Deep Compression {Song Han}

1. Pruning

a. Codes to use : **pruning.py, prune.py, models.py**

b. First, we **run the pruning.py**

   i. which first checks for availability of CUDA.
   ii. It creates the train dataset.
   iii. Set the hyper-parameters of batch_size, validation set.
   iv. Then the validation dataset is created using the validation split.
   v. Then, we load the model (in this case it is LeNet-5) which is given with pretrained as **False**. This call is explained when talking about **models.py**
   vi. Change the last fc layer of the model to fit the particular dataset at hand(so made the last fc layer to have 10 classes according to MNIST dataset).
   vii. Then convert the model to CUDA and then train the model
   viii. With that training, we test it on the validation set and obtain the results as shown in **Fig 1**
   ix. Then, we call the **prune_by_std(sensitivity)** in **models.py** where sensitivity tells about the number of weights that can be pruned as shown as **Fig 2**
   x. After pruning, we again re-train the model and obtain the output for the test-set from there again as shown in **Fig 3**

c. **Note: In this particular example, masking is done the fc layer which zeros out the weights below a threshold whereas not much is done to convolution layer(just the gradient is set to 0 when the weights are 0).**

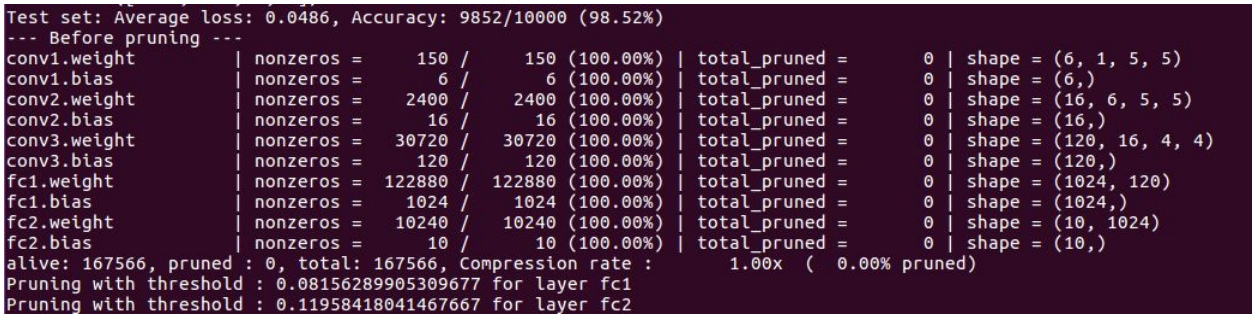

Fig 1 The result obtained when we ran the model on the test-set before pruning

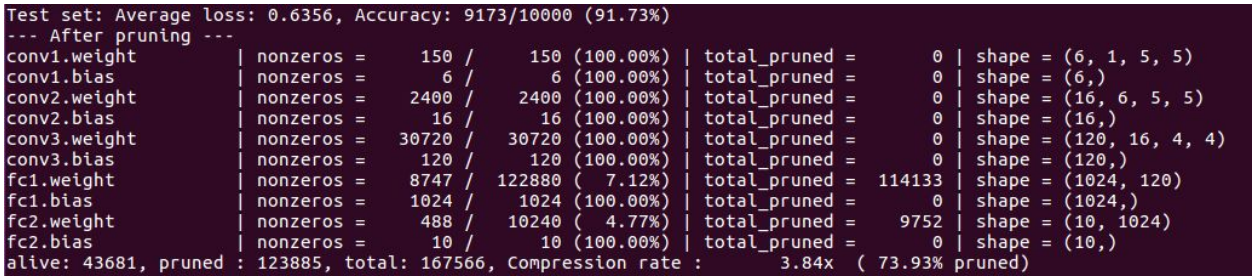

Fig 2 This is pruning which compresses the model size.

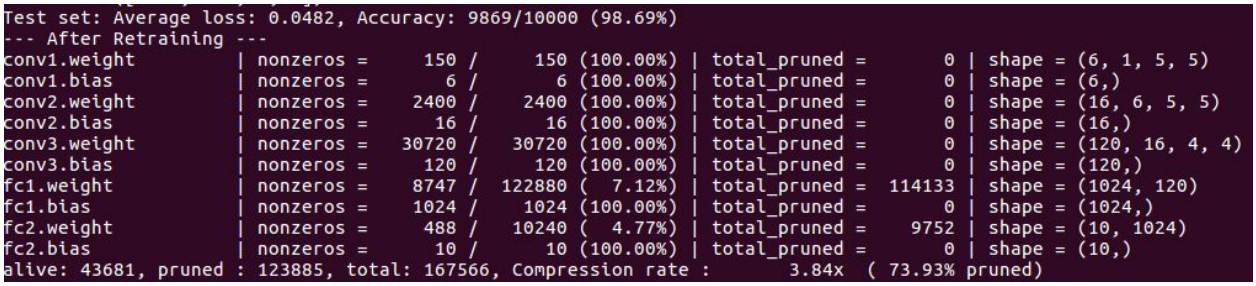

Fig 3 This is the accuracy obtained after pruning the fc layers in LeNet model.

In **models.py file**, in the case of LeNet, we have defined the architecture of LeNet and in this model, the Linear layer at the end is replaced with a **maskedLinear layer(present in the prune.py file)** which places a mask layer between the weights and the bias(if exists).
The maskedLinear class was skipped from pruning
This maskedLinear layer was initialized with all 1's and then when run, it multiplied the weight matrix with the mask matrix(element-wise) and obtained the new weight matrix.
After performing pruning, where the threshold was calculated using the np.std of the weights and multiplied with the scarcity value, the mask was set to zero if the particular weight was less than the threshold which meant that those weights are pruned away which gave a lesser model size.

Pruning Resnet:

- Load the model with pretrained weights.
- Load the model Architecture with the calling the *PruningModule* **Class object** of **prune.py** file which contains the threshold on which we have to pruning.
- We also mask the linear/fc layer of model so that we can prune it based on threshold.
- On training the model with dataset, we are zeroing the gradient zero whose value is zero.
- Then we will test the the accuracy.
- We compare the accuracy before and after pruning.
- Finally we retrain the model with pruned weights and compare the accuracy it shouldn't be less than initial.

Observation:

Some layer in ResNet18 are of sequential type so need to parse layer and do quantization and also we need to store the weight in same order in order to preserve the structure of model.

From the paper : https://arxiv.org/pdf/1711.05908.pdf , we find that when pruning independently the least important ones in the first few layers could have a good impact on the important weights of the future layer. So it is better to do it **globally (to get a good understanding of what is not needed) than doing the pruning locally(independently across layers)**

From the paper: https://arxiv.org/pdf/1707.01213.pdf, Han et al pruned the weights whose absolute value are smaller than a given threshold. This approach requires iteratively pruning and fine-tuning which is very time-consuming. To tackle this problem, Guo et al. proposed dynamic network surgery to prune parameters during training.
2 main problems with pruning:

- The first issue is the possibility of irretrievable network damage. Since the pruned connections have no chance to come back, incorrect pruning may cause severe accuracy loss. In consequence, the compression rate must be over suppressed to avoid such loss.
- Another issue is learning inefficiency. As in the paper [9], several iterations of alternate pruning and retraining are necessary to get a fair compression rate on AlexNet, while each retraining process consists of millions of iterations, which can be very time consuming.

Han et al. explore the magnitude-based pruning in conjunction with retraining, and report promising compression results without accuracy loss. It has also been validated that the sparse matrix-vector multiplication can further be accelerated by certain hardware design, making it more efficient than traditional CPU and GPU calculations [7]. The drawback of Han et al.'s method [9] is mostly its potential risk of irretrievable network damage and learning inefficiency. When considering techniques like taking the diagonal elements of the Hessian matrix for the pruning, we find that it takes a lot of time to process for larger networks.
Pruning and Slicing: which makes use of the second derivatives of the loss function to balance training loss and model complexity.

TRAINED TERNARY QUANTIZATION

Han et al. (2015) proposed deep compression to prune away trivial connections and reduce precision of weights. Unlike above models using zero or symmetric thresholds to quantize high precision weights, Deep Compression used clusters to categorize weights into groups. In Deep Compression, low precision weights are fine-tuned from a pre-trained full precision network, and the assignment of each weight is established at the beginning and stay unchanged, while representative value of each cluster is updated throughout fine-tuning.

https://paperswithcode.com/task/model-compression