CV_HW3 Big vs Small Models 107081028 常安彥

<u>Discussion</u>

I implemented all of the training within 20 epochs in order to fit in the limited resource of colab, recording the testing accuracy and testing loss.

20 epochs Test_acc (loss)	**resnet18**	**resnet50**	**resnet18 (pretrained)**	**resnet50 (pretrained)**
Whole	0.76 (0.7490)	0.74 (0.803)	0.85 (0.4489)	0.87 (0.4041)
1/2	0.74 (0.7485)	0.43 (2.0357)	0.84 (0.5058)	0.84 (0.4742)
1/16	0.54 (1.2784)	0.42 (1.5748)	0.74 (0.7823)	0.74 (0.7994)

1. Big vs Small

   According to the result, both models didn't show obvious difference when training the whole cifar-10 dataset. Cifar-10 is not a huge dataset with huge numbers of features, so big model doesn't show higher performance.

   Nevertheless, as the dataset shrinks, big model takes more time to converge. We can see from the result, big models didn't learn well within 20 epochs, while small models predicted better.

2. Pretrained

   In my case, I only trained the models for 20 epochs, which is a short duration. Therefore, transfer learning benefits in this situation. **Pre-training on ImageNet made the models simply apply on new tasks,** performing a lot better than training from scratch.

   Furthermore, pre-training helped models acknowledge general information. **Few data in target domain didn't strongly affect the performance of the models.** According to the result, pre-trained models trained on smaller dataset still outperformed original models trained from scratch.

<u>Best Performance</u>

1. Model: resnet152 (output layer 2048 to 10), weights = "IMAGENET1K_V2"
2. Preprocessing & Augmentation
   a. Random Crop (padding = 4)
   b. Resize (224)

      The pretrained model is implemented on 'ImageNet,' which is composed of 224*224*3 images. Therefore, I resized input image into 224*224

   c. Random Horizontal Flip
   d. Normalization ((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))

      This set of normalization is extracted from 'ImageNet' training.

e. Rotation (X)

Rotation is a good augmentation method, but images in cifar-10 are too small, so rotation only make the performance worse.

f. High Pass Filter (X)

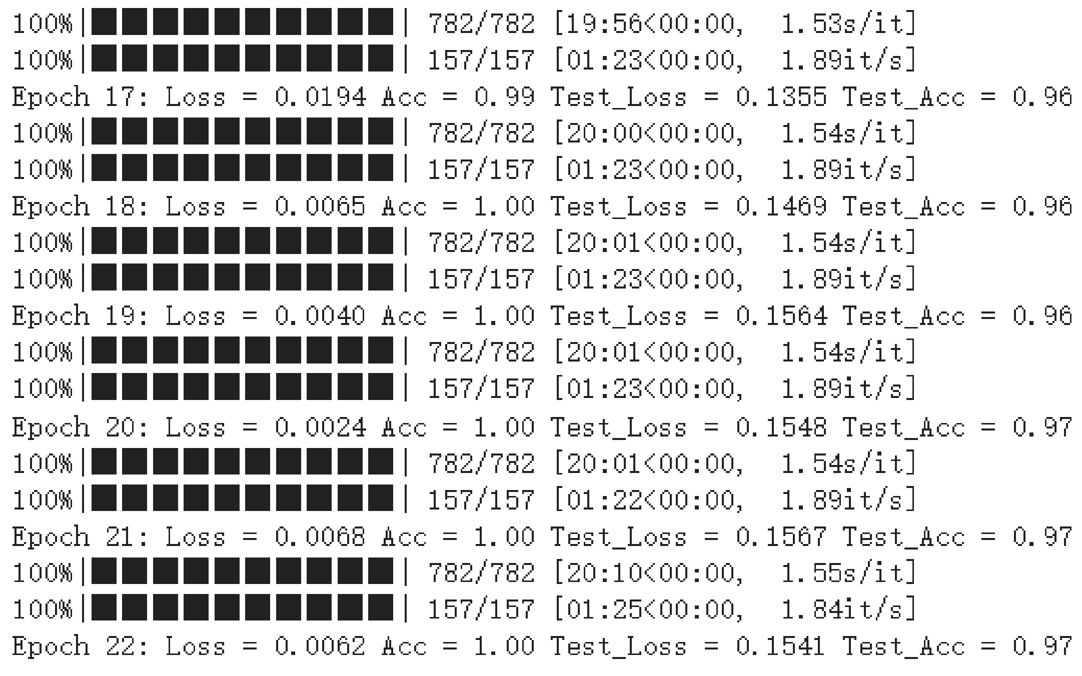
Same reason as above, images include too little information. These kinds of augmentation only make it worse.

3. Transfer Learning & Early Stopping

Google Colab doesn't provide enough GPU usage for users to train this case until it converges in once. Therefore, I store the model with the lowest testing loss during training, and load the best model the next day after running out of GPU usage. Furthermore, I appropriately decrease the learning rate every time I reload the model in order to look for a better optimization.

Finally, the code stops at a threshold where the testing loss continuously increases three times. This early stopping helps to cut down the training at a suitable timing, and avoid over-fitting.

4. Best Testing Accuracy: **0.97**, Best Testing Loss: 0.1355



```
100%|███████████| 782/782 [19:56<00:00,  1.53s/it]
100%|███████████| 157/157 [01:23<00:00,  1.89it/s]
Epoch 17: Loss = 0.0194 Acc = 0.99 Test_Loss = 0.1355 Test_Acc = 0.96
100%|███████████| 782/782 [20:00<00:00,  1.54s/it]
100%|███████████| 157/157 [01:23<00:00,  1.89it/s]
Epoch 18: Loss = 0.0065 Acc = 1.00 Test_Loss = 0.1469 Test_Acc = 0.96
100%|███████████| 782/782 [20:01<00:00,  1.54s/it]
100%|███████████| 157/157 [01:23<00:00,  1.89it/s]
Epoch 19: Loss = 0.0040 Acc = 1.00 Test_Loss = 0.1564 Test_Acc = 0.96
100%|███████████| 782/782 [20:01<00:00,  1.54s/it]
100%|███████████| 157/157 [01:23<00:00,  1.89it/s]
Epoch 20: Loss = 0.0024 Acc = 1.00 Test_Loss = 0.1548 Test_Acc = 0.97
100%|███████████| 782/782 [20:01<00:00,  1.54s/it]
100%|███████████| 157/157 [01:22<00:00,  1.89it/s]
Epoch 21: Loss = 0.0068 Acc = 1.00 Test_Loss = 0.1567 Test_Acc = 0.97
100%|███████████| 782/782 [20:10<00:00,  1.55s/it]
100%|███████████| 157/157 [01:25<00:00,  1.84it/s]
Epoch 22: Loss = 0.0062 Acc = 1.00 Test_Loss = 0.1541 Test_Acc = 0.97
```