Mango Grade

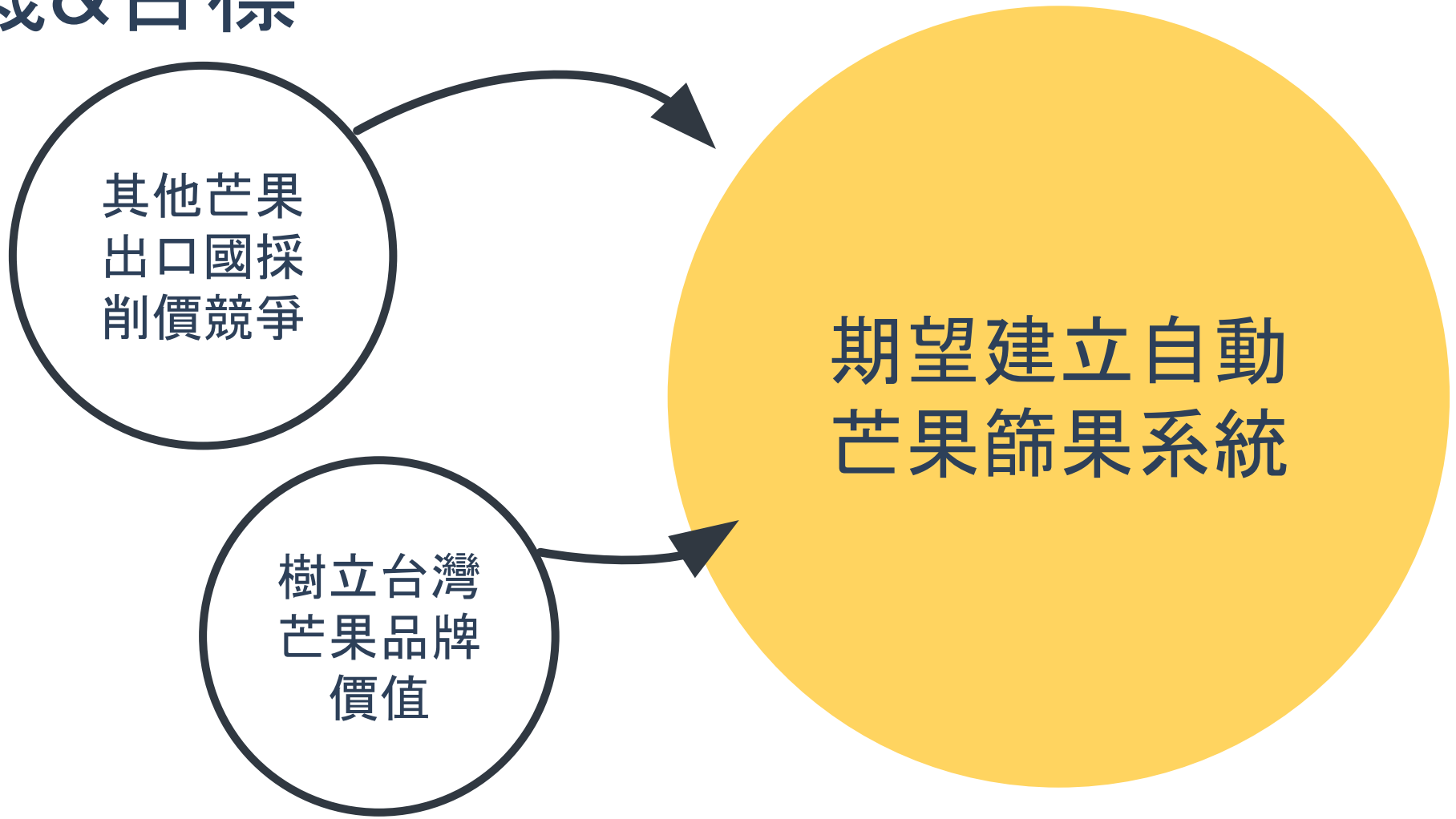
金融五 張靜如
統研一 林澤慶

動機&目標

其他芒果
出口國採
削價競爭

樹立台灣
芒果品牌
價值

動機&目標



Data

等級A

等級B

等級C

評估方式

評估方式採用 WAR（Weighted Average Recall），其公式如下：

$$WAR = \sum_{i=1}^{I} w(i) * Recall(i) \text{ ，其中} I = 3$$

- $w(i)$: Weighting factor of each class
- $Recall(i)$: Recall of each class

資料前處理

Step1/ Install Packages

In [1]:

```
# pytorch深度學習模組套件
from torchvision import transforms
from torchvision import models
import torch
from torch.autograd import Variable
import torch.nn as nn
from torch.optim import lr_scheduler
from torch import optim
from torchvision.datasets import ImageFolder
from torchvision.utils import make_grid
import torch.nn.functional as F
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader
import torch.utils.data as data
import glob
```

Step1/ Install Packages

In [2]:
```
# 資料處理套件
import os
import cv2
import csv
import random
import time
import numpy as np
import pandas as pd
import matplotlib.image as mpimg # mpimg 用於讀取圖片
import matplotlib.pyplot as plt # plt 用於顯示圖片
import seaborn as sns
from PIL import Image
```

In [4]:
```
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
```

In [5]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
```

Step2\讀取資料\label

```
[10] train_label.head()
```

	image_id	label
0	00002.jpg	C
1	00003.jpg	C
2	00004.jpg	B
3	00005.jpg	A
4	00007.jpg	A

Step2\讀取資料\label

```
[10] train_label.head()
```

	image_id	label
0	00002.jpg	C
1	00003.jpg	C
2	00004.jpg	B
3	00005.jpg	A
4	00007.jpg	A

A → 0
B → 1
C → 2

Step2\資料前處理\data

```
In [0]: train_label[train_label['label']=='A']=0
        train_label[train_label['label']=='B']=1
        train_label[train_label['label']=='C']=2
```

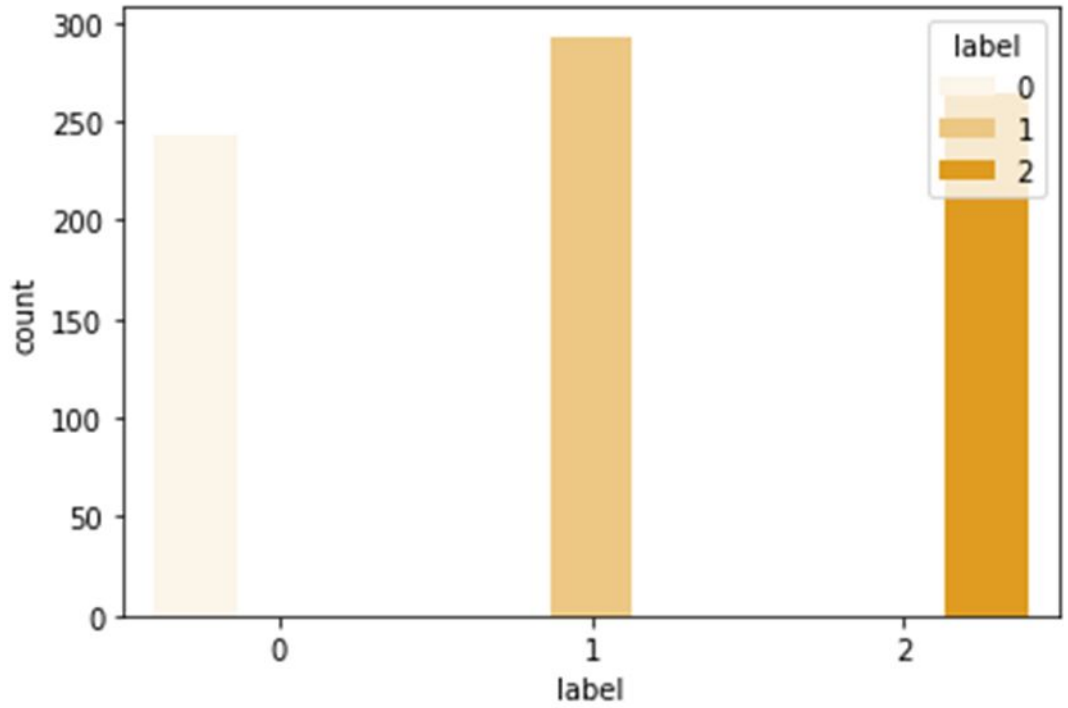
```
In [49]: dev_label[dev_label['label']=='A']=0
         dev_label[dev_label['label']=='B']=1
         dev_label[dev_label['label']=='C']=2
         dev_label.head()
```

Out[49]:

	image_id	label
0	1	1
1	0	0
2	1	1
3	1	1
4	2	2

Step2\資料前處理\data\EDA

```
# Show the bar plot
sns.countplot(dev_label['label'],hue = dev_label['label'],color='orange')
```



Training Data : 5,600筆
Testing Data : 800筆

```
train_sec = dev_label.groupby('label')
train_sec.size()
```

```
label
0    243
1    293
2    264
dtype: int64
```

Step3/ Transform

```
In [15]: img = mpimg.imread("C1-P1_Train/00002.jpg")
```

```
In [16]: # 顯示原始圖片的比例
         img.shape

Out[16]: (1008, 1344, 3)
```

Step3/ Transform

```
In [15]: img = mpimg.imread("C1-P1_Train/00002.jpg")
```

```
In [16]: # 顯示原始圖片的比例
         img.shape

Out[16]: (1008, 1344, 3)
```

Resize image as (224, 224)

隨意水平翻轉

標準化

Step3/ Resize image as (224, 224)

Resize image as (224, 224)

隨意水平翻轉

標準化

```
In [18]: transform = transforms.Compose([transforms.Resize((224, 224)), ##調整size 到同一尺寸
                                         transforms.RandomHorizontalFlip(), #隨意做旋轉
                                         transforms.ToTensor(),
                                         transforms.Normalize((0.485,0.456,0.406),(0.229,0.224,0.225))])
```

建立dataset

Step1/ 建立dataset

In [17]:
```python
class mangoImageFloder(Dataset):
    def __init__(self, img_path, label_path,transform = None):
        super().__init__()
        ##############################################
        ### Initialize paths, transforms, and so on
        ##############################################
        self.label = pd.read_csv(label_path)
        self.image_all_path = glob.glob(os.path.join(img_path, '*.jpg'))
        self.transform = transform
    def __len__(self):
        ##############################################
        # 1. Read from file (using numpy.fromfile, PIL.Image.open)
        # 2. Preprocess the data (torchvision.Transform).
        # 3. Return the data (e.g. image and label)
        ##############################################
        return len(self.image_all_path)
    def __getitem__(self, index):
        ##############################################
        ### Indicate the total size of the dataset
        ##############################################
        img = Image.open(self.image_all_path[index])
        img_name = self.label['image_id'][index]
        img_label = self.label["label"][index]
        if self.transform is not None:
            return self.transform(img), img_label
        else:
            return img, img_label
```

Step1/ 建立dataset

```
[40] Train_dataloader = DataLoader(
         dataset=mangoImageFloder(img_path=path+'C1-P1_Train/',
                                  label_path=path+'train.csv',transform= transform),
         batch_size=4,
         shuffle=False)
```

```
[42] Test_dataloader = DataLoader(
         dataset=mangoImageFloder(img_path=path+'C1-P1_Dev/',
                                  label_path=path+'dev.csv',transform= transform),
         batch_size=4,
         shuffle=False)
```

建模

Resnet18

Resnet101

Densenet161

Densenet121

Resnet18

Resnet101

Densenet161

Densenet121

Step1/ 導入模型 / Resnet18

```
[31] model = models.resnet18(pretrained=True)
     model.fc
```

```
Linear(in_features=512, out_features=1000, bias=True)
```

```
[32] num_ftrs  = model.fc.in_features
     model.fc = nn.Linear(num_ftrs, 3)
     model.fc
```

```
Linear(in_features=512, out_features=3, bias=True)
```

Step2/ 確認使用環境

In [21]:
```
# GPU
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
print('GPU state:', device)
```

```
GPU state: cuda:0
```

In [30]:
```
model = model.cuda()
```

Step3/ 設定loss fn & optim fn

In [31]:
```
# Loss and Optimizer
learning_rate = 0.001
loss_function = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

Step4/ Training

```
for epoch in range(2):

    running_loss= 0.0

    for i,data in enumerate(Train_dataloader,0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device) ##將資料交給GPU運算

        optimizer_ft.zero_grad()

        outputs = model(inputs)
        loss = loss_function(outputs, labels)
        loss.backward() ##呼叫 loss 的 backward()
        optimizer_ft.step() ## 更新權重

        running_loss += loss.item() ##每次累計其loss值
        if i%200 == 199:
            print('[%d, %5d] loss: %.3f'%(epoch+1,i+1,running_loss/200)) ##計算每500筆資料的平均loss
            running_loss = 0.0

print("Training finished! Yay!!")
```

Step4/ Training

```python
for epoch in range(2):

    running_loss= 0.0

    for i,data in enumerate(Train_dataloader,0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(devi

        optimizer_ft.zero_grad()

        outputs = model(inputs)
        loss = loss_function(outputs, labels)
        loss.backward() ##呼叫 loss 的 backward()
        optimizer_ft.step() ## 更新權重

        running_loss += loss.item() ##每次累計其loss值
        if i%200 == 199:
            print('[%d, %5d] loss: %.3f'%(epoch+1,i+1,runni
            running_loss = 0.0

print("Training finished! Yay!!")
```

```
[1,   200] loss: 1.043
[1,   400] loss: 0.937
[1,   600] loss: 1.002
[1,   800] loss: 0.888
[1,  1000] loss: 0.932
[1,  1200] loss: 0.941
[1,  1400] loss: 0.861
[2,   200] loss: 0.785
[2,   400] loss: 0.711
[2,   600] loss: 0.765
[2,   800] loss: 0.696
[2,  1000] loss: 0.756
[2,  1200] loss: 0.803
[2,  1400] loss: 0.699
Training finished! Yay!!
```

Step5/ Testing

```
# Test
correct = 0
total = 0
with torch.no_grad():
    for data in Test_dataloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 800 test inputs: %d %%' % (100 * correct / total)) ##計算預測的正確率
```

```
Accuracy of the network on the  800  test inputs: 71 %
```

Step5/ Testing

In [32]:
```
classes=("A","B","C")
```

In [34]:
```
class_correct = list(0. for i in range(3))
class_total = list(0. for i in range(3))
with torch.no_grad():
    for data in Test_dataloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze() ##torch.squeeze() 若torch.size:(1*3)=>(3)
        for i in range(4): #batch_size=4
            label = labels[i]
            class_correct[label] += c[i].item() ##若該類預測正確 則+1
            class_total[label] += 1 ##計算各類的總數

for i in range(3):
    print('Accuracy of %5s : %2d %%' % (classes[i], 100 * class_correct[i] / class_total[i])) ##計算各類的準確率
```

```
Accuracy of     A : 76 %
Accuracy of     B : 61 %
Accuracy of     C : 76 %
```

Step5/ Testing

In [39]:
```
torch.save(model,"resnet18_2")
```

In [32]:
```
resnet18_2 = torch.load("resnet18_2")
```

In [47]:
```
predictions =[]
```

In [50]:
```
with torch.no_grad():
    for data in Test_dataloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        predictions.append(predicted.cpu().numpy())
```

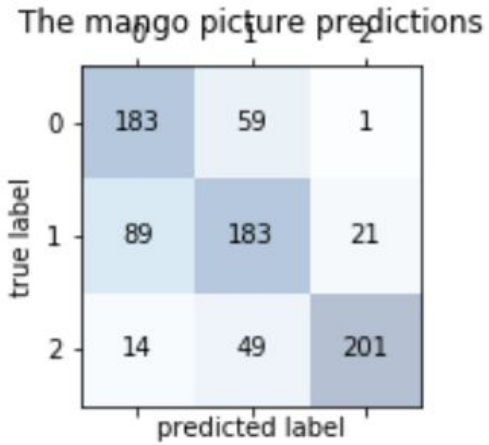
In [56]:
```
y_pred=[]
```

In [58]:
```
for i ,data in enumerate(predictions):
    for j,data_2 in enumerate(data):
        y_pred.append(data_2)
```

Step5/ Testing

In [63]:
```
y_true = [k for k in test_label["label"]]
```

In [72]:
```
confmat = confusion_matrix(y_true=y_true, y_pred=y_pred)
fig, ax = plt.subplots(figsize=(2.5, 2.5))
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
    for j in range(confmat.shape[1]):
        ax.text(x=j, y=i, s=confmat[i,j], va='center', ha='center')
plt.xlabel('predicted label')
plt.ylabel('true label')
plt.title("The mango picture predictions")
plt.show()
```



Step6/ Training more epoch

```
Loss and Optimizer
learning_rate = 0.001
loss_function = nn.CrossEntropyLoss()
optimizer_ft = optim.SGD(resnet18_2.parameters
exp_lr_scheduler = lr_scheduler.StepLR(optimiz
```

```
for epoch in range(2,4): ##繼續迭代2次

    running_loss= 0.0

    for i,data in enumerate(Train_dataloader,0
        inputs, labels = data
        inputs, labels = inputs.to(device), la

        optimizer_ft.zero_grad()

        outputs = resnet18_2(inputs)
        loss = loss_function(outputs, labels)
        loss.backward()
        optimizer_ft.step()

        running_loss += loss.item()
        if i%200 == 199:
            print('[%d, %5d] loss: %.3f'%(epoc
            running_loss = 0.0

int("Training finished! Yay!!")
```

```
[1,   200] loss: 1.043
[1,   400] loss: 0.937
[1,   600] loss: 1.002
[1,   800] loss: 0.888
[1,  1000] loss: 0.932
[1,  1200] loss: 0.941
[1,  1400] loss: 0.861
[2,   200] loss: 0.785
[2,   400] loss: 0.711
[2,   600] loss: 0.765
[2,   800] loss: 0.696
[2,  1000] loss: 0.756
[2,  1200] loss: 0.803
[2,  1400] loss: 0.699
Training finished! Yay!!
```

```
[3,   200] loss: 0.611
[3,   400] loss: 0.601
[3,   600] loss: 0.643
[3,   800] loss: 0.609
[3,  1000] loss: 0.636
[3,  1200] loss: 0.720
[3,  1400] loss: 0.645
[4,   200] loss: 0.481
[4,   400] loss: 0.514
[4,   600] loss: 0.528
[4,   800] loss: 0.559
[4,  1000] loss: 0.537
[4,  1200] loss: 0.589
[4,  1400] loss: 0.567
Training finished! Yay!!
```

```
torch.save(resnet18_2,"resnet18_4") ##儲存model
```

Step6/ Testing more epoch

```
# Test
correct = 0
total = 0
with torch.no_grad():
    for data in Test_dataloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = resnet18_2(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 800 test inputs: %d %%' % (100 * correct / total))
確率
```

```
Accuracy of the network on the 800 test inputs: 73 %
```

Step6/ Testing more epoch

```
predictions_18_4 = []
```

```
class_correct = list(0. for i in range(3))
class_total = list(0. for i in range(3))
with torch.no_grad():
    for data in Test_dataloader:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(de
        outputs = resnet18_2(inputs)
        _, predicted = torch.max(outputs, 1)
        predictions_18_4.append(predicted.cpu().numpy())
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(3):
    print('Accuracy of %5s : %2d %%' % (classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of     A : 78 %
Accuracy of     B : 65 %
Accuracy of     C : 75 %
```

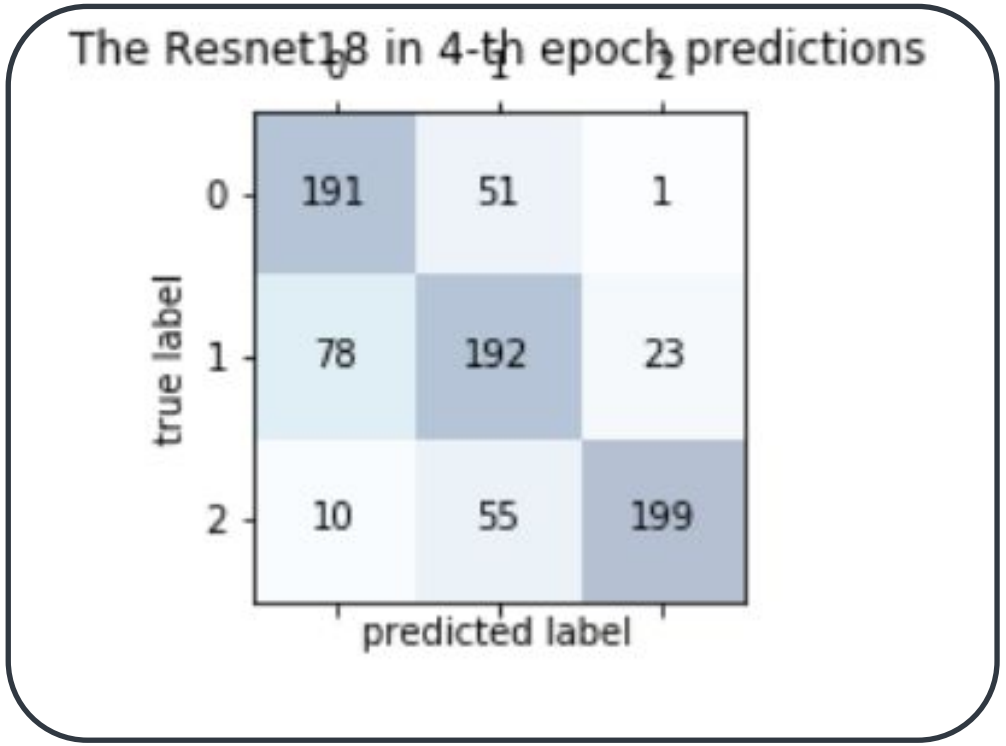
Step6/ Testing more epoch

```
y18_4_pred=[]
```

```
for i ,data in enumerate(predictions_18_4):
    for j,data_2 in enumerate(data):
        y18_4_pred.append(data_2)
```

```
y_true = [k for k in test_label["label"]]
```

```
confmat = confusion_matrix(y_true=y_true, y_pred=y18_4_pred)
fig, ax = plt.subplots(figsize=(2.5, 2.5))
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
    for j in range(confmat.shape[1]):
        ax.text(x=j, y=i, s=confmat[i,j], va='center', ha='center')
plt.xlabel('predicted label')
plt.ylabel('true label')
plt.title("The Resnet18 in 4-th epoch predictions")
plt.show()
```



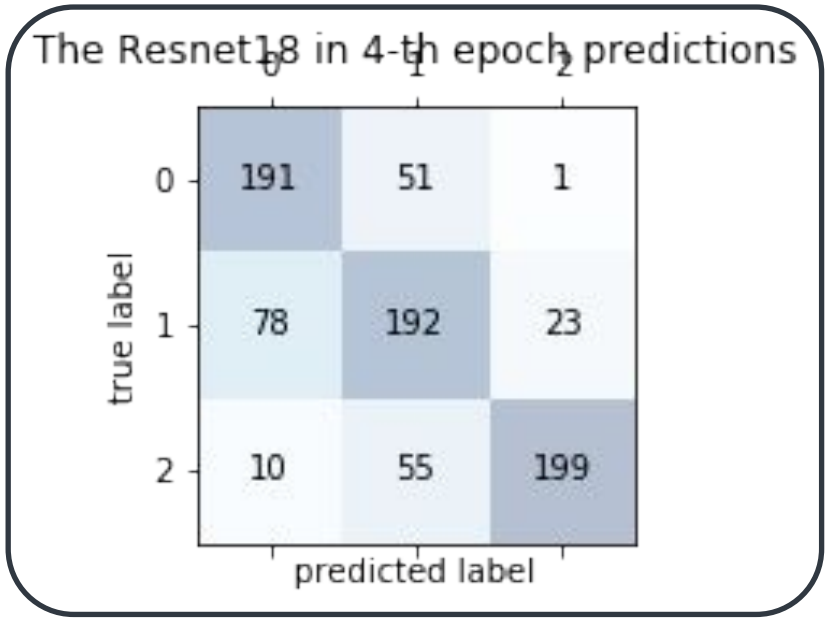
Resnet18

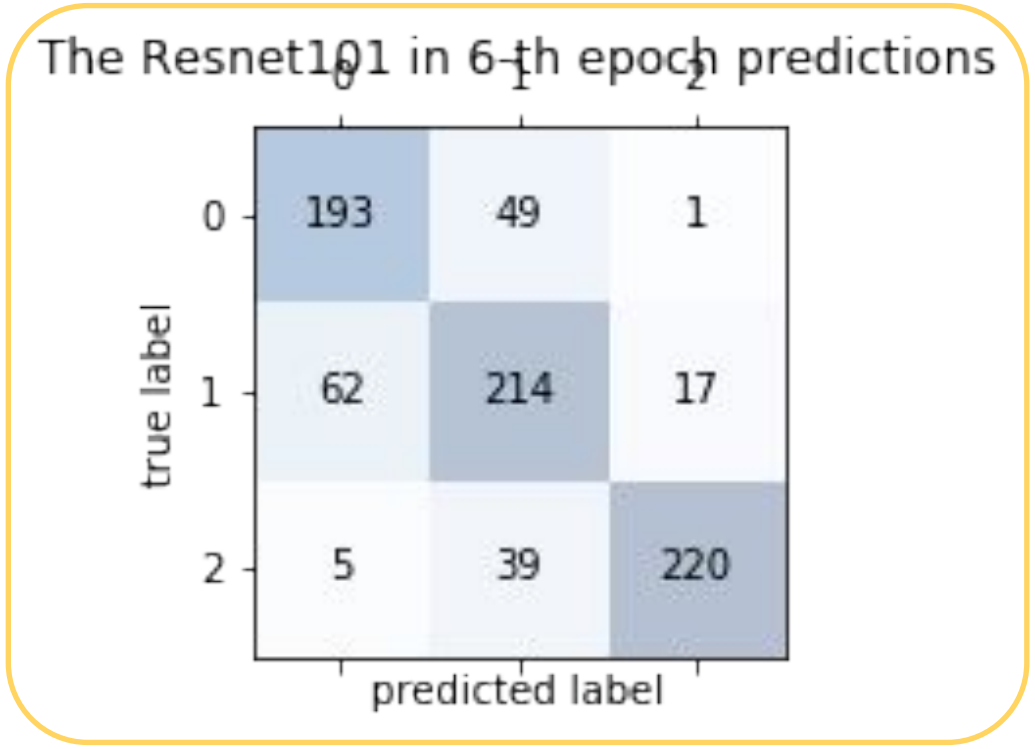
Resnet101

Densenet161

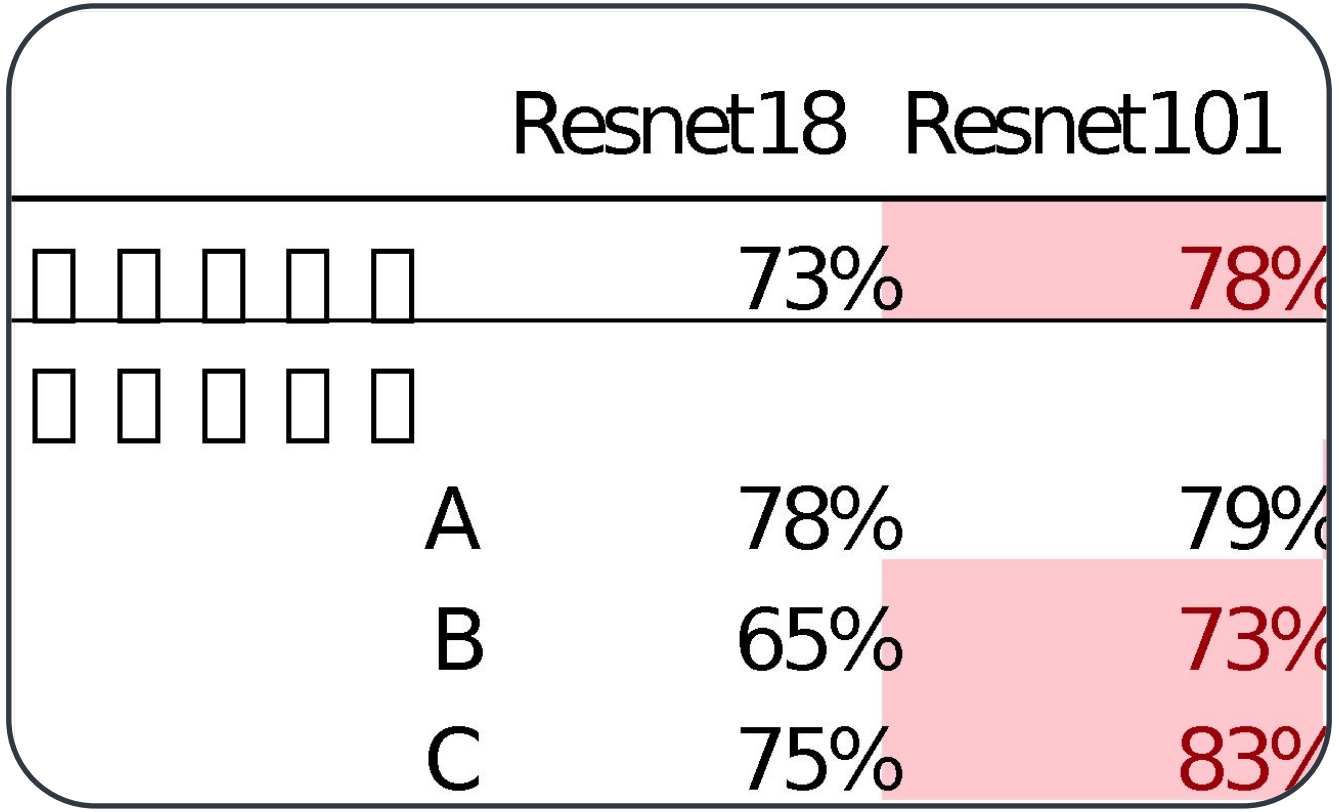
Densenet121

Step2/ Compare Resnet101 with Resnet18





Step2/ Compare Resnet101 with Resnet18



Resnet18

Resnet101

Densenet161

Densenet121

Step1/ 導入模型 / Densenet161

```
densenet161 = torch.hub.load('pytorch/vision:v0.6.0', 'densenet161', pretrained=True)
```

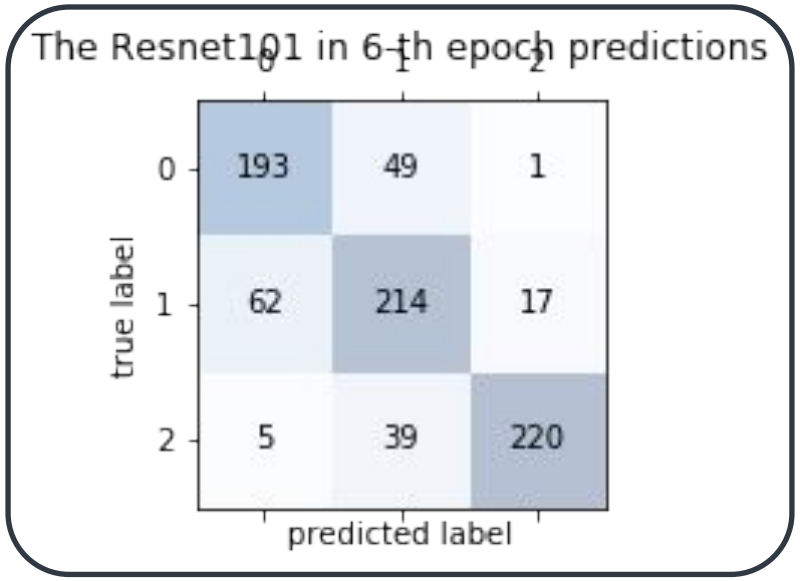
```
transform_den = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

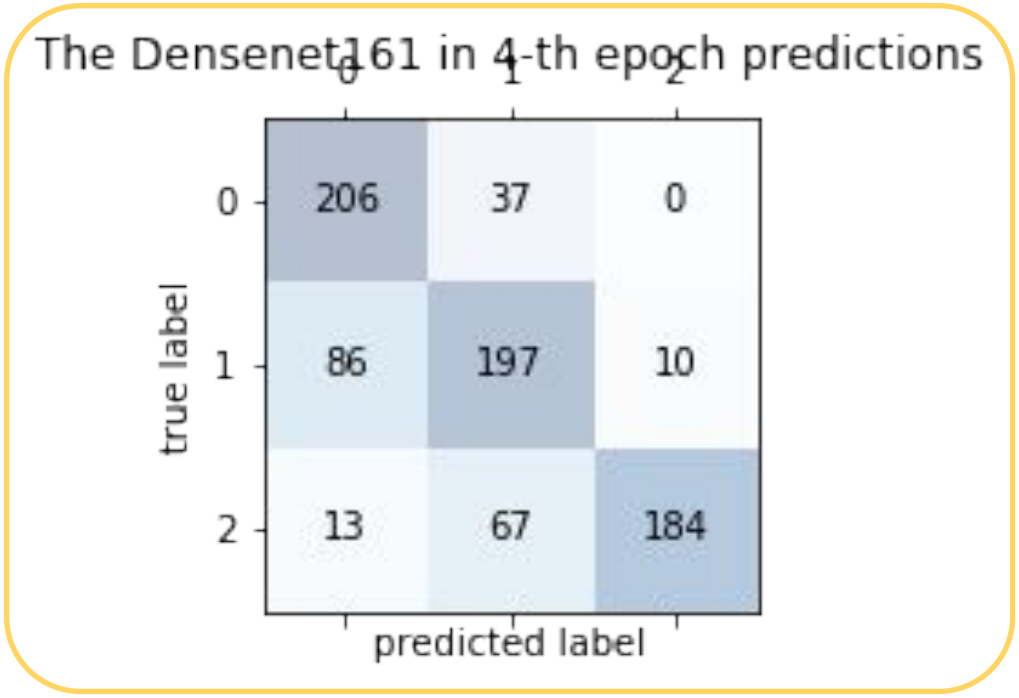
Step1/ 導入模型 / Densenet161

```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (denseblock1): _DenseBlock(
                                          .
                                          .
                                          .
                                          .
                                          .
                                          .
      (norm5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (classifier): Linear(in_features=2208, out_features=1000, bias=True)
  )
```

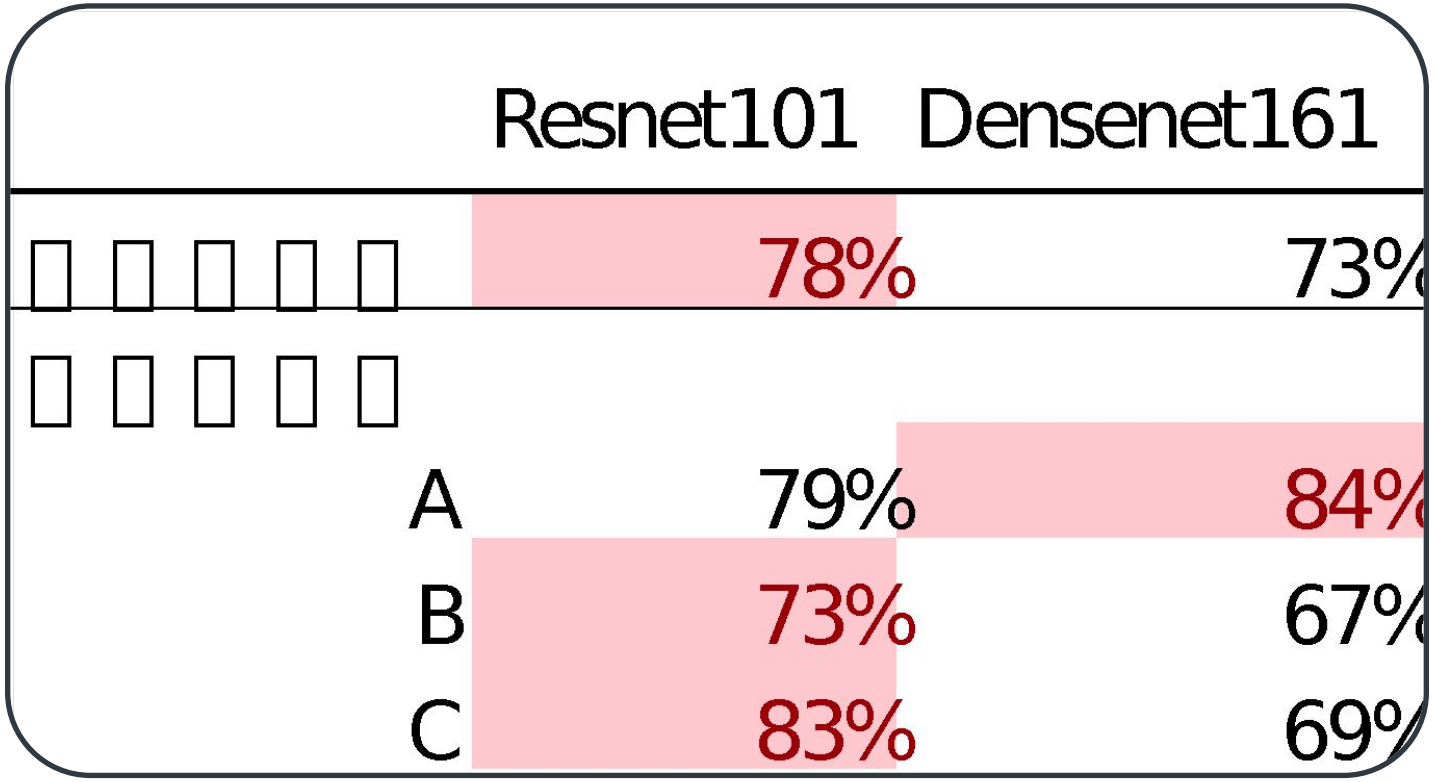
```
densenet161.classifier = nn.Linear(in_features=2208 , out_features=3,bias=True) ##分成3類

densenet161 = densenet161.cuda()
```

Step2/ Compare Resnet101 with Densenet161





Step2/ Compare Resnet101 with Densenet161



Resnet18

Resnet101

Densenet161

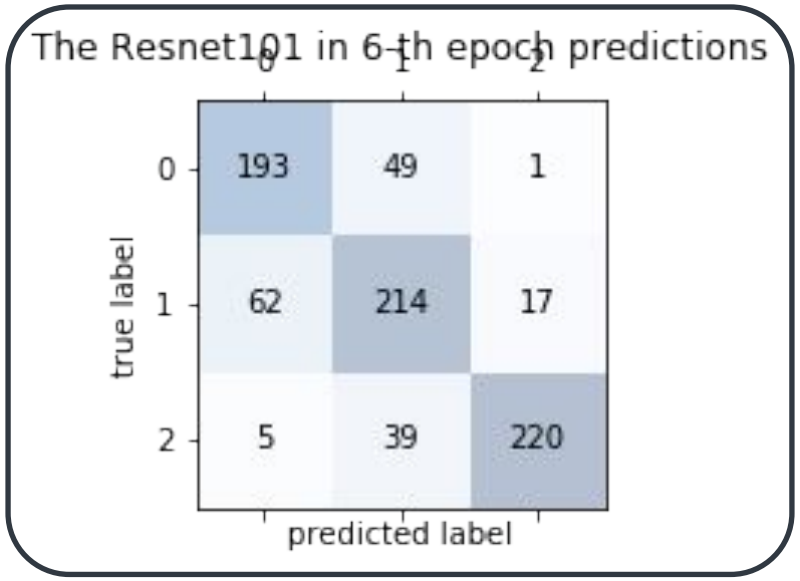
Densenet121

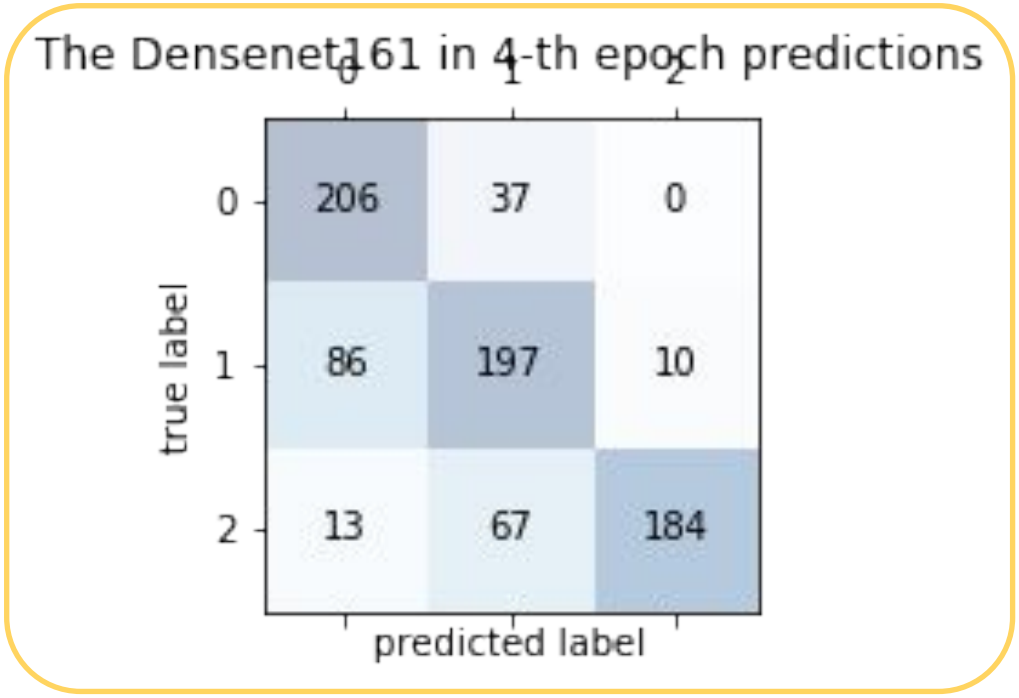
Step1/ 導入模型 / Densenet161

```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (denseblock1): _DenseBlock(

                                          .
                                          .
                                          .
                                          .
                                          .
                                          .

      (norm5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (classifier): Linear(in_features=1024, out_features=1000, bias=True)
  )
```

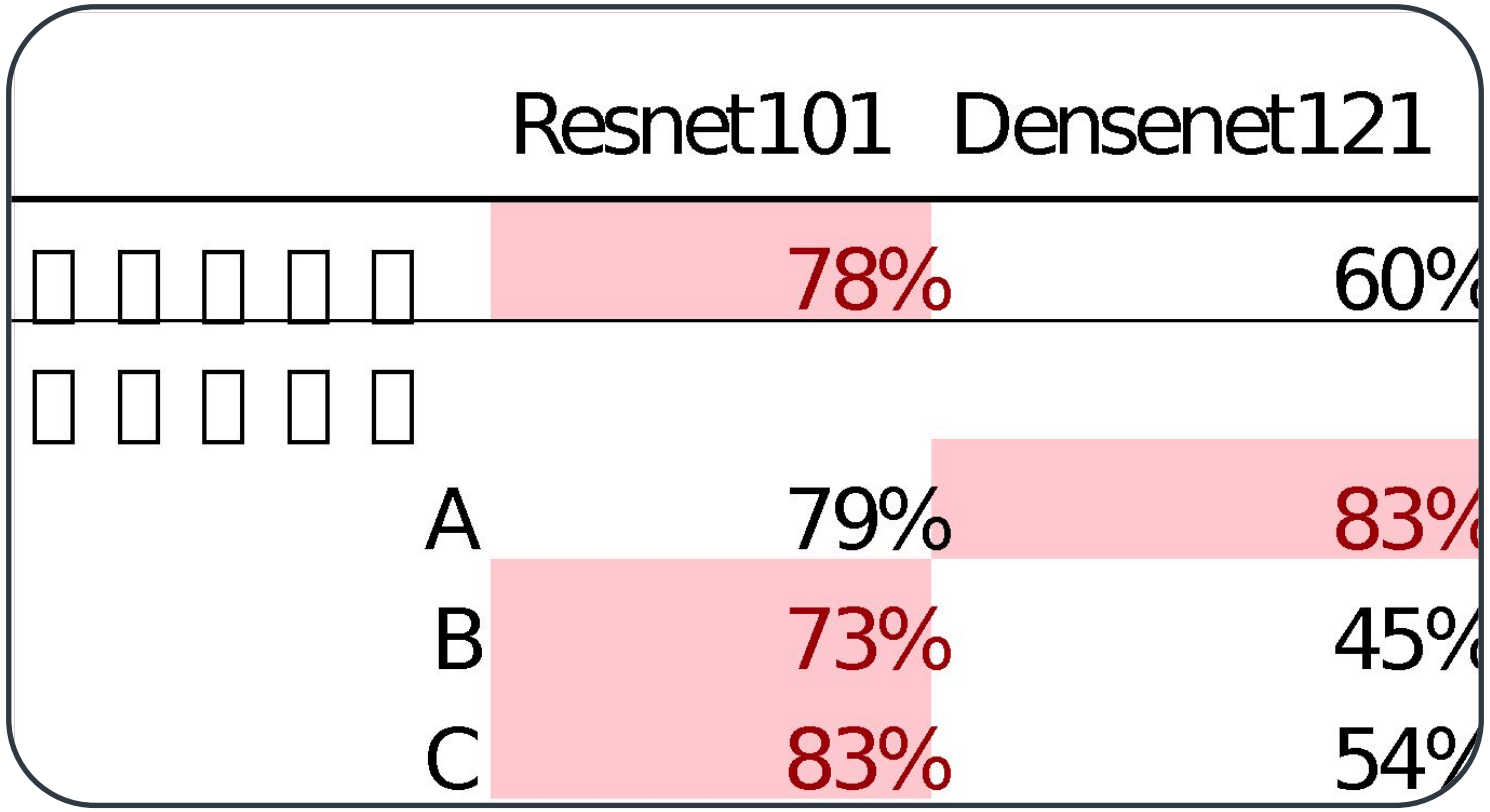
```
densenet121.classifier = nn.Linear(in_features=1024, out_features=3, bias=True)
```

Step2/ Compare Resnet101 with Densenet121





Step2/ Compare Resnet101 with Densenet121



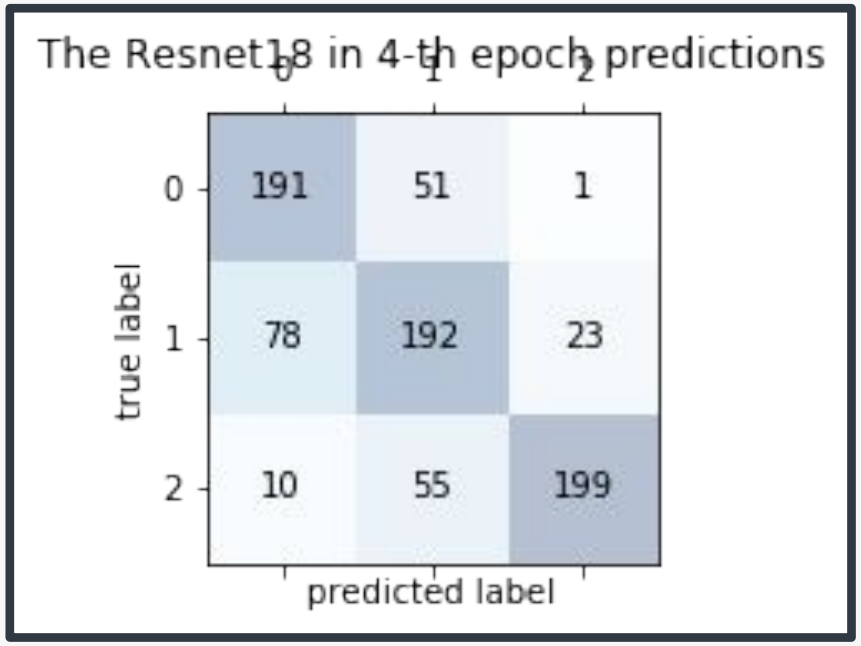
比較

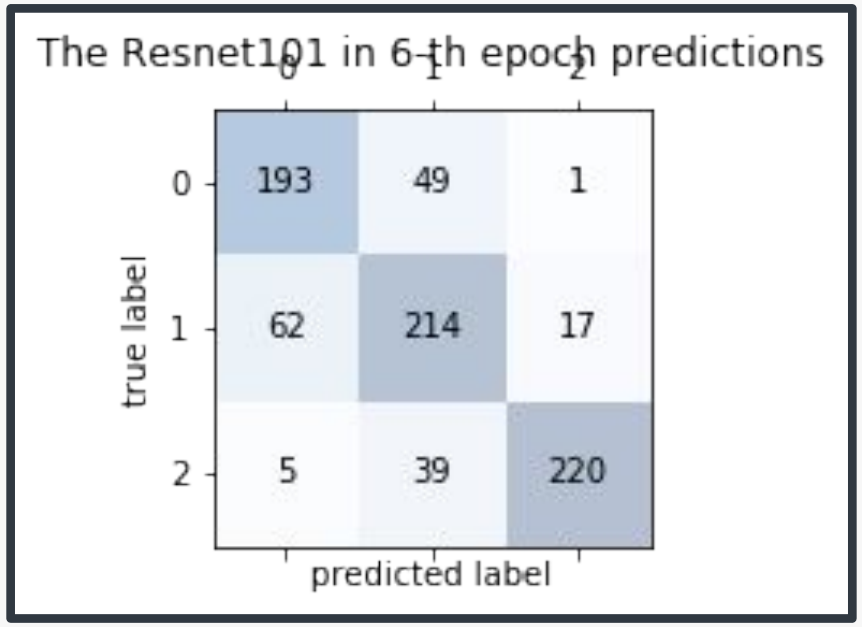
Resnet18

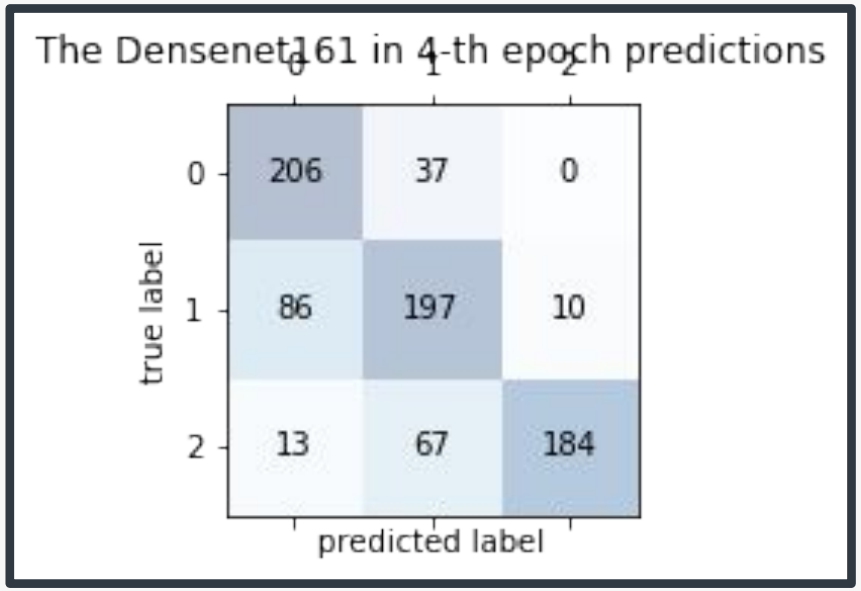
Resnet101

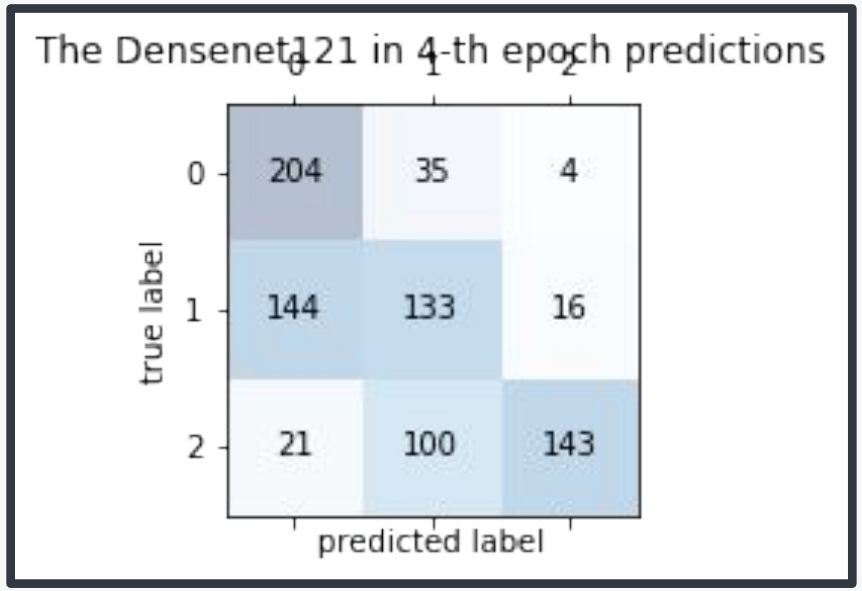
Densenet161

Densenet121


The Resnet18 in 4-th epoch predictions
0
1
2
0
1
2
191
51
1
78
192
23
10
55
199
true label
predicted label


The Resnet101 in 6-th epoch predictions
0
1
2
0
1
2
193
49
1
62
214
17
5
39
220
true label
predicted label


The Densenet161 in 4-th epoch predictions
0
1
2
0
1
2
206
37
0
86
197
10
13
67
184
true label
predicted label


The Densenet121 in 4-th epoch predictions
0
1
2
0
1
2
204
35
4
144
133
16
21
100
143
true label
predicted label

比較&總結

	Resnet18	Resnet101	Densenet161	Densenet121
整體準確率	73%	78%	73%	60%
個別準確率				
A	78%	79%	84%	83%
B	65%	73%	67%	45%
C	75%	83%	69%	54%
wi*準確率				
A	23.69%	24.00%	25.52%	25.21%
B	23.81%	26.74%	24.54%	16.48%
C	24.75%	27.39%	22.77%	17.82%
WAR	72.25%	78.12%	72.82%	59.51%

Thanks For Your Listening