EE6485 Computer Vision: Homework 3

111061553 謝霖泳

November 16, 2022

1. The codes for Problem 1 and Problem 2

   (a) Problem 1

   Since the original **ResNet18** is designed for ImageNet dataset to predict 1000 classes, we should simply change the output of the model to 10 class.

```
model = torch.hub.load('pytorch/vision:v0.10.0',
                       'resnet18', weights=None)
model.fc = nn.Linear(in_features=512, out_features=10,
                     bias=True)
model = model.to(device)
```

   For **ResNet50**, we should mind that the input size of the very last layer is 2048, which is different to that of **ResNet18**. Then we modify it in a similar way.

```
model = torch.hub.load('pytorch/vision:v0.10.0',
                       'resnet50', weights=None)
model.fc = nn.Linear(in_features=2048, out_features=10,
                     bias=True)
model = model.to(device)
```

   (b) Problem 2

   For `train()` and `test()` functions, just follow the codes provided in the tutorial.

2. Results

(a) Models without initialized weights

i. 1/16 data

- ResNet18

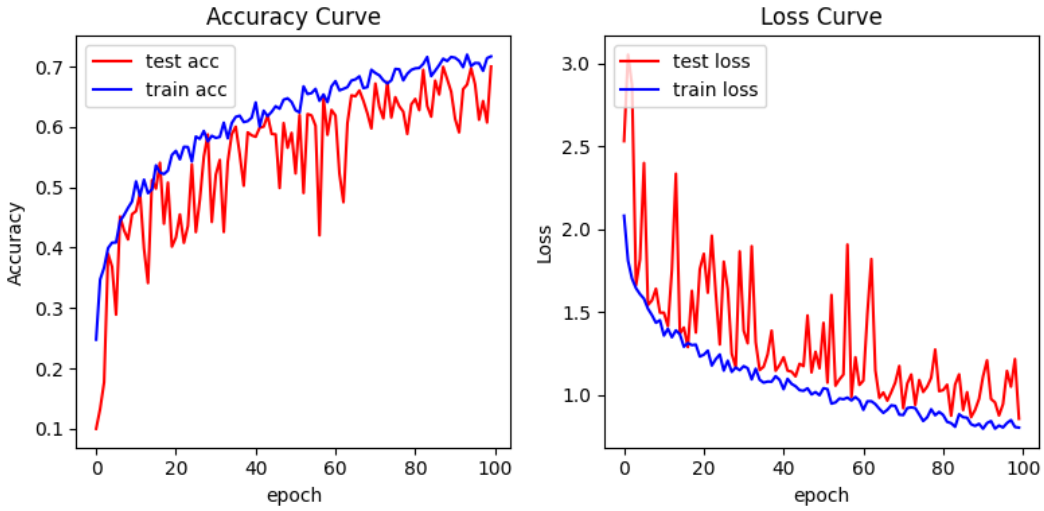

Figure 1: ResNet18 training on 1/16 dataset

- ResNet50

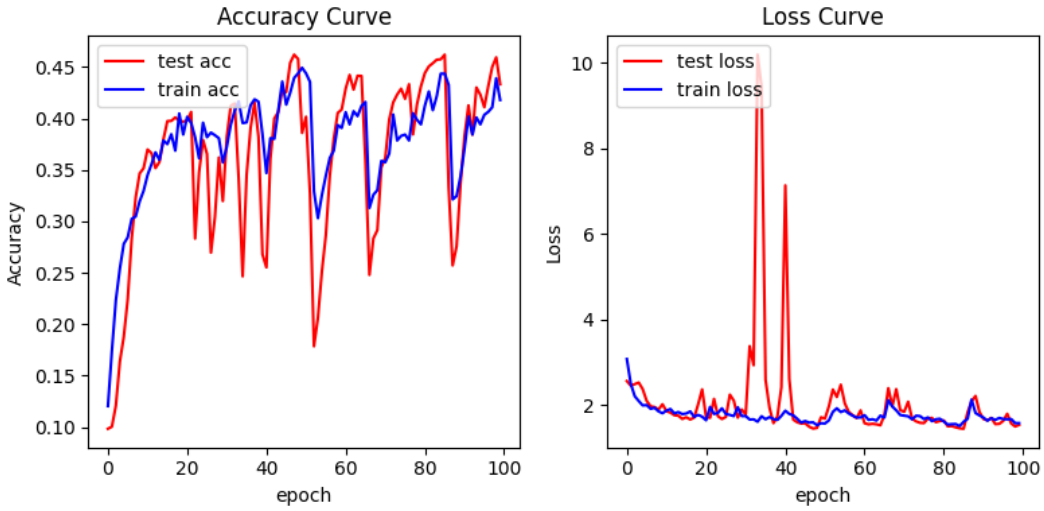

Figure 2: ResNet50 training on 1/16 dataset

ii. half data

- ResNet18

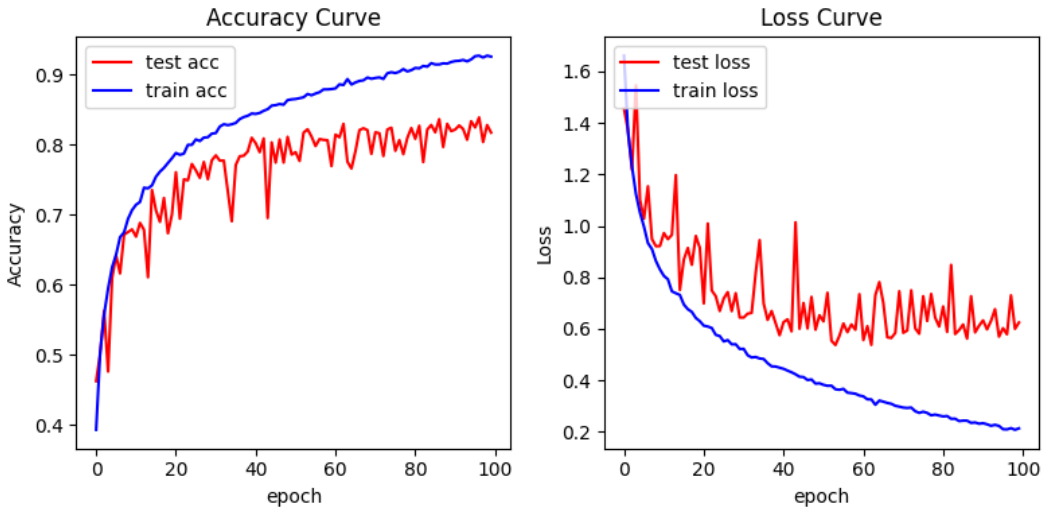

Figure 3: ResNet18 training on half dataset

- ResNet50

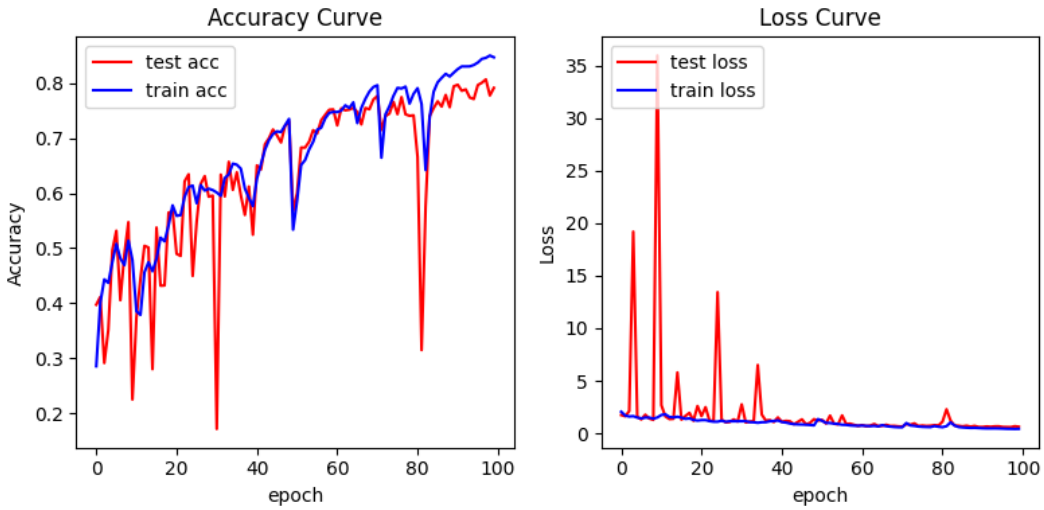

Figure 4: ResNet50 training on half dataset

iii. all data

- ResNet18

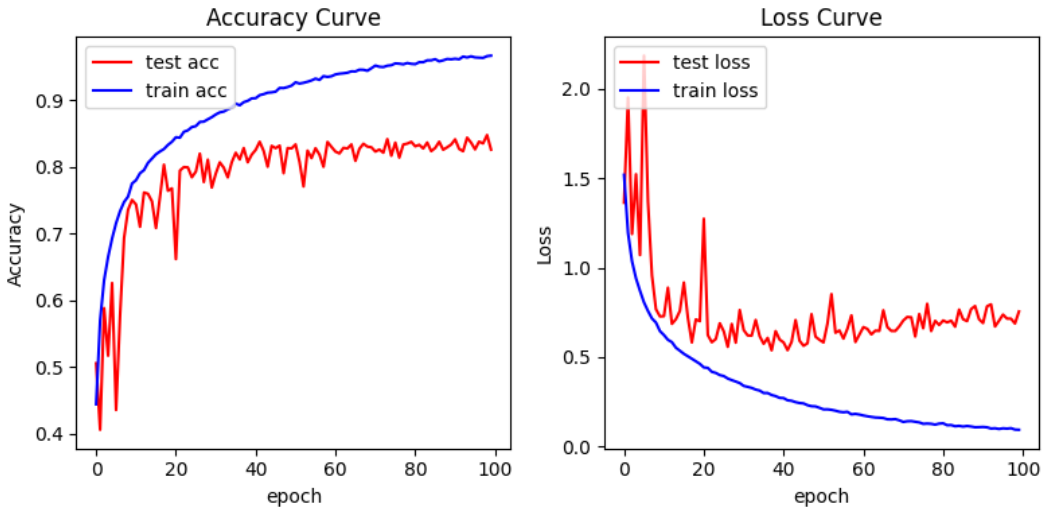

Figure 5: ResNet18 training on all dataset

- ResNet50

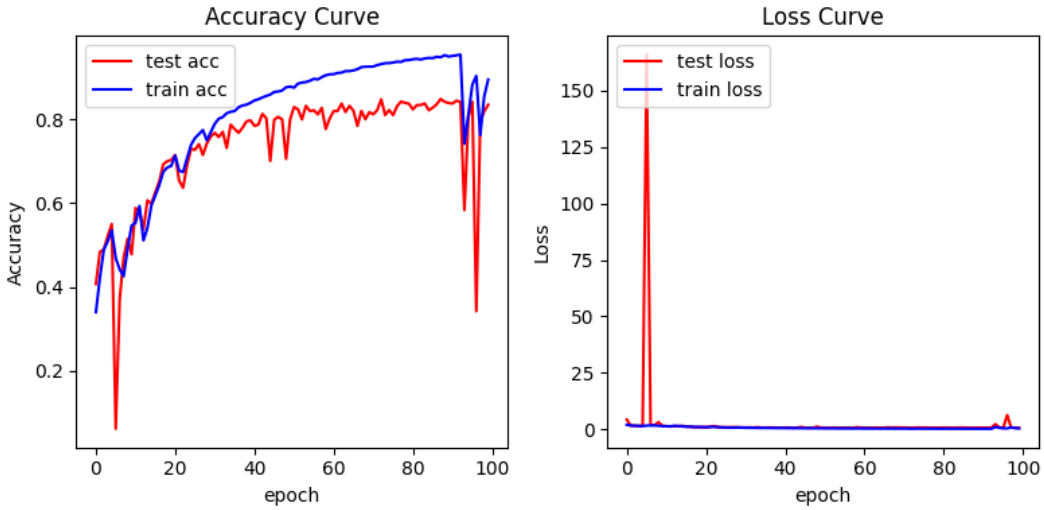

Figure 6: ResNet50 training on all dataset

(b) Models with initialized weights (represented by `Model*`)

(c) 1/16 data

- ResNet18*

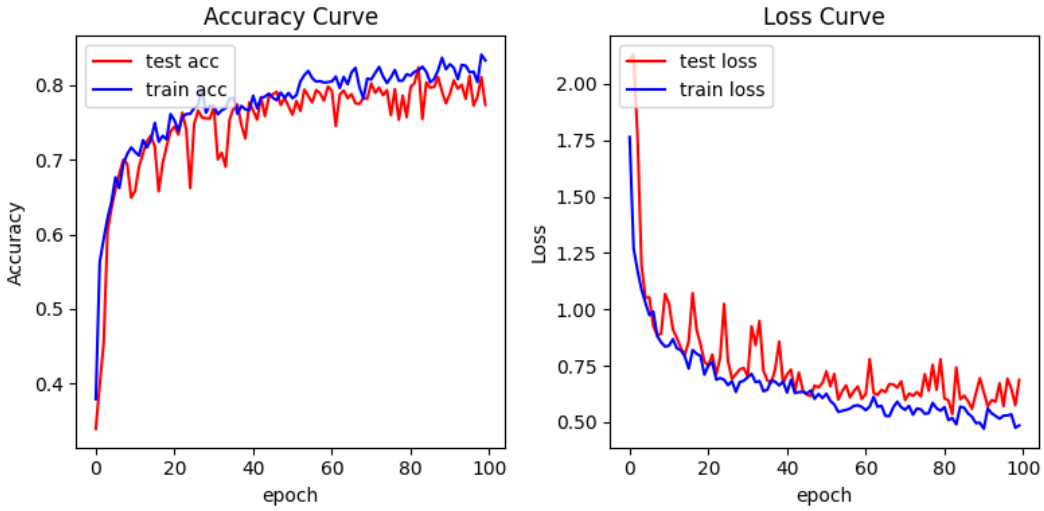

Figure 7: ResNet18* training on 1/16 dataset

- ResNet50*

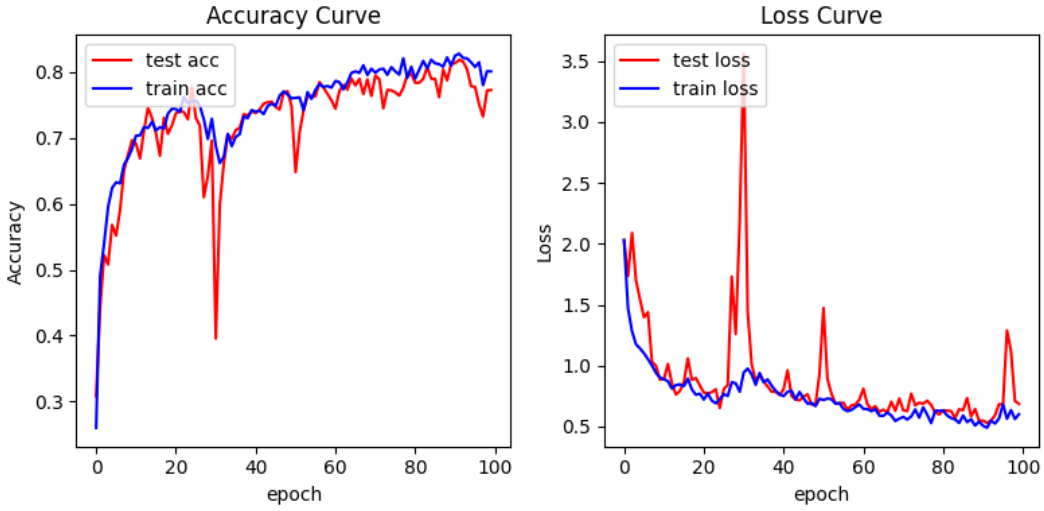

Figure 8: ResNet50* training on 1/16 dataset

(d) half data

- ResNet18*

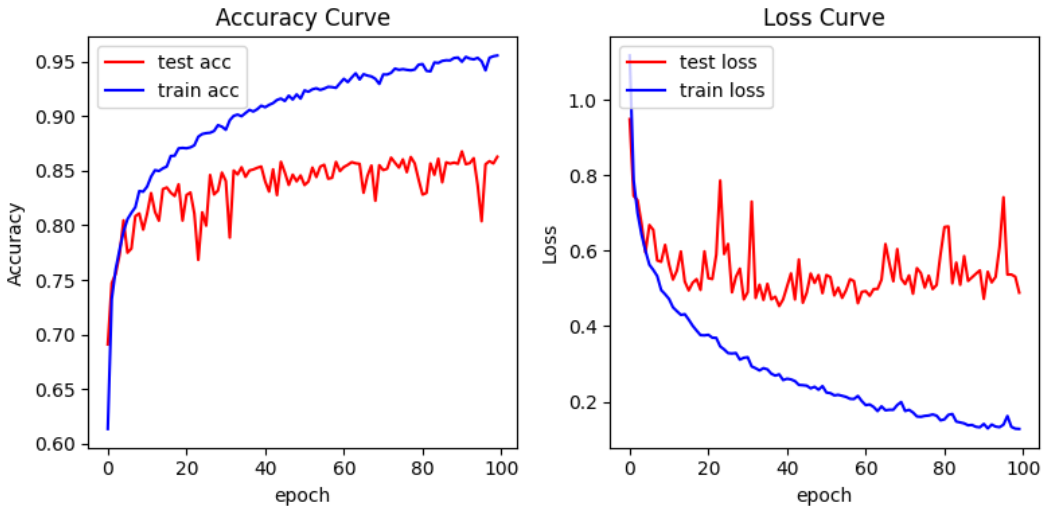

Figure 9: ResNet18* training on half dataset

- ResNet50*

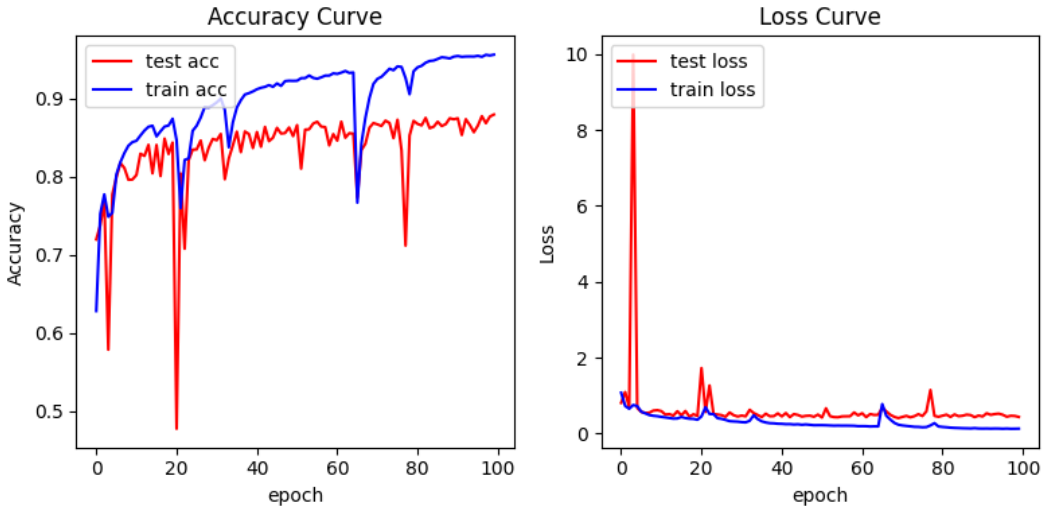

Figure 10: ResNet50* training on half dataset

(e) all data

- ResNet18*

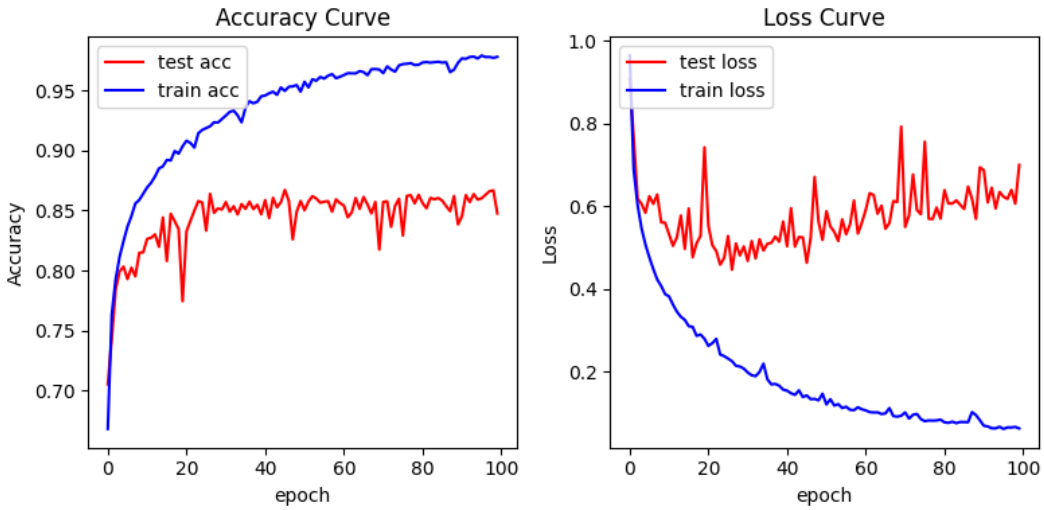

Figure 11: ResNet18* training on all dataset

- ResNet50*

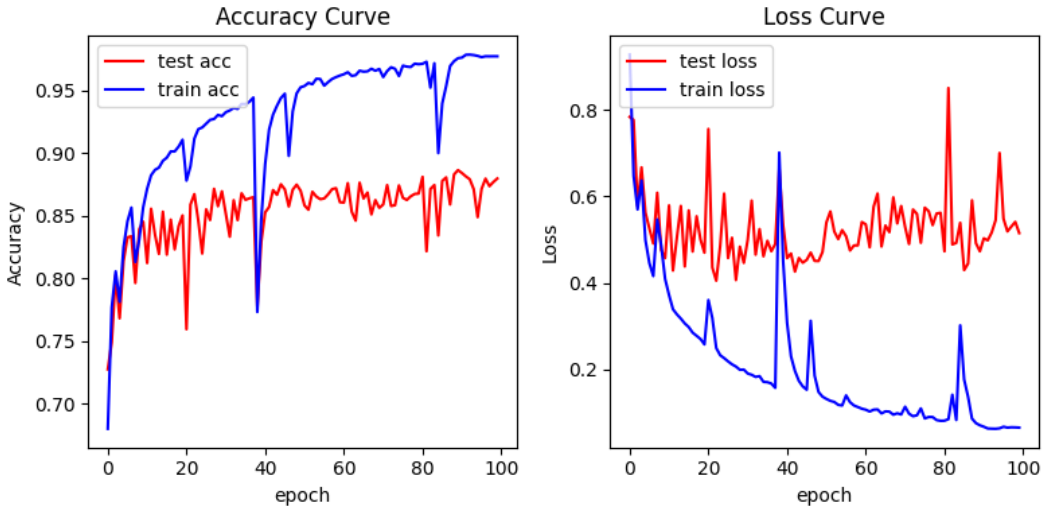

Figure 12: ResNet50* training on all dataset

3. Comparison for model size and dataset

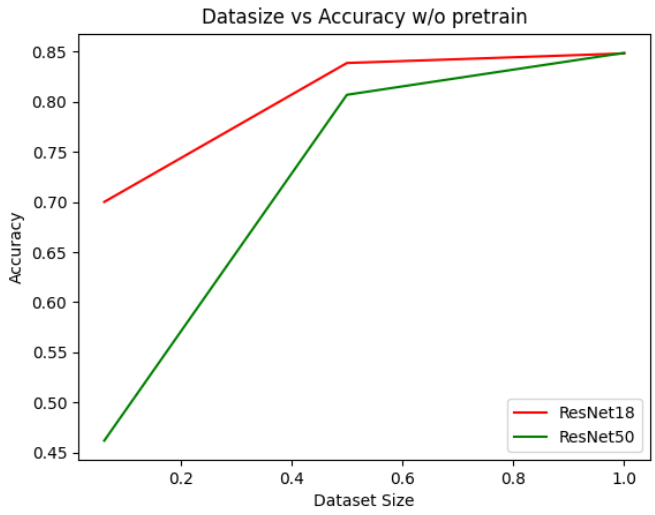

Figure 13: Comparison on models without initialized weights

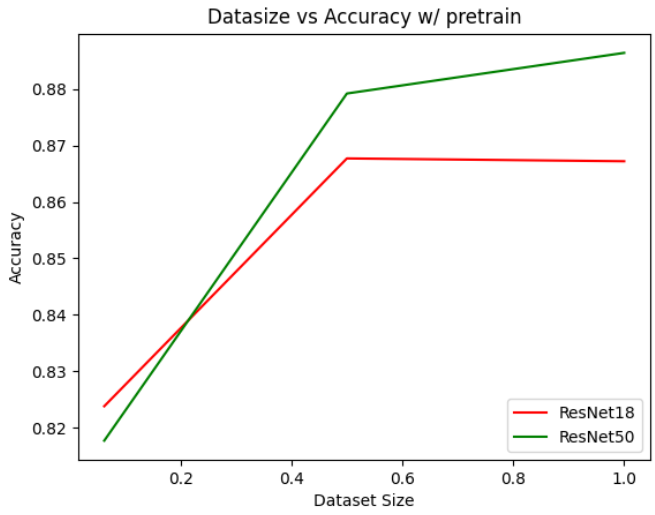

Figure 14: Comparison on models with initialized weights

From fig.13 and 14, we can see that big model (**ResNet50**) outperforms small model (**ResNet18**) when the size of training dataset is larger. The reason is that ResNet50 is a big model relative to ResNet18, so it tends to overfit on small dataset, which makes it perform worse when the size of dataset is only 1/16.

Observing at the point at datasize = 0.5, we can find that ResNet50 performs worse than ResNet18 when we don't apply pretrained weights. However, this condition becomes different when we apply pretrained weights. This might indicates that pretrained weights help ResNet more when the size of dataset is half.

4. Achieve the best performance

   I search for some state-of-the-art networks for CIFAR10 here and select **DenseNet121** as my backbone network, training it in 100 epochs with pretrained weight `IMAGENET1K_V1`. I also change the output size of DenseNet to 10 in order to perform classification on CIFAR10 as I previously did. The following shows the modified code section and the learning curves.

```
model = torch.hub.load('pytorch/vision:v0.10.0', 'densenet121',
                       weights='IMAGENET1K_V1')
model.classifier = nn.Linear(in_features=1024, out_features=10,
                             bias=True)
model = model.to(device)
```

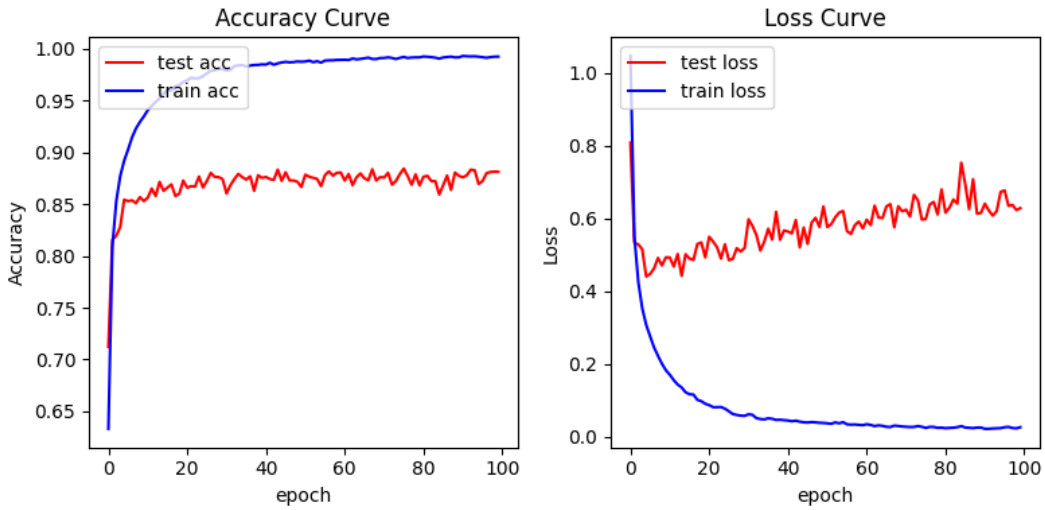

Figure 15: DenseNet* training on all dataset

Fig.16 and Table.1 compares the accuracy of DenseNet* with ResNet50* and ResNet18* at each epoch. We can see that **DenseNet121*** performs the best than the other backbones.

Model	Accuracy	Model	Accuracy
ResNet18	0.848	ResNet18*	0.867
ResNet50	0.849	ResNet50*	0.886
DenseNet121	-	**DenseNet121***	**0.893**

Table 1: Accuracy of different networks

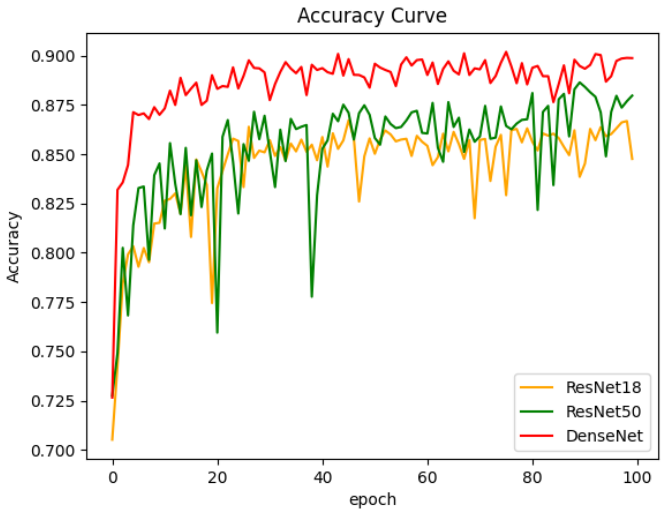

Figure 16: Accuracy curve on whole dataset during 100 epochs