Network Pruning Based on Multiple Datasets for Improved Performance and Generalization

Adriana Salcedo
Department of Medical Biophysics
University of Toronto
Toronto, ON M5S 1A1
a.salcedo@mail.utoronto.ca

Shashwat Sharma
Edward S. Rogers Sr. Department of Electrical & Computer Engineering
University of Toronto
Toronto, ON M5S 1A1
shash.sharma@mail.utoronto.ca

Abstract

The abstract paragraph should be indented ½ inch (3 picas) on both the left- and right-hand margins. Use 10 point type, with a vertical spacing (leading) of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 Introduction

Modern neural networks tend to be immensely large with several millions of connections and weights. The memory and energy requirements for deploying such networks is significant, and constantly growing with the complexity of state-of-the-art networks. Although having a large number of trainable parameters is beneficial to network accuracy, it was shown by Han et al. (2015), LeCun et al. (1990) and Hassibi et al. (1993) that the impact of some parameters is negligible in comparison with others. Successful identification and removal ("pruning") of such parameters would lead to smaller and faster networks, which in turn would significantly reduce the computational cost at test time. The goal of most pruning algorithms is to achieve computational advantages without an appreciable impact on accuracy. The importance and relevance of network pruning was recently highlighted via MorphNet, proposed by Gordon et al. (2018) at Google to optimize existing networks through pruning.

It was argued by Molchanov et al. (2017) that pruning can also be applied in the context of transfer learning, for generalization of a trained network to related datasets. Improved ability of networks to generalize as a result of pruning was also achieved by Liu et al. (2019). However, these approaches target networks that have been trained on a single dataset. In contrast, the concept of learning the hyperparameters of a network by using multiple datasets, known as "meta-learning", has also been proposed by Finn et al. (2017). To the best of our knowledge, the meta-learning concept has not been applied in the context of network pruning.

The above points motivate the main idea proposed in this work: designing pruning methodologies that specifically target improved generalization of a given network, by leveraging multiple datasets during training. The goal is to optimize existing networks not only to be faster and smaller, but also to improve their ability to generalize. We first implement and study some simple and commonly-used pruning strategies and analyze their impact on network size, speed and accuracy. We then discuss

CSC2516 S 2019: project report.

and implement a simple meta-learning algorithm and study its performance. Finally, we combine the pruning strategies with the meta-learning algorithm to introduce a proof-of-concept “meta-pruning” methodology.

2 Formal Description

2.1 A Survey on Simple Pruning Algorithms

2.1.1 Masking of Classifier Layers Based on a Threshold

2.1.2 Learned Masking of Classifier Layers

2.1.3 Pruning Convolution Filters

One method to prune convolutional layers is to reduce the number of output channels in each layer, which involves reducing the size of associated weight matrices. This impacts not only the convolution layer being pruned, but also the number of input channels in the next layer of the network. This approach was proposed by Molchanov et al. (2017), and has since been implemented via open-source code provided by Gildenblat (2017). However, this implementation only allows for pruning a single output channel at a time. We have implemented a version of this technique that allows pruning out several filters simultaneously, greatly improving the efficiency of aforementioned the code.

Although a relatively sophisticated strategy for picking pruned channels was presented by Molchanov et al. (2017), in our implementation we use two simple metrics to decide which channels are to be pruned: the 2-norm of the layer's activations, and the 2-norm of weights. In each case, pruning occurs after every N_i epochs, until training is complete. The number of output channels to prune, N_p, is set via the hyperparameter p, which is a percentage of the total available parameters in that network, N_{total}:

$$N_p = \mathrm{round}\left(\frac{p}{100} N_{\mathrm{total}}\right). \tag{1}$$

In the activation-based approach, the activations of each layer are accumulated over the N_i epochs preceding a single pruning pass via the function `TrackConv2DNorms`. This is done as follows: at each training iteration, we compute the 2-norm of the activations of each layer, across all of the input data dimensions (function `ComputeConv2DActNorms`). The output of this is a vector whose size is the number of output channels of a particular layer, and one such vector is constructed for each convolution layer. These vectors represent the strength of the activations of each filter. Optionally, these vectors can be normalized to their maximum values to ensure that every set of N_i epochs is equally represented.

The new activation norm vectors computed at each iteration are added to the previously stored vectors. Every N_i epochs, we look at the accumulated activation vector and extract the indices corresponding to the N_p smallest activations (function `PruneAllConv2DLayers`). This tells us the N_p filters of each layer which had, on average, the weakest activations over the last N_i epochs. Those filters are then pruned away, and the old model is replaced with the updated, smaller model (function `PruneConvLayers`). At this stage, the accumulated activation norm vectors are reset to their initial state, ready to start accumulating activations for the next N_i epochs.

When pruning based on weights, the strategy is the same as above, except that rather than accumulating activations, we accumulate the 2-norms of the weight matrices themselves. The 2-norms are computed over the input channel dimension, and over the kernel of each filter (function `ComputeConv2DWeightNorms`). Again, this yields, for each layer, a vector containing 2-norms corresponding to the weights of each output channel. This can also be optionally normalized to the maximum value in each vector. Every N_i epochs, these accumulated vectors are used to prune away the weakest N_p filters.

The methodology described above is summarized in Algorithm 1. The results of both methods in comparison to the baseline (no pruning) are discussed below.

Algorithm 1 Helper functions for iterative convolution filter pruning - single dataset

```
function ComputeConv2DWeightNorms($\text{model}, \mathbf{n}_w$)
    for all $i$ in convolution layers of $\text{model}$ do
        $\widetilde{\mathbf{n}}_{w,i} \leftarrow$ 2-norm of weights of layer i across dims 1, 2, 3
        $\widetilde{\mathbf{n}}_{w,i} \leftarrow \widetilde{\mathbf{n}}_{w,i} / \max |\widetilde{\mathbf{n}}_{w,i}|$ (optional)
        $\mathbf{n}_{w,i} \leftarrow$ append $\widetilde{\mathbf{n}}_{w,i}$
    return $\mathbf{n}_{w,i}$

function ComputeConv2DActNorms($\text{model}, \mathbf{n}_a$)
    for all $i$ in convolution layers of $\text{model}$ do
        $\widetilde{\mathbf{n}}_{a,i} \leftarrow$ 2-norm of activations of layer i across dims 0, 2, 3
        $\widetilde{\mathbf{n}}_{a,i} \leftarrow \widetilde{\mathbf{n}}_{a,i} / \max |\widetilde{\mathbf{n}}_{a,i}|$ (optional)
        $\mathbf{n}_{a,i} \leftarrow$ append $\widetilde{\mathbf{n}}_{a,i}$
    return $\mathbf{n}_{a,i}$
```

Algorithm 2 Computational functions for iterative convolution filter pruning - single dataset

```
function PruneConvLayers($\mathbf{l}_{\text{idx}}, \mathbf{f}_{\text{idx}}, \text{model}$)
    $\mathbf{c}_{\text{idx}} \leftarrow \mathbf{l}_{\text{idx}}^{\text{th}}$ conv layer of $\text{model}$
    $\mathbf{w}_0 \leftarrow$ weights of $\mathbf{c}_{\text{idx}}$
    $\mathbf{w}_0 \leftarrow$ delete entries corresponding to filters in $\mathbf{f}_{\text{idx}}$
    $\mathbf{c}_{\text{idx}} \leftarrow$ copy updated weights $\mathbf{w}_0$
    $\text{model} \leftarrow$ updated conv layer $\mathbf{c}_{\text{idx}}$ at layer $\mathbf{l}_{\text{idx}}$
    return $\text{model}$

function PruneAllConv2DLayers($\mathbf{n}_a, \mathbf{n}_w, N_p, \text{model}$)
    for all $i$ in convolution layers of $\text{model}$ do
        if activation-based pruning then
            $\mathbf{n}_i \leftarrow \mathbf{n}_{a,i}$
        else if weight-based pruning then
            $\mathbf{n}_i \leftarrow \mathbf{n}_{w,i}$
        $\mathbf{f}_{\text{idx,i}} \leftarrow$ indices of bottom $N_p$ elements of $\mathbf{n}_i$
        $\text{model} \leftarrow$ PruneConvLayers($i, \mathbf{f}_{\text{idx,i}}, \text{model}$)
    return $\text{model}$
```

2.2 Pruning with Multiple Datasets

2.2.1 Intersection Pruning of Classifier Layers Based on a Threshold

2.2.2 Meta-Pruning Classifier Layers

2.2.3 Intersection Pruning of Convolution Filters

Our initial attempt at applying meta-pruning to convolution filters by using the L0 norm-based masking yielded significantly deteriorated training results. At the same time, the results obtained by weight- or activation-based pruning (rather than learned masking) were comparable to the baseline case of no pruning, discussed further in Section 4.1.3. These observations motivated us to leverage multiple datasets in the sense of intersection pruning, as described earlier in Section 2.2.1, to prune convolution layers. This method proceeds exactly as the description in Section 2.1.3, except that after every N_i epochs, the input data is resampled from a different dataset. The associated algorithm is summarized in Algorithm 4.

Algorithm 3 Driver functions for iterative convolution filter pruning - single dataset

```
function TrainModel(model, input, $N_i$, $N_p$)
    for all $i$ in input do
        $\mathbf{n}_a + \leftarrow$ ComputeConv2DActNorms(model, $\mathbf{n}_a$)
        $\mathbf{n}_w + \leftarrow$ ComputeConv2DWeightNorms(model, $\mathbf{n}_w$)
        if $i \% N_i == 0$ then
            model $\leftarrow$ PruneAllConv2DLayers($\mathbf{n}_a$, $\mathbf{n}_w$, $N_p$, model)
            $\mathbf{n}_a \leftarrow$ reset
            $\mathbf{n}_w \leftarrow$ reset
    return model

model $\leftarrow$ VGG16
$p \leftarrow 5$
$N_p \leftarrow$ value as per (1)
$N_i \leftarrow 7$
input $\leftarrow$ sample 10% of data from CIFAR-10
while $i < 50$ do
    model $\leftarrow$ TrainModel(model, input, $N_i$, $N_p$)
```

Algorithm 4 Driver function for meta convolution filter pruning - multiple datasets

```
model $\leftarrow$ VGG16
$p \leftarrow 5$
$N_p \leftarrow$ value as per (1)
$N_i \leftarrow 7$
datasets $\leftarrow$ dataloaders for several different datasets, in this case 8
$j \leftarrow 0$
while $i < 50$ do
    if $i \% N_i == 0$ then
        input $\leftarrow$ sample 10% of data from dataset $j$
        $j \leftarrow j + 1$
    model $\leftarrow$ TrainModel(model, input, $N_i$, $N_p$)
```

3 Related Work

4 Demonstration

The existing PyTorch implementation of VGG16 was used as the model to prune in all tests conducted in this work. The optimizer and other settings used were the same in all tests conducted, and can be found in accompanying code.

4.1 A Study on Simple Pruning Algorithms

In this study, the model was trained only on CIFAR-10 images. Due to limitations in computational resources available via Google Colab, only 2-10% of the total available training images were used, to get results in a reasonable amount of time. Since the main point to be studied here is the performance of a pruned network compared to an unpruned network, we are only interested in the relative loss and accuracy. Thus, using the partial dataset is justified.

4.1.1 Masking of Classifier Layers Based on a Threshold

4.1.2 Learned Masking of Classifier Layers

4.1.3 Pruning Convolution Filters

Both activation-based and weight-based filter pruning approaches were studied. Since pruning is generally applied to fine-tune pre-trained network, the pre-trained version of VGG16 was used in this study. Pre-training was done on ImageNet, and the fine-tuning while pruning was done on CIFAR-10 images. In each case, pruning was performed after every $N_i = 7$ epochs. The percentage of filters pruned each time was 5%. Since the same settings were used in both methods, the number of filters pruned was the same in both cases. At the start, the model contained a total of 4224 filters. After 49 epochs, 3136 filters survived, thus reducing the network size to 74% of its original size, not counting the classifier layers. Although the training time is usually not something that is critical to reduce, it is worth noting that training becomes faster as it proceeds due to the decreasing size of the network. The first 7 epochs took 19 minutes on average to train, while the last 7 epochs took 14 minutes on average. This is an indication of the expected improvement in network agility.

Training and validation results for both pruning approaches are shown in Figure 1. The top row shows accuracy, while the bottom row shows loss. The first column on the left shows baseline results with no pruning. The second column shows activation-based filter pruning results, and the third column shows weight-based pruning results. It was found that normalization of the accumulated weight or activation vectors does not have a significant impact on the results, so we only show results for the case when the vectors are not normalized.

The figures show that the both filter pruning approaches studied here decrease training accuracy compared to the baseline model by approximately 10% on average. However, the validation accuracy is only 3–5% lower than that obtained with the baseline model, on average. Furthermore, it is clear that the difference between training and validation accuracy in the baseline model is approximately 10%, while in the pruned networks, both training and validation performance are quite well balanced with respect to one another. This is an indicator that, as hypothesized, the pruned network has the potential to generalize better than the original network. The pruned network loss and accuracy are much less stable over epochs than the original model, which could perhaps be improved with more gradual pruning, or more sophisticated algorithms for choosing which filters to prune.

4.2 Pruning with Multiple Datasets

4.2.1 Intersection Pruning of Classifier Layers Based on a Threshold

4.2.2 Meta-Pruning Classifier Layers

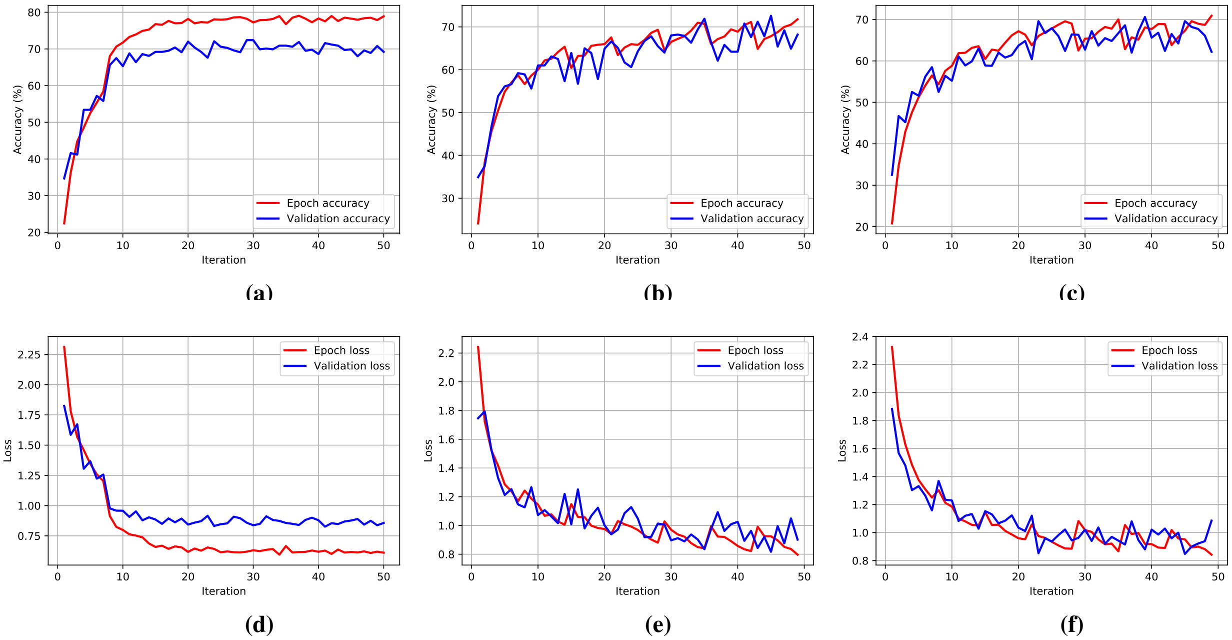
4.2.3 Intersection Pruning of Convolution Filters

5 Limitations

6 Conclusions

7 List of Contributions

*************** Dumping figures here for now ***************



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Figure 1 Accuracy and loss of pre-trained VGG16 during fine-tuning. All layers were fine-tuned on 10% of CIFAR-10 data (4000 training images, 1000 validation images) for 50 epochs. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Middle column: activation-based pruning of 5% of all filters every 7 epochs. Right column: weight-based pruning of 5% of all filters every 7 epochs.

*************** Text below is copied from proposal ****************

A variety of pruning algorithms have been proposed, many of which apply to convolutional neural networks. Some remove individual weights (leading to sparse weight matrices) while others remove entire filters, channels or layers. Units may be removed based on many criteria, including their magnitude, their contribution to the final loss (approximated through their gradients), and information criteria such as entropy Han et al. (2015); Molchanov et al. (2017); Luo and Wu (2017); Liu et al. (2017). Pruning may be applied towards a human-defined architecture (e.g. removing a certain proportion of weights in each layer) Yu et al. (2017) or help determine the final architecture autonomously Luo et al. (2017).

8 Goals

After each dataset sample, only connections that survive pruning and thus contribute to the final loss will be retained. At the beginning of each dataset sample, weights will be initialized with the value of the updated weights in the previous sample. We hypothesize that this approach will result in weights that are important for a broad set of related tasks.

It has been suggested Liu et al. (2019) that pruning can also improve algorithm efficiency by virtue of the resulting architecture, rather than just the resulting weights. Therefore, we will also assess a modified meta-pruning algorithm that only retains the architecture, and not the weights, across datasets:

Algorithm 5 Meta-pruning for weights

```
$\theta \leftarrow$ *Random initialization*
**for** all $D$ in collection of datasets $\mathcal{D}$ **do**
    $T \leftarrow$ *Sample tasks from dataset* $D$
    **for** all $i$ in $T$ **do**
        $G_i \leftarrow \nabla_{\theta,i}\mathcal{L}\left(f_{\theta,i}\right)$
        $\theta_i \leftarrow \theta_i - \beta G_i$
    $\theta \leftarrow$ *Pruned and fine-tuned* $\theta$
```

Algorithm 6 Meta-pruning for architecture

```
**for** all $D$ in collection of datasets $\mathcal{D}$ **do**
    $\theta \leftarrow$ *Random initialization*
    $T \leftarrow$ *Sample tasks from dataset* $D$
    **for** all $i$ in $T$ **do**
        $G_i \leftarrow \nabla_{\theta,i}\mathcal{L}\left(f_{\theta,i}\right)$
        $\theta_i \leftarrow \theta_i - \beta G_i$
    $\theta \leftarrow$ *Pruned* $\theta$
```

We hypothesize that this approach will result in architectures that are more generalizable and efficient than the full model. Comparing the two approaches in algorithms 1 and 2 will illuminate the relative contributions of the weights and the architecture to the value of pruning for generalizability.

9 Methods

We will first select a relatively simple pruning algorithm that can be adapted to the meta-pruning approach described above, as a proof of concept. Since pruning studies to date have largely focused on convolutional neural networks, we will develop our algorithm in the context of image recognition, initially working with the CIFAR-100 data. For the initial model, we will use VGGNet due to its simple architecture and expand to ResNet if time allows. To assess our algorithms' baseline performance, we will use test images which have the same categories as the training images. We will then assess our algorithms' capacity to generalize using a test set that contains a distinct set of categories from the training set. We will use cross entropy loss and assess the algorithms' performance by comparing its classification accuracy to the accuracy of the full model, as well as a reduced model with the same number of connections but where connections are randomly dropped. We will also compare the training time, memory requirements, and the number of retained parameters to assess the algorithms' efficiency and quantify the amount of compression achieved.

An important consideration in the proposed algorithms is the consequence of pruning to backpropagation. Particularly in algorithm 1, depending on the order in which operations are applied, special care must be taken to ensure that gradients are computed correctly for the pruned model, at each outer iteration. One way to circumvent the issue is to recompute gradients every time the model is updated via pruning. However, it may be possible to do this more intelligently by applying advanced techniques for gradients of discrete problems. This will be an area to explore, if time permits.

10 Nice-to-haves

There are several additional analyses we can incorporate if time allows:

- To ensure our results are not specific to a particular pruning method, we can test meta-pruning using several pruning algorithms that span a range of approaches.
- Explore how to modify the meta-pruning approach to preserve connections that are retained in most, but not all training sets.
- Leverage the meta-learning framework to learn how to prune Han et al. (2015).

- Explicitly incorporate transfer learning into the meta-learning framework by assessing loss on a pseudo-test validation set with different categories Li et al. (2017).
- Apply meta-pruning to a more complex real-world dataset in a transfer learning context.

Acknowledgments

Use unnumbered third level headings for the acknowledgments. All acknowledgments go at the end of the paper. Do not include acknowledgments in the anonymized submission, only in the final paper.

References

Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *CoRR*, abs/1703.03400.

Gildenblat, J. (2017). Pruning deep neural networks to make them fast and small.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. (2018). MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning Both Weights and Connections for Efficient Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1135–1143, Cambridge, MA, USA. MIT Press.

Hassibi, B., Stork, D. G., and Wolff, G. J. (1993). Optimal Brain Surgeon and General Network Pruning. In *IEEE International Conference on Neural Networks*, pages 293–299 vol.1.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Advances in Neural Information Processing Systems 2. chapter Optimal Brain Damage, pages 598–605. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Li, D., Yang, Y., Song, Y., and Hospedales, T. M. (2017). Learning to Generalize: Meta-Learning for Domain Generalization. *CoRR*, abs/1710.03463.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning Efficient Convolutional Networks through Network Slimming. *CoRR*, abs/1708.06519.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2019). Rethinking the Value of Network Pruning. In *International Conference on Learning Representations*.

Luo, J. and Wu, J. (2017). An Entropy-based Pruning Method for CNN Compression. *CoRR*, abs/1706.05791.

Luo, J., Wu, J., and Lin, W. (2017). ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *CoRR*, abs/1707.06342.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2017). Pruning Convolutional Neural Networks for Resource Efficient Inference. *CoRR*, abs/1611.06440.

Yu, R., Li, A., Chen, C., Lai, J., Morariu, V. I., Han, X., Gao, M., Lin, C., and Davis, L. S. (2017). NISP: Pruning Networks using Neuron Importance Score Propagation. *CoRR*, abs/1711.05908.