香港中文大學
The Chinese University of Hong Kong

CMSC5743

Lab05 Introduction to Distiller

Qi Sun

(Latest update: October 13, 2020)

Fall 2020

Distiller

- Distillerm is an open-source Python package (PyTorch environment) for neural network compression research.
- Comprehensive documentation and a mature forum.
- Example implementations of state-of-the-art compression algorithms.
- A friendly framework that you can add your own pruning, regularization and quantization algorithms easily.
- Supports of lots of mainstream DNN models and datasets, *e.g.*, SqueezeNet and ImageNet.

Using The Sample Application

An example python file is provided:

./examples/classifier_compression/compress_classifier.py

- Check all of the program options via `python ./compress_classifier.py -h`, including the pretrained models.
- You can try the Jupyter notebook to learn the usage of Distiller.
- Specify the algorithm configurations in a YAML file.

```yaml
version: 1
pruners:
  my_pruner:
    class: 'SensitivityPruner'
    sensitivities:
      'features.module.0.weight': 0.25
      'features.module.3.weight': 0.35
      'classifier.1.weight': 0.875
```

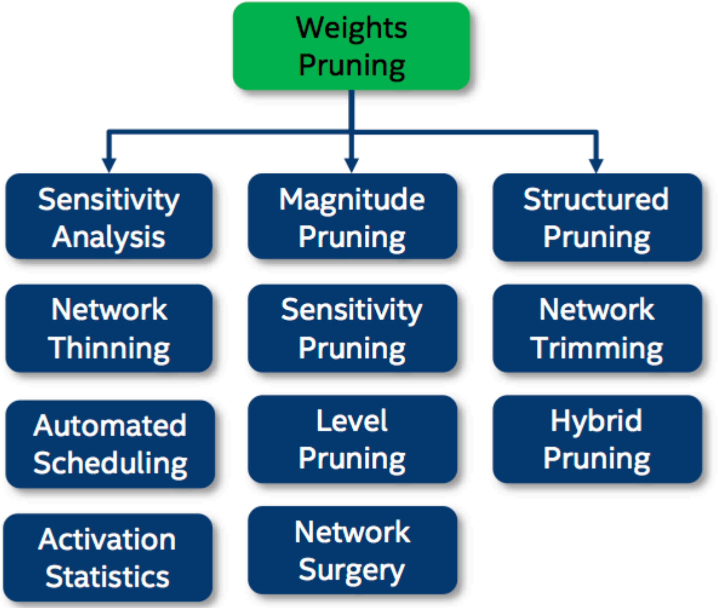
Pruning Sensitivity Analysis

Command flag

`--sense = element or filter`

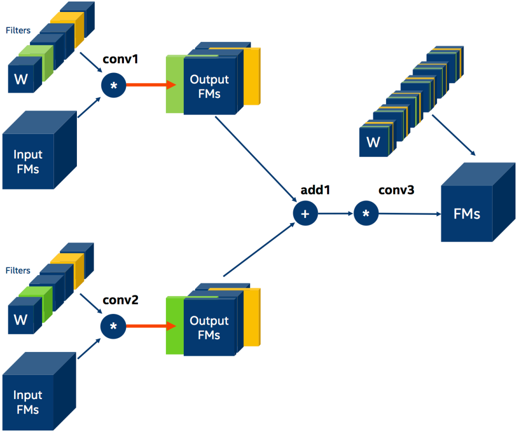
- Distiller supports element-wise and filter-wise pruning sensitivity analysis.
- In both cases, L1-norm is used to rank which elements or filters to prune.
- For example, when running filter-pruning sensitivity analysis, the L1-norm of the filters of each layer's weights tensor are calculated, and the bottom $x\%$ are set to zero.
- Use a small dataset for this would save much time, if this will provide sufficient results.

Pruning Algorithms



Pruning Algorithms

- All of the pruning algorithms are defined in `./distiller/pruning`.
- Channel and filter pruning.
- Pay attention to the model structure to guarantee the pruning strategies are mutually compatible.



Magnitude Pruner

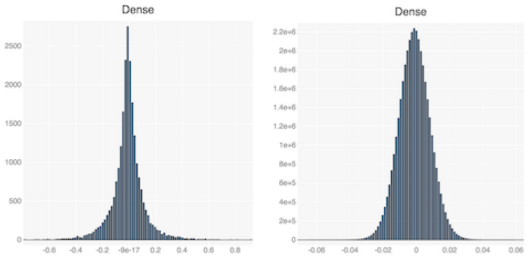
- It applies a thresholding function, $thresh(\cdot)$, on each element, w_i, of a weights tensor.
- Because the threshold is applied on individual elements, this pruner belongs to the element-wise pruning algorithm family.

$$thresh(w_i) = \begin{cases} w_i : \text{if} |w_i| > \lambda \\ 0 : \text{if} |w_i| \leq \lambda \end{cases} \tag{1}$$

Sensitivity Pruner

- The model weights approximately follow the Gaussian distributions, with standard deviation σ and mean value μ.
- $3-\sigma$ rule: $68-95-99.7$ rule.

$$\Pr(\mu-\sigma \leq X \leq \mu+\sigma) \approx 0.6827 \tag{2}$$



- If we set the threshold to $s \times \sigma$, then basically we are thresholding $s \times 68\%$ of the tensor elements.

Automated Gradual Pruner (AGP)

- The sparsity is increased from an initial sparsity value s_i (usually 0) to a final sparsity value sf over a span of n pruning steps.
- The intuition behind this sparsity function is to prune the network rapidly in the initial phase when the redundant connections are abundant and gradually reduce the number of weights being pruned each time as there are fewer and fewer weights remaining in the network.

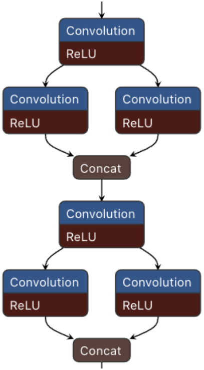
$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad \text{for} \quad t \in \{t_0,\ t_0 + \Delta t,\ \ldots,\ t_0 + n\Delta t\}$$

Post-training Quantization

- It does not require any Policies nor a Scheduler.
- A checkpoint with the quantized model will be dumped in the run directory.
- It will contain the quantized model parameters (the data type will still be FP32, but the values will be integers).
- The calculated quantization parameters (scale and zero-point) are stored as well in each quantized layer.

Check Model Parameters

- Use **Netron**. If a prototxt file is available, you can visualize the model.
- Use `model['state_dict'].items()`.
- Use `named_parameters()`.



Experiment Reproducibility

To guarantee the reproducibility of your results.

- Set $j = 1$ to use only one data loading worker.
- Use `--deterministic` flag.