NodePrune: Sparsifying Neural Networks via Iterative Pruning

Ebuka Johnbosco Okpala [*1]

Abstract

Deep neural networks are often over-parameterized; this is a problem in resource-constrained devices and real-time systems as inference needs to be performed with less computation and instantaneously. Inspired by DropNet (Tan & Motani, 2020), we propose NodePrune, an algorithm that iteratively removes neurons/feature maps reducing the complexity of neural networks. NodePrune removes neurons in feed-forward multi-layer perceptrons (MLPs) by using its neighboring neurons and prunes feature maps in convolutional neural networks (CNNs) based on the change in the model loss. We show that NodePrune can drop 90% of neurons/feature maps without affecting accuracy, performs significantly better in terms of accuracy than DropNet for MLPs and smaller CNN models, and is comparable to DropNet's performance on bigger models like ResNet18 and VGG19. The code of this work can be found here (https://github.com/burunkus/NodePrune).

1. Previous paper summary

DropNet (Tan & Motani, 2020) uses an iterative pruning algorithm to drop nodes/filters in multilayer perceptrons and CNNs using the lowest average post-activation values across all training examples. They use various metrics as shown in Table 1 to determine which nodes/filters to remove from a network based on the importance of the nodes/filters. The importance of a node is determined by:

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

over all training examples, where V $(a_i|x_j)$ is the post-activation value of the node a_i with input sequence x_j. Since a filter is made up of constitutent nodes, its importance over all training examples is determined by:

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

Table 1. Metrics

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

The metrics are employed layer-wise or globally. The `mininum_layer`, `maximum_layer` and `random_layer` metrics remove `p` nodes/filters layer-wise while `minimum`, `maximum` and `random` metrics remove `p` nodes globally. `p` nodes/filters with the least post-activation values are dropped by the `minimum` metric and the `maximum` metric which does the opposite serves as a way to verify how effective the `minimum` metric is. The `random` metric serves as a control and it prunes `p` of the nodes randomly.

2. Introduction

Deep neural networks have shown incredible advancement in different fields such as natural language processing, computer vision, recommendation systems, etc. Modern deep neural networks in computer vision and natural language processing are often overparameterized as researchers keep adding more layers to these networks to improve generalization. These networks are often overparameterized. This is due to the difficulty in knowing the exact number of neurons and filters, among others, that a network needs to be generalizable without being overparameterized.

Overparameterized networks have high computational and memory costs. There is an additional cost if a real-time system is involved. Real-time systems require processes to be almost instantaneous, and a model with a considerable

[*]Equal contribution [1]School of Computing, Clemson University, Clemson, United States. Correspondence to: Ebuka Johnbosco Okpala <eokpala@clemson.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

evaluation time could be damaging. An example is a model deployed to automatically detect hate speech in online social media platforms like Twitter or Facebook. Evaluation time needs to be significantly low to prevent harmful posts from being shared or retweeted across these platforms.

Also, resource-constrained devices are now ubiquitous as they are used in IoT for large amounts of data collection where they are used as edge devices. The limited resources of these devices make it difficult to run models like convolutional neural networks (CNN). In practice, the real-time inference is performed in the cloud, and the actions are sent back to the device. The cycle between the cloud and millions of devices can be costly and introduces delays due to network traffic. Making training and inference run on-device would alleviate this cost.

Model sparsification can be seen as a neural architecture search (NAS). NAS is a technique used to find good neural network architectures automatically. In model sparsification, transformer and attention heads, neurons, filters, and channels can be pruned. Sparsification can be performed after training, before training, and sparsification by removing and adding elements during training. Sparification after training is the most common and usually requires retraining to improve the accuracy of the sparsified model.

Recent works try to reduce network complexity or computational cost by finding a subnetwork or subset of the parameters within the network which have close to the same accuracy as the original network. Neurons can be pruned based on their similarity (Srinivas & Babu, 2015). The smallest absolute weights have been used for weight removal (Li et al., 2016; Narang et al., 2017). Based on input or output sensitivity, (Chandrasekaran et al., 2000) prune neurons based on the linear combination of other neurons' outputs. (Zeng & Yeung, 2006) determine the importance of a neuron based on the change in its inputs, and (Tan & Motani, 2020) uses the absolute magnitude of activations across examples to determine which node/filter to prune. There have been works on first order loss function approximation (You et al., 2019; Molchanov et al., 2017), and regularization (Tartaglione et al., 2018; Pan et al., 2016; Yang et al., 2019; Liu et al., 2015).

In our work, to determine the importance of nodes in feed-forward multi-layer perceptrons, we utilize the absolute sum of the post-activation values of nodes in the previous and next layer across training examples to determine the importance of a node in the current layer. The input and output layers are used if the previous layer is the input layer and the next layer is the output layer. A channel's importance can be determined by observing its contribution to the loss across examples with and without the channel present. This process is not computationally efficient, but the same effect can be observed by observing the change in the loss due to a change in a channel. To determine the importance of channels/feature maps, we utilize the derivative of the loss with respect to each of the channels/feature maps across examples in mini-batches.

Our Contributions:

- We introduce NodePrune, which iteratively prune nodes with the lowest post-activation values based on the post-activation values in the previous and next layer across examples for MLPS. For CNNs, we compute the sum of the change in the loss wrt to a change in feature maps across examples in mini-batches. To the best of our knowledge, our work is the first to consider a node's input nodes and nodes reached from it in determining its importance for MLPs. Also, our work is the first to use the effect of a change in feature maps in the loss to determine the importance of feature maps in CNNs.
- NodePrune improves on the method DropNet (Tan & Motani, 2020) uses determine the importance of a node in MLPs and show that our CNN method is competitive to theirs.

3. Related work

Research in reducing neural network complexity has increased due to the need to compute in low-power devices and reduce inference time in real-time use cases. Sparsification of neural networks can defend as adversarial attacks (Sehwag et al., 2020).

(Tan & Motani, 2020) propose to prune nodes/filters iteratively similar to (Frankle & Carbin, 2018) based on the lowest average post-activation value across all training examples. They use the expected absolute value of a neuron across all training data points to determine its significance. For CNNs, the model elements in a channel/feature map as neurons. The significance of a channel/feature map is then determined by taking the average of the expected absolute value of the neurons in a channel across all training data points.

(Chandrasekaran et al., 2000) introduces linear dependence pruning. They model the outputs of hidden neurons as the linear combination of the outputs of other neurons. The neuron that causes an increase in the training error is found to be the least important neuron. This neuron is then replaced with its model and retrained. In their linear dependence model, the combination of post-activation values in the current and previous layers plus a threshold models the hidden neurons.

To reduce the number of parameters in neural networks (Tartaglione et al., 2018) uses a regularization term in the loss function that utilizes the sensitivity of the output with

respect to each parameter in the network. They define sensitivity and insensitivity, which measures how the network output changes from a change in the network parameter. The sensitive (important) parameters are left as they were, and the insensitive parameters are slowly moved towards zeros introducing sparsification. (Pan et al., 2016) drops neurons during training by using regularization. Inspired by group lasso, their regularization forces all the weights coming into a neuron and all the weights leaving a neuron to be zero, hence dropping that neuron.

To drop neurons in multilayer perceptrons (Zeng & Yeung, 2006) utilized a pruning technique that removes neurons based on their sensitivity from the hidden layer. The importance of a neuron is determined by multiplying its sensitivity by the sum of the absolute values of its outgoing weights. The degree to which a neuron with fixed incoming weights reacts to input changes is defined as sensitivity.

Our method is close to (Chandrasekaran et al., 2000) but differs in the sense that the importance of a neuron is modeled as the multiplication of the neuron's post-activation values with the sum of the post-activation value of neurons in the previous and next layer across all training examples. (Tartaglione et al., 2018) prunes based on the change of the output layer with respect to parameters. We prune based on the change in the loss with respect to activations across training examples by mini-batches for CNNs. Finally, (Tan & Motani, 2020) drops neurons/filters using the lowest average post-activation values across training examples for both MLPs and CNNs. Our method differs, as previously stated.

4. Methodology

In the following sections, our setup is based on (Tan & Motani, 2020). NodePrune was tested on different MLP and CNN architectures and on three datasets - MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky et al., 2009).

4.1. NodePrune Algorithm

NodePrune algorithm is described in this section.

The goal of NodePrune is to find a subnetwork in a network having close to the same accuracy as the original network but with fewer parameters. For example, we are given a neural network $f(t;p)$ initialized with initial weights and biases and composed of p neurons/feature maps. It is trained until it achieves the lowest validation loss l and test accuracy c. The goal is to find a subnetwork $f(t; m \odot p)$ in $f(t;p)$ trained with a mask $m \in \{0,1\}^{|p|}$ having almost the same accuracy as c, lowest validation loss $l^{'}$ and with fewer parameters. Where $\odot$ is element-wise multiplication. NodePrune algorithm is shown in Algorithm 1.

Importance of a neuron in MLPS: Algorithm 1 prunes neurons with the lowest absolute sum of the post-activation values of neurons in the previous and next layer multiplied by the current layer neuron's sum of absolute post-activation value across all training examples. Formally, given a_i^l i.e the i^{th} neuron in layer l and training examples $p_1, p_2, ..., p_T$ the neuron's importance is determined by:

$$a_i^l = \sum_{t=1}^{T} |a_i^l| * \sum_{j=1}^{m} \sum_{t=1}^{T} |a_j^{l-1}| + \sum_{k=1}^{n} \sum_{t=1}^{T} |a_k^{l+1}|$$

where m and n is the number of nodes in layers $l-1$ and $l+1$ respectively, and T is the total number of training examples.

Importance of a channel/feature map in CNNs: Inspired by (Tartaglione et al., 2018), algorithm 1 prunes feature maps that are less sensitive i.e. feature maps that have the least effect on the loss. Considering that each feature map is made up of post-activation values (from here on referred to as neurons), the sensitivity of a feature map is modeled based on the average sum of the absolute value of the change in loss with respect to each neuron in the feature map across all training examples in mini-batches. Let $a_{i,f}^l$ be the i^{th} neuron a in a feature map f in layer l. Let $a_{i,f}^l$ change by $\Delta a_{i,f}^l$ such that the total loss C vary by ΔC. Then the change in the loss C that resulted from $\Delta a_{i,f}^l$ is given by:

$$g_{a,f}^l \approx \frac{\partial C}{\partial a_{i,f}^l}$$

the sensitivity of a neuron a in a feature map f in layer l is given by the absolute sum of the change in the loss C with respect to the neuron across all training examples in mini-batches:

$$S_{a,f}^l = \sum_{i \in B} |g_{a,f}^l|$$

where each mini-batch in B is of size 256.

The sensitivity of a filter f in layer l is then given by:

$$S_f^l = \sum S_{a,f}^l / |B|$$

where $|B|$ is the number of examples (size of the mini-batches).

4.2. Metrics for Pruning

The pruning metrics (see Table 1) are same as in (Tan & Motani, 2020) for comparison purposes.

Algorithm 1 DropNet Algorithm

Input: initial neurons/feature maps n, pruning metric and neural network with initialized state θ_0 and initialized starting mask $m = \{1\}^{|n|}$
Hyperparameters: Number of training iterations j, pruning fraction p between 0 and 1.
1. Save initial model parameters
repeat
2. Initialize a new model $f^{'}$ using the parameters from step 1 and apply mask.
3. Train $f^{'}$ for j iterations or until early stopping.
4. Apply pruning metric to drop p neurons/feature maps and update mask.
until validation $a^{'} \leq ka$
Run steps 2 to 3

4.3. Experiment Details

The experiments used the MNIST and CIFAR-10 datasets. Datasets are split into training, validation, and test sets. For the MNIST dataset, 54000 images were used in training, 6000 as validation, and 10000 for testing. For the CIFAR-10, 45000 images were used for training, 5000 for validation, and 10000 for testing. The pixels in each image are scaled to a value between 0 and 1 for faster training.

All layers used the $ReLU$ activation function except for the output layer. The output layer used a softmax function as the problem is formulated as a multi-classification problem. For MLPs, the output layer uses $ReLU$ as activation function and softmax for classification. This is so that the determination of the importance of nodes in layer $L-1$ is consistent with other layers with $ReLU$ activation, here L is given as the output layer. Experiments are repeated over 2 runs except for the VGG19 and ResNet18 experiment where experiments where repeated only once due to resource constraint. Stochastic gradient descent (SGD) is used as the optimization function with a learning rate of 0.1. Cross entropy is used as the loss function.

At the beginning of each training cycle, masks are applied. Each training cycle comprises 100 epochs with validation loss used for early stopping if the loss does not decrease consecutively after five epochs (patience of 5). $p = 0.2$ of the nodes are dropped over each training cycle.

4.4. Model Details

Various experiments are conducted on different network models. The three architectures considered were feed-forward architectures. Model A is a multi-perceptron network with two fully connected layers of nodes. Model B contains two 2D convolutional layers, and Model C has four 2D convolutional layers. The architectures are shown in Table 2

5. Results

5.1. MLP - MNIST

Q1. Given MLPs of different starting configurations, can NodePrune perform robustly well?

This question is answered by performing experiments on different configurations of Model A using the MNIST dataset, and the configurations are as follows:

1.1) **Model A: FC40-FC40**. Figure 1 shows the test accuracy plotted against the fraction of nodes remaining for different metrics.
1.2) **Model A: FC20-FC40**. Figure 2 shows the test accuracy plotted against the fraction of nodes remaining for different metrics.
1.3) **Model A: FC40-FC20**. Figure 3 shows the test accuracy plotted against the fraction of nodes remaining for different metrics.

The training, validation, and test accuracies follow a similar trend. For that reason, only the results of the test accuracy are shown.

For 1.1) As shown in (Fig.1) all the metrics are very competitive except for `maximum` which started to perform poorly when the proportion of nodes remaining is 0.65, and below. The `minimum_layer` metric is the best performing metric, following it is the `random_layer`, then `maximum_layer`, `random`, and `minimum`.

For 1.2) As shown in (Fig. 2) all the metrics are very competitive except for `maximum` which started to perform poorly when the proportion of nodes remaining is 0.65, and below. The `minimum_layer` and `minimum` are both competitive followed by `maximum_layer`, `random_layer`, and `random`.

For 1.3) As shown in (Fig. 3), all the metrics are also very competitive except for `maximum` which started to perform poorly when the proportion of nodes remaining is 0.65, and below. The `minimum_layer` performs the best followed by `maximum`, `random`, `minimum`, and `random_layer`. The `minimum` metric is very competitive but started to drop in accuracy when the proportion of nodes remaining 0.25 and below.

Evaluation: For fully connected networks, the results show that when the hidden layer sizes are equal, the `minimum_layer` metric is competitive. With unequal hidden layer sizes, `minimum_layer` and `minimum` metrics are both competitive. This indicates that `minimum_layer` and `minimum` are competitive metrics for **Model A**. NodePrune reduced the number of nodes by 65% or more without affecting accuracy, indicating that it effectively reduces the complexity of neural networks. For all metrics except for `maximum`, NodePrune maintained good accuracy above

Table 2. The architectures considered. Different numbers of nodes/filters are used throughout the experiments, but the baseline architectures are the same. Models B and C are a variation of the VGG architecture (Simonyan & Zisserman, 2014). The first model **Model A: FC40-FC40** has two fully connected hidden layers with 40 nodes each. The middle model, **Model B: Conv64-Conv64** has two 2D convolutional layers, each having 64 filters of size 3 x 3 with 'same' padding. Mask is applied after each convolutional layer but before the MaxPooling2D layer. The third model, **Model C: Conv64-Conv64-Conv128-Conv128** has four 2D convolutional layers. The first two each have 64 filters, and the last two have 128 filters. Each filter has a size of 3 x 3 and with 'same' padding. After each convolutional layer, the mask is applied just before the MaxPooling2D layer.

MODEL A	MODEL B	MODEL C
INPUT	INPUT	INPUT
FLATTEN	CONV2D(64, (3,3), RELU)[MASK]	CONV2D(64, (3,3), RELU)[MASK]
FC(40, RELU)[MASK]	MAXPOOLING2D((2,2))	MAXPOOLING2D((2,2))
FC(40, RELU)[MASK]	CONV2D(64, (3,3), RELU)[MASK]	CONV2D(64, (3,3), RELU)[MASK]
FC(10, SOFTMAX)	MAXPOOLING2D((2,2))	MAXPOOLING2D((2,2))
	FLATTEN	CONV2D(128, (3,3), RELU)[MASK]
	FC(10, SOFTMAX)	MAXPOOLING2D((2,2))
		CONV2D(128, (3,3), RELU)[MASK]
		MAXPOOLING2D((2,2))
		FLATTEN
		FC(10, SOFTMAX)

60% after 10% of the nodes are remaining. From figures (3, 4 and 5) in DropNet (Tan & Motani, 2020) accuracy drops significantly after around 30% of the nodes are remaining showing how effective our algorithm is in pruning feed-forward neural networks.

5.2. CNN - MNIST

Q2. Given CNNs of different starting configurations, can NodePrune perform robustly well?

This question is answered by performing experiments on different configurations of Model B using the MNIST dataset. The configurations are as follows:

2.1) **Model B: Conv64-Conv64**. Figure 4 shows the test accuracy plotted against the fraction of filters remaining for different metrics.

2.2) **Model B: Conv32-Conv64**. Figure 5 shows the test accuracy plotted against the fraction of filters remaining for different metrics.

2.3) **Model B: Conv64-Conv32**. Figure 6 shows the test accuracy plotted against the fraction of filters remaining for different metrics.

The training, validation, and test accuracies follow a similar trend. For that reason, only the results of the test accuracy are shown.

For 2.1), the `minimum` metric is the best performing metric (Fig. 4), followed by `minimum_layer`, `random_layer`, `maximum_layer`, `random` and `maximum` metric. The `maximum` metric remained stable and became worse when the proportion of feature maps remaining is 0.65 and below.

For 2.2), (Fig. 5) shows that all metrics are very competitive. The `maxumum_layer` started to drop when 0.13 of the nodes were remaining but stabilized at an accuracy of 60%. The `maximum` metric is also competitive but not as others as its accuracy dropped a little when 0.35 of the feature maps were remaining but maintained a good accuracy.

For 2.3) The `minimum` metric outperforms other metrics, followed by `minimum_layer`, `random_layer` and `random` which are both competitive, `maximum_layer`, and lastly, `maximum` metric, as shown in Fig. 6. The `maximum` was as competitive as others until the number of feature maps remaining was 0.8 but maintained a good accuracy

Evaluation: For convolutional layers, the results show that the `minimum` metric is competitive when the hidden layer sizes are equal. With unequal hidden layer sizes, all metrics are competitive. Given that `minimum` and `minimum_layer` metrics performed very well in all hidden layer sizes, it indicates that they are a good metric for **Model B**. NodePrune can reduce the number of feature maps by 90% or more without affecting accuracy, suggesting that it effectively reduces the complexity of neural networks. Our algorithm also maintains a good accuracy above 50% after 0.2 of the feature maps are remaining compared to DropNet.

5.3. CNN - CIFAR-10: Model C

Q3. Given a larger dataset can NodePrune perform well?

This question is answered by performing experiments on different configurations of Model C using the CIFAR-10 dataset. The configurations are as follows:

3.1) **Model C: Conv64-Conv64-Conv128-Conv128**. Fig-

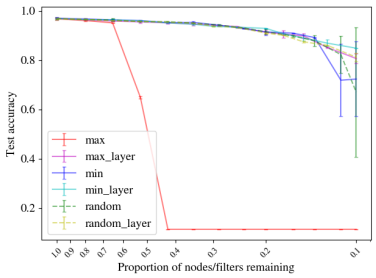

Figure 1. Model A: FC40-FC40 on MNIST. Plot of various metrics test accuracy vs proportion of nodes remaining.

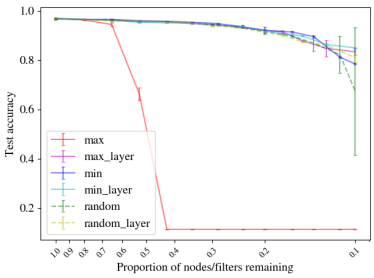

Figure 2. Model A: FC20-FC40 on MNIST. Plot of various metrics test accuracy vs proportion of nodes remaining.

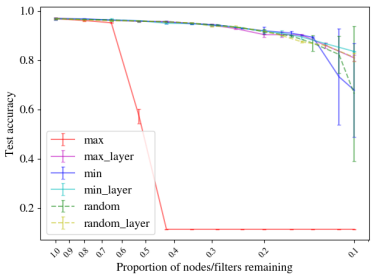

Figure 3. Model A: FC40-FC20 on MNIST. Plot of various metrics test accuracy vs proportion of nodes remaining.

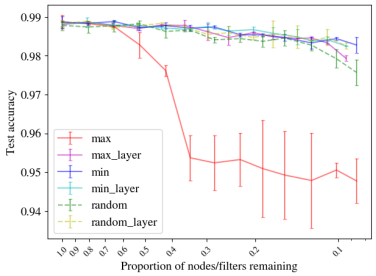

Figure 4. Model B: Conv64-Conv64 on MNIST. Plot of various metrics test accuracy vs proportion of filters remaining.

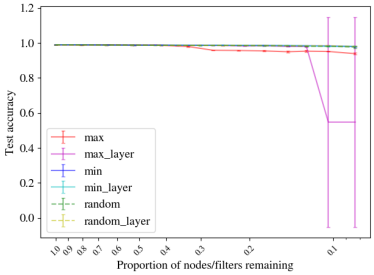

Figure 5. Model B: Conv32-Conv64 on MNIST. Plot of various metrics test accuracy vs proportion of filters remaining.

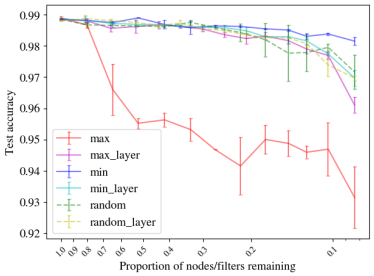

Figure 6. Model B: Conv64-Conv32 on MNIST. Plot of various metrics test accuracy vs proportion of filters remaining.

ure 7 shows the test accuracy plotted against the fraction of filters remaining for different metrics.

3.2) **Model C: Conv128-Conv128-Conv256-Conv256**. Figure 8 shows the test accuracy plotted against the fraction of filters remaining for different metrics.

The training, validation, and test accuracies follow a similar trend. For that reason, only the results of the test accuracy are shown.

For 3.1), the `minimum_layer` metric is the best performing metric (Fig. 7), followed by `minimum`, `random`, `random_layer`, `maximum_layer`, and `maximum` metric. The `maximum` metric was competitive when the proportion of feature maps remaining was greater than 0.8 but performed worse than other metrics below that. The `maximum_layer` metric did not perform well at the start but gained in accuracy as the number of feature maps remaining was 0.8 and above until 0.25 when the accuracy dropped again.

For 3.2), the results shown in (Fig. 8) indicate that the `minimum_layer` metric is the best performing metric, followed by `random_layer`, `maximum_layer`, `random`, `minimum`, and lastly, `maximum` metric. The `random_layer` and `maximum_layer` metrics are competitive. The `maximum` metric performed worse when the proportion of feature maps remaining is 0.65 and below.

Evaluation: The results indicate that the `minimum_layer` metric is competitive. NodePrune is more effective when the starting number of feature maps is 128 and above and can reduce the number of feature maps by 65% and above without significantly affecting accuracy. When the starting number of feature maps is less than 128, NodePrune can reduce the number of feature maps by 90% and more without affecting accuracy using all metrics except for the `maximum_layer` metric.

5.4. CNN - CIFAR-10: ResNet18/VGG19

Q4. ResNet18 and VGG19 are even larger models, can NodePrune perform well?

This question is answered by performing experiments on both ResNet18 and VGG19 on the CIFAR-10 dataset. The implementation of both models is very close to their original implementations, respectively (Simonyan & Zisserman, 2014; He et al., 2016). It is interesting to state that in ResNet18, there is no Batch Normalization, but there is before every MaxPooling2D layer in VGG19.

Figures 9 and 10 show the test accuracy plotted against the proportion of filters remaining for different metrics for ResNet18 and VGG19, respectively.

For ResNet18, figure 9 indicates that the `minimum_layer`

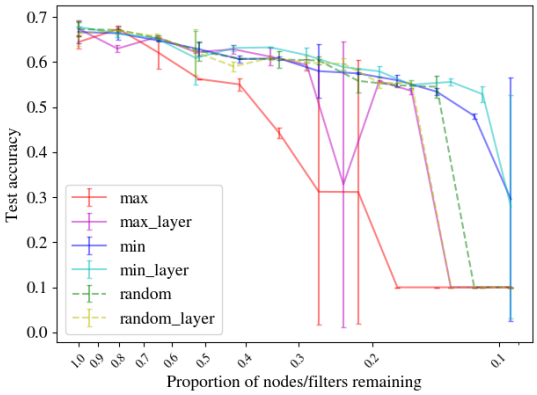

Figure 7. Model C: Conv64-Conv64-Conv128-Conv128 on MNIST. Plot of various metrics test accuracy vs proportion of filters remaining.

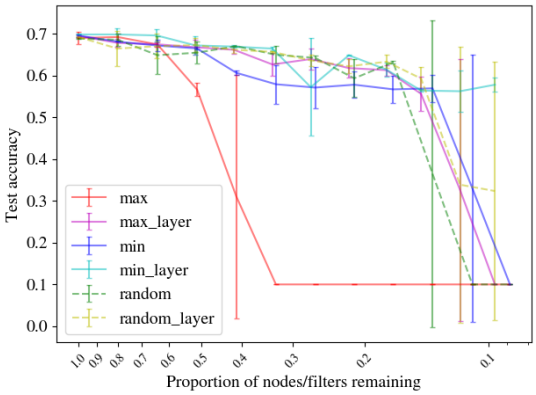

Figure 8. Model C: Conv128-Conv128-Conv256-Conv256 on MNIST. Plot of various metrics test accuracy vs proportion of filters remaining.

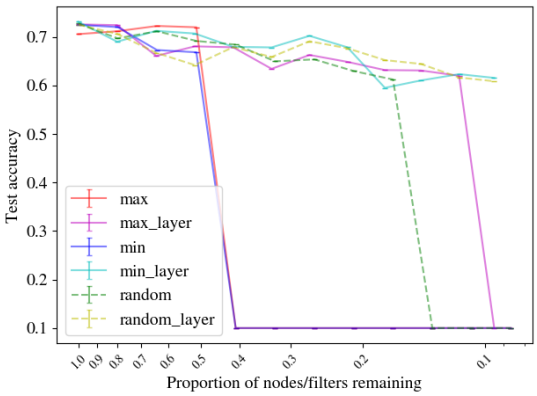

Figure 9. ResNet18 on CIFAR-10. Plot of various metrics test accuracy vs proportion of nofiltersdes remaining.

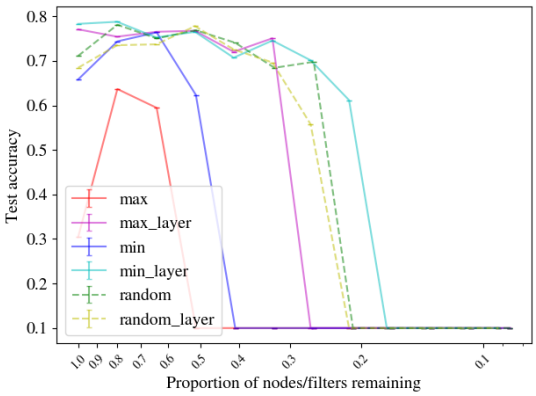

Figure 10. VGG19 on CIFAR-10. Plot of various metrics test accuracy vs proportion of filters remaining.

and `random_layer` are competitive metrics, followed by `maximum_layer`, `random`, `maximum`, and lastly `minimum`. The `maximum` and `minimum` metrics performed poorly when the number of feature maps remaining was 0.85 and below. The `minimum_layer` started poorly but attained some stability as the number of feature maps remaining kept reducing.

For VGG19, figure 10 shows that the `minimum_layer` metric is the most competitive, followed by the `random`, `random_layer`, `maximum_layer`, `minimum`, and, lastly, `maximum`.

It can be seen that for both ResNet18 and VGG19, the `maximum` metric is the worst performing metric. The `maximum` metric performed very poorly in VGG19.

Evaluation: The `minimum_layer` metric is the most competitive for larger models. Aside from `random`, metrics that prune layer-wise do outperform metrics that prune globally for larger models. For global pruning metrics, the importance of a feature map is determined by comparing the changes in the neurons in the feature maps across examples. This might not be a good metric due to the statistical differences across layers, as observed in DropNet. We can see that `minimum_layer` consistently performs well, indicating it is a good metric. In NodePrune, the `minimum_layer` is competitive for both ResNet18 and VGG19.

NodePrune can reduce the number of feature maps in VGG19 by 80% without affecting the model's accuracy, effectively lowering network complexity in larger models.

5.5. Empirical Analysis - Oracle Comparison

Q5. Compared to an Oracle how competitive is NodePrune?

Similar to DropNet, rather than comparing NodePrune to all the different pruning methods and algorithms in existence, an **oracle** is used to confirm how competitive NodePrune

is. The oracle removes a node/filter **greedily** out of all the nodes/filters remaining after each iteration of Algorithm 1. The goal is to get the overall training loss to a minimum. Different metrics are used in dropping the nodes/filters are dropped one at a time. The performance is evaluated on a smaller model as follows:

5.1) **Model A: FC20-FC20**. Figure 11 shows the test accuracy plotted against the fraction of nodes remaining when compared to the oracle on MNIST.
5.2) **Model B: Conv20-Conv20**. Figure 12 shows the test accuracy plotted against the fraction of filters remaining when compared to the oracle on MNIST.

For 5.1), figure 11 shows that all metrics perform competitively except for `oracle` which performed slightly better. The `random_layer` is used because it provides a solid baseline performance.

For 5.2), figure 12 shows that all metrics perform competitively. All metrics achieved good accuracy even the `maximum` metric.

Evaluation: Even to an `oracle` that minimizes training loss, the `minimum` metric is very competitive - indicating that the `minimum` metric is a good metric for dropping nodes/filters.

5.6. Empirical Analysis - Random Initialization

Q6. How important is the starting initialization of weights?

In this experiment, the performance of a network that retained its initial weights and biases during pruning is compared to a network that has been pruned using random initialization (`randominit`). The following experiments is conducted on the models below:

6.1) **Model A: FC20-FC20**. The plot in Figure 13 14 shows the comparison between the original and random initialization, showing the test accuracy against the proportion of nodes remaining.
6.2) **Model B: Conv64-Conv64**. The plot in Figure 13 14 shows the comparison between the original and random initialization, showing the test accuracy against the proportion of feature maps remaining.
6.3) **Model C: Conv64-Conv64-Conv128-Conv128**. The plot in Figure 13 14 shows the comparison between the original and random initialization, showing the test accuracy against the proportion of feature maps remaining.

From Figures 13, 14, and 15, it is shown that when up to 90% of the nodes/feature maps can be dropped when randomly initialized without loss in accuracy for MLPs and CNNs with a smaller number of feature maps. NodePrune, when initialized randomly, does not suffer in a performance like the Lottery Ticket Hypothesis (Frankle & Carbin, 2018) except for bigger CNN models with a starting feature map size of 64.

Evaluation: This shows that in NodePrune, the final pruned network is more important than the weights and biases assigned initially to the network. The learning rate is set high enough (0.1), which might be why the original initialization is not important as the network can comfortably retrain using only the model architecture. The result of Model C would have been different if our algorithm was tested using the `minimum_layer` metric since it is what performed the best throughout the experiments.

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

Q7. Can the model be pruned faster without affecting accuracy if more nodes/filters are dropped at a time to reduce number of training cycles?.

This question is answered by exploring the performance of the `minimum` metric. Different pruning fraction values are used $p = 0.2, 0.3, 0.4, 0.5$ and 0.9 to evaluate the performance of the model. The following experiments are conducted:

7.1) **Model A: FC20-FC20**. Figure 16 shows the test accuracy against the proportion of nodes remaining for different pruning fractions p on MNIST.
7.2) **Model B: Conv64-Conv64**. Figure 17 shows the test accuracy against the proportion of feature maps remaining for different pruning fractions p on MNIST.
7.3) **Model C: Conv64-Conv64-Conv128-Conv128**. Figure 18 shows the test accuracy against the proportion of feature maps remaining for different pruning fractions p on CIFAR-10.

Evaluation: In general, when larger p values like $p = 0.9$ are used in dropping nodes/filters per training cycle, it leads to poor performance, as seen in Figures (16, 17, and 18). We observe that except for $p = 0.9$, other p values are competitive. Overall, $p = 0.2$ performs very well.

6. Discussion

This paper introduces NodePrune, an iterative pruning algorithm for removing nodes/filters in neural networks, reducing the network's complexity. NodePrune can reduce the size of a network by up 80% without affecting the accuracy of the network. Up to 90% of the nodes/feature maps can be dropped without special initialization like the Lottery Ticket Hypothesis except in bigger CNN models. Compared to an oracle that greedily removes nodes/feature maps one after the other to minimize the training loss, NodePrune shows it is competitive by achieving similar performance.

Similar to DropNet, NodePrune uses different metrics to

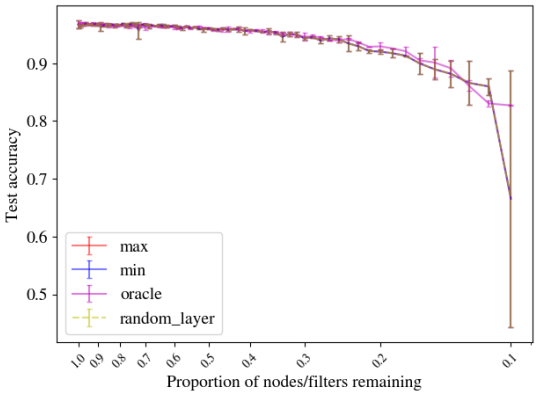

Figure 11. Model A: FC20-FC20 on MNIST. Plot of various metrics test accuracy vs proportion of nodes remaining when compared to an oracle.

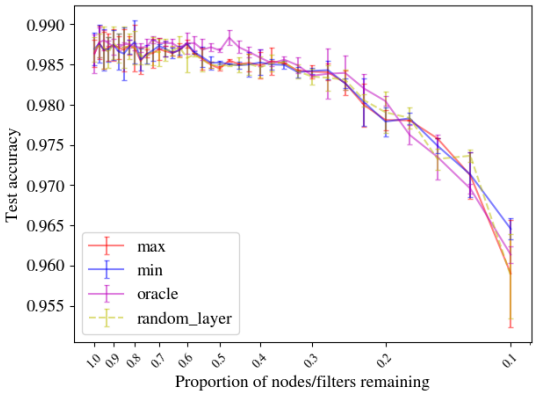

Figure 12. Model B: Conv20-Conv20 on MNIST. Plot of various metrics test accuracy vs proportion of filters remaining when compared to an oracle.

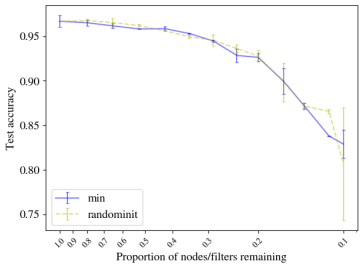

Figure 13. Model A: FC20-FC20 on MNIST. Plot of original initialization and random initialization for the proportion of nodes remaining vs test accuracy.

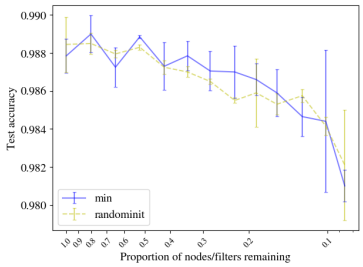

Figure 14. Model B: Conv64-Conv64 on MNIST. Plot of original initialization and random initialization for the proportion of filters remaining vs test accuracy.

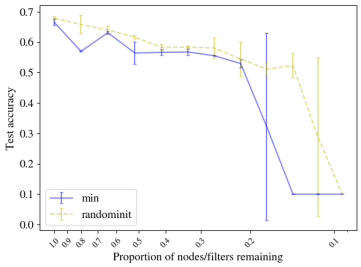

Figure 15. Model C: Conv64-Conv64-Conv128-Conv128 on CIFAR-10. Plot of original initialization and random initialization for the proportion of filters remaining vs test accuracy.

prune nodes/filters. The `minimum_layer` and `minimum` metrics are the most competitive for small Models like A and B on MNIST. The `minimum` metric is the most competitive in Model B. The `minimum_layer` is the most consistent for Model C. For larger models like ResNet18 and VGG19, `minimum_layer` is the most competitive. Also, layer-wise pruning methods outperform global pruning methods. The `minimum_layer` metric performs well in larger models because the post-activation values across node/feature maps are statistically different across each layer. Therefore, if a single metric is used to prune globally, it may not be as good as pruning layer-wise.

The `minimum_layer` metric is a good metric for dropping nodes/feature maps, as shown in the experiments. NodePrune outperforms DropNet in MLPs and maintains high accuracy as the number of nodes/feature maps remaining decreases. A similar observation is seen on smaller CNN models (Model B) on the MNIST dataset. On bigger models like VGG19, NodePrune is comparable to DropNet and achieves accuracy above 70% using `minimum_layer`, `random`, `random_layer` and `maximum_layer` metrics until the proportion of feature maps remaining is 0.3 and below. For ResNet18, NodePrune is also comparable to DropNet. For medium models like model C, NodePrune does not perform as well as DropNet, but `minimum_layer` metric, which is our best performing metric, is comparable.

The limitation of this work is that it only works for feed-forward architectures like MLPs and CNNs. The performance of this algorithm on neural network architectures like RNNs and language models like BERT (Devlin et al., 2018) is unknown and could be explored in the future.

References

Chandrasekaran, H., Chen, H.-H., and Manry, M. T. Pruning of basis functions in nonlinear approximators. *Neurocomputing*, 34(1-4):29–53, 2000.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert:

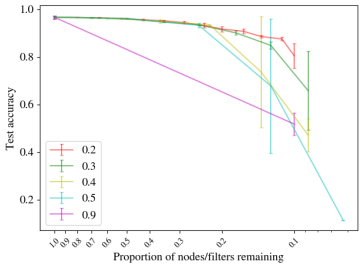

Figure 16. Model A: FC20-FC20 on MNIST. Plot of the proportion of nodes remaining vs test accuracy for different prunning fractions p.

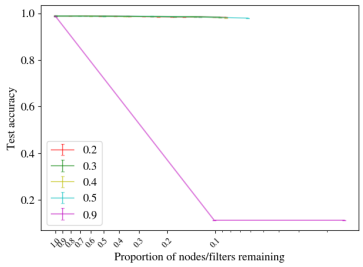

Figure 17. Model B: Conv64-Conv64 on MNIST. lot of the proportion of nodes remaining vs test accuracy for different pruning fractions p.

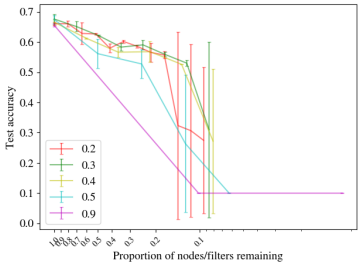

Figure 18. Model C: Conv64-Conv64-Conv128-Conv128 on CIFAR-10. plot of the proportion of filters remaining vs test accuracy for different pruning fractions p.

Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

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.

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 806–814, 2015.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pp. 2498–2507. PMLR, 2017.

Narang, S., Elsen, E., Diamos, G., and Sengupta, S. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

Pan, W., Dong, H., and Guo, Y. Dropneuron: Simplifying the structure of deep neural networks. *arXiv preprint arXiv:1606.07326*, 2016.

Sehwag, V., Wang, S., Mittal, P., and Jana, S. Hydra: Pruning adversarially robust neural networks. *arXiv preprint arXiv:2002.10509*, 2020.

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

Srinivas, S. and Babu, R. V. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

Tan, C. M. J. and Motani, M. Dropnet: Reducing neural network complexity via iterative pruning. In *International Conference on Machine Learning*, pp. 9356–9366. PMLR, 2020.

Tartaglione, E., Lepsøy, S., Fiandrotti, A., and Francini, G. Learning sparse neural networks via sensitivity-driven regularization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 3882–3892, 2018.

Yang, H., Wen, W., and Li, H. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019.

You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1909.08174*, 2019.

Zeng, X. and Yeung, D. S. Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure. *Neurocomputing*, 69(7-9):825–837, 2006.