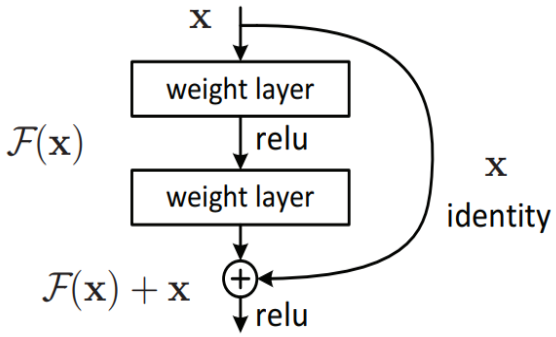
Deep Learning and Practice Lab 3

309554005 黃睿宇

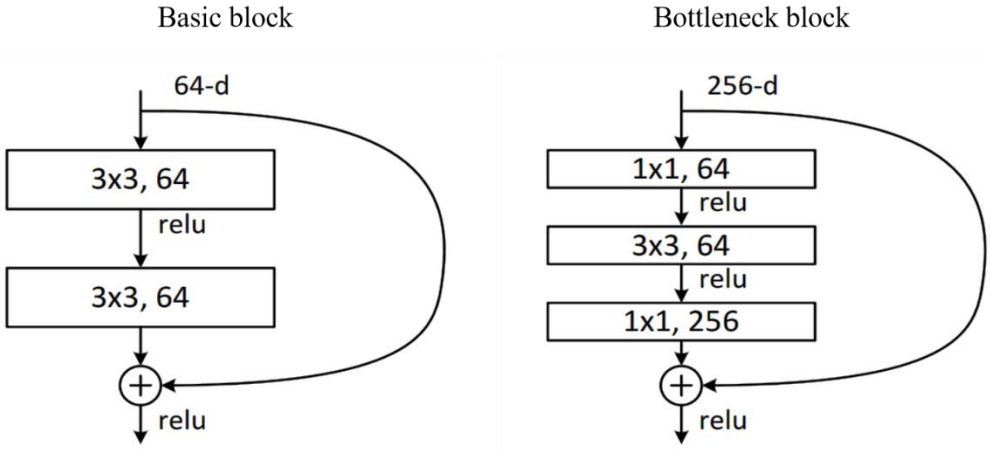
1 Introduction

Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people. In this lab, we implement ResNet, including ResNet18 and ResNet50, architecture to classify diabetic retinopathy grading and compare the performance between pre-trained model and non-pre-trained model in the same architecture.

ResNet (Residual Network) is the Winner of ILSVRC 2015 in image classification, detection, and localization, as well as Winner of MS COCO 2015 detection, and segmentation. To solve the problem of vanishing and exploding gradients, a skip connection is added to add the input *x* to the output after few weight layers as below.



A basic block for ResNet18 and a bottleneck block for ResNet50 is shown below.



The dataset is from diabetic retinopathy detection, providing with a large set of high-resolution retina images taken under a variety of imaging conditions. The dataset contains 35124 images, and we divide it into 28099 training data and 7025 testing

data. The image resolution is 512×512 and has been preprocessed.

2 Experiment Setups

2.1 ResNet

Our ResNet model refers to torchvision.models.resnet. In BasicBlock, the network is connected as x = (in, H, W) → conv2D → (out, H, W) → conv2D→ (out, H, W) + x.

```
class BasicBlock(nn.Module):
    def __init__(self, in_channel, out_channel, s):
        super(BasicBlock, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channel, out_channel, kernel_size=3,
                      stride=s, padding=1, bias=False),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(),
            nn.Conv2d(out_channel, out_channel, kernel_size=3,
                      stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(),
        )

        self.downsample = None

    def forward(self, x):
        identity = x
        out = self.block(x)
        if self.downsample is not None:
            identity = self.downsample(x)
        out += identity
        out = F.relu(out, inplace=True)

        return out
```

In Bottleneck, the network is connected as x = (in, H, W) → conv2D (1×1) → conv2D → (out, H, W) → conv2D (1×1)→ (out×4, H, W) + x.

```
class Bottleneck(nn.Module):
    def __init__(self, in_channel, out_channel, s):
        super(Bottleneck, self).__init__()
        self.expansion = 4
        self.block = nn.Sequential(
            nn.Conv2d(in_channel, out_channel, kernel_size=1,
                      stride=1, padding=0, bias=False),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(),
            nn.Conv2d(out_channel, out_channel, kernel_size=3,
                      stride=s, padding=1, bias=False),
            nn.BatchNorm2d(out_channel),
            nn.ReLU(),
            nn.Conv2d(out_channel, out_channel * self.expansion,
                      kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(out_channel * self.expansion),
            nn.ReLU(),
        )

        self.downsample = None

    def forward(self, x):
        identity = x
        out = self.block(x)
        if self.downsample is not None:
            identity = self.downsample(x)
        out += identity
        out = F.relu(out, inplace=True)

        return out
```

Then referring to [torchvision.models.resnet](), we could use BasicBlock and Bottleneck to build ResNet18 and ResNet50, and we could also use `torchvision` to make the pre-training version.

```python
class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()
        self.in_channel = 64
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, self.in_channel, kernel_size=7,
                      stride=2, padding=3, bias=False),
            nn.BatchNorm2d(self.in_channel),
            nn.ReLU(inplace=True),
        )
        self.conv2_0 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.conv2_1 = self.make_layer(64, 2, 1)
        self.conv3 = self.make_layer(128, 2, 2)
        self.conv4 = self.make_layer(256, 2, 2)
        self.conv5 = self.make_layer(512, 2, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.linear = nn.Sequential(
            nn.Linear(512, 5)
        )

    def make_layer(self, out_channel, blocks, stride):
        strides = [stride] + [1] * (blocks - 1)
        layers = []
        for s in strides:
            layers.append(BasicBlock(self.in_channel, out_channel, s))
            self.in_channel = out_channel

        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2_0(x)
        x = self.conv2_1(x)
        x = self.conv3(x)
        x = self.conv4(x)
```

```
        x = self.conv5(x)
        x = self.avg_pool(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)

        return x
```

```
class ResNet50(nn.Module):
    def __init__(self):
        super(ResNet50, self).__init__()
        self.expansion = 4
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,
                      bias=True),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
        )
        self.conv2_0 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.conv2_1 = self.make_layer(64, 64, 3, 1)
        self.conv3 = self.make_layer(64*self.expansion, 128, 4, 2)
        self.conv4 = self.make_layer(128*self.expansion, 256, 6, 2)
        self.conv5 = self.make_layer(256*self.expansion, 512, 3, 2)
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.linear = nn.Sequential(
            nn.Linear(512 * self.expansion, 5)
        )

    def make_layer(self, in_channel, out_channel, blocks, stride):
        strides = [stride] + [1] * (blocks - 1)
        layers = []
        for s in strides:
            layers.append(Bottleneck(in_channel, out_channel, s))
            in_channel = out_channel * self.expansion

        return nn.Sequential(*layers)
```

```
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2_0(x)
        x = self.conv2_1(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.avg_pool(x)
        x = x.view(x.size(0), -1)
        x = self.linear(x)

        return x
```

2.2 Dataloader

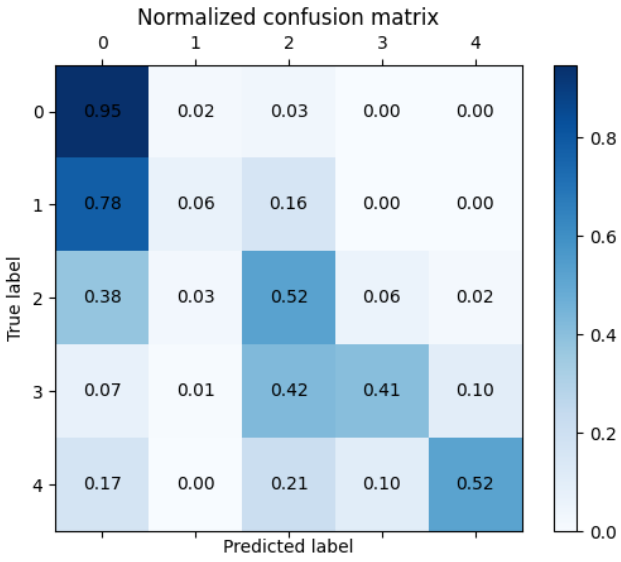
In training phase, we transform the .jpeg rgb images by flipping randomly, resizing the image shape from [H, W, C] (512×512×3) to [C, H, W] (3×512×512) via using `torchvision.transform.ToTensor`, and normalization. In testing phase, we only resize and normalize the data.

```
def __getitem__(self, index):
    path = self.root + self.img_name[index] + '.jpeg'
    img = Image.open(path)
    label = self.label[index]

    if self.mode == 'train':
        transform_method = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(
                mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
        ])
    elif self.mode == 'test':
        transform_method = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(
                mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
```

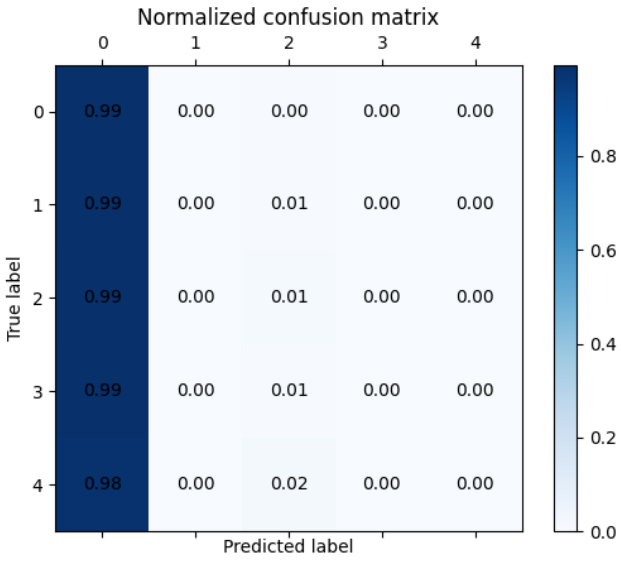
```
                mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
        ])

    img = transform(img)

    return img, label
```

2.3 Confusion matrix

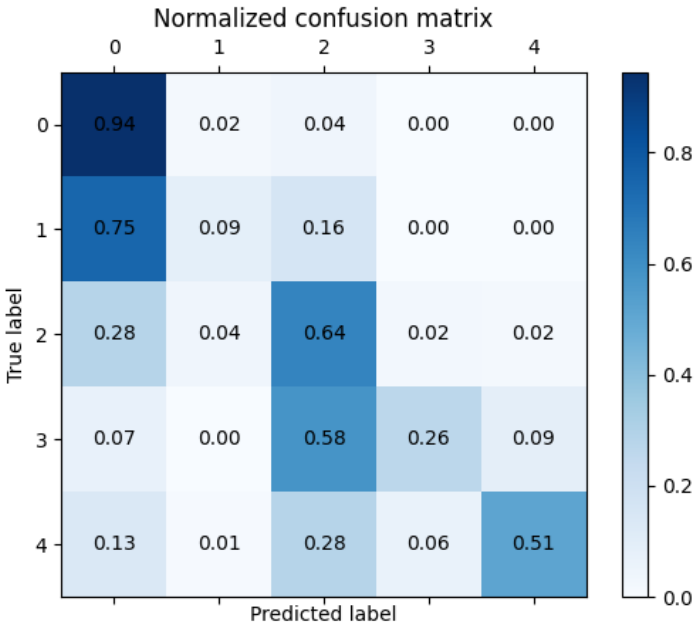
- ResNet18 with pre-training



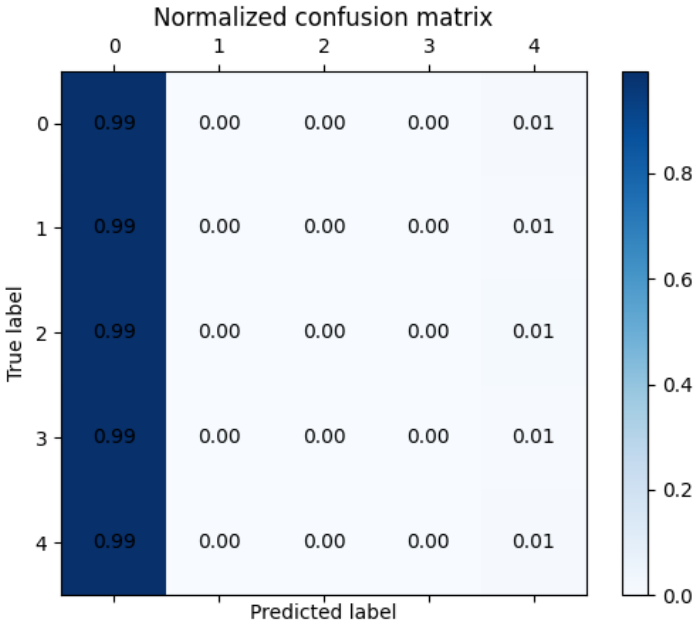
- ResNet18 without pre-training



- ResNet50 with pre-training



- ResNet50 without pre-training



In confusion matrix of pre-trained model, we could see that the accuracy of class 0 is well. However, the data in class 1 seems to have a high probability of being predicted as class 0. It is likely that there are no strong features existing between class 0 and class 1. Still, class 3 has a certain probability of being predicted incorrectly as class 2 as well.

3 Experimental Results

3.1 The highest testing accuracy

The highest testing accuracy are trained on following parameters:

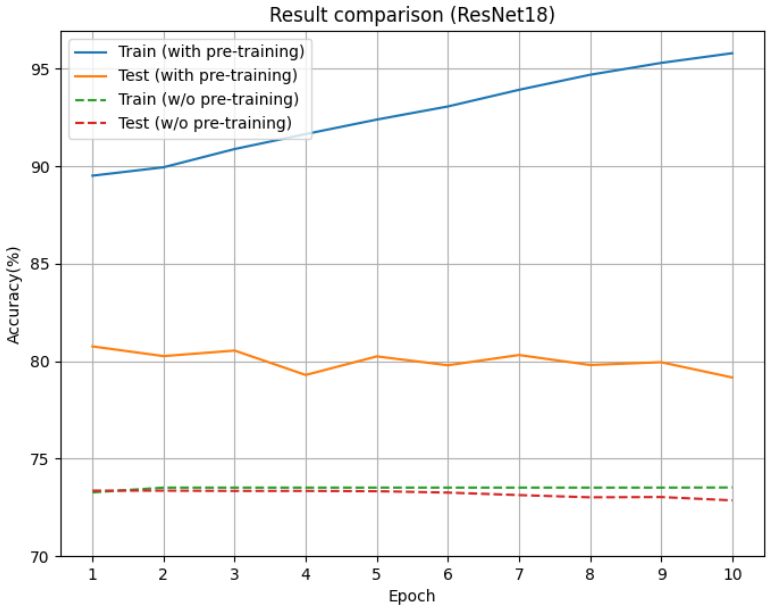
- optimizer: SGD
- loss function: cross entropy
- epochs for ResNet18 = 10
- epochs for ResNet50 = 5
- batch size = 8
- learning rate = 1e-4
- momentum = 0.9
- weight_decay = 5e-3

ResNet18 with pre-training	ResNet18 without pre-training	ResNet50 with pre-training	ResNet50 without pre-training
80.75%	73.35%	82.02%	73.32%

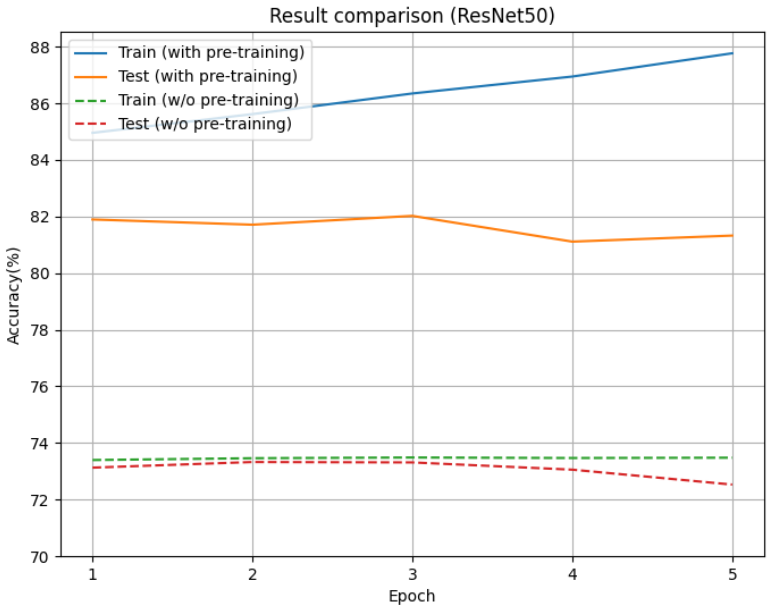
```
Using device:  cuda
> Found 28099 images...
> Found 7025 images...
/home/ubuntu/anaconda3/envs/pytorch/lib/python3.8/site-packages/torch/nn/functional.
py:718: UserWarning: Named tensors and all their associated APIs are an experimental
 feature and subject to change. Please do not use them for anything important until
they are released as stable. (Triggered internally at  /opt/conda/conda-bld/pytorch_
1623448234945/work/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)

ResNet18 with pre-training
epoch 1   Train accuracy: 0.895156 | Test accuracy: 0.807544
epoch 2   Train accuracy: 0.899463 | Test accuracy: 0.802562
epoch 3   Train accuracy: 0.908822 | Test accuracy: 0.805409
epoch 4   Train accuracy: 0.916509 | Test accuracy: 0.792883
epoch 5   Train accuracy: 0.923947 | Test accuracy: 0.802420
epoch 6   Train accuracy: 0.930674 | Test accuracy: 0.797865
epoch 7   Train accuracy: 0.939215 | Test accuracy: 0.803132
epoch 8   Train accuracy: 0.946938 | Test accuracy: 0.798007
epoch 9   Train accuracy: 0.953023 | Test accuracy: 0.799431
epoch 10  Train accuracy: 0.958006 | Test accuracy: 0.791601
Highest testing accuracy: 0.807544

ResNet18 without pre-training
epoch 1   Train accuracy: 0.732635 | Test accuracy: 0.733523
epoch 2   Train accuracy: 0.735080 | Test accuracy: 0.733523
epoch 3   Train accuracy: 0.735080 | Test accuracy: 0.733381
epoch 4   Train accuracy: 0.735080 | Test accuracy: 0.733381
epoch 5   Train accuracy: 0.735080 | Test accuracy: 0.733238
epoch 6   Train accuracy: 0.735080 | Test accuracy: 0.732527
epoch 7   Train accuracy: 0.735080 | Test accuracy: 0.731246
epoch 8   Train accuracy: 0.735080 | Test accuracy: 0.730107
epoch 9   Train accuracy: 0.735080 | Test accuracy: 0.730249
epoch 10  Train accuracy: 0.735151 | Test accuracy: 0.728541
Highest testing accuracy: 0.733523

ResNet50 with pre-training
epoch 1   Train accuracy: 0.849532 | Test accuracy: 0.818932
epoch 2   Train accuracy: 0.856151 | Test accuracy: 0.817082
epoch 3   Train accuracy: 0.863447 | Test accuracy: 0.820198
epoch 4   Train accuracy: 0.869426 | Test accuracy: 0.811103
epoch 5   Train accuracy: 0.877647 | Test accuracy: 0.813238
Highest testing accuracy: 0.820198

ResNet50 without pre-training
epoch 1   Train accuracy: 0.733941 | Test accuracy: 0.731246
epoch 2   Train accuracy: 0.734581 | Test accuracy: 0.733238
epoch 3   Train accuracy: 0.734830 | Test accuracy: 0.733096
epoch 4   Train accuracy: 0.734652 | Test accuracy: 0.730534
epoch 5   Train accuracy: 0.734759 | Test accuracy: 0.725267
Highest testing accuracy: 0.733238
```

3.2 Comparison figures

- ResNet18



- ResNet50

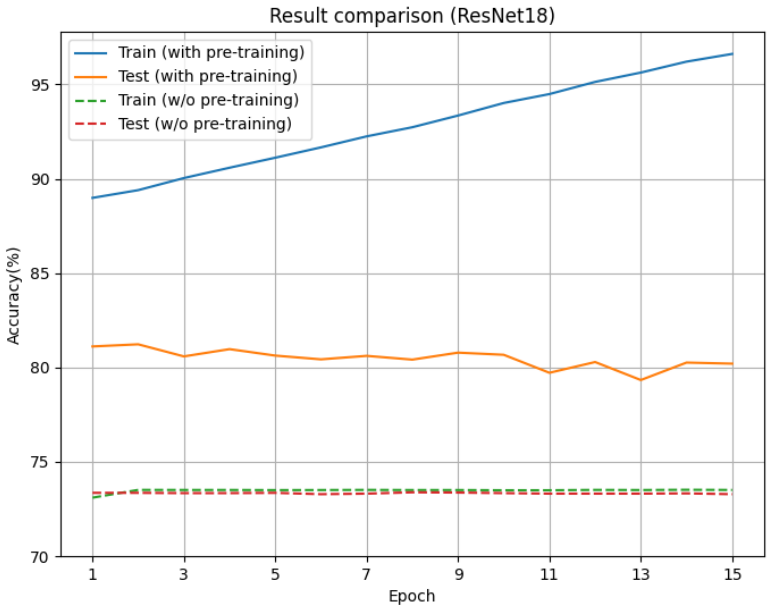


In comparison result, we could see that the accuracy of model without pre-training will not increase either in training data or testing data. The accuracy of model with pre-training grows up in training data, while it declines in testing data. It is likely that there is overfitting in pre-trained model.

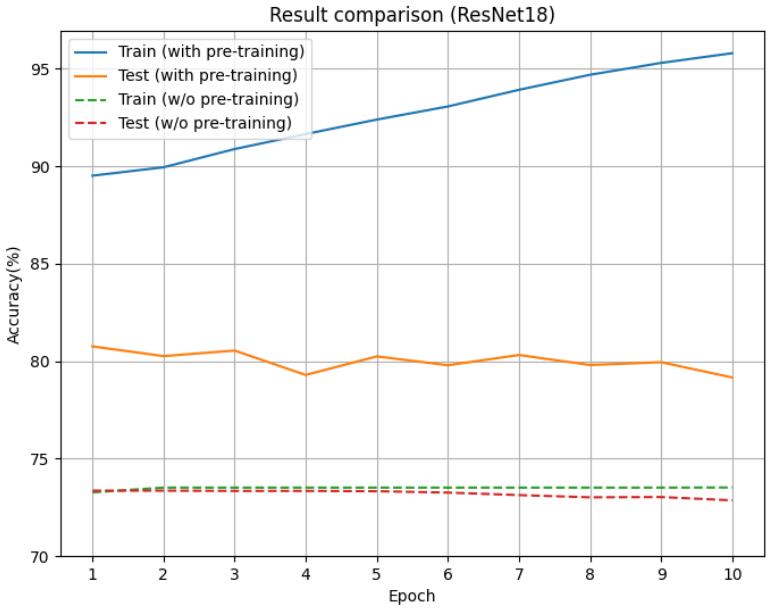
4 Discussion

To make model without pre-training trainable, we set epoch size and batch size more than before. However, the result is not better than the original one. And still, we could find that the pre-trained model seems to be overfitting as well since the accuracy in training phase increases while that in testing phase decreases. We guess that it is likely that there are some features existing only in testing dataset.

- ResNet18: batch size = 16, epochs = 15



- ResNet18: batch size = 8, epochs = 10


Result comparison (ResNet18)
Train (with pre-training)
Test (with pre-training)
Train (w/o pre-training)
Test (w/o pre-training)
Accuracy(%)
Epoch
95
90
85
80
75
70
1
2
3
4
5
6
7
8
9
10