers. The baseline model before pruning achieves a test error rate of 6.69% (resp. 27.92%) on CIFAR10 (resp. CIFAR100), with BN scaling factor's sparsity regularization parameter λ set to 0.0001. The parameter value of 0.0001 is the same as the grid search result in [23]. The initial learning rate is 0.1, decaying at the rate of 0.1 at 50% and 75% of the 160 training epochs. The gradient is accelerated by Nesterov momentum (with a value of 0.9).

If only the global pruning ratio (instead of layer-specific pruning ratio) is specified, the resulting pruned architecture from DUS will likely have different numbers of filters in different layers from the output of NS due to dynamic nature of DUS. As early-stage convolutional layers in VGG-like networks have a smaller number of filters and have larger feature map size, they contribute more to the number of FLOPs and less to the number of parameters, than the later stage convolutional layers. Thus, globally setting a fixed percentage of filters to prune would result in comparing two networks with different architectures and computational resources. To make a fair performance comparison between the two methods under identical levels of parameter/FLOP saving, we fix the final pruned architecture for the two methods. In this way, the result is not affected by different cross-layer normalization schemes for the saliency measure. For instance, Molchanov [25] normalizes layer-wise saliency measure by layer-wise ℓ_2-norm of the saliency vector for cross-layer comparison. As Liu [23] does not use cross-layer normalization scheme, globally pruning 80% of the filters results in pruning an entire layer, disconnecting the input from the target. This shortcoming is expected since each layer has different distributions of BN scaling factors in general. To mitigate this and focus on improvement from dynamic component selection under equivalent computational requirement, we use the same target architecture for both DUS and NS.

We test two types of pruned architectures: (a) keep 60% of the filters each layer after pruning, and (b) NS's multi-pass pruning architecture. Keeping an equivalent percentage of filters within a layer (a) is commonly adopted to evaluate saliency approximation methods, and (b)'s architecture results after applying NS for five times, which uses 5%, 23% of the original network's parameters and FLOPs, respectively. For each of the 16 convolutional layers, it keeps 34%, 97%, 65%, 93%, 75%, 66%, 33%, 16%, 6%, 6%, 6%, 6%, 6%, 6%, 6%, 7% of the original layers' filters. Regarding architecture (b), we aim to use DUS to recover original architecture performance within a single pass, as opposed to NS's five passes. Following [23], the single pass re-training session uses the same setup as in original training, without sparsity regularization on the γ parameters.

Table I shows the performance before pruning and after pruning/re-training. We observe that DUS consistently outperforms its static counterpart. Notably, on CIFAR10, DUS achieves more than four times FLOPs reduction without any performance degradation, in a single pass. DUS also consistently outperforms NS on CIFAR100 in both pruned architectures tested, suffering no performance drop for architecture (b) which achieves almost three times FLOPs reduction. Furthermore, DUS outperforms NS by 4.5%. This suggests that there exists larger suboptimality from static pruning methods when classification becomes more fine-grained and that DUS's postponed smooth pruning alleviates the performance loss.

C. Pruning ResNet for CIFAR Datasets

In this experiment, we prune residual network [14] with the proposed DUS. As in [23], we use the pre-activation ResNet with bottleneck structure [14], with 164 layers and ℓ_1-regularization parameter set to 10^{-5} (which is from grid search in [23] under equivalent setup). The other setups are the same as in Section IV-B. We also adopt the approach in Section IV-B to prune all layers with the corresponding BN layers. In [30], experimental results showed that the residual network possesses an ensemble-like feature which makes it relatively robust to removal/permutation of residual blocks (collections of paths) with only modest performance degradation. Based on the observation, in this Section, we show that DUS can retain performance even after pruning 50% of the filters in every layer, leading to 75% of parameters/FLOPs saving.

Table II show that DUS outperforms NS with a more significant gap on the ResNet architecture. As the ResNet has a significantly larger number of layers, any instant saliency approximation of network component will have larger approximation error in ResNet. DUS tackles this issue in two ways: (i) update the saliency (γ) after each update, and (ii) gradually increases the impact of pruning via a decaying mask.

Figure 4 shows how the choice of component changes per each of the 164 layers, along with the training process of the first 120 epochs on the CIFAR10 dataset. The ResNet-164 consists of three types of residual blocks, represented by Block1, Block2, and Block3 on the y-axis. Each dot / entry of the heatmap represents the percentage of the layer's components of which status changed in the mask. For instance, if half of the pruned components in the previous epoch are now recovered and vice versa, it is represented with color showing 50%. To emphasize the state of no change in choice of pruned component, 0% entries are replaced by white color.

From figure 4, we note that most of the mask elements are determined from the first block to the last block, in a feed-forward fashion. Brighter colors in layers of block groups 2 and 3 suggest they go through more updates in mask values. However, we also observe that there exist a minority of mask elements in layers of block groups 1 and 2, are fixed after masks in block group 3 are fully converged. This indicates that pruning choices made closer to the output layer do have an impact on previous layers, made possible by the standard back-propagation process. It is also notable that the horizontal stripe pattern in the heatmap (dark row repeating for every three layers) corresponds to the first layers of each residual blocks. They tend to modify masks less frequently and converge faster, due to the reduced number of feature maps across the first layer.

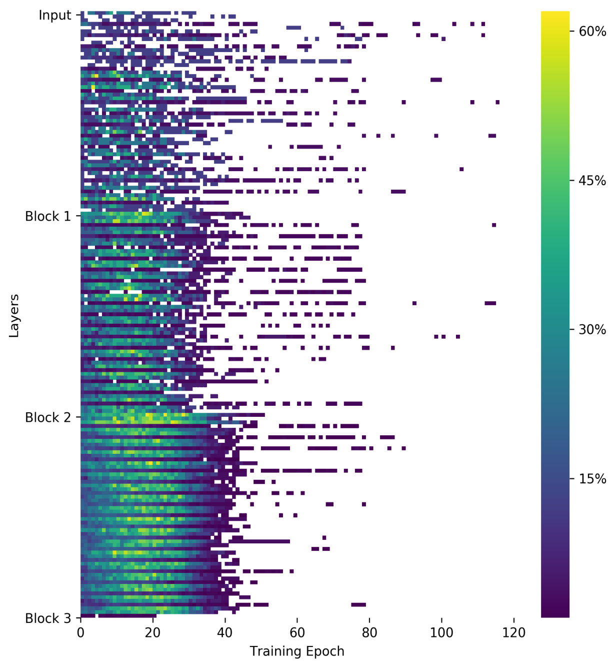


Fig. 4: Switching and convergence of the ResNet-164 mask on CIFAR10.

D. Feature Transfer for Fine-grained Image Classification

Neural network pruning can be particularly useful under the transfer learning setting. First, not all features learned on source domain A are necessarily helpful for target domain B. Second, target domain B can have a significantly smaller dataset size or complexity than target domain A, where the reduced complexity of a pruned architecture can lead to regularization. Third, many transfer learning applications are adopted in the industry, where obtaining an efficient architecture (via pruning) can be critical for the applications' scalability.

TABLE I: DUS and NS on the CIFAR datasets.

	architecture	pruning method	removed parameter (%)	saved FLOPs (%)	test error (%)
CIFAR-10	original VGG16 architecture		0	0	6.69
	(a) keep 60% per layer	DUS	64.0	64.2	6.96
		NS	64.0	64.2	7.21
	(b) NS multi-pass	DUS	95.6	77.2	6.54
		NS	95.6	77.2	7.19
CIFAR-100	original VGG16 architecture		0	0	27.92
	(a) keep 60% per layer	DUS	63.9	64.2	27.61
		NS	63.9	64.2	32.18
	(b) NS multi-pass	DUS	95.4	77.2	31.15
		NS	95.4	77.2	33.11

TABLE II: Pruning ResNets on the CIFAR datasets.

	network	setup	test error
CIFAR-10	ResNet-164	unpruned, trained with ℓ_1	5.17%
		pruned 50%, DUS	5.53%
		Pruned 50%, NS	6.52%
CIFAR-100	ResNet-164	unpruned, trained with L1	23.53%
		pruned 50%, DUS	23.95%
		pruned 50%, NS	26.92%

TABLE III: DUS for feature transfer.

	network	setup	test error
Flowers-102	VGG-D	unpruned, fine-tuned with ℓ_1	5.07%
		pruned 30%, DUS	5.58%
		pruned 30%, NS	6.01%
Birds-200	VGG-D	unpruned, fine-tuned with ℓ_1	30.98%
		pruned 30%, DUS	33.34%
		pruned 30%, NS	36.19%
Cars-196	VGG-A	unpruned, fine-tuned with ℓ_1	13.8%
		pruned 30%, DUS	12.24%
		pruned 30%, NS	12.46%

In this Section, we test DUS's feature selection and network pruning capability in the context of feature transfer for fine-grained image classification. We use the VGG network pre-trained on ImageNet dataset to fine-tune on Caltech-UCSD-Birds 200 (Birds-200) [31], Oxford Flowers 102 (Flowers-102) [26], and Stanford Cars 196 (Cars-196) [19] datasets. Birds-200 dataset contains 3000 training images and 3033 test images for 200 classes of birds. Flowers-102 dataset has 2040 training images and 6129 test images for 102 species of flowers. Finally, Cars-196 has 8144 and 8041 training and test images for 196 classes of cars.

Table III shows the test accuracies of (1) the original VGG network after fine-tuning, (2) network pruned and re-trained with DUS, and (3) network pruned and re-trained with NS. We use ImageNet pre-trained VGG-D and VGG-A networks, both of which are directly available from the PyTorch library [27]. The initial fine-tuning before pruning uses standard SGD with Nesterov momentum 0.9 and weight decay 10^{-4}. For Flowers-102/Cars-196, we use learning rate 10^{-2}, and for Birds-200, 10^{-3}. For proper comparison with NS, we incorporate ℓ_1-sparsity regularization on BN's γ parameters with the original paper's suggested hyperparameter value of 10^{-4}.

For fine-tuning after pruning, we use 1/10th of the learning rate used before pruning. Using a smaller learning rate yields better result for both pruning methods and all datasets. For simplicity, we prune a fixed proportion (30%) of the filters and units from all layers, resulting in a 51% reduction of parameters / FLOPs. We also note that we achieve a new state-of-the-art baseline result of 5.07% error rate on Flowers-102 by adding random rotation of images during training. The results show that DUS better retains the original network performance after pruning and re-training for all three fine-grained image classification tasks. On Cars-196, pruning the original ImageNet pre-trained model leads to significant regularization effect which highlights the motivation for using pruned architectures for feature transfer.

We also note that merely using higher ℓ_1-regularization hyperparameter value for more aggressive pruning leads to optimization difficulty for the original unpruned network, yielding a much lower testing accuracy. Consequently, NS prunes filters with a lot of nonzero scaling parameters remaining, instead of relying on the regularization hyperparameter to completely prune filters. In order to solely rely on regularization to prune parameters, Huang and Wang [33] and Ye *et al.* [17] adopt layer-wise hyperparameter and separately modified optimization schemes for the scaling parameters.

V. Conclusion

In this paper, we proposed Dynamic Unit Surgery to prune neural network components for model compression and acceleration. The proposed method shows consistent improvement over its greedy/static counterpart, highlighting the advantages from dynamic component choice and smoothly decaying continuous-valued mask in multiple image classification benchmarks and network architectures. In the future, we would like to compare different heuristics for dynamic component choice in DUS framework. Furthermore, applying trust-region method would allow masks to decay at independent rate, while avoiding significant performance degradation. Moreover, application of the technique to other computer vision tasks like semantic segmentation and object detection would improve network performances on edge devices.

References

[1] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Neural Information Processing Systems*, pages 3180–3189, 2017.

[2] J. M. Alvarez and M. Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867. 2017.

[3] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Neural Information Processing Systems*, pages 2654–2662, 2014.

[4] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[5] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. Preprint arXiv:1511.05641, 2015.

[6] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[7] T. Cohen and M. Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning*, pages 2990–2999, 2016.

[8] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1. Preprint arXiv:1602.02830, 2016.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *International Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[10] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Neural Information Processing Systems*, pages 4860–4874, 2017.

[11] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient DNNs. In *Neural Information Processing Systems*, pages 1379–1387. 2016.

[12] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Neural Information Processing Systems*, pages 1135–1143. 2015.

[13] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Neural Information Processing Systems*, pages 164–171, 1993.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645, 2016.

[15] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision*, 2017.

[16] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. Preprint arXiv:1503.02531, 2015.

[17] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. Preprint arXiv:1707.01213, 2017.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[19] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3D object representations for fine-grained categorization. In *International Workshop on 3D Representation and Recognition*, 2013.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Neural Information Processing Systems*, pages 598–605. 1990.

[22] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. Preprint abs/1608.08710, 2016.

[23] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *International Conference on Computer Vision*, pages 2755–2763, 2017.

[24] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. Preprint arXiv:1707.06342, 2017.

[25] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. Preprint arXiv:1611.06440, 2016.

[26] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008.

[27] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop*, 2017.

[28] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. Preprint arXiv:1412.6550, 2014.

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. Preprint arXiv:1409.1556, 2014.

[30] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Neural Information Processing Systems*, pages 550–558, 2016.

[31] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[32] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Neural Information Processing Systems*, pages 2074–2082, 2016.

[33] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. Preprint arXiv:1802.00124, 2018.