Deep Learning Spring 2020
Assignment 2

Minghao Zhang

1 Task A

It's easy to train a pre-trained model. The train loss and valid loss converge within 80 epoches. The result curves are listed as follow:

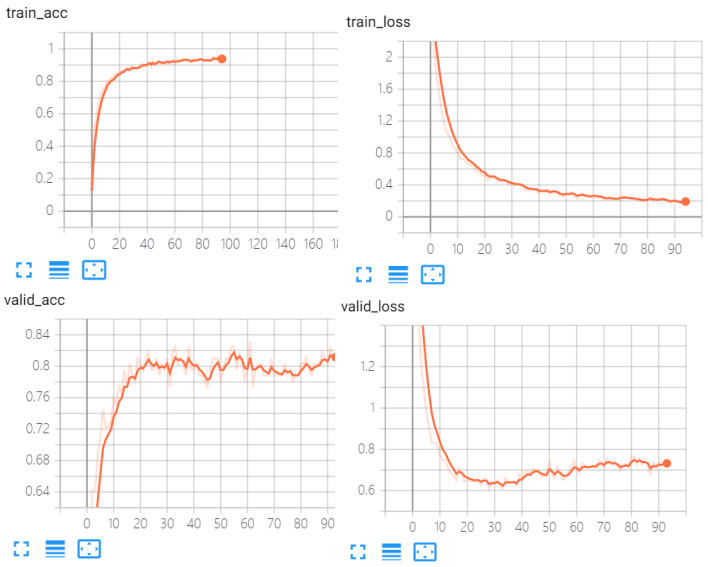

Figure 1: ResNet with pretrain

2 Task B

When turning the $pre\_trained = False$, it became so hard to get a good result. To ensure the convergence of training, I extended the training epoches to 200 and got the following results with comparison to the pretrained model:

100th Conference on Neural Information Processing Systems (NeurIPS 2086), Beijing, China.

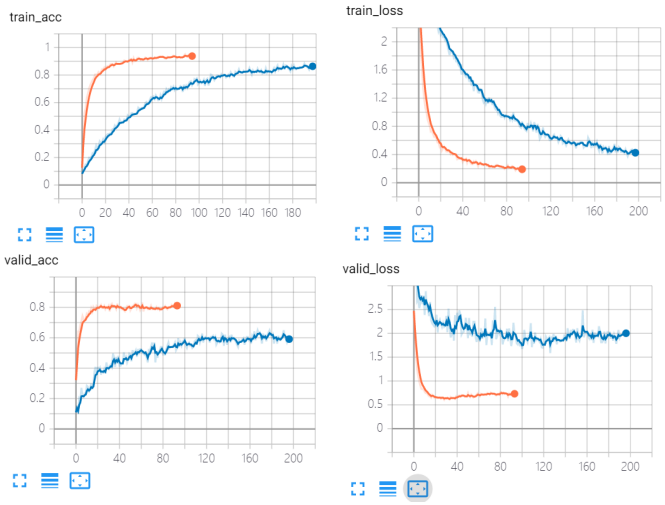

Figure 2: Blue curve represents ResNet without pretrain

3 Task C

3.1 Model Design

Before finding a good model for a task, we should look deep into the task first. The given dataset is so small so any deep and large model like Inception[1], ResNet[2], DenseNet[3] will easily overfit. So one key point for our model is to ensure the depth of network couldn't be large. So first I tried a simple model with 4 convolutional layers and two linear layers with some batch normalization layers and relu layers (the architecture can be found in the code).

Luckily, it directly outperformed the baseline (resnet without pre-train). With some tuning for hyperparameters, I got a best group:

- batch size: 8. As the number of data is small, the large batch size of inputs may have some bad influences;
- optimizer: Adam with weight decay 5e-9. Adam ensures the stable training process and weight decay applies the ability of generalization;
- learning rate: 0.0012. This learning rate is smaller than the given code setting. But for a small dataset, it's sufficient and efficient.

3.2 Results

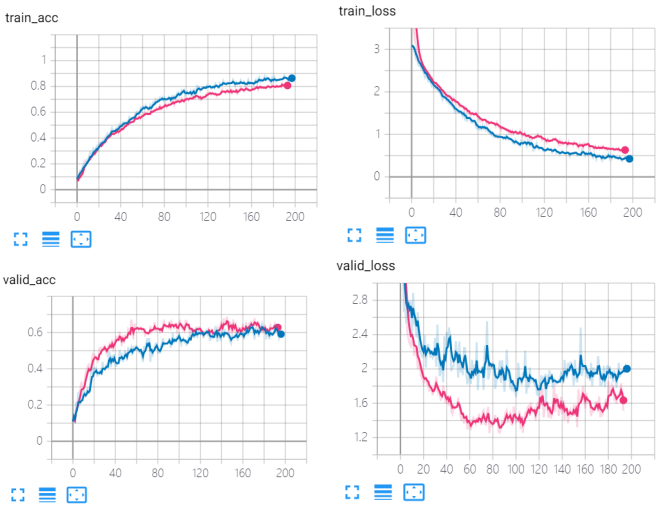

Figure 3: Blue curve represents ResNet without pretrain, Pink curve represents my model

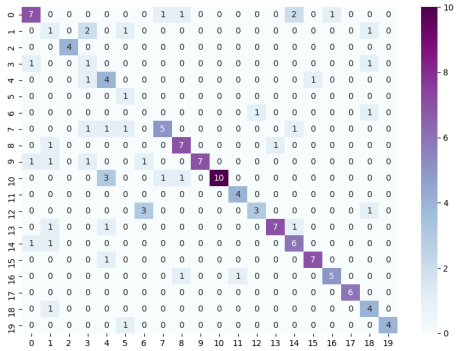

Figure 4: Confusion Matrix of Validation Set

3.3 Further Improvement

First, I tried the required two tricks: data augmentation and learning rate decay. The first one was done by changing the data transforms function in *data.py*. Specifically, I changed the default parameters of *RandomResizedCrop* function to add the variety of data. The second only need to use the *StepLR* optimizer in *torch.nn.optim*. The results shows that **data augmentation works better than the lr decay**.

To overcome the overfitting problem, I tried *dropout* strategy. I added some dropout layer with different probability (0.1/0.2/0.3) among the original layers after *relu*. However, the best result is not so efficient either:

During experiment, I found a significant phenomenon: the inconsistency between validation loss and validation accuracy. Sometimes the loss was smaller but the accuracy was not higher. I guess this is because **the CE loss only computes a global distance between the two distribution but such a small dataset requires more attention to the class probability**. In another word, we must know which classes lead the big loss. Focal loss[4] works well for solving the imbalance of classes in dataset in classificaion and object detection task. It introduced the class probability into the original CE loss, which is similar to what I wanted. So I implemented it and found it work well with $\alpha = 1.0,\ \gamma = 1.5$. With the focal loss, I got the final best results:

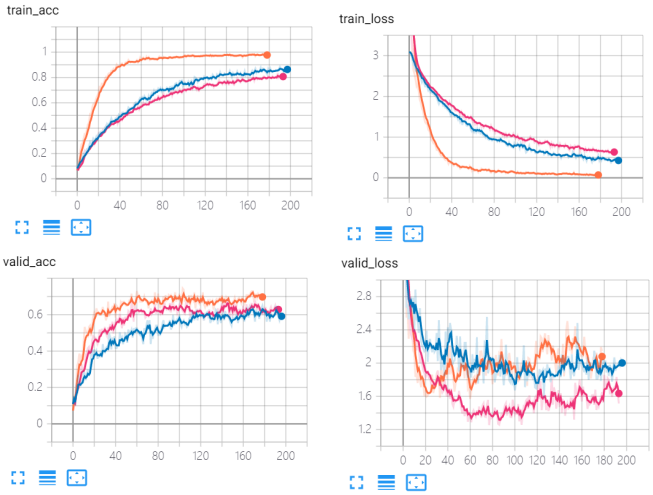

Figure 5: Orange curve represents data augmentation

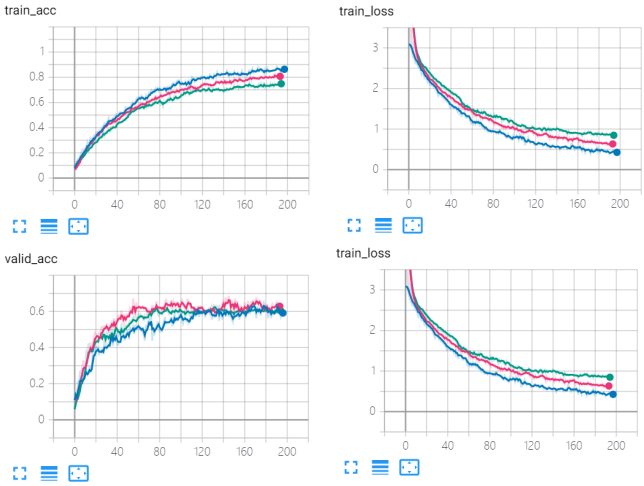

Figure 6: Green curve represents l

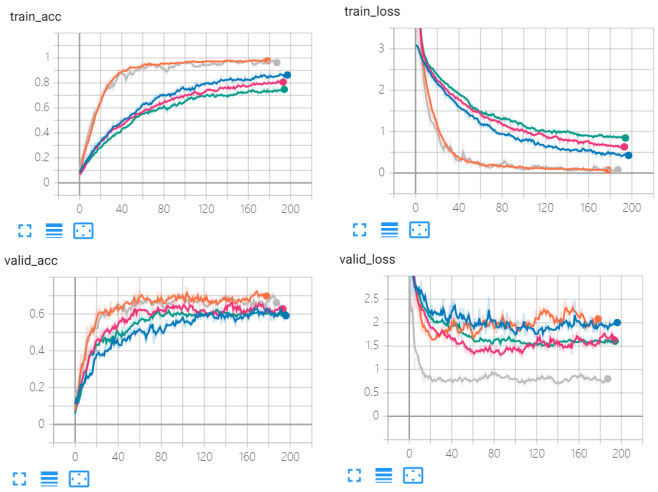

Figure 7: Grey curve represents lr decay+data augmentation

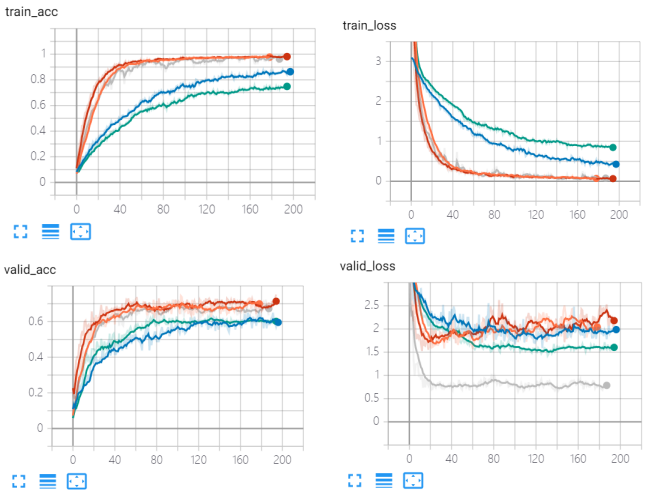

Figure 8: Red curve represents the best results with focal loss

3.4 Visualization

I developed the t-SNE code by myself and used the given PyTorch visualization toolkit https://github.com/utkuozbulak/pytorch-cnn-visualizations to complete this part. For conv feature visualization, I borow the cnn visualization tool from the mentioned repository.
For t-SNE, I used all the data in validation set and used the sklearn class TSNE with $n\_components = 2$. For cnn feature visualization, I randomly chose 4 images and got the features in 4 convolutional layers for each image. Since the pictures are a bit of large, I put them after the reference.

References

[1] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[4] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

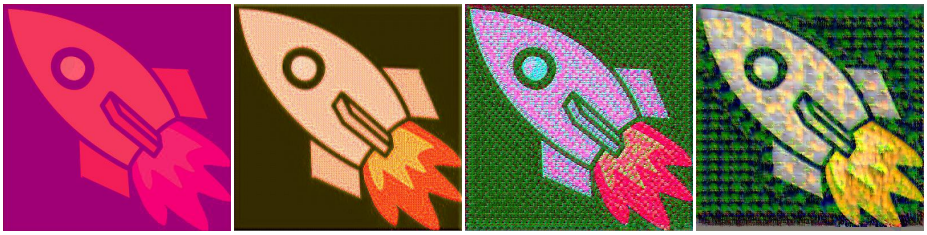

Figure 9: The feature of Conv1-4

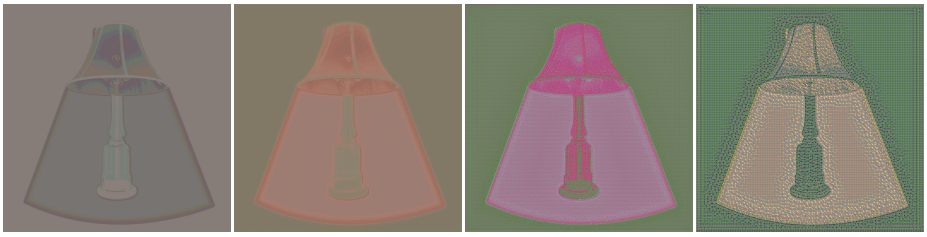

Figure 10: The feature of Conv1-4

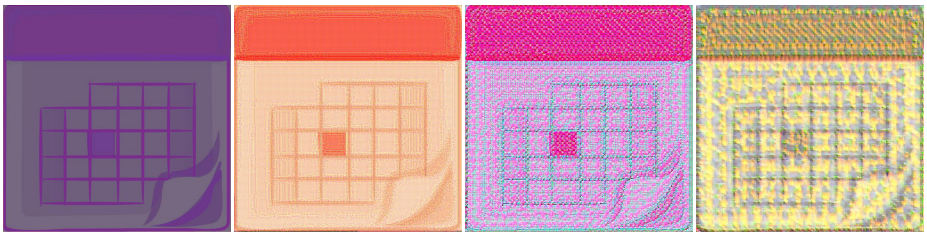

Figure 11: The feature of Conv1-4

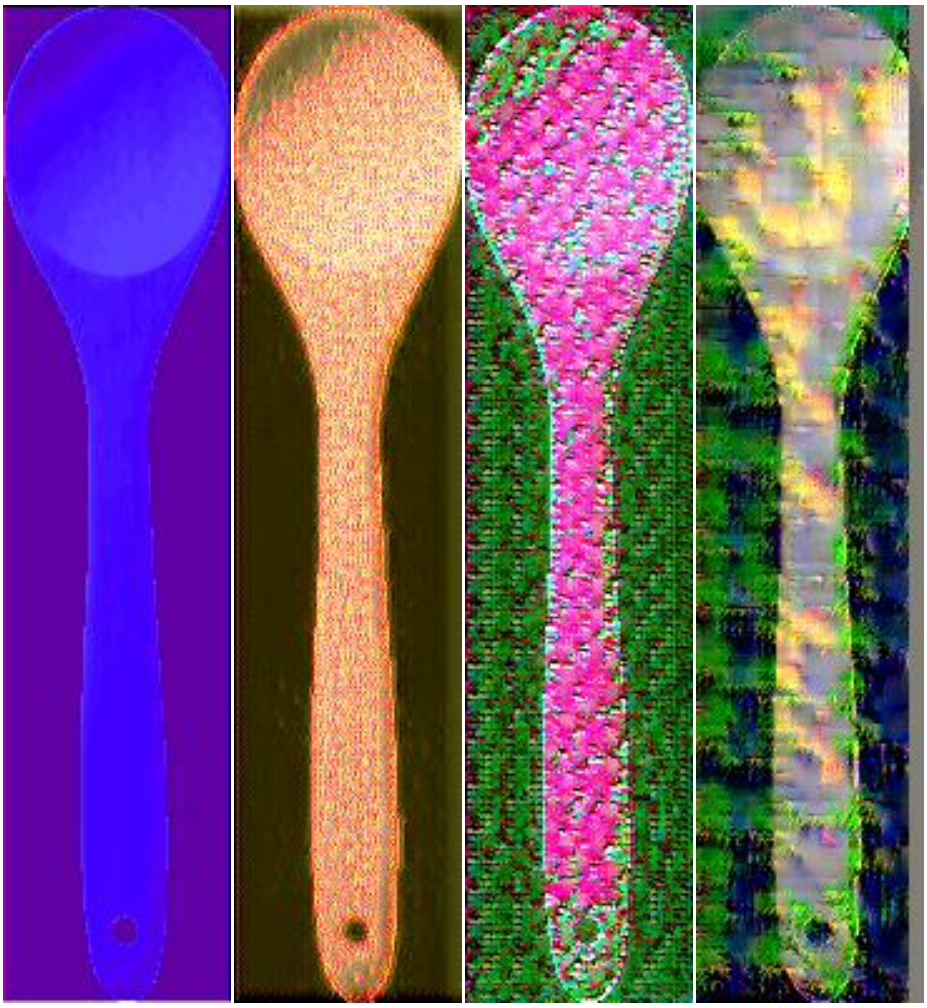

Figure 12: The feature of Conv1-4

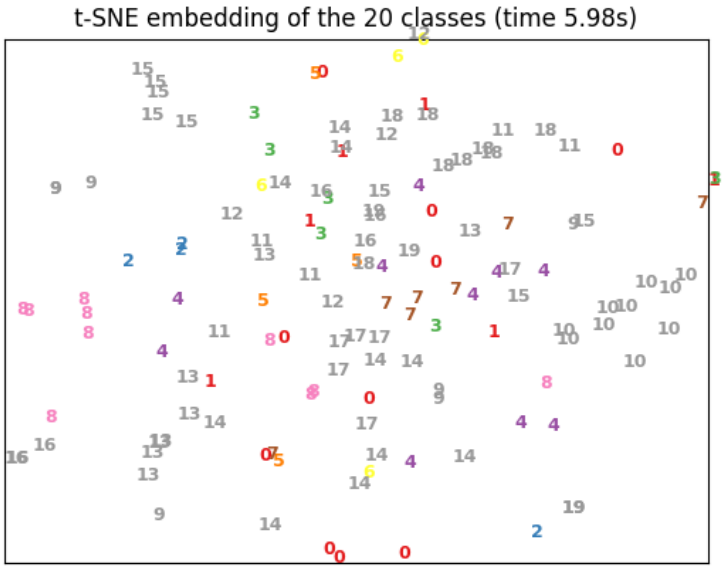

Figure 13: t-SNE on Validation Set