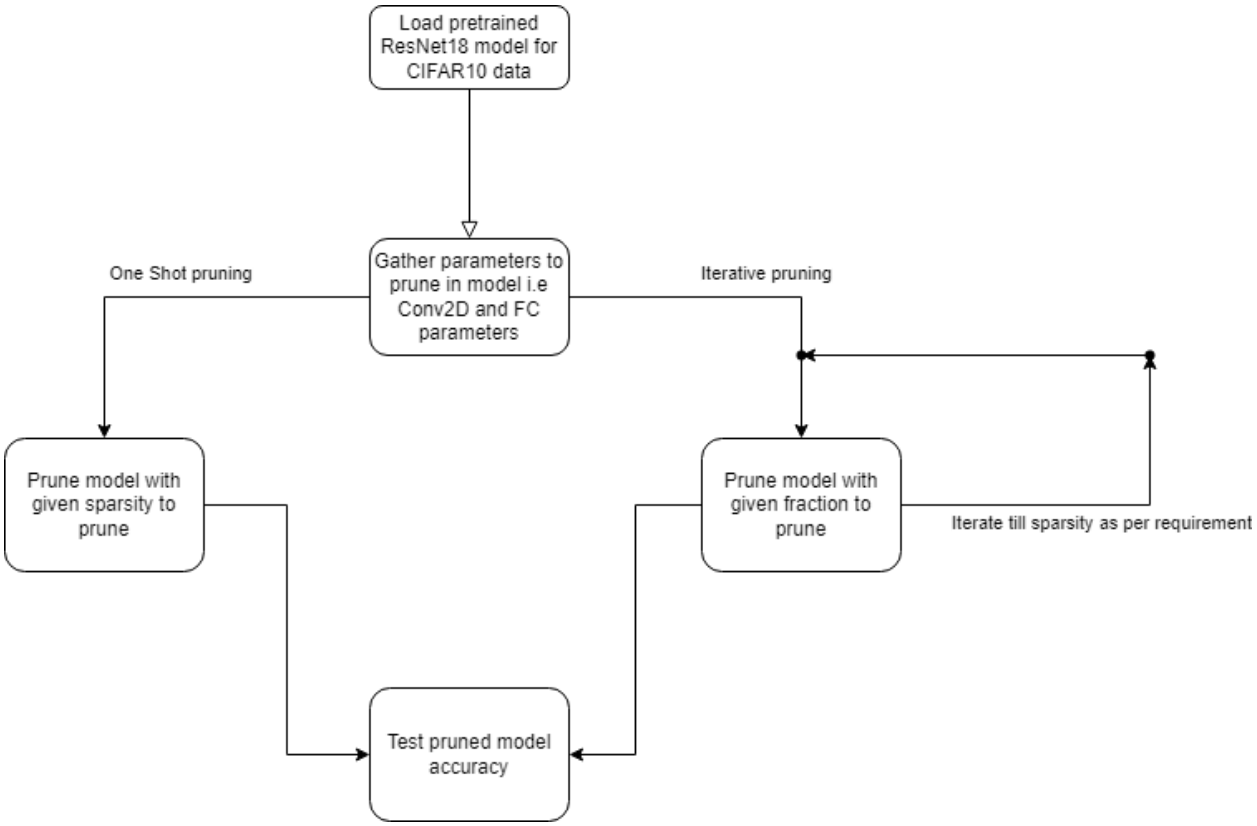
Project 2 Report - CNN Pruning Comparisons

Authors - Indrajeet Nandy, Nidhi Chandra, Yamini Kashyap

Section 1 - **Problem statement**

In Convolutional Neural Networks, we convolve a filter of certain dimensions continuously over the image. This results in multiple calculations being needed over every pass of the filter to yield desired results. As layers increase and layer input sizes get larger, this method becomes very inefficient. Especially when training such a model on a low end device, the time and resource requirement needed would be extremely large.
To solve this, we utilize pruning. This results in the lowest x% of connections in the model being pruned and leads to a much lighter model in terms of resource and computation time with only a marginal decrease in training accuracy.
In this project, we perform OneShot pruning and iterative magnitude based pruning with varying sparsity which results in marginal changes in accuracy with lowering of computational load. In one shot pruning, we prune in a one time operation manner and in iterative pruning, we iteratively prune a certain percentage of weights to achieve desired sparsity and accuracy.

Section 2 - **Implementation**



To implement this,

- **Overall Program Design**
    - We first take a pretrained model of ResNet18 on CIFAR10 dataset.

- We then decide which parameters should be considered for pruning. We allow pruning if the layer is either a Conv2D layer or a Fully Connected layer since in other layers such as Batch Normalization, pruning wouldn't have as much effect.
- We prune the model till we achieve the required sparsity.
- We then test the model's accuracy.
- This is repeated for all sparsity ratios for both Iterative and One-shot pruning.

- **How pruning algorithms implemented**
  - The source code for pruning can be found in the file **train.py**. The source code for loading and testing original and pruned models can be found in the file **load_and_test.py**.
  - For implementing One Shot pruning, we utilize PyTorch's `torch.nn.utils.prune.global_unstructured` API and pass in the parameters to prune which are all of the Conv2D or Fully Connected layers in the model and pass in the pruning method as l1 unstructured pruning with the amount being the sparsity that we want to achieve (say x) which are 50%, 75%, and 90%. Under the hood, this would then prune the parameters in the tensor with the lowest x% l1 norm values.
  - For implementing Iterative pruning, we loop over PyTorch's `torch.nn.utils.prune.global_unstructured` API 5 times and pass in the parameters to prune which are all of the Conv2D or Fully Connected layers in the model and pass in the pruning method as l1 unstructured pruning with the amount being the sparsity that we want to achieve (say x). The sparsity values passed in are 13%, 24.3% and 36.9%. Under the hood, this would then prune the parameters in the tensor with the lowest x% l1 norm values.
  - In iterative pruning, we achieve a final sparsity value from these small values since we are pruning iteratively. Final sparsity = $1 - (1-\gamma)^n$, where γ is the proportion of weights pruned in every iteration for n iterations.

Section 3 - **Experimental results**

Evaluation Methodology:

- To calculate the sparsity, we iterate over all layers of the pruned ResNet18 models using `torch.nn.Module.named_buffers()` and calculate the ratio of number of zero weights to total number of weights for each module that were Conv2D(2D convolution) or Linear(fully connected) layers. This is because the only layers of ResNet18 that are candidates of pruning are 2D convolutional layers and fully connected layers.

Significant Results:

Type of Pruning	Sparsity(%)	Test Accuracy(%)	Test Accuracy Drop(%)

Unpruned	0	93.069	0
One-shot	50	93.089	-0.02
One-shot	75	93.029	0.04
One-shot	90	92.528	0.541
Iterative	50	93.089	-0.02
Iterative	75	92.989	0.08
Iterative	90	92.538	0.531

Observations and Inferences:
We observe that the original ResNet18 model pretrained on the CIFAR10 dataset achieved an accuracy of 93.069% on the test data.
After pruning to a sparsity of 50%, we observe that the accuracy on test data indeed increased by a small amount 0.02%. This essentially means that the pretrained model is slightly overfitting on the training dataset and after pruning, it becomes a little more generalized so as to perform better on the test dataset.
After pruning to a sparsity of 75%, we observe a slight dip (<0.1%) in the test accuracy.
After pruning to a sparsity of 90%, we observe a slight dip (~0.5%) in the test accuracy.
The latter two metrics are expected because CIFAR10 is a simple dataset and ResNet18 is quite a complex Computer Vision model. All the layers and weights of the ResNet18 network are not essentially contributing towards achieving a good performance on the CIFAR10 test data split. Therefore, even pruning a large number of weights in the network does not cause significant accuracy loss.

Section 4 - **Summary and Takeaway**

To summarize:

- Original model has accuracy of 93.069%.
- We implement One Shot pruning for sparsity ratios 50%, 75% and 90% and observe accuracies 93.089%, 93.029% and 92.528% respectively and with iterative magnitude based pruning for sparsity ratios 50%, 75% and 90%, we observe accuracies as 93.089%, 92.989% and 92.538% respectively for a CIFAR10 image dataset using a ResNet18 model.
- We perform pruning by using PyTorch API `prune.global_unstructured()`.
- For One shot pruning we call the above API just once and for iterative pruning we call it multiple times till sparsity is as desired.
- We then test the models and compare accuracies with sparseness.

Our takeaways include:

- Since the accuracy of the original model was only ~93%, from these results we see that pruning to even as much as 90% sparsity results in an acceptably marginal accuracy drop in a model especially with the dataset being a CIFAR10 dataset with a ResNet18 model and gives us a much lighter model than an original computationally expensive one.
- Pruning therefore speeds up the model significantly with a very small drop in accuracy.
- This leads to less resource usage by the model and the ability to run the model in a lower specification machine as well.
- Pruning can also lead to an improved generalization of the model evident via test accuracy obtained for the 50% pruned models. Although this is counterintuitive, pruning in this case essentially prevents the model from overfitting on the training set.

Section 5 - **Team Contribution**

Yamini Kashyap's contributions and duties were:

- Getting one shot pruning to work and adjusting till target accuracy was attained.
- Looking into sparsity calculations of all approaches.
- Building report sections.

Nidhi Chandra's contributions and duties were:

- Getting iterative pruning to work and adjusting till target sparsity and accuracy were attained.
- Setting up the skeleton of code for plugging in pruning and model loading/testing.
- Building report sections.

Indrajeet Nandy's contributions and duties were:

- Setting up a pretrained ResNet18 model with CIFAR10 dataset.
- Building report sections.
- Setting up code for saving, loading and testing pruned and unpruned models.

Section 6 - **Artifact evaluation**

Steps to run the project:

1. Create a virtual environment where you can install the required libraries and run the pruned models:

    virtualenv -p python3 pruning
    source pruning/bin/activate

2. Install required packages:

    pip install -r requirements.txt

3. Run the shell script run.sh, which downloads the model and also runs load_and_test.py:

    source run.sh

run.sh first downloads pruned models from the google drive and then runs our main script for loading and testing pruned models (**load_and_test.py**).

The script uses PyTorch API to download CIFAR10 data and then tests the original model (unpruned) and prints its accuracy. Then it runs the pruned models one by one and prints their accuracies one by one.

1. one_shot_50.pth: The model with 50% sparsity pruned using one shot pruning
2. one_shot_75.pth: The model with 75% sparsity pruned using one shot pruning
3. one_shot_90.pth: The model with 90% sparsity pruned using one shot pruning
4. iterative_50.pth: The model with 50% sparsity pruned using iterative pruning
5. iterative_75.pth: The model with 75% sparsity pruned using iterative pruning
6. iterative_90.pth: The model with 90% sparsity pruned using iterative pruning