AlexNet VS. ResNet

Zhuoquan Chen

INSTRUCTION:

The purpose of this test is to compare the performance of AlexNet and ResNet models in fine tuning. In this test, I will focus on two parts, which the accuracies of prediction how different for both models with resetting final fully connected layers and freezing all the network except the final layer. Here are what I prepared:

- AlexNet model and ResNet model
- "hymenoptera_data" dataset

AlexNet:

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False))
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
    )
)
```

Phase 1: The outcome of resetting final fully connected layers

```
Epoch 0/14
----------
train Loss: 0.6595 Acc: 0.6598
val Loss: 0.4382 Acc: 0.8235

Epoch 1/14
----------
train Loss: 0.4752 Acc: 0.7664
val Loss: 0.4759 Acc: 0.8105

Epoch 2/14
----------
train Loss: 0.4995 Acc: 0.7787
val Loss: 0.4776 Acc: 0.7778

Epoch 3/14
----------
train Loss: 0.3268 Acc: 0.8566
val Loss: 0.6494 Acc: 0.8301

Epoch 4/14
----------
train Loss: 0.4099 Acc: 0.8279
val Loss: 0.4451 Acc: 0.8301

Epoch 5/14
----------
train Loss: 0.4701 Acc: 0.7951
val Loss: 0.5185 Acc: 0.6928

Epoch 6/14
----------
train Loss: 0.3866 Acc: 0.8320
val Loss: 0.2749 Acc: 0.8889
Epoch 7/14
----------
train Loss: 0.2740 Acc: 0.8852
val Loss: 0.2851 Acc: 0.8627

Epoch 8/14
----------
train Loss: 0.2431 Acc: 0.9057
```

```
val Loss: 0.3033 Acc: 0.8824

Epoch 9/14
----------
train Loss: 0.2497 Acc: 0.8893
val Loss: 0.2900 Acc: 0.8889

Epoch 10/14
----------
train Loss: 0.2187 Acc: 0.8975
val Loss: 0.2969 Acc: 0.8824

Epoch 11/14
----------
train Loss: 0.2277 Acc: 0.8934
val Loss: 0.2945 Acc: 0.8889

Epoch 12/14
----------
train Loss: 0.1979 Acc: 0.9426
val Loss: 0.2922 Acc: 0.8889

Epoch 13/14
----------
train Loss: 0.1998 Acc: 0.9180
val Loss: 0.2901 Acc: 0.9020

Epoch 14/14
----------
train Loss: 0.1743 Acc: 0.9262
val Loss: 0.2924 Acc: 0.8889
Training complete in 14m 23s
Best val Acc: 0.901961
```

Phase 2: The outcome of freezing all the network except the final layers

```
Epoch 0/14
----------
train Loss: 0.6314 Acc: 0.6598
val Loss: 0.2297 Acc: 0.9477

Epoch 1/14
----------
train Loss: 0.5601 Acc: 0.7500
```

```
val Loss: 0.2439 Acc: 0.9281

Epoch 2/14
----------
train Loss: 0.4822 Acc: 0.7869
val Loss: 0.1763 Acc: 0.9346

Epoch 3/14
----------
train Loss: 0.4494 Acc: 0.7910
val Loss: 0.1831 Acc: 0.9412

Epoch 4/14
----------
train Loss: 0.4496 Acc: 0.7992
val Loss: 0.2922 Acc: 0.9085

Epoch 5/14
----------
train Loss: 0.3606 Acc: 0.8525
val Loss: 0.3132 Acc: 0.9020

Epoch 6/14
----------
train Loss: 0.4802 Acc: 0.7828
val Loss: 0.1922 Acc: 0.9477
Epoch 7/14
----------
train Loss: 0.3938 Acc: 0.8156
val Loss: 0.2064 Acc: 0.9412

Epoch 8/14
----------
train Loss: 0.3838 Acc: 0.8361
val Loss: 0.2250 Acc: 0.9216

Epoch 9/14
----------
train Loss: 0.3087 Acc: 0.8566
val Loss: 0.2187 Acc: 0.9346

Epoch 10/14
----------
train Loss: 0.3046 Acc: 0.8689
```

```
val Loss: 0.2033 Acc: 0.9346

Epoch 11/14
----------
train Loss: 0.3461 Acc: 0.8361
val Loss: 0.1966 Acc: 0.9477

Epoch 12/14
----------
train Loss: 0.3689 Acc: 0.8443
val Loss: 0.2092 Acc: 0.9412

Epoch 13/14
----------
train Loss: 0.4293 Acc: 0.8279
val Loss: 0.1785 Acc: 0.9477

Epoch 14/14
----------
train Loss: 0.3445 Acc: 0.8607
val Loss: 0.2066 Acc: 0.9346
Training complete in 11m 6s
Best val Acc: 0.947712
```

ResNet:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True))
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
  )
)
(layer2): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
(layer3): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
(layer4): Sequential(
  (0): BasicBlock(
```

```
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Phase 1: The outcome of resetting final fully connected layers

```
Epoch 0/14
----------
train Loss: 0.5882 Acc: 0.7459
val Loss: 0.1426 Acc: 0.9346

Epoch 1/14
----------
train Loss: 0.5981 Acc: 0.7951
val Loss: 0.4745 Acc: 0.8301

Epoch 2/14
----------
train Loss: 0.6853 Acc: 0.7828
val Loss: 0.8639 Acc: 0.7778

Epoch 3/14
----------
train Loss: 0.9354 Acc: 0.6967
val Loss: 0.3016 Acc: 0.8758
```

```
Epoch 4/14
----------
train Loss: 0.6290 Acc: 0.7541
val Loss: 0.3368 Acc: 0.8562

Epoch 5/14
----------
train Loss: 0.5338 Acc: 0.7541
val Loss: 0.4152 Acc: 0.8301

Epoch 6/14
----------
train Loss: 0.4063 Acc: 0.8320
val Loss: 0.3412 Acc: 0.9020

Epoch 7/14
----------
train Loss: 0.4077 Acc: 0.8279
val Loss: 0.2137 Acc: 0.9477

Epoch 8/14
----------
train Loss: 0.3439 Acc: 0.8770
val Loss: 0.2119 Acc: 0.9412

Epoch 9/14
----------
train Loss: 0.3291 Acc: 0.8607
val Loss: 0.2117 Acc: 0.9346

Epoch 10/14
----------
train Loss: 0.4157 Acc: 0.8443
val Loss: 0.2027 Acc: 0.9412

Epoch 11/14
----------
train Loss: 0.2794 Acc: 0.9016
val Loss: 0.2076 Acc: 0.9346

Epoch 12/14
----------
train Loss: 0.2976 Acc: 0.8770
val Loss: 0.2107 Acc: 0.9281
```

```
Epoch 13/14
----------
train Loss: 0.2501 Acc: 0.9057
val Loss: 0.1999 Acc: 0.9412

Epoch 14/14
----------
train Loss: 0.3074 Acc: 0.8811
val Loss: 0.2140 Acc: 0.9412

Training complete in 20m 60s
Best val Acc: 0.947712
```

Phase 2: The outcome of freezing all the network except the final layers

```
Epoch 0/14
----------
train Loss: 0.6285 Acc: 0.6475
val Loss: 0.2612 Acc: 0.9085

Epoch 1/14
----------
train Loss: 0.5436 Acc: 0.7459
val Loss: 0.2110 Acc: 0.9150

Epoch 2/14
----------
train Loss: 0.5486 Acc: 0.7500
val Loss: 0.3061 Acc: 0.8824

Epoch 3/14
----------
train Loss: 0.7213 Acc: 0.7336
val Loss: 0.2644 Acc: 0.9216

Epoch 4/14
----------
train Loss: 0.3707 Acc: 0.8443
val Loss: 0.1817 Acc: 0.9477

Epoch 5/14
----------
train Loss: 0.4422 Acc: 0.8197
```

```
val Loss: 0.3583 Acc: 0.8758

Epoch 6/14
----------
train Loss: 0.5729 Acc: 0.7869
val Loss: 0.2564 Acc: 0.9216

Epoch 7/14
----------
train Loss: 0.4123 Acc: 0.8320
val Loss: 0.1989 Acc: 0.9477

Epoch 8/14
----------
train Loss: 0.3596 Acc: 0.8238
val Loss: 0.1899 Acc: 0.9477

Epoch 9/14
----------
train Loss: 0.3535 Acc: 0.8484
val Loss: 0.1852 Acc: 0.9477

Epoch 10/14
----------
train Loss: 0.3129 Acc: 0.8525
val Loss: 0.1914 Acc: 0.9412

Epoch 11/14
----------
train Loss: 0.2913 Acc: 0.8689
val Loss: 0.1811 Acc: 0.9542

Epoch 12/14
----------
train Loss: 0.3694 Acc: 0.8361
val Loss: 0.1829 Acc: 0.9542

Epoch 13/14
----------
train Loss: 0.3626 Acc: 0.8607
val Loss: 0.2132 Acc: 0.9216

Epoch 14/14
----------
```

```
train Loss: 0.3358 Acc: 0.8279
val Loss: 0.2175 Acc: 0.9346

Training complete in 9m 5s
Best val Acc: 0.954248
```

CONCLUSION:

In this text, the phase 1 of AlexNet model's training completes in 14m and 23s, the best accuracy is 0.901961, while the phase 2 training completes in 11m and 6s, the best accuracy is 0.947712, so the final outcome of AlexNet model is 25m and 29s. The phase 1 of ResNet model's training completes in 20m and 60s, the best accuracy is 0.947712, while the phase 2 training completes in 9m and 5s, the best accuracy is 0.954248, so the final outcome of ResNet model is 30m and 5s. From resetting final fully connected layers to freezing all the network except the final layers, the accuracy does improve. In this case, the improvement rate of AlexNet model is higher than ResNet model, and ResNet uses more run time than AlexNet, which is over expected. Because ofthe parameters of AlexNet is more than Resnet, so AlexNet's calculation should be more expensive than ResNet. Therefore, I guess that the dept of layer determines the run time. Finally, the test shows that the accuracy of ResNet will be higher a little than AlexNet.