0860832 - Thong The Nguyen (阮世聰)

Lab 03 Report

1. Introduction

In this lab, I will implement the ResNet18, ResNet50, with and without pretrained weights to analyze diabetic retinopathy by using the Pytorch framework. I do the lab in the following steps:

- Write my own DataLoader
- Use ResNet (loaded from Pytorch) to classify images.
- Calculate and visualize the confusion matrix by sklearn.

The result is not very good, although I tried to use augmented data to issue the problem of imbalanced dataset, the accuracy is not better than guessing that all the images belong to class 0. The highest accuracy is around 74.8185% for ResNet50 with pretrained.

2. Experimental Setup

A. The details of model

```
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)
    )
  )
```

Resnet18 with Basic Block repeated

```
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=5, bias=True)
```

Last layer of ResNet18

```
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): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=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)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
```

Resnet50 with Bottleneck Block repeated

```
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=5, bias=True)
```

Last layer of ResNet50

```
""" Get resnet model from torchvision """
resnet18 = models.resnet18()
resnet50 = models.resnet50()
resnet18_pretrained = models.resnet18(pretrained=True)
resnet50_pretrained = models.resnet50(pretrained=True)

""" Only update gradient for last layer """
freeze_model(resnet18_pretrained)
freeze_model(resnet50_pretrained)

""" replace the last layer of resnet """
resnet18.fc = nn.Linear(512, 5)
resnet50.fc = nn.Linear(2048, 5)
resnet18_pretrained.fc = nn.Linear(512, 5)
resnet50_pretrained.fc = nn.Linear(2048, 5)
```

Code for creating models

B. The details of your Data Loader

After taking a quick look at the data, I notice that the data is imbalanced, the frequencies for each class is : *[20655, 1955, 4210, 698, 581]* in training set (the same proportions in testing set), so I need to use various techniques to of Pytorch data loader to augment the dataset for only classes that have too little of data sample.

```
class RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode, transforms=None):
        self.root = root
        self.img_name, self.label = getData(mode)
        self.mode = mode
        self.transforms = transforms
        self.label0_transforms = tfs.Compose([tfs.ToTensor()])

        print("> Found %d images..." % (len(self.img_name)))
```

I changed the *__init__* function of RetinopathyLoader to have 2 more items:

- *self.label0_transforms:* transforms for class 0 only.
- *self.transforms*: transforms for images of their class.

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

    """ Transform image """
    if self.transforms is not None and label != 0:
        img = self.transforms(img)
    else:
        img = self.label0_transforms(img)

    return img, label
```

I implemented the *__getitem__* function, that returns a pair of data points in the dataset. I used PIL library for reading image and convert to numpy array, then use various transforms for images that is not class 0, and normalize pixel value to [0, 1] and transpose the image shape from [H, W, C] to [C, H, W].

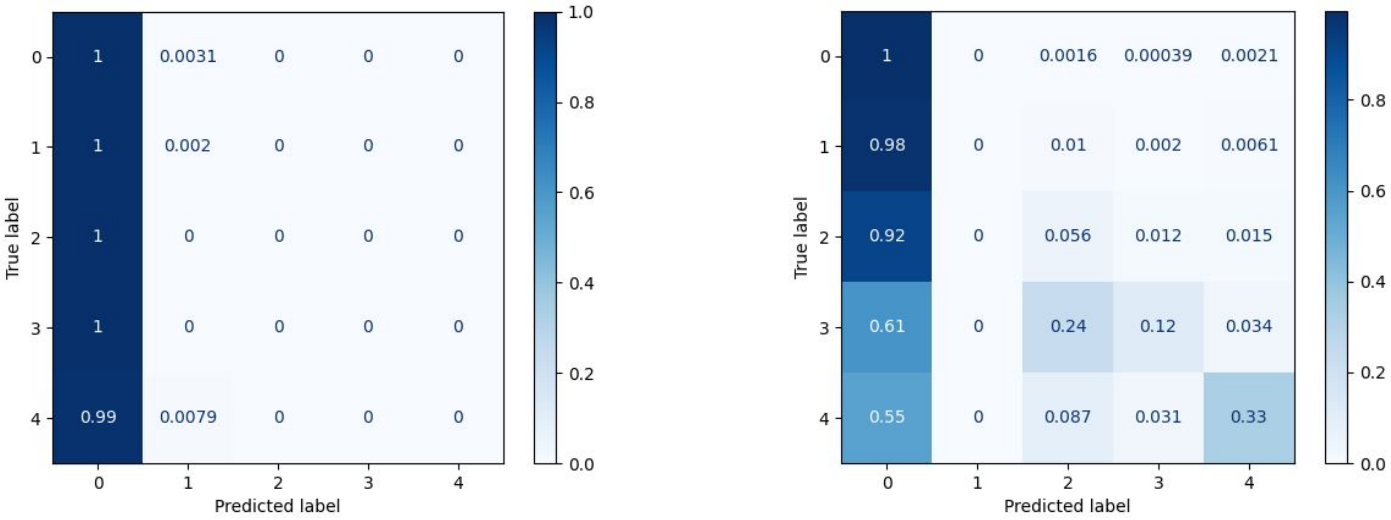
```
""" Load data """
trainset = RetinopathyLoader('data/', 'train', transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.RandomRotation(65),
    transforms.ToTensor()
]))
testset = RetinopathyLoader('data/', 'test', transforms.Compose([
    transforms.ToTensor()
]))
```

This is how the code looks when creating dataset and then creating data loader of Pytorch.

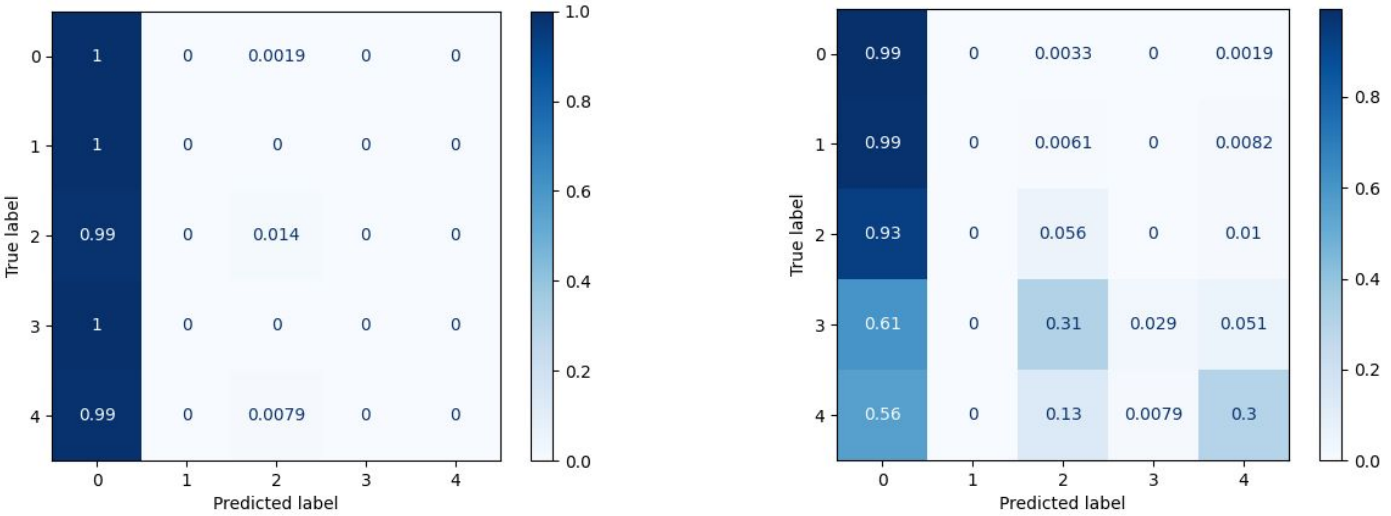
```
class_weights = 1. / torch.tensor([20655, 1955, 4210, 698, 581], dtype=torch.float)
point_weights = [class_weights[out] for inp, out in trainset]
sampler = WeightedRandomSampler(
    weights=point_weights, num_samples=len(point_weights), replacement=True
)

trainloader = DataLoader(trainset, batch_size=4, sampler=sampler)
testloader = DataLoader(testset, batch_size=4, shuffle=True)
```

I also created a sampler to draw batch data that is balanced with respect to the frequency of image in each class. The weights to choose an image depends on its class, so images with class 0 have a smaller probability to be selected than others.

C. Describing your evaluation through the confusion matrix



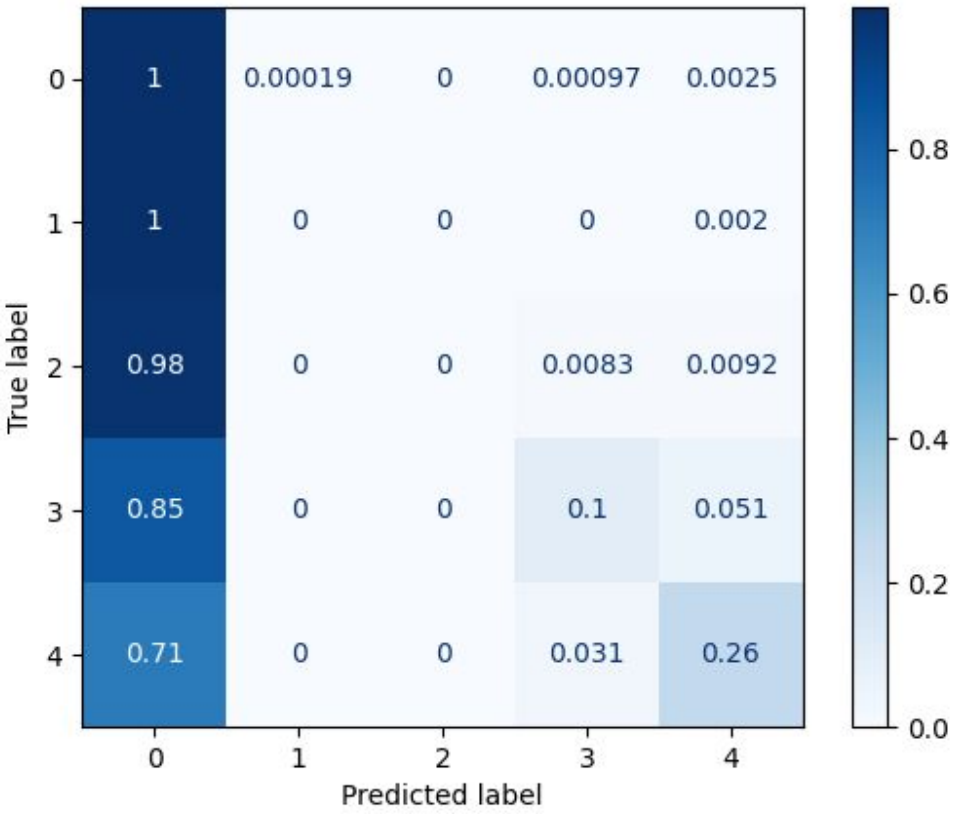
Confusion matrix of ResNet50 without pretrained (left) and pretrained (right) on dataset without augmentation



Confusion matrix of ResNet18 without pretrained (left) and pretrained (right) on dataset without augmentation

Most of the images in the testing set belong to class 0, so the cell (0, 0) is nearly 1. While the version without pretrained weights always predicts 0, the pretrained

network somehow recognizes images of classes other than 0, but it still does not recognize it well enough.



Confusion matrix of ResNet18 without pretrained dataset augmentation

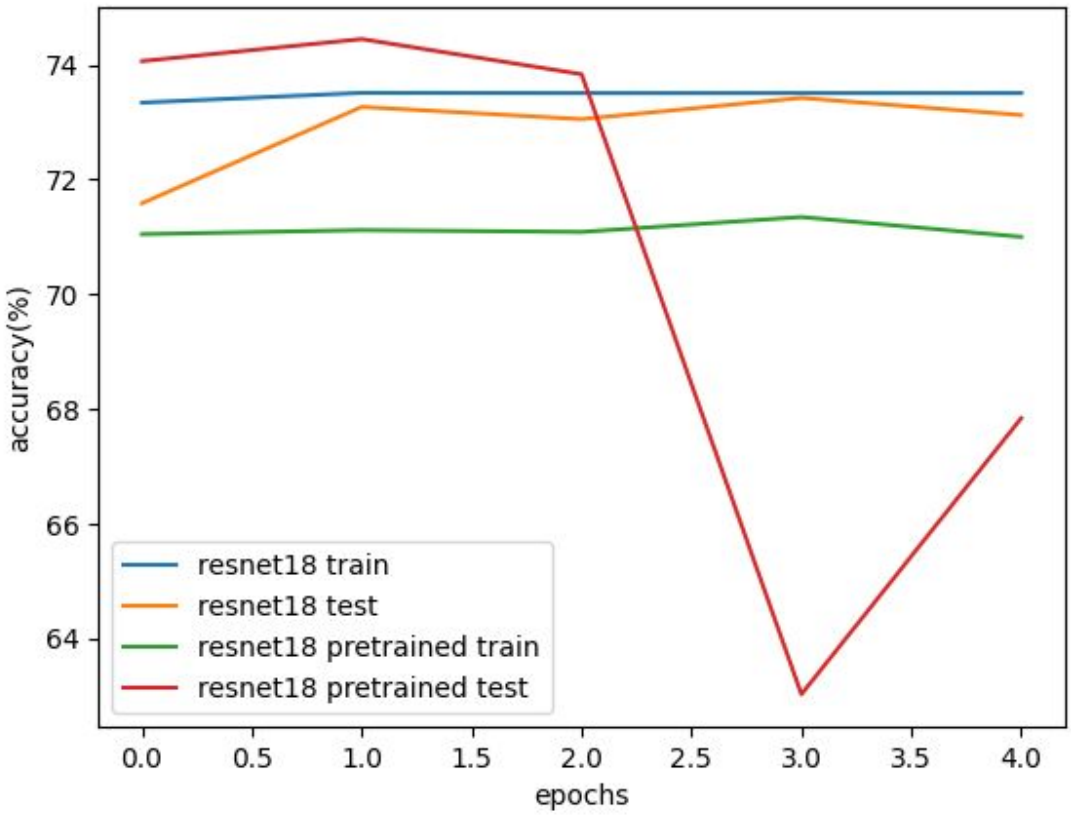
After augmenting data, the result does not improve much, it even performs worse in class 3 and 4, still does not recognize class 1 and 2 correctly and the false negative with class 0 is higher.

3. Experimental Result

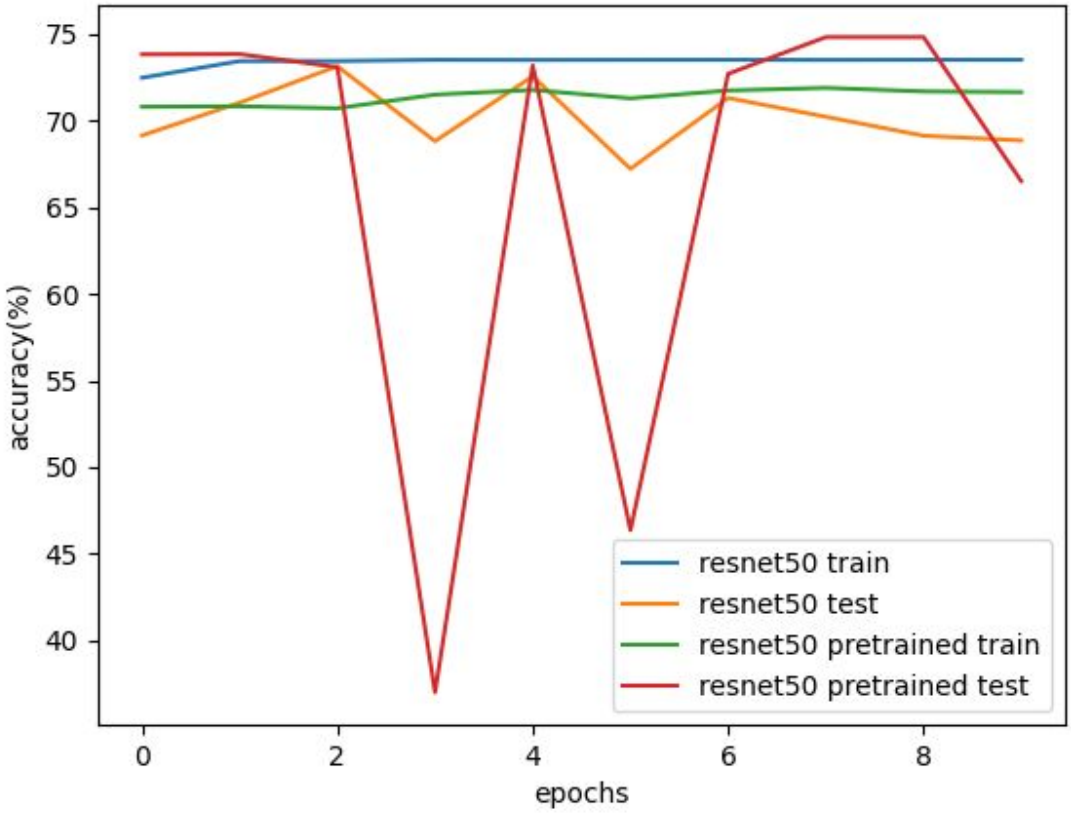
The highest testing accuracy

ResNet18 without pretrained	73.8078%
ResNet18 pretrained	73.3665%
ResNet50 without pretrained	73.1388%
ResNet50 pretrained	74.8185%

Comparison figures



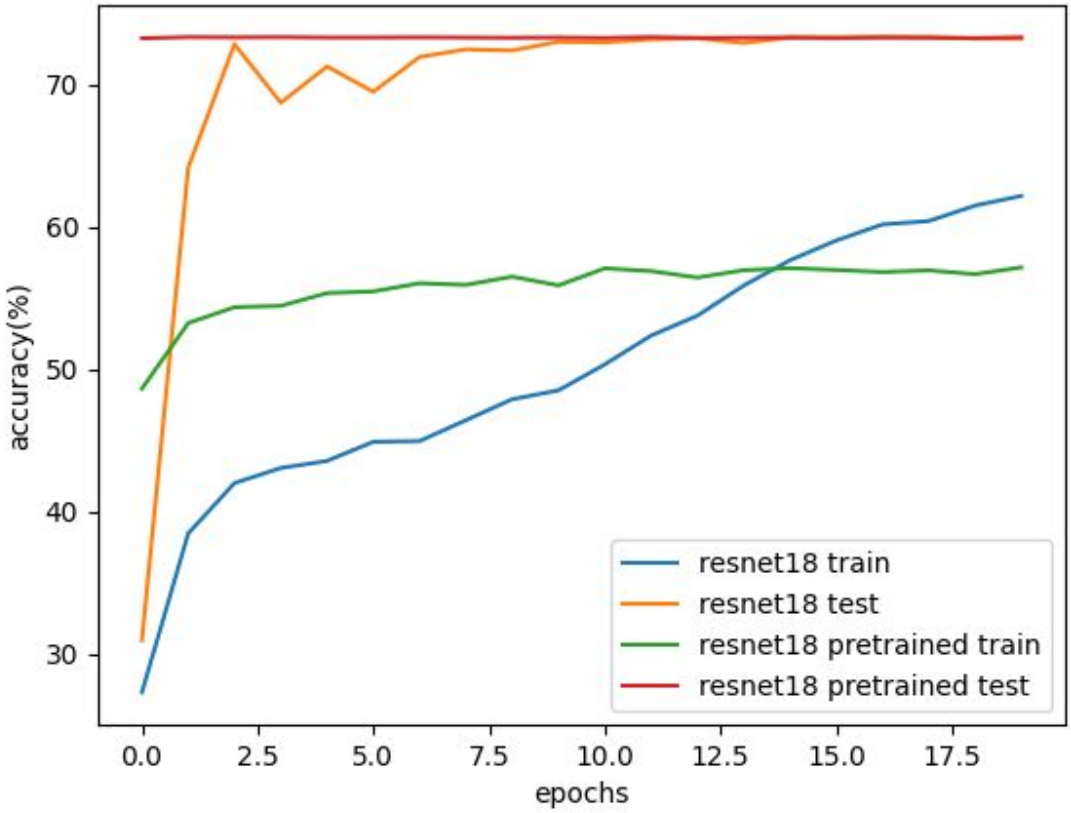
Accuracy of ResNet18



Accuracy of ResNet50

This is the accuracy before data augmentation, both 2 versions of Resnet with pretrained weights achieve lower accuracy of the dataset. The highest testing result was around 75%. Because the data is imbalanced, the models result in almost 0 for every image but still achieve high accuracy on the test set.

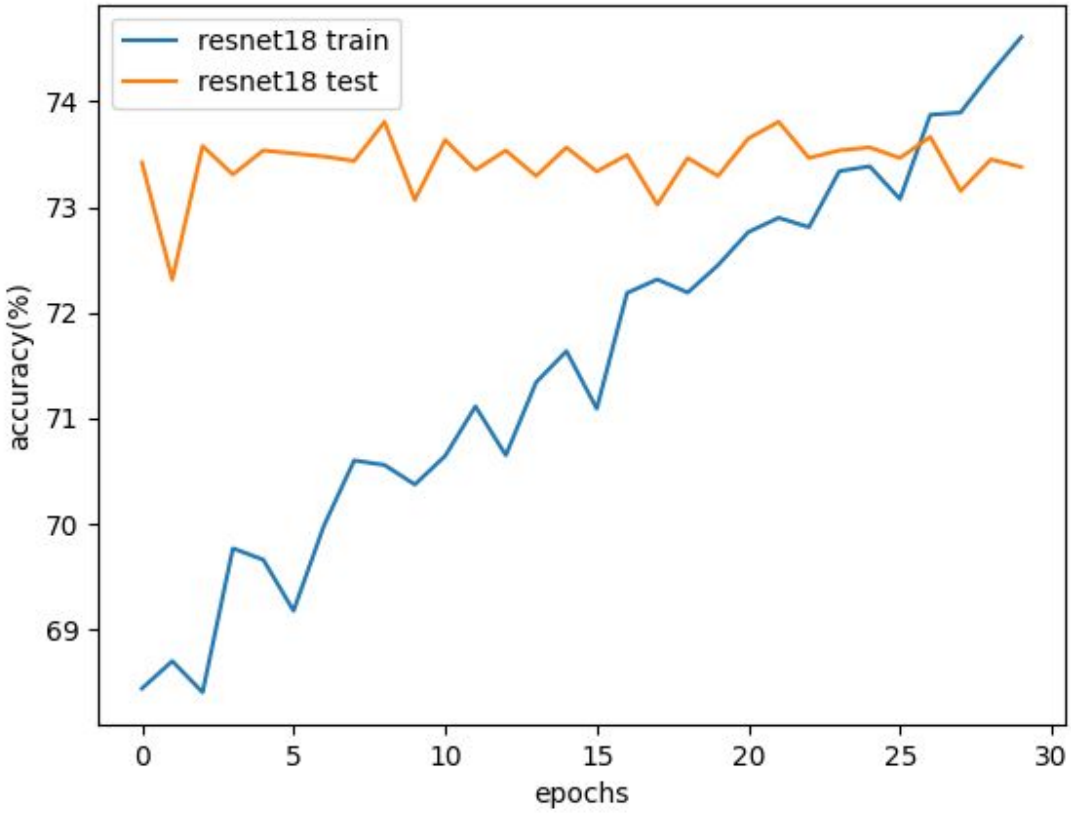
The pretrained version seems to be more unstable than the version without pretrained weights.



With resnet18 (without pretrained) after data augmentation, with learning rate 0.0001, batch size is 4, after 20 epochs, the training accuracy is around 62%, while the test accuracy is around 73%. The accuracy of testing is higher than the training set because the data in the test set is also imbalanced (mostly of class 0), and the model output almost 0 everywhere.

For pretrained resnet18, the situation is worse, the training accuracy does not increase much after 20 epochs.

Based on the training result of resnet18, we can still expect the model to learn more to achieve higher training accuracy, then increasing testing accuracy too. So that I train the resnet18 with 30 epochs to see what happens.



The next 30 epochs increase training accuracy but the testing accuracy does not increase, maybe the network can not generalize anymore.

It took too much time to train a ResNet50, so I don't have time to experiment with how resnet50 performs on an augmented dataset.

4. Discussion

It took so much time to train a ResNet on a personal computer.

With an imbalanced dataset, we can try various ways to augment the dataset by random cropping, flipping, rotation. We can also sample the data belonging to the minority class more than normal.