
Iterative Network Pruning 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

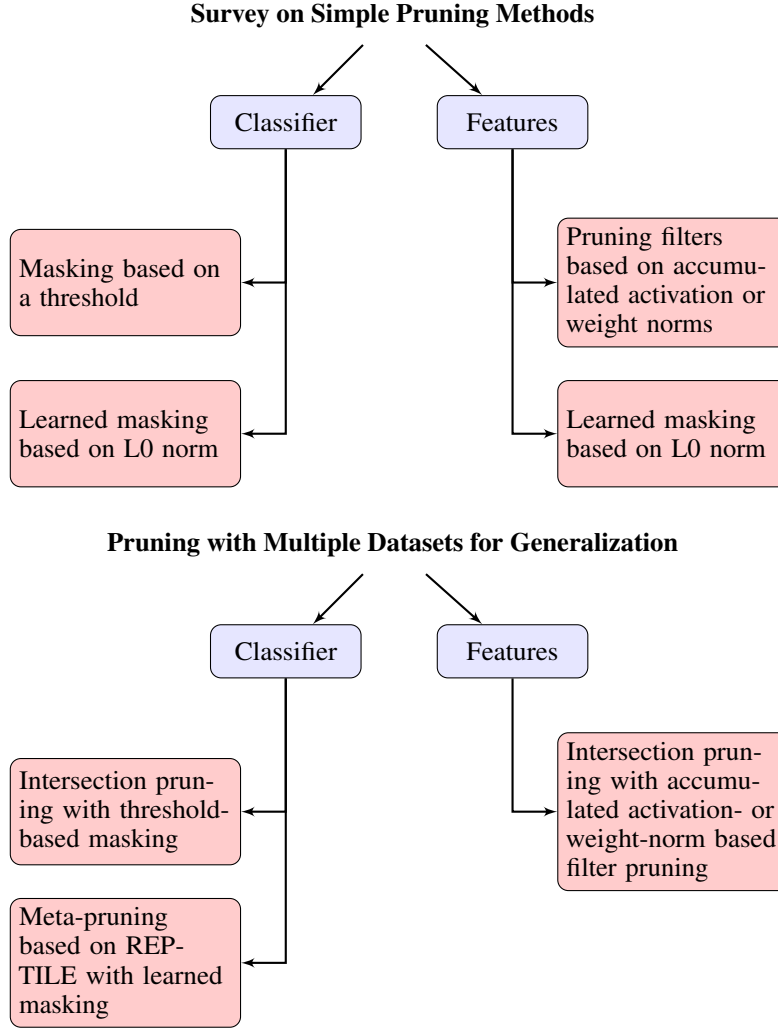


Figure 1 Outline of this paper.

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. The general outline and structure of this paper are summarized in Figure 1.

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

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

```
1: function COMPUTECONV2DWEIGHTNORMS(model,  $\mathbf{n}_w$ )
2:   for all  $i$  in convolution layers of model do
3:      $\tilde{\mathbf{n}}_{w,i} \leftarrow$  2-norm of weights of layer  $i$  across dims 1, 2, 3
4:      $\tilde{\mathbf{n}}_{w,i} \leftarrow \tilde{\mathbf{n}}_{w,i} / \max |\tilde{\mathbf{n}}_{w,i}|$  (optional)
5:      $\mathbf{n}_{w,i} \leftarrow$  append  $\tilde{\mathbf{n}}_{w,i}$ 
6:   return  $\mathbf{n}_{w,i}$ 
7:
8: function COMPUTECONV2DACTNORMS(model,  $\mathbf{n}_a$ )
9:   for all  $i$  in convolution layers of model do
10:     $\tilde{\mathbf{n}}_{a,i} \leftarrow$  2-norm of activations of layer  $i$  across dims 0, 2, 3
11:     $\tilde{\mathbf{n}}_{a,i} \leftarrow \tilde{\mathbf{n}}_{a,i} / \max |\tilde{\mathbf{n}}_{a,i}|$  (optional)
12:     $\mathbf{n}_{a,i} \leftarrow$  append  $\tilde{\mathbf{n}}_{a,i}$ 
13:  return  $\mathbf{n}_{a,i}$ 
```

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} :

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

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 2 Computational functions for iterative convolution filter pruning - single dataset

```
1: function PRUNECONVLAYERS( $l_{idx}, f_{idx}, model$ )
2:    $c_{idx} \leftarrow l_{idx}^{th}$  conv layer of model
3:    $w_0 \leftarrow$  weights of  $c_{idx}$ 
4:    $w_0 \leftarrow$  delete entries corresponding to filters in  $f_{idx}$ 
5:    $c_{idx} \leftarrow$  copy updated weights  $w_0$ 
6:   model  $\leftarrow$  updated conv layer  $c_{idx}$  at layer  $l_{idx}$ 
7:   return model
8:
9: function PRUNEALLCONV2DLAYERS( $n_a, n_w, N_p, model$ )
10:  for all  $i$  in convolution layers of model do
11:    if activation-based pruning then
12:       $n_i \leftarrow n_{a,i}$ 
13:    else if weight-based pruning then
14:       $n_i \leftarrow n_{w,i}$ 
15:     $f_{idx,i} \leftarrow$  indices of bottom  $N_p$  elements of  $n_i$ 
16:    model  $\leftarrow$  PRUNECONVLAYERS( $i, f_{idx,i}, model$ )
17:  return model
```

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

```
1: function TRAINMODEL(model, input,  $N_i, N_p$ )
2:   for all  $i$  in input do
3:      $n_a + \leftarrow$  COMPUTECONV2DACTNORMS(model,  $n_a$ )
4:      $n_w + \leftarrow$  COMPUTECONV2DWEIGHTNORMS(model,  $n_w$ )
5:     if  $i \% N_i == 0$  then
6:       model  $\leftarrow$  PRUNEALLCONV2DLAYERS( $n_a, n_w, N_p, model$ )
7:        $n_a \leftarrow$  reset
8:        $n_w \leftarrow$  reset
9:   return model
10:
11: model  $\leftarrow$  VGG16
12:  $p \leftarrow 5$ 
13:  $N_p \leftarrow$  value as per (1)
14:  $N_i \leftarrow 7$ 
15: input  $\leftarrow$  sample 10% of data from CIFAR-10
16: while  $i < 50$  do
17:   model  $\leftarrow$  TRAINMODEL(model, input,  $N_i, N_p$ )
```

2.2 Pruning with Multiple Datasets

In this section, the goal is to study two possible techniques to train a network based on samples from multiple similar datasets, rather than a single one, to promote generalization of the network. Either technique can be applied either by training the network from scratch, based on multiple datasets, or by fine-tuning a network that was previously trained on a single dataset. Since transfer learning and pruning are both generally handled as a post-processing step of an existing trained network, we will focus on the latter approach. The two techniques we consider for training on multiple datasets are:

- **Intersection Pruning:** In this approach, after every N_i epochs, the pruning step is accompanied by resampling data from a different but related dataset. Thus, every successive set of N_i epochs corresponds to a different dataset. Our hypothesis is that as the network gets pruned, the filters that are eliminated are the ones that are least important to that dataset. Thus, at the end of training, we expect that the surviving weights are the ones that are important for the intersection of all datasets.

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

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

- **Meta-Pruning:** The goal of this approach is to learn a masking matrix applied to weights, where the mask is learned based on a probability metric and decides which weights are to be pruned. **Adriana can add more if needed, here or elsewhere**

The goals of the methods proposed in this section are twofold:

1. To study whether learning by sampling multiple datasets, or subsampling the same dataset multiple times throughout the fine-tuning phase, improve the ability of the network to generalize, and
2. To study the effects of intersection pruning on fine-tuning the network on multiple datasets, or subsamples of the same dataset. Furthermore, to study whether pruning has an effect on how the network generalizes.

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, 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.

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 2. 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 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

The following two studies were conducted:

- **Iterative subsampling from the same dataset:** Training was performed on CIFAR-10, and 10% of the available data (4000 images) was resampled with replacement after every $N_i = 7$ epochs. This study is meant to be a toy example of the more realistic case when several similar datasets are available, along with the computational resources required to use larger numbers of training samples at each iteration. 5% of the network’s convolutional filters were pruned every $N_i = 7$ epochs, based on the un-normalized 2-norms of the layers’ weight matrices as well as their activations, as described above. This was compared to the baseline case of training the network in the same way, but without pruning. The results are shown in Figure 3.

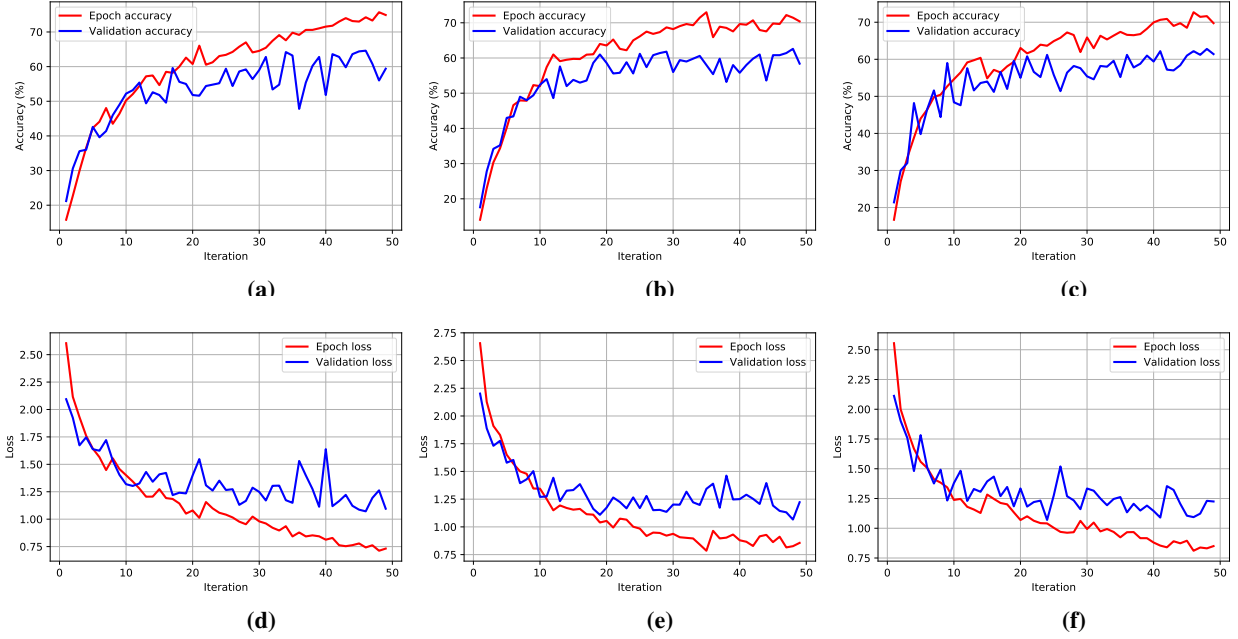


Figure 2 Filter pruning on a single dataset: Accuracy and loss of pre-trained VGG16 during fine-tuning on 5% of CIFAR-10 data (2000 training images, 500 validation images) for 50 epochs. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Middle column: activation-based pruning of 5% of all filters every 7 epochs. Right column: weight-based pruning of 5% of all filters every 7 epochs.

Similar to the results seen in Section 4.1.3, the figures show that both filter pruning approaches studied here decrease training accuracy compared to the baseline model by approximately 10% on average. Again, the validation accuracy is only 3–5% lower than that obtained with the baseline model, on average. 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. At least, the pruned network performs comparably well on validation data, but with 74% of the original network’s filters. The pruned network loss and accuracy are again much less stable over epochs than the original model.

- **Iterative sampling from the multiple datasets:** training was performed by sampling 5% of available training data from six different datasets provided via `torchvision`. A different dataset was sampled every $N_i = 6$ epochs. A total of 120 epochs were run, so that every dataset was sampled twice. The six datasets used were CIFAR-10, CIFAR-100, MNIST, KMNIST, FashionMNIST and EMNIST. This study is meant to be a stress-test where the model is fine-tuned and pruned based on multiple datasets with images of very different categories. 5% of the network’s convolutional filters were pruned every $N_i = 6$ epochs, based on the un-normalized 2-norms of the layers’ weight matrices. This was compared to the baseline case of training the network in the same way, but without pruning. The results are shown in Figure 4.

The sudden jumps in the figures indicate when a new dataset was sampled. The first six sets of peaks correspond to the first pass over each different dataset, while the last set of six peaks corresponds to the second pass over each dataset. Overall, it is seen that in both the baseline and pruning cases, the accuracy improves during the second pass over the same dataset. Although this seems to be obviously expected, it is an important observation, for the following reason: between the first and the second time that the same dataset is encountered, the model trains on five other datasets in between (with possibly a completely different

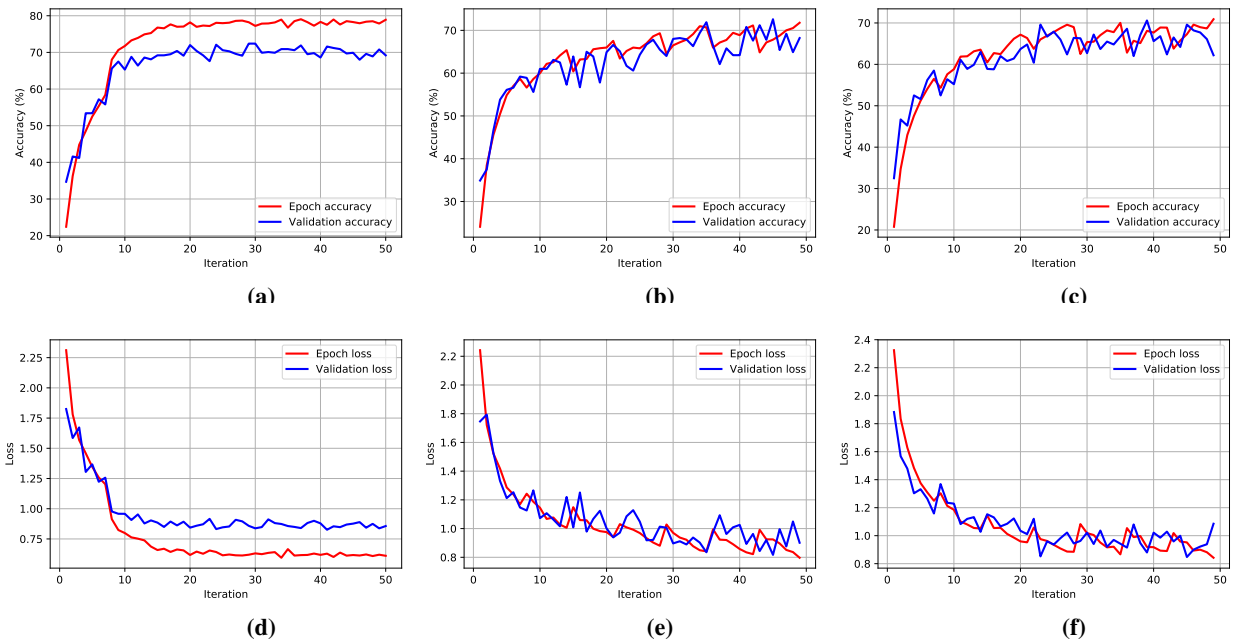


Figure 3 Iterative subsampling from CIFAR-10: Accuracy and loss of pre-trained VGG16 during fine-tuning. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Middle column: activation-based pruning of 5% of all filters every 7 epochs. Right column: weight-based pruning of 5% of all filters every 7 epochs.

set of categories and image types). One may expect these intermediate epochs to hurt the model’s performance on each previous dataset it was trained on. However, it seems that the model becomes relatively robust to being trained on multiple different datasets; in other words, introducing it to a new dataset does not make it “forget” what was learned on previous datasets. This indicates that training on multiple datasets does indeed hold promise in making networks generalize well. However, an important point to remember is that each time a new dataset is encountered, the classifier layer weights will change significantly to learn the possible categories of that dataset. So at the end of training, the network will only perform well when tested on the last dataset encountered. To test on a different dataset, a very small amount of fine-tuning only the classifier is needed. However, our claim is that fine-tuning the entire network on multiple datasets should mean that minimal fine-tuning of the classifier should be required when switching among datasets.

An important

5 Limitations

6 Conclusions

7 List of Contributions

The work discussed in this paper is not in any way related to either authors’ research or other activities, and was done purely for the purpose of submission as a project for CSC2516 at the University of Toronto. Specific contributions of each author are listed in Table 1.

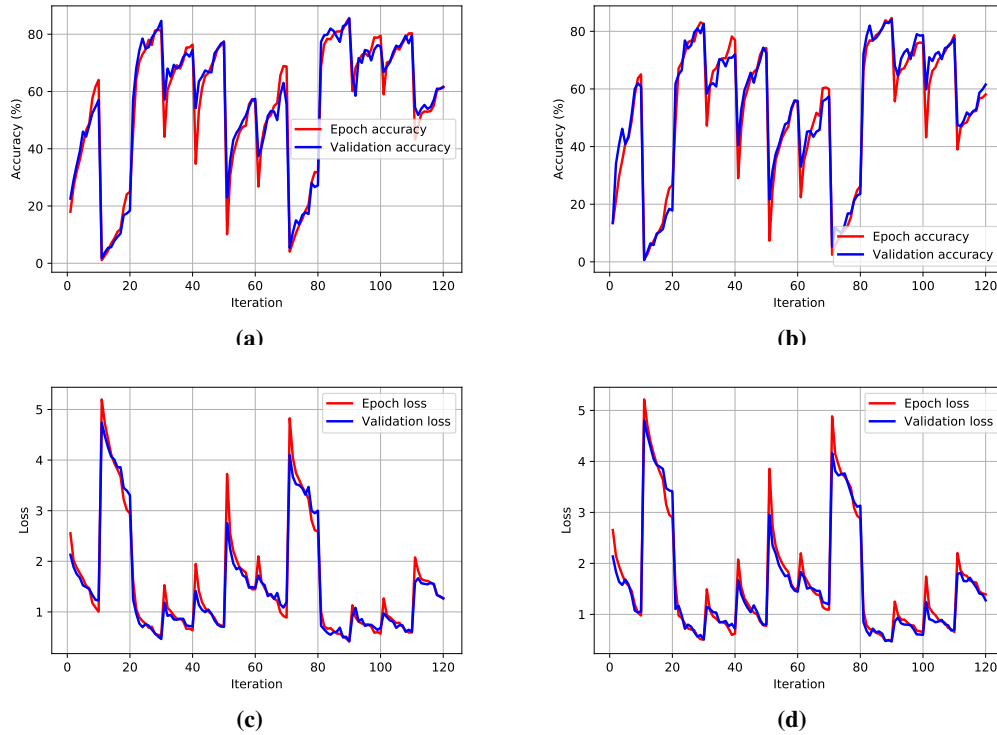


Figure 4 Iterative sampling from multiple datasets: Accuracy and loss of pre-trained VGG16 during fine-tuning. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Right column: weight-based pruning of 5% of all filters every 6 epochs.

Table 1 Contributions of each author.

	Adriana	Shashwat
Code	Formulated and coded threshold-based masking of weights in the classifier Discovered and implemented L0-based masking of classifier layers Discovered and implemented REPTILE meta-learning to learn classifier masks	Formulated and coded activation- and weight-based pruning of convolution layer filters Extended Adriana's implementation of L0-based masking for convolution layers Wrote basic data-loading and visualization interfaces for the various studies
Writeup	Wrote all sections pertaining to classifier masking Wrote all sections pertaining to REPTILE meta-learning Wrote the related work section Editing	Wrote all sections pertaining to filter pruning Wrote the introduction Typesetting in \LaTeX Editing
Overall	$\sim 70\%$ of total work done in this project	$\sim 30\%$ of total work done in this project

Acknowledgments

We would like to acknowledge the instructors (Roger Grosse and Jimmy Ba) and TAs of CSC2516 for providing a valuable learning experience through this course.

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.