ResNet18 架構

ResNet-18 是一個相對較淺的深度神經網絡，它包括了輸入層以及其他18層。這些層分佈在不同的模塊中，每個模塊都包含多個卷積層和標準的正則化層。

layer name	output size	18-layer	34-layer	50-layer	101-layer
conv1	112×112	7×7, 64, stride 2			
		3×3 max pool, stride 2			
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax			
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9

ResNet-18的關鍵組成部分：

1. 輸入層：接收圖像數據，圖像大小通常是224x224像素，具有三個通道（紅、綠、藍）
2. 第一個卷積層：7x7的卷積層，用於提取圖像中的低級特徵，具有64個濾波器
3. 最大池化層：用於降低圖像分辨率，同時保留最顯著的特徵

ResNet18 架構

4. **殘差塊（Residual Block）**：ResNet的關鍵構建塊。每個Residual Block包含兩個卷積層，通常是3x3大小的濾波器。與傳統卷積層不同，Residual Block引入了skip connection，可將輸入直接添加到輸出。這種設計允許信息在深度網絡中更自由地傳播，減輕了梯度消失問題。

5. 全局平均池化層：在卷積層之後，ResNet-18使用全局平均池化層，將每個通道的特徵圖進行平均化，以生成一個特徵向量。

6. 全連接層：將特徵向量映射到類別標簽的概率分佈。在ResNet-18中，通常使用一個包含1000個節點的全連接層，因為它最初是為ImageNet挑戰設計的，其中有1000個類別。

ResNet18 架構

ResNet-18包含了四個Residual Block:

```python
class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()
        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.build_layer( 64,  64, 2, 1)
        self.layer2 = self.build_layer( 64, 128, 2, 2)
        self.layer3 = self.build_layer(128, 256, 2, 2)
        self.layer4 = self.build_layer(256, 512, 2, 2)

        self.avgpool = nn.AdaptiveAvgPool2d(output_size=(1,1))
        self.fc = nn.Linear(512, 1000, bias=True)

    def forward(self, x):
        x = self.maxpool(self.relu(self.bn1(self.conv1(x))))

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x).view(x.size(0), -1)
        x = self.fc(x)

        return x

    def build_layer(self, ch_in, ch_out, num_Block, stride=1):
        layers = []
        layers.append(BasicBlock(ch_in, ch_out, stride))

        for _ in range(1, num_Block):
            layers.append(BasicBlock(ch_out, ch_out, stride=1))

        return nn.Sequential(*layers)
```

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

        if stride != 1 or ch_in != ch_out:
            self.downsample = nn.Sequential(
                nn.Conv2d(ch_in, ch_out, kernel_size=1, stride=stride),
                nn.BatchNorm2d(ch_out)
            )
        else:
            self.downsample = nn.Sequential()


    def forward(self, inputs):
        x = self.relu(self.bn1(self.conv1(inputs)))
        x = self.bn2(self.conv2(x))
        inputs = self.downsample(inputs)
        x = F.relu(x+inputs)
        return x
```

ResNet18 優勢

1. 殘差塊（Residual Block）：引入了殘差塊的概念，允許網絡更輕鬆地訓練非常深的架構。傳統的深度網絡可能會受到**梯度消失的問題**，而ResNet通過**跳躍連接**允許梯度在網絡中更容易傳播。

2. 深度：雖然ResNet18是相對較淺的網絡，但ResNet系列的**深度可擴展性**是其優勢之一。在處理更複雜的任務時，可以使用更深的ResNet架構，如ResNet50、ResNet101等。相比傳統深度學習模型，這種深度擴展的能力使得ResNet18可以捕獲更豐富和抽象的特徵，從而提高了其在複雜任務上的性能。

ResNet18 優勢

3. 性能:

- **更高的準確度**: 由於深度和殘差塊結構的使用，ResNet18能夠**更好地捕獲圖像特徵**，從而實現更高的分類準確度

- **更好的泛化能力**: 由於深度網絡的特性，ResNet18在大規模數據集上表現出色，同時在**小規模數據集**上也具有出色的泛化能力，這使它在圖像分類任務中表現出色

4. 訓練效率:

- **更快的收斂速度**: 由於殘差塊的跳躍連接，梯度能夠更自由地傳播，導致網絡更快地收斂到最佳解決方案。

- 更少的訓練數據需求: ResNet-18通常需要較少的訓練數據就可以獲得良好的性能，這對於任務和數據稀缺的情況非常有用。

Task 1： Build Resnet18 by yourself

1. 根據ResNet18的架構，我也構建了最大池化層以及全局平均池化層 。

最大池化層（ MaxPooling ）

- 强調顯著特徵：選取每個池化窗口的最大值，因此强調了顯著的特徵，有助於保留對圖像中最重要的特徵的敏感性
- 位置不變性：即使物體在圖像中稍微移動，最大池化層仍然能夠檢測到它，對於物體檢測非常重要
- 減小特徵維度：有助於減少計算複雜度，降低模型的參數數量

全局平均池化層（AveragePooling）

- 特徵向量生成：計算整個特徵圖的平均值，將其轉換為一個特徵向量，有助於減小數據維度
- 減輕過擬合：提供一個正則化效果，減輕模型的過擬合，因為它會顯著減少參數數量
- 適用於分類：在圖像分類任務中表現良好，因為它可以將特徵圖變換為固定長度的向量，適用於全連接層的輸入

Task 1： Build Resnet18 by yourself

```
transform = transforms.Compose([
                transforms.RandomRotation(degrees=20),
                transforms.Resize((img_size,img_size)),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
                ])
```

2. **圖像預處理變換(transform)：** 我使用了隨機旋轉、調整大小、標準化等。這些變換的優勢包括：

-數據增强：隨機旋轉可以增加訓練數據的多樣性，有助於模型更好地適應不同的數據變化和視角，減輕過擬合問題。數據增强是一種有效的正則化方法。

- 標準化和尺寸一致性：將圖像調整到相同大小並進行標準化，有助於模型更快地收斂。尺寸一致性也可以確保圖像與模型的輸入尺寸匹配，避免了尺寸不匹配導致的錯誤。

- 預處理：通過對圖像進行預處理，如標準化（Normalize），可以更好地處理不同圖像通道和像素值的變化。這有助於模型更好地處理圖像的亮度、對比度和顏色差異。而 ToTensor 則可以將圖像數據從 PIL Image 轉換為 PyTorch 張量（Tensor），以確保圖像數據與深度學習模型的輸入格式匹配。此外，它還有助於數值穩定性和模型的收斂。

- 提高訓練效率：合適的transform可以提高模型的訓練效率。數據變換的處理使得模型能夠更快地學習和收斂。

Task 1： Build Resnet18 by yourself

```
learning_rate = 0.0005
optimizer = optim.SGD(model.parameters(), lr=learning_rate, weight_decay=1e-5, momentum=0.9)
criterion = nn.CrossEntropyLoss()
```

3. 自定義優化參數：我自定義了學習率、權重衰減和動量參數。這些參數可以影響訓練的速度和性能，微調模型的訓練過程。

Task 1： Build Resnet18 by yourself

Achieve at least 75% validation accuracy:

最後我的準確度有達到84.1945%

```
epoch: 36
Training Accuracy: 97.5702% Training Loss: 0.0880
Validation Accuracy: 84.1945% Validation Loss: 0.4685
model saved
```

Task 2： Improve accuracy

1. ResNet18 → ResNet50：ResNet50有更深的網絡結構，這意味著它可以更好地捕獲圖像的高級特徵，因為它有更多的層和參數，可以更有效地處理更複雜的任務和數據集。

2. 圖像預處理變換(transform)：我使用了一些圖像預處理變換，包括隨機旋轉、水平翻轉、調整大小、標準化等。這些變換的優勢包括：

- **數據增强**：隨機旋轉和水平翻轉可以增加數據集的多樣性，有助於網絡更好地泛化到不同的角度和變化。這有助於減輕過擬合問題。

- **調整大小**：將圖像調整到相同的大小，以確保它們與網絡的輸入尺寸匹配。

- **標準化**：通過均值和標準差的標準化，確保數據的尺度一致，有助於網絡更快地收斂。

Task 2： Improve accuracy

3. 優化：我使用了隨機梯度下降（SGD）作為優化器， 并採用了合理的學習率（0.00035）和動量參數（0.9）。優化器的選擇和參數的調整對模型的性能至關重要，它們影響了模型的收斂速度和準確度。

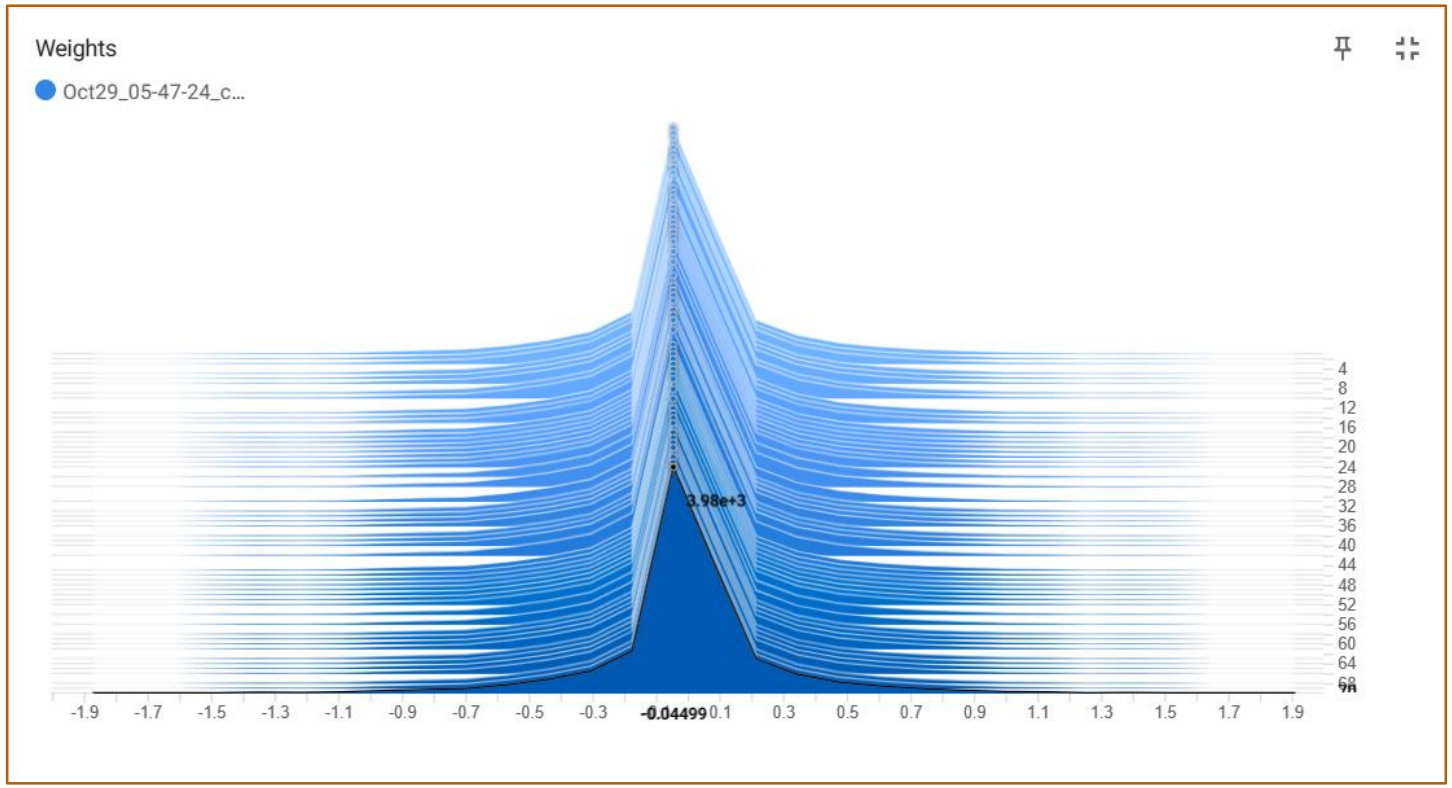
Task 2： Improve accuracy

Upload your result to Kaggle

The highest accuracy in Kaggle: 0.92571

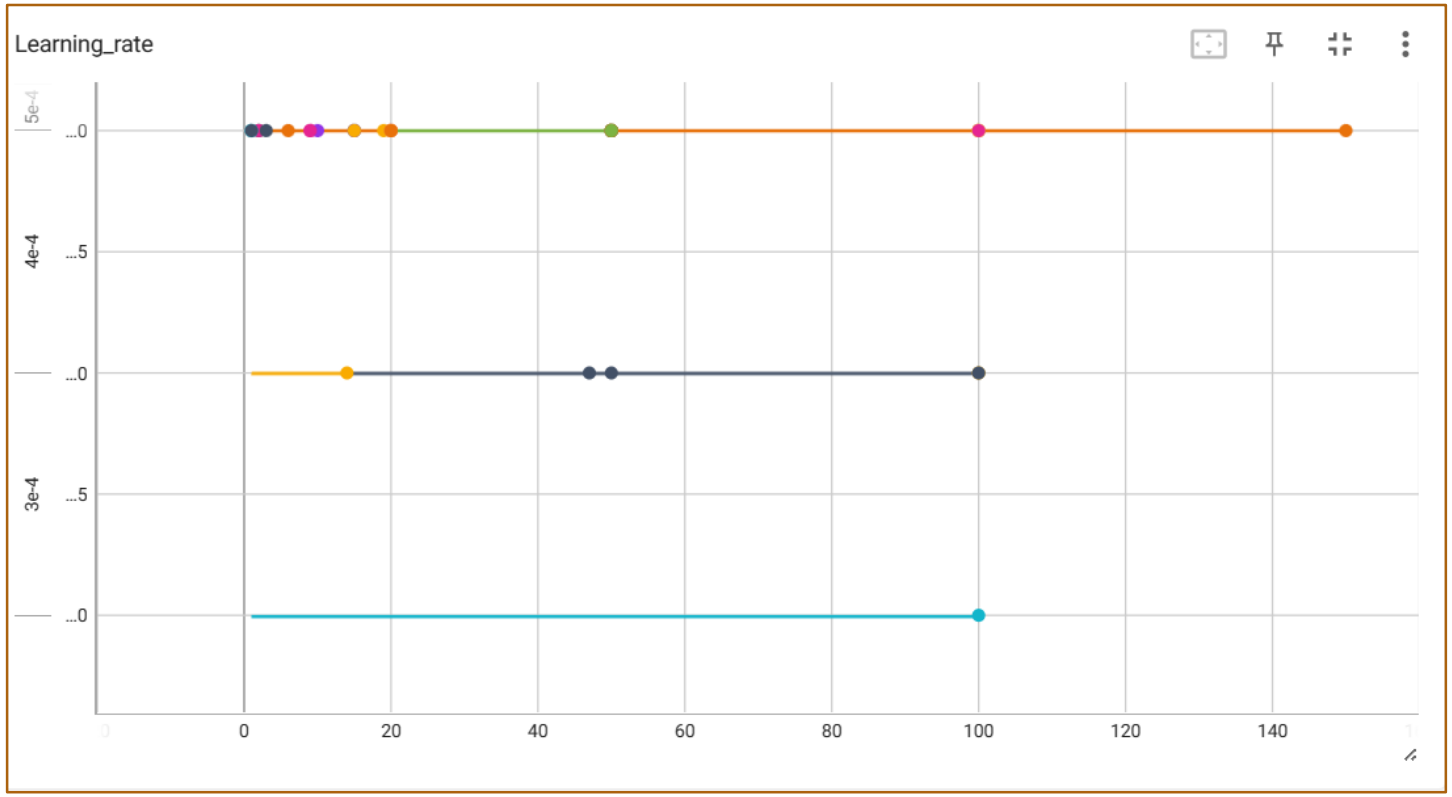
Task 2： Improve accuracy

Make graphs with Tensorboard: ResNet18 (weights)



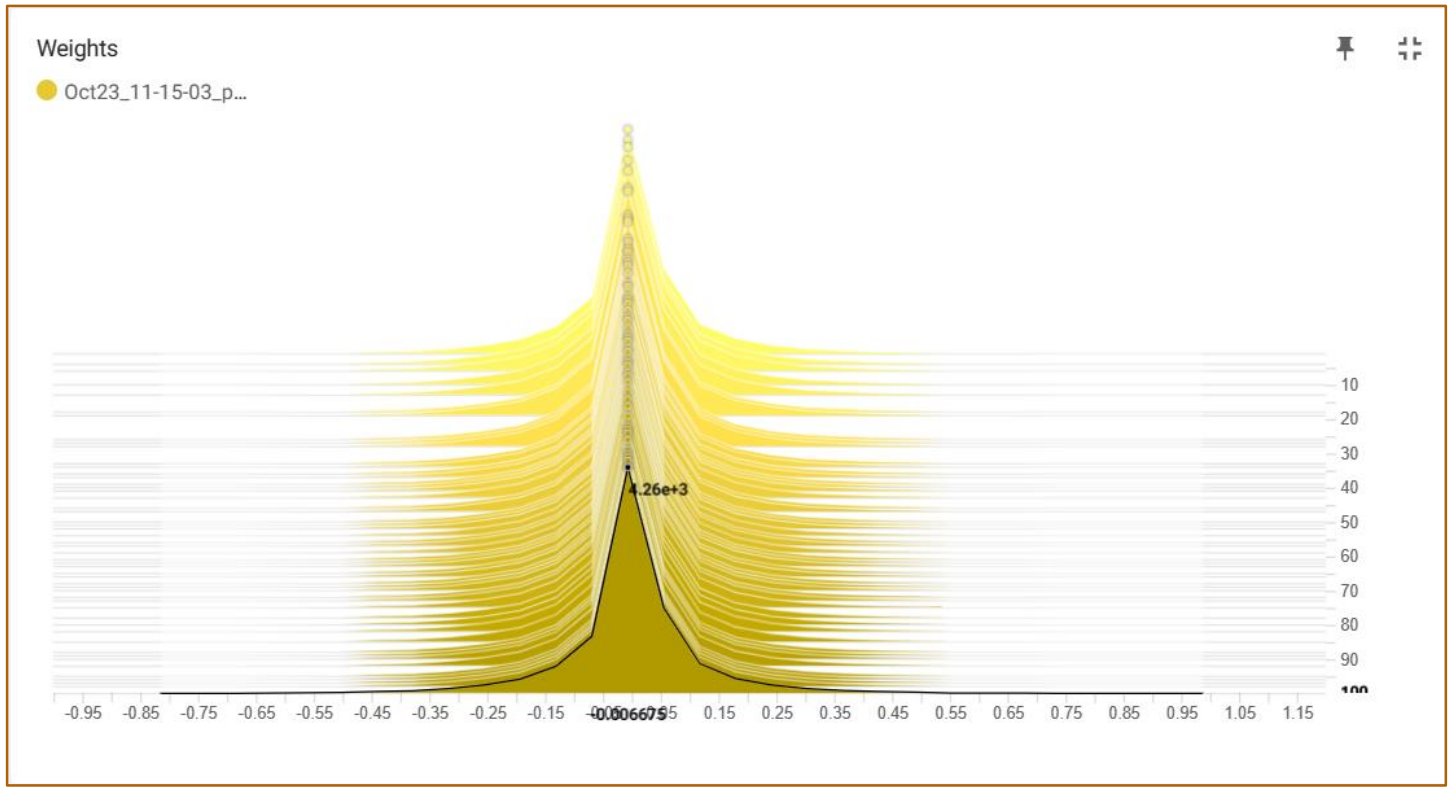
Task 2： Improve accuracy

Make graphs with Tensorboard: ResNet18 (learning rate)



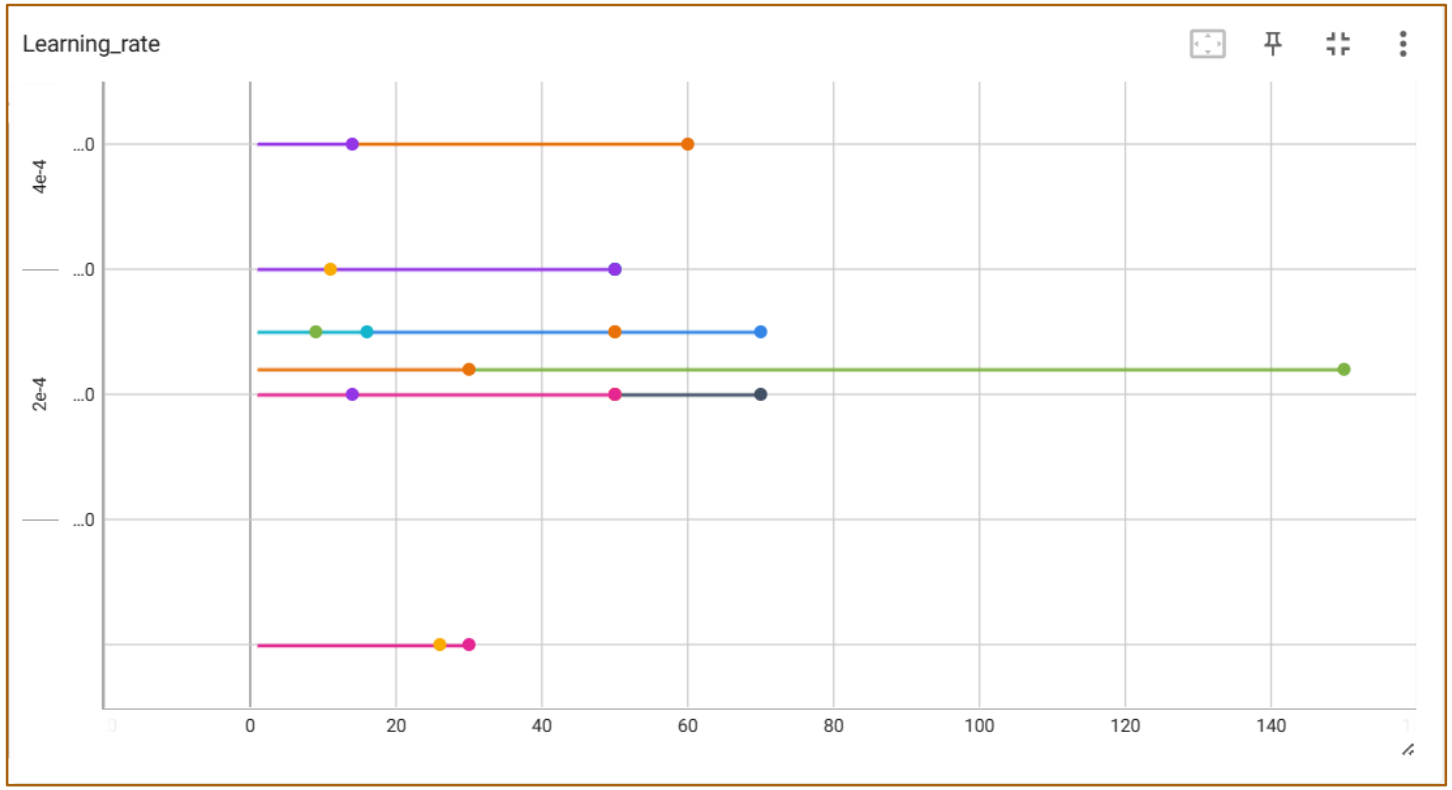
Task 2： Improve accuracy

Make graphs with Tensorboard: ResNet50 (weights)



Task 2： Improve accuracy

Make graphs with Tensorboard: ResNet50 (learning rate)



左圖所示，由於這次作業的數據量較小，learning rate不需要太大，因此我選擇訓練模型的學習率介於2e-4至4e-e之間。

Task 3： Compare ResNet18

WITHOUT PRETRAINED

- 權重隨機初始化：模型沒有經過先前任務的知識，需要從頭開始學習特定任務的特徵。
- 訓練需求：需要更多的數據和更長的時間
- 性能：非預訓練模型更適用於自定義任務，需要大規模數據集來訓練，可能不如與訓練模型在泛化各類任務上表現出色。

WITH PRETRAINED

- 權重初始化：預訓練模型的權重是通過在大規模數據集上進行預訓練而獲得的，已經學會了對廣泛的特徵進行表示，這有助於模型更快地收斂和更好的泛化。
- 遷移學習：預訓練模型可應用於與預訓練數據集不同但相關的任務，通常只需要微調以適應新任務，而不是從頭開始訓練。
- 性能：預訓練模型已經學會了在大規模數據上提取特徵，具有較高的性能，這對於圖像分類、物體檢測、圖像分割等任務非常有用。