Pruning in Deep Neural Network

Zhong Tian Ou Yang
Dept. of Computer Science
Columbia University
New York, US
zo2145@columbia.edu

Shusen Xu
Dept. of Computer Science
Columbia University
New York, US
sx2261@columbia.edu

***Abstract*—In this paper, We first introduced some basic concepts related to pruning and some technical challenges we faced during our experiments. Then we show the results from experiments of various pruning methods on a variety of deep neural models and evaluate the performance and efficiency of the pruned networks. The models we used are AlexNet, BERT, VGG and ResNet. After pruned each model, we compared their performances with the original models. In the case of structured pruning, we also evaluated the inference time of the pruned model on GPU. Finally, we show some insights drawn from our observations.**

I. Introduction

The Goal of our project is to evaluate different pruning methods on multiple popular network architectures, and identify which architecture is benefits more from pruning and the best pruning method for each architecture. As neural networks become more capable in recent years, the number of parameter increased significantly, from 60K parameter in the classical LeNet-5, to 175B parameters in the just released GPT-3. While being very powerful, these recent networks require a hefty amount of resources to compute. What is worse is that many of these parameters are not fully utilized. By pruning the networks, we can lower the barrier for them to run on edge devices and make it more available to people in need. By doing this project, we can get a more detailed understanding of pruning techniques. We can also gain insights into the benefits of pruning and difficulties involved to do so. Through our experiments, we also identify and compare the pros and cons of difference pruning method and settings.

II. Background Work

1) Iterative pruning
   A major consideration in pruning is where to put it in the training/testing machine learning timeline.For example, if you are using a weight magnitude-based pruning approach, you would want to prune after training. However, after pruning, you may observe that the model performance has suffered. This can be fixed by fine-tuning, meaning retraining the model after pruning to restore accuracy. Iterative pruning methods are brought about based on the above intuition. As the Figure shows, in iterative pruning, it alternate iterations of pruning and further fine-tuning until reaching the target trade-off between accuracy and pruning objective, e.g. inference time(FLOPs) or memory utilization. Also, there are different heuristics and methods of determining which nodes are less important and can be removed with minimal effect on accuracy. You can use heuristics based on the weights or activations of a neuron to determine how important it is for the model's performance. The magnitude(such as L1 or L2 norm) of weights and activations is a good choice.

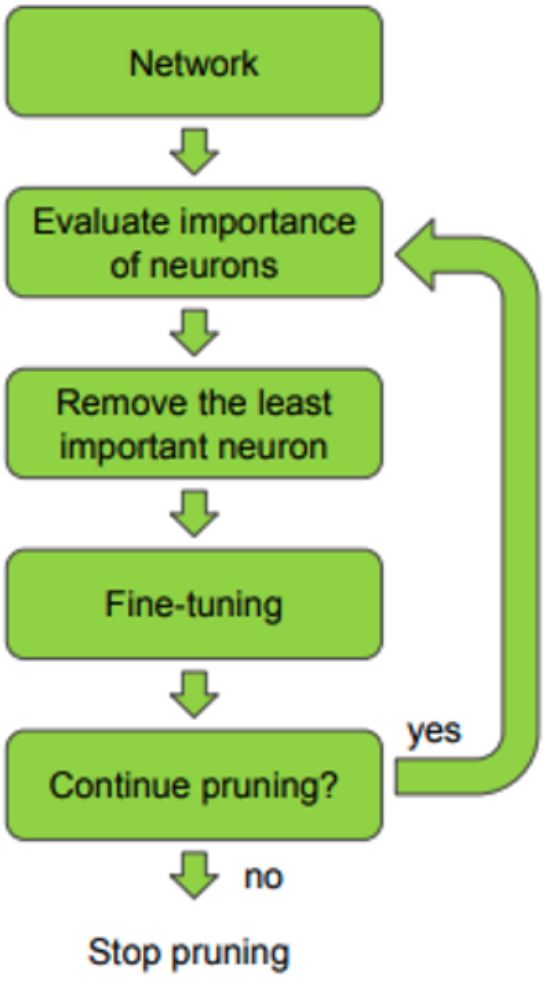

Fig. 1. iterative architecture

2) Structured and Unstructured pruning

In machine learning, pruning is removing unnecessary neurons or weights. There are two kinds of methods to prune a neural network. One is unstructured pruning which removes weight connections by setting individual parameters to zero and making the network sparse. This would lower the number of parameters in the model while keeping the architecture the same. Another is structured pruning which removes entire groups of nodes such as entire channels or filters. Thus, structured pruning has the effect of changing the input and output shapes of layers and weight matrices, aiming to make the network architecture itself smaller and keep the accuracy of the initial larger network.

3) Pytorch Pruning

For PyTorch built-in unstructured pruning method (such as prune.l1_unstructured) generates binary weight mask(1,0) saved as named-buffers. Pruning is applied prior to each forward pass using PyTorch's forward_pre_hooks, computing weights by combining the mask with the original parameters, introducing the effects of removing connections/weights.

Structured pruning (such as prune.ln_structured) prunes by removing the specified amount of(currently unpruned) channels along the specified dimension. It modifies module in place by adding a named buffer and applies binary mask to specific channels or filtering when computing. However, the implementation of structured pruning is not ideal since it only maskes weight, not bias. Actually, it is not actually structured pruning since masking does not brings about any variation in terms of size and shapes, compared to original model.

For realize actually structured pruning, we introduces a pytorch toolkit for structured neural network pruning and layer dependency maintaining. This tool will automatically detect and handle layer dependencies (channel consistency) during pruning. This package will run your model with fake inputs and collect layer information. Then a dependency graph is established to describe the computational graph. When a pruning function is applied on certain layer through DependencyGraph.get_pruning_plan, this package will traverse the whole graph to fix inconsistent modules such as BN. The pruning index will be automatically mapped to correct position.

4) Models explain(illustrate model architecture)
In our experiments, we run various pruning methods on a variety of deep neural models and evaluate the performance and efficiency of the pruned networks in order to get insights about how the variation of architecture affect the pruning effects. The models we used are AlexNet, BERT, VGG16 and ResNet. For different models, the strategy and operation can vary a lot when pruning

AlexNet and VGG16 have similar architectures, consisting of Convolutional (CONV) layers and Fully Connected (FC) layers with different depth and shape of kernel. However, ResNet contains special modules, making use of shortcut connections and residual block to solve the vanishing gradient problem. And the BERT, as a transformer-based model with attention, is also another story.

III. Technical Challenges

- Currently, pruning is not well optimized by major frameworks such as TF/Keras and Pytorch. With unstructured pruning, even though we can implement the pruning method using masks and evaluate the accuracy of the model after pruning, we won't be able to observe the improvement in model size and computation time. Actually structured pruning in PyTorch is unreasonable, resulting in "incorrect" pruning fraction and wrong number of parameters left after pruning. As the figures 2 shows, the number of parameters left (not masked-out parameters) for various pruned fraction is significantly more than the actually(ground-truth) pruned models. The main reason is that the structured pruning methods in PyTorch does not pay attention to the dependency of modules, removing the neurons and changing the shape of one module such as channels or filter may continuously need to modify other modules' shape, thus more reduction of parameters and neurons. What's more, another reason is that structured pruning in PyTorch only masked weight, not bias, so we need to retrain after we actually prune the whole filter.

- Variation of models' architectures brings about huge influence on pruning effects. Pruning ResNet is not working as expected due to the skip connections and residual block. RNN for NLP, which we originally proposed, is not very suitable for pruning no pre-trained model, so we replace it with BERT model.

- Deciding the correct hyperparameters and setups for pruning (pruning fraction, number of iterations, which part to prune) is quite complex, requiring rounds of examination. Aslo it is quite hard to decide which metrics to use to evaluate the result . Since we are evaluating different networks with different datasets, we need to choose a representative metric to determine the reduction in performance after pruning.

IV. Approach

For each can model, we will first train the model from scratch or fine-tune a pre-trained model to a specific task

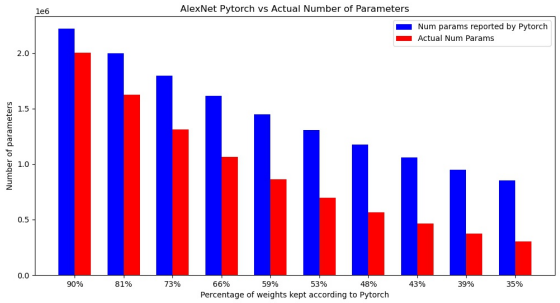

Fig. 2. Number of params: Pytorch vs Actual

and measure its accuracy. Then, we will start pruning and fine-tuning those models iteratively to observe how the accuracy of model change as more and more of the module are pruned. The pruning process is repeated multiple times with different hyperparameters and settings to experiment the tradeoffs between different choices. Last we Examine the overall results and provide insights based on our observations

We will also consider different variants of pruning methods. For unstructured pruning, we adopting globally pruning of weights/connections using random and L1 norm to decide the importance of each weight. For structured pruning, we prune only CNN layers respectively removing specified amount of channels. We evaluate the performance considering two main criterion, one is accuracy kept over various fractions of parameters(sparsity) left for both unstructured and structured pruning. Another is inference time over various fractions of parameters left(for CNN layers).

For structured pruning, we adopt two methods and compared their performance. First we take the PyTorch built-in structured pruning methods on each CNN layers, which is based on weight mask, not actually changing the shape of original model, so evaluating the variation inference time over various fractions of parameter makes no sense for this method. Torch-Pruning inspires another idea which brings about ground-truth pruned model and improvement in inference time. The detailed design is as follows:

- Determined the filters to prune in each convolutional layer using L2 norm
- Build dependency graph of the model
- Remove the pruned weights and their descendents from the weight matrix
- Finetune the model again to compensate the missing bias values

V. Implementation Details

A. Hardware

For training, any GPU should do the trick for us. During our experiment, we used RTX 2070, Tesla P100, Tesla V100. Each GPU is equipped with appropriate CPU cores to fully utilize it. We only benchmark the inference time on structurally-pruned models on GPU.

B. Software

For the ML framework, we used Pytorch installed in an Anaconda environment. Some other python packages we used are Transformers [1] package for BERT, and torch_pruning [2] package for actually prune specific filters from the convolutional layers. For AlexNet, VGG and ResNet the implementation we used are the ones included in Pytorch. All models are using pre-trained weights.

C. Dataset

The Dataset will be based on the model we are testing. For image classification test, we fine-tune the models with weight trained on ImageNet to CIFAR-10 classification task. For BERT, the model is pretrained on Toronto Book corpus and Wikipedia dataset. The dataset we used to fine tune BERT is SST-2 [3]. SST-2 is a classification task consists of 70K samples of sentences from movie reviews along the corresponding labels, which are the sentiments (either positive or negative) of the sentences.

D. Github Repository

Here is our github repository https://github.com/leoouyang/PracDL_Final_Project. All the codes we used for finetuning models, pruning models and plot the results are included in the repository. We also include the results matrices stored as text files in our repository. Due to time limit, we are not able to finish structured pruning of BERT, ResNet and any linear layers.

VI. Results and Evaluations

In this section, we show the accuracy and inference time results we got from the pruned models using plots and tables. We then discuss these results and what information is provided by those results. Since the results for AlexNet, VGG and ResNets are similar, we will mostly focused on the AlexNet results.

A. AlexNet Unstructured Pruning Results

Table I illustrates the sparsity in each layer of AlexNet after pruning. The sparsity table suggests that most of parameters pruned are the weights of the first two linear layers in the classifier of AlexNet. This is expected since those two layers contains most of the parameters in the model. And, compared to ResNet's classifier, AlexNet's classifier has much more parameter. Given that ResNet has a better performance on ImageNet, it is very likely that AlexNet classfier is over-complicated with many close to zero weights.
Figure 3 is a plot of AlexNet accuracy after each iteration of pruning. We can observe that after pruning, it is critical to fine tune the model again to restore the performance of model. With fine tuning, the model can preserve a similar accuracy on the test set with only 12% of connections kept in the model. One interesting effect we can observe from the plot is the pruning have a similar behavior to regularizers. As

TABLE I
SPARSITY IN EACH LAYER OF ALEXNET

Layer	Num of Params	S=50%	S=75%	S=87.5%	S=93.75%	S=96.87%	S=98.44%
Conv1.Weight	23232	9.28%	15.71%	21.18%	26.09%	30.90%	36.76%
Conv1.Bias	64	1.56%	4.69%	6.25%	7.81%	7.81%	15.62%
Conv2.Weight	307200	18.51%	31.54%	41.73%	50.50%	58.73%	67.10%
Conv2.Bias	192	3.12%	6.25%	6.77%	8.33%	8.85%	13.54%
Conv3.Weight	663552	18.17%	31.34%	42.01%	51.44%	60.33%	69.83%
Conv3.Bias	384	3.91%	7.55%	42.01%	14.06%	18.75%	22.66%
Conv4.Weight	884736	20.57%	35.28%	47.14%	57.42%	67.06%	76.87%
Conv4.Bias	256	1.17%	3.12%	47.14%	5.47%	5.86%	7.81%
Conv5.Weight	589824	20.90%	35.92%	47.90%	58.30%	67.98%	77.79%
Conv5.Bias	256	3.91%	5.47%	7.81%	9.77%	12.50%	13.28%
Linear1.Weight	37748736	53.49%	79.93%	92.04%	97.23%	99.25%	99.87%
Linear1.Bias	4096	24.34%	39.45%	52.15%	63.33%	72.97%	82.74%
Linear2.Weight	16777216	46.72%	70.11%	83.67%	91.76%	96.48%	98.96%
Linear2.Bias	4096	10.67%	18.65%	83.67%	31.54%	38.77%	48.14%
Linear3.Weight	4096000	18.40%	31.58%	42.50%	50.72%	58.28%	66.33%
Linear3.Bias	1000	0.00%	10.00%	20.00%	30.00%	30.00%	30.00%

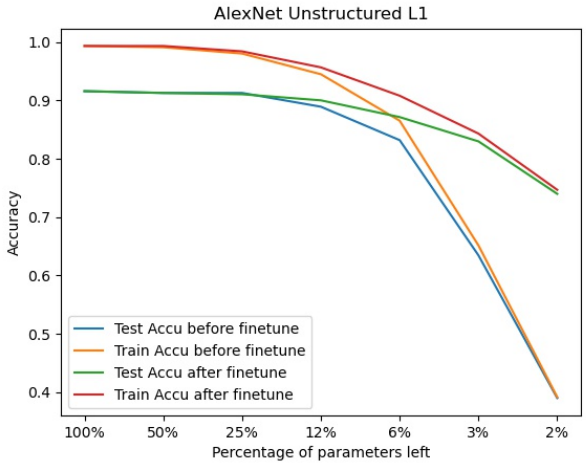

Fig. 3. Performance of unstructurally pruned AlexNet

more and more connections are pruned away from the model, the difference between training accuracy and test accuracy narrows. One possible explanation for this effect is that the connections that make the model overfit the training set tend to have smaller L1 norm. By removing the connections with the smallest L1 norms, we remove those connections.

To measure the effectiveness of using L1 norm when deciding

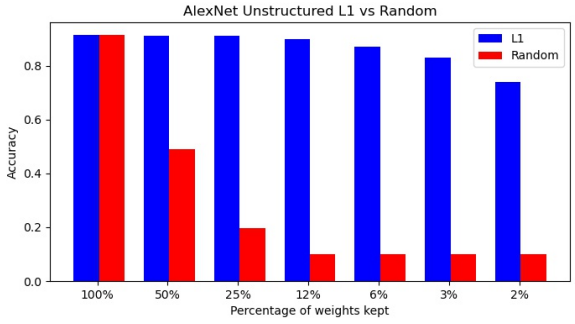

Fig. 4. Test Accuracy of L1 vs. Random

which connections to prune, we also prune the same AlexNet randomly. We can observe from Figure 4 that using L1 norm yields much better results.

Table II illustrates the difference in performance of the pruned model when using iterative pruning and direct pruning. Overall, the model pruned using the iterative method has slightly better performance. With direct pruning, the model is pruned to the designated sparsity in one iteration. In comparison, with iterative pruning, a fraction the weights are pruned in each iterations, and the process of pruning and fine tuning repeats until the sparsity of model reach the goal. In our case here, the fraction we used is 50%. With this iterative approach, the weights are re-calibrated between every pruning and thus the connections removed are more likely to be the less important ones.

TABLE II
PERFORMANCE OF ALEXNET WHEN SPARSITY = 93.75%

Model	Iterative pruning	Direct pruning
Test Accu. before fine-tune	83.19%	68.74%
Train Accu. before fine-tune	86.52%	71.19%
Test Accu. after fine-tune	87.15%	86.83%
Train Accu. after fine-tune	90.81%	89.86%

B. AlexNet Structured Pruning Results

Figure 5 shows the performance of the pruned model before and after fine tuning for each iteration of pruning. The plot is different from the unstructured one (Fig 3) in several ways. Firstly, the performance drops more rapidly compared to the one from unstructured pruning even though we are using a much smaller prune fraction here. Also, the performance before fine tuning is very unstable. Fortunately, it can be easily restored by the fine tuning process. Lastly, we can not observe the same regularization effect as in unstructured pruning.

One major advantage of using structured pruning though, is the improvement of inference time on both GPU and CPU. With unstructured pruning, only improvement on CPU is possible, and it is time consuming to do so. To actually prune the AlexNet, we used the torch_pruning package mentioned before. The experiment result are shown in Figure 6 and 7. We can see that as the batch size increase, the inference time

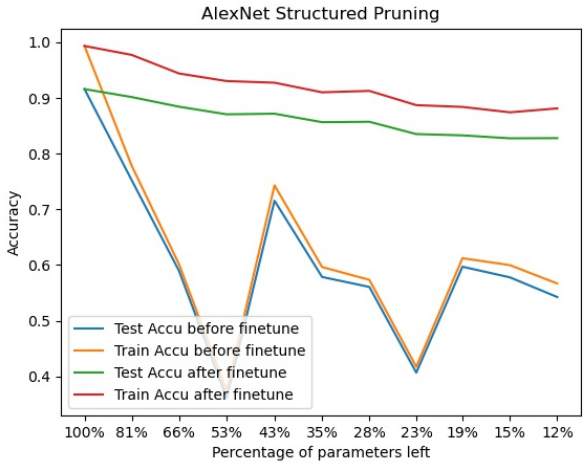

Fig. 5. Performance of structurally pruned AlexNet

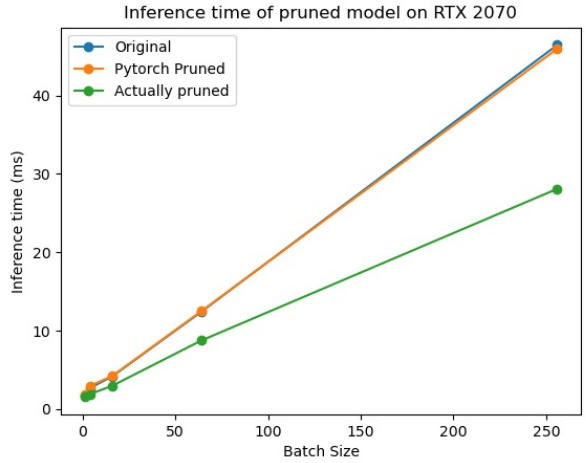

Fig. 6. AlexNet Inference time vs Batch Size

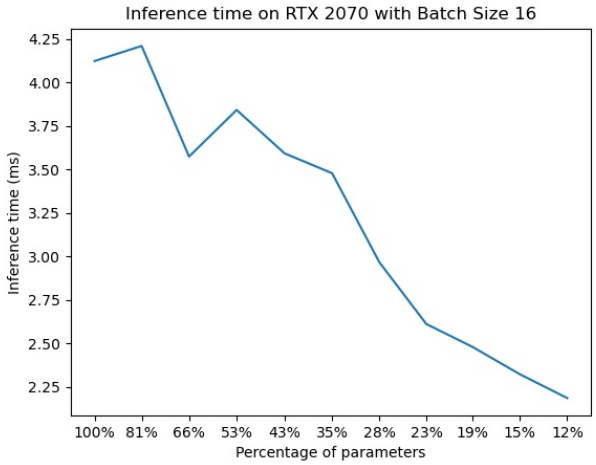

Fig. 7. AlexNet Inference time vs Parameters

increase linearly. The Pytorch pruned model and the original model have almost identical inference times, while the actually pruned model's inference is only two thirds of that. And with batch size 16, the inference time continuously decreases as more and more parameters are pruned away from the model.

C. BERT Results

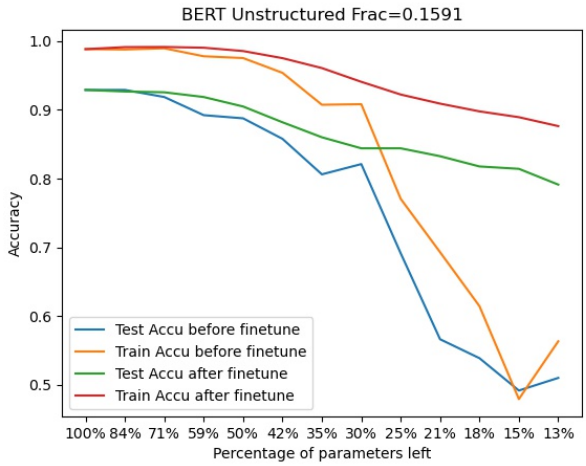

Fig. 8. Performance of unstructurally pruned BERT

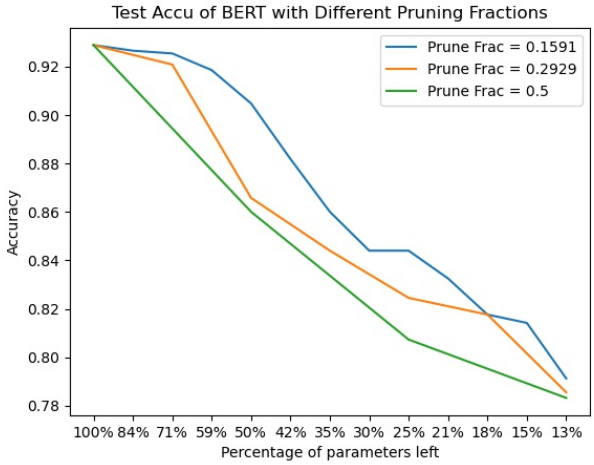

Fig. 9. BERT performance with different prune fraction

From Figure 8, it is obvious that BERT's performance drops more rapidly as parameters are pruned from the Model. This could indicate that NLP models are not very suitable for pruning. We also tested pruning the BERT model with different prune fraction. As shown in Figure 9, model pruned with a smaller prune fraction has slightly higher accuracy on test set but requires more iterations to prune the same percentage of parameters.

D. VGG16 Results

We won't show plot for VGG16's results here since they are very similar to AlexNet's results, only a a better performance overall. The inference time for structurally pruned VGG16 model also resemble the ones from AlexNet.

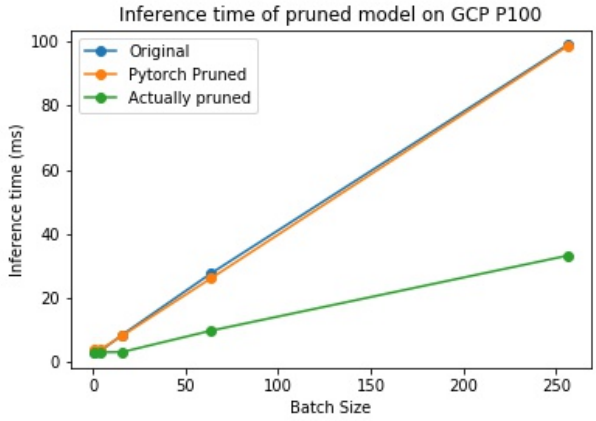

Fig. 10. Inference time of ResNet18 (Sparsity = 6.27%)

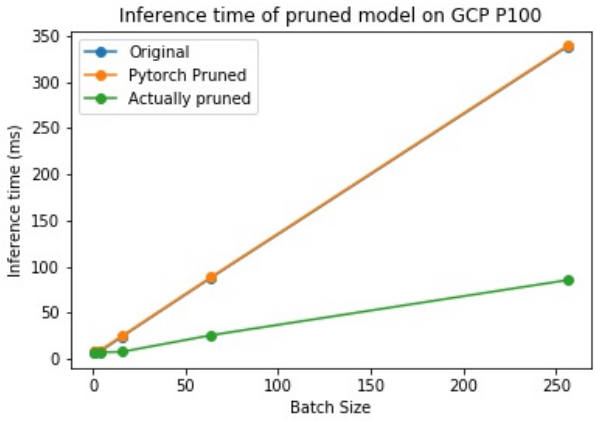

Fig. 11. Inference time of ResNet50 (Sparsity = 17.21%)

E. ResNet Results

The unstructured pruned ResNet18 and ResNet50 also yields performance curves similar to the one for AlexNet (Fig. 3.)

For structured performance, we can not get an correct accuracy vs parameters plot since the implementation of torch_pruning package force more filters to be pruned from the model to ensure number of filters are the same for the no-parameter skip connections. This makes the actually pruned models have inferior performances compare to the PyTorch pruned models. Unfortunately, we don't have time to finetune and evaluate the actually pruned models.

However, we are able to collect inference times of the actually pruned models. (Fig. 10 & Fig. 11) Interestingly, while PyTorch report 53.13% of parameters remained for both ResNet18 and ResNet50, the actual percentage of parameters for ResNet18 is 6.27%, and for ResNet50, that value is 17.21%. Despite having a larger fraction of parameters left, the inference time for ResNet50 improves by a larger fraction.

F. Unstructured comparison

Figure 12 compares the test accuracy of different CV models when pruned using the same setting. From the plot, it looks like VGG16 outperforms the other models when more parameters are pruned, but this could be due to the extremely large classifier the VGG16 has. The performance degrade at a similar rate for the other three models.

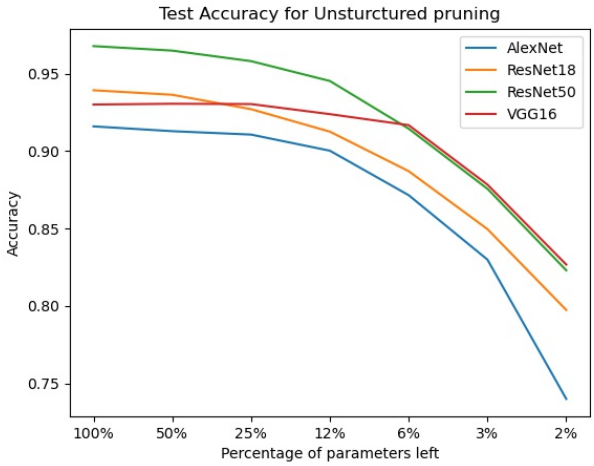

Fig. 12. Test Accuracy after unstructured pruning

VII. Related work

In Ref [4], Han et al. introduce the iterative pruning method and pruning with L1 or L2 regularization. This is a good starting point for understanding how pruning works. This article [5] is also a useful source to get an overview of pruning. While in Ref [6], Blalock et al. summarize the recent pruning methods and compare their performance.

VIII. Conclusion

Overall, the results are promising, especially on relatively easy tasks. The models we tested maintain a similar performance even with only a small fraction of parameters. And with structured pruning, it is plausible to actually pruned the model using current tools and observe improvements in inference times. However, whether the improvement in inference time worth the decrease in performance need to be discussed case by case. Also, the overall support for pruning is still not sufficient in popular frameworks such as TensorFlow and Pytorch.

IX. Future Work

Due to time restrictions, we are unable to do some the works below.

- It would be great to actually remove the masked connections in the unstructually pruned models and experiment the inference time on CPU and GPU. Theoretically, the inference time should improve on CPU. While for GPU, the inference time shouldn't improve due to the lack of parallel computation in sparse matrix computations.
- Experiment the difference in performance of the actually pruned ResNet and Pytorch pruned ResNet. Due to the skip connections in ResNet, when pruning a convolutional layer, the torch_pruning package will prune some filters

from the previous convolutional layers. As a comparison, with Pytorch's mask implementation, those filters are kept and passed into the next ResNet block.

- Structured pruning of Linear layers and more experiment with BERT
- The heuristics we mentioned in the paper are only a fraction of the heuristics available for pruning. We are interested in testing other heuristics such as using the variances of the neurons' activation values for the training set to determine which neurons to prune.

REFERENCES

[1] Huggingface, "huggingface/transformers." [Online]. Available: https://github.com/huggingface/transformers

[2] VainF, "Vainf/torch-pruning." [Online]. Available: https://github.com/VainF/Torch-Pruning

[3] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[4] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[5] R. Bandaru, "Pruning neural networks," Sep 2020. [Online]. Available: https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9#:~:text=Neuralnetworkpruningisa,removingunnecessaryneuronsorweights.

[6] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *arXiv preprint arXiv:2003.03033*, 2020.