**PRUNING CONVOLUTIONAL NEURAL NETWORKS FOR RESOURCE EFFICIENT INFERENCE**

Gist

1. Oracle/Brute-force pruning (Remove a feature map/channel, get the change in the training losses) is the best, but slow.
2. Absolute oracle criterion does not reduce accuracy as much as signed oracle criterion. More stable as the loss shouldn’t decrease OR increase too much by pruning.
3. Use Spearman coefficient to decide a good substitute to oracle criterion.
4. Lecun’s OBD and Taylor expansion performs the best. But OBD requires Hessian (second order derivative) which is expensive to compute.
5. In Taylor expansion, only use the first order expansion. Remainder requires Lagrangian hence it is omitted due to high computation. Substitute first order expansion into absolute loss results in absolute product of loss gradient w.r.t activation and activation itself.
6. Average Taylor criterion over pixels/neurons and also minibatch examples.
7. Scale of activation values varies according to network depth, normalize layer using L2 norm.
8. Use FLOPs regularization to reduce compute time. Prune layers with high FLOPs first.
9. The more training updates are done between pruning iterations, the higher the accuracy.
10. Additional fine-tuning at the end of pruning significantly restores lost accuracy.
11. Generally, shallower layers are more important than deeper layers.

Baseline: U-Net on Carvana Dataset

Training size: 4071 (images are split into left & right halves)

Validation size: 1018

Batch size: 2

Image scale: 0.5

GPU Memory: 8GB

Training Loss (BCE): 0.02259

Validation Metric (Dice Coeff): 0.9853

Ideas

Since model only has Conv-BN-ReLU modules, only remove previous convs’ filters & biases, corresponding BN parameters, and next conv’s channels (but not the bias).

Cannot remove maxpool or bilinear upsampling since these are needed for output shape consistency and they are parameter-less.

Also, cannot remove final conv layer since it is used to generate mask and it only has 64 parameters.

**Prune class**

Initialization

- Initialise with network to be pruned.

- Need containers for Taylor ranks, activations, gradients.

- Also store conv2d layers and corresponding BN layers

- Register forward and backward hooks on construction.

- Hooks add tensors to containers. Tensors must be detached from computation graph, else they would not be deallocated upon exit.

Ranking

- After each mini-batch, compute Taylor ranks by averaging (activation \* gradient) over the spatial dimensions. Absolute the average and then average across examples in the minibatch. Accumulate into ranks container.

- After sufficient number of mini-batch iterations, rank the channels by first dividing the accumulated ranks by the number of iterations. Perform layer-wise L2-normalization and store normalized ranks in a hierarchy-less container. Also keep track of the layer index of each channel and the channel index wrt to the layer of each channel.

- Get the top-K channels to be removed next.

Pruning

- Initialize a list of indices for each channel of each layer. Do this for input and output channels.

- Remove each of the top-K indices out of the list.

- Remove the output channel index of the current layer, and the input channel index of the next layer.

- Certain layers have residual connection to a later layer, so the index to that input channel of that later layer should be removed as well.

- Certain layers have to be concatenated with an earlier layer’s output, with the earlier tensor at the top and the current tensor at the bottom. An offset equivalent to the earlier layer’s number of output channels should be applied to the index.

- Using the updated list, index each layer’s parameters. This includes BN and Conv2D layers.

Pruning Example:

- To prune 5th conv2d’s output tensor’s 62nd channel, we remove 5th conv2d’s 62nd output filter & bias and 5th batchnorm’s 62nd elements from its weight, bias, running mean and variance. Also remove 6th conv2d’s 62nd input channels for all its output filters.

- If 6th conv2d receives a concatenated input of 3rd and 5th conv2d, then we should remove 6th conv2d’s *(c + 62)*th, where c is the number of output channels for 3rd conv2d, input channels for all its output filters.

- If 5th conv2d’s output also input to 8th conv2d by concatenating with another layer’s output, then remove the 62nd input channels of 8th conv2d’s filters.

Pruning Pitfalls:

- Cannot remove channels sequentially. An index *j* may not be pointing to the *jth*channel due to removal of other channels in the same layer.

- Eg. we removed 7th channel in 64-channel 2nd layer. Now 63 channels in 2nd layer. Then remove 22nd channel in 2nd layer, but 22nd channel is actually 23rd channel.

- Unless indices are updated each time, wrong parameters are removed.

Saving

- Save the pruned parameters back into the network. Save the network’s *state\_dict* and the pruned number of channels of each layer.

- Could not reuse the same network with updated parameters, as *nn.Module*s still expected the old shapes (checked by printing the network).

- Destroy the network, recreate with the reduced number of channels and load the weights.

**Experiment**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Ranking iterations | Pruned channels | Finetuning | | Val. DICE | File Size (MB) |
| Epochs | Iterations |
| 1 | 500 | 300 | 0 | 1500 | 0.948 | 44.4 |
| 2 | 500 | 300 | 0 | 1500 | 0.861 | 38.9 |
| 3 | 500 | 300 | 0 | 1500 | 0.933 | 33.2 |
| 4 | 500 | 300 | 5 | 0 | 0.955 | 27.2 |

Size Reduction: (52.4 – 27.2) / 52.4 x 100% = 48.1%

Validation DICE Loss: 98.53% – 95.5% = 3.03%