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

Neural networks are commonly designed to have extremely large numbers of trainable parameters to obtain accurate models for complex problems. Yet the actual number of parameters needed to describe the model may be significantly smaller than the initial choice. Pruning is a commonly used strategy to reduce the size of an existing network by removing parameters that are relatively less important. This reduces memory and time requirements when deploying the model. Pruning has also been shown to improve the ability of a model to generalize. In this work, we study two classes of simple methods to prune an existing model, and extend those methods in the context of generalization, by iteratively training on multiple related datasets, while pruning the model. We show that pruning does indeed yield smaller and sparser models without serious detriment to model accuracy, and that pruning does, to some extent, have a beneficial impact on the model's ability to generalize.

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

CSC2516 S 2019: project report.

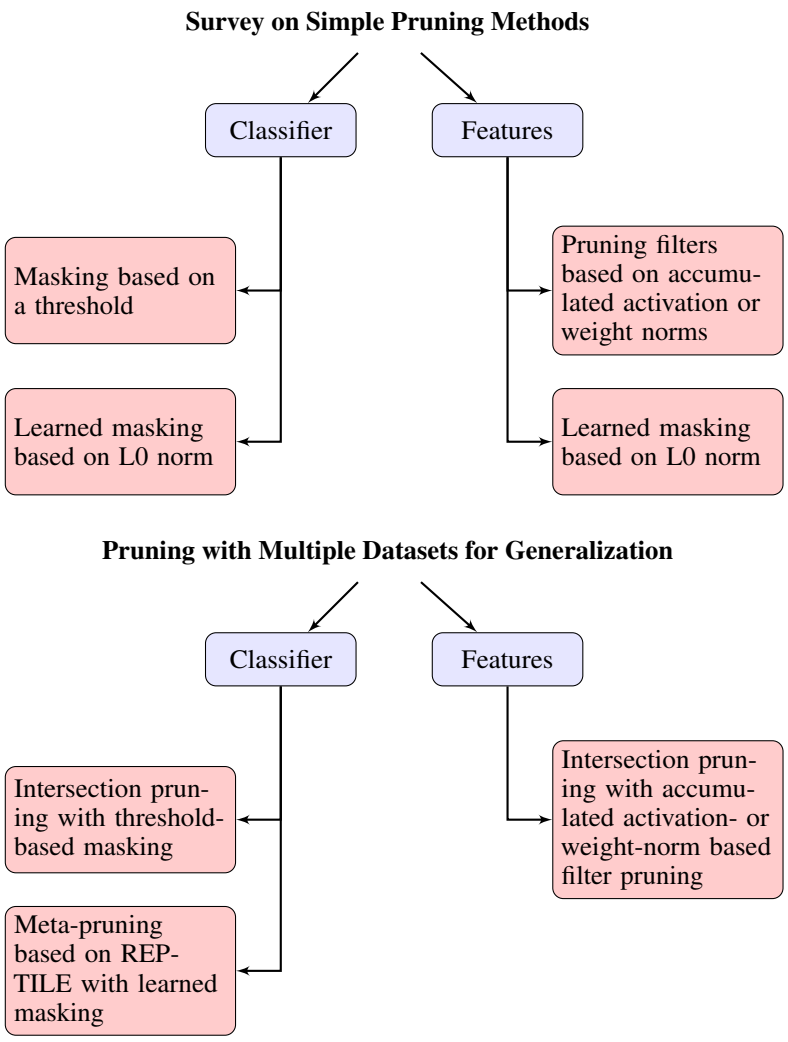

Figure 1 Outline of this paper.

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 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 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}:

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

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.

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

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

function COMPUTECONV2DWEIGHTNORMS(model, $\mathbf{n}_w$)
  for all i in *convolution layers of* 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(model, $\mathbf{n}_a$)
  for all i in *convolution layers of* 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($\mathbf{l}_{\text{idx}}, \mathbf{f}_{\text{idx}}$, model)
  $\mathbf{c}_{\text{idx}} \leftarrow \mathbf{l}_{\text{idx}}^{\text{th}}$ *conv layer of* 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* $\mathbf{w}_0$
  model $\leftarrow$ *updated conv layer* $\mathbf{c}_{\text{idx}}$ *at layer* $\mathbf{l}_{\text{idx}}$
  return model

function PRUNEALLCONV2DLAYERS($\mathbf{n}_a, \mathbf{n}_w, N_p$, model)
  for all i in *convolution layers of* 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* $\mathbf{n}_i$
    model $\leftarrow$ PRUNECONVLAYERS($i, \mathbf{f}_{\text{idx,i}}$, model)
  return model

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 described below. 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

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

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)

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.

We apply this idea both to masking classifier layers, as well as pruning the filters of convolution layers. For threshold masking of classifier weights, this method proceeds similarly to the description in Section 2.1.1. For pruning filter weights, this method proceeds similarly to the description in Section 2.1.3. However, in both cases the exception is that after every N_i epochs, the input data is resampled from a different dataset. The associated algorithm for filter pruning (also applicable to classifier threshold masking) is summarized in Algorithm 4.

2.2.2 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

3 Related Work

A wide variety of pruning approaches have been developed to improve network generalization and reduce computational cost while preserving accuracy. Pruning was first attempted by LeCun et al. (1990) with Optimal Brain Damage. They proposed removing weights based on their impact on the training error, which they calculate using the diagonal second derivatives of the loss. Removing

off-diagonal weights was later shown, by Hassibi et al. (1993), to allow for more extensive pruning (up to 90% weight reduction) with negligible impacts on accuracy. However, computing the Hessian is usually computationally infeasible for modern deep neural networks, although these networks can show extensive redundancy in weights according to Denil et al. (2013).

The majority of modern pruning algorithms build on these ideas and prune by estimating the impact on loss of weights, units, or filters in more computationally efficient ways. Han et al. (2015) achieved this by imposing an L2 penalty, and removing weights below an arbitrary cutoff, and re-training weights iteratively. Although the majority of weights occur in the fully connected layers of a CNN, pruning CNN filters rather than weights can be more effective at reducing model training time and size. Molchanov et al. (2017) estimate the importance of units using a Taylor series approximation of the derivative of the loss without the unit in question. They show this can be approximated by the gradient of the loss with respect to that unit. In our work, we use the approach of Han et al. (2015) (which we refer to as threshold pruning) to mask out weights, and extend the concept to filters. We also investigate whether threshold pruning can be applied periodically during training instead of on a fully trained network to improve the resulting network's generalizability.

There have been limited efforts on pruning during training, rather than as a post-processing step after training. This is a difficult task because of the intractability of differentiating through binary functions. One method by Louizos et al. (2018) provides a way to transform the discrete problem of masking the weights of a network into a continuous problem, allowing computation of derivatives. This mask can then be applied to network layers to sparsify the weight matrices, and has been implemented in this work. We have further extended this idea in the context of learning from multiple datasets.

There has also been an effort by Gordon et al. (2018) to automate the process of optimizing and pruning a network by iteratively shrinking and expanding it, to remove parameters that are less important to the problem at hand. However, they do not leverage multiple datasets, but show great promise in making networks faster and smaller without hurting accuracy.

Our motivation to leverage multiple datasets to develop "meta-pruning" approaches was derived from previous works in meta-learning. One such approach was MAML by Finn et al. (2017), where the initial weights of a network are learned in an outer iteration over consecutive training runs. However, this approach requires invasively altering the way gradients are computed within the model being optimized, and is not easy to implement given an existing model. A simpler approach, Reptile, was proposed by Nichol et al. (2018) to learn the initial weights of a network based on a finite difference-type approximation of the gradients of that layer. We have also implemented this, and extended it in the context of network pruning by combining it with the work of Louizos et al. (2018).

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 Colaboratory, 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 maintain training accuracy compared to the baseline model, despite the 74% reduction in network size. The validation accuracy, on average, is not only similar to that of the baseline, but is slightly more stable across epochs. It is clear that in all cases, there is a difference between training and validation accuracy of approximately at least 10% on average. However, towards the end of 50 epochs, the difference between training and validation accuracy in the baseline is noticeably higher (about 15%) than in the pruned models (still about 10%), which reinforces the idea that pruned networks may generalize better, as hypothesized. It is confirmed that the pruned network performs at least as well as the original network, but with significantly fewer weights, which also reinforces the importance and usefulness of pruning. It is also important to note that in all curves, the performance on validation in consistently lower than on the training data, which motivates the need to develop ways for the model to generalize better. The fluctuations in all curves is most likely because a relatively small training set (2000 images) was used due to limited computational resources available through Google Colaboratory.

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**: In this study, 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.

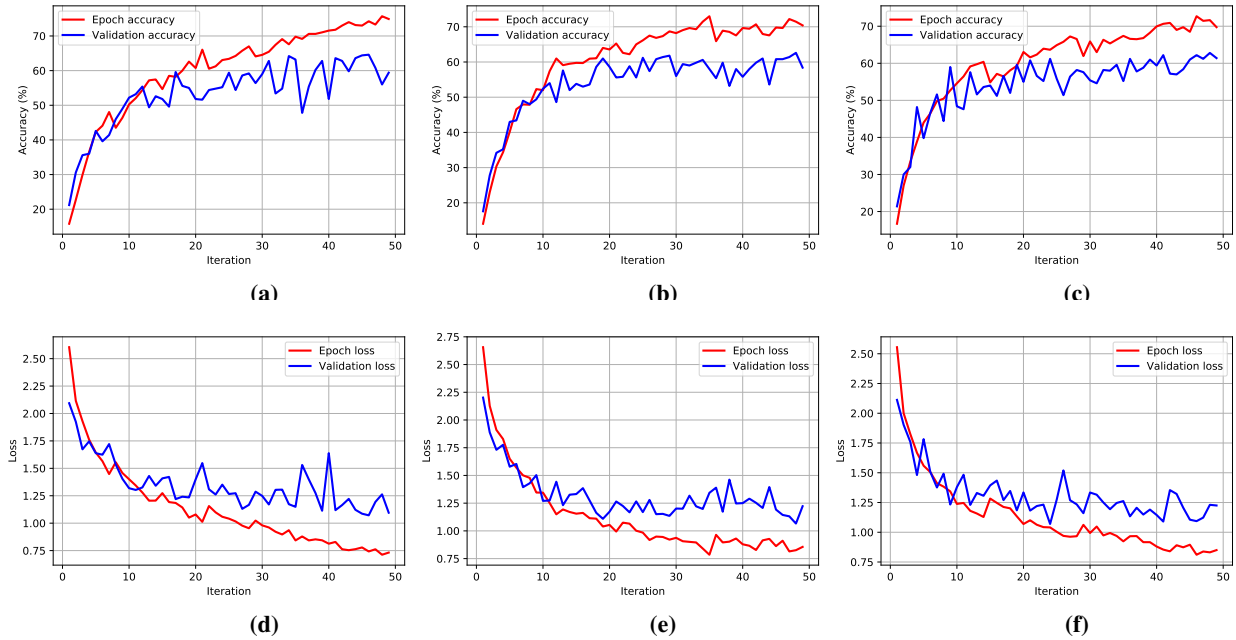

Figure 2 Filter pruning on a single dataset: Accuracy and loss of pre-trained VGG16 during fine-tuning on 5% of CIFAR-10 data (2000 training images, 500 validation images) for 50 epochs. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Middle column: activation-based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

Unlike 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. However, the validation accuracy is only $\sim$ 3% lower than that obtained with the baseline model, on average, but with 74% of the original network's filters. Although a larger proportion of the total CIFAR-10 dataset is represented during training, due to iterative subsampling, there is still a significant difference between training and validation accuracy in the baseline model (approximately 10%). However, the main result of this study is that in the pruned networks, performance on both training and validation data are very well balanced with respect to one another. This is a clear indicator that, as hypothesized, the pruned network shows better potential to generalize well, compared to the original network. The pruned network loss and accuracy are again much less stable over epochs than the original model, while the baseline model is much more stable. The fluctuations in the pruned model curves may be improved by pruning more gradually, for a larger number of epochs, and / or by using more sophisticated algorithms to pick which filters to prune.

To further stress-test the impact of pruning the model more aggressively, we repeated the above study, subsampling 5% of available training data from CIFAR-10, for a total of 150 epochs. 5% of the total network filters were pruned every 10 epochs. The total number of filters was reduced by 50%, from 4224 to 2117. As a baseline, the same study was performed without pruning. The purpose was to see how far we can push the reduction in the number of filter, and the results are shown in Figure 4. The jumps in accuracy correspond to resampling the dataset every 10 epochs. The training accuracy plateaus around 68% in the pruned model, and is again closely tracked by the validation accuracy. The baseline model provides on average 5% percent better accuracy than the pruned model, but its validation accuracy is still within $\sim$ 1–3% of the pruned model, which again indicates better generalization. Additionally, it is clear that the baseline model starts to overfit, as evidenced by the increasing difference between training and validation accuracy. However, this is not true for the pruned model; its validation accuracy consistently remains nearly as good

as the training accuracy, further indicating better generalization. Furthermore, it took the pruned model 6.75 minutes to train over final set of 10% epochs, while the baseline model took approximately 14.5 minutes per 10 epochs. Thus, it is to be expected that the pruned model will be at least twice as fast as the original upon deployment, in addition to potentially improved generalizability.

- **Iterative sampling from multiple datasets**: In this study, 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 5.

  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. Two major observations can be made from these results:

  1. 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 it encountered, the model trains on five other datasets in between (with possibly a completely different 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 be 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.
  2. Both the pruned and original baseline models perform equally well in general; in fact it requires careful observation of the curves to see where they differ in their performance. However, the total number of convolution filters in the pruned model, after 120 epochs, was reduced from 4224 to 2449, which is a 42% reduction in the size of the network, not counting the classifier layer.

  In summary, these results indicate that pruning a network to 58% of its original size can be achieved while maintaining its performance, and also maintaining or possibly improving its ability to generalize, compared to the baseline model.

5 Limitations

Some of the major limitations of the proposed methodologies are:

- Relatively simple metrics are used to determine what to prune, and the gradients of the loss with respect to network parameters have not been taken into account, unlike existing works in literature.
- The process of repeatedly sampling from different datasets would work much better if there was a larger overlap between their image categories, and were more closely related. Since

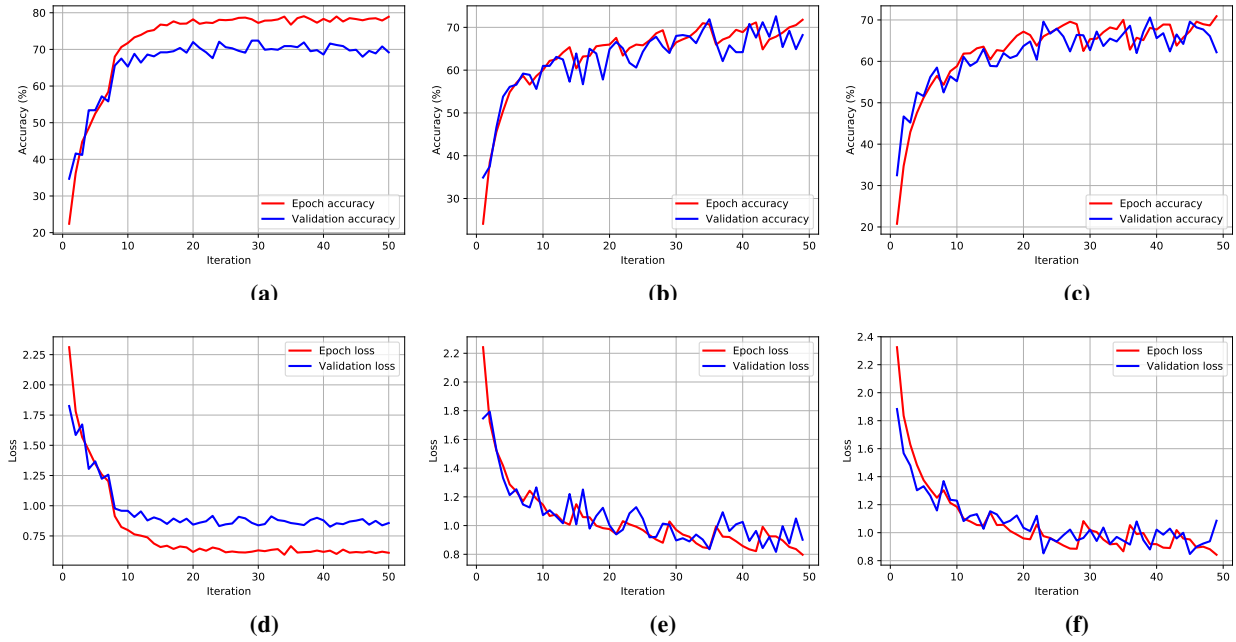

Figure 3 Iterative subsampling from CIFAR-10: Accuracy and loss of pre-trained VGG16 during fine-tuning. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Middle column: activation-based pruning of 5% of all filters every 7 epochs. Right column: weight-based pruning of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

we only used datasets easily available via `torchvision`, the choices were limited. However, even if the datasets were more closely related, they could also just be combined into one large dataset, so the benefit of resampling seems quite limited.

- In threshold masking, one limitation is that weights that are masked once can not come back. We attempted to alleviate this via meta-pruning, but it did not seem to work very well.
- We have not thoroughly quantified network performance with pruning in terms of memory and power consumption.
- The only GPUs we had access to were through Google Colaboratory, which limited the amount of training data we could use within the time constraints.
- We have only explored VGG16 - it would be interesting to study pruning on other networks, particularly those that rely on dropout.

6 Conclusions

In this work, we have surveyed simple techniques to prune large neural networks, using VGG16 as our test model. Due to the potential for pruning to improve network generalization, we have also attempted to combine pruning with the concept of meta-learning in two ways: one is by iteratively sampling from different but related datasets ("intersection pruning"), and the other is by learning mask matrices which are applied to classifier layers ("meta-pruning"). We found that pruning does indeed allow a significant reduction in network size with a corresponding test-time speed-up. However, in the case of intersection pruning of filter weights, this comes with up to 10% reduction in training accuracy, but without a significant reduction in validation accuracy. This indicates potentially better generalization properties of the pruned model. It was also observed that when sampling from multiple datasets, pruning filters has a very minor effect on both training and validation accuracy.

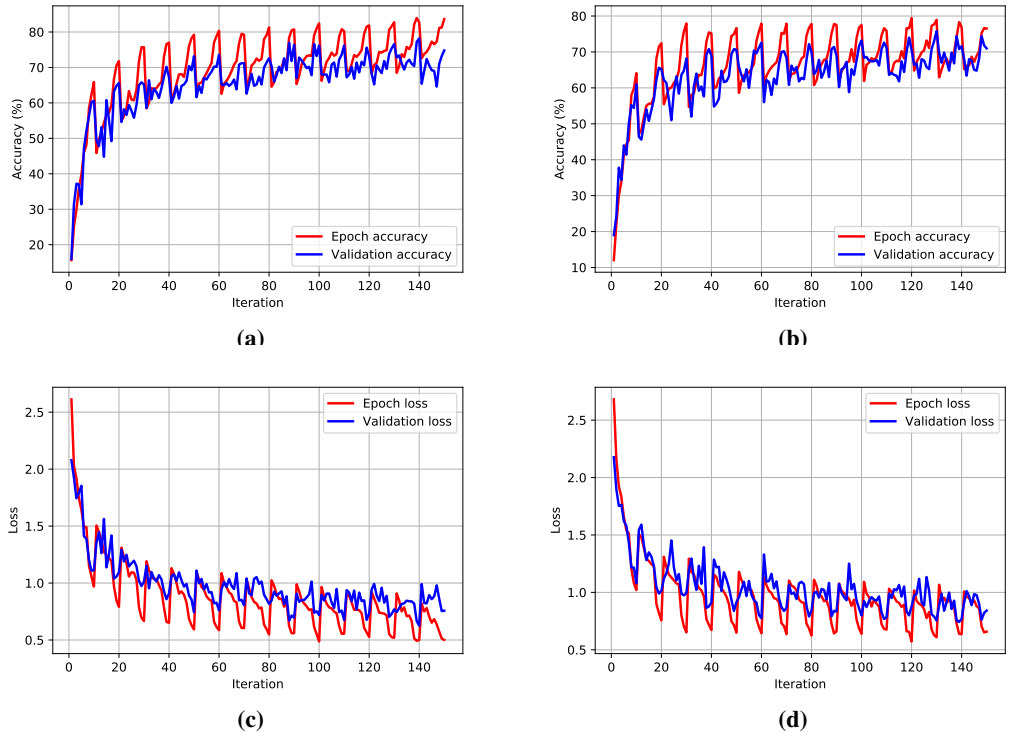

Figure 4 Iterative subsampling from CIFAR-10: Accuracy and loss of pre-trained VGG16 during fine-tuning. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Right column: weight-based pruning of 5% of all filters every 10 epochs.

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.

Acknowledgments

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

References

Denil, M., Shakibi, B., Dinh, L., Ranzato, M. A., and de Freitas, N. (2013). Predicting Parameters in Deep Learning. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2148–2156. Curran Associates, Inc.

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.

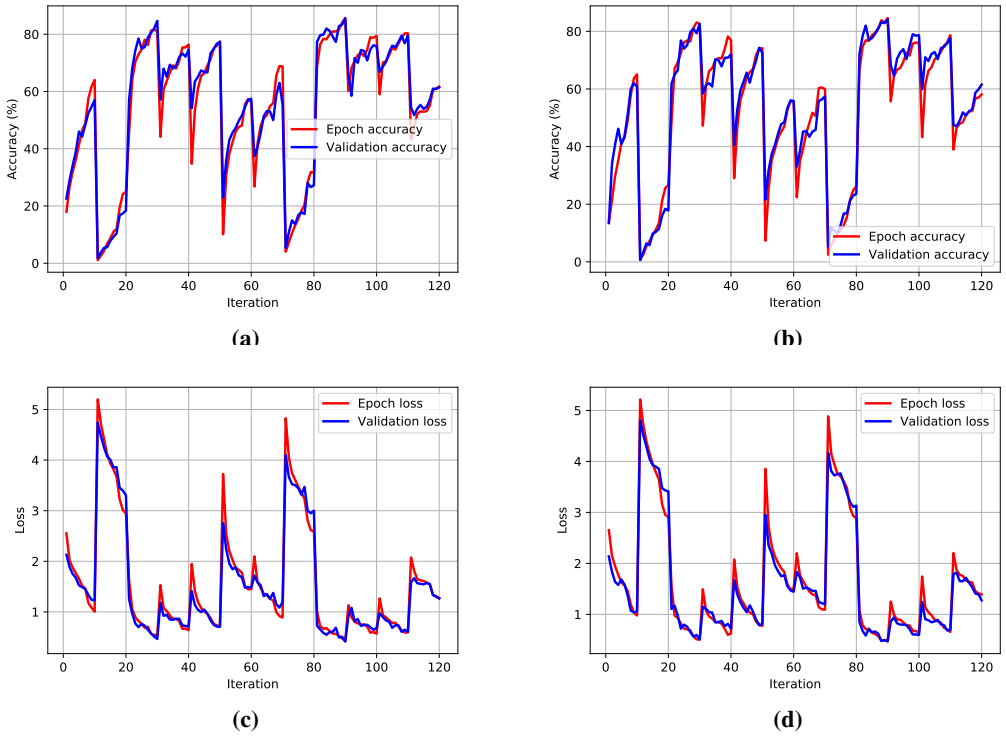

Figure 5 Iterative sampling from multiple datasets: Accuracy and loss of pre-trained VGG16 during fine-tuning. Top row: accuracy. Bottom row: loss. Left column: baseline model with no pruning. Right column: weight-based pruning of 5% of all filters every 6 epochs.

Table 1 Contributions of each author.

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

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.

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

Louizos, C., Welling, M., and Kingma, D. P. (2018). Learning Sparse Neural Networks through L0 Regularization. In *International Conference on Learning Representations*.

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

Nichol, A., Achain, J., and Schulman, J. (2018). On First-Order Meta-Learning Algorithms. *CoRR*, abs/1803.02999.