DropNet: Reducing Neural Network Complexity via Iterative Pruning

John Tan Chong Min [1] **Mehul Motani** [1]

Abstract

Modern deep neural networks require a significant amount of computing time and power to train and deploy, which limits their usage on edge devices. Inspired by the iterative weight pruning in the Lottery Ticket Hypothesis (Frankle & Carbin, 2018), we propose *DropNet*, an iterative pruning method which prunes nodes/filters to reduce network complexity. *DropNet* iteratively removes nodes/filters with the lowest average post-activation value across all training samples. Empirically, we show that *DropNet* is robust across diverse scenarios, including MLPs and CNNs using the MNIST and CIFAR datasets. We show that up to 90% of the nodes/filters can be removed without any significant loss of accuracy. The final pruned network performs well even with reinitialization of the weights and biases. *DropNet* also has similar accuracy to an oracle which greedily removes nodes/filters one at a time to minimise training loss, highlighting its effectiveness.

1. Introduction

The surprising effectiveness of neural networks in domains such as image recognition in ImageNet (Russakovsky et al., 2015) has inspired much recent research. Current state-of-the-art deep models include Transformers (Vaswani et al., 2017) for language modelling, InceptionNet (Szegedy et al., 2015) for image modelling, and ResNets (He et al., 2016) which include over 100 layers. In fact, the number of configurable parameters per model has risen significantly from hundreds to tens of millions. The increased computational complexity required for modern neural networks has made deployment in edge devices challenging.

Previous Work on Complexity Reduction: Current methods of reducing complexity include quantization to 16-bit (Gupta et al., 2015), group L1 or L2 regularization (Alemu et al., 2019), node pruning (Castellano et al., 1997; Zhang & Qiao, 2010; Alvarez & Salzmann, 2016; Wen et al., 2016), filter pruning for CNNs (Li et al., 2016; Wen et al., 2016; Alvarez & Salzmann, 2016; He et al., 2017; Liu et al., 2017), weight pruning using magnitude-based methods (Han et al., 2015) or second-order Hessian-based methods such as Optimal Brain Damage (LeCun et al., 1990) or Optimal Brain Surgeon (Hassibi et al., 1993).

Previous Work on Node/Filter Pruning: For node pruning, previous work includes introducing regularization terms based on input weights using group lasso to the loss function (Alvarez & Salzmann, 2016; Wen et al., 2016) and selecting the least important node to drop based on mutual information (Zhang & Qiao, 2010). In CNNs, previous work includes layer-wise pruning such as layer-wise lasso regression on filters (He et al., 2017), and global pruning methods such as pruning filters with the lowest sum of absolute input weights globally (Li et al., 2016), using second-order Taylor expansion to prune unimportant filters (Molchanov et al., 2016; Lin et al., 2018), introducing structured sparsity using a particle filter approach (Anwar et al., 2017), or to repeatedly perform an L1 regularization of the batch normalization layer's scaling factor in CNNs (Liu et al., 2017).

Previous Work on Iterative Pruning: Most techniques that can perform one-shot pruning can also be applied recursively using iterative pruning. It has been shown that iterative pruning achieves better performance than one-shot pruning (Li et al., 2016). More notably, iterative pruning with reinitialization can reduce parameter counts by over 90% (Frankle & Carbin, 2018). Such iterative pruning methods shed new insight into how more effective pruning approaches might be developed.

Comparison with Similar Work: *DropNet* iteratively removes nodes/filters with the lowest average post-activation values across all training samples. Similar work to ours include removing nodes with the highest average percentage of zero activation values across a validation set - Average Percentage of Zeros (APoZ) (Hu et al., 2016), which measures sparsity of a node's activations. In contrast, our method (i) utilizes average magnitude of post-activation values, and (ii) does so over the training set. Another simi-

[1]Department of Electrical and Computer Engineering, National University of Singapore. Correspondence to: John Tan Chong Min <e0441892@nus.edu.sg>, Mehul Motani <motani@nus.edu.sg>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

lar metric is to prune channels to a filter using variance of post-activation values (Polyak & Wolf, 2015). Our method instead prunes the entire filter using average post-activation values.

Comparison with Dropout: Dropout (Srivastava et al., 2014) randomly drops a fraction p of nodes during training, but keeps the entire network intact during test time. *DropNet*, however, drops nodes/filters permanently during training time and test time.

Our Contributions:

- We propose *DropNet*, an iterative node/filter pruning approach with reinitialization, which iteratively removes nodes/filters with the lowest average post-activation value across all training samples (either layer-wise or globally) and, hence, reduces network complexity. To the best of our knowledge, our method is the first to prune both MLPs and CNNs based on the average post-activation values of nodes/filters, which utilizes both the information about the input weights as well as the input data to make an informed selection of the nodes/filters to drop.
- *DropNet* achieves a robust performance across a wide range of scenarios compared to several benchmark metrics. *DropNet* achieves similar performance to an *oracle* which greedily removes nodes/filters one at a time to minimise training loss.
- *DropNet* does not require special initialization of weights and biases (unlike (Frankle & Carbin, 2018)). It is shown in subsequent experiments that a random initialization of the pruned model will do just as well as original initialization. This means the architecture pruned by *DropNet* can be readily deployed using off-the-shelf machine learning libraries and hardware.

2. *DropNet* Algorithm

In this section, we describe the *DropNet* algorithm and discuss its properties.

Similar to the Lottery Ticket Hypothesis (Frankle & Carbin, 2018), which iteratively drops weights with reinitialization, *DropNet* applies iterative dropping for nodes/filters with reinitialization. Next, we state the optimization problem.

Model: Consider a dense feed-forward neural network $f(x; n)$ with initial configuration of weights and biases $\theta = \theta_0$ and initial configuration of nodes/filters n. Let f reach minimum validation loss l with test accuracy a when optimizing with stochastic gradient descent (SGD) on a training set. Consider also training $f(x; m \odot n)$ with a mask $m \in \{0, 1\}^{|n|}$ on its nodes/filters such that its initialization is $f(x; m \odot n)$, where $m \odot n$ is an element-wise multiplication between m and n. Let f with the mask reach minimum validation loss l' with with test accuracy a'.

Algorithm 1 Iterative Pruning Algorithm

Input: Neural network with initial state θ_0, inital nodes/filters n, pruning metric.
Hyperparameters: Training iterations j, pruning fraction $p \in (0, 1]$, maximum loss factor $\kappa \in (0, 1]$
Initialize starting mask $m = \{1\}^{|n|}$
repeat
1. Revert network to initial state θ_0
2. Apply mask to nodes/filters: $f(x; m \odot n)$
3. Train network for at most j iterations until early stopping
4. Apply pruning metric to choose a fraction p of nodes/filters to drop and update m

until validation accuracy $a' \leq \kappa a$
Run steps 1 to 3

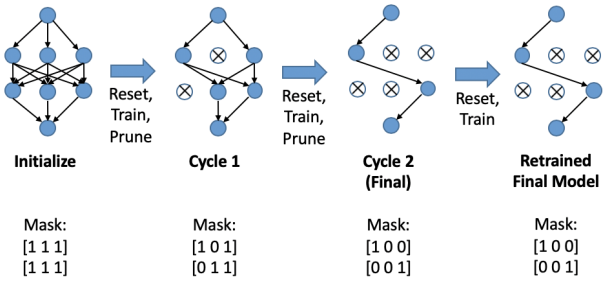


Figure 1. Illustration of DropNet algorithm as described in Algorithm 1. The mask is initially set to all 1s, meaning all nodes/filters are present. As the training cycle progresses, more and more nodes/filters are dropped. The final model at cycle 2 is then reset to initial state and retrained to give the final parameters.

Problem: Find a subnetwork $f(x; m \odot n)$ such that $a' \approx a$ *(similar accuracy)* and $||m||_0 \ll n$ *(fewer parameters)*.

Algorithm: The proposed iterative pruning algorithm is shown in in Algorithm 1. *DropNet* applies Algorithm 1 with the following metric: the lowest average post-activation value across all training samples (either layer-wise or globally). An example of the training cycle using Algorithm 1 is shown in Fig. 1.

Expected Absolute Value of a Node: Unlike weights, which can be of the same value for all training samples, nodes will change their post-activation values according to the input training samples. Hence, we use a node's *expected absolute value* across all training samples $(x_1, x_2, ..., x_t)$ to evaluate its importance. For each node $a_i, i \in \mathbb{Z}^+, 1 \leq i \leq n$, out of all n nodes in the network, we have its *expected absolute value* as:

$$E(a_i) = \frac{1}{t} \sum_{j=1}^{t} |V(a_i|x_j)|,$$

where $V(a_i|x_j)$ is the post-activation value of the node a_i with input sequence x_j.

Expected Absolute Value of a Filter: For CNNs, the filters are comprised of a set of constituent nodes. Choosing which filter to drop is thus equivalent to choosing a set of constituent nodes to drop. Hence, in order to evaluate the importance of each filter $f_i, i \in \mathbb{Z}^+, 1 \leq i \leq n$, out of all n filters in the network, we take the average of the *expected absolute value* of all its constituent nodes $a_1, a_2, ..., a_r, r \in \mathbb{Z}^+$. That is:

$$E(f_i) = \frac{1}{r} \sum_{j=1}^{r} |E(a_j)|,$$

where $E(f_i)$ is the *expected absolute value* of the filter f_i across all training samples $x_1, x_2, ..., x_t$.

Intuition: We propose two reasons for the competitiveness of dropping nodes/filters with lowest average post-activation value across all training samples. (i) Firstly, Rectified Linear Unit ($ReLU$) activation will lead to inactive nodes which do not "fire" once the value of node reaches 0 or below, due to the $ReLU(x)$ activation function setting the post-activation value of a node to be 0 once $x \leq 0$. Hence, if a node does not fire most of the time, it will have a low expected absolute value and removing it will affect only a small amount of classifications when its post-activation value is non-zero. (ii) Secondly, during backpropagation, the input weights of a node with low expected absolute value will be updated by only a small amount, which means that the node is less adaptive to learning from the inputs. Removing these less adaptive nodes should impact classification accuracy less than removing more adaptive ones.

3. Methodology

In order to evaluate the effectiveness of the *DropNet* algorithm, we test it empirically using MLPs and CNNs on MNIST (LeCun et al., 2010) and CIFAR-10 (Krizhevsky et al., 2009) datasets.

3.1. Pruning Metrics

The pruning metrics we consider are listed in Table 1. At the end of each training cycle in Algorithm 1, we use the metric to evaluate the importance score for each node/filter, and drop the nodes/filters with the lowest importance scores. Note that ties are broken randomly.

The `minimum`, `maximum` and `random` metrics prune a fraction p of nodes/filters globally. The `minimum` metric drops a fraction p the nodes/filters globally with the lowest post-activation values. The `maximum` metric serves as a comparison to the `minimum` metric to compare the effectiveness of the metric. The `random` metric, which prunes a fraction p of nodes randomly, serves as a control.

Table 1. Pruning Metrics

METRIC	IMPORTANCE SCORE	TYPE
MINIMUM	$E(a_i)$ OR $E(f_i)$	GLOBAL
MAXIMUM	$-E(a_i)$ OR $-E(f_i)$	GLOBAL
RANDOM	0	GLOBAL
MINIMUM_LAYER	$E(a_i)$ OR $E(f_i)$	LAYER-WISE
MAXIMUM_LAYER	$-E(a_i)$ OR $-E(f_i)$	LAYER-WISE
RANDOM_LAYER	0	LAYER-WISE

We consider layer-wise pruning metrics. These metrics are termed `minimum_layer`, `maximum_layer` and `random_layer`, and they prune a fraction p of nodes/filters layer-wise.

DropNet utilizes Algorithm 1 with either the `minimum` or `minimum_layer` metrics. As will be shown in our experimental results, these metrics prove to be quite competitive for different scenarios.

3.2. Experiment Details

Train-Validation-Test Split: For MNIST, the dataset is split into 54000 training, 6000 validation and 10000 testing samples. For CIFAR-10, the dataset is split into 45000 training, 5000 validation and 10000 testing samples.

Pre-processing: The input pixel values are scaled to be between 0 and 1.

Activation Function: The model activation functions are all *ReLU*, except the final classification layer where it is softmax in order to choose one out of multiple classes.

Optimization Function: The optimization function used is SGD with a learning rate of 0.1.

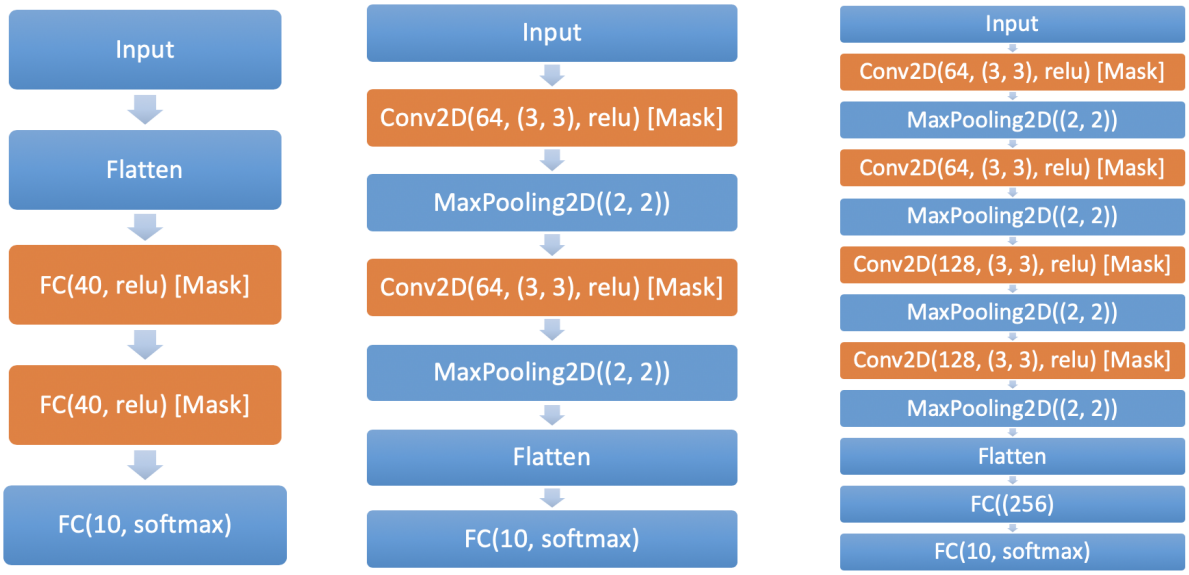
Loss Function: The loss function used is cross entropy.

Training Runs: The experiments are repeated over 15 runs, each with a different initial random seed. To serve as comparison between the various metrics, the average accuracy against the fraction of nodes/filters remaining across all 15 runs are plotted, together with the error bars denoting the 95% confidence interval.

Training Cycles: The masks are applied at the start of each training cycle, which comprises 100 epochs, with early stopping using validation loss with patience of 5 epochs. Over each training cycle, a fraction $p = 0.2$ of the nodes are dropped.

3.3. Model Details

The experiments are performed on a variety of network models. We use three types of feed-forward architectures: two fully-connected (FC) hidden layers of nodes (Model A),



Model A Model B Model C

Figure 2. Models of the neural network architectures considered. Varying numbers of initial hidden nodes/filters are used in different experiments, but the baseline architectures remain the same. The layers where the masks are applied are written with a postfix '[Mask]', and shown in orange. The CNN models (Model B and C) are variants of the VGG architecture (Simonyan & Zisserman, 2014). ***Model A: FC40 - FC40 (Left)***: This is a network with two fully-connected (FC) hidden layers, each with 40 nodes. The mask is applied after each hidden layer. ***Model B: Conv64 - Conv64 (Middle)***: This is a network with a two 2D convolutional layers each comprising of 64 filters of size 3x3 with 'same' padding. The mask is applied after the convolutional layer and before the MaxPooling2D layer. ***Model C: Conv64 - Conv64 - Conv128 - Conv128 (Right)***: This is a network with a four 2D convolutional layers. The first two convolutional layers have 64 filters, while the next two convolutional layers have 128 filters. The filter is of size 3x3 with 'same' padding. The mask is applied after the convolutional layer and before the MaxPooling2D layer.

two 2D convolutional (Conv) layers (Model B) and four 2D convolutional layers (Model C). The model architectures are detailed in Fig. 2.

4. Results

4.1. MLP - MNIST

Q1. Can DropNet perform robustly well on MLPs of various starting configurations?

To address this question, we conduct the experiments on different versions of Model A on MNIST, listed as follows:

1.1) **Model A: FC40 - FC40**. The plot of test accuracy against fraction of nodes remaining for various metrics is shown in Fig. 3.
1.2) **Model A: FC20 - FC40**. The plot of test accuracy against fraction of nodes remaining for various metrics is shown in Fig. 4.
1.3) **Model A: FC40 - FC20**. The plot of test accuracy against fraction of nodes remaining for various metrics is shown in Fig. 5.

As the trends for training, validation, and test accuracies are similar, we only show the plots for test accuracy.

For 1.1), it can be seen (Fig. 3) that the `minimum_layer` performs the best, followed closely by `minimum`, then `random_layer`, `maximum_layer`, `random` and lastly `maximum`. The `maximum` metric performs poorly when the fraction of nodes remaining is 0.5 and below.

For 1.2), it can be seen (Fig. 4) that the `minimum` metric performs the best, followed by the layer-wise and random metrics, and lastly the `maximum` metric.

For 1.3), it can be seen (Fig. 5) that the `minimum` and `minimum_layer` are both competitive, followed by `random_layer`, `random`, `maximum_layer`, and lastly `maximum` metric. `minimum` performs well for all fractions of nodes remaining except between 0.1 and 0.3 where `minimum_layer` performs slightly better. The `maximum` metric can be seen to be consistently poor when the fraction of nodes remaining is 0.6 and below.

Evaluation: The results indicate that, for fully connected networks, when the hidden layer sizes are equal (i.e., no bottleneck layer), the `minimum_layer` metric is competitive. When the hidden layer sizes are unequal (i.e., there potentially exists a bottleneck layer), the `minimum` metric is competitive. This shows that `minimum` and `minimum_layer`

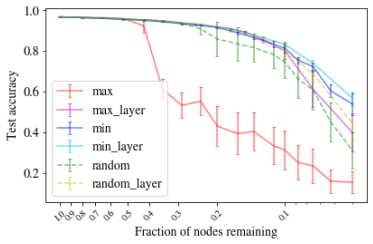


Figure 3. Test accuracy vs. fraction of nodes remaining for various metrics in Model A: FC40 - FC40 on MNIST

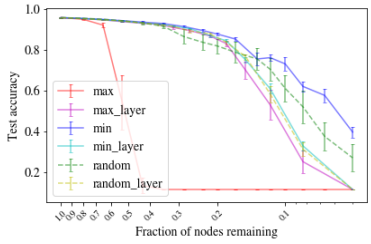


Figure 4. Test accuracy vs. fraction of nodes remaining for various metrics in Model A: FC20 - FC40 on MNIST

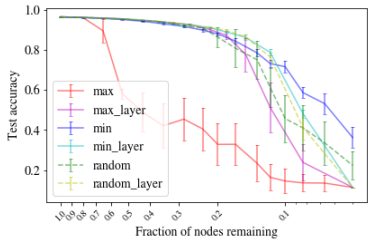


Figure 5. Test accuracy vs. fraction of nodes remaining for various metrics in Model A: FC40 - FC20 on MNIST

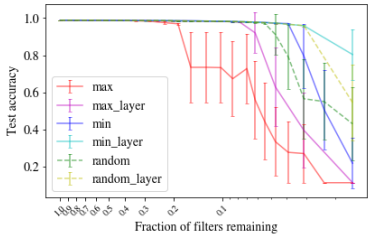


Figure 6. Test accuracy vs. fraction of filters remaining for various metrics in Model B: Conv64 - Conv64 on MNIST

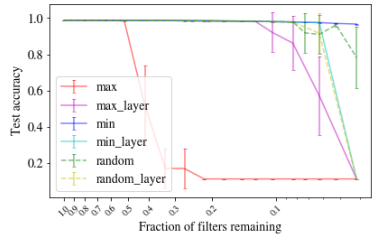


Figure 7. Test accuracy vs. fraction of filters remaining for various metrics in Model B: Conv32 - Conv64 on MNIST

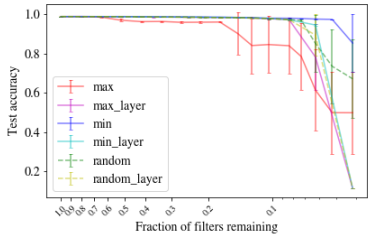


Figure 8. Test accuracy vs. fraction of filters remaining for various metrics in Model B: Conv64 - Conv32 on MNIST

are competitve metrics for **Model A**. Using *DropNet*, we are able to reduce the number of nodes by 60% or more without significantly affecting model accuracy, highlighting its effectiveness in reducing network complexity.

4.2. CNN - MNIST

Q2. Can DropNet perform robustly well on CNNs of various starting configurations?

To address this question, we conduct experiments on different versions of Model B on MNIST, listed as follows:

2.1) **Model B: Conv64 - Conv64**. The plot of test accuracy against fraction of filters remaining for various metrics is shown in Fig. 6.
2.2) **Model B: Conv32 - Conv64**. The plot of test accuracy against fraction of filters remaining for various metrics is shown in Fig. 7.
2.3) **Model B: Conv64 - Conv32**. The plot of test accuracy against fraction of filters remaining for various metrics is shown in Fig. 8.

As the trends for training, validation, and test accuracies are similar, we only show the plots for test accuracy.

For 2.1), it can be seen (Fig. 6) that the `minimum_layer` metric performs the best, followed by `random_layer`, `minimum`, `random`, `maximum_layer`, and lastly `maximum` metric. The `maximum` metric can be seen to be consistently poor when the fraction of filters remaining is 0.3 and below.

For 2.2), it can be seen (Fig. 7) that the `minimum` metric performs the best, followed by `minimum_layer`, `random_layer`, `random`, `maximum_layer`, and lastly `maximum` metric. The `maximum` metric can be seen to be consistently poor when the fraction of filters remaining is 0.5 and below. The random metric is in between the performance of the `minimum` and `maximum` metrics.

For 2.3), it can be seen (Fig. 8) that the `minimum` metric performs the best, followed by `minimum_layer`, `random`, `random_layer`, `maximum_layer`, and lastly `maximum` metric. The `maximum` metric can be seen to be consistently poor when the fraction of filters remaining is 0.2 and below. The performance of the random metrics is in between the performance of the `minimum` and `maximum` metrics.

Evaluation: The findings are similar to Section 4.1. The results indicate that for convolutional layers, when the hidden layer sizes are equal (i.e., no bottleneck layer), the `minimum_layer` metric is competitive. When the hidden layer sizes are unequal (i.e., there potentially exists a bottleneck layer), the `minimum` metric is competitive. This shows that `minimum` and `minimum_layer` are competitive metrics for **Model B**. Using *DropNet*, we are able to reduce the number of filters by 90% or more without

significantly affecting model accuracy, highlighting its effectiveness in reducing network complexity.

4.3. CNN - CIFAR-10: Model C

Q3. Can DropNet perform well on a larger dataset like CIFAR-10?

To address this question, we conduct an experiment using Model C on CIFAR-10, listed as follows:

3.1) **Model C: Conv64 - Conv64 - Conv128 - Conv128**. The plot of test accuracy against fraction of filters remaining for various metrics are shown in Fig 9.
3.2) **Model C: Conv128 - Conv128 - Conv256 - Conv256**. The plot of training accuracy and test accuracy against fraction of filters remaining for various metrics are shown in Fig. 10.

As the trends for training, validation, and test accuracies are similar, we only show the plots for test accuracy.

It can be seen (Figs. 9 and 10) that the `minimum_layer` metric performs the best, followed by `minimum`, `random_layer`, `random`, and `maximum_layer` and lastly `maximum` metric. The `minimum` and `minimum_layer` perform equally well when the fraction of filters remaining is 0.5 and above. The `maximum` metric can be seen to be consistently poor when the fraction of filters remaining is 0.5 and below. The `random` metric is in between the `minimum` and `maximum` metrics.

Evaluation: The results show that `minimum` and `minimum_layer` are both competitive when less than of half of the filters are dropped. Thereafter, `minimum_layer` performs significantly better. Using *DropNet*, we can reduce the number of filters by 50% or more without significantly affecting model accuracy, highlighting its effectiveness in reducing network complexity.

The results indicate that, for larger convolutional models like **Model C**, global pruning methods like the `minimum` metric may not be as competitive as layer-wise pruning methods like the `minimum_layer` metric.

4.4. CNN - CIFAR-10: ResNet18/VGG19

Q4. Can DropNet perform robustly well on even larger models such as ResNet18 and VGG19?

To address this question, we conduct an experiment using Algorithm 1 for ResNet18 and VGG19 on CIFAR-10, both following closely to the implementation in their respective papers (Simonyan & Zisserman, 2014; He et al., 2016). Details of the model are in the Supplementary Materials. Of note, ResNet18 is implemented without Batch Normalization, while VGG19 had a Batch Normalization before every MaxPooling2D layer.

The plot of test accuracy against fraction of filters remaining for various metrics for ResNet18 and VGG19 is shown in Figs. 11 and 12 respectively.

For ResNet18, it can be seen (Fig. 11) that the `minimum_layer` metric performs the best, followed by `minimum`, then `random_layer`, `random`, `maximum_layer` and and lastly `maximum` metric. The `minimum_layer` and `minimum` are both competitive.

For VGG19, it can be seen (Fig. 12) that the `minimum_layer` metric performs the best, followed by `random_layer`, `random`, `max_layer`, `minimum`, and and lastly `maximum` metric. The `minimum_layer` metric is the most competitive.

For both ResNet18 and VGG19, the `maximum` metric can be seen to be consistently poor when the fraction of filters remaining is 0.5 and below. The `maximum_layer` metric is consistently poor when the fraction of filters remaining is 0.2 and below.

Evaluation: The results show that for larger models, the `minimum_layer` is the most competitive. It can also be seen that with the exception of `minimum` and `maximum_layer`, the layer-wise metrics outperform the global metrics for larger models. This shows that there may be significant statistical differences between layers for larger models such that comparing magnitudes across layers may not be a good way to prune nodes/filters. That said, the `minimum_layer` can be seen to perform very well and consistently performs better than random, which shows promise that it is a good metric.

The `minimum` metric proves to be almost as competitive as `minimum_layer` for ResNet18, but performs worse than random for VGG19. This shows that the skip connections in ResNet18 help to alleviate some of the pitfalls of global metrics. Interestingly, the `minimum` metric tends to prune out some skip connections completely, which shows that certain skip connections are unnecessary. This means that *DropNet* using the `minimum` metric is able to automatically identify these redundant connections on its own.

Overall, using *DropNet*, we can reduce the number of filters by 80% or more without significantly affecting model accuracy, highlighting its effectiveness in reducing network complexity even in larger models.

5. Empirical Analysis - Oracle Comparison

Q5. How competitive is the DropNet algorithm compared to an oracle?

There are numerous node/filter pruning methods and algorithms available, hence, rather than comparing the performance of *DropNet* with these individual methods and

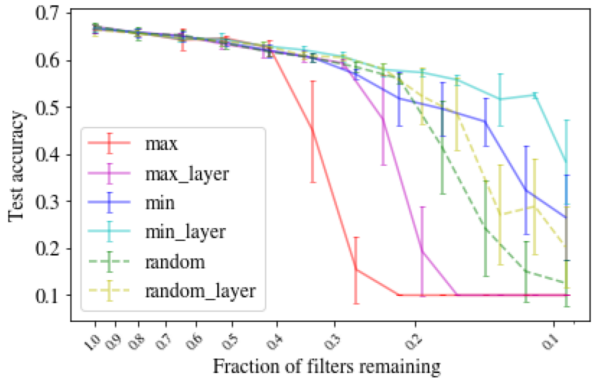


Figure 9. Test accuracy vs. fraction of filters remaining for various metrics in Model C: Conv64 - Conv64 - Conv128 - Conv128 on CIFAR-10

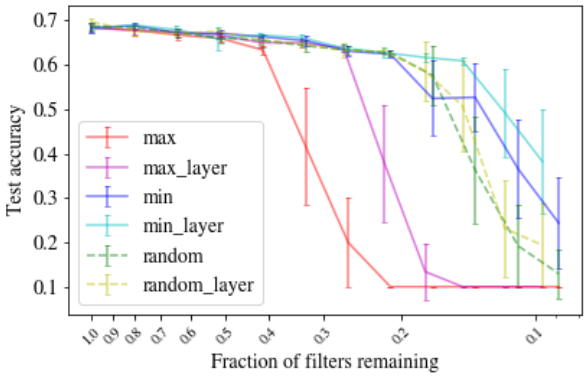


Figure 10. Test accuracy vs. fraction of filters remaining for various metrics in Model C: Conv128 - Conv128 - Conv256 - Conv256 on CIFAR-10

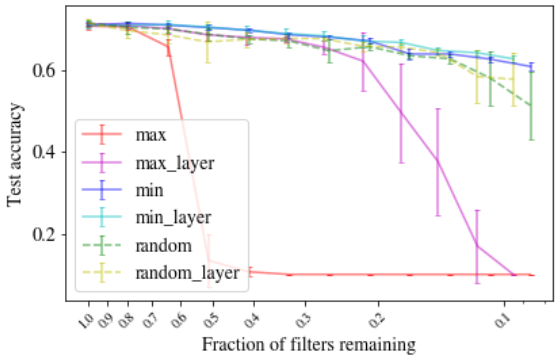


Figure 11. Test accuracy vs. fraction of filters remaining for various metrics in ResNet18 on CIFAR-10

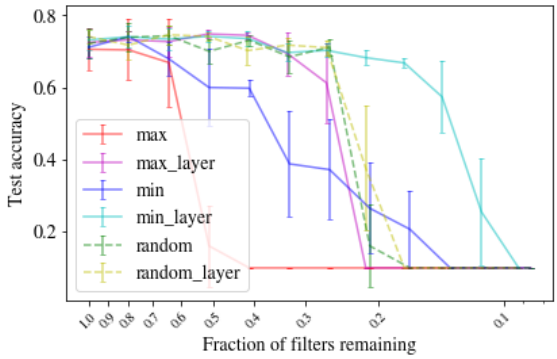


Figure 12. Test accuracy vs. fraction of filters remaining for various metrics in VGG19 on CIFAR-10

algorithms, we utilize an **oracle** in order to establish the competitiveness of *DropNet*. We define the oracle as the algorithm which **greedily** drops a node/filter out of all remaining node/filters available at every iteration of Algorithm 1 such that the overall training loss is minimized. In order to provide a fair comparison with the oracle, nodes/filters are also pruned one at a time when using the various metrics.

In order to perform a "stress" test on the various metrics compared to the oracle, we analyze their performance on a smaller scale model, listed as follows:

5.1) **Model A: FC20 - FC20**. The plot of test accuracy against fraction of nodes remaining when compared against an oracle on MNIST is shown in Fig. 13.

5.2) **Model B: Conv20 - Conv20**. The plot of test accuracy against fraction of filters remaining when compared against an oracle on MNIST is shown in Fig. 14.

For 5.1), it can be seen (Fig. 13) that the oracle performs the best, followed closely by `minimum`, then `random` and lastly `maximum` metrics. The `random` metric used here is actually the `random_layer` metric, as it provides a stronger baseline performance. Although not shown, `minimum_layer` has similar performance to `minimum`.

For 5.2), it can be seen (Fig. 14) that the oracle, `minimum` and `random` perform equally well. The worst performing is the `maximum` metric, with poor performance with 0.4 or less fraction of filters remaining.

Evaluation: The results indicate that overall, the `minimum` metric is competitive even to an oracle which minimizes training loss. This shows that the `minimum` metric is indeed a competitive criterion to drop nodes/filters.

We note that Algorithm 1 with a pruning metric runs in linear time. The oracle runs in polynomial time, as it has to iterate through all possible node/filter selections at the end of each iteration of Algorithm 1.

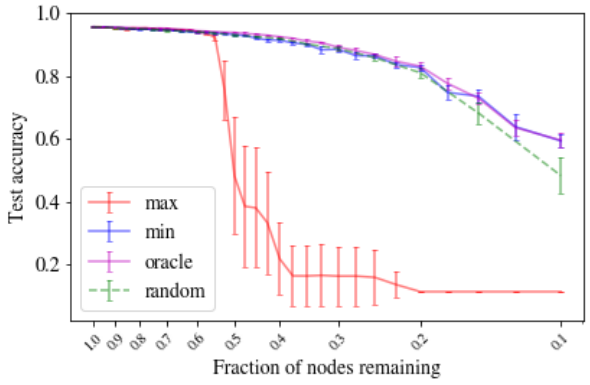


Figure 13. Test accuracy vs. fraction of nodes remaining when compared to an oracle in Model A: FC20 - FC20 on MNIST

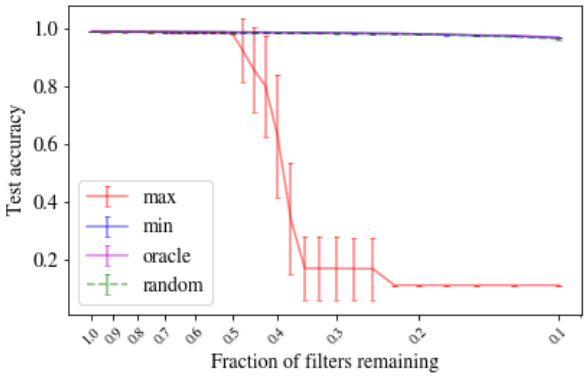


Figure 14. Test accuracy vs. fraction of filters remaining when compared to an oracle in Model B: Conv20 - Conv20 on MNIST

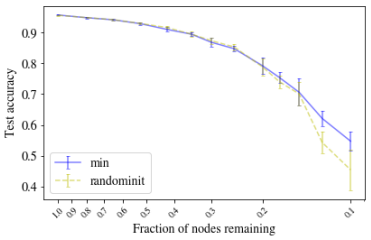


Figure 15. Test accuracy vs. fraction of nodes remaining for original initialization and random initialization in Model A: FC20 - FC20 on MNIST

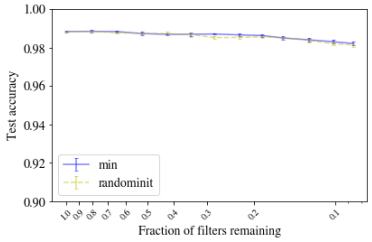


Figure 16. Test accuracy vs. fraction of filters remaining for original initialization and random initialization in Model B: Conv64 - Conv64 on MNIST

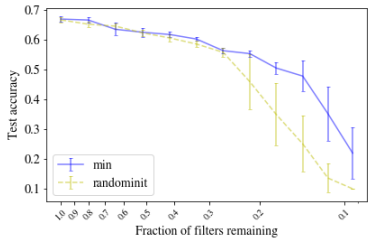


Figure 17. Test accuracy vs. fraction of nodes remaining for original init. and random init. in Model C: Conv64 - Conv64 - Conv128 - Conv128 on CIFAR-10

6. Empirical Analysis - Random Initialization

Q6. Is the starting initialization of weights and biases important?

We compare the performance of a network retaining its initial weights and biases θ_0 when performing iterative node/filter pruning, as compared to a network with the pruned architecture but with a random initialization (`randominit`). We conduct the following experiments:

6.1) **Model A: FC20 - FC20**. The plot of test accuracy against fraction of nodes remaining for original and random initialization is shown in Fig. 15.

6.2) **Model B: Conv64 - Conv64**. The plot of test accuracy against fraction of filters remaining for original and random initialization is shown in Fig. 16.

6.3) **Model C: Conv64 - Conv64 - Conv128 - Conv128**. The plot of test accuracy against fraction of filters remaining for original and random initialization is shown in Fig. 17.

It can be seen (Figs. 15, 16, and 17) that unlike the Lottery Ticket Hypothesis (see Figure 4 in (Frankle & Carbin, 2018)), *DropNet* does not suffer from loss of performance when randomly initialized for up to 70% to 80% of the nodes/filters being dropped.

Evaluation: This means that for *DropNet*, only the final pruned network architecture is important, and not the initial weights and biases of the network. One reason that the original initialization is not important may be because the learning rate is high enough (0.1) for the network to retrain sufficiently given just the model architecture. This concurs with the finding that the 'winning ticket' in the Lottery Ticket Hypothesis does not confer significant advantages over random reinitialization if a larger learning rate of 0.1 is used instead of 0.01 (Liu et al., 2018).

Similar results are also obtained for larger models such as ResNet18 and VGG19 (details in Supplementary Materials), which shows that this finding is a general one.

7. Empirical Analysis - Percentage of nodes/filters to Drop

Q7. Can we drop more nodes/filters at a time to reduce number of training cycles and prune the model faster without affecting accuracy?

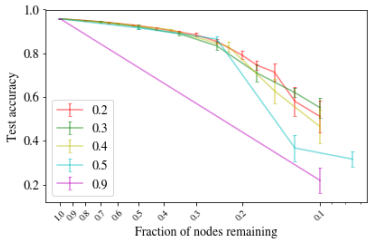


Figure 18. Test accuracy vs. fraction of filters remaining for various pruning fractions p in Model A: FC20 - FC20 on MNIST

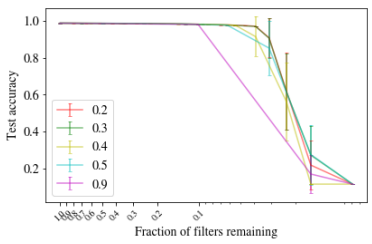


Figure 19. Test accuracy vs. fraction of filters remaining for various pruning fractions p in Model B: Conv64 - Conv64 on MNIST

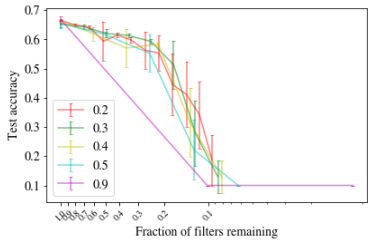


Figure 20. Test accuracy vs. fraction of filters remaining for various pruning fractions p in Model C: Conv64 - Conv64 - Conv128 - Conv128 on CIFAR-10

We explore this question by analyzing the performance of the `minimum` metric. We compare the performance of the model using different values of the pruning fraction $p = 0.2, 0.3, 0.4, 0.5$ and 0.9. The experiments performed are as follows:

7.1) **Model A: FC20 - FC20**. The plot of test accuracy against fraction of nodes remaining for various pruning fractions p on MNIST is shown in Fig. 18.

7.2) **Model B: Conv64 - Conv64**. The plot of test accuracy against fraction of filters remaining for various pruning fractions p on MNIST is shown in Fig. 19.

7.3) **Model C: Conv64 - Conv64 - Conv128 - Conv128**. The plot of test accuracy against fraction of nodes remaining for various pruning fractions p on CIFAR-10 is shown in Fig. 20.

Evaluation: Overall, it can be seen (Figs. 18, 19, and 20) that dropping a greater fraction p of nodes/filters per training cycle leads to poorer performance. For one-shot pruning methods which attempt to remove 90% of nodes/filters, it is not ideal as it generally leads to worse performance. This further reinforces the competitiveness of iterative pruning as compared to one-shot pruning.

The results also show that larger p has similar performance when dropping up to 70% of the nodes/filters. Hence, it may be possible to iterate faster through Algorithm 1 by adopting a larger p under certain conditions. Further experiments need to be done to determine the optimal pruning fraction p, but $p = 0.2$ seems to be competitive.

8. Concluding Remarks

Reflections: In this paper, we propose *DropNet*, which iteratively drops nodes/filters and, hence, reduces network complexity. We illustrate how *DropNet* helps to reduce network size by up to 90% without any significant loss of accuracy. Also, there does not need to be a particular initialization required when dropping up to 70% of the nodes/filters. *DropNet* also has similar performance to an oracle which greedily removes nodes/filters one at a time to minimise training loss, which shows its competitiveness.

DropNet, utilizing either the `minimum` or `minimum_layer` metric, proves to be competitive over a variety of model architectures. The `minimum` metric seems to work robustly well for smaller models such as Model A and B, while the `minimum_layer` metric appears to be better for larger models such as Model C, ResNet18 and VGG19, in the configurations we considered.

We conjecture that one reason for the better performance of the `minimum_layer` metric in larger models is that as the number of layers increases, the statistical properties of the post-activation values of the nodes/filters will be significantly different in each layer. Hence, using a single metric to prune globally may not be as good as pruning layer-wise. The exception is when using skip connections, as empirical results suggest that the `minimum` metric is also competitive for larger models for ResNet18. This seems to suggest that skip connections work well with the DropNet architecture.

Future Work: We present a preliminary study of the *DropNet* algorithm on MLPs and CNNs. We have observed situations where layer-wise pruning performs better than global pruning, and vice versa. More analysis needs to be done on larger models and datasets in order to understand which pruning method is more competitive for various models.

To further show *DropNet*'s generalizability, more experiments can be done on i) alternative neural network architectures such as RNNs, as well as ii) in other domains such as NLP/reinforcement learning. *DropNet* has been shown to work well empirically with $ReLU$ activation functions, and it remains to be seen whether other metrics may be required for other activation functions such as $sigmoid$ or $tanh$.

Source Code: To encourage further research on iterative pruning techniques, the source code used for our simulations is publicly available at https://github.com/tanchongmin/DropNet.

Acknowledgements

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-GC-2019-002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

Alemu, H. Z., Zhao, J., Li, F., and Wu, W. Group $l_{1/2}$ regularization for pruning hidden layer nodes of feedforward neural networks. *IEEE Access*, 7:9540–9557, 2019.

Alvarez, J. M. and Salzmann, M. Learning the number of neurons in deep networks. *Advances in Neural Information Processing Systems*, (Nips):2270–2278, 2016. ISSN 10495258.

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):1–18, 2017.

Castellano, G., Fanelli, A. M., and Pelillo, M. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pp. 1737–1746, 2015.

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, 2015.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993.

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, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

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

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

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

Lin, S., Ji, R., Li, Y., Wu, Y., Huang, F., and Zhang, B. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, pp. 2425–2432, 2018.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

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

Polyak, A. and Wolf, L. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.

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

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pp. 2074–2082, 2016.

Zhang, Z. and Qiao, J. A node pruning algorithm for feedforward neural network based on neural complexity. In *2010 international conference on intelligent control and information processing*, pp. 406–410. IEEE, 2010.