COMPSCI 532 - SYSTEMS FOR DATA SCIENCE - Fall 21 Project 2 - CNN Pruning Comparisons

Team members: Sakshi Bhalerao, Snehal Thakur, Vishnupriya Varadharaju

Section 1: Problem statement

Most deep neural networks that have high performance, require large amounts of computations and memory. This not only increases infrastructure costs, but it also makes the deployment of such large networks to resource constrained environments such as smart devices very challenging.

One way to tackle this problem and reduce the resource requirement is to perform pruning of the model. Pruning is a method of compression which involves systematically removing weights from a trained model. This model compression should be performed without compromising on the performance of the model i.e. a smaller network should be produced, but with the same accuracy. This is important in order to reduce memory, battery, and hardware consumption without sacrificing accuracy, so as to deploy lightweight models on device, and guarantee privacy with private on-device computation.

In this project, we are implementing one-shot pruning and iterative magnitude-based pruning on a deep neural network model for classifying the CIFAR-10 dataset. Weight pruning which is an unstructured pruning algorithm is used here. This global pruning, removes the lowest x% of connections across the whole model, instead of removing the lowest x% in each layer. Maximum sparsity with maximum accuracy is the end goal of this project. The pruned models are then re-trained and analysed for certain metrics like model sparsity, test accuracy and speedups.

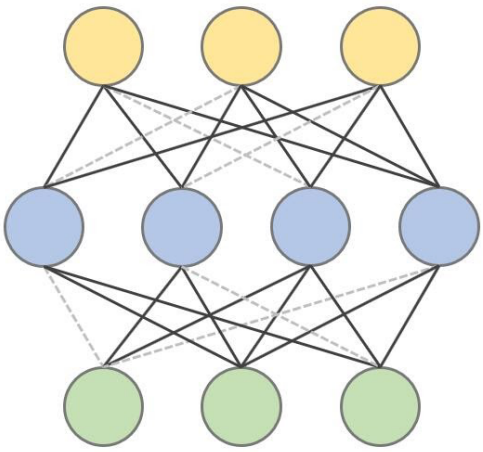

Fig. Unstructured Pruning

Section 2: Implementation

In our project, we have performed pruning after training the model and then removed those weights with the lowest values. Pruning does seem to compress the neural networks during test time, but there is no reduction in the cost of training.

Pruning also reduces the accuracy of the network. So, it has to be further trained in order to get back the desired accuracy. This process is known as fine-tuning. The process of pruning and fine-tuning is then iterated several times, gradually reducing the network's size.

Algorithm 1 Pruning and Fine-Tuning

Input: N, the number of iterations of pruning, and
X, the dataset on which to train and fine-tune

1: $W \leftarrow initialize()$
2: $W \leftarrow trainToConvergence(f(X; W))$
3: $M \leftarrow 1^{|W|}$
4: **for** i in 1 to N **do**
5: $\quad M \leftarrow prune(M, score(W))$
6: $\quad W \leftarrow fineTune(f(X; M \odot W))$
7: **end for**
8: **return** M, W

Fig. Pruning and Fine-Tuning Algorithm

Design and Working -

1. The base model used is ResNet18, obtained from the github repository - https://github.com/kuangliu/pytorch-cifar
   The original base model was created as per this repo and then it was trained to obtain an accuracy of 95.29% on test data iterated over 199 epochs.

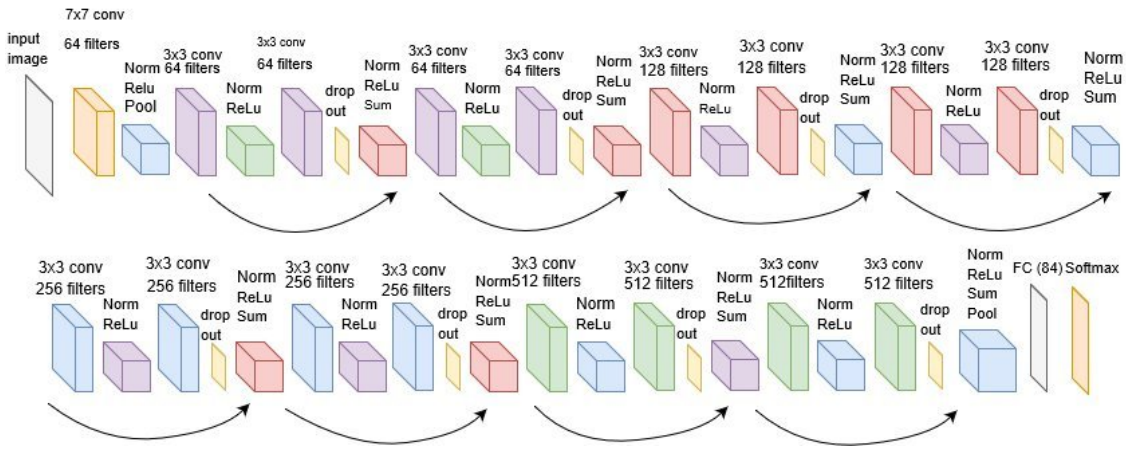

Fig. ResNet18 Model Architecture

2. The weights of this trained model for which the highest accuracy was obtained are stored in a file `checkpoint/ckpt.pth`. These weights were then used as the initial weights during pruning.
3. Pruning the model -
   Two types of pruning were performed on the base trained model - **one-shot pruning** and **iterative magnitude pruning**. To perform this pruning, the PyTorch Pruning API was used.
   First, a pruning technique was chosen amongst those available in the `torch.nn.utils.prune` library. Then, the module i.e. the ResNet18 network and the parameter to be pruned within that module is specified. Finally, appropriate keyword arguments are used to select the required pruning technique and the necessary parameters for retraining the model after pruning.

   For example -

```
!python main.py –pos –pa 0.75
```

   Here, `–pos` stands for pruning one-shot and `–pa 0.75` implies pruning of the model with 75% sparsity.

   Function used for pruning in PyTorch:
   `global_unstructured()`

   Parameters taken as the input:
   `parameters_to_prune` = weights of the network
   `pruning_method` = L1 unstructured (i.e. Pruning units in a tensor by zeroing out the ones with the lowest L1 norm)
   `prune_amount` = sparsity of the network

4. The pruned model's performance is then evaluated by determining its accuracy on the test data.
5. The same steps were performed for different sparsity at 50%, 75% and 90% for both one-shot pruning and iterative-magnitude pruning and their results were tabulated. In *one-shot pruning*, the model is pruned all at once and then it is trained. The best accuracy is compared over the different iterations and only those weights for which highest accuracy is obtained is stored.
   In *iterative pruning* the network is pruned in iterations. For example, to reach 50% sparsity the model is iterated 5 times with sequential increase (10%) in sparsity and then trained on every iteration. Finally the test data is run on the model and the weights for the best test accuracy is stored.
6. The base model was trained for 200 epochs and the one-shot and iterative pruning models were trained for 100 epochs.

Section 3: Experimental results

In our project, we have used the test accuracy for different sparsity as the main evaluation criteria for the two different pruning types.

The following test accuracies were obtained on pruning the base model -

Pruning type / Sparsity Ratio	50%	75%	90%
Iterative Magnitude Pruning	90.99	91.1	91.16
One shot pruning	92.36	92.26	92.16
No pruning	95.29		

Key Findings -

1. From the results, we can see that the accuracy drop when the model is pruned. The accuracy of the model without pruning was 95.29%.
2. In case of one-shot pruning, the accuracy drops by 3% for 75% and 95% sparsity and by around 4% for 50% sparsity. In case of iterative pruning, the accuracy drops by almost 4% for all the sparsity ratios.
3. As read in the research paper, the accuracy drops, on pruning the model and iterative performs better than one shot pruning as the sparsity ratio increases. In this implementation, it is seen that one-shot pruning out performs iterative magnitude pruning in all the sparsity ratios. One of the reason for this could be that in iterative pruning we slowly increase the sparsity over multiple iterations, so the model is able to learn better and effectively remove those weights that seem not very important. Hyperparameter tuning such as number of epochs, learning rate, number of iterations and regularization is more effective for this mode of pruning. This can owe it to learning better even with lesser weights. Whereas in one shot pruning, the weights are zeroed all at once. This can cause key features to be zeroed out causing the drop in accuracy.
4. The model that we are using is Resnet18, is a highly complex neural network and designed for Imagenet dataset for capturing its complex features and classifying images into 1000 categories. But here, we are training it for CIFAR10 dataset which is relatively smaller dataset that doesn't need such a complex model. This could be a possible reason for the high accuracy values obtained even after pruning the weights as the model is able to capture most of the features of the dataset effectively.

Section 4: Summary and Takeaway

Neural network pruning is a method that revolves around the intuitive idea of removing superfluous parts of a network that performs well but costs a lot of resources. Indeed, even though large neural networks have proven countless times how well they could learn, it turns out that not all of their parts are still useful after the training process is over. The idea is to eliminate these parts without impacting the network's performance.

In our project, we have successfully pruned a ResNet18 model using both one-shot and iterative magnitude pruning, which come under unstructured pruning. The initial accuracy of the test data from the base model was around 95%. Whereas, after pruning the accuracy dropped by almost 5% to an accuracy of 90% (approximately).

Thus, we can conclude that pruning reduces the accuracy of the network. Also, one shot pruning seemed to perform better than iterative pruning for lower values of sparsity.
To add on, these methods of unstructured pruning, present a major fatal drawback because, most frameworks and hardware cannot accelerate sparse matrices computation. Basically, no matter with how many zeros the parameter tensors are filled with, it does not impact the actual cost of the network. However, pruning in a way directly alters the very architecture of the network, which any framework can handle, in our case – PyTorch.

Finally, the key takeaway is that despite the small loss in accuracy, pruning algorithms can still be used for executing very large models in low-end devices like mobile phones and smart devices saving on memory and computations.

Section 5: Team Contribution

All 3 members of the team took the task of analysing the problem statement and researching about the concept and implementation of pruning. Sakshi was responsible for executing the training of the base model and tuning of that model. Snehal and Vishnupriya jointly implemented the one shot and iterative prunings of the model and tuning of the hyperparameters. Sakshi contributed towards implementation of the command line arguments for the various sparsity and pruning techniques. Each member contributed equally towards the documentation.

Section 6: Artifact evaluation

The code for running the one-shot pruning algorithm can be seen in `RunProject2.ipynb` and the code for running the iterative magnitude pruning algorithm can be seen in `RunProject2_2.ipynb`.

To test the functioning of the code, run the `Test.ipynb` file. It runs the test.py file which runs all the implementations of the pruning algorithms for all the three sparsity ratios of 50%, 75% and 90%. The final results can be seen in the same ipynb notebook itself.