CS 532 22S - Project 2: CNN Pruning comparisons

Christopher Shi
University of Massachusetts Amherst
140 Governors Dr, Amherst, MA 01002
cshi@umass.edu

1 Section 1: Problem Statement

In this project, we will evaluate and compare the influence of one-shot and iterative magnitude-based pruning on model sparsity, and test accuracy on a deep neural network model optimized for image classification tasks utilizing the Cifar10 dataset.

2 Section 2: Implementation

2.1 Initial Training of resnet18 Model

To begin, a deep neural network model optimized for image classification tasks utilizing the Cifar10 dataset must be trained to attain a test set accuracy of at least 93%. The chosen model was the resnet18 model, and the utilized loss function was the cross entropy loss function optimized through stochastic gradient descent.Hyperparameters were taken from https://github.com/kuangliu/pytorch-cifar in order to save time. To tune the number of epochs to be ran, the model was ran an indefinite number of epochs until the test accuracy was able to reach a value over 93%.(At each epoch the test accuracy of the model was checked.) The number of epochs until the model attained a test accuracy over 93% was 90 epochs. Other hyperparameters were kept consistent with those found in the model trained by https://github.com/kuangliu/pytorch-cifar. The state of the model that was able to attain 93% test accuracy was saved utilizing the torch.save() function.

2.2 Implementation of One-Shot Pruning Model

The following was repeated three times, with the following sparsity ratios, respectively: $[0.5, 0.75, 0.9]$.

The pre-trained model that achieved 93% test accuracy was first loaded from the state saved during the initial training of the resnet18 model. In order to determine the exact names of the layers to prune, the model was iterated through utilizing the named_modules() function. When iterating through the named_modulues() function, we specifically looked for layers that were an instance of torch.nn.Conv2d as well as torch.nn.Linear. In total, 20 convolutional layers were found and 1 linear layer were found. Once the names of all the convolutional and linear layers were compiled, the layers were appended to a list in the following format: tuple(torch.get_submodule('layername'), 'weight'). Once this was complete, the list was converted to a tuple, and served as the first argument for the torch.global_unstructured() pruning function. The pruning_method utilized was prune.RandomUnstructured, and the pruning amount was equal to the given sparsity ratio. Once global pruning was complete, the sparsity of individual layers were checked in order to calculate the global sparsity which ensures that the pruning process pruned the exact amount we desired. Following that, the model was re-trained utilizing the same training function and hyperparameters as the pre-trained model.

2.3 Implementation of Iterative Pruning Model

The following was repeated three times, with the following sparsity ratios serving as targets, respectively: $[0.5, 0.75, 0.9]$.

The pre-trained model that achieved 93% test accuracy was first loaded from the state saved during the initial training of the resnet18 model. In order to determine the exact names of the layers to prune, the model was iterated through utilizing the named_modules() function. When iterating through the named_modulues() function, we specifically looked for layers that were an instance of torch.nn.Conv2d as well as torch.nn.Linear. In total, 20 convolutional layers were found and 1 linear layer were found. Once the names of all the convolutional and linear layers were compiled, the layers were appended to a list in the following format: tuple(torch.get_submodule('layername'), 'weight'). Once this was complete, the list was converted to a tuple, and served as the first argument for the torch.global_unstructured() pruning function. The pruning_method utilized was prune.RandomUnstructured, and the pruning amount was equal to 0.1. Following that, the model was re-trained utilizing the same training function and hyperparameters as the pre-trained model. Following the training, the sparsity of individual layers were checked in order to calculate the global sparsity. If the calculated global sparsity was less than the target sparsity ratio, all prior steps were repeated (except for loading the state of the pre-trained model) until the global sparsity is equal to or greater than the target sparsity ratio.

3 Section 3: Experimental results

Given that hyperparameters such as number of epochs utilized during training time remained constant across the pre-trained, one-shot pruning, and iterative pruning models, and that the test accuracy was checked after each epoch, the following evaluation methods were utilized:

1. For one-shot pruning we chose to utilize the highest test case accuracy out of the 90 training epochs to be representative of each sparsity ratio.

2. For iterative pruning we chose to utilize the highest test case accuracy out of the 90 training epochs of the final iteration of the iterative pruning to be representative of each sparsity ratio.

The baseline accuracy of the trained from the ground up resnet18 model was equal to 93.06%. (Figure 1.) From Figure 2. and Table 1., we can see that the performance of one-shot pruning utilizing 0.5 and 0.75 sparsity ratios are relatively similar. Both sparsity ratios were able to achieve maximum test set accuracies of more than 92%, which is comparable to the baseline accuracy of 93.06%. Additionally, we can see that for a sparsity ratio of 0.9, the maximum test set accuracy dropped significantly, returning a value of 89.99% as compared to the baseline test accuracy of 93.06%.

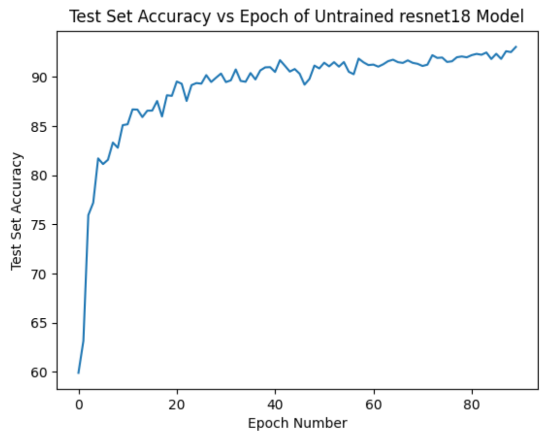

Figure 1: Test set accuracy vs number of epochs trained of an unpruned resnet18 model trained from scratch. An unpruned untrained resnet18 model was trained for 90 epochs. Maximum test set accuracy achieved was equal to 93.06%

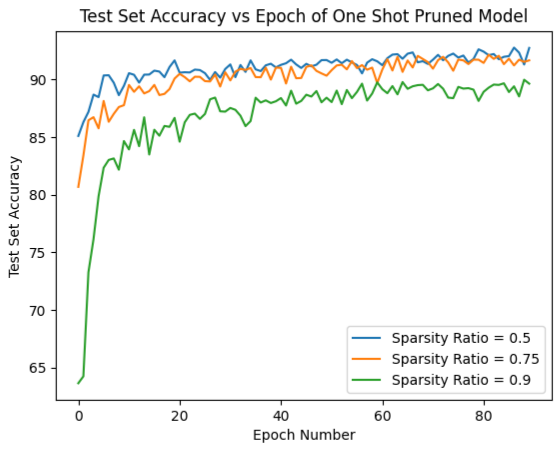

Figure 2: Test set accuracy vs number of epochs trained of a one shot pruned pre-trained resnet18 model. Sparsity ratios of $0.5, 0.75, 0.9$ were utilized for one shot pruning over 90 training epochs. Maximum test set accuracies achieved were 92.78%, 92.16%, and 89.99% respectively.

Sparsity Ratio	Max Test Accuracy
0.5	92.78
0.75	92.16
0.9	89.99

Table 1: Results of One Shot Pruning at Given Sparsity Ratios.

From Figure 3. and Table 2., we can see that during iterative tuning, for a sparsity ratio of 0.5, the maximum test accuracy seems to remain constant across iterations. This may be because during each round of re-training, the parameters that remain unpruned are able to relearn the information lost from the pruned parameters. One thing to note is that for each iteration of 10% pruning, the model was re-trained for 90 epochs which is equivalent to the number of training epochs from the original resnet18 model. (As such, given that 7 rounds of pruning occurred $7 * 90 = 630$ total epochs of re-training occurred, compared to the original 90 epochs)

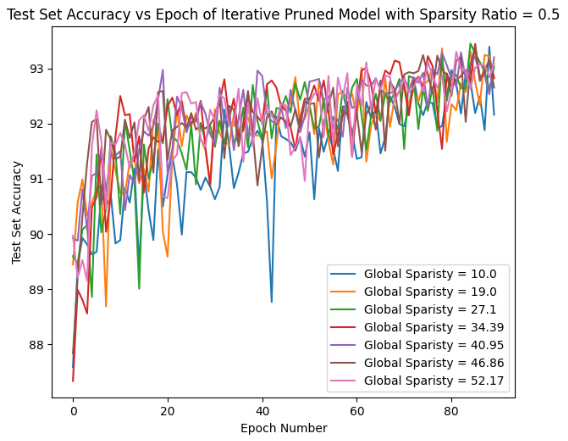

Figure 3: Test set accuracy vs number of epochs trained of a iteratively pruned pre-trained resnet18 model. Sparsity ratio of 0.5 was achieved after 7 iterations of 10% global pruning and re-training (Each re-training consisted of 90 epochs). Maximum test set accuracies across each round of iterative pruning were 93.39%, 93.36%, 93.45%, 93.44%, 93.29%, 93.4%, and 93.3% respectively.

Global Sparisty	Max Test Accuracy
10.0	93.39
19.0	93.36
27.1	93.45
34.39	93.44
40.95	93.29
46.86	93.4
52.17	93.3

Table 2: Results of 10% Iterative Pruning of a Pretrained resnet18 Model to a Global Sparsity of 0.5. Maximum test accuracies achieved for each round of 10% global tuning are shown.

From Figure 4. and Table 3., the results of iterative pruning with a sparsity ratio of 0.75 show that even when iteratively pruning down to a global sparsity of 77.12%, the model is still able to consistently maintain a maximum test set accuracy of 93.22%.

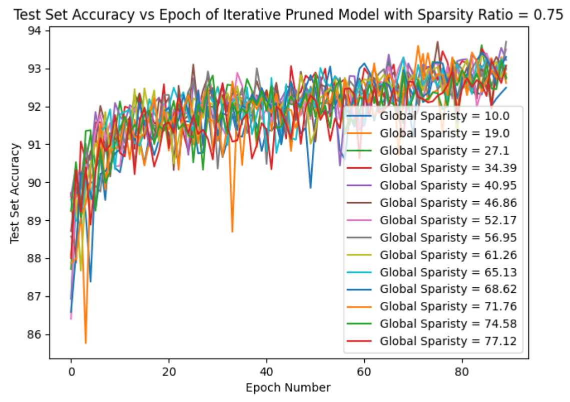

Figure 4: Test set accuracy vs number of epochs trained of a iteratively pruned pre-trained resnet18 model. Sparsity ratio of 0.75 was achieved after 14 iterations of 10% global pruning and re-training (Each re-training consisted of 90 epochs). Maximum test set accuracies across each round of iterative pruning were 93.06%, 93.38%, 93.61%, 93.52%, 93.54%, 93.7%, 93.5%, 93.7%, 93.33%, 93.29%, 93.41%, 93.59%, 93.2%, and 93.22% respectively.

Global Sparisty	Max Test Accuracy
10.0	93.06
19.0	93.38
27.1	93.61
34.39	93.52
40.95	93.54
46.86	93.70
52.17	93.50
56.95	93.70
61.26	93.33
65.13	93.29
68.62	93.41
71.76	93.59
74.58	93.20
77.12	93.22

Table 3: Results of 10% Iterative Pruning of a Pretrained resnet18 Model to a Global Sparsity of 0.75. Maximum test accuracies achieved for each round of 10% global tuning are shown.

From Figure 5. and Table 4., the results of iterative pruning with a sparsity ratio of 0.9 shows that at around 81.47% global sparsity, the model finally degrades in performance. When examining the maximum test accuracy for global sparsity ratios equal to or greater than 81.47%, the maximum test accuracy seems to decrease in a linear fashion. This may be because at this point, the remaining unpruned parameters are not able to sufficiently learn the information lost from the pruned

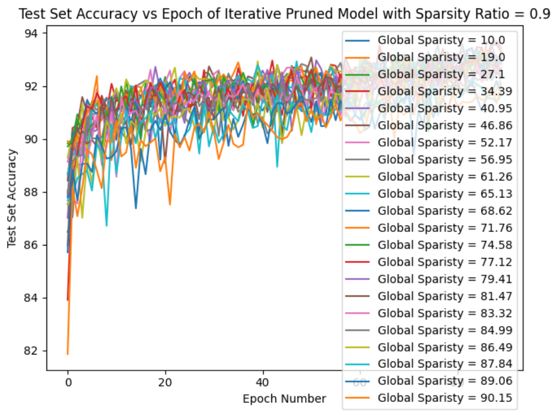

Figure 5: Test set accuracy vs number of epochs trained of a iteratively pruned pre-trained resnet18 model. Sparsity ratio of 0.9 was achieved after 22 iterations of 10% global pruning and re-training (Each re-training consisted of 90 epochs). Maximum test set accuracies across each round of iterative pruning were 93.1%, 93.23%, 93.46%, 93.57%, 93.66%, 93.4%, 93.34%, 93.7%, 93.2%, 93.35%, 93.44%, 93.36%, 93.17%, 93.1%, 93.14%, 92.84%, 92.92%, 92.48%, 92.37%, 92.25%, 92.19%, and 91.99% respectively.

Global Sparisty	Max Test Accuracy
10.0	93.10
19.0	93.23
27.1	93.46
34.39	93.57
40.95	93.66
46.86	93.40
52.17	93.34
56.95	93.70
61.26	93.20
65.13	93.35
68.62	93.44
71.76	93.36
74.58	93.17
77.12	93.10
79.41	93.14
81.47	92.84
83.32	92.92
84.99	92.48
86.49	92.37
87.84	92.25
89.06	92.19
90.15	91.99

Table 4: Results of 10% Iterative Pruning of a Pre-trained resnet18 Model to a Global Sparsity of 0.9. Maximum test accuracies achieved for each round of 10% global tuning are shown.

parameters, and as such, is not able to maintain a test set accuracy comparable to that of the baseline unpruned model.

4 Section 4: Summary and Takeaway

From our results, we can see that iterative pruning yields higher test set accuracies across all three tested sparsity ratios as compared to its corresponding test set accuracy from one shot pruning. (Figure 5.)

Sparsity Ratio	One-Shot	Iterative
0.5	92.78	93.30
0.75	92.16	93.22
0.9	98.99	91.99

Table 5: Test Set Accuracies of Given Sparsity Ratios and Given Pruning Method.

Additionally, from the experimental results of the iterative pruning, we can see that iterative pruning is able to maintain a test set accuracy better than or comparable to the accuracy of an unpruned model, up to a specific global sparsity. Once the global sparsity was pruned to any value equal to or greater than 81.47%, the maximum test set accuracy yielded results lower than the baseline unpruned model. This may be because at values less than 81.47% global pruning, the unpruned parameters are able to effectively learn the information contained in the pruned parameters during re-training. At values greater than 81.47%, there may not be enough unpruned parameters to effectively learn all of the information contained within the pruned parameters. As such, at values greater than 81.47% we can see the maximum test set accuracy decreasing.

5 Section 5: Team Contribution

Christopher Shi finished the whole project by himself.

6 Section 6: Artifact evaluation

The total output of 'UnprunedResnet18.py' was first copied from the terminal and saved to a text file named 'pretrainedmodeloutput.txt'. The total output of 'OneShotPrunedModels.py' was first copied from the terminal and saved to a text file named 'oneshotpruned.txt'. 'IterativePrunedModels.py' was ran and 'iterativeOutput.csv' was generated as the output. 'read.py' was executed to directly generate figures 1,2,3,4, and 5. Tables 1, 2, 3, and 4 were generated from reading the terminal output of 'read.py'. Table 5 was generated from the results of Tables 1, 2, 3, and 4.