NYCU ML 2022 - HW2

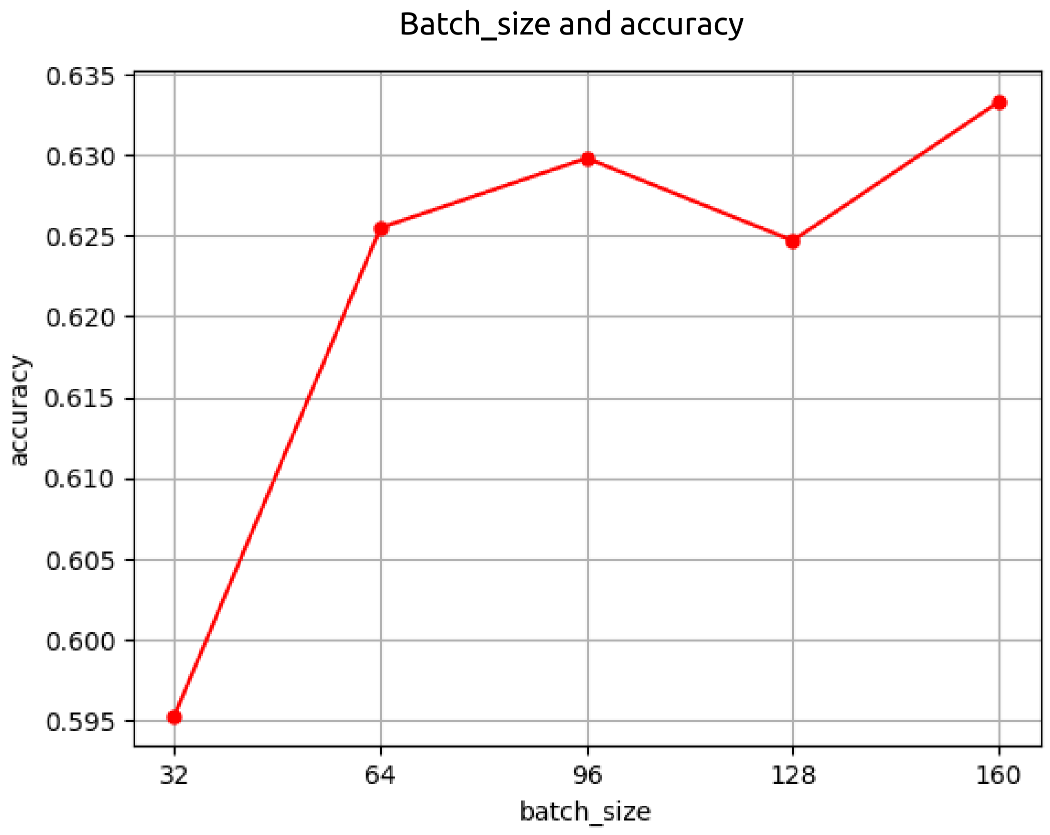
> Student : M093839 邱柏鎧

Test1

> Abstract :
>   Different batch_size effect.
> Usage :
>   Model = ResNet50
>   Epoch = 20
>   Optimizer = Adam

- Result :

Epoch (generation)	32	64	96	128	160
Accuracy	59.53%	62.55%	62.98%	62.47%	63.33%



- Conclusion :

  We can see that accuracy will increase as the batch_size become larger in most case, and there is a huge gap between batch_size=32 and others, it seems that the best decision is using batch_size=160 (or more). However, the larger batch_size is, the longer time should be undergoing theoretically. Combining two siginificant effect, since batch_size of 160 is only slightly better than 64, we choose 64 for latter tests.

Test2

> Abstract :
>   Changing Std and Mean.

> Usage :
> Model = ResNet50
> Batch_size = 32
> Epoch = 20
> Optimizer = Adam

- Result :
  - Accuracy = 60.61% (origin)

    ```
    mean = (0.5071, 0.4867, 0.4408)
    std = (0.2675, 0.2565, 0.2761)
    ```

  - Accuracy = 60.27% (modify)

    ```
    mean = (0.4914, 0.4822, 0.4465)
    std = (0.2023, 0.1994, 0.2010)
    ```

- Conclusion :

  It seems that both calculation is correct, since the results are similar.

- Reference :

  1. How to calculate mean and std for normalization

Test3

> Abstract :
> Different Epoch effect.
> Usage :
> Model = ResNet50
> Batch_size = 32
> Optimizer = Adam

- Result :
  - Wide Range - 20~200

Epoch (generation)	20	50	100	150	200
Accuracy	61.49%	61.61%	61.64%	61.98%	61.82%

  - Narrow Range (odd) - 155~195

Epoch (generation)	155	165	175	185	195
Accuracy	60.87%	61.18%	60.22%	60.55%	61.07%

  - Narrow Range (even) - 160~190

Epoch (generation)	150	160	170	180	190
Accuracy	61.97%	60.87%	61.24%	61.05%	62.03%

- Total Range - 20~200

Result 1

Epoch (generation)	20	50	100	150	160	170	180	190	200
Accuracy	60.63%	61.47%	61.97%	61.56%	61.76%	61.47%	61.34%	60.52%	61.79%

Result 2

Epoch (generation)	20	50	100	150	160	170	180	190	200
Accuracy	61.15%	60.92%	61.14%	60.79%	60.34%	61.15%	60.64%	60.86%	60.87%

Result 3

Epoch (generation)	20	50	100	150	160	170	180	190	200
Accuracy	61.24%	61.12%	62.07%	61.88%	61.64%	62.19%	62.22%	61.67%	61.67%

- Conclusion :

  There is no absolute consequence for epoch's effect, since different data combination might differ the result. However, we can roughly conclude that the higher epoch we produce, the higher probability to get better performence, or just ran 200 epoch and find the best accuracy for each different model and parameters.

Test4

> Abstract :
> Trying Data Augmentation.
> Usage :
> Model = ResNet50
> Batch_size = 64
> Epoch = 20
> Optimizer = Adam

Random Adding

- Result :

  1. Accuracy = 57.88% (Without any augmentation)
  2. Accuracy = 56.86% (Crop + HorizontalFlip + RandomRotation)

     ```
     train_transform = transforms.Compose(
       [transforms.RandomCrop(32, padding=4),
       transforms.RandomHorizontalFlip(),
       transforms.RandomRotation(15),
       transforms.ToTensor(),
       transforms.Normalize(mean, std)])
     ```

  3. Accuracy = 60.52% (HorizontalFlip - TA's basic way)
  4. Accuracy = 41.44% (HorizontalFlip + VerticalFlip + Affine + Grayscale)

```
train_transform = transforms.Compose(
[transforms.RandomHorizontalFlip(p=0.5),
transforms.RandomVerticalFlip(p=0.5),
transforms.RandomAffine(45,(0.1,0.1),(0.5,2),(0,45)),
transforms.RandomGrayscale(p=0.5),
transforms.ToTensor(),
transforms.Normalize(mean, std)])
```

5. Accuracy = 57.74% (VerticalFlip + HorizontalFlip - p = 0.5)
6. Accuracy = 57.74% (VerticalFlip + HorizontalFlip - p = 0.25)
7. Accuracy = 55.79% (Randomcrop + VerticalFlip)

```
train_transform = transforms.Compose(
  [transforms.RandomCrop(32, padding=4),
  transforms.RandomVerticalFlip(p=0.5),
  transforms.ToTensor(),
  transforms.Normalize(mean, std)])
```

- Reference :
  1. 常用图像处理与数据增强方法合集（torchvision.transforms)
  2. "TRANSFORMING AND AUGMENTING IMAGES" - Pytorch office web
  3. Data Augmentation in Pytorch
- Conclusion :

  It seems that more data augmentation reduce the accuracy, i guess the problem is that for complex model, quantity of cifar100 data set is not enough, even not suitable for using cross validation. So there are some other ways to deal with this issue, and i will try them for latter test.

RandomErasing

- Result :
  - Accuracy = 61.07% (batch_size = 64) / 58.88% (batch_size = 32)

```
train_transform = transforms.Compose(
[transforms.RandomCrop(32, padding=4),
 transforms.RandomHorizontalFlip(p=0.5),
 transforms.ToTensor(),
 transforms.Normalize(mean, std),
 transforms.RandomErasing(p=0.5, scale=(0.02,0.4), ratio=(0.3, 3.3))])
```

  - Accuracy = 61.15% (modify mean and std as paper)
- Reference :
  1. Random Erasing Augmentation - CSDN介紹
  2. "RandomErasing" - Pytorch office web

3. "Random Erasing Data Augmentation" Paper Code - github

- Conclusion :

  There might be some misunderstanding when i implement the papper's idea, or this solution is not fitting my expectation.

Test5

> Abstract :
>   Optimizer test.
> Usage :
>   Model = ResNet50
>   Batch_size = 64
>   Epoch = 20

SGD & Adam with different learning rate and weight_decay

- Result :

  1. Adam (lr = 0.0001 ; wd = 1e-4) -- 60.94%
  2. Adam (lr = 0.0001 ; wd = 5e-4) -- 60.90%
  3. Adam (lr = 0.1 ; wd = 1e-4) -------- 0.88%
  4. SGD (lr = 0.0001 ; wd = 5e-4) ----- 63.97% | 64.54%
  5. SGD (lr = 0.1 ; wd = 5e-4) ----------- 38.75%
  6. SGD (lr = 0.0001 ; wd = 1e-4) ----- 64.83% || 15min44s

  ```
  optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=5e-4)
  optimizer = optim.SGD(model.parameters(), lr=0.0001,
  momentum=0.9,weight_decay=5e-4)
  ```

- Conclusion :

  SGD behaves better than Adam in my test, many other paper also maintain using SGD as the optimizer. We can also find that when learning-rate is 0.0001 and weight-decay is 1e-4, the best performence occurs, i wish to continue reducing learning-rate, however training time will simutaneously increase, so i also try using dynamic lr.

Dynamic learning rate

- Aim :

  Increase Number of Epoch can improve accuracy, however, it will also take lots of time for training. Let learning rate be dynamic can theoretically reduce time but maintain well behavior of model.

- Modify :

  ```
  optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-
  4)
  def get_lr(optimizer):
      for param_group in optimizer.param_groups:
          return param_group['lr']
  ```

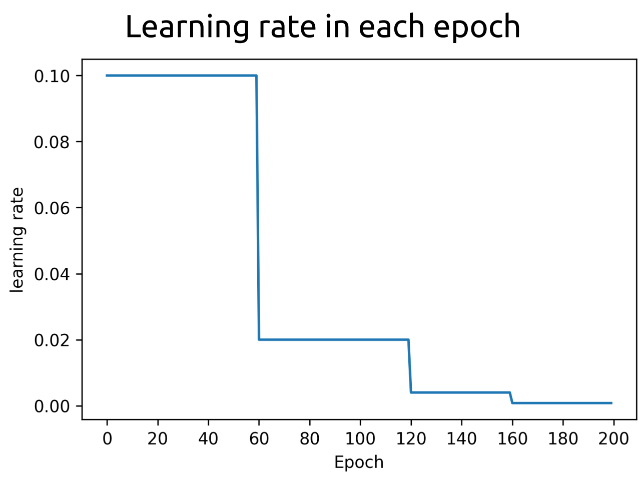
```
# dynamic learning rate
train_scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=
[60,120,160], gamma=0.2)

# training part
for epoch in range(total_epoch):
  check_lr = get_lr(optimizer)
  lrs.append(check_lr)
  for i, data in enumerate(trainloader, 0):
      ......
      # forward + backward + optimize
      outputs = model(inputs)
      loss = criterion(outputs, labels)
      loss.backward()
      optimizer.step()
  train_scheduler.step()
  torch.save(model, save_path)
```

- Result :

Type	**Accuracy**	**Time**
Adam_20	61.34	13m12s
SGD_20	64.31	11m20s
SGD_200	65.20	1hr52m34s
SGD_200_dynamic	55.12	1hr51m56s

* 20 or 200 means epoch



- Reference :

  1. Image Classification of CIFAR100 Dataset Using PyTorch
  2. Pytorch_學習效率調整法
  3. 加速Pytorch訓練技巧

- Conclusion :

  There is no obvious changing after using dynamic learning rate. Even worse, the accuracy drop dramatically. Normally, warm-up is also implemented while using dynamic learning rate, this might be the reason affect accuracy.

Test6

> Abstract :
> 　Modify Model.
> Usage :
> 　Batch_size = 64
> 　Epoch = 200
> 　Optimizer = SGD

(1) SEResNet

- Aim :

  The major problem of cifar100 is mount of data set. I think attention might be a good way to make model learn more focus on the right feature, so does some paper say, attention can solve unbalanced or small data set. SEResNet is a improvement of ResNet, and the architecture makes it performence similar to self-attention.

- Modify :

4

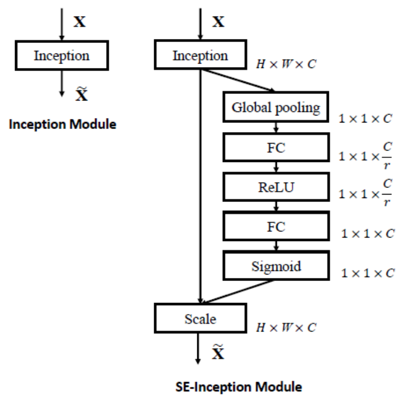


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

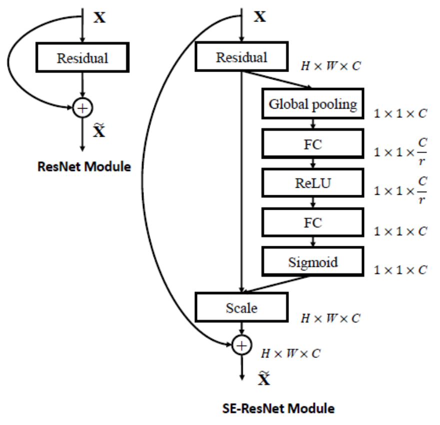


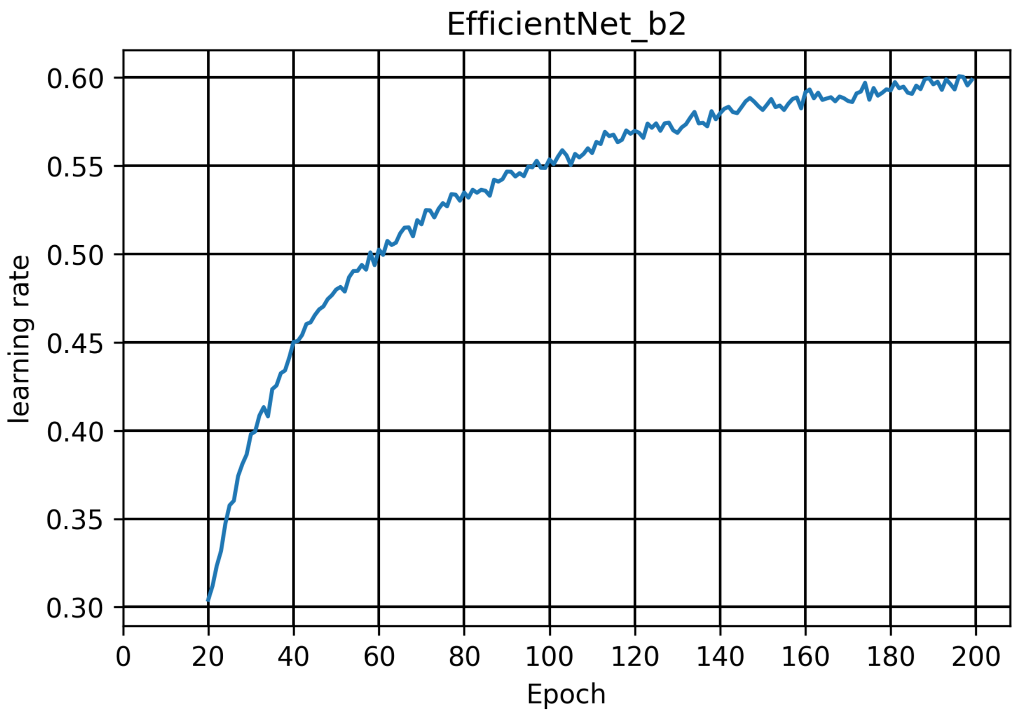
Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

- Result :
  - Accuracy = 37.45%

- Reference :
  1. SE模塊理解 + SE-Resnet模塊pytorch實踐
  2. 深度學習基礎-SEResNet

(2) Efficient Net

- Accuracy = 60.07% (efficientNet_b2 at epoch 197)

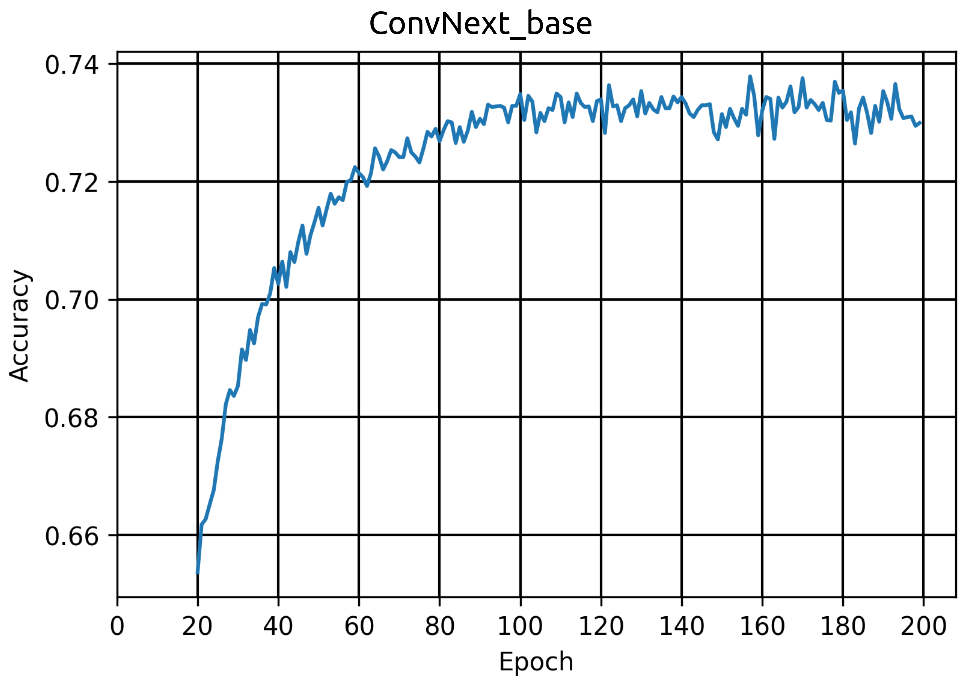


- Accuracy = 1% (efficientNet_b3)
- Accuracy = 34.85% (efficientNet_b4 at epoch 199)

(3) ConvNext

- Result :
  - Accuracy = 25.23% (ConvNext_tiny)
  - Accuracy = 73.78% (ConvNext_base at epoch 157)

```
model = models.convnext_base(pretrained=True)
model.classifier[2] = torch.nn.Linear(1024, num_classes)
```



(4) Others

- inceptionV3 - failed
- inceptionV1 (googlenet) - Accuracy = 1%
- Byol (self_supervised) - Accuracy = 0.01%

(5) Modify ResNet50

- Modify :

  change the kernal_size as the try of others.

  ```
  model = models.resnet50(pretrained=True)
  model.fc = torch.nn.Linear(2048, num_classes)
  model.conv1 = nn.Conv2d(3,64,kernel_size=(3,3), stride=(1,1), padding=(1,1),
  bias=False)
  model.maxpool = nn.Identity()
  ```

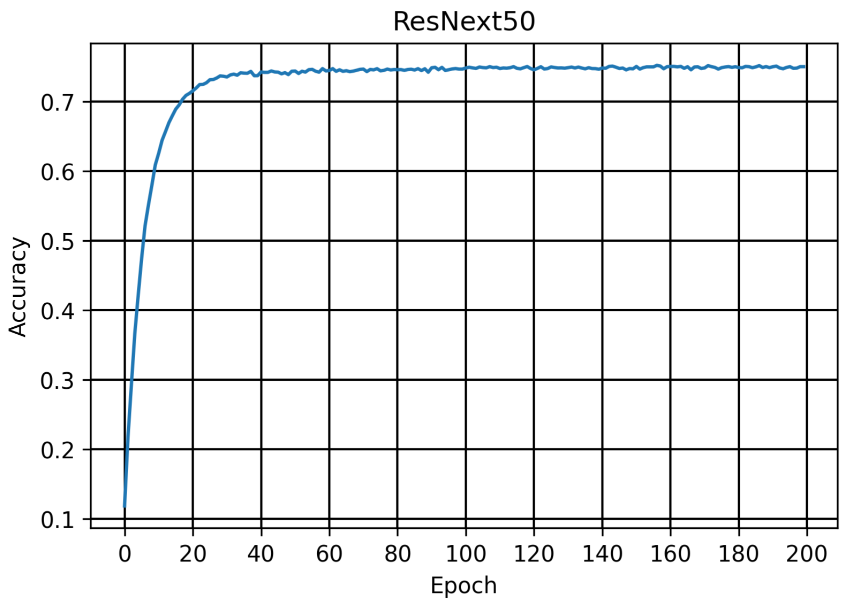
- Result :
  - Accuracy = 73.44% / 72.71% (test 1 / test 2)
- Reference :
  1. 在CIFAR100上训练ResNet

(6) Modify ResNext

- Modify :

  ```
  model = models.resnext50_32x4d(pretrained=True)
  model.conv1 = nn.Conv2d(3,64,kernel_size=(3,3), stride=(1,1), padding=(1,1),
  bias=False)
  model.maxpool = nn.Identity()
  model.fc = torch.nn.Linear(2048, num_classes)
  ```

- Result :
  - Accuracy = 75.18% (at epoch 156)

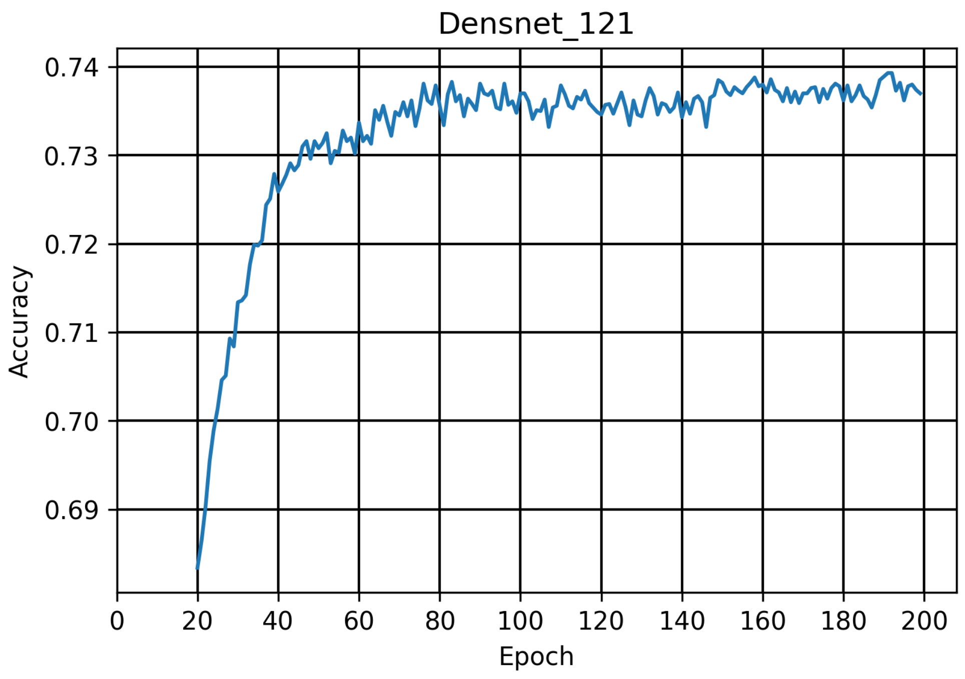


(7) Modify Densenet

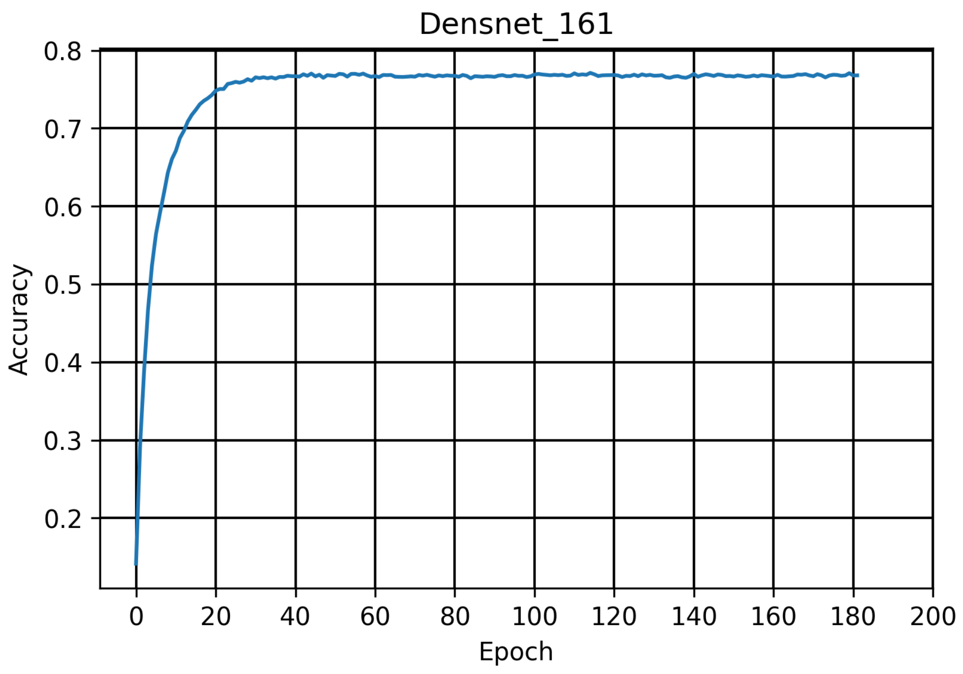
- Modify :

```
model = models.densenet201(pretrained=True)
# model.classifier = torch.nn.Linear(1024, num_classes)
# model.classifier = torch.nn.Linear(2208, num_classes)
# model.classifier = torch.nn.Linear(1664, num_classes)
model.classifier = torch.nn.Linear(1920, num_classes)
model.features.conv0 = nn.Conv2d(3,64,kernel_size=(3,3), stride=(1,1), padding=
(1,1), bias=False)
model.features.pool0 = nn.Identity()
```

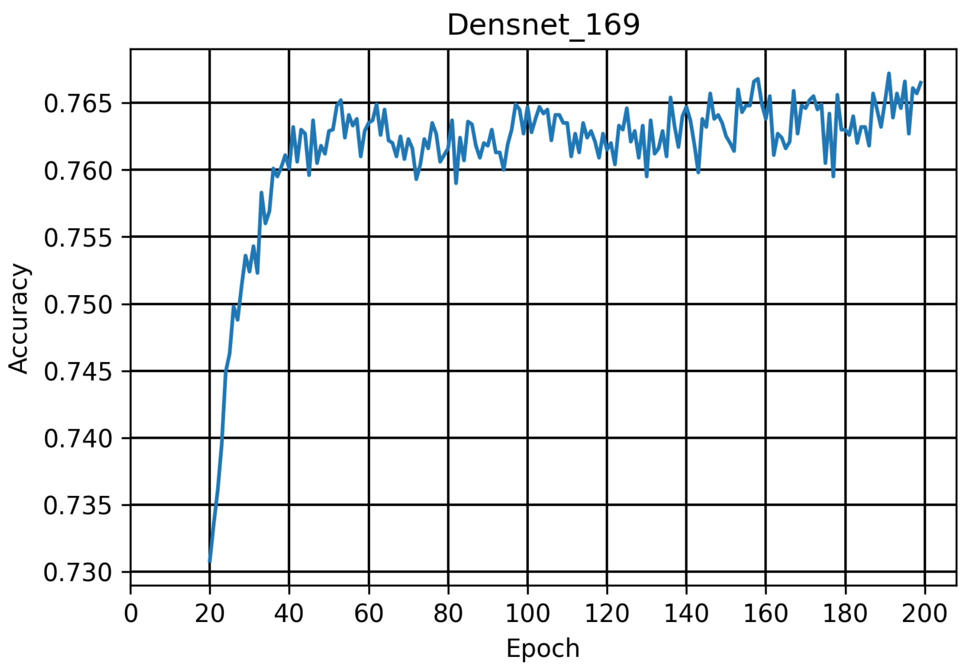
- Result :
  - Accuracy = 73.93% - (densenet_121 at epoch 193)



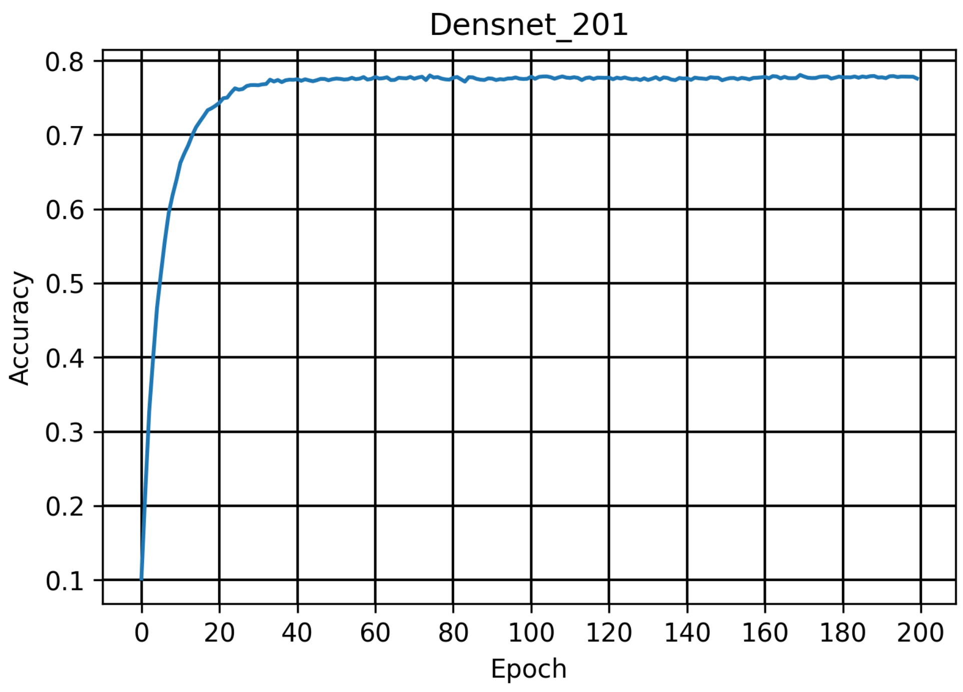
- Accuracy = 77.11% - (densenet_161 at epoch 114)



- Accuracy = 76.72% - (densenet_169 at epoch 191)



- Accuracy = 78.09% - (densenet_201 at epoch 169)



- Conclusion :

  It seems that after modify, Densenet is the better choice. Among all model, densenet_161 has largest size, and it is the only one over size limit (100MB). The most siginificant improvment of densenet is emphasize the importance of features, this is also the reason that why it is better than ResNet.

- Reference :

  1. DenseNet：比ResNet更优的CNN模型

Test7_Final

> Abstract :
>   Make data more bigger (double the data set).
> Usage :
>   Model = Modify DenseNet201
>   Batch_size = 64
>   Total_Epoch = 200
>   Optimizer = SGD

Add another Data

- Aim :

  Since data augmentation somehow performs like increasing the mount of data for training, it still not work very well, and the real training data set is still small. So, I try to combine two training set, one is original one, and another is data set that all experience data augmentation, I think it will significantly affect result.

- Modify :

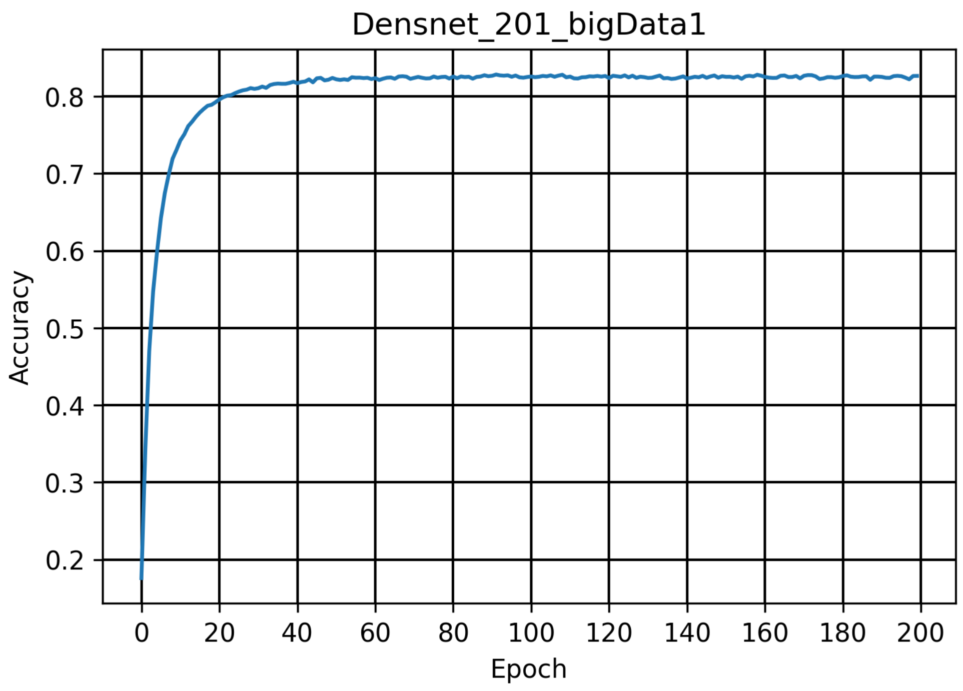
```
# For data set 1
train_transform1 = transforms.Compose(
    [transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)])
```

```
# For data set 2
train_transform2 = transforms.Compose(
    [transforms.RandomResizedCrop(size=32, scale=(0.08, 1.0), ratio=(0.75, 1.33),
interpolation=2),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)])

trainset1 = torchvision.datasets.CIFAR100(root='./data', train=True,
download=True, transform=train_transform1)
trainset2 = torchvision.datasets.CIFAR100(root='./data', train=True,
download=True, transform=train_transform2)

trainloader = torch.utils.data.DataLoader(trainset1+trainset2,
batch_size=batch_size, shuffle=True, num_workers=2)
```

- Result :
  - Accuracy = 82.83% (at epoch 91)



- Conclusion :

  Since it takes over 20 hours to train 200 epoch, I can't do other attempt. This is the final version for this homework, it seems that increasing the data set directly helps improving accuracy, also decrease overfitting.

- Reference :
  1. Training model to classify cifar100

Future

There are still a lot of model, structure can be implementation, like SAM, however, they are also hard to implement for me currently. On the other hand, maybe we can add more data with augmentation just like my last try, maybe we can make the accuracy better.