1. Compare resnet18 with and without pretrained

With pretrained：

- 預訓練的 resnet18 模型已經在大型數據上學習了通用的特徵，通常是 ImageNet，也就是模型已經知道了圖中常出現的各種形狀、紋理等信息，如此有利於提早達到期望的 training data accuracy。
- 由於預訓練的 resnet18 模型會具有良好的權重初始化，因此能較快的收斂，省去了前期訓練的時間和資源。
- 遷移學習：可以將預訓練的 resnet18 模型當成特徵提取器，並在其尾端加入額外的層，以達到相對應的目標。

Without pretrained：

- 從頭開始訓練通常要更多的 training data，因為模型需要自己學習特徵，所以這種方法需要有大量的數據、充足的計算資源、足夠的迭代次數，才較為合適。
- 若最終目標和預訓練的方向差異非常大，即可利用這種方法自行定義模型的初始權重，使其更符合最終目標的特徵，如此一來，達到的效果或許會比預訓練 model 來得好。

2. Screenshot of task1 (>75% accuracy)

```python
class RN18(nn.Module):
    def __init__(self, num_classes=4):
        super(RN18, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self.make_layer(64, 2)
        self.layer2 = self.make_layer(128, 2, stride=2)
        self.layer3 = self.make_layer(256, 2, stride=2)
        self.layer4 = self.make_layer(512, 2, stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)

    def make_layer(self, out_channels, blocks, stride=1):
        layers = [BasicBlock(self.in_channels, out_channels, stride)]
        self.in_channels = out_channels
        for _ in range(1, blocks):
            layers.append(BasicBlock(out_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

- make_layer 是用來創建 BasicBlock 層。

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

        # If the dimensions of input and output feature maps don't match, use a shortcut connection
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.shortcut = nn.Identity()

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

- 條件判斷：若 stride!=1 或 input channels!=output channels，則進入 shortcut，以確保 weights size 一致，同時減輕梯度消失的問題。

jupyter Lab2_ResNet18 Last Checkpoint: 1 小時前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

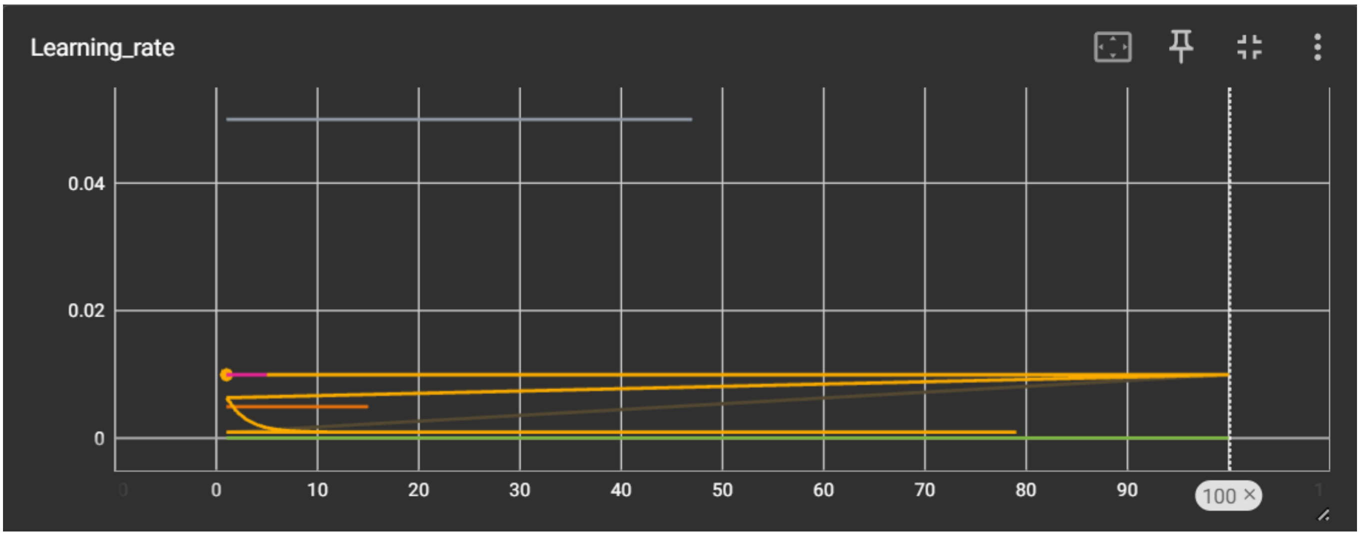
Run Code

```python
for epoch in range(1, epochs+1):

    print('epoch:', epoch)
    train(model, criterion, optimizer)
    accuracy = valid(model, criterion)

    if accuracy > acc_best:
        acc_best = accuracy
        print("model saved")
        # save the model
        torch.save(model, "model.pth")
```

```
--------------start training--------------
epoch: 1
Training Accuracy: 72.9237% Training Loss: 0.7074
Validation Accuracy: 63.0303% Validation Loss: 1.3662
model saved
epoch: 2
Training Accuracy: 71.7758% Training Loss: 0.7662
Validation Accuracy: 70.3030% Validation Loss: 0.7243
model saved
epoch: 3
Training Accuracy: 77.6502% Training Loss: 0.6253
Validation Accuracy: 43.0303% Validation Loss: 2.8168
epoch: 4
Training Accuracy: 78.6631% Training Loss: 0.5947
Validation Accuracy: 75.1515% Validation Loss: 0.5842
model saved
epoch: 5
```

3. In task2, make graphs for learning rate schedule, weights and gradients (With Tensorboard)

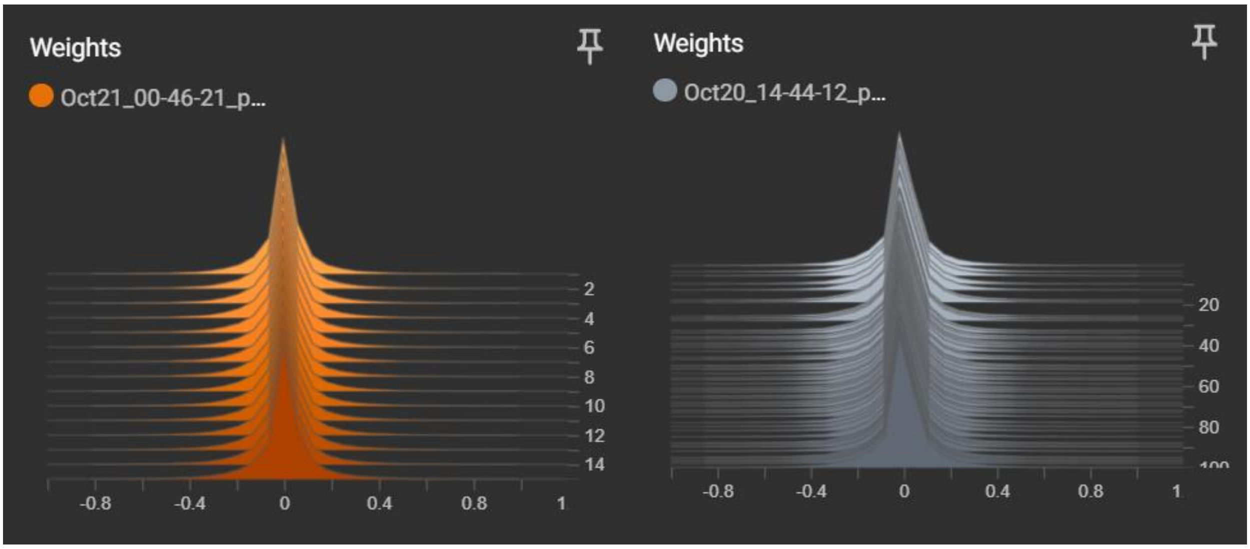
- Learning rate：



	Run ↑	Smoothed	Value	Step	Relative
●	Oct20_14-44-12_pc414-83	0.01	0.01	100	6.642 hr
●	Oct20_22-58-52_pc414-103	0.01	0.01	5	18.14 min
●	Oct20_23-45-26_pc414-91	0.001	0.001	28	38.34 min
●	Oct21_00-27-33_pc414-91	0.001	0.001	35	4.045 min
●	Oct21_00-33-42_pc414-91	0.0001	0.0001	100	10.62 min
●	Oct21_00-46-21_pc414-91	0.005	0.005	15	1.629 min
●	Oct21_00-50-56_pc414-91	0.05	0.05	47	5.241 min
●	Oct22_11-14-50_pc414-100	0.01	0.01	5	13.79 min
●	Oct23_16-00-54_pc414-92	0.01	0.01	1	0

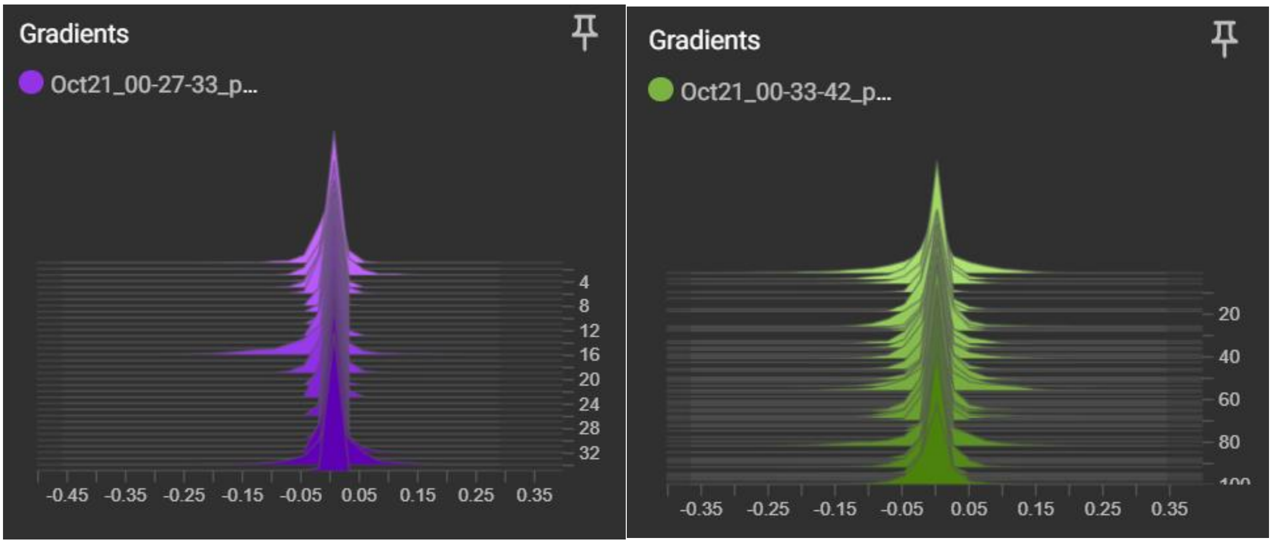
我一開始的 optimizer 是採用 Adam、learning rate = 0.01，不過我發現它學習的速度很慢，經過 100 次 epoch 後，training data accuracy 仍未突破 80%。所以後來我改成採用 SGD，、learning rate = 0.01，學習速度便馬上提高，經過 5 次 epoch 後，training data accuracy 便達到 80%左右，兩者差異非常大。因此由上圖可以發現，幾乎每條 learning rate 都是橫線，因為我大部分都採用 SGD，只有少數幾條是採用 Adam，所以會呈現出斜線。而我嘗試了 0.01 至 0.0001 的 learning rate，發現 0.01 的效果是最好的，既不會太大而無法收斂，也不會太小而速度緩慢。

- Conv1.weights：



我認為由於是預訓練的 resnet18，所以可以看出 weights 的初始化是有根據 data 進行預設的，而非隨機預設。這樣圖中的 weights 才會從頭到尾都差不多，同時 loss 有明顯減少、accuracy 大幅提升。(左圖共 15 次 epoch，右圖共 100 次 epoch)

- Conv1.gradients：



我認為由於是 resnet18 這類的殘差神經網路，透過 shortcut 機制有助於減輕梯度消失的問題。從圖中可以發現，梯度雖然有慢慢減少，不過並沒有消失的問題，例如右圖的第 80 次 epoch，它的 gradient 類似第 0 次 epoch 的 gradient 範圍、左圖的第 34 次 epoch 甚至比第 0 次 epoch 的 gradient 範圍還大。經多次實驗和分析圖可知，resnet18 的確能有效改善梯度消失的問題。(左圖共 35 次 epoch，右圖共 100 次 epoch)