# Laboration3B

# Transfer Learning, Network Pruning, and Quantization

Group1 Ruijia Dai, Ahmad Traboulsi, Sergiu Popescu

# 3.1 Learning From Scratch

To train a model to recognize ants and bees, we begin with defining a new net. In our first design we add two more convolutional layers, two more pooling layers and two more fully connected layers to LeNet-5 to form our new net. To regularize the size of all the images, we use resize function to transform all the images into 320x320x3 images.

The parameters of each layer is as below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| layer | filter | Filter size | stride | Feature map size | Activation function |
| input | - | - | - | 320 x 320 x 3 | - |
| convolution1 | 6 | 5 x 5 | 1 | 316 x 316 x 6 | tanh |
| Average pooling1 | - | 2 x 2 | 2 | 158 x 158 x 6 | - |
| convolution2 | 16 | 5 x 5 | 1 | 154 x 154 x 16 | tanh |
| Average pooling2 | - | 2 x 2 | 2 | 77 x 77 x 16 | - |
| convolution3 | 36 | 5 x 5 | 1 | 73 x 73 x 36 | tanh |
| Average pooling3 | - | 2 x 2 | 2 | 36 x 36 x 36 | - |
| convolution4 | 78 | 5 x 5 | 1 | 32 x 32 x 78 | tanh |
| Average pooling4 | - | 2 x 2 | 2 | 16 x 16 x 78 | - |
| Fully connected1 | - | - | - | 16 x 16 x 3 | tanh |
| Fully connected2 | - | - | - | 120 | tanh |
| Fully connected3 | - | - | - | 84 | tanh |
| Fully connected4 | - | - | - | 10 | softmax |

Table Scratch net structure

Accuracy: 45% (epochs=20)

Reasons that the accuracy is this low:

1. Training model with net structure not complex enough on a complex(many pixels, small difference) dataset
2. Training model on a dataset not large enough

We also tested another architecture (also based on the LeNet-5), but this time, instead of increasing the number of layers, we focused to catch more details in the feature maps.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| layer | filter | Filter size | stride | Feature map size | Activation function |
| input | - | - | - | 320 x 320 x 3 | - |
| convolution1 | 6 | 3 x 3 | 1 | 64 x 64 x 6 | tanh |
| Average pooling1 | - | 2 x 2 | 2 | 31 x 31 x 6 | - |
| convolution2 | 32 | 4 x 4 | 1 | 28 x 28 x 32 | tanh |
| Average pooling2 | - | 2 x 2 | 2 | 14 x 14 x 32 | - |
| Fully connected1 | - | - | - | 80 | tanh |
| Fully connected2 | - | - | - | 40 | tanh |
| Fully connected3 | - | - | - | 2 | tanh |

Table 2 Scratch net structure 2

Accuracy: 64% (epochs=5)

Reasons that the accuracy is higher:

1. Even if the architecture does not have more layers, the feature maps are larger, thus gathering more details about the image
2. The classification layers focus only on the desired classes

# 3.2 Transfer Learning

We load pretrained AlexNet model[[1]](#footnote-1) and modify it to implement transfer learning in this task.

## 3.2.1 Modify the last(output) layer

Since AlexNet can recognize 1000 classes and we only need it to recognize 2 classes in this task, we modify the output layer of the model. We change the neurons of output layer to two.

## 3.2.2 Retrain the last(output) layer

After modifying the output layer of the model, we need to retrain and modify the weights and biases in the output layer. In this case we don’t want to modify the other parameters in the model, so we configure the model with code below to avoid unexpected parameters modification.

for param in model\_conv.parameters():

param.requires\_grad = False

for param in model\_conv.classifier[-1].parameters():

param.requires\_grad = True

Best validation accuracy: 87.58%

Training configuration:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Learning rate | momentum | Step\_size | gamma | epochs |
| 0.0001 | 0.9 | 7 | 0.1 | 25 |

Source code: https://github.com/ArtificialCoincidence/IL2230\_lab3/blob/main/transfer\_retrain\_lastlayer.ipynb

## 3.2.3 Retrain the whole model

In this task, we retrain the whole model on the hymenoptera dataset. Since it is more data centric compared to 3.2.2, we are expecting a higher best validation accuracy.

Best validation accuracy: 88.24%

Training configuration:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Learning rate | momentum | Step\_size | gamma | epochs |
| 0.0001 | 0.9 | 7 | 0.1 | 25 |

Source code: https://github.com/ArtificialCoincidence/IL2230\_lab3/blob/main/transfer\_retrain\_lastlayer.ipynb

## 3.2.4 Summary

Table Transfer learning result

**Benefits of transfer learning**:

1. Increase validation accuracy apparently
2. Do not increase training time apparently

**Retrain whole model vs. retrain output layer only**

More parameters are to calculate when retaining whole model, resulting:

1. Retrain whole model need longer training time
2. Retrain whole model help achieve higher accuracy

# 3.3 Network Pruning

We use CNNOptimized generated in task3.2.3 as the base model. We choose prune.random\_unstructured as pruning function.[[2]](#footnote-2)

## 3.3.1 First Layer Pruning

We prune 30% of the weights in the first layer of the model with code as below:

prune\_module = model.features[0]

# randomly prune 30% of the connections in the parameter named weight in the first layer

prune.random\_unstructured(prune\_module, name="weight", amount=0.3)

model.features[0] = prune\_module

Accuracy before pruning: 92.1569%

Accuracy after pruning: 75%

Accuracy after retraining: 91.5033%

Converge steps: 10

Retraining configuration:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Learning rate | momentum | Step\_size | gamma | epochs |
| 0.0001 | 0.9 | 7 | 0.1 | 30 |

Source code: <https://github.com/ArtificialCoincidence/IL2230_lab3/blob/main/prune_firstlayer.ipynb>

## 3.3.2 Whole Model Pruning

We prune 20% of the weights in all convolutional layers and 40% of the weights in all linear layers with code as below:

for name, module in model.named\_modules():

# prune 20% of connections in all 2D-conv layers

if isinstance(module, torch.nn.Conv2d):

prune.random\_unstructured(module, name='weight', amount=0.2)

# prune 40% of connections in all linear layers

elif isinstance(module, torch.nn.Linear):

prune.random\_unstructured(module, name='weight', amount=0.4)

Accuracy before pruning: 92.1569%

Accuracy after pruning: 66.67%

Accuracy after retraining: 85.62%

Converge steps: 8

Retraining configuration:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Learning rate | momentum | Step\_size | gamma | epochs |
| 0.0001 | 0.9 | 7 | 0.1 | 30 |

Source code: https://github.com/ArtificialCoincidence/IL2230\_lab3/blob/main/prune\_whole\_gpu.ipynb

## 3.3.3 Summary

**Benefits of network pruning**:

Save computation time at run time apparently when running with GPU. The computation time after pruning increases when running with CPU because when weights are pruned, the resulting weight matrices become sparse (many zero values). CPU is not optimized for sparse matrices operations as GPU.

The size of file storing parameters of the model(generated by function save\_state\_dict) increases after pruning because of the implementation of pruning in function random\_unstructured.

1. https://pytorch.org/vision/main/\_modules/torchvision/models/alexnet.html#AlexNet\_Weights [↑](#footnote-ref-1)
2. https://pytorch.org/docs/stable/generated/torch.nn.utils.prune.random\_unstructured.html#torch.nn.utils.prune.random\_unstructured [↑](#footnote-ref-2)