10-605 Mini Project Pruning

Balasubramanyam Evani
bevani@andrew.cmu.edu

I. Methodology

The gist of this mini project is to implement pruning methodologies for large neural networks since their size consumes a considerable amount of computational resources. Especially for embedded systems, these resource demands become more prohibitive.

I have used the following three methods for this mini-project to prune the provided CNN model.

- Magnitude-based pruning
- L1-norm-based filter pruning
- ThiNet

A. Magnitude-based pruning [1]

Magnitude-based pruning methodology employs three steps. The first step is to learn the entire network. The second step is to prune the low-weight connections. This removal step is based on a threshold value. For my implementation, I have experimented with choosing a global threshold and choosing a layer-wise threshold. The threshold value in each of the cases is calculated as follows:

Algorithm 1 Magnitude Pruning (Global threshold)

```
Function CalcStd(weights):
  • Create an empty list and append each weight flattened to it
  • Stack the flattened weights horizontally into a 1-D array
  • Calculate and return the standard deviation of the 1-D array
Function Prune(weights, stdval, factor)
  • Create a deep copy of the weights array
  • Calculate the threshold as the product of stdval and factor
  • For each weight in the copied array:
      – Create a binary mask array based on the threshold
      – Apply the mask array to the weight by element-wise multiplication
  • Return the pruned weights array.
```

To be brief, the weights are zeroed using the binary mask if the absolute value of the weight value is not within a certain amount of times the standard deviation value. The intuition here is to remove any weights which doesnt seem much important on a global/layer-wise level. **In the case of layer-wise thresholding, the standard deviation is calculated layer-wise and is not a single global value like in Global thresholding**.

B. L1 norm-based pruning [2]

The paper's authors primarily used this technique to prune out whole filters in the network together with their connecting feature maps. This pruning is based on the L1 norm calculation of these filters. **Since this technique can be applied only to convolutional filters, for fully connected layers I have employed magnitude-based pruning** as described in the earlier subsection. The pseudo-code for L1 norm-based pruning applied to convolutional layers is shown below

Algorithm 2 L1 norm Pruning (Conv Layer)

```
Function Prune(model, factor)
  • Initialize an empty list to store pruned weights
  • For each layer in the model:
      – If the layer is a convolutional layer:
          ∗ Get the layer's weights and biases
          ∗ Create a deep copy of the weights and biases
          ∗ Calculate the L1-norms of the weights over the spatial dimensions
          ∗ Sort the L1-norms in descending order and get the sorted indices
          ∗ Calculate the cutoff index based on the factor
          ∗ Set the weights and biases of the bottom cutoff filters to zero
          ∗ Append the pruned weights and biases to the list
  • Return the list of pruned weights
```

The **cutoff** is calculated using a factor that represents the percentage of filter that needs to be pruned/removed. The intuition behind this method is that if a filterś L1 norm (at a specific layer i) is low then itś highly possible that it would not contribute much to the subsequent layers.

C. ThiNet [3]

This paper introduces a new methodology for pruning convolutional filter connections. It uses layer i+1 to guide the pruning in layer i, where a subset of channels in layer i+1 input is used to approximate the output in layer i+1. The method is specific to convolutional filters and differs from the previous method that uses layer i statistics to guide the pruning of layer i filters.

The pseudo-code for the greedy selection criteria is given below. This is called for all layers in my implementation (including Dense).

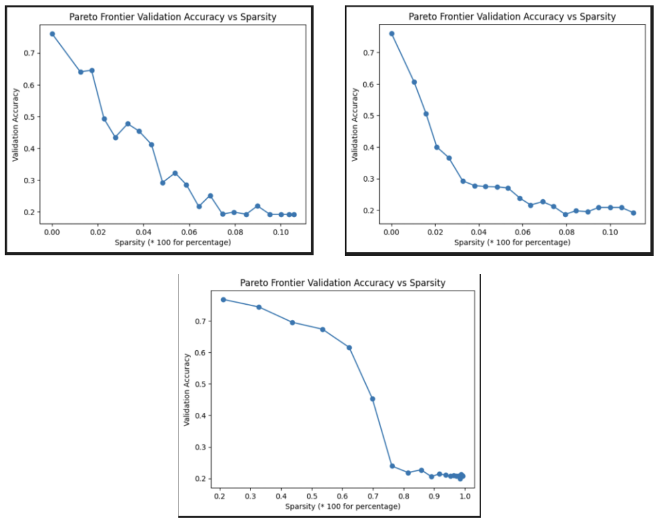
II. Comparison

Few Notes: All methods have been subjected to prune() and retrain() procedures for multiple iterations. Though **The**

```
Function ComputeValue(inputs, tmpT):
  • Select the inputs tensor along the channel dimension indexed by tmpT
  • Calculate the element-wise square of the sum of the selected tensor over the last dimension
  • Calculate the sum of the resulting tensor
  • Return the sum as the objective value
Function GreedyChannelSelection(inputs, C, sparsity):
  • Initialize an empty list to store the indices of pruned channels
  • Calculate the number of channels to prune based on the C and sparsity
  • While the number of pruned channels is less than the calculated number:
      – Initialize the minimum objective value and its corresponding index
      – For each unpruned channel:
          ∗ If the channel is already pruned, skip it
          ∗ Select the channel and the already pruned channels as the temporary channel indices
          ∗ Calculate the objective value of the temporary channel indices using ComputeValue
          ∗ If the objective value is smaller than the current minimum, update the minimum and its index
      – Append the index of the channel with the minimum objective value to the list of pruned indices
  • Compute the indices of the stayed channels as the complement of the pruned indices
  • Return the lists of pruned and stayed indices
```

magnitude-based pruning method is subjected to a masked gradient update step instead. Also, for the magnitude pruning-based method on top of the algorithm described in the earlier section, label smoothing and data augmentation only on train data has been applied. For the other two methods, no such data augmentation was applied and the vanilla loss function was used while retraining. This was done since with method 1 I was able to achieve a good score and I implemented the other two for comparison tasks.

*1) **Provide a plot exploring the methods in terms of the Pareto frontier of sparsity vs accuracy:*** The Pareto plots for all the methods have been shown below. The x-axis here is the amount of sparsity added and the y-axis is the validation accuracy achieved. To be fair in comparison, sparsity has been added to the conv filter pruning methods at only conv layers (no sparsity has been added to Dense layers) and also all graphs shown are results plotted without retraining (i.e I take the unpruned weights and apply individual pruning method and plot the graph. Also I didnt use data augmentation and label smoothing methods for magnitude pruning as well for this graph.

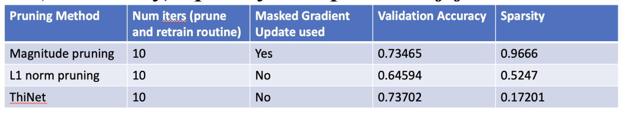


Top left: ThiNet, Top right: L1 norm
Bottom: Magnitude

Based on the Pareto chart we can see that conv filter pruning methods have sharper slopes as compared to the magnitude pruning method. My reasoning is that the earlier conv layers are very essential for the classification/feature extraction tasks and hence a small amount of pruning can lead to a large drop in accuracy. ThiNet seems to do a better job than the L1 norm where Magnitude performs best (almost 2/3rd weights can be removed and we'd still end with reasonable acc.)

*2) **Which was easiest to implement:*** The L1 norm and magnitude-based pruning methods were the easiest to implement. The difficulty with ThiNet was passing the input layer by layer and calculating the channels greedily which might not be too intuitive to understand. Runtime wise ThiNet runs slower than the other two methods.

*3) **Accuracy, Sparsity Comparison of final three models:***

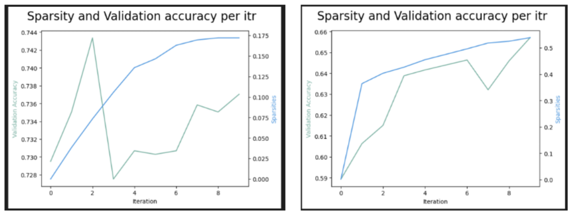


Pruning Method	Num iters (prune and retrain routine)	Masked Gradient Update used	Validation Accuracy	Sparsity
Magnitude pruning	10	Yes	0.73465	0.9666
L1 norm pruning	10	No	0.64594	0.5247
ThiNet	10	No	0.73702	0.17201

*4) **which method led to the best overall model score:*** The magnitude pruning-based method (which was easier to implement) along with data augmentation and label smoothing performed the best out of the three methods. The masked gradient update helped with maintaining the sparsity across iterations. But, I do suspect that the other two methods which were not subjected to masked gradient update can also match magnitude pruning performance when run for more iterations. Unfortunately, I didn't have the bandwidth to run more iterations since it wasn't the main scope of this project.

III. Reflection

*1) **Did the results match what you expected:*** In total I had tried three methods and two gradient update techniques (one where I used masked gradient update for magnitude pruning and in other two cases I let the prune and retrain routine update the gradients). The results did match some of my expectations. Firstly, with the masked gradient update the sparsity didn't decrease after the first retrain and it only increased. Secondly, in the other methods where I didn't implement the masked gradient update, the prune and retrain routine itself made sure that only strong weight connections remained, only for the first retrain the sparsity dropped but then I observed a generally increasing trend as well (this is shown in the figure below), which was unexpected and interesting for me.



Left: ThiNet, Right: L1 norm

As far as validation accuracy goes, I did expect my first method to perform better since I made sure to not overfit the data by applying label smoothing and data augmentation. But the other two did perform well despite not including label smoothing and data augmentation. Especially for ThiNet where I added a very small sparsity percentage for the Dense layers as compared to the L1 norm method. All in all, I think all three methods could have reached magnitude prunings performance but due to computing constraints and being an only member team, I didn't have the bandwidth.

*2) **What did you learn about model pruning as part of this project**:* The most amazing part of this project was the realization that I could almost prune out 97% of the parameters and still make the network perform really well (on par with the original model). I also learned the various techniques currently being used for pruning CNNs and how one differs from the other in terms of finding out which weight connections are irrelevant. Finally, I also learned that in production may be a combination of the earlier methods might make prove to give better performance. As mentioned earlier despite not explicitly masking the dense gradients the overall routine did ensure increased sparsity while maintaining a reasonable accuracy which I didn't expect initially but understood later. One more interesting thing I found was that using a layer-wise threshold in magnitude pruning didnt́ help me get good results through retraining and neither it helped when I applied conjecture to L1 pruning. This was also something I didnt expect but if I had to reason as to why this happened, Iď probably guess that the layer-wise thresholding doesnt́ take into account some form of weights correlation.

References

[1] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *CoRR*, vol. abs/1506.02626, 2015. arXiv: 1506.02626. [Online]. Available: http://arxiv.org/abs/1506.02626.

[2] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016. arXiv: 1608.08710. [Online]. Available: http://arxiv.org/abs/1608.08710.

[3] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *CoRR*, vol. abs/1707.06342, 2017. arXiv: 1707.06342. [Online]. Available: http://arxiv.org/abs/1707.06342.