PruneTrain: Gradual Structured Pruning from Scratch for Faster Neural Network Training

Sangkug Lym [1] **Esha Choukse** [1] **Siavash Zangeneh** [1] **Wei Wen** [2] **Mattan Erez** [1] **Sujay Sanghavi** [1]

Abstract

Model pruning is a popular mechanism to make a network more efficient for inference. In this paper, we explore the use of pruning to also make the *training* of such neural networks more efficient. Unlike all prior model pruning methods that sparsify a pre-trained model and then prune it, we train the network from scratch, while gradually and structurally pruning parameters during the training. We build on our key observations: 1) once parameters are sparsified via regularization, they rarely re-appear in later steps, and 2) setting the appropriate regularization penalty at the beginning of training effectively converges the loss. We train ResNet and VGG networks on CIFAR10/100 and ImageNet datasets from scratch, and achieve 30-50% improvement in training FLOPs and 20-30% improvement in measured training time on modern GPUs.

1. Introduction

Motivation: The computational cost of pruning: Large-scale neural networks often provide state-of-the-art accuracy for tasks like image classification; however the memory and computational cost of inference can be quite high. Model pruning involves reducing the number of parameters in a dense fixed network (leading to lower memory and inference cost) while retaining the accuracy of the original model; it is already widely used and an active area of research (Wen et al., 2016; 2017; Feng & Darrell, 2015; He et al., 2018).

Prior pruning approaches start with a pre-trained dense network model and proceed to zero out parameters using trial-and-error or group-lasso regularization learning (Li et al., 2016; Molchanov et al., 2016; Wen et al., 2016; He et al., 2017). This process involves many complete training rounds and includes additional hyper parameters. This significantly increases the overall training time compared to a dense model, which is made even worse by fine-tuning training rounds after pruning is complete. Our **goal** is to develop a new way to reduce the training effort required in effectively pruning a large dense baseline neural network.

We propose **PruneTrain**, a *one-shot training mechanism that gradually prunes a model while training it from scratch*. Gradual pruning speeds up training because computation costs decrease as the model is pruned further and further during training. We demonstrate that this speedup and gradual pruning does not come at the cost of either compression rates or inference accuracy. The underlying idea is as follows: consider pruning as done via group lasso regularization (Yuan & Lin, 2006). We apply group lasso regularization from the beginning including to the untrained model and set the lasso penalty to constrain the regularization loss to a small portion of the total loss. This reduces the weights of regularized channel parameters to near zero with minimal impact on accuracy. We then periodically remove such channels, which sparsifies the network. Instead of keeping this sparse network as is, we reconfigure it into a new dense one with fewer parameters. This significantly reduces FLOPs and memory accesses of training without adding overheads for handling sparse data structures.

Additionally, we show that most of the unimportant parameters can be identified and removed in the early stages of training, thereby resulting in high savings in terms of FLOPs during the rest of the training. This algorithm is justified by our empirical **observation** that once (groups of) weights are regularized to zero at some stage in training, they extremely rarely revive to become non-zero in the final version, and if they do, those weights are small and have little impact. As a result, PruneTrain can train a pruned network in even less time than the original dense training.

We evaluate PruneTrain on two main metrics: the FLOPs reduced and the actual measured end-to-end training time on a GPU. Evaluating end-to-end time is particularly important because some pruning techniques require sparse data structures whose execution on modern processors is less efficient. Specifically, we introduce an optimization of PruneTrain for modern CNNs (Convolutional Neural Networks) with short-

[1]The University of Texas at Austin, Austin, Texas, USA [2]Duke University, Durham, North Carolina, USA. Correspondence to: Sangkug Lym <sklym@utexas.edu>, Mattan Erez <mattan.erez@utexas.edu>.

Preprint, work in process

cut connections. This *channel union* algorithm prunes only the intersection of the sparsified channels of the layers in the same stage. Channel union avoids sparse channel indexing overheads with only a minor increase in FLOPs (3%) and accelerates the layer-wise performance by 1.9X; if indexing is used, training is slowed down rather than accelerated.

We demonstrate the effectiveness of our work on several network architectures, and on different datasets for an image classification problem. Our experiment results show that PruneTrain can speed up training for network pruning by 50% in FLOPs and 19% in the measured end-to-end time of ResNet50 on CIFAR10 (also, 28% and 18% for ResNet50 on ImageNet). The pruned network shows minor accuracy drop compared to the dense baseline, while the training speedup is proportional to the degree of model compression.

We summarize our contributions as follows.

- We show that the network structure components that are sparsified by group lasso regularization rarely revive during the rest of the training process. Thus removing them early does not impact the net prediction accuracy.
- We accelerate network model training by training the model from the scratch and gradually pruning the unimportant channels and layers during training. We also introduce a way to set the regularization penalty coefficient to apply model pruning from the beginning of training.
- We demonstrate that overlapping regularization groups of input and output channels naturally leads to removing unimportant layers for CNNs with multi-branch modules.
- We propose channel union, a practical sparse network reconfiguration method for CNNs with short-cut connections. Channel union removes the tensor reshaping overhead, leading to training and inference speedups on GPU accelerators.

2. Related Work

Model pruning has been studied primarily for CNNs, to make their models more compact and their inference fast. Prior pruning methods compress a network model by removing small-valued parameters with a fine-tuning process to minimize accuracy loss (Han et al., 2015a;b). Pruning algorithms can be unstructured or structured. Unstructured pruning can maximize model-size reduction but requires fine-grained indexing with irregular data access patterns. Such accesses and extra index operations lead to poor performance on deep learning accelerators with vector or matrix computing units despite reducing the number of parameters and FLOPs (Han et al., 2017; Yu et al., 2017; Anwar et al., 2017). Structured-pruning algorithms reduce fine-grained indexing and sparsity to better match the needs of hardware and thus effectively realize performance gains.

Trial and Error Based Structured Model Pruning. One approach to structured pruning is to start with a pre-trained dense model and then attempt to remove parameters in a structured manner, generally removing channels rather than individual weights (He et al., 2017; Hu et al., 2016; Molchanov et al., 2016). Unimportant channels are removed based on the value of their parameters or hints derived from lasso regression (Tibshirani, 1996). The removed channels are rolled back if accuracy is severely affected. *He et al.* propose a *reinforcement learning* based mechanism to automate this channel pruning process (He et al., 2018). However, the search space of such a trial-and-error based model pruning substantially increases with the complexity of the network architecture, which can substantially increase training time.

Learning Network Structures By Regression. An alternative mechanism to trial-an-error pruning uses *group lasso regularization* (Yuan & Lin, 2006) to learn either the number of parameters (Alvarez & Salzmann, 2016), or the network structure from a pre-trained model (Wen et al., 2016; 2017; Feng & Darrell, 2015). Group lasso based structured pruning regularizes the parameters of a channel, a layer, or a filter to systematically sparsify each of these network components. This approach can provide substantial improvements and *Wen et. al* demonstrate pruning $>$50% of the channels or 30% of the layers from some modern CNNs in using SSL (Structured Sparsity Learning) (Wen et al., 2016). Their results show a 3X layer-wise inference speedup on TITAN Xp GPU compared do a dense baseline when pruning *AlexNet* (Krizhevsky et al., 2012) on ImageNet (Deng et al., 2009), while degrading accuracy by only 1%. We further observe that learning the compressed structure by training with regularization yields a better pruning rate for a given prediction accuracy compared to the trial-and-error based model pruning approaches.

Unsurprisingly, learning the network structure during training significantly increases the training time. First, the network is first trained without the sparsification, followed by another additional training for a sparse model. This requires roughly twice the time as baseline dense training. Second, after the sparse network is learned, prior work uses additional epochs for fine-tuning the compressed network to improve accuracy. Third, additional hyper-parameters, like the lasso penalty coefficient, lead to a larger hyper-parameter search space. Since training time was not a major concern for prior work, the cost of training or how it can be sped up has not been discussed.

3. PruneTrain

Training acceleration is an important challenge as the training resources are costly and limited. Furthermore, reducing the training time enables additional exploration in the hyper-parameter space to further improve the net accuracy. Prior work proposing pruning requires the network to be trained without any sparsification first, followed by an equally long sparsifying process. Throughout this process, the network remains unpruned. Instead of using such a resource-heavy approach, PruneTrain focuses on accelerating training, by using group lasso regression right from the beginning, and by gradually pruning the sparsified model during training.

We first describe our training mechanism along with the underlying motivation, followed by a discussion of practical methods to deploy PruneTrain on to the modern GPUs.

3.1. Pruning a Network from Scratch

Group lasso regularization is a way to structurally sparsify a network, such that the sparse sections of the network can be easily pruned out, leaving behind a dense network. Prior work has used this technique to induce sparsity in a pre-trained model with the network reconfigured only after the whole regularization process completes, resulting in a smaller network for faster inference but much longer training time.

Each convolution layer l of a CNN contains four-dimensional learning weights $\boldsymbol{W}_l \in \mathbb{R}^{C_l \times K_l \times H_l \times W_l}$, where C_l, K_l, H_l, and W_l indicate the number of input channels, output channels, and filter height and width. To sparsify these model parameters during training, we use group lasso regularization with the following overall optimization function:

$$\min_{\boldsymbol{W}} \frac{1}{N} \sum_{i=1}^{N} l(y_i, f(x_i, \boldsymbol{W})) + \lambda_g \cdot \sum_{g=1}^{G} ||\boldsymbol{W}_g||_2 \tag{1}$$

The first term represents the baseline cross entropy loss, the second term is responsible for the structural sparsification using group lasso, f is the network's prediction on the input x_i, $\boldsymbol{W}$ are the network weights, l is the cross entropy loss function between the prediction and ground truth y_i, and N is the mini-batch size. The group lasso regularizer is added to this baseline loss. It is proportional to the sum of the L_2-$norm$ of the weights in each group, where the groups represent the granularity at which we want to induce the structural sparsity. The intensity of group lasso regularization is proportional to the tunable coefficient λ_g.

Group-lasso design. We group the weights of each input and output channel of each layer for lasso regularization. Then, we formulate the regularizer in Eq. 1 as:

$$\lambda_g \cdot \sum_{l=1}^{L} \left(\sum_{c_l=1}^{C_l} ||\boldsymbol{W}_{c_l,:,:,:}||_2 + \sum_{k_l=1}^{K_l} ||\boldsymbol{W}_{:,k_l,:,:}||_2 \right) \tag{2}$$

L indicates the number of layers in the network. (Simon & Tibshirani, 2012; Alvarez & Salzmann, 2016) propose to penalize each group proportionally to its number of weights in order to maintain similar regularization intensity across all groups. We choose to use a single global penalty coefficient instead because this emphasizes reducing computation over reducing model size. Layers with a larger number of weights tend to have smaller features and therefore require fewer operations per channel. In other words, we aim to prune channels with large features and thus keep the penalty coefficient fixed. This is equivalent to a larger effective coefficient for large features because the operations required for each layer in recent CNNs is roughly fixed. We do not regularize the input channels of the first convolution layer and the output channel of the last fully-connected layer, because the input data and output logits of a network have logical significance and should always be dense.

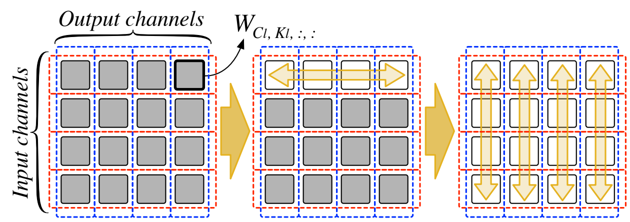


Fig. 1. Group lasso regularization structure of a convolution layer: Weights of a filter (each square box) affect the regularization of both input and output channels (red and blue dotted boxes). The white filters are regularized to zero.

(Wen et al., 2016) also propose to use layer-wise lasso groups for regularization in order to remove layers of CNNs with short-cut connections. However, we find that because there is an overlap in the weights between input and output channel lasso groups (Fig. 1a), unimportant layers are eventually removed even without additional layer-wise group regularization. As an example, when an input channel becomes sparse (Fig. 1b), it gradually sparsifies all the intersecting output channels to zero (c), eventually leading to the entire layer becoming zero.

To regularize the model from the beginning of training, the penalty coefficient λ_g should be carefully set to avoid training divergence. We define the ratio of regularization loss out of the total loss as *regularization penalty ratio*:

$$Reg.\ penalty\ ratio = \frac{\lambda \sum_g^G ||\boldsymbol{W}_{g,:}||}{l(y_i, f(x_i, \boldsymbol{W})) + \lambda \sum_g^G ||\boldsymbol{W}_{g,:}||} \tag{3}$$

The regularization loss coefficient is computed before training using the normally-distributed random initial weights, and is kept constant during the training process Based on our observation, setting the regularization penalty ratio as small as $< 20\%$ leads to high compression with minor accuracy loss for all networks and datasets we evaluate.

3.2. Motivation for Pruning During Training

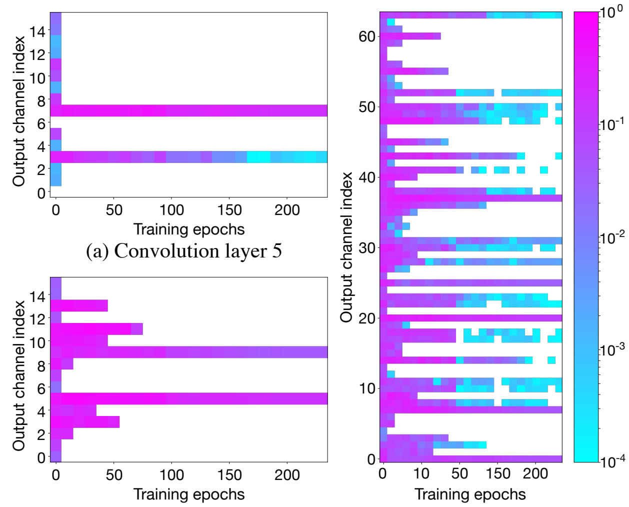
In addition to using a pre-trained model, prior work using group lasso regularization (SSL) maintains the original unpruned network architecture until the end of pruning (Wen et al., 2016; 2017). We observe that dynamically pruning the network architecture while learning its structure can significantly accelerate the training process. Fig. 3(a) shows the normalized computation FLOPs per ResNet50 training iteration on CIFAR10 when the network is gradually pruned using group-lasso from scratch. Each line indicates the result using a different regularization ratio. Regardless of the pruning intensity, the majority of FLOPs are pruned during early epochs and the pruning rate gradually saturates. This earlier pruning of the network leads to a larger reduction in the total FLOPs during training. Another interesting observation is that even though the pruning slows down, it still goes on, even until 300 epochs, as shown in Fig. 3(b).

A potential issue with pruning while training is that it prohibits sparsified parameters from "reviving" and becoming non-zero as training proceeds. This can happen as gradients flow back from the last FC layer and potentially increase the value of previously-sparsified parameters. However, we observe that already-regularized input and output channels of convolution layers are likely to suppress such revived weights from ever becoming large. This can be inferred from the equation of the local weight gradients for a layer l:

$$\frac{\partial L}{\partial \boldsymbol{W}_l} = \boldsymbol{z}_{l-1} \circledast \frac{\partial L}{\partial \boldsymbol{x}_l}^T \tag{4}$$

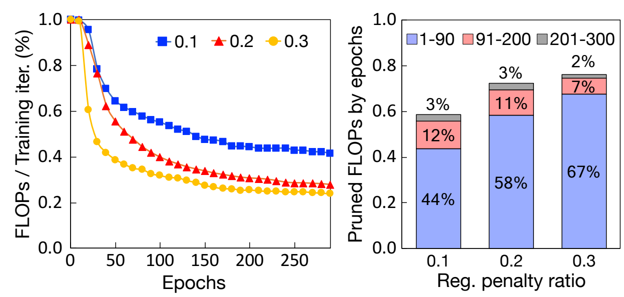
Here, $\circledast$ is convolution operator, and z_{l-1} and $\frac{\partial L}{\partial \boldsymbol{x}_l}$ are the input activations and the upstream gradients from the subsequent normalization layer. If a channel is regularized, its convolution outputs x_{l-1} are zeroed and they remain zero after normalization and activation layers, meaning that z_{l-1} is zero. Also, if an input channel of the subsequent convolution layer (l) is regularized, the upstream gradients of this input channel are forced to be small. Thus, the gradients after passing the normalization layer $\frac{\partial L}{\partial x_l}$ are also kept small by the gradient equation from (Ioffe & Szegedy, 2015). Therefore, using Eq. 4, the gradients of regularized weights are forced to remain small and often zero, effectively restricting the previously regularized weights from reviving.

This behavior is apparent in Fig. 2 that shows the output channel sparsity of three layers of ResNet50 (He et al., 2016a) trained on CIFAR10 across training epochs. Each point in the graph is the absolute maximum value among the parameters of each output channel. Convolution layers 5 and 6 are typical and none of the weights from the regularized output channels revive. Although some parameters in output channels of convolution layer 7 revive, their weight values are still very small and near the threshold, indicating very small contribution to the prediction accuracy of the final learned model. Similar patterns are observed in all convolution layers of different ResNet and VGG networks on CIFAR10 and CIFAR100, with the vast majority of layers exhibiting no revived parameters.



(a) Convolution layer 5
(b) Convolution layer 6
(c) Convolution layer 7

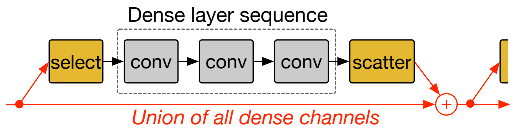
Fig. 2. The absolute maximum weight value of each output channel over training epochs. Three convolution layers belong to one residual path of ResNet50 trained on CIFAR10.



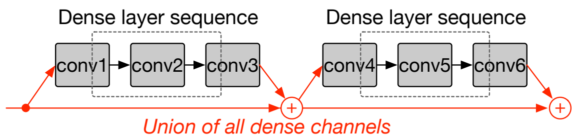
(a) (b)

Fig. 3. (a) Normalized FLOPs per training iteration (ResNet50 on CIFAR10): Each line is the result of a different regularization penalty ratio. (b) Accumulated pruned FLOPs over epochs.

Enabling larger mini-batches. Our gradual channel/layer pruning also helps to gradually reduce the memory capacity requirement for training. Network training requires keeping the output of each layer on device memory to compute its local gradients in back-propagation. As early layers have larger feature maps and our lasso function penalizes them relatively more, we effectively decrease the memory requirement. The reduced memory requirement allows using a larger mini-batch for each regression. This then furhter accelerates training by increasing data parallelism of each iteration and reducing the model parameter update frequency per epoch. Again, this mini-batch size increase during training is made possible by PruneTrain as it dynamically prunes the network. We discuss experimental results of increasing mini-batch size in Section. 4.1.



(a) Channel gating: *Channel select* and and *channel scatter* layers match the channels indexes.



(b) Channel union: The first and the last convolution layers of a residual block contain sparse channels.

Fig. 4. Channel indexing mechanisms for the CNNs with shortcuts.

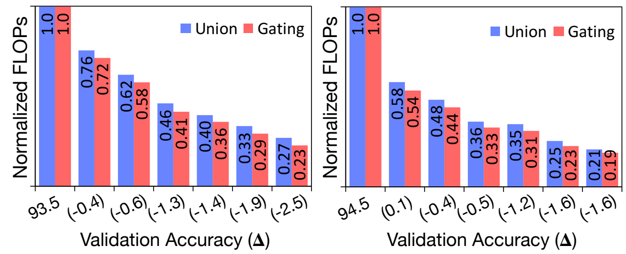
3.3. Network Architecture Reconfiguration Mechanism

we now discuss the practical mechanisms for performing dynamic reconfiguration.

We define a reconfiguration interval, such that after every such interval, the sparsified input and output channels are pruned out. Note that if all the sparsified input and output channels are pruned, there is a possibility of a mismatch between the dimensions of the output channels of one layer to the input channels of the next. To maintain dimension consistency, we only prune the intersection of the sparsified channels of any two adjacent layers. At any reconfiguration, all training variables of the remaining channels (e.g., parameter momentums) are kept as is.

The reconfiguration interval is the only additional hyperparameter added by PruneTrain. Intuitively, a very short reconfiguration interval may degrade learning quality while a long interval offers less speedup opportunity. We extensively evaluate the impact of the reconfiguration interval in Section. 4.1 and show that training is robust within a range of $10-30$ epochs.

CNNs with Short-cut Connections. For CNNs with short-cut connections (e.g., ResNet (He et al., 2016a) and its variations (He et al., 2016b; Xie et al., 2017; Hu et al., 2017; Huang et al., 2017)), the channels of the convolution layers at a merge-point should match in dimensionality after reconfiguration for proper feature propagation. We propose two mechanisms to ensure this occurs. We first introduce **channel gating** layers and add gating to each residual branch to match their dimensions, as shown in Fig. 4(a). This ensures that all convolution layers in a residual block operate only on dense channels by gathering and scattering the dense channel indices. This improves on the channel sub-sampling approach proposed by (He et al., 2017), with channel sub-sampling only avoiding redundant computation of the very first convolution layer of each residual block.



(a) Inf. FLOPs (ResNet32) (b) Inf. FLOPs (ResNet50)

Fig. 5. Normalized training and inference FLOPs of ResNet32 and ResNet50 on CIFAR10 by different pruning intensity.

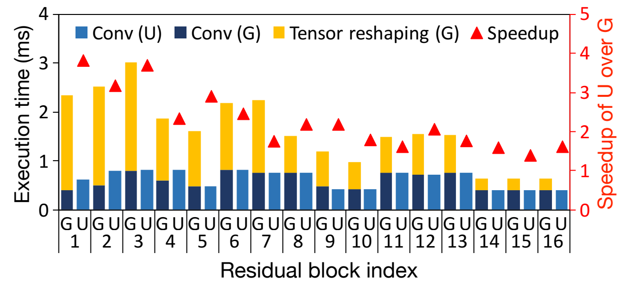


Fig. 6. Per-layer execution time of channel gating and channel union for ResNet50 on ImageNet. G and U indicate channel gating and channel union respectively.

We use GPU kernel profiling and find that channel gating involves significant memory accesses for tensor reshaping that often slows down the training. Therefore, as an alternative, we propose **channel union** that avoids any tensor reshaping and data indexing. Channel union prunes only the intersection of sparse channels of all neighboring convolution layers within a residual stage. For instance, in Fig. 4(b), the union of the dense input channels of convolution layer 1 and 4 and the dense output channels of convolution layer 3 and 6 are maintained. As each residual branch adds new information, the early convolution layers in the stage (convolution layer 1) have to process operations from the sparse channels, thereby performaing redundant operations.

Our experiments show that the additional FLOPs from channel union, as compared to channel gating are very small. Fig. 5 compares the normalized inference FLOPs of channel gating and channel union for ResNet32 and ResNet50 pruned with different intensities. Across different pruning rates, the FLOPs difference is only 1-6%, but the overhead saved from indexing is substantial. Additionally, this FLOPs difference does not grow with increasing layer depth as shown in Fig. 5 comparing ResNet32 and ResNet50. The measured per-layer (the last layer of each residual block) execution time of ResNet50 on ImageNet is shown in Fig. 6. For all residual blocks, the execution time of channel gating overwhelms channel union's by a large margin. Especially, the tensor reshaping time of early layers has bigger overhead as their activation size is eight times bigger than the layers in the last residual block.

4. Evaluation

We evaluate PruneTrain on ResNet32/50/56 (He et al., 2016a) and VGG11/13 (Simonyan & Zisserman, 2014) on CIFAR10/100 (Krizhevsky & Hinton, 2009) and compare the results to the prior work. We also evaluate PruneTrain on ResNet50 with ImageNet. All networks are trained from scratch. We use a mini-batch size of 128 for the experiments on the CIFAR datasets, and 256 for ImageNet. For all ResNets, we use **channel union** for architecture reconfiguration. All experiments on CIFAR10/100 are conducted on NVIDIA TITAN Xp GPU and ResNet50 on ImageNet is trained using both TITAN Xp and TESLA V100 GPUs (nvidia, 2017).

4.1. Evaluation on CIFAR

We present our evaluation results in Tab. 1, with 4 metrics: the training and inference FLOPs, training time, and validation accuracy. The results from the ResNet and VGG networks using PruneTrain on CIFAR10/100 are compared against the dense baseline. In this experiment, we use the same number of training iterations as the dense baseline, and the regularized results are not fine-tuned. For ResNet32 and ResNet50 on CIFAR10, PruneTrain reduces the training FLOPs down to 47-50% with minor accuracy drop compared to the dense network. Additionally, given that SSL (Structured Sparsity Learning) (Wen et al., 2016) runs the group lasso regressions as an additional training pass on a pre-trained model, PruneTrain accelerates the end-to-end training by >2X, by taking just one, gradually-pruned pass at training. The compressed models observe only 34% and 30% of the dense inference cost for ResNet32 and ResNet50 respectively.

The results of ResNet32/50 on CIFAR100 show similar patterns, which certifies the robustness of PruneTrain, given that CIFAR100 is a more difficult classification problem. For CIFAR100, PruneTrain reduces the training and inference FLOPs down to 68% and 54% for ResNet32, and 47% and 31% for ResNet50, while losing only 1.4% and 0.7% of validation accuracy respectively. These results show that PruneTrain reduces more training FLOPs from a deeper network, since more number of unimportant channels and layers are removable early on in the training. The training time reduction is smaller, compared to reduced FLOPs. This is attributed to both SW, and HW issues e.g. GPU kernel launch time and smaller data parallelism due to the reduced number of channels.

PruneTrain also achieves a high model pruning rate with similar validation accuracy loss for VGG networks. One major distinction from ResNets is that the measured training time is shorter. This is because VGG networks have a small number of layers, and each of them has a wider computation parallelism. This results in smaller overheads from kernel calls and GPU HW underutilization.

Tab. 1. Training FLOPs and time comparison to the dense baseline: 182 training epochs (He et al., 2016a) are used. Top1 validation accuracy of the dense baselines for CIFAR10: ResNet32(93.6), ResNet50(94.2), VGG11(92.1), VGG13(93.9), and for CIFAR100: ResNet32(71.0), ResNet50(73.1), VGG11(70.6), VGG13(74.1)

Dataset	Network	Val. Acc. Δ	Inference FLOPs	Train. FLOPs (E2E time)
CIFAR10	ResNet32	-1.8%	34%	47% (81%)
	ResNet50	-1.1%	30%	50% (81%)
	VGG11	-0.7%	35%	43% (57%)
	VGG13	-0.6%	37%	44% (57%)
CIFAR100	ResNet32	-1.4%	54%	68% (88%)
	ResNet50	-0.7%	31%	47% (66%)
	VGG11	-1.3%	43%	53% (74%)
	VGG13	-1.1%	48%	58% (67%)

Tab. 2. Training FLOPs and time comparison to SSL: Networks are trained with additional training epochs to learn the compressed network. The accuracy is compared to the dense baseline, as mentioned in Tab. 1.

Dataset	Network	Val. Acc. Δ	Inference FLOPs	Train. FLOPs (E2E time)
CIFAR10	ResNet32	-2.0%	29%	40% (73%)
	ResNet50	-1.3%	25%	42% (72%)
	VGG11	-0.6%	29%	39% (55%)
	VGG13	-0.3%	34%	41% (55%)
CIFAR100	ResNet32	-2.4%	41%	53% (81%)
	ResNet50	-0.5%	23%	39% (61%)
	VGG11	-0.9%	39%	49% (83%)
	VGG13	-1.3%	46%	54% (65%)

Since learning the compressed architecture can require a large number of training iterations (Section. 3.2), we train both ResNets and VGGs on CIFAR10/100 until the compression rate reaches a plateau. The model is compressed only at high learning rate of 0.1. Then, the learned architecture is further trained for the rest of the epochs with decaying learning rates, as per the original schedule, without the regularization. Tab. 2 compares the validation accuracy and the inference FLOPs of the described training method to the baseline, and the training FLOPs to SSL (assumed trained from scratch). Note that SSL uses a 2-pass training to get a pruned network. However, to estimate our traning-time speedup, we run SSL with our 1-pass group lasso training, but without the gradual pruning. For most of the network-dataset pairs, extended architecture learning further com-

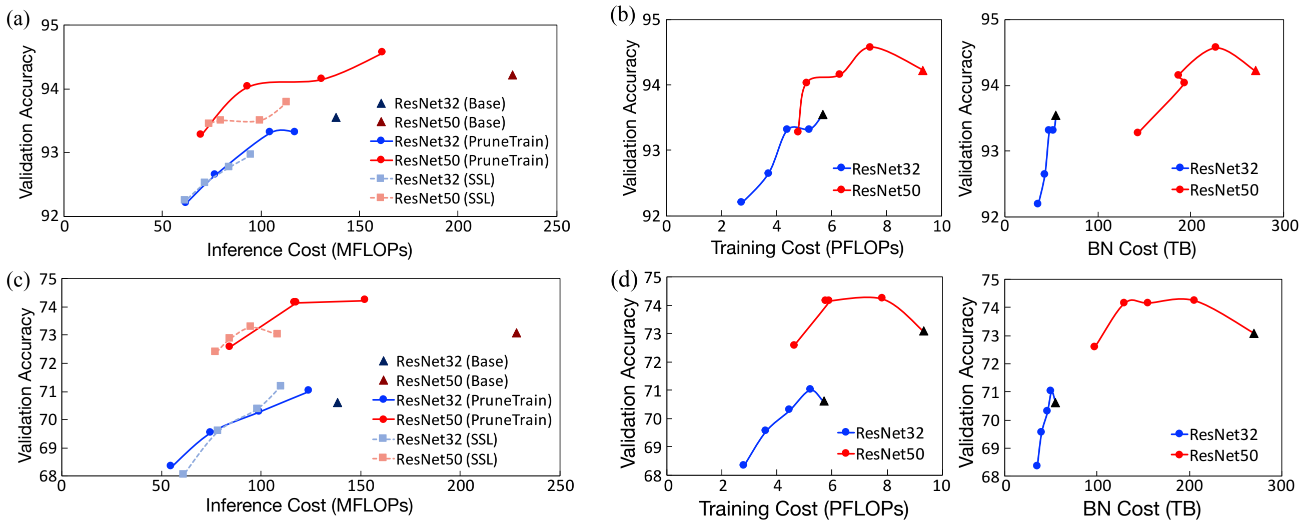


Fig. 7. (a) Inference FLOPs and the validation accuracy by different regularization ratios of PruneTrain and SSL for ResNet32/50 on CIFAR10 (c) and on CIFAR100, (b) Training FLOPs and BN cost by accuracy of PruneTrain for ResNet32/50 on CIFAR10 and on (d) CIFAR100. (The triangles in all figures represent the dense baseline)

presses the inference FLOPS by the range of 2-13% almost without losing any accuracy. Since the training runs longer, the training time reduction compared to SSL is also larger, as compared to Tab. 1.

Tab. 3. Comparison to AMC (Auto ML for Model Compression): Compression results of ResNet56 on CIFAR10. The results of AMC after fine-tuning are taken from (He et al., 2018).

Method	Base Val. accuracy	Val. Acc. Δ	Inference FLOPs	Removed layers
PruneTrain	94.5%	-0.5%	34%	18 (21%)
AMC	92.8%	-0.9%	50%	Not known

Comparison to Model Pruning with Trial-and-Error. We also compare the results of PruneTrain to AMC (Auto ML for model compression) (He et al., 2018), to show that learning the architecture by regression leads to a better compression and accuracy trade-off than trial-and-error based pruning (Tab. 3). We use ResNet56 on CIFAR10 as AMC presents the structured pruning result on this combination. While AMC reduces the inference FLOPs to 50% with 0.9% accuracy drop with fine-tuning, PruneTrain reduces an additional 16% FLOPs while achieving a higher accuracy by 0.4%, as compared to AMC. While the capability of learning the depth of network was not discussed in AMC, PruneTrain learns the depth of the network, and removes 21% of the convolution layers of ResNet56.

Comparison to SSL. Fig. 7(a) and Fig. 7(c) compare training results of dynamic (PruneTrain) and **static model pruning that is trained from the pre-trained model (SSL)**. We measure the inference FLOPs and the validation accuracy of both PruneTrain and SSL trained with the same group lasso coefficient determined by our regularization penalty ratio ranging from 0.05 to 0.2. For different networks, datasets, and pruning intensity, PruneTrain shows similar or slightly better tradeoff in accuracy-to-inference FLOPs than SSL. This indicates that PruneTrain effectively accelerates model training and compression both by pruning from scratch and gradually pruning while training. Fig. 7(a) and Fig. 7(c) also show that the compressed ResNet50 architecture has a higher validation accuracy than the dense ResNet32, given the same inference cost. Therefore, PruneTrain can learn a network structure by training a network with a larger initial complexity eventually for lower computation cost, and better prediction accuracy.

Fig. 7(b) and Fig. 7(d) show the training cost of ResNet32/50 on CIFAR10/100; the FLOPs of neuronal layers and the memory traffic of batch normalization layers that dominate the training time. PruneTrain reduces both the computation, and memory traffic with a minor accuracy loss. Interestingly, unlike the computation FLOPs, the memory traffic does not linearly scale with regularization penalty ratio. This is because, the regression learns different number of channels for different layers and the per-channel computation and memory cost are not always correlated. E.g. removing a channel for 1x1 convolution reduces fewer computations than a 3x3 convolution, but their memory cost reduction is the same.

Network Reconfiguration Interval. Network reconfiguration interval should, intuitively, affect the network pruning rate and the prediction accuracy. Note that the case with the longest possible reconfiguration interval is *static pruning*

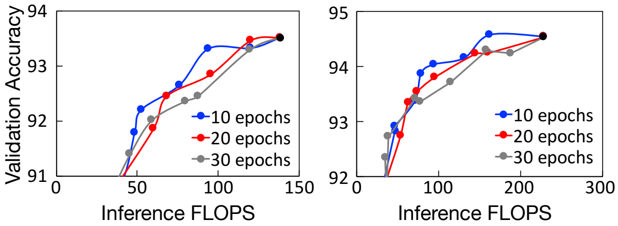


Fig. 8. Reduced inference FLOPs and validation accuracy by different network reconfiguration intervals. ResNet32 (Left) ResNet50 (Right) on CIFAR10.

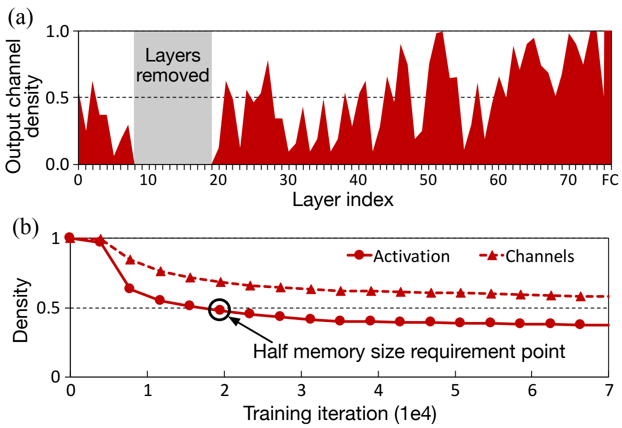


Fig. 9. ResNet50 on CIFAR100: (a) Per-layer output channel density of pruned network. (b) Output channel density and training memory requirement over training iteration.

Tab. 4. Training FLOPs and time reduction by dynamic mini-batch size adjustment for ResNet50 trained on CIFAR100.

Network	Method.	Train. time reduction	Inference FLOPs	Val. Acc. Δ
ResNet50	Naive	61%	23%	-0.5%
	Adjusted	42%	25%	-0.8%

(SSL), given that the reconfiguration happens at the end of training. Considering that even SSL shows a similar pruning quality as PruneTrain, the training is insensitive to the hyper-parameter. Such insensitivity is shown in Fig. 8, where we compare the inference FLOPs and validation accuracy of the pruned network for three different reconfiguration rates. This shows that the prediction accuracy given the compression rate is robust to the architecture reconfiguration interval. Since using a shorter reconfiguration interval reduces training cost more, it is encouraging to use a short architecture reconfiguration interval.

Pruned Network Architecture. Our group lasso regularization applies equal penalty to all groups, thus making the channels of early layers have a larger probability of pruning, as discussed in Section 3.1. This reduces the channel density of early layers more than the later layers. This is shown in Fig. 9a, demonstrating per-layer channel density of ResNet50 on CIFAR100. Since each channel in the early layers generates a larger feature map than the later layers, removing such channels reduce the memory requirement for training by a large degree. Fig. 9b shows the output channel density and the training memory requirement that is of the reconfigured network over training epochs. As early layers' channel density is lower, the memory requirement is smaller than network's average channel density.

Increasing Mini-Batch Size. To further accelerate training, we increase the mini-batch size by 2X when the memory requirement becomes lower than half. While doing so, we also increase the learning rate by 2X to achieve the same accuracy as discussed in (Smith et al., 2017). Tab. 4 compares the changes in validation accuracy, network compression rate, and the end-to-end training time. Adjusting mini-batch size generates equally compressed network architectures with minor accuracy loss, and the training time is further reduced by 19%.

4.2. Evaluation on ImageNet

We train and prune ResNet50 on ImageNet from scratch using PruneTrain with a regularization penalty ratio of 0.2. We train with both baseline training and PruneTrain for same number of iterations, using the same schedule of 90 epochs. As compared to the dense baseline training, PruneTrain reduces the total training FLOPs by 29% and the end-to-end training time by 17% using TITAN Xp GPU and 18% using TESLA V100 GPU. Also, compared to the dense baseline, it reduce the inference FLOPs by 41% losing a minor accuracy as shown in Tab. 5.

Tab. 5. ResNet50 training results on ImageNet with PruneTrain.

Baseline Val. Acc.	Val. Acc. Δ	Inference FLOPs	Train. FLOPs (E2E time TITAN Xp/V100)
76.2%	-1.5%	59%	71% (83 / 82%)

5. Conclusion

With the ever growing hyper-parameter search space and increasing datasets, cost-efficient network training is as important as devising a network architecture. To accelerate network training and prune computation, we propose PruneTrain, a mechanism to gradually and structurally trim the network architecture in a single pass of training. Gradual pruning leads to faster training, while structural pruning guarates faster inference. PruneTrain is based on our observations that with the right hyper-parameter chosen, a single pass of training is enough to learn a pruned network, and once a parameter is sparsified by group lasso regressions, it rarely revives. We further speed up the training by reconfiguring short-cuts containing CNNs into an execution time-efficient form, by using channel union and increasing mini-batch size during training. PruneTrain can be used to build cost-efficient deep neural network architectures in less than half the training cost as compared to the current state-of-the-art.

6. Acknowledgment

The authors acknowledge Texas Advanced Computing Center (TACC) for providing GPU resources.

References

Alvarez, J. M. and Salzmann, M. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2270–2278, 2016.

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Feng, J. and Darrell, T. Learning the structure of deep convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2749–2757, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., Xie, D., Luo, H., Yao, S., Wang, Y., et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 75–84. ACM, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, 2017.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018.

Hu, H., Peng, R., Tai, Y.-W., and Tang, C.-K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 7, 2017.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, volume 1, pp. 3, 2017.

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

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR, abs/1611.06440*, 2016.

nvidia. Nvidia tesla v100 gpu architecture. *White paper*, 2017.

Simon, N. and Tibshirani, R. Standardization and the group lasso penalty. *Statistica Sinica*, 22(3):983, 2012.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2074–2082. Curran Associates, Inc., 2016.

Wen, W., He, Y., Rajbhandari, S., Zhang, M., Wang, W., Liu, F., Hu, B., Chen, Y., and Li, H. Learning intrinsic sparse structures within long short-term memory. *arXiv preprint arXiv:1709.05027*, 2017.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 5987–5995. IEEE, 2017.

Yu, J., Lukefahr, A., Palframan, D., Dasika, G., Das, R., and Mahlke, S. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *ACM SIGARCH Computer Architecture News*, volume 45, pp. 548–560. ACM, 2017.

Yuan, M. and Lin, Y. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68 (1):49–67, 2006.