Project 2: CNN Pruning Comparisons

Team Name: Team 9A
Team Members: Neeharika Karanam, Roshitha Bezawada, Sirisha Annamraju

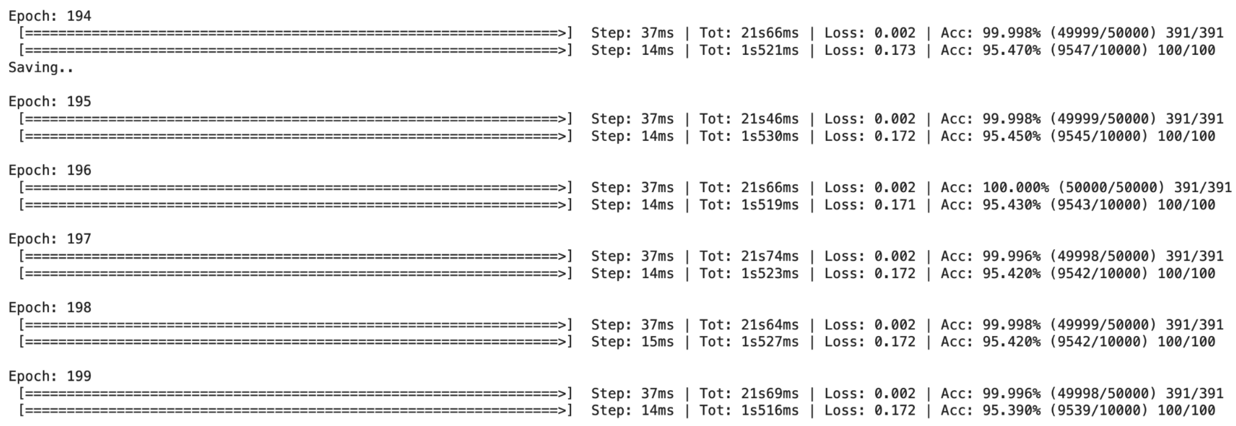
Problem Statement

Pruning is mainly used so that our neural network model will be small and more efficient. The main goal is to optimize the model by removing the values of the weight tensors so that we can get a computationally cost-efficient model that will take very less time for the training. However, removing the entire neurons will very easily hurt the accuracy of the neural network.
The necessity of pruning is that it will save time and the resources as well as it is essential for the execution of the model on the low-end devices such as mobile phones. CNNs suffer with issues like the computational complexity and the number of the parameters. Therefore, different pruning methodologies are performed to overcome these issues.

Implementation

Step 1: Train a model on CIFAR10

We first trained the CIFAR10 dataset with the ResNet18 model with a learning rate of 0.1. We were able to achieve an accuracy of 95.3% in 200 epochs and the checkpoint of the model after 200 epochs is saved in “ckpt.pth” file in the checkpoint folder.



```
Epoch: 194
 [=================================================================>]  Step: 37ms | Tot: 21s66ms | Loss: 0.002 | Acc: 99.998% (49999/50000) 391/391
 [=================================================================>]  Step: 14ms | Tot: 1s521ms | Loss: 0.173 | Acc: 95.470% (9547/10000) 100/100
Saving..

Epoch: 195
 [=================================================================>]  Step: 37ms | Tot: 21s46ms | Loss: 0.002 | Acc: 99.998% (49999/50000) 391/391
 [=================================================================>]  Step: 14ms | Tot: 1s530ms | Loss: 0.172 | Acc: 95.450% (9545/10000) 100/100

Epoch: 196
 [=================================================================>]  Step: 37ms | Tot: 21s66ms | Loss: 0.002 | Acc: 100.000% (50000/50000) 391/391
 [=================================================================>]  Step: 14ms | Tot: 1s519ms | Loss: 0.171 | Acc: 95.430% (9543/10000) 100/100

Epoch: 197
 [=================================================================>]  Step: 37ms | Tot: 21s74ms | Loss: 0.002 | Acc: 99.996% (49998/50000) 391/391
 [=================================================================>]  Step: 14ms | Tot: 1s523ms | Loss: 0.172 | Acc: 95.420% (9542/10000) 100/100

Epoch: 198
 [=================================================================>]  Step: 37ms | Tot: 21s64ms | Loss: 0.002 | Acc: 99.998% (49999/50000) 391/391
 [=================================================================>]  Step: 15ms | Tot: 1s527ms | Loss: 0.172 | Acc: 95.420% (9542/10000) 100/100

Epoch: 199
 [=================================================================>]  Step: 37ms | Tot: 21s69ms | Loss: 0.002 | Acc: 99.996% (49998/50000) 391/391
 [=================================================================>]  Step: 14ms | Tot: 1s516ms | Loss: 0.172 | Acc: 95.390% (9539/10000) 100/100
```

The loss function we have used in this model is cross entropy loss and we implemented stochastic gradient descent with momentum of 0.9 and weight decay of 5e-4 to update the weights.

Step 2: Implement one-shot magnitude-based pruning

- In one-shot pruning, the checkpoint from the well-trained model is retrieved and then the pruning is done on all the convolutional layers and the FC layer. ResNet contains about 20 convolution layers, a Fully Connected layer and some BatchNormalization layers but the pruning is done only on the convolution layers and the FC layer as they contain the maximum model parameters. The function used in pruning l1_unstructure which prunes the 3 smallest entries in the bias by L1 norm.

- After pruning the model using an input argument of global model sparsity, it is re-trained for 20 epochs. The pruned weights are then removed from the re-trained model using prune.remove().This pruned model is then saved in "<global_model_sparsity> percent_single_ckpt.pth" in the checkpoint folder.

- We pruned the pre-trained model for three different global model sparsity ratios of 50%, 75% and 90% and saved the checkpoints of all these three differently pruned models in the checkpoint folder.

Step 3: Implement iterative magnitude-based pruning

- In iterative magnitude-based pruning, the checkpoint from the pre-trained model is retrieved and copied into a new model. This new model is pruned with a sparsity ratio of 11% and re-trained for 5 epochs.

- Pruning and re-training(5 epochs) of the new model is continued in iterations until the global sparsity ratio of the pruned model reaches the specified sparsity ratio which is considered as threshold. Once the global sparsity ratio reaches the threshold, the pruned model is trained for 100 epochs.

- After removing pruned weights from the pruned and trained model, the model is saved in "<global_model_sparsity>percent_iter_ckpt.pth" in the checkpoint folder.

- In both the cases, the pruned weights are removed after retraining because prune.remove() is used to remove the sparsity mask. During retraining, there is a high probability that the zeroed weights might become non-zero due to gradient update.

Step 4: Create test.py file

- To check the above created models, we created a test.py file to build the models using the checkpoints, train the model to load the weights and test the models using test_data.

Experimental Results

Evaluation Metrics

1. **Model sparsity**: It can be defined as the total number of zero parameters to the total number of the parameters.

$$Sparsity = \frac{\#\, zero\, parameters}{\#\, parameter}$$

The sparsity value indicates the percentage of the parameters which have been set to zero during the pruning stage. In this project we have considered three different sparsity ratios 50%, 75% and 90% to evaluate the model performance.

2. **Test accuracy and test accuracy drop**: The test accuracy can be defined as the correctly classified data points from the test data to the total amount of data points from the test data.

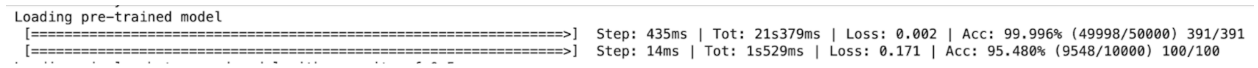
$$Test\ Accuracy = \frac{Correctly\ classified\ data\ points\ from\ the\ test\ data}{Total\ amount\ of\ the\ data\ points\ from\ the\ test\ data}$$

   Accuracy drop is the difference between the difference between the accuracy before pruning and after pruning.

   Accuracy drop = Test accuracy without pruning – Test accuracy after pruning

Results

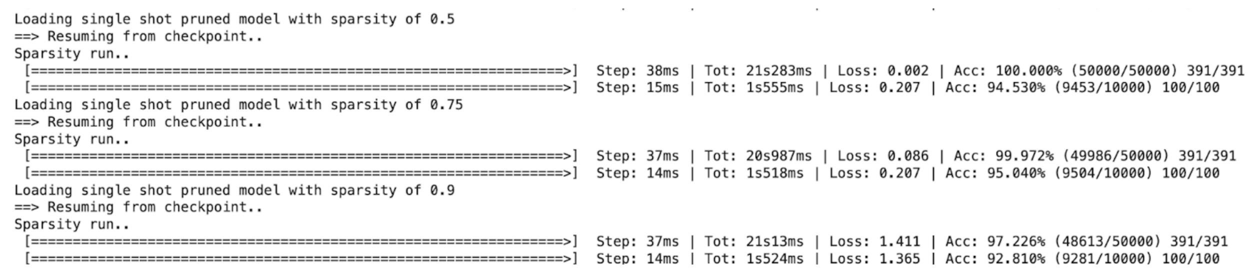
For the pre-trained model, the test-accuracy after 200 epochs is 95.480% .



```
Loading pre-trained model
 [================================================================>]  Step: 435ms | Tot: 21s379ms | Loss: 0.002 | Acc: 99.996% (49998/50000) 391/391
 [================================================================>]  Step: 14ms | Tot: 1s529ms | Loss: 0.171 | Acc: 95.480% (9548/10000) 100/100
```

One-shot Magnitude-based pruning

After one-shot pruning the pre-trained model with 50%, 75% and 90% model sparsity ratios, the respective test accuracies are given below in the screenshot.



```
Loading single shot pruned model with sparsity of 0.5
==> Resuming from checkpoint..
Sparsity run..
 [================================================================>]  Step: 38ms | Tot: 21s283ms | Loss: 0.002 | Acc: 100.000% (50000/50000) 391/391
 [================================================================>]  Step: 15ms | Tot: 1s555ms | Loss: 0.207 | Acc: 94.530% (9453/10000) 100/100
Loading single shot pruned model with sparsity of 0.75
==> Resuming from checkpoint..
Sparsity run..
 [================================================================>]  Step: 37ms | Tot: 20s987ms | Loss: 0.086 | Acc: 99.972% (49986/50000) 391/391
 [================================================================>]  Step: 14ms | Tot: 1s518ms | Loss: 0.207 | Acc: 95.040% (9504/10000) 100/100
Loading single shot pruned model with sparsity of 0.9
==> Resuming from checkpoint..
Sparsity run..
 [================================================================>]  Step: 37ms | Tot: 21s13ms | Loss: 1.411 | Acc: 97.226% (48613/50000) 391/391
 [================================================================>]  Step: 14ms | Tot: 1s524ms | Loss: 1.365 | Acc: 92.810% (9281/10000) 100/100
```

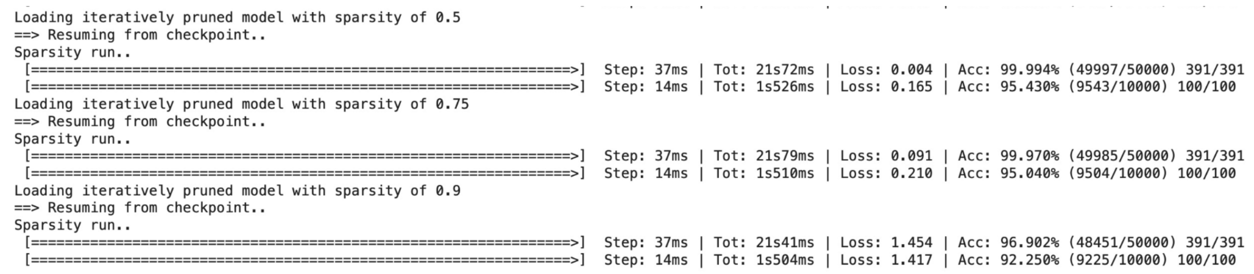
The above screenshot is taken after running test.py. Since we are dropping a few insignificant weights from the network in each layer the test-accuracy is supposed to decrease. But as we are re-training the model after pruning, the model learns again and updates weights. So pruning helped us in reducing the complexity of the model with little drop in the accuracy of the model.

Model	**Test Accuracy Drop**
One-shot pruning with 50% sparsity	0.95%
One-shot pruning with 75% sparsity	0.44%
One-shot pruning with 90% sparsity	2.67%

From the above accuracy drop values, we can say that if the model is pruned with 50% sparsity ratio, there is a decrease in the accuracy of the model as there is decrease in the weights there is a decrease in the accuracy. For the sparsity ratio of 75%, there isn't much of accuracy drop but then 75% of least weight weights are removed which reduces the complexity significantly. For the sparsity ratio of 90%, the drop in accuracy is 2.67% but still the test accuracy we got for the pruned model is an acceptable value.

Iterative Magnitude-based pruning

After iterative magnitude-based pruning the pre-trained model with 50%, 75% and 90% model sparsity ratios, the respective test accuracies are given below in the screenshot.



```
Loading iteratively pruned model with sparsity of 0.5
==> Resuming from checkpoint..
Sparsity run..
 [===============================================================>]  Step: 37ms | Tot: 21s72ms | Loss: 0.004 | Acc: 99.994% (49997/50000) 391/391
 [===============================================================>]  Step: 14ms | Tot: 1s526ms | Loss: 0.165 | Acc: 95.430% (9543/10000) 100/100
Loading iteratively pruned model with sparsity of 0.75
==> Resuming from checkpoint..
Sparsity run..
 [===============================================================>]  Step: 37ms | Tot: 21s79ms | Loss: 0.091 | Acc: 99.970% (49985/50000) 391/391
 [===============================================================>]  Step: 14ms | Tot: 1s510ms | Loss: 0.210 | Acc: 95.040% (9504/10000) 100/100
Loading iteratively pruned model with sparsity of 0.9
==> Resuming from checkpoint..
Sparsity run..
 [===============================================================>]  Step: 37ms | Tot: 21s41ms | Loss: 1.454 | Acc: 96.902% (48451/50000) 391/391
 [===============================================================>]  Step: 14ms | Tot: 1s504ms | Loss: 1.417 | Acc: 92.250% (9225/10000) 100/100
```

The above screenshot is also taken after running test.py.

Model	**Test Accuracy Drop**
Iterative pruning with 50% sparsity	0.05%
Iterative pruning with 75% sparsity	0.44%
Iterative pruning with 90% sparsity	3.23%

From the above accuracy drop, we observed that there is hardly any drop in the accuracy drop for 50% sparsity ratio which is a good thing as there is a reduction in the complexity of model but negligible difference in the accuracy. For the sparsity ratio of 75%, there is a slight drop in the accuracy as the reduction in the weights have increased the drop in accuracy is valid. For the sparsity ratio of 90%, there is a significant drop in accuracy of 3.23% as 90% of the weights have been removed, the drop in the accuracy can be justified. However, the test accuracy we got for the pruned model is an acceptable value.

Summary and Takeaway

In the recent decade it is observed that the large neural networks provide better results, but these large deep learning models come with an enormous cost. With the help of pruning in a very large neural network there is a much smaller subset which can provide the same accuracy as that of the original model without any significant penalty on its performance. Once the model goes through the entire training phase, we can remove many of its parameters(weights) and shrink the original size of the model. This provides various benefits such as obviating the need of sending the user data to the cloud servers and providing the real time inference. In some areas, small neural networks will make it possible to employ deep learning on devices which can be powered by solar batteries or button cells.

We have understood the importance of pruning in order to compress the model by removing the weights and still get a good test accuracy. We have also understood the concept of Lottery Ticket Hypothesis where a small subnetwork in the dense neural network when randomly-initialised and trained in isolation can match the test accuracy of the original network after training will have almost the same iterations. It also states that the optimal neural network structures can be learned from the start.

Team Contribution

Neeharika Karanam - Has implemented the pre-trained model and is also involved in both one shot and iterative pruning algorithms in fine tuning based on hyper parameters. Has prepared the report for the project.

Roshitha Bezawada - Has implemented the iterative pruning algorithm for the pre-trained model.

Sirisha Annamraju - Has implemented the one-shot pruning algorithm for the pre-trained model.

Artifact Evaluation

The first three steps in the implementation part are executed in the main.py file.
To train the model without pruning,

$$!python\ \ main.py$$

There are two arguments you can pass to determine the sparsity ratio and pruning method. The arguments are given below.

$$-\ pr\quad --> \ Indicates\ the\ sparsity\ ratio$$
$$-\ shot\ --> \ Indicates\ the\ pruning\ method$$

$$'iter'\ is\ for\ iterative\ pruning\ and\ 'single'\ is\ for\ 1-shot\ pruning$$

To prune the model using one-shot pruning using a global sparsity ratio of x(decimal point in our case), below is the command we use.

$$!\,python\ main.py\ -pr\ x\ -shot\ 'single'$$

If we have to prune the model using iterative magnitude-based pruning with a global sparsity ratio of x, below is the command we use.

$$!\,python\ main.py\ -pr\ x\ -shot\ 'iter'$$

Since we have saved the checkpoints for the trained model and all pruned models, we can just load the checkpoints containing the weights of different models, and test them using test_final() in test.py. The test.py file does the above work for us. To run the test.py file, below is the following command.

$$!\,python\ test.py$$

- After the model is trained with the pruned weights the model is tested on the test dataset where the model performs better than that of the model before pruning.
- The global model sparsity, test accuracy before pruning and test accuracy after accuracy are noted for a predefined set of 50%, 75% and 90% respectively.