深度學習系統與實現

Lab 4 – Model compression and deployment (I)

Dept. of Computer Science and
Information Engineering

National Chiao Tung University

Model compression and deployment (I)

- In lab 4, we will focus on how to get performance improvement in the inference phase

- There are two topics we will discuss in this lab:
  - Model compression
    - Pruning
    - Quantization
  - Model deployment for CPU
    - Intel OpenVINO toolkit

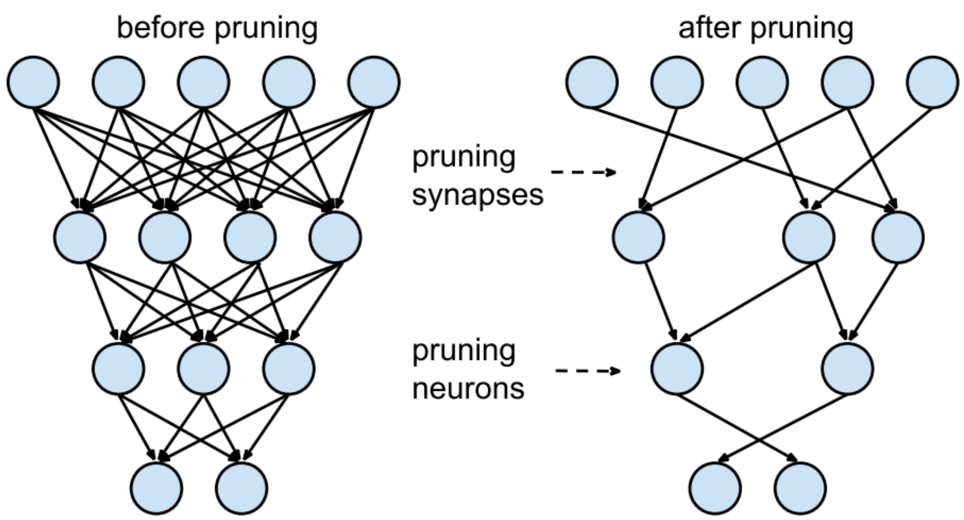
Outline

- Concepts overview
  - Pruning
  - PyTorch pruning API
  - Introduction of OpenVINO
  - Quantization
  - OpenVINO: Accuracy Checker Tool
  - OpenVINO: Post-Training Optimization Tool
- Lab 4 specification
- Questions
- Grading
- Notices & Hints

Pruning

- Network pruning is widely used for reducing the heavy inference cost of deep models in low-resource settings

- According to the "lottery ticket" hypothesis, only partial subnetworks affect overall accuracy critically
- Thus, we can try to uncover these subnetworks, and prune other nodes or edges to get a better performance



Reference:
Jonathan Frankle, Michael Carbin (ICLR 2019). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks
Song Han, Jeff Pool, John Tran, William J. Dally (NIPS 2015). Learning both Weights and Connections for Efficient Neural Networks

PyTorch Pruning API

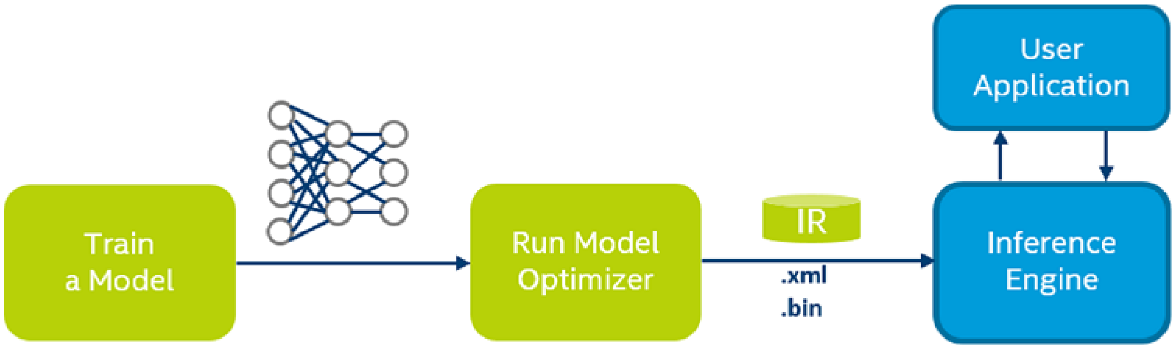
- PyTorch release nn.utils.prune module in version 1.4, you can apply a mask on your model parameters through this module, and get a sparse network finally
- Please take a look at the following webpages to learn how to use this module
  - [PyTorch pruning tutorial] https://pytorch.org/tutorials/intermediate/pruning_tutorial.html
  - [PyTorch pruning API] https://pytorch.org/docs/stable/nn.html#basepruningmethod

Reference:
[PyTorch pruning tutorial] https://pytorch.org/tutorials/intermediate/pruning_tutorial.html
[PyTorch pruning API] https://pytorch.org/docs/stable/nn.html#basepruningmethod
[PyTorch 1.4 introduction] https://blog.exxactcorp.com/pytorch-release-v1-4-0-mobile-build-customization-distributed-model-parallel-training-java-bindings/

PyTorch

Introduction of OpenVINO

- A toolkit provided by Intel to facilitate faster inference of deep learning models on Intel’s hardware
- Deployment working flow:
  - Compile model to Intermediate Representation (IR)
  - Perform inference with Inference Engine

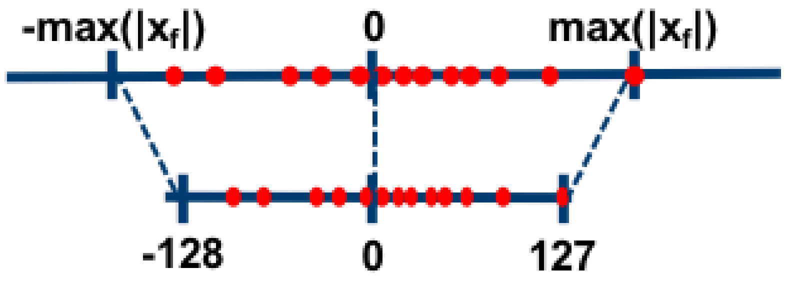


Quantization

- Quantization refers to the process of reducing the number of bits that represent a number

- Normally, data type of model parameter is FP32, which requires many floating-point operations in the inference phase

- However, if we perform quantization and cast some nodes to int8, then we can get performance improvement from replacing floating-point operations with integer operations



e.g. Symmetric Quantization

Reference:
https://docs.openvinotoolkit.org/latest/_compression_algorithms_quantization_README.html
https://nervanasystems.github.io/distiller/algo_quantization.html

OpenVINO: Accuracy Checker Tool

- A general accuracy checker tool in OpenVINO
- We have to provide a YAML config file to define our model IR path, preprocess, postprocess, annotation converter, metrics, etc.
- Then type “accuracy_check -c yourConfig.yml” to run the tool

OpenVINO: Post-Training Optimization Tool

- A quantization tool for OpenVINO model IR provided by Intel
- There are several quantization algorithms implemented in this tool:
  - DefaultQuantization:
    - Used as a default method to get fast but in most cases accurate results for 8-bits quantization
  - AccuracyAwareQuantization:
    - Allows staying at the pre-defined range of accuracy drop after the quantization while sacrificing performance improvement

Specification

Dataset - food11

- Food11 Download link - https://www.kaggle.com/tohidul/food11
- Note: in lab4, full or skewed food11 are both ok

LAB 4 (a) Pruning with PyTorch

- [20%] Element-wise pruning
  - Require >= 50% global sparsity with less than 10% accuracy drop

- [20%] Channel-wise pruning
  - Try your best to get the highest global sparsity with less than 20% accuracy drop

- Please show **global sparsity**, **sparsity of each layer**, and **accuracy** in your experiment

- Sample output
- You can have your own output format, but just keep in mind to show **global sparsity**, **sparsity of each layer**, and **accuracy** respectively

```
----------------------------------------------------------------
Model Evaluation:  Before pruning
----------------------------------------------------------------
Test set: Top 1 Accuracy: 3029/3347 (90%)
Class 0 :  311/368    84.51%
Class 1 :  131/148    88.51%
Class 2 :  431/500    86.20%
Class 3 :  292/335    87.16%
Class 4 :  259/287    90.24%
Class 5 :  384/432    88.89%
Class 6 :  143/147    97.28%
Class 7 :  95 /96     98.96%
Class 8 :  278/303    91.75%
Class 9 :  483/500    96.60%
Class 10:  222/231    96.10%

----------------------------------------------------------------
Model Evaluation:  After pruning
----------------------------------------------------------------
Test set: Top 1 Accuracy: 2798/3347 (84%)
Class 0 :  250/368    67.93%
Class 1 :  125/148    84.46%
Class 2 :  404/500    80.80%
Class 3 :  269/335    80.30%
Class 4 :  214/287    74.56%
Class 5 :  341/432    78.94%
Class 6 :  143/147    97.28%
Class 7 :  96 /96     100.00%
Class 8 :  286/303    94.39%
Class 9 :  478/500    95.60%
Class 10:  192/231    83.12%

----------------------------------------------------------------
Parameters                     Zeros      Total      Sparsity
----------------------------------------------------------------
conv1.weight                   1882       9408       20.00%
...                            ...        ...        ...
fc.bias                        0          11         0.00%
================================================================
Global sparsity:                                     52.94%
```

LAB 4 (b) OpenVINO deployment

- [10%] Export your PyTorch model to ONNX, and compile it to IR with OpenVINO Model Optimizer

- [20%] Perform inference with OpenVINO inference engine, and show **accuracy**, **latency**, and **throughput** of your model in PyTorch and OpenVINO respectively

- *Requirement*
  - *You have to use CPU for inference*
  - *You have to provide your hardware information in your report (e.g. CPU, memory)*
  - *Any accuracy drop in OpenVINO compared with PyTorch is not accepted in this part, or you will get a little penalty in your score*

- Sample output

- You can only show **accuracy**, **latency**, and **throughput (FPS)** in your experiment, but it's recommended to show more information in your experiment

```
[ INFO ] Loading network files:
        resnet_food11/resnet.xml
        resnet_food11/resnet.bin
[ INFO ] Accuracy: 90.4990%  3029/3347
[ INFO ] Startup loading time = 210.94 ms
[ INFO ] Average image preprocessing and loading time = 6.46 ms
[ INFO ] Average inference time = 7.79 ms
[ INFO ] Average latency = 14.40 ms
[ INFO ] Total execution time = 48.18 s
[ INFO ] FPS = 69.46
[ INFO ] FPS(without startup time) = 69.77
```

- *Hint: you can set a timer in your inference script to get latency and FPS*

LAB 4 (c) Accuracy Checker Tool

- [10%] Generate annotation .pickle file of Food-11 evaluation dataset through OpenVINO Accuracy Checker Tool, and show output **accuracy** from this tool

- The annotation .pickle file will be used in Post-Training Optimization Tool later

Sample dataset config

- *If you want to use the custom food11 annotation converter provided by TA, please follow this format to fill dataset part in your config*
- *data_source should be equal to data_dir, which means the path to your food11 dataset*

```
datasets:
  - name: food11_dataset
    data_source: /train-data/food11re/evaluation
    annotation_conversion:
      converter: food11
      data_dir: /train-data/food11re/evaluation
      labels_file: ../label_map_food11.txt
    annotation: food11_eva_annotation.pickle
```

- *Requirement*
  - *Any accuracy drop is not accepted in this part, or you will get a little penalty in your score*

- Sample output

- You can find the **accuracy** calculated by the tool here

```
Processing info:
    model: Resnet_food11_test
    launcher: dlsdk
    device: CPU
    dataset: food11_dataset
    OpenCV version: 4.3.0-openvino
    IE version: 2.1.42025
    Loaded CPU plugin version:
        CPU - MKLDNNPlugin: 2.1.42025
    Input info:
            Layer name: input
            precision: FP32
            shape [1, 3, 224, 224]

    Output info
            Layer name: output
            precision: FP32
            shape: [1, 11]

     100%|#######################| 3347/3347
    3347 objects processed in 53.085 seconds
    accuracy: 90.50%
```

LAB 4 (d) Post-Training Optimization Tool

- [10%] DefaultQuantization
  - Try to apply DefaultQuantization on your model IR generated in part b
  - Show benchmark of the quantized model IR (**accuracy**, **latency**, and **throughput**)

- [10%] AccuracyAwareQuantization
  - Check the accuracy drop after perform DefaultQuantization, then set a smaller drop range using AccuracyAwareQuantization
  - Show benchmark of the quantized model IR (**accuracy**, **latency**, and **throughput**)

Questions

- Do you get any performance improvement in part (a) ? Why ?
- Why can we get speedup using OpenVINO ?
- Please briefly explain how DefaultQuantization and AccuracyAwareQuantization works

Grading

- (a) Pruning with PyTorch (40%)
- (b) OpenVINO deployment (30%)
- (c) Accuracy Checker Tool (10%)
- (d) Post-Training Optimization Tool (20%)
- Bonus (max 10%)
  - Any extra work, we will give bonus base on your effort (e.g. Try to use other techniques provided by OpenVINO to improve your FPS, try to use other pruning and quantization tool, etc)

Total
110

- Submission: source code + report (.ipynb is accepted) [e3]
  - zip format (ex: DLSR_lab04_{group id}.zip)
  - 5% penalty for the wrong submission format
- Deadline : 2020/05/11, 23:59 (Mon) [2 week]
- Demo: (TAs will announce on New-E3 later)

Notices & Hints

- (a) Pruning with PyTorch
  - Remember to remove pruning re-parametrization after pruning
- (b) OpenVINO deployment
  - You can take a look at this official sample, and we recommend to use Python API of OpenVINO inference engine
  - https://github.com/openvinotoolkit/openvino/tree/2020/inference-engine/ie_bridges/python/sample/classification_sample
  - We recommend to use ResNet model, or you have to check op supported by OpenVINO before deployment
  - Keep your input transform as simple as possible
  - Behavior of OpenCV and Pillow is different, please check how PyTorch deal with your input data carefully
- (c) and (d)
  - Take a look at the following path in the Docker container, you can find some sample configs in these folder
  - /opt/intel/openvino/deployment_tools/tools/post_training_optimization_toolkit/configs/examples/quantization/classification/
  - /opt/intel/openvino/deployment_tools/tools/post_training_optimization_toolkit/configs/
  - /opt/intel/openvino/deployment_tools/open_model_zoo/tools/accuracy_checker/configs/

OpenVINO environment provided by TA

- We have built a Docker image for OpenVINO, so you can simply pull it from Docker Hub in your Linux system

- chilinhs/openvino_dlsr_lab4:2020.2-py3.6
  - https://hub.docker.com/r/chilinhs/openvino_dlsr_lab4/tags

- Docker image contents:
  - OpenVINO 2020.2 installed
  - Accuracy Checker Tool installed
  - Post-Training Optimization Tool installed
  - Custom Food-11 dataset annotation converter installed

Reference

- PyTorch Pruning tutorial
  - https://pytorch.org/tutorials/intermediate/pruning_tutorial.html
- Export your PyTorch model to ONNX
  - https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html
- OpenVINO Model Optimizer
  - https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_convert_model_Converting_Model.html
- OpenVINO Inference Engine
  - https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Integrate_with_customer_application_new_API.html
- OpenVINO Accuracy Checker Tool
  - https://docs.openvinotoolkit.org/latest/_tools_accuracy_checker_README.html
- OpenVINO Post-Training Optimization Tool
  - https://docs.openvinotoolkit.org/latest/_README.html