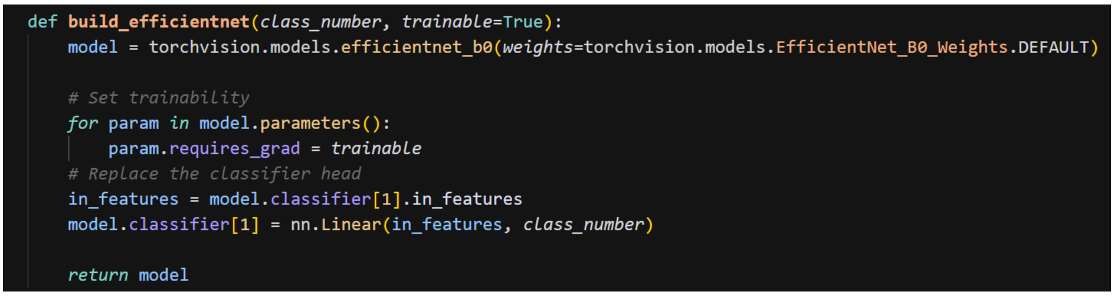
機器視覺作業報告

image classification

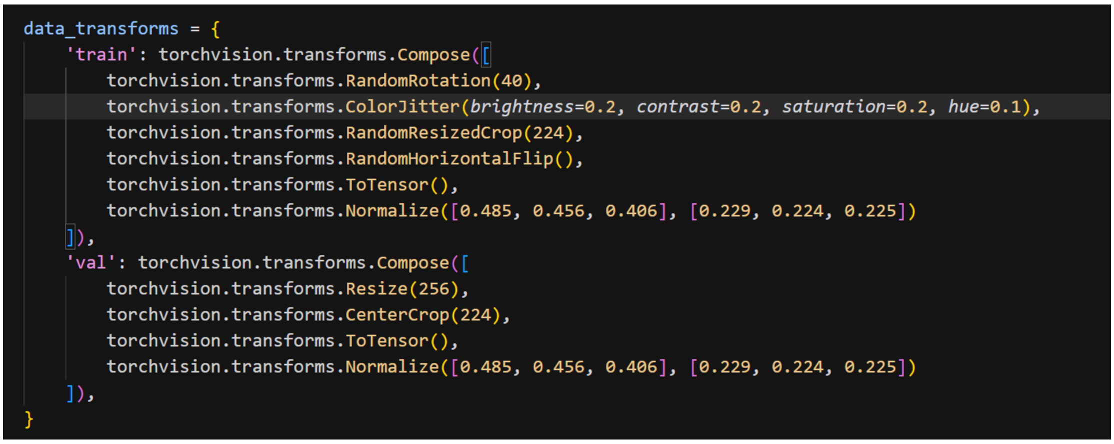
姓名:邱顯成
學號:01157007
日期:114/6/9

方法



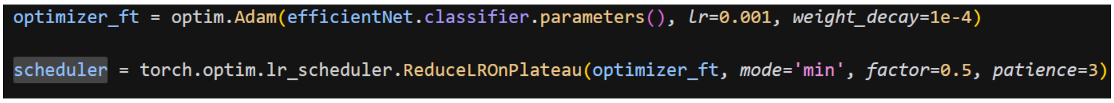
```python
def build_efficientnet(class_number, trainable=True):
    model = torchvision.models.efficientnet_b0(weights=torchvision.models.EfficientNet_B0_Weights.DEFAULT)

    # Set trainability
    for param in model.parameters():
        param.requires_grad = trainable
    # Replace the classifier head
    in_features = model.classifier[1].in_features
    model.classifier[1] = nn.Linear(in_features, class_number)

    return model
```

建立 efficientnet，並且把分類層的輸出改為 11(class_number)



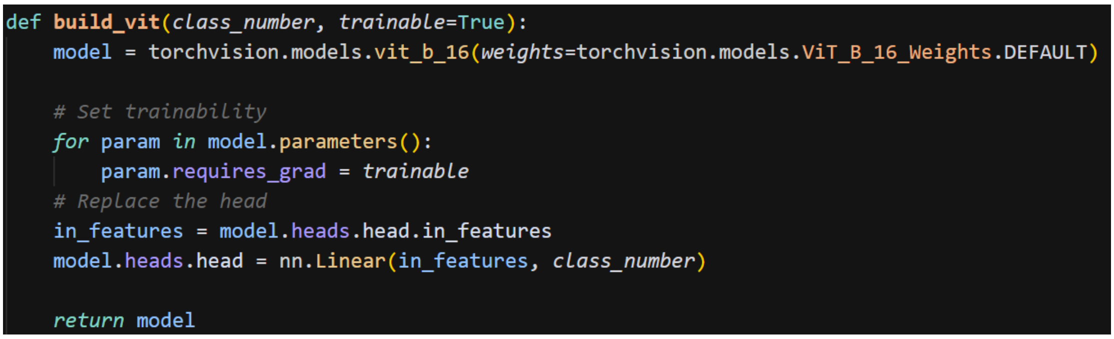
```python
data_transforms = {
    'train': torchvision.transforms.Compose([
        torchvision.transforms.RandomRotation(40),
        torchvision.transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
        torchvision.transforms.RandomResizedCrop(224),
        torchvision.transforms.RandomHorizontalFlip(),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': torchvision.transforms.Compose([
        torchvision.transforms.Resize(256),
        torchvision.transforms.CenterCrop(224),
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

DataTransform:這裡除了用老師原本有的，額外加上 ColorJitter，我認為資料集很多圖片都有光線，加上 ColorJitter 能夠讓模型看過不同顏色的樣本。



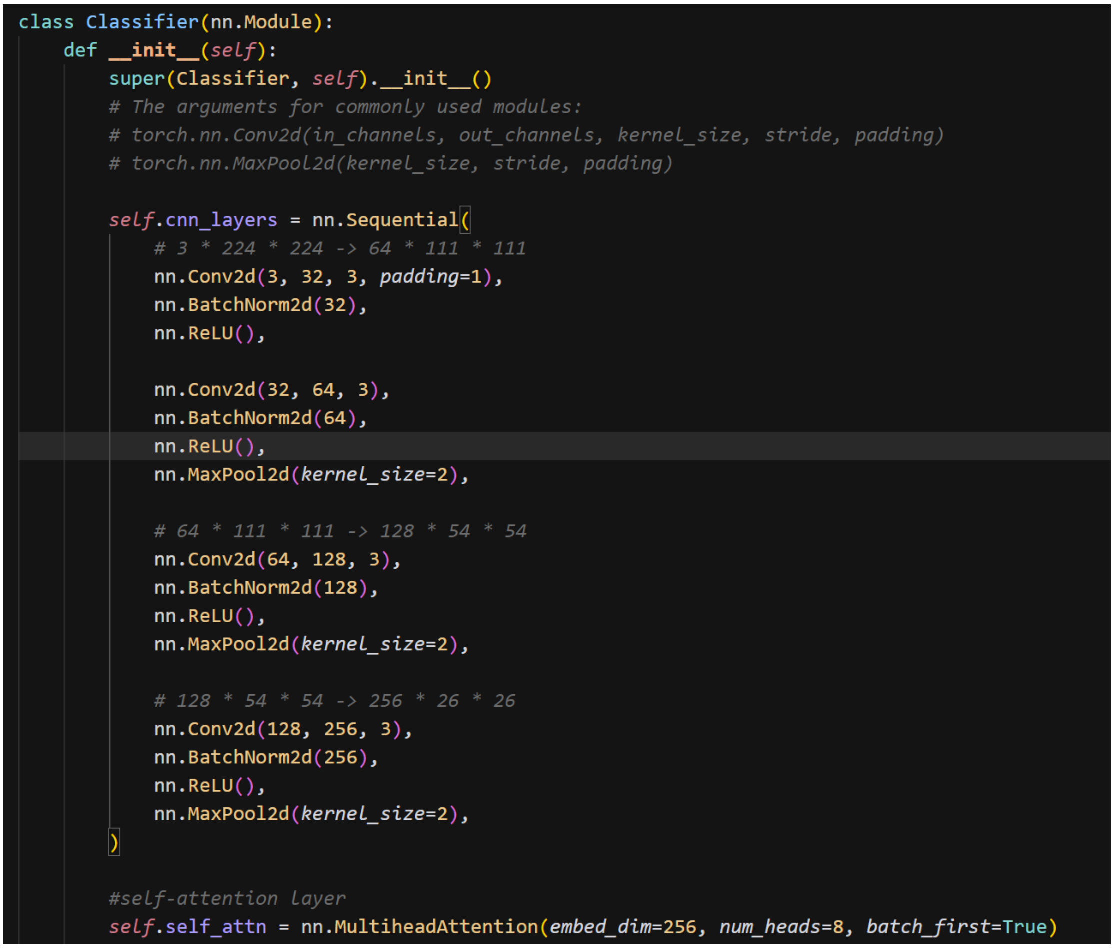
```python
optimizer_ft = optim.Adam(efficientNet.classifier.parameters(), lr=0.001, weight_decay=1e-4)

scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer_ft, mode='min', factor=0.5, patience=3)
```

只把 classifier layer 傳到優化器裡，只訓練 classifier layer。同時也加上 learning rate scheduler，當模型連續訓練 n 次沒有更好，就進行一次學習率衰降。

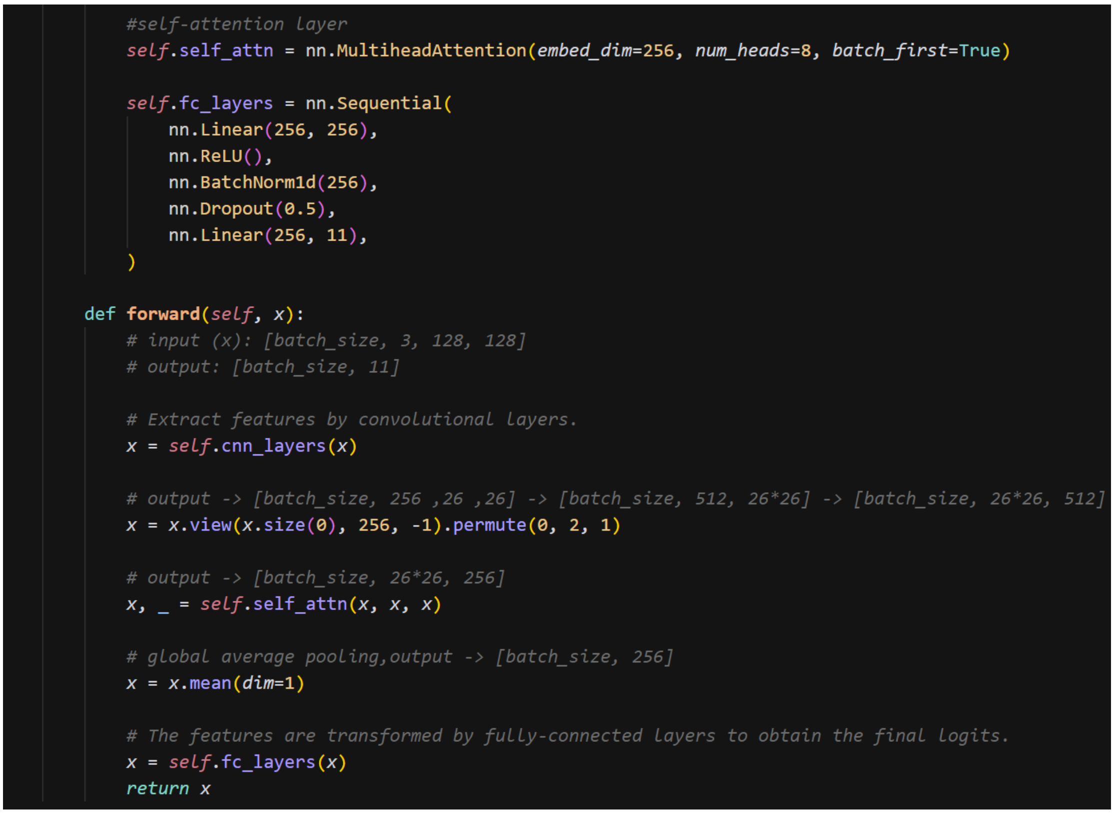


```python
def build_vit(class_number, trainable=True):
    model = torchvision.models.vit_b_16(weights=torchvision.models.ViT_B_16_Weights.DEFAULT)

    # Set trainability
    for param in model.parameters():
        param.requires_grad = trainable
    # Replace the head
    in_features = model.heads.head.in_features
    model.heads.head = nn.Linear(in_features, class_number)

    return model
```

建立 vit，並把 head layer 輸出改為 11(class_number)



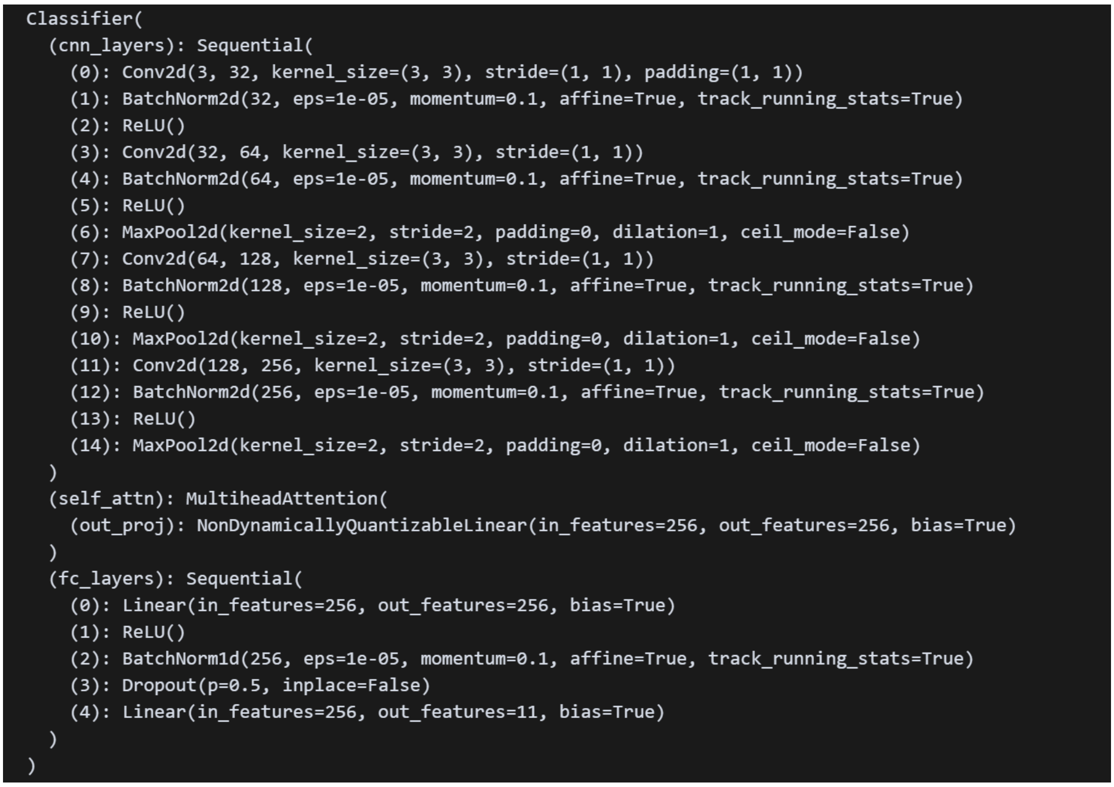
```python
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        # The arguments for commonly used modules:
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
        # torch.nn.MaxPool2d(kernel_size, stride, padding)

        self.cnn_layers = nn.Sequential(
            # 3 * 224 * 224 -> 64 * 111 * 111
            nn.Conv2d(3, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.Conv2d(32, 64, 3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),

            # 64 * 111 * 111 -> 128 * 54 * 54
            nn.Conv2d(64, 128, 3),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),

            # 128 * 54 * 54 -> 256 * 26 * 26
            nn.Conv2d(128, 256, 3),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )

        #self-attention layer
        self.self_attn = nn.MultiheadAttention(embed_dim=256, num_heads=8, batch_first=True)
```



```python
        #self-attention layer
        self.self_attn = nn.MultiheadAttention(embed_dim=256, num_heads=8, batch_first=True)

        self.fc_layers = nn.Sequential(
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.BatchNorm1d(256),
            nn.Dropout(0.5),
            nn.Linear(256, 11),
        )

    def forward(self, x):
        # input (x): [batch_size, 3, 128, 128]
        # output: [batch_size, 11]

        # Extract features by convolutional layers.
        x = self.cnn_layers(x)

        # output -> [batch_size, 256 ,26 ,26] -> [batch_size, 512, 26*26] -> [batch_size, 26*26, 512]
        x = x.view(x.size(0), 256, -1).permute(0, 2, 1)

        # output -> [batch_size, 26*26, 256]
        x, _ = self.self_attn(x, x, x)

        # global average pooling,output -> [batch_size, 256]
        x = x.mean(dim=1)

        # The features are transformed by fully-connected layers to obtain the final logits.
        x = self.fc_layers(x)
        return x
```

自行建立了 4 層的 CNN 後面再接一層 self-attention layer。用 MaxPooling 逐層降

低特徵圖尺寸，壓縮資訊。

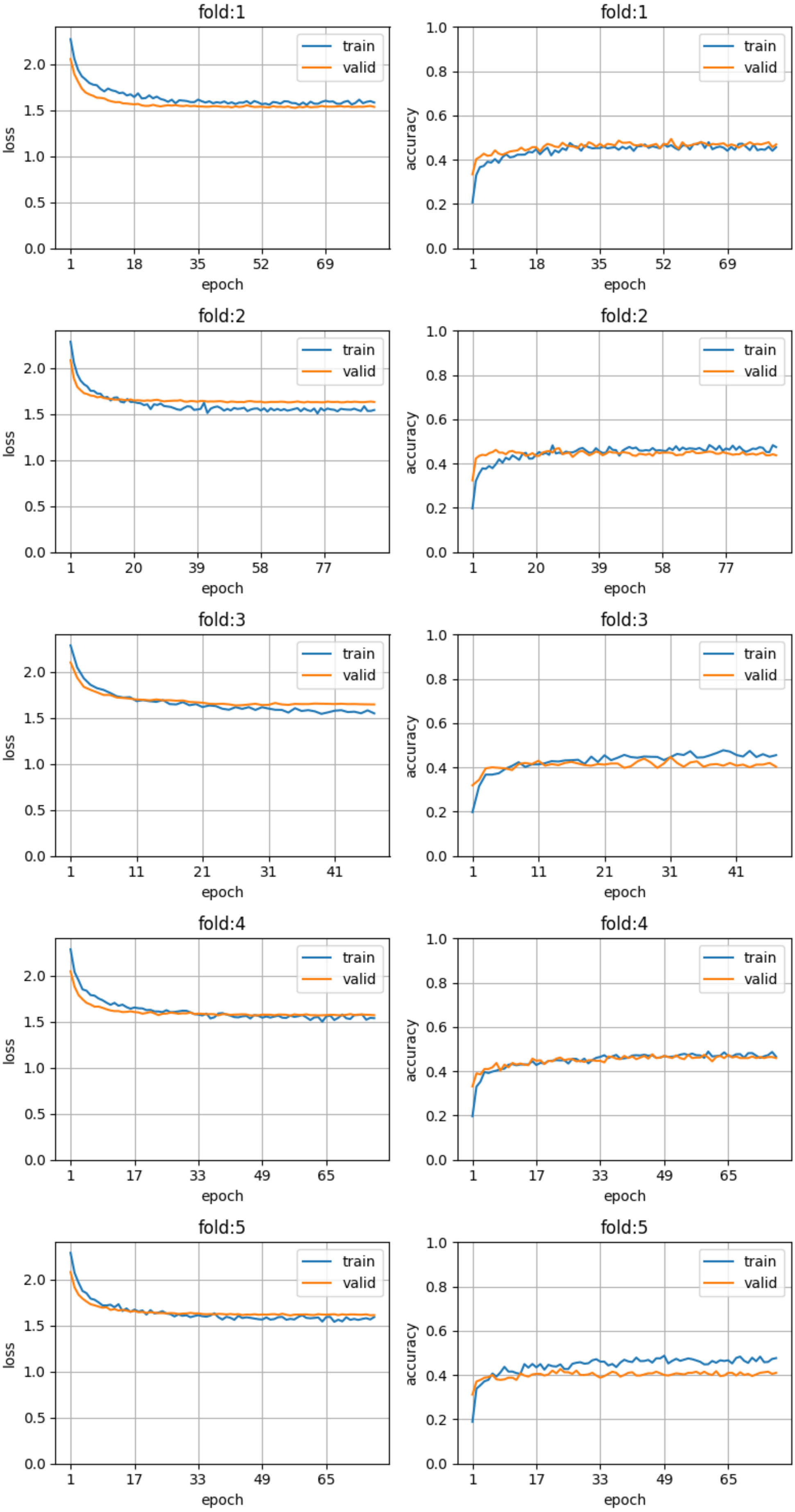


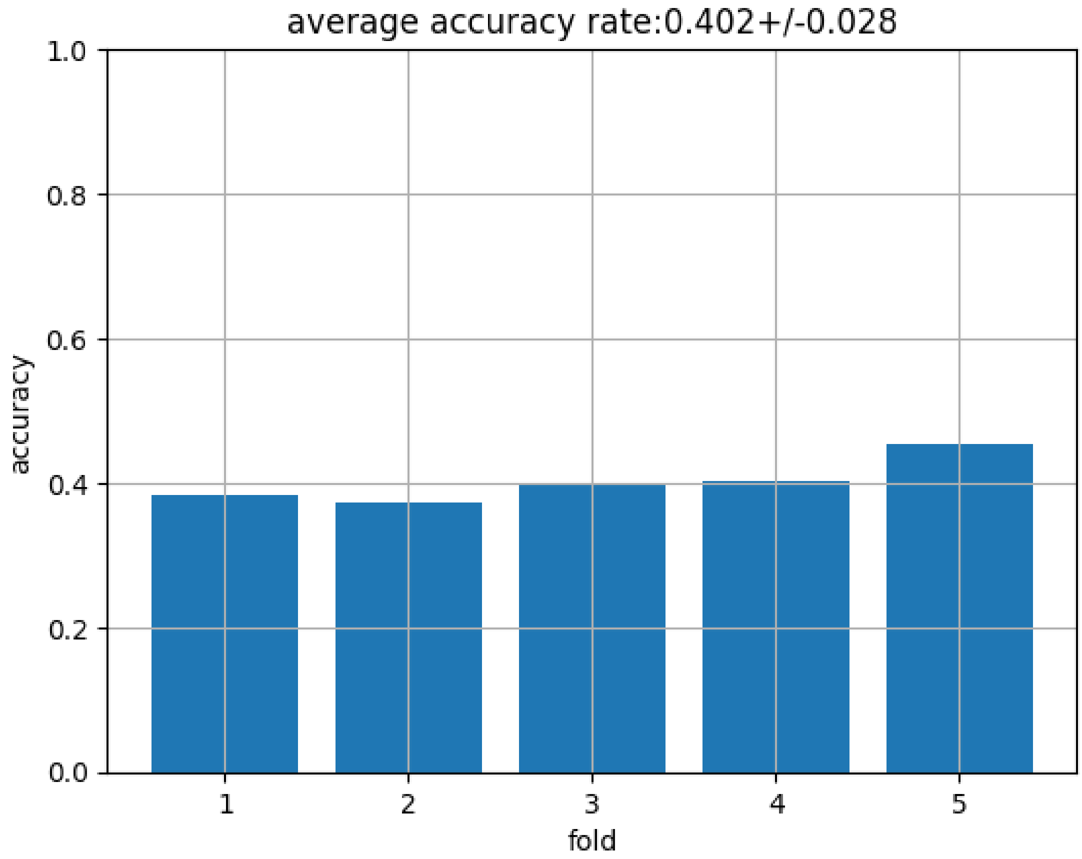
```
Classifier(
  (cnn_layers): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU()
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (self_attn): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256, bias=True)
  )
  (fc_layers): Sequential(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): ReLU()
    (2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=11, bias=True)
  )
)
```

模型的架構

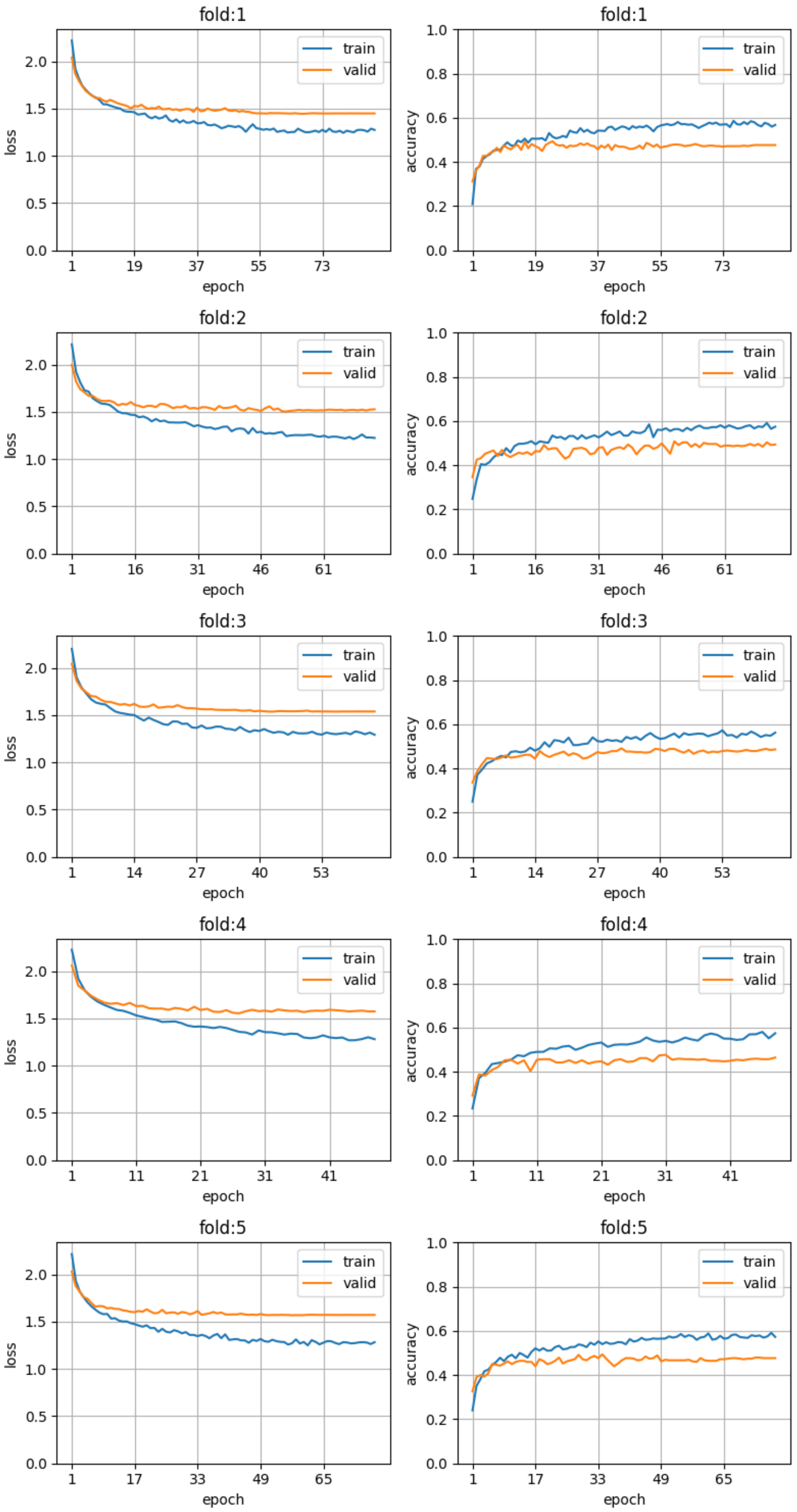
結果

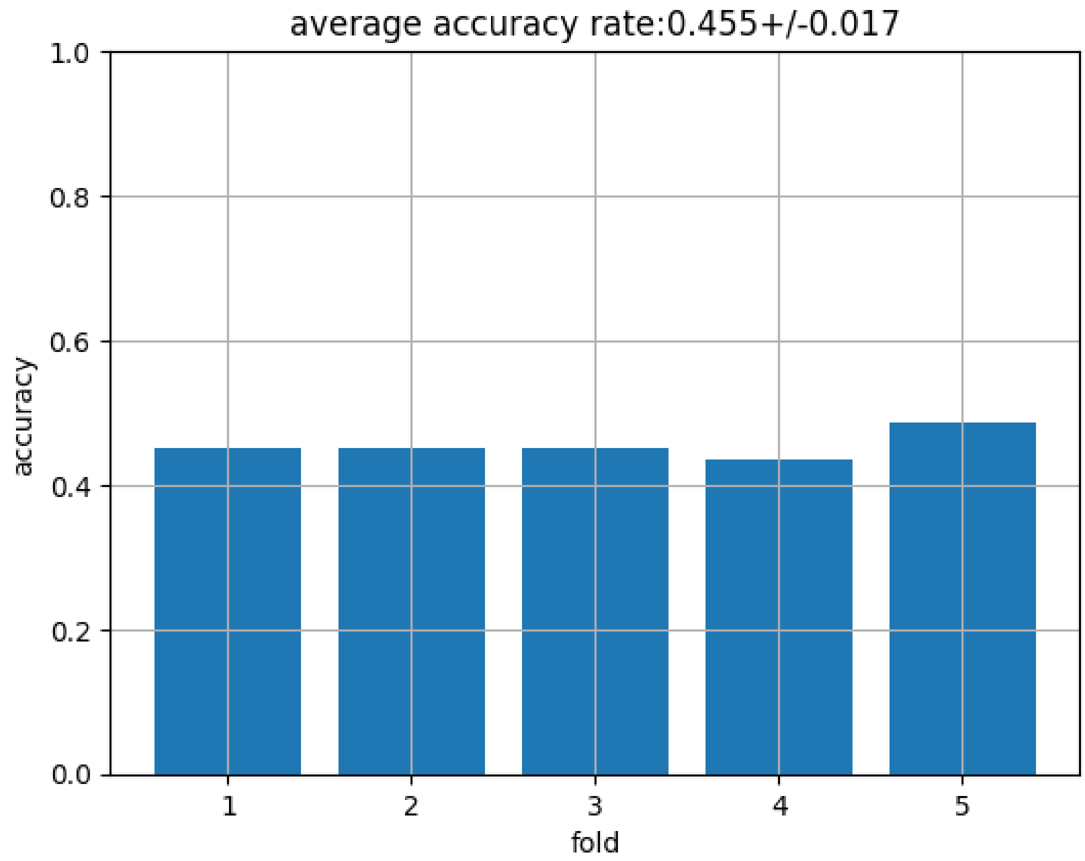
Efficientnet: opt:Adam, batch_size=64, epoch=100, lr=0.001


fold:1
loss
epoch
1
18
35
52
69
2.0
1.5
1.0
0.5
0.0
train
valid
fold:1
accuracy
epoch
1
18
35
52
69
1.0
0.8
0.6
0.4
0.2
0.0
train
valid
fold:2
loss
epoch
1
20
39
58
77
2.0
1.5
1.0
0.5
0.0
train
valid
fold:2
accuracy
epoch
1
20
39
58
77
1.0
0.8
0.6
0.4
0.2
0.0
train
valid
fold:3
loss
epoch
1
11
21
31
41
2.0
1.5
1.0
0.5
0.0
train
valid
fold:3
accuracy
epoch
1
11
21
31
41
1.0
0.8
0.6
0.4
0.2
0.0
train
valid
fold:4
loss
epoch
1
17
33
49
65
2.0
1.5
1.0
0.5
0.0
train
valid
fold:4
accuracy
epoch
1
17
33
49
65
1.0
0.8
0.6
0.4
0.2
0.0
train
valid
fold:5
loss
epoch
1
17
33
49
65
2.0
1.5
1.0
0.5
0.0
train
valid
fold:5
accuracy
epoch
1
17
33
49
65
1.0
0.8
0.6
0.4
0.2
0.0
train
valid

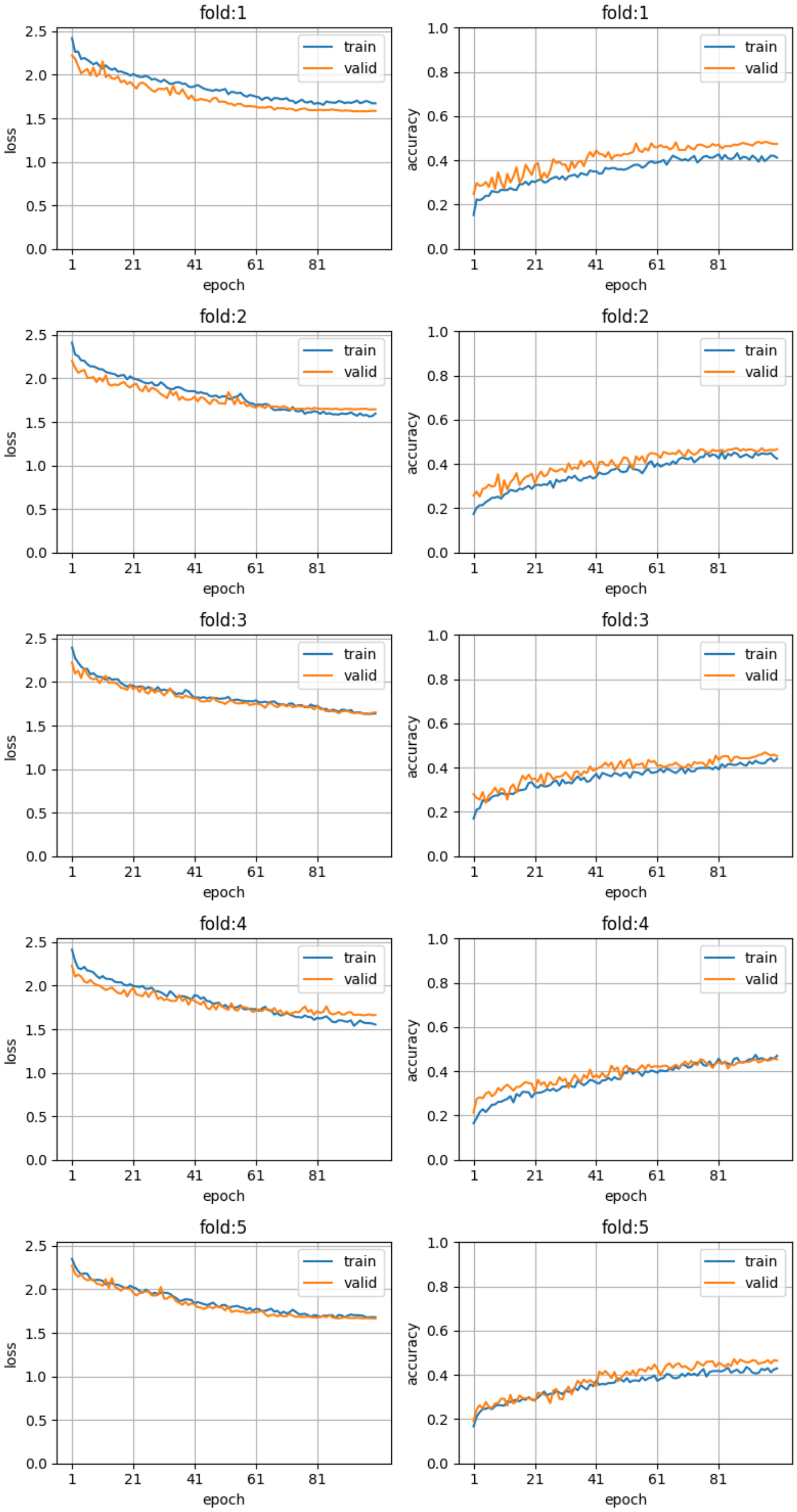


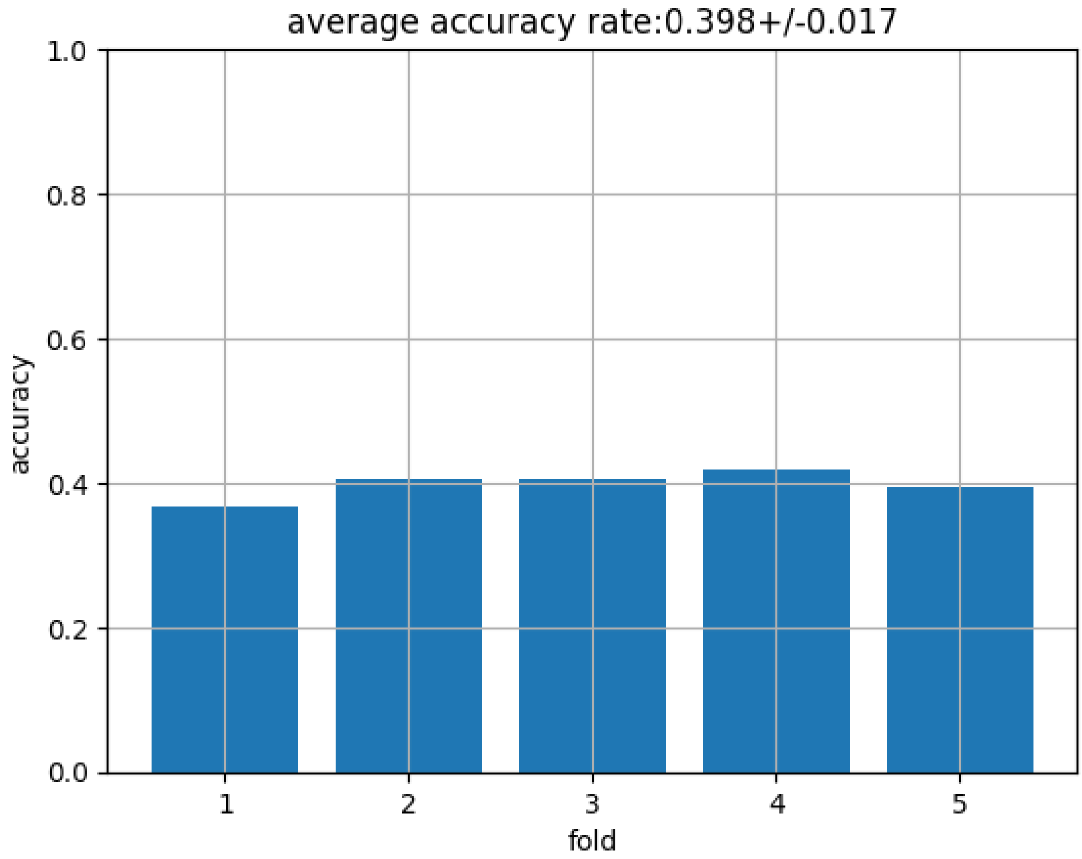
VIT: opt=Adam, batch_size=64, epoch=100, lr=0.001


fold:1
loss
epoch
train
valid
0.0
0.5
1.0
1.5
2.0
1
19
37
55
73
accuracy
0.2
0.4
0.6
0.8
1.0
fold:2
16
31
46
61
fold:3
14
27
40
53
fold:4
11
21
31
41
fold:5
17
33
49
65

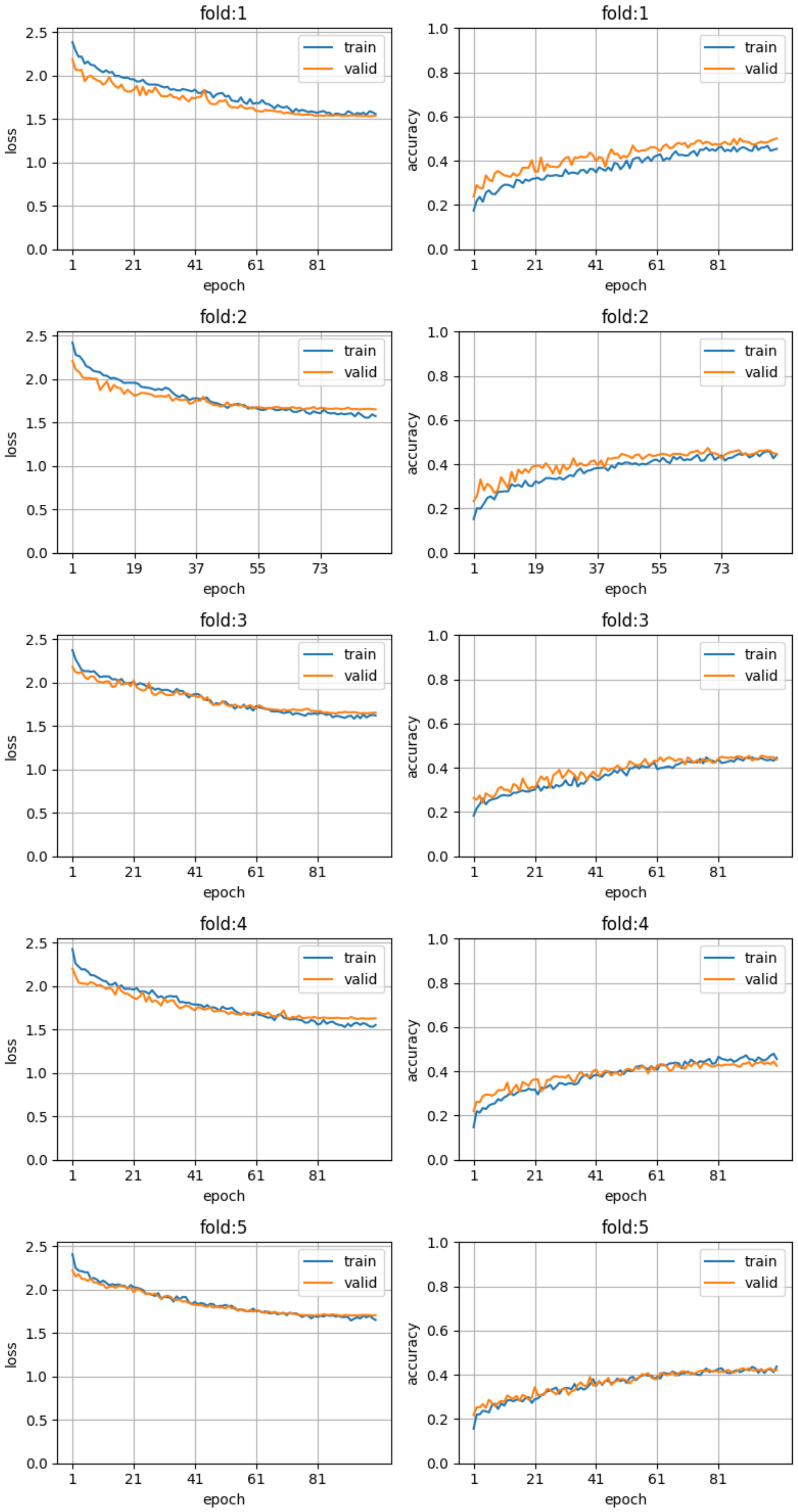


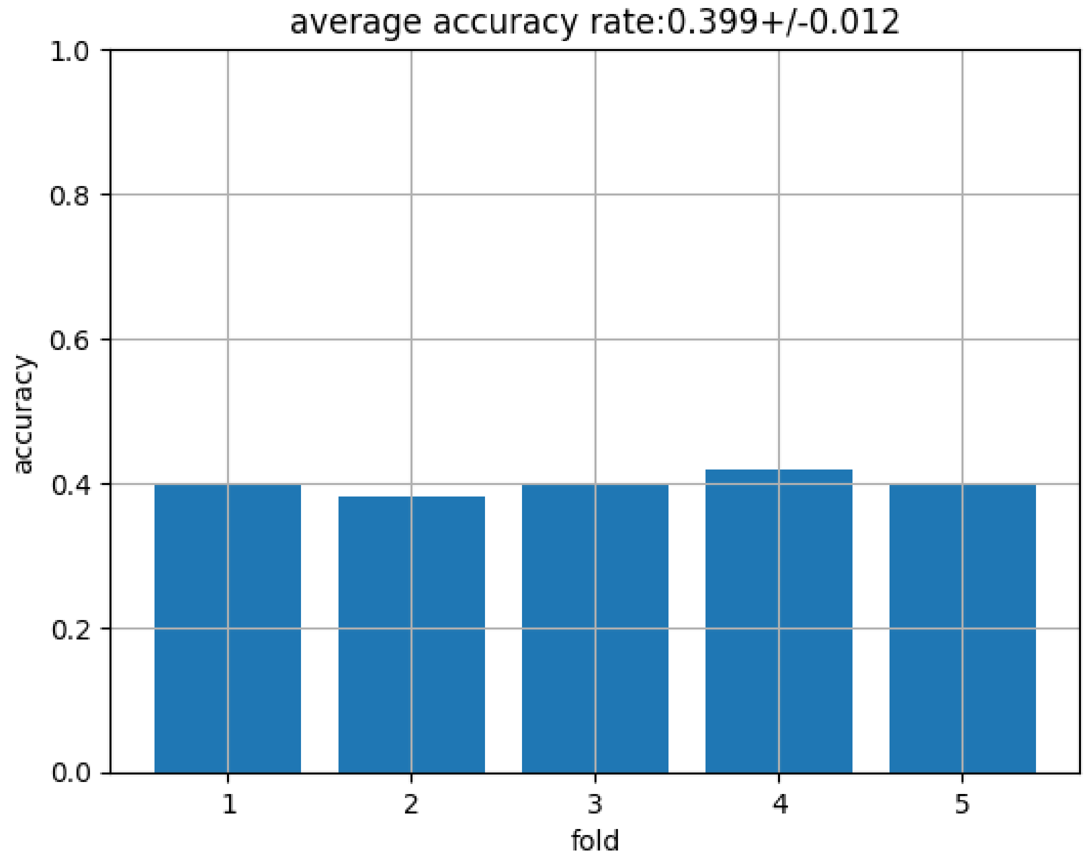
MyClassifier: opt=Adam, batch_size=64, epoch=100, lr=0.001,attentionHead=8


fold:1
fold:1
fold:2
fold:2
fold:3
fold:3
fold:4
fold:4
fold:5
fold:5
train
valid
loss
accuracy
epoch
1
21
41
61
81
0.0
0.5
1.0
1.5
2.0
2.5
0.0
0.2
0.4
0.6
0.8
1.0



MyClassifier: opt=Adam, batch_size=64, epoch=100, lr=0.001,attentionHead=4


fold:1
loss
2.5
2.0
1.5
1.0
0.5
0.0
1
21
41
61
81
epoch
train
valid
fold:1
accuracy
1.0
0.8
0.6
0.4
0.2
0.0
1
21
41
61
81
epoch
train
valid
fold:2
loss
1
19
37
55
73
epoch
train
valid
fold:2
accuracy
1
19
37
55
73
epoch
train
valid
fold:3
loss
1
21
41
61
81
epoch
train
valid
fold:3
accuracy
1
21
41
61
81
epoch
train
valid
fold:4
loss
1
21
41
61
81
epoch
train
valid
fold:4
accuracy
1
21
41
61
81
epoch
train
valid
fold:5
loss
1
21
41
61
81
epoch
train
valid
fold:5
accuracy
1
21
41
61
81
epoch
train
valid



結論

其實之前就有自己架過神經網路的經驗，CNN 的倒是第一次，學到最多的就是圖片經過這一層大小變得如何，要算 kernel 和 stride，期末考出這題我馬上秒答，接著要自己設置各種超參數來實驗，過程其實蠻花時間的。結論上來看 VIT 的準確率是最高的，但是訓練時間是第二久的，花了 3 個小時， 自己建立的架構訓練了 8 個小時，即使把 attention 層凍結住還是花很多時間，後來有把 head 調低，訓練時間並沒有大幅降，而且準確率也沒有那些 pretrain model 來得好。

參考文獻

【第 12 天】訓練模型-Learning Rate - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天

Convolutional Neural Network