HT2023: IL2230 HADL Lab 3B
Transfer Learning, Network Pruning, and Quantization

Zhonghai Lu

November 22, 2023

1 Introduction

1.1 Transfer learning

Training a neural network consumes significant time, and is difficult to optimize, even for a reasonably sized learning problem, because the design space for neural network structure or topology (hyper-parameters) and network weights/biases for a particular selection of neural network are huge. Fortunately, effective learning does not have to start from scratch. Instead, transfer learning can largely mitigate this problem.

The basic idea of transfer learning is to re-use part of an already well-trained neural network and its parameters as a starting point for neural network design and training. This is feasible because many learning problems consist in extracting features from low to high levels in order to find an effective mapping from input to the eventual output. Transfer learning provides an effective starting point by reusing an existing successful model, still it needs to be adapted for the specific problem in hand and to be re-trained to solve the specific problem.

1.2 Network pruning

Neural networks often suffer from high complexity (e.g. a huge number of weights and biases). However, they contain very often large redundancy. Many connections within the neural networks can be pruned away to simplify the networks, reducing the size and thus storage requirement of the networks and further reducing the amount of computation. However, a naive pruning would result in significant accuracy loss. To help recover accuracy, network re-training can nevertheless be used. In fact, iterative pruning and retraining is an essential step when applying network pruning for network simplification.

1.3 Data quantization

Quantization is a common technique to speed up the execution of neural networks. This is especially important for the edge device when deploying machine learning applications in resource-constrained environments. Quantization often implies processing with small bit-width low precision data, usually Int8 (8-bit signed integer) rather than Float32 (32-bit floating point number). This enables performance gains in model size, memory bandwidth, and inference time consumption.

2 Ojectives

The purpose of the lab is to practice transfer learning, network pruning, and quantization. It has a few intended learning outcomes after completing the lab:

- be able to apply transfer learning for a specific learning problem
- be able to prune a neural network
- be able to recover accuracy through re-training after network pruning
- understand different quantization methods and be able to do data quantization on a trained network

The lab shall be carried out using PyTorch. It is a team work with typically 3 to 4 students in one group.

3 Tasks

The lab intends to solve an image classification problem. The data set is the hymenoptera dataset (a subset of the ImageNet dataset), which consists of only 2 classes. It is well known that image classification can be well addressed by convolutional neural network (CNN) which can automatically extract features in a way much more efficient and effective than manually-made hand-crafted features. After feature extraction by convolutional layers, the neural network can perform classification using fully connected layers.

3.1 Learning from scratch

Before going for transfer learning, you will design a CNN for the image classification problem from scratch. In this step, you have the full freedom to design your network structure (number of convolutional layers, pooling layers, fully connected layers and nuerons in each layer). Then you train your CNN and evaluate for its accuracy.

Your tasks are as follows:

1. Initial CNN construction: Design your own CNN to classify the hymenoptera dataset. (Download the data https://download.pytorch.org/tutorial/hymenoptera_data.zip, put it into dataset directory and unzip it.)
2. Evaluate the accuracy of your CNN.

Questions: Can your CNN achieve satisfactory performance, say above 90% of accuracy? Analyse why or why not?

3.2 Transfer learning

After the first step of learning from scratch, now we move towards transfer learning. We have touched upon a few famous well-established CNNs, for example, AlexNet, ResNet etc. These networks are designed to do classification for the ImageNet dataset which has about 1.5 million images with 1000 classes.

Our dataset for this experiment is Hymenoptera, which is a subset of the ImageNet dataset. The hymenopetra has two classes: Ant and Bee. It has 245 images for training and

153 images for inferring. Although the number of the images is much smaller, the images are equally complex and hard to recognize. This means the task of feature extraction is equally challenging. Therefore we decide to use transfer learning by adapting AlexNet or ResNet for this task.

Your tasks are as follows:

1. Model adaptation: Obtain the AlexNet or ResNet from web (load the models from torch with pretrained=True), and then choose one of them, preferably AlexNet, to classify the hymenopera data set. Call this adapted CNN as CNNAdapted.

2. After adaption, record the classification accuracy of your adapted CNN without retraining the whole model, but only the output layer.

3. Model optimization: Retrain your adapted CNN to obtain an optimized CNN for the 2-class classification problem, and then perform inference and record the classification accuracy of your optimized CNN. Call this model CNNOptimized.

Suggestion and hint:

- To realize this task, go through the transfer learning tutorial on the PyTorch website.
- Note that AlexNet/ResNet has 1000 neurons in the last layer in order to classify 1000 classes. In this task, you only have 2 classes, thus need only 2 neurons, one for each class. You need to modify the last layer.

Questions: Do you see any benefits with transfer learning? Why or why not?

3.3 Network pruning

After we are satisfied with the optimized model, we experiment the idea of network pruning. This part uses the CNNOptimized as the base model to study pruning.

Your tasks are as follows.

1. Network pruning: Design a random pruning method to prune weights randomly in a certain given percentage. There are different ways of pruning a network. You can do it either locally per layer or globally per network.

   (1) First option is to do this random pruning with the first convolutional layer.

   (2) Second option is to do this random pruning globally. Called the new model CNNPrunedF (prune the first layer) and CNNPrunedG (prune globally).

2. With the pruned models, perform inference and record accuracy.

3. Retrain to recover accuracy. Retrain the pruned models to get two new models, CNNPrunedFO, and CNNPrunedGO.

4. With the pruned models being retrained, perform inference and record accuracy of the two new models, CNNPrunedFO, and CNNPrunedGO.

Suggestions for implementing the above task:

- To realize this task, go through the network pruning tutorial on the PyTorch website.

- When re-training neural networks, see how many epochs can make the training converged.
- You may check the sizes of the models after pruning. You should be able to see the shrink of the sizes.
- You may check the computation time after pruning. For this, go through the PyTorch tutorial about Profiling first.

Questions: Do you see any benefits of network pruning? Why or why not?

3.4 Data quantization

Now we are going to explore the data quantization techniques, which turns the model from float32 to int8. There are three ways for quantization in PyTorch: Dynamic quantization, Static quantization, and QAT (Quantization Aware Training). We will learn all three methods in this lab. You can first read the documentation from https://pytorch.org/docs/stable/quantization.html#quantization-api-summary

3.4.1 Quantization tasks

Your tasks are as follows.

1. Run the Demo and tutorial below with codes provided, and answer questions.
2. Write your own code to quantize LeNet (from your Lab 1) with three methods in Pytorch. The dataset is MNIST.

3.4.2 Demo and tutorial

1. Observe the data-type changes.

   In this demo, you will observe the data types of weights changing from floating point 32 to qint8. The values of weights and results are also different. For this demo, you only need to run the code and record what you observe. If you are interested, you can change the seed to observe more data.

```
input f32 tensor([[-2.3166, -0.1232]])
You could observe that the results below are different
result,fp32 tensor([[0.3139]], grad_fn=<AddmmBackward>)
result,int8 tensor([[0.3113]])
You could observe that the weights' values and types below are different
modelf32 weight tensor([[-0.2883,  0.0234]])
modelint8 weight tensor([[-0.2872,  0.0226]], size=(1, 2), dtype=torch.qint8,
       quantization_scheme=torch.per_tensor_affine, scale=0.002261500107124448,
       zero_point=0)
modelint8 weight (int_repr) tensor([[-127,   10]], dtype=torch.int8)
```

Figure 1: One example to observe the changes when the seed is 123

The code is located at https://gits-15.sys.kth.se/wenyao/IL2230/blob/master/demo0.ipynb

2. Dynamic quantization

   The key idea with dynamic quantization as described here is that we are going to determine the scale factor for activations dynamically based on the data range observed at runtime. Go through the dynamic quantization tutorial below.

   https://pytorch.org/tutorials/recipes/recipes/dynamic_quantization.html

   Record the model size, latency and accuracy before quantization and after quantization. Notice that this example is a simple one-layer LSTM and the accuracy only shows the difference of the output.

3. Static quantization

   You can use the ResNet18 from section 3.2 and do static quantization on it. You can use the link below as a reference. https://gits-15.sys.kth.se/wenyao/IL2230/blob/master/Static_Quantization.ipynb

   This code should directly work if you set the directory and import package correctly.

   The position of your "hymenoptera_data" should be in the same layer as this Jupyter file. Or you can revise the code "$data\_dir = ./data/$" to be the location of your dataset.

4. Quantization Aware Training (QAT)

   You can follow the tutorial in the link below to run QAT.

   https://pytorch.org/tutorials/intermediate/quantized_transfer_learning_tutorial.html (Only part0 and part2 are needed for QAT).

   (a) Prepare the prerequisites Part0 in ref. website. You may use CPU to avoid installing cuda.

   ```
   #if you don't have CUDA, use CPU for load data and other parts
   device = torch.device("cpu")
   #if you have CUDA, use the code below
   device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
   ```

   (b) The pretrained and not quantized model (for example, resnet18) is loaded form *torchvision/models/quantization*.

   ```
   import torchvision.models.quantization as models
   model = models.resnet18(pretrained=True, progress=True, quantize=False)
   ```

   (c) To get a "baseline", transfer the model as you have done in CNNOptimized, and record the performance of model with fused modules and without quantization. You should use a memeber function $fuse\_model()$ to fuse the conv, bn, and relu modules.

   ```
   model.fuse_model()
   ```

   Model size, accuracy, and time consumption for inference should be recorded for this model. You need to write code to record these results.

(d) Use the *create_combined_model* function in the reference. This operation is to connect the feature extractor with a custom head which requires dequantizing the output of the feature extractor. After this step, you got a new model named model_ft.

```
model_ft = create_combined_model(model)
```

(e) Insert *fake-quantization* modules by using *torch.quantization.prepare_qat*.

```
model_ft=model_ft.to('cpu')
model_ft = torch.quantization.prepare_qat(model_ft, inplace=True)
```

(f) You can start "finetuning" the model. The function train_model is defined in the reference tutorial. The num_epochs can vary to reduce time consumption or increase accuracy.

```
model_ft_tuned = train_model(model_ft, criterion, optimizer_ft,
                             exp_lr_scheduler, num_epochs=25, device=device)
```

(g) Convert it to a fully quantized version. The default quantization data type is int8. In this lab, only the feature extractor is quantized.

```
model_quantized_and_trained = convert(model_ft_tuned, inplace=False)
```

(h) Record the model size, accuracy, and time consumption after conversion and compare it with the model before conversion.

Answer the question below:

- What are your achievements in model size, accuracy, and time consumption for inference?

3.4.3 Quantization practice on LeNet

As you have trained a LeNet for MNIST in Lab 1, you can practice the data quantization techniques with it, and see their impact on performance.

You need to write your own codes and debug them. The demos and tutorials in Section 3.4.2 can be your references to implement your designs.

1. Dynamic quantization of LeNet.
2. Static quantization of LeNet.
3. Quantize LeNet with QAT.

Answer the following questions:

- What are your achievements in model size, accuracy and time consumption for inference?
- Comparing with the inference time of LeNet in Lab 1, does the dynamic/ static/QAT quantization technique improve the performance? Why or why not?

3.5 Documentation

There are two deliverables as documentation:

1. Write a technical report (names of team members on the first page), describing your procedures for implementing the tasks, reporting results you obtained, and discussing your results.

   You need to answer the questions when your lab results are checked by TAs and write them in your report.

2. Upload your source code and the report to the Canvas course page. The source code should accompany a simple Readme.txt for how to run the code and obtain the graphs/numbers in your report.

   After your lab results have been approved by a lab assistant, submit your report via the Canvas course website:

   https://kth.instructure.com/courses/42938

3.6 Appendix

Links to PyTorch recipes and tutorials:

- PyTorch recipes

  https://pytorch.org/tutorials/recipes/recipes_index.html

  This includes a number of useful tools which help you in various aspects, for example, Loading data (how to use PyTorch packages to prepare and load common datasets for your model), Defining a neural network (how to create and define a neural network for the MNIST dataset), Saving and loading models for inference or a general checkpoint (how to save and load models for inference or a general checkpoint model for inference or resuming training), and Profiler (how to use PyTorch's profiler to measure operators time and memory consumption) etc.

- Transfer learning for computer vision

  In this tutorial, you will learn how to train a convolutional neural network for image classification using transfer learning.

  https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

- Network pruning

  In this tutorial, you will learn how to use torch.nn.utils.prune to sparsify your neural networks, and how to extend it to implement your own custom pruning technique.

  https://pytorch.org/tutorials/intermediate/pruning_tutorial.html

- Introduction to Quantization on PyTorch

  https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

- Computer vision models in PyTorch

  Here you can find a number of well-known neural networks (and their weights) for computer vision.

  https://github.com/pytorch/vision/tree/master/torchvision/models

  Alexnet

  https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py