A Project on

CNN Pruning comparisons

by

Alok Kumar
Ritvika Pillai
Siddharth Nammara Kalyana Raman

INDEX

Section 1: Problem Statement

The goal of this project is to implement network pruning on a resnet18 model trained on the CIFAR10 dataset. Resnet18 is an eighteen-layer CNN model based on resnet architecture. We have to implement two types of pruning on this model **iterative pruning** and **one-shot pruning** with the sparsity of 50,75 and 90 for both cases. After pruning the models we need to retrain them again and evaluate them on various evaluation metrics such as

Sparsity = zero parameters/parameter

Test Accuracy = Correctly classified data points from test data/total amount of data points from test data

Accuracy Drop = test accuracy without pruning - test accuracy after pruning

Inference time = averaged time spent on the inference of one image

Speedup = inference time before pruning/inference time after pruning

Section 2: Implementation

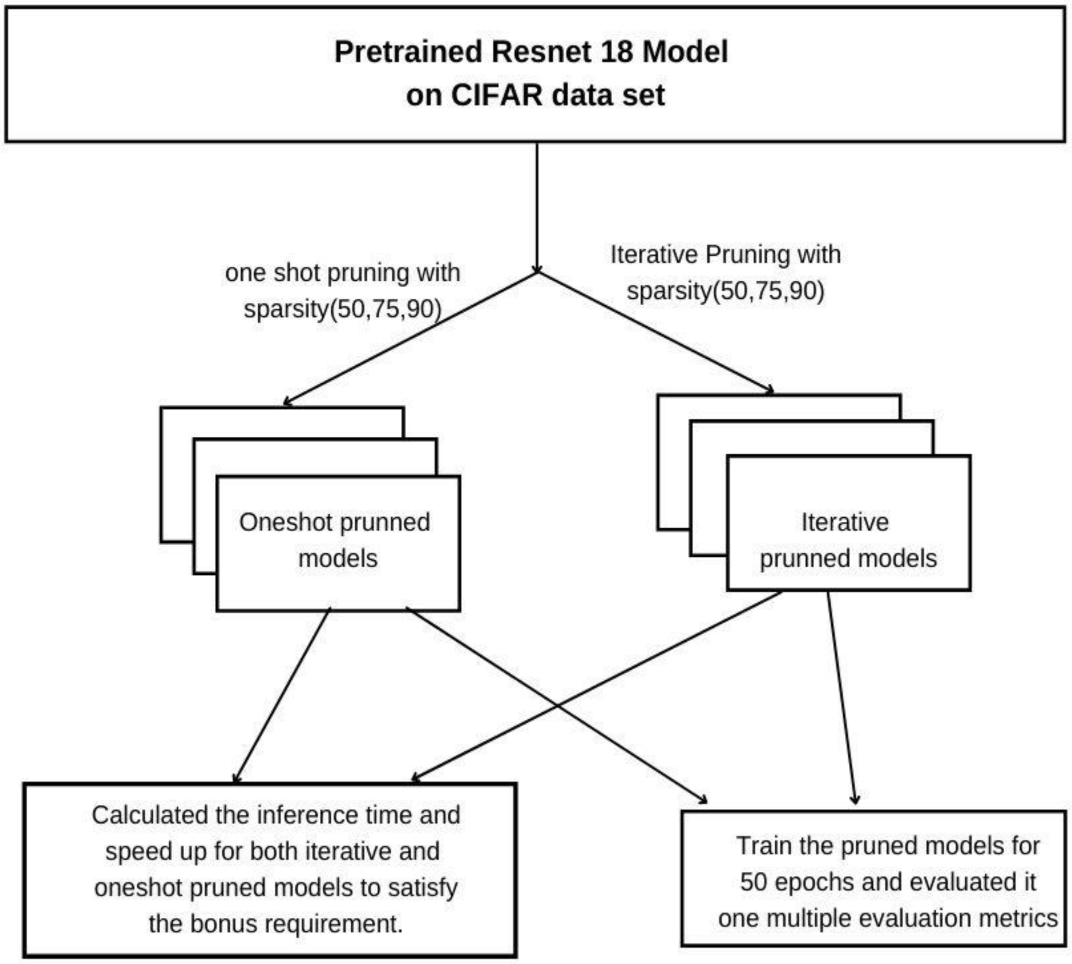

Pretrained Resnet 18 Model on CIFAR data set
one shot pruning with sparsity(50,75,90)
Iterative Pruning with sparsity(50,75,90)
Oneshot prunned models
Iterative prunned models
Calculated the inference time and speed up for both iterative and oneshot pruned models to satisfy the bonus requirement.
Train the pruned models for 50 epochs and evaluated it one multiple evaluation metrics

Fig 2.1: Block diagram for various steps in the implementation of the problem statement.

Fig 2.1 shows a rough block diagram of various steps that we took for implementing the problem statement. A detailed description of each of these steps is given below.

Steps for implementing the problem statement.

Steps for pruning the models

1. We loaded the weights of a resnet18 model on CIFAR10 dataset from the https://github.com/huyvnphan/PyTorch_CIFAR10 repository.
2. Downloaded the CIFAR10 train and test dataset using PyTorch's inbuilt torchvision dataset downloader.
3. Evaluated the pre-trained model on the CIFAR 10 dataset which gave an accuracy of ~93% on the test set.
4. Implemented one-shot pruning using **torch.nn.utils.prune** for three sparsity percentage (50,75,90)**.** The code snippet for the same is provided below.

```
prune.global_unstructured(
parameters_to_prune,
pruning_method=prune.L1Unstructured,
amount=0.5,
)
```

5. Reverified the sparsity of pruned models to check if everything is fine. The code snippet for calculating the sparsity of models is given below.

```
def calculate_sparsity(model):
  total_count = 0
  zero_count = 0

  for buffer_name, buffer in
model.named_buffers():
      zero_count += torch.sum(buffer ==
0).item()
      if zero_count>0:
      total_count += buffer.nelement()

  print("Total params: ", total_count)
  print("Zero params: ", zero_count)
  return
(math.ceil(zero_count*100/total_count))
```

6. Retrained the pruned models on the train set.
7. Implemented iterative pruning by running a for loop on top of a one-shot pruning API for three sparsity percentages (50,75,90). The code snippet for the same is provided below.

```
for i in range(5):
  prune.global_unstructured(
  parameters_to_prune,
  pruning_method=prune.L1Unstructured,
  amount=0.128,
  )
```

8. Verified the sparsity of the pruned models and retrained the pruned models for 50 epochs in each pruning iteration.
9. Saved the weights of pruned models in state dictionary.

Steps for Evaluating the model

1. Calculate the accuracy of models on the validation set. Code snippet for the same is provided below.

```
def test(epochs,model):
     val_acc=0
     val_loss=0
     val_correct=0
     val_total = 0
     with torch.no_grad():
     for batch_idx, (inputs,targets) in enumerate(testloader):
          inputs,targets = inputs.to(device),targets.to(device)
          outputs = model(inputs)
          loss = criterion(outputs,targets)
          val_loss += loss.item()
          _,predicted = outputs.max(1)
          val_total += targets.size(0)
          val_correct += predicted.eq(targets).sum().item()
          val_acc = 100.*val_correct/val_total
          if batch_idx%40==39:
               print("Evaluating....")
               print(batch_idx,len(testloader),'Loss: %.3f |
Acc: %0.3f (%d/%d)' %
(val_loss/(batch_idx+1),100*val_correct/val_total,val_correct,va
l_total))
```

```
    return val_acc
```

2. Calculated the accuracy drop by finding the difference of accuracy between unpruned and pruned models.

   Accuracy drop = test accuracy without pruning - test accuracy after pruning

3. Plotted the sparsity vs accuracy and sparsity vs drop-off graphs for better visualization of the results.

Implementing the bonus section of the problem statement

1. Calculated the inference time of pruned models and unpruned models which is equal to the time taken by the models for evaluating a single image.
2. Calculated the speed up which is equal to the inference time difference between unpruned and pruned models.
3. Plotted the sparsity vs inference time graph and sparsity vs speed up graphs for better visualization of the results.

Section 3: Experimental results

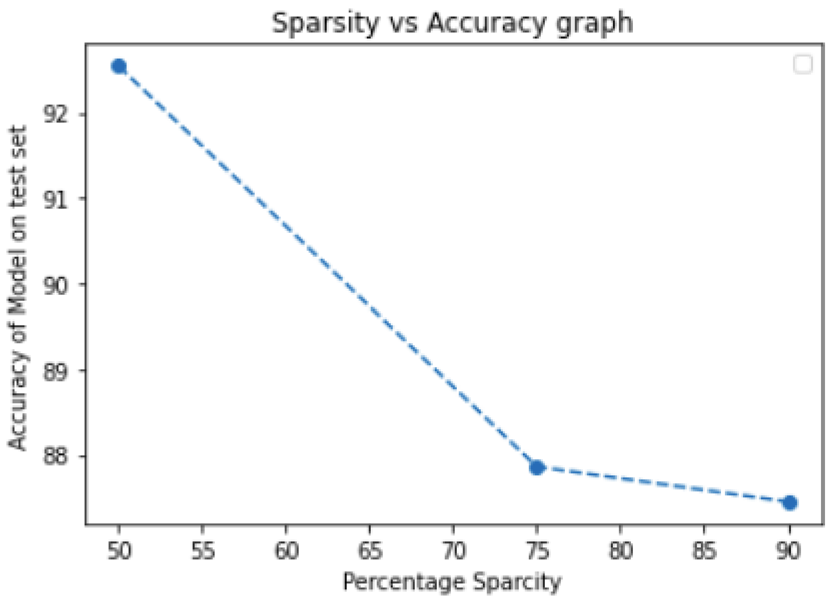
Accuracy of the unpruned model = 92.6

We calculated accuracy drop-off for pruned models relative to this accuracy.

Sparsity vs Accuracy

For one-shotPruning

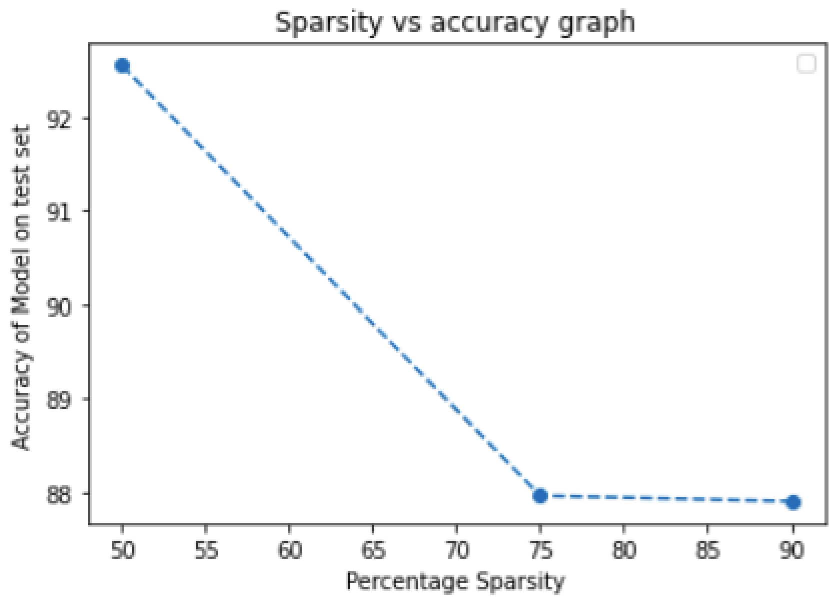
Sparsity	Accuracy
50	92.55
75	87.86
90	87.45



For Iterative pruning

Sparsity	Accuracy
50	92.55
75	87.97

90	87.91



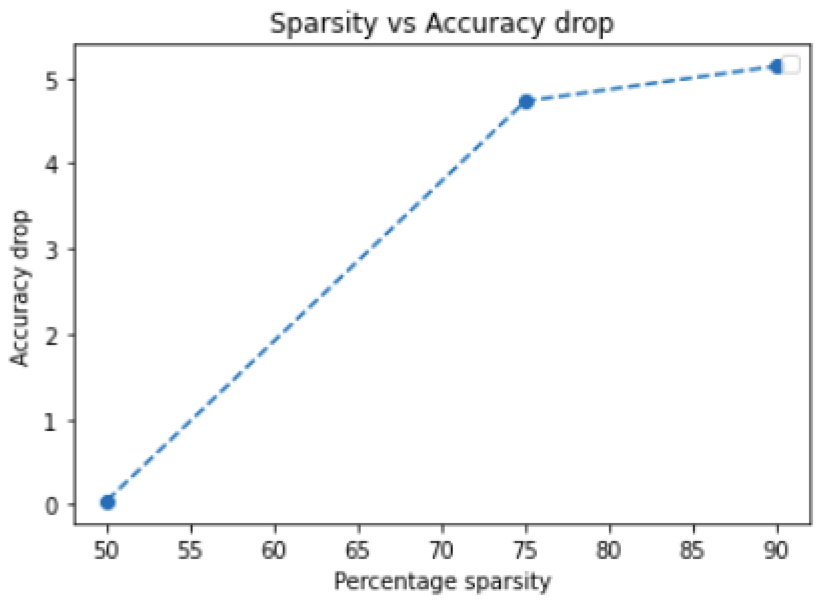
Inference

From the above table and the graph, we can see that as the sparsity ratio increases there is a decrease in the accuracy. This happens because when we increase the sparsity ratio we are increasing the number of zero parameters and decreasing the number of non-zero parameters which results in a drop in accuracy.

Sparsity vs Accuracy drop

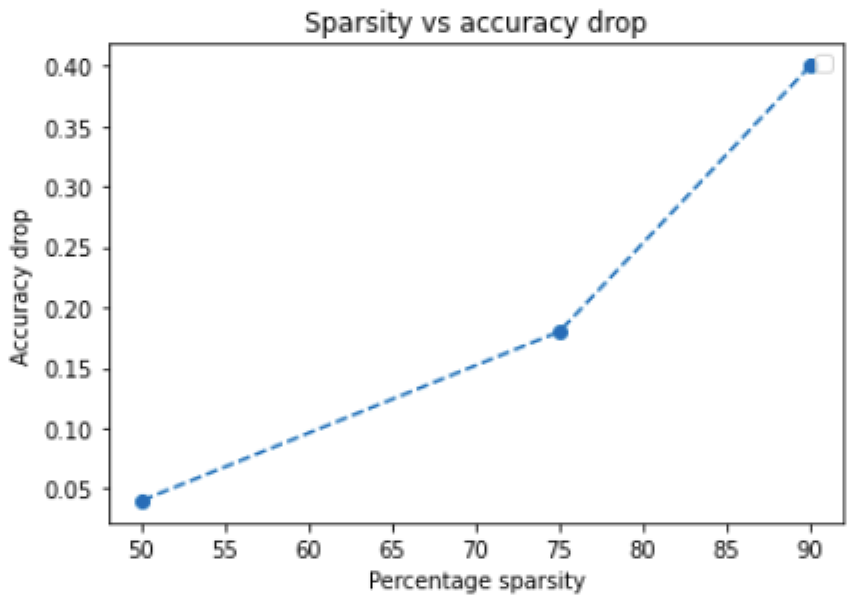
For one-shot pruning

Sparsity	**Accuracy drop**
50	0.04
75	4.73
90	5.14



For iterative pruning

Sparsity	**Accuracy drop**
50	0.04
75	4.62
90	4.68



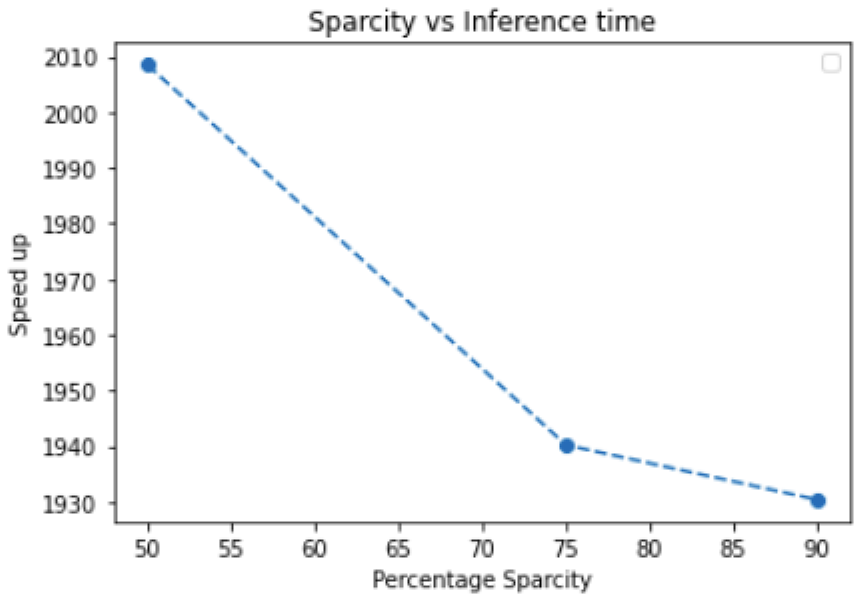
Inference

From the above table and graph, we can infer that as the sparsity ratio increases the accuracy drop increases. This is because as the sparsity ratio increases the accuracy percentage decreases and hence the accuracy drop increases.

<u>Sparsity vs Inference</u>

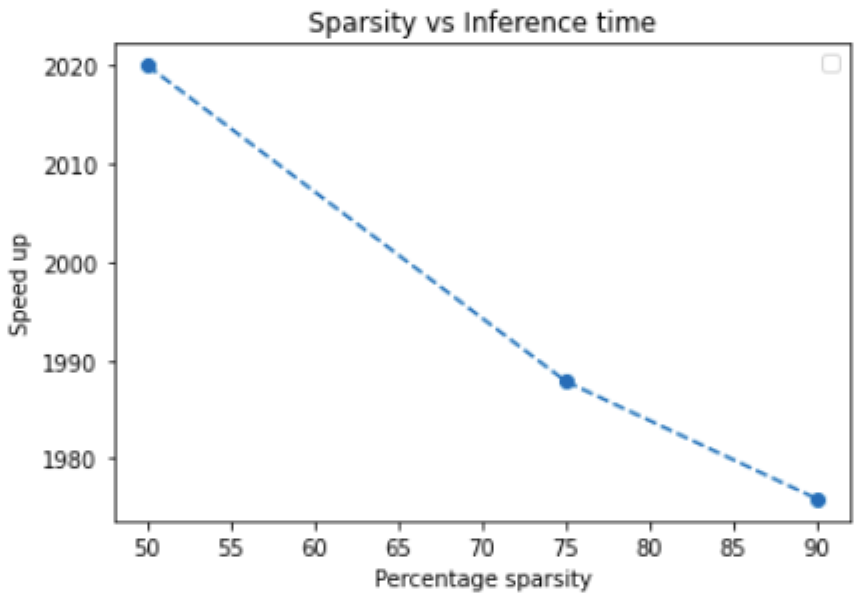
For one-shot pruning

Sparsity	**Inference Time**
50	1875.58
75	1798.71
90	1700.71



For iterative pruning

Sparsity	**Inference Time**
50	1882.50
75	1761.02
90	1732.05

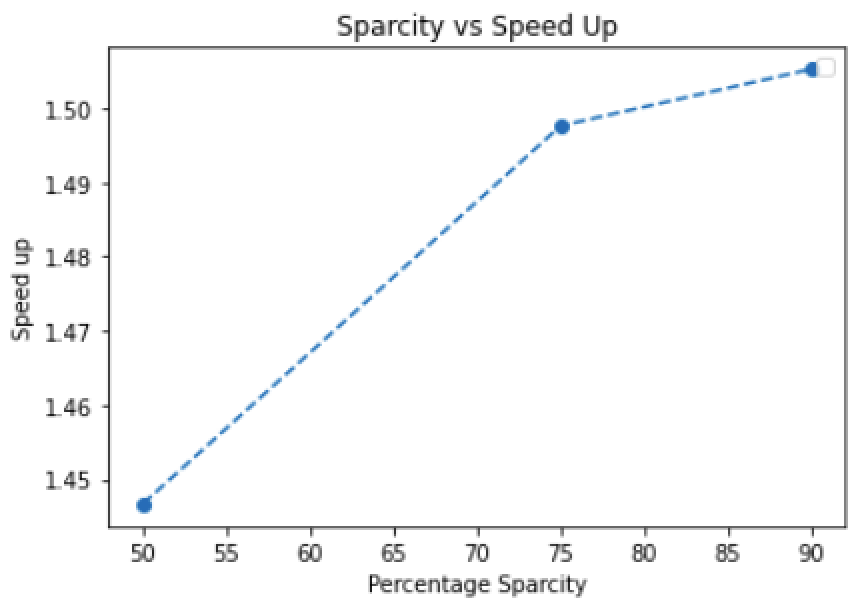


With the increase in sparsity ratio, the model becomes less complex which decreases the inference time of the model as is evident from the above results.

<u>Sparsity vs Speed up</u>

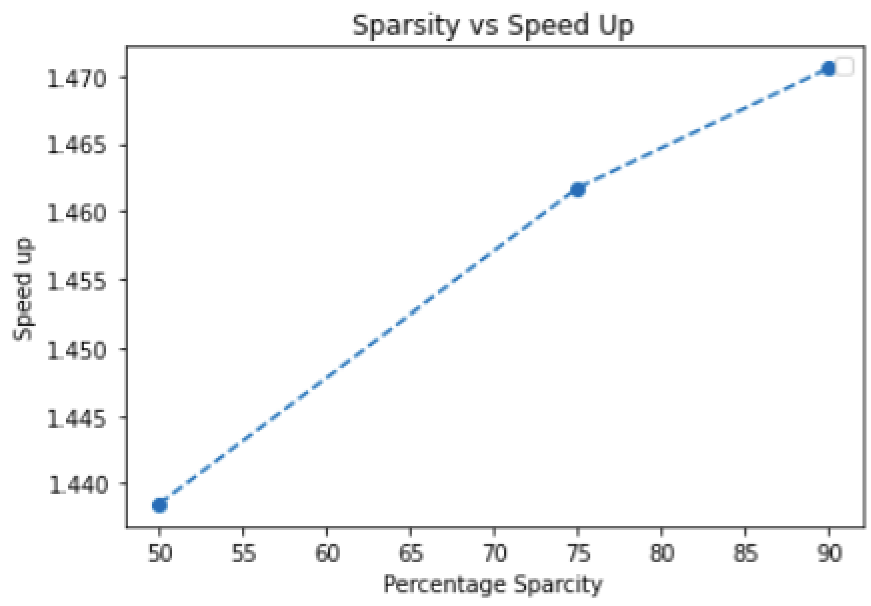
For one-shot pruning

Sparsity	**Speed up**
50	1.48
75	1.55
90	1.63



For iterative pruning

Sparsity	**Speed up**
50	1.42
75	1.52
90	1.55



As we increase the sparsity ratio the model becomes less complex and light weight which decreases the inference time hence increasing the speed of the model, the same is evident from the above results.

Section 4: Summary and Takeaway

In this project, we have taken a pre-trained model Restnet18. We have implemented two types of pruning which are one short pruning and iterative pruning. First, we take the model and calculate the weights which are least important. Here pruning is done globally that is over the entire model rather than layer by layer. So when you set a sparsity ratio of 0.5, 50% of the weights of the entire model is removed, not 50% in each layer. Similarly, there is another approach called iterative pruning where this process is done in every iteration. Pruning algorithms are basically a trade-off between the accuracy and complexity of the model. We give up a small amount of accuracy to make the model less complex and small in size so that it can be easily deployed on smaller systems with minimal resources.

Section 5: Team Contribution

Alok: Implemented One-shot pruning and calculated evaluation metrics like model sparsity, test accuracy and test drop. Visualized the results by plotting a graph between sparsity vs accuracy and sparsity vs accuracy drop. Wrote validation script for one-shot and iterative pruning. Worked on Documentation.

Ritvika: Implemented Bonus question by calculating inference time and speedup for both one-shot pruning and iterative pruning. Visualized the results by plotting a graph between sparsity vs inference time and sparsity vs speedup. Worked on Documentation.

Siddharth: Implemented Iterative pruning and calculated evaluation metrics like model sparsity, test accuracy and test drop. Visualized the results by plotting a graph between sparsity vs accuracy and sparsity vs accuracy drop. Worked on Documentation.

Section 6: Artifact evaluation

Steps to run the model:

A. On Google Colab (Recommended)

System requirements : A google collab account with GPU setup as a hardware accelerator.

Evaluate the model

Evaluating the model is quite simple. You just need to follow the steps given below.

1. Clone the repo from github using
   git clone https://github.com/cs532-2021-fall/cnn-pruning-status_200.git

2. Downloads all the pre-trained models from the drive link

https://drive.google.com/drive/u/1/folders/1shjh5fCQ_UZZuVOVNWs9xv8i1w5PWMaZ

3. Unzip the downloaded file which will give you a folder named **state_dicts.** This folder contains all the pre-trained models including original unpruned models and pruned models with different sparsity ratios. If downloading went right then you should be able to 7 pre-trained models in this folder, 6 pruned models for various sparsity ratio,s and one unpruned original model.
4. Place this folder inside **cifar10_models** directory which is present at the project root level.
5. Upload the whole project to your google drive so that it can be accessed from colab
6. To Evaluate the pruned models and generate all the results shown in the result section you just need to run the following two notebooks

Validate_iterative_pruning.ipynb and **validate_one_shot_pruning.ipynb**

7. Make sure to make any changes in the mount script provided in the notebook based on the location of the project on your drive to avoid any errors. The code snippet that may require modification is provided below.

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
FOLDERNAME = 'cs532/project_2_cnn_pruning/'
%cd drive/My\ Drive/$FOLDERNAME
```

Prune and Tain the model

Although pre-trained models are provided for each of the sparsity ratio for both iterative and one-shot pruning and they should be good enough for the purpose of visualization but if you still want to train the model from scratch. Then it can be done using the following steps.

1. Run oneshot_prunning.ipynb and iterative_prunning.ipynb which will train the models. The training time should take around 3-4 hrs if done parallelly on GPU.
2. Once the model is trained same steps can be used to evaluate the model.