機器視覺作業報告

HW4

姓名:岳紀伶
學號:01057113
日期:2024/06/11

方法

你**至少**可以說明下面項目:

(1)

1. 用efficientNet

   由於CCSN只有2543張圖片，選擇太大的模型會導致overfitting，因此選擇s，並在efficientNet的基礎上，加入30%的dropout

```
def build_efficientnet(class_number, trainable=True):
    efficientnet =
torchvision.models.efficientnet_v2_s(weights=torchvision.models.EfficientNet_V2_S_Weights.DEFAULT)

    for param in efficientnet.parameters():
        param.requires_grad = trainable

    num_ftrs = efficientnet.classifier[1].in_features
    efficientnet.classifier = nn.Sequential(
        nn.Dropout(0.3),
        torch.nn.Linear(num_ftrs, class_number)
    )

    return efficientnet
```

   由於選擇learning rate=1e-4會導致loss暴增，因此將learning rate改成1e-5，且若超過五次loss沒有下降就把learning rate乘以0.5

```
optimizer_ft = optim.Adam(efficientnet.parameters(),lr=1e-5)
scheduler = lr_scheduler.ReduceLROnPlateau(optimizer_ft,
mode='min', factor=0.5, patience=5, verbose=True)
trained_model, history = train_model(efficientnet,
criterion, optimizer_ft,

dataloaders={'train':trainloader,'val':validloader},

dataset_sizes={'train':X_train.shape[0],'val':X_valid.shape[0]},
                                         patience=patience,
                                         num_epochs=epochs,
                                         scheduler =
scheduler)
```

2. 用TNT
   在TNT的基礎上，加入50%的dropout

```
def build_tnt(class_number, trainable=True):
    tnt = timm.create_model('tnt_s_patch16_224',
pretrained=True)
    for param in tnt.parameters():
        param.requires_grad = trainable

    num_ftrs = tnt.head.in_features
    tnt.head = nn.Sequential(
        nn.Dropout(p=0.5),
        nn.Linear(num_ftrs, class_number)
    )

    return tnt
```

   藉由optimizer_ft選擇AdamW為optimizer，並設定learning rate為0.0001
   藉由scheduler設定如過超過十次loss沒有下降就把learning rate乘以0.1

```
    optimizer_ft = optim.AdamW(tnt.parameters(), lr=1e-4)
    scheduler = lr_scheduler.ReduceLROnPlateau(optimizer_ft,
mode='min', factor=0.1, patience=10, verbose=True)

    trained_model, history = train_model(tnt, criterion,
optimizer_ft,

dataloaders={'train': trainloader, 'val': validloader},

dataset_sizes={'train': X_train.shape[0], 'val':
X_valid.shape[0]},
                                         patience=patience,

scheduler=scheduler,
                                         num_epochs=epochs
                                         )
```

(2)
架構:用InceptionBlock + ResBlock

InceptionBlock:
參考GoogleNet，先產生2個1x1 捲積層，之後再分別經過5x5和3x3的捲積層，最後用kernel=3x3的MaxPool建立池化層，並調整大小到1x1
在完成上述步驟後加入residual減少運算量，也使網路能夠更深

```
class InceptionBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(InceptionBlock, self).__init__()

        self.branch1x1 = nn.Conv2d(in_channels, out_channels,
kernel_size=1)

        self.branch5x5_1 = nn.Conv2d(in_channels, out_channels,
kernel_size=1)
        self.branch5x5_2 = nn.Conv2d(out_channels, out_channels,
kernel_size=5, padding=2)

        self.branch3x3dbl_1 = nn.Conv2d(in_channels,
out_channels, kernel_size=1)
        self.branch3x3dbl_2 = nn.Conv2d(out_channels,
out_channels, kernel_size=3, padding=1)
        self.branch3x3dbl_3 = nn.Conv2d(out_channels,
out_channels, kernel_size=3, padding=1)

        self.branch_pool = nn.Conv2d(in_channels, out_channels,
kernel_size=1)

self.residual=nn.Conv2d(in_channels,out_channels*4,kernel_size=1)

    def forward(self, x):
        branch1x1 = self.branch1x1(x)

        branch5x5 = self.branch5x5_1(x)
        branch5x5 = self.branch5x5_2(branch5x5)

        branch3x3dbl = self.branch3x3dbl_1(x)
        branch3x3dbl = self.branch3x3dbl_2(branch3x3dbl)
        branch3x3dbl = self.branch3x3dbl_3(branch3x3dbl)
        branch_pool = nn.functional.max_pool2d(x, kernel_size=3,
stride=1, padding=1)
        branch_pool = self.branch_pool(branch_pool)
        outputs = [branch1x1, branch5x5, branch3x3dbl,
branch_pool]
        output = torch.cat(outputs, 1)

        residual = self.residual(x)
        return nn.functional.relu(output+residual)
```

ResBlock:
輸入經過kernel=3x3的卷積層->BatchNorm->relu->kernel=3x3的卷積->BatchNorm
最後加入shortcut減少運算量，也使網路能夠更深

```
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels,
kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels,
kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels,
kernel_size=1, stride=stride),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = self.relu(out)
        return out
```

CustomNet:
結合InceptionBlock 和 ResBlock，以一層InceptionBlock、一層ResBlock、一層MaxPool的形式往下訓練，在最後一層將層MaxPool改為層AvgPool

```
class CustomNet(nn.Module):
    def __init__(self, num_classes=11):
        super(CustomNet, self).__init__()

        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2,
padding=3)
        self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=2,
padding=1)

        self.inception1 = InceptionBlock(64, 32)
        self.resblock1 = ResBlock(128, 64)
        self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2,
padding=1)

        self.inception2 = InceptionBlock(64, 32)
        self.resblock2 = ResBlock(128, 64)
        self.maxpool3 = nn.MaxPool2d(kernel_size=3, stride=2,
padding=1)

        self.inception3 = InceptionBlock(64, 32)
        self.resblock3 = ResBlock(128, 64)
        self.maxpool4 = nn.MaxPool2d(kernel_size=3, stride=2,
padding=1)

        self.inception4 = InceptionBlock(64, 32)
        self.resblock4 = ResBlock(128, 64)
        self.maxpool5 = nn.MaxPool2d(kernel_size=3, stride=2,
padding=1)

        self.inception5 = InceptionBlock(64, 32)
        self.resblock5 = ResBlock(128, 64)
        self.maxpoo6 = nn.MaxPool2d(kernel_size=3, stride=2,
padding=1)

        self.inception6 = InceptionBlock(64, 32)
        self.resblock6 = ResBlock(128, 64)
        self.maxpool7 = nn.MaxPool2d(kernel_size=3, stride=2,
padding=1)
```

```python
        self.inception7 = InceptionBlock(64, 32)
        self.resblock7 = ResBlock(128, 64)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.drop=nn.Dropout(p=0.3)

        self.fc = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool1(x)

        x = self.inception1(x)
        x = self.resblock1(x)
        x = self.maxpool2(x)

        x = self.inception2(x)
        x = self.resblock2(x)
        x = self.maxpool3(x)

        x = self.inception3(x)
        x = self.resblock3(x)

        x = self.avgpool(x)
        #x = self.drop(x)

        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

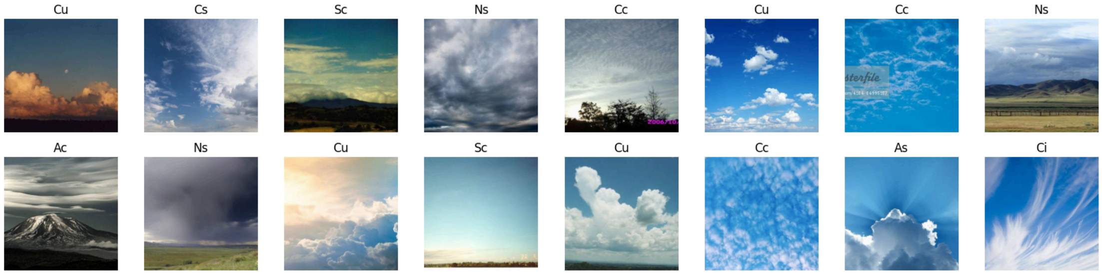
藉由optimizer_ft選擇AdamW為optimizer，並設定learning rate為0.0001
藉由scheduler設定如過超過五次loss沒有下降就把learning rate乘以0.1

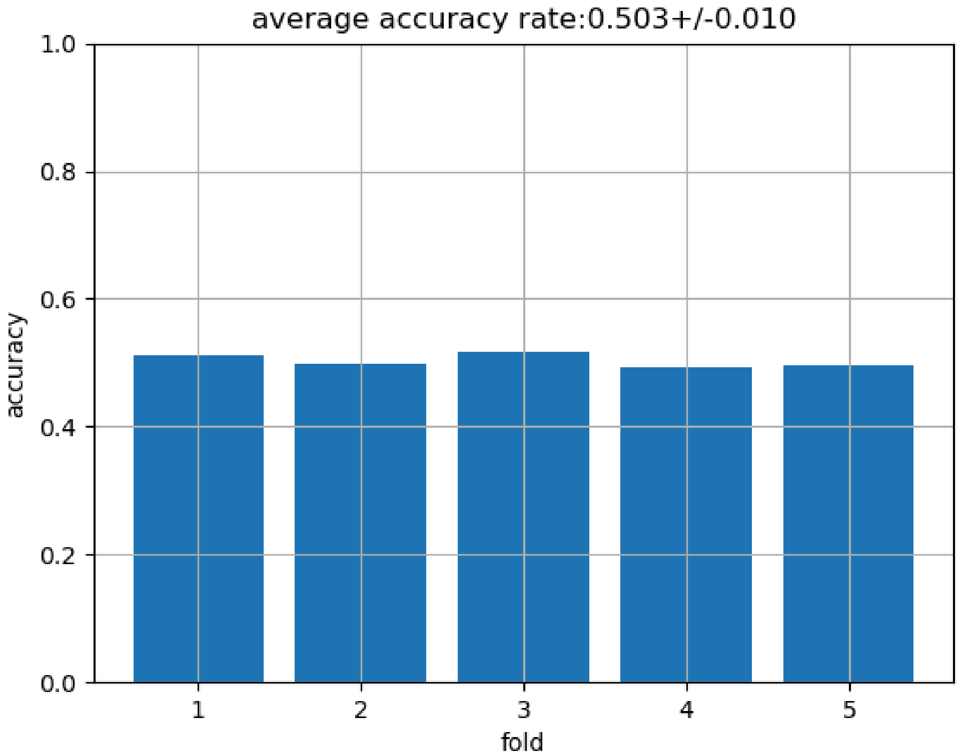
```
    optimizer_ft = optim.AdamW(model.parameters(), lr=1e-4)
    scheduler = lr_scheduler.ReduceLROnPlateau(optimizer_ft,
mode='min', factor=0.1, patience=5, verbose=True)

    trained_model, history = train_model(model, criterion,
optimizer_ft,
                                         dataloaders={'train':
trainloader, 'val': validloader},
                                         dataset_sizes={'train':
X_train.shape[0], 'val': X_valid.shape[0]},
                                         patience=patience,
                                         scheduler=scheduler,
                                         num_epochs=epochs
                                         )
```

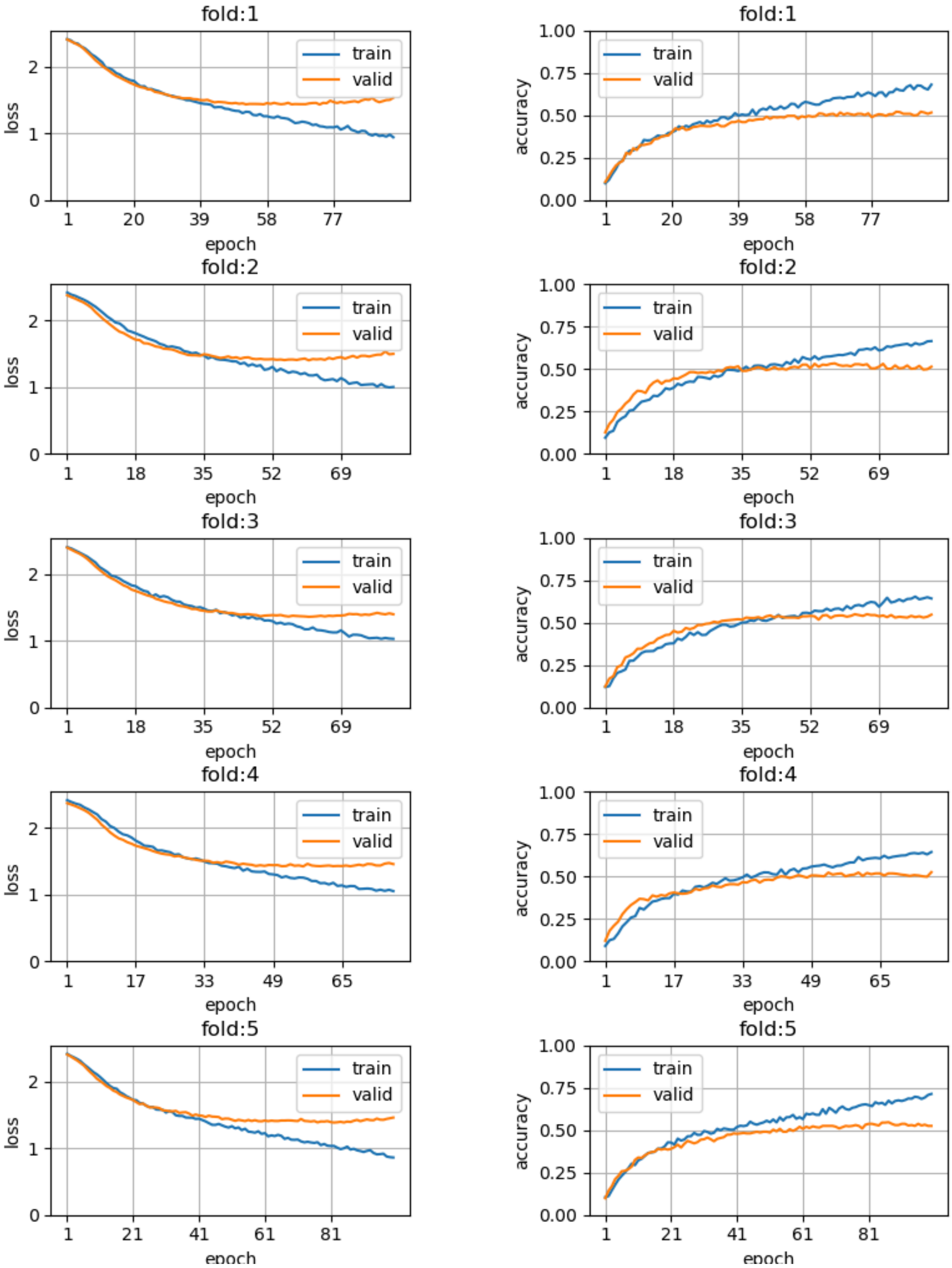
結果

(1)

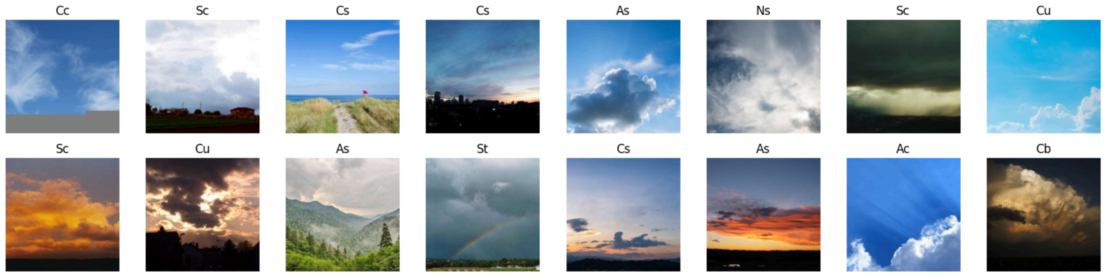
1. 在這個結果中雖然validation loss有在下降，但到最後仍有些微上升，若將learning rate在條小應該可以避免，但就會導致訓練速度過慢

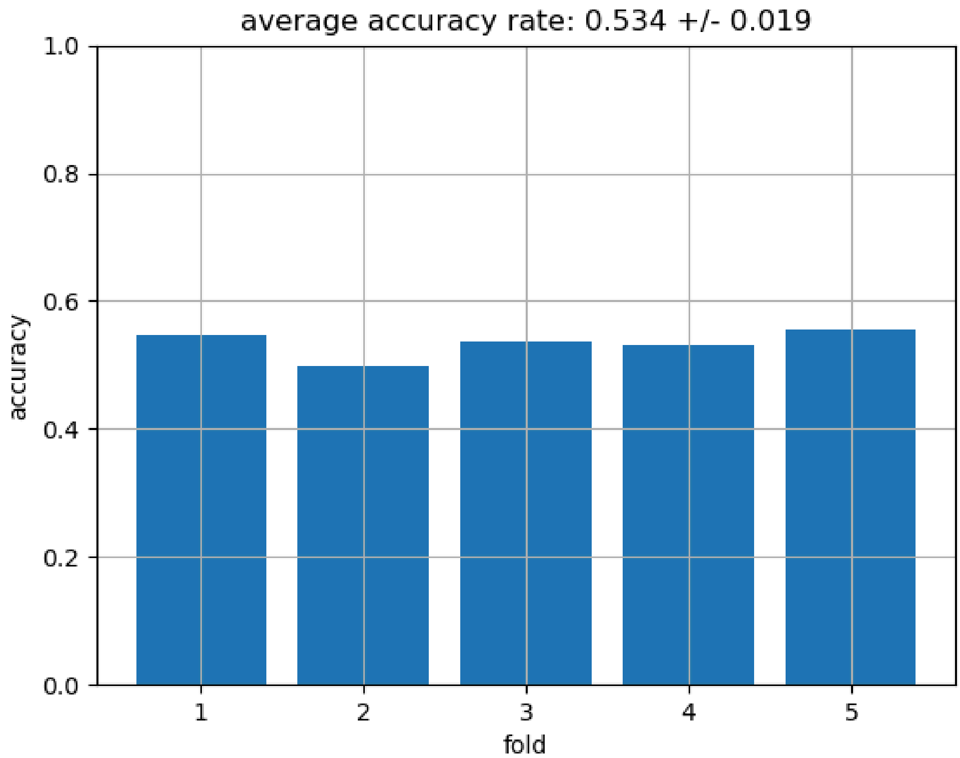


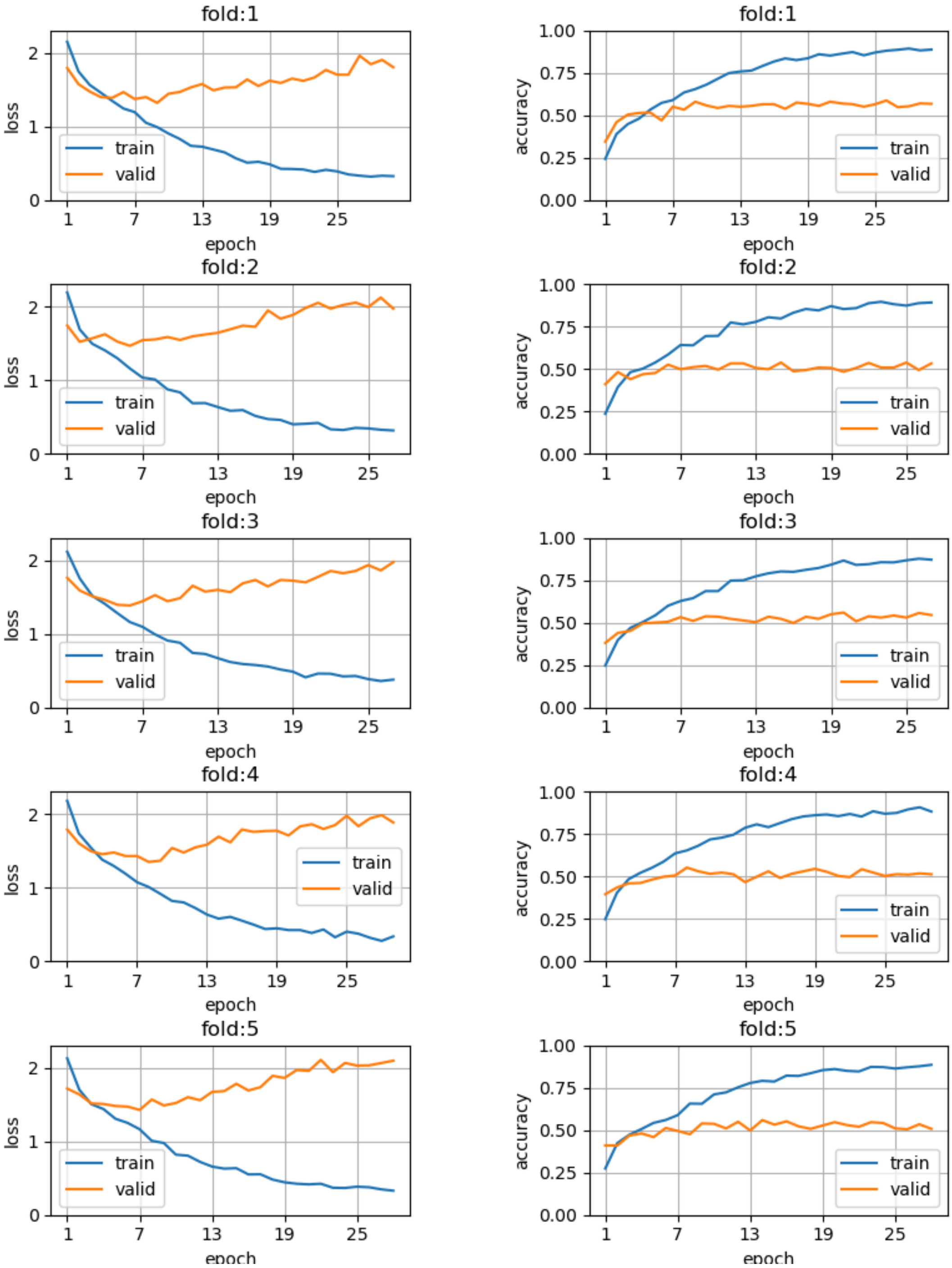



fold:1
train
valid
loss
epoch
1 20 39 58 77
0 1 2
accuracy
0.00 0.25 0.50 0.75 1.00
fold:2
1 18 35 52 69
fold:3
1 18 35 52 69
fold:4
1 17 33 49 65
fold:5
1 21 41 61 81

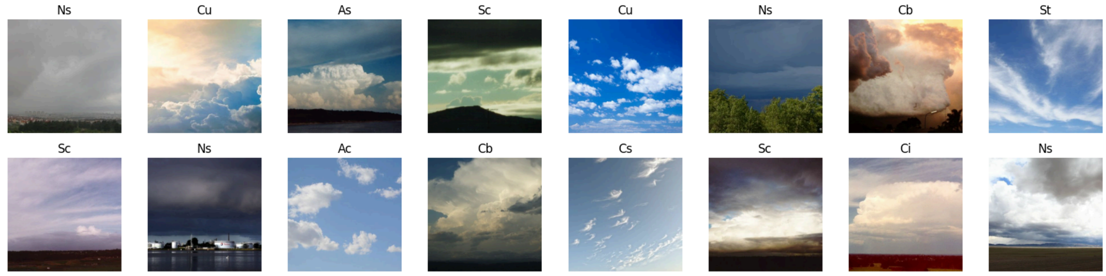
2. 在這個程式中雖然training loss一直在下降，但validation loss卻沒有，甚至還上升，表示這個預訓練模型對於這個data set來說可能過大。但正確率相較於ResNet高許多。

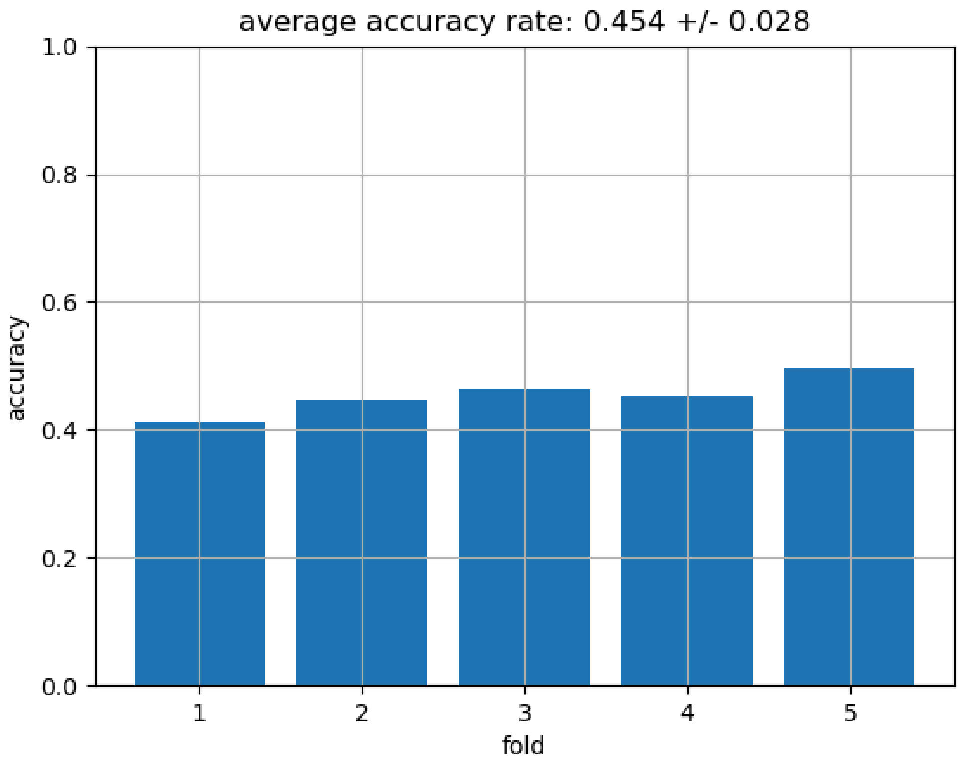


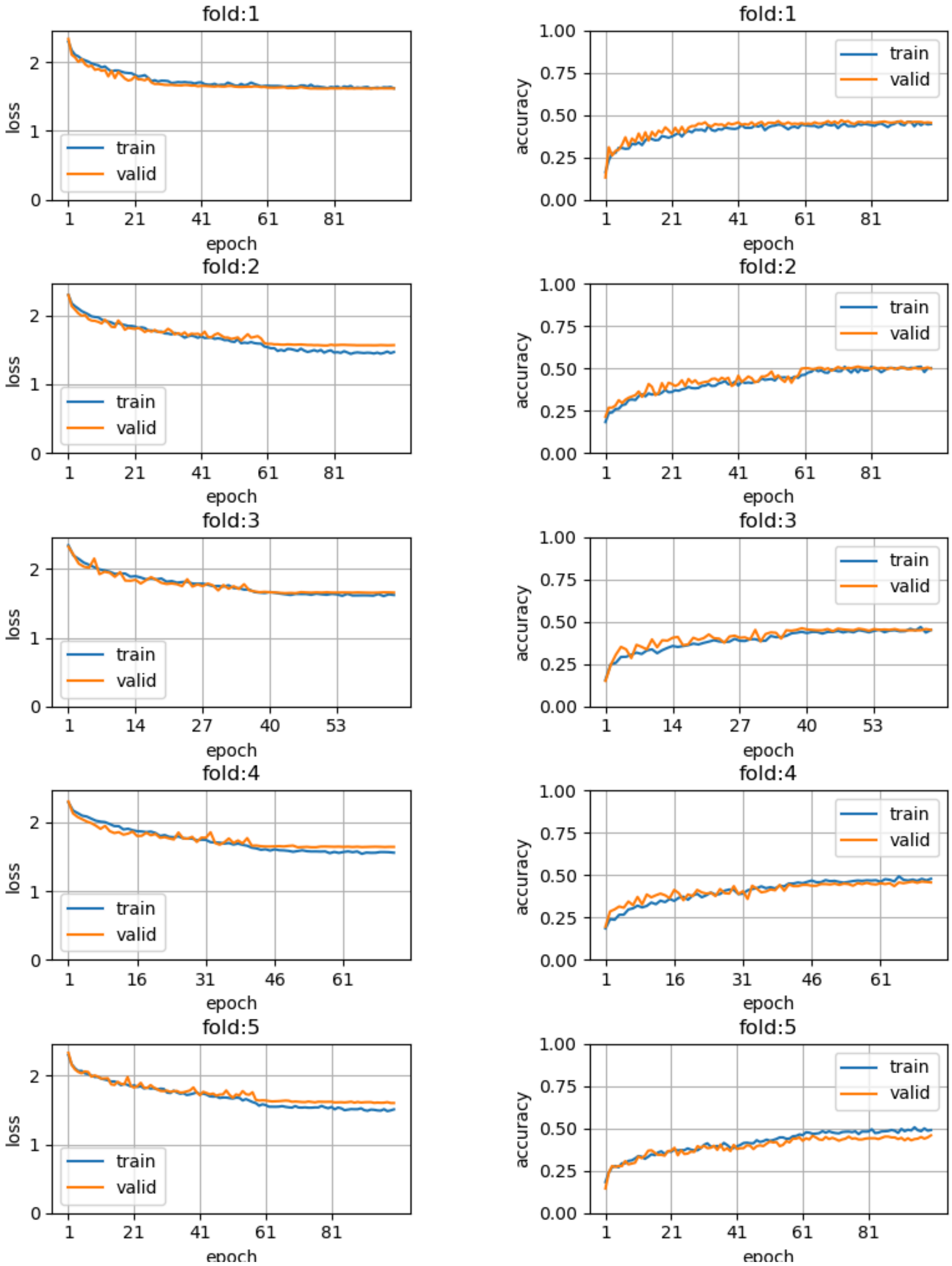



fold:1
fold:2
fold:3
fold:4
fold:5
loss
accuracy
epoch
train
valid
0
1
2
0.00
0.25
0.50
0.75
1.00
1
7
13
19
25

(2)
第二題由於是自己參考InceptionBlock 和 ResBlock設計，在效果上較預訓練模型來的差，且經過嘗試發現，以InceptionBlock 、ResBlock和MaxPool為一組，用兩組進行訓練效果最好，若是增加到六組反而效果沒那麼好。另外在learning rate為1e-4下效果也相對較好，若是改成1e-3 loss會突然增大，改成1e-5 loss則訓練速度較慢。除此之外，也發現trainig和validation的曲線幾乎相同。







結論

在這個作業中，我對於撰寫訓練神經網路進行分類有了更清楚的認知，並知道如何藉由調整參數、增加dropout、調整圖片等方法提升正確率，但因為對於神經網路中的參數數量計算方式仍不太熟悉，因此在寫題二題時花了許多時間調整每一層的大小，使他能夠順利運行，希望之後能夠盡快熟悉。

參考文獻

https://blog.csdn.net/weixin_40100431/article/details/84311430

https://ithelp.ithome.com.tw/m/articles/10337660

https://blog.csdn.net/liu24244/article/details/106206612

https://blog.csdn.net/qq_39777550/article/details/108038677

https://ithelp.ithome.com.tw/articles/10265328

https://blog.paperspace.com/popular-deep-learning-architectures-densenet-mnasnet-shufflenet/

https://ithelp.ithome.com.tw/articles/10205210

https://medium.com/ching-i/inception-%E7%B3%BB%E5%88%97-inceptionv4-inception-resnet-v1-inception-resnet-v2-42be5d23b2ec

https://blog.csdn.net/qq_43391414/article/details/127227029

https://medium.com/ching-i/%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1-cnn-%E7%B6%93%E5%85%B8%E6%A8%A1%E5%9E%8B-googlelenet-resnet-densenet-with-pytorch-code-1688015808d9

https://medium.com/ai-blog-tw/deep-learning-residual-leaning-%E8%AA%8D%E8%AD%98resnet%E8%88%87%E4%BB%96%E7%9A%84%E5%86%A0%E5%90%8D%E5%BE%8C%E7%B9%BC%E8%80%85resnext-resnest-6bedf9389ce

https://blog.csdn.net/ytusdc/article/details/119823301