1. Introduction

ResNet相比於一般的深層網路多了一條跨越多層Conv的路線，並與通過多層Conv的數值加起來，這樣的好處是可以克服隨著層數增加，梯度消失或是變小，導致參數更新緩慢的問題。這次的作業使用ResNet18跟ResNet50的網路架構，並比較使用pretrained weights再finetune跟沒有使用pretrained weights就直接訓練的結果。

2. Experiment setups

the details of my 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)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

(resnet18局部)

不管是哪一種resnetXX，整體的架構都可以分成三部分:

1. Input layer: 使用一般Conv，並用大型stride降低解析度(7,7)
2. Middle layers: 共有4曾layer，resnet18一層layer裡有兩個basic block，resnet50則有三個basic block，大部分basic block都會降低解析度，增加output channel
3. Output layer: 用 fully connected layer

the details of my dataloader:

```python
class RetinopathyDataSet(Dataset):
    def __init__(self, img_path, mode):
        self.img_path = img_path
        self.mode = mode

        self.img_names=np.squeeze(pd.read_csv('train_img.csv' if mode=='train' else 'test_img.csv').values)
        self.labels=np.squeeze(pd.read_csv('train_label.csv' if mode=='train' else 'test_label.csv').values)
        assert len(self.img_names)==len(self.labels),'the amount of imga and labels are not the same'
        self.data_len=len(self.img_names)

        self.transformations=transforms.Compose([transforms.RandomHorizontalFlip(p=0.5),transforms.RandomVerticalFlip(p=0.5),transforms.ToTensor(),
        # self.transformations=transforms.Compose([transforms.Resize(256), transforms.RandomHorizontalFlip(p=0.5),transforms.RandomVerticalFlip(p=0

        #,transforms.Normalize((0.3749, 0.2602, 0.1857),(0.2526, 0.1780, 0.1291))])

        print(f'>> Found {self.data_len} images...')



    def __len__(self):
        return self.data_len

    def __getitem__(self, index):
        single_img_name=os.path.join(self.img_path,self.img_names[index]+'.jpeg')
        single_img=Image.open(single_img_name)  # read an PIL image
        img=self.transformations(single_img)

        label=self.labels[index]

        return img, label
```

```python
dataset_train=RetinopathyDataSet(img_path='data',mode='train')
loader_train=DataLoader(dataset=dataset_train,batch_size=batch_size,shuffle=True)#,num_workers=4)

dataset_test=RetinopathyDataSet(img_path='data',mode='test')
loader_test=DataLoader(dataset=dataset_test,batch_size=batch_size,shuffle=False)#,num_workers=4)
```

我的dataloader code會先打開csv檔讀取裡面的img_name後存成numpy array，再根據numpy array一一讀取圖片，做transformations，分batch再丟入model

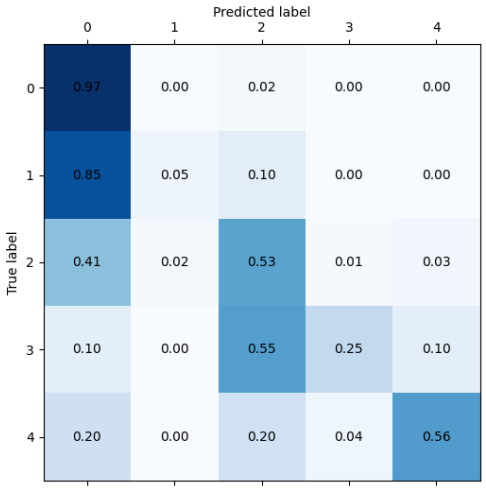
Describing my evaluation through the confusion matrix:

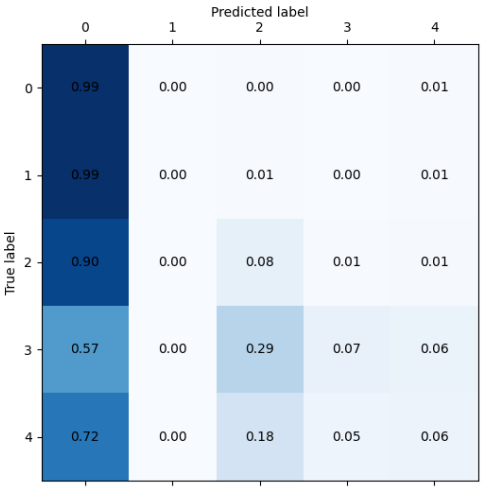
因為數據的分布不均，70%左右的圖片都是label 0的集合，因此在沒有pretrained weights時model趨向將所有圖片都預測成label 0。

而pretrained的model因為在訓練時接受的數據分布會比現在的數據還均勻，因此預測的結果相比於前項好上很多。

left: resnet18 with pretrained weights

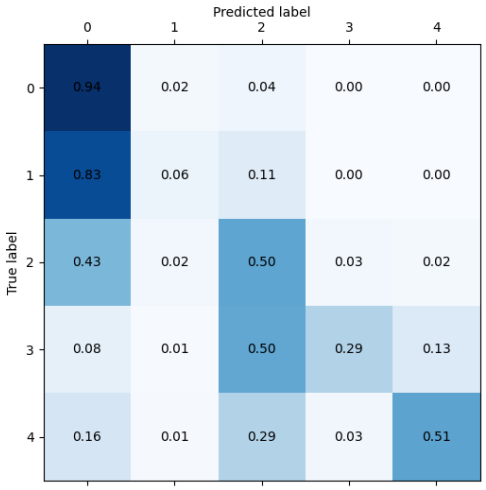
right: resnet18 without pretrained weights

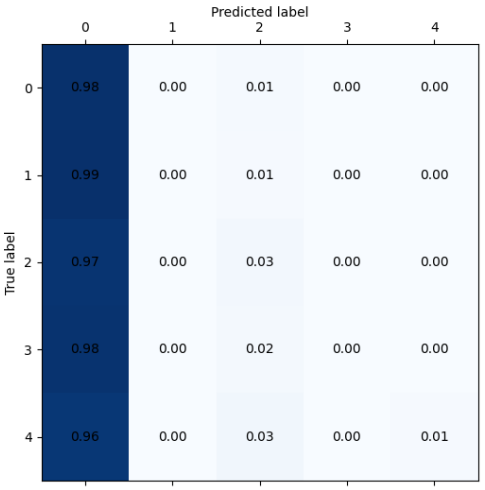




left: resnet50 with pretrained weights

right: resnet50 without pretrained weights





3. Experimental results
The highest testing accuracy:

	resnet18	resnet50
with pretrained	81.67	80.37
w/o pretrained	73.92	73.22

Screenshot:

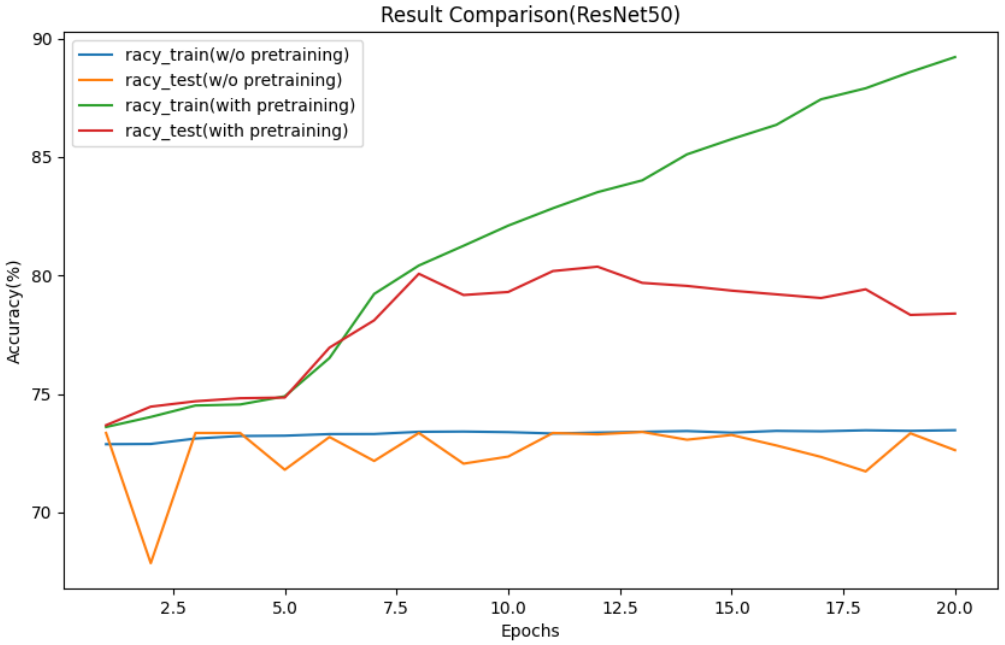
```
resnet18 wo pretrained weights
epoch 1 loss:0.1118 train_accuracy:73.24%
test_accuracy:72.94%
epoch 2 loss:0.1088 train_accuracy:73.48%
test_accuracy:73.28%
epoch 3 loss:0.1081 train_accuracy:73.44%
test_accuracy:73.34%
epoch 4 loss:0.1077 train_accuracy:73.50%
test_accuracy:72.27%
epoch 5 loss:0.1073 train_accuracy:73.50%
test_accuracy:72.61%
epoch 6 loss:0.1070 train_accuracy:73.51%
test_accuracy:71.15%
epoch 7 loss:0.1068 train_accuracy:73.50%
test_accuracy:72.57%
epoch 8 loss:0.1065 train_accuracy:73.51%
test_accuracy:72.48%
epoch 9 loss:0.1065 train_accuracy:73.50%
test_accuracy:72.16%
epoch10 loss:0.1063 train_accuracy:73.50%
test_accuracy:72.53%
epoch11 loss:0.1063 train_accuracy:73.51%
test_accuracy:72.71%
epoch12 loss:0.1060 train_accuracy:73.51%
test_accuracy:72.80%
epoch13 loss:0.1061 train_accuracy:73.51%
test_accuracy:72.50%
epoch14 loss:0.1059 train_accuracy:73.52%
test_accuracy:72.19%
epoch15 loss:0.1058 train_accuracy:73.50%
test_accuracy:72.90%
epoch16 loss:0.1057 train_accuracy:73.49%
test_accuracy:72.91%
epoch17 loss:0.1048 train_accuracy:73.53%
test_accuracy:73.45%
epoch18 loss:0.1031 train_accuracy:73.52%
test_accuracy:73.40%
epoch19 loss:0.1003 train_accuracy:73.83%
test_accuracy:73.49%
epoch20 loss:0.0980 train_accuracy:74.25%
test_accuracy:73.92%
```

```
resnet18 with pretrained weights
epoch 1 loss:0.1063 train_accuracy:72.98%
test_accuracy:74.01%
epoch 2 loss:0.1031 train_accuracy:73.02%
test_accuracy:74.28%
epoch 3 loss:0.1031 train_accuracy:72.98%
test_accuracy:72.41%
epoch 4 loss:0.1011 train_accuracy:73.45%
test_accuracy:74.25%
epoch 5 loss:0.1011 train_accuracy:73.56%
test_accuracy:74.25%
epoch 1 loss:0.0941 train_accuracy:75.55%
test_accuracy:78.46%
epoch 2 loss:0.0807 train_accuracy:78.89%
test_accuracy:79.46%
epoch 3 loss:0.0759 train_accuracy:80.30%
test_accuracy:79.83%
epoch 4 loss:0.0730 train_accuracy:80.84%
test_accuracy:78.90%
epoch 5 loss:0.0708 train_accuracy:81.20%
test_accuracy:78.46%
epoch 6 loss:0.0689 train_accuracy:81.98%
test_accuracy:80.98%
epoch 7 loss:0.0672 train_accuracy:82.18%
test_accuracy:81.01%
epoch 8 loss:0.0656 train_accuracy:82.59%
test_accuracy:80.56%
epoch 9 loss:0.0646 train_accuracy:82.85%
test_accuracy:80.85%
epoch10 loss:0.0630 train_accuracy:83.29%
test_accuracy:81.38%
epoch11 loss:0.0617 train_accuracy:83.60%
test_accuracy:79.72%
epoch12 loss:0.0606 train_accuracy:83.93%
test_accuracy:80.88%
epoch13 loss:0.0597 train_accuracy:83.94%
test_accuracy:80.94%
epoch14 loss:0.0584 train_accuracy:84.15%
test_accuracy:81.67%
epoch15 loss:0.0574 train_accuracy:84.47%
test_accuracy:81.58%
```

```
resnet50 wo pretrained weights
epoch 1 loss:0.1176 train_accuracy:72.40%
test_accuracy:72.80%
epoch 2 loss:0.1110 train_accuracy:73.31%
test_accuracy:72.87%
epoch 3 loss:0.1099 train_accuracy:73.33%
test_accuracy:71.07%
epoch 4 loss:0.1093 train_accuracy:73.45%
test_accuracy:73.31%
epoch 5 loss:0.1088 train_accuracy:73.44%
test_accuracy:70.56%
epoch 6 loss:0.1081 train_accuracy:73.47%
test_accuracy:72.19%
epoch 7 loss:0.1079 train_accuracy:73.49%
test_accuracy:72.40%
epoch 8 loss:0.1074 train_accuracy:73.51%
test_accuracy:72.04%
epoch 9 loss:0.1075 train_accuracy:73.50%
test_accuracy:72.21%
epoch10 loss:0.1072 train_accuracy:73.48%
test_accuracy:73.22%
epoch11 loss:0.1070 train_accuracy:73.50%
test_accuracy:72.94%
epoch12 loss:0.1066 train_accuracy:73.50%
test_accuracy:73.08%
epoch13 loss:0.1068 train_accuracy:73.50%
test_accuracy:73.20%
epoch14 loss:0.1065 train_accuracy:73.51%
test_accuracy:71.59%
epoch15 loss:0.1064 train_accuracy:73.51%
test_accuracy:70.23%
epoch16 loss:0.1064 train_accuracy:73.51%
test_accuracy:71.93%
epoch17 loss:0.1064 train_accuracy:73.50%
test_accuracy:70.23%
```

```
resnet50 with pretrained weights
Downloading: "https://download.pytorch.org/
eckpoints/resnet50-19c8e357.pth
100%|
epoch 1 loss:0.0255 train_accuracy:73.60%
test_accuracy:73.68%
epoch 2 loss:0.0243 train_accuracy:74.03%
test_accuracy:74.46%
epoch 3 loss:0.0239 train_accuracy:74.51%
test_accuracy:74.69%
epoch 4 loss:0.0236 train_accuracy:74.55%
test_accuracy:74.82%
epoch 5 loss:0.0235 train_accuracy:74.90%
test_accuracy:74.85%
epoch 1 loss:0.0219 train_accuracy:76.52%
test_accuracy:76.95%
epoch 2 loss:0.0195 train_accuracy:79.22%
test_accuracy:78.11%
epoch 3 loss:0.0183 train_accuracy:80.42%
test_accuracy:80.07%
epoch 4 loss:0.0174 train_accuracy:81.25%
test_accuracy:79.17%
epoch 5 loss:0.0166 train_accuracy:82.10%
test_accuracy:79.30%
epoch 6 loss:0.0156 train_accuracy:82.84%
test_accuracy:80.19%
epoch 7 loss:0.0150 train_accuracy:83.52%
test_accuracy:80.37%
epoch 8 loss:0.0143 train_accuracy:84.01%
test_accuracy:79.69%
epoch 9 loss:0.0134 train_accuracy:85.11%
test_accuracy:79.56%
epoch10 loss:0.0127 train_accuracy:85.75%
test_accuracy:79.36%
epoch11 loss:0.0120 train_accuracy:86.36%
test_accuracy:79.20%
epoch12 loss:0.0113 train_accuracy:87.43%
test_accuracy:79.05%
epoch13 loss:0.0105 train_accuracy:87.90%
test_accuracy:79.42%
epoch14 loss:0.0100 train_accuracy:88.59%
test_accuracy:78.33%
epoch15 loss:0.0094 train_accuracy:89.22%
test_accuracy:78.39%
```

Comparison figures:



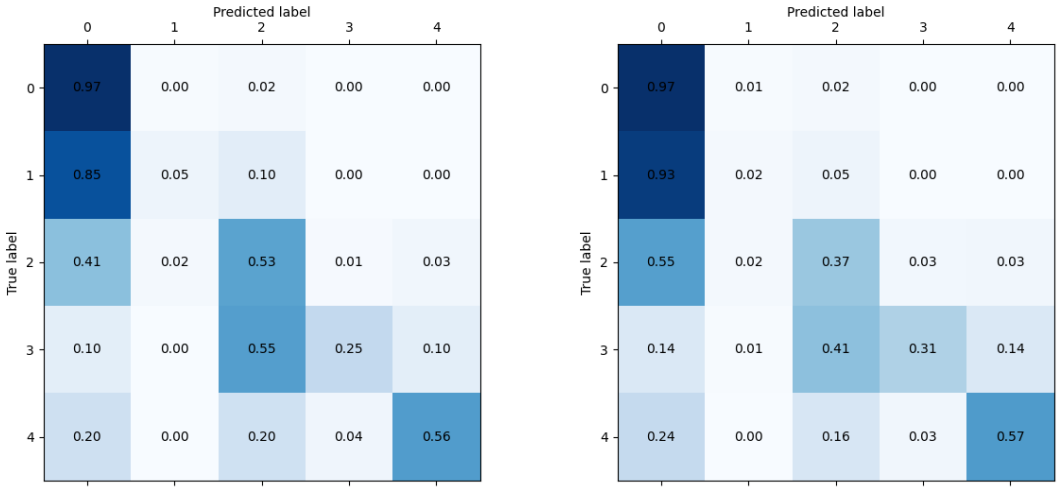
4. Discussion

除了resnet 18 50，with w/o pretrained之外，為了讓GPU ram 不爆掉，我原本在transformation裡加了resize將圖片壓縮(521>256)，壓縮圖片後雖然可以讓batch size設大一點，讓model可以更快跑完，但是卻讓結果變爛了:

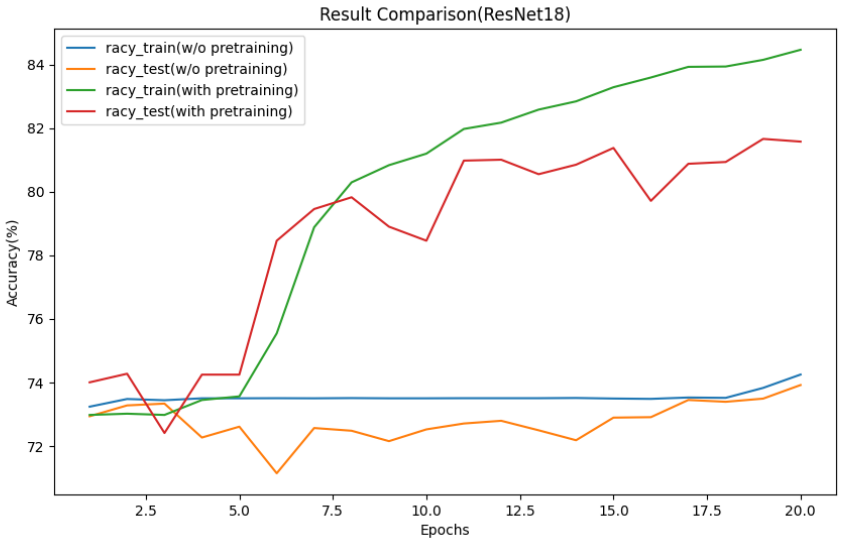
下圖展示resize對結果的影響，可以看到如果resize的話model會偏向預測label 0，我推測應該是因為resize太破壞特徵，導致model很難萃取其他重要特徵，所以結果變爛。

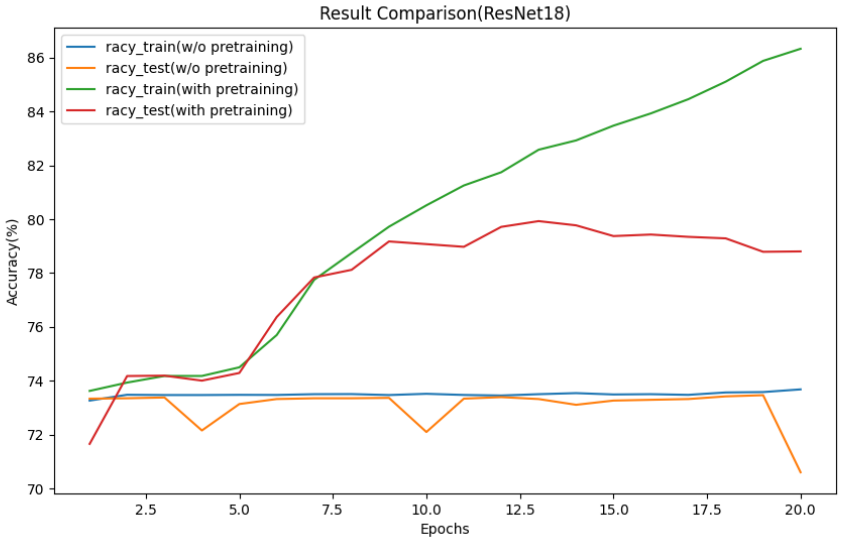
left: resnet18 with pretrained weights (no resized)

right: resnet18 with pretrained weights (resized)



再看比較曲線圖就更明顯了，沒有resize的test曲線雖然起起落落，但是還是有向上爬升，有resize的test曲線表現得平穩卻沒甚麼進步。顯示接收沒有resize圖片的model一直在調整藥萃取的特徵，所以雖然不穩定卻可以越來越精準。而接收resize過圖片的model看不太到有甚麼可以萃取的特徵，再怎麼調整都得不到越高的分數。


Result Comparison(ResNet18)
racy_train(w/o pretraining)
racy_test(w/o pretraining)
racy_train(with pretraining)
racy_test(with pretraining)
Accuracy(%)
Epochs


Result Comparison(ResNet18)
racy_train(w/o pretraining)
racy_test(w/o pretraining)
racy_train(with pretraining)
racy_test(with pretraining)
Accuracy(%)
Epochs

+