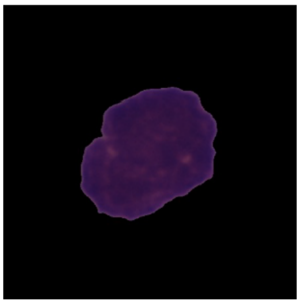
Lab3: Leukemia Classification

系級:智能系統　學號:312581006　姓名:張宸瑋

1. Introduction

這次 lab 要實作出 ResNet18、ResNet50、ResNet152 三種不同的架構，並且自行撰寫 dataloader 程式，設計自己的數據預處理方法，並且透過 ResNet，來對急性淋巴細胞白血病（acute lymphoblastic leukemia,ALL）來做分類，判斷輸入的圖片是正常細胞，還是白血病細胞。



Class
0 - Normal cell
1 - Leukemia blast

分類目的

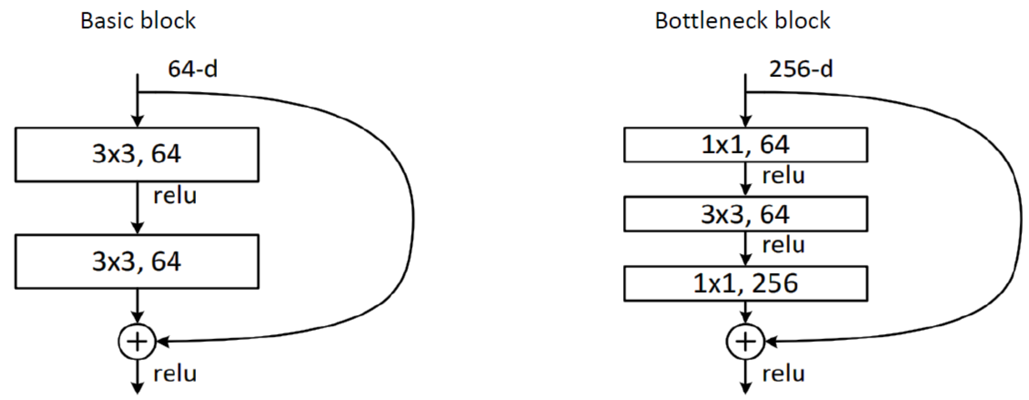
而這次的模型，主要會根據下面這張架構圖來去坐實作。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	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	11.3×10^9

ResNet 模型架構

而 ResNet 最大的特色就是引入了殘差模塊(Residual Block)，透過殘差連接將每個層的輸入映射到輸出，有效的解決了梯度消失與梯度爆炸的問題，也因為這樣子，透過搭建更深的網路，能夠讓模型學到更多的特徵，提升模型的表達能力。

以下是兩個不同的殘差模塊的架構差異，在這次實驗中，ResNet18 是採用 Basic block 架構，而 ResNet50 以及 ResNet152 是採用 Bottleneck block 架構。



2. Experiment setups

A. The detail of your model

```python
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1,drop_prob = 0.0):
        super(BasicBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Dropout(drop_prob),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels)
        )
        self.relu = nn.ReLU(inplace = True)
        self.dropout = nn.Dropout(drop_prob)
        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, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.conv(x)
        out += self.shortcut(x)
        out = self.relu(out)
        out = self.dropout(out)
        return out
```

BasicBlock 架構實現

```python
class Bottleneck(nn.Module):
    expansion = 4
    def __init__(self, in_channels, out_channels, stride=1, drop_prob=0.0):
        super(Bottleneck, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Dropout(drop_prob),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Dropout(drop_prob),
            nn.Conv2d(out_channels, out_channels * self.expansion, kernel_size=1, stride=1, bias=False),
            nn.BatchNorm2d(out_channels * self.expansion)
        )

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels * self.expansion:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * self.expansion, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels * self.expansion)
            )
        self.relu = nn.ReLU(inplace = True)
        self.dropout = nn.Dropout(drop_prob)
    def forward(self, x):
        residual = x
        out = self.conv(x)
        out += self.shortcut(residual)
        out = self.relu(out)
        out = self.dropout(out)
        return out
```

Bottleneck 架構實現

```python
class ResNet18(nn.Module):
    def __init__(self, num_classes=1000,drop_prob=0.0):
        super(ResNet18, 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.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self.make_layer(64, 2, stride=1,drop_prob = drop_prob)
        self.layer2 = self.make_layer(128, 2, stride=2,drop_prob = drop_prob)
        self.layer3 = self.make_layer(256, 2, stride=2,drop_prob = drop_prob)
        self.layer4 = self.make_layer(512, 2, stride=2,drop_prob = drop_prob)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)

    def make_layer(self, out_channels, num_blocks, stride,drop_prob = 0.0):
        layers = []
        layers.append(BasicBlock(self.in_channels, out_channels, stride, drop_prob))
        self.in_channels = out_channels
        for _ in range(1, num_blocks):
            layers.append(BasicBlock(out_channels, out_channels, drop_prob = drop_prob))
        return nn.Sequential(*layers)

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

ResNet18 架構

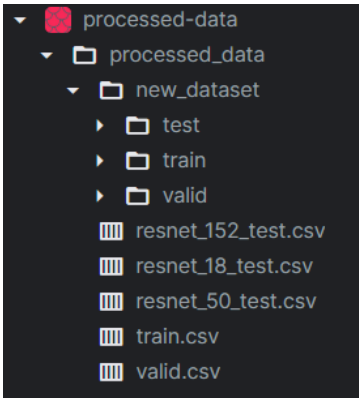
```python
class ResNet152(nn.Module):
    def __init__(self, num_classes=1000, drop_prob=0.0):
        super(ResNet152, self).__init__()
        self.drop_prob = drop_prob
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = 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, 3, stride=1,drop_prob = drop_prob)
        self.layer2 = self._make_layer(128, 8, stride=2,drop_prob = drop_prob)
        self.layer3 = self._make_layer(256, 36, stride=2,drop_prob = drop_prob)
        self.layer4 = self._make_layer(512, 3, stride=2,drop_prob = drop_prob)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * Bottleneck.expansion, num_classes)

    def _make_layer(self, out_channels, num_blocks, stride,drop_prob):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(Bottleneck(self.in_channels, out_channels, stride, drop_prob=self.drop_prob))
            self.in_channels = out_channels * Bottleneck.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.relu(self.bn1(self.conv1(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 = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

ResNet50 架構

```python
class ResNet50(nn.Module):
    def __init__(self, num_classes=1000, drop_prob=0.0):
        super(ResNet50, self).__init__()
        self.drop_prob = drop_prob
        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, 3, stride=1,drop_prob = drop_prob)
        self.layer2 = self._make_layer(128, 4, stride=2,drop_prob = drop_prob)
        self.layer3 = self._make_layer(256, 6, stride=2,drop_prob = drop_prob)
        self.layer4 = self._make_layer(512, 3, stride=2,drop_prob = drop_prob)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * Bottleneck.expansion, num_classes)

    def _make_layer(self, out_channels, num_blocks, stride,drop_prob)   :
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(Bottleneck(self.in_channels, out_channels, stride, drop_prob=self.drop_prob))
            self.in_channels = out_channels * Bottleneck.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.relu(self.bn1(self.conv1(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 = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

ResNet152 架構

B.　The details of your Dataloader



資料集目錄

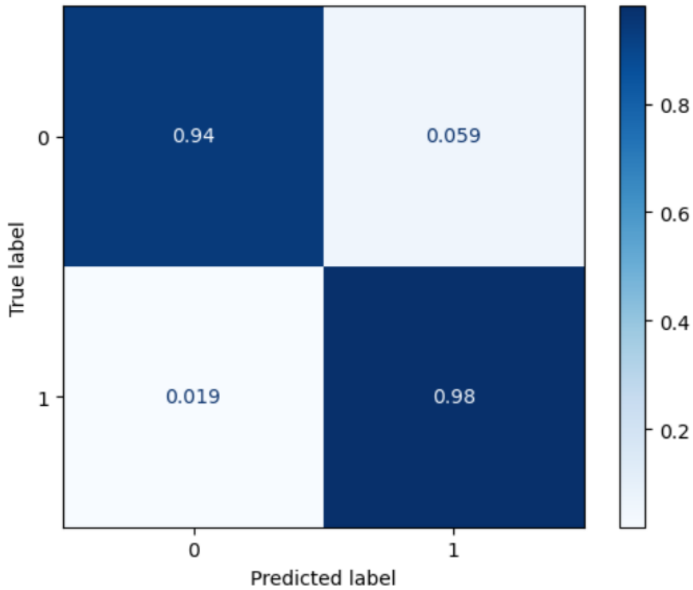
```python
def getData(mode):
    if(mode == "train"):
        df = pd.read_csv("/kaggle/input/processed-data/processed_data/train.csv")
        path = df["Path"].tolist()
        label = df["label"].tolist()
        return path,label
    elif(mode == "valid"):
        df = pd.read_csv("/kaggle/input/processed-data/processed_data/valid.csv")
        path = df["Path"].tolist()
        label = df["label"].tolist()
        return path,label
    else:
        df = pd.read_csv("/kaggle/input/processed-data/processed_data/resnet_18_test.csv")
        path = df["Path"].tolist()
        label = [0] * len(path)
        return path,label
```

透過 getData 搭配參數 mode，來決定要讀哪個 csv 的檔案，因為這次的實驗我是在 kaggle notebook 上進行的，所以 csv 檔以及訓練用圖片都放在 kaggle/input 這個路徑下。

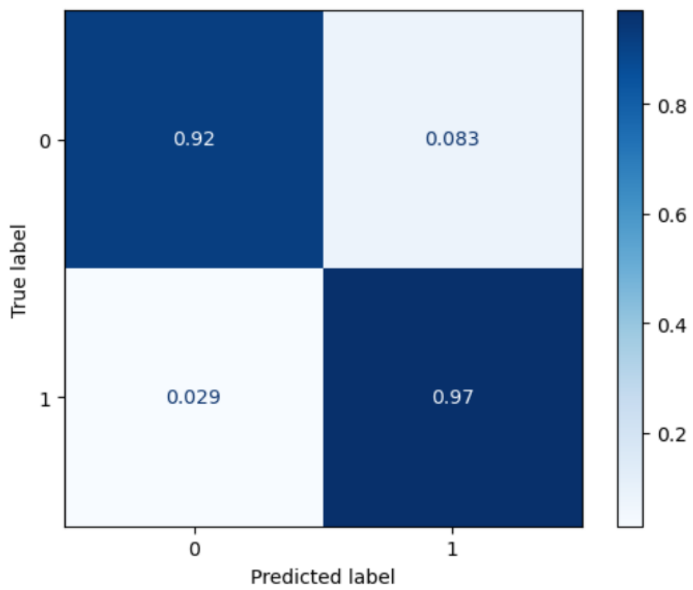
```python
class RetinopathyLoader(Dataset):
    def __init__(self,root,mode,transform=None):
        self.root = root
        self.img_name, self.labels = getData(mode)
        self.transform = transform
        self.mode = mode
        print("> Found %d images..."%(len(self.img_name)))
    def __len__(self):
        return len(self.img_name)
    def __getitem__(self,idx):
        img_name = os.path.join(self.root, os.path.normpath(self.img_name[idx]))
        image = Image.open(img_name).convert('RGB')
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        return image, label
```

1. __init__(self, root, mode, transform=None)：
   - root：數據集的根目錄。
   - mode：數據加載模式，用於指定是訓練集、驗證集還是測試集。
   - transform：數據預處理和轉換的函數，用於對圖像進行預處理。
2. __len__(self)：
   - 返回數據集的樣本數量。
3. __getitem__(self, idx)：
   - 根據給定的索引 idx，獲取數據集中的一個樣本。
   - img_name：根據索引獲取圖像的文件名。
   - image：從文件名加載圖像，並將其轉換為 RGB 格式。
   - label：根據索引獲取對應圖像的標籤。
   - 如果 transform 函數存在，則將圖像應用預處理和轉換。

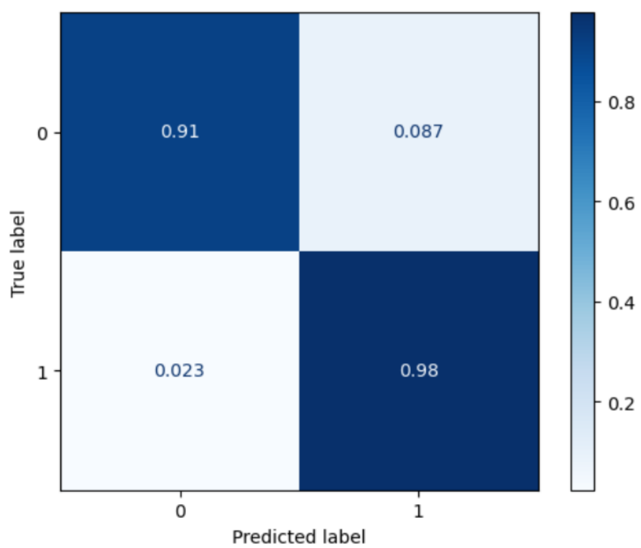
C. Describing your evaluation through the confusion matrix



ResNet18 confusion matrix



ResNet50 confusion matrix



ResNet152 confusion matrix

討論:

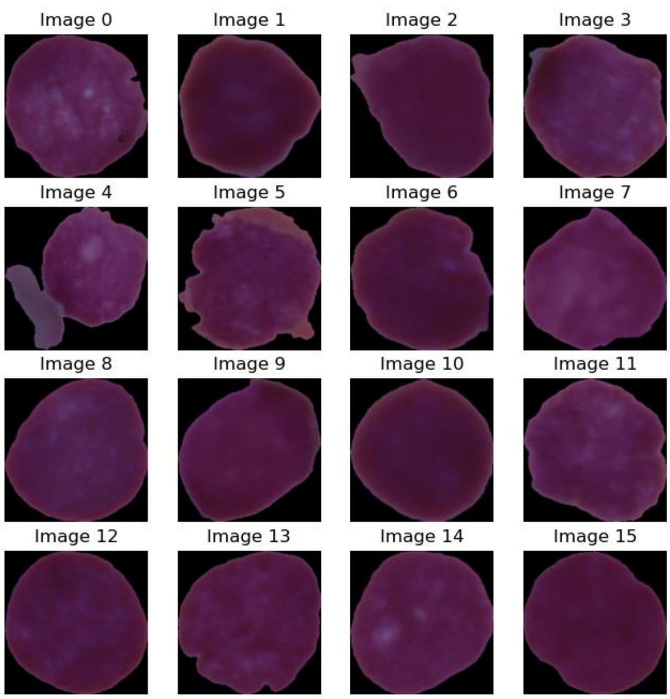
從三個模型的 confusion martix 我們可以看到，在細胞為正常的情況下模型預測是白血球細胞的情況比較高，因此這代表我們可能需要增加正常細胞的訓練樣本，或者是平衡正常細胞跟白血球細胞的樣本數量，來優化我們的模型。

3. Data Preprocessing

A. How you preprocessed your data?

```
import cv2 as cv
import numpy as np
import os
import glob
# 資料夾路徑
folder_path = "/kaggle/input/2023-summer-nycu-dl-lab3/new_dataset/new_dataset/train"
# 存放處理後影像的列表
img_list = []
# 取得所有影像檔案路徑
image_files = glob.glob(os.path.join(folder_path, "*.bmp"))
# 迭代處理每個影像檔案
for image_file in image_files:
    # 讀取影像
    image = cv.imread(image_file)
    # 轉成灰階
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    # 二值化處理
    thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU)[1]
    # 根據二值化的遮罩來裁切影像
    result = cv.bitwise_and(image, image, mask=thresh)
    result[thresh == 0] = [255, 255, 255]
    (x, y, z_) = np.where(result > 0)
    mnx = (np.min(x))
    mxx = (np.max(x))
    mny = (np.min(y))
    mxy = (np.max(y))
    crop_img = image[mnx:mxx, mny:mxy, :]
    crop_img_r = cv.resize(crop_img, (224, 224))
    img_list.append(crop_img_r)
```

在這次的預處理中，我先將整份 dataset 透過上述程式碼先做了預處理，並且保存在資料夾中，供我之後訓練做使用，而這段程式碼最主要的功能是對每張圖像做二值化，並且根據二值化的遮罩裁切我們感興趣的區域(細胞本身)，最後調整成我們所需要的大小，以下為預處理過後的示意圖。



透過上述程式碼所得到的結果

```
/kaggle/working
    precossed_dataset
        train
        test
        valid
```

將整份資料集做上述操作後，會得到這份檔案，將這份檔案下載下來，透過這份檔案的資料集來做訓練

```
data_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # 隨機水平翻轉
    transforms.RandomRotation(15),  # 隨機旋轉圖像
    transforms.RandomApply([transforms.ColorJitter(brightness=0.005, contrast=0.005, saturation=0.005, hue=0.005)], p=0.0
    transforms.RandomApply([transforms.GaussianBlur(kernel_size=7,sigma=(2))], p=0.005),  # 添加高斯模糊，以20%的機率進行處理
    transforms.ToTensor(),
    transforms.Normalize(
    mean=[0.23292791,0.08690643,0.2023041],
    std=[0.14168608,0.06099727,0.12721768]
)])
```

上述程式碼是用來對訓練資料進行預處理的 pytorch 轉換，裡面包括了隨機水平翻轉影像、隨機旋轉影像、隨機應用高斯模糊，以及隨機改變影像亮度、對比度、飽和度、色調，最後再將影像轉換成 pytorch 張量，並且對影像進行標準化。

```
data_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(
    mean=[0.23292791,0.08690643,0.2023041],
    std=[0.14168608,0.06099727,0.12721768])
])
```

而因為評估模型的過程中，我們希望使用原始的訓練集跟驗證集，以及固定的預處理方法來做評估，這樣才可以確保評估的一致性，以及公平性，因此我們僅將資料做 pytorch 張量轉換，以及標準化。

```
channel_sums = [0, 0, 0]
channel_squared_diff_sum = [0, 0, 0]
num_pixels = 0

for batch in train_loader:
    images, _ = batch
    batch_size = images.size(0)
    num_pixels += batch_size * images.size(2) * images.size(3)  # 累加像素数目
    channel_sums += torch.sum(images, dim=(0, 2, 3)).cpu().numpy()
    channel_squared_diff_sum += torch.sum((images ** 2), dim=(0, 2, 3)).cpu().numpy()

# 計算像素值的平均值
means = channel_sums / num_pixels
# 計算像素值的標準差
stds = np.sqrt(channel_squared_diff_sum / num_pixels - means ** 2)
print("平均值 (mean):", means)
print("標準差 (std):", stds)

平均值 (mean): [0.23292791, 0.08690643, 0.2023041]
標準差 (std): [0.14168608, 0.06099727, 0.12721768]
```

然而，因為我們需要知道資料的平均值以及標準差，才能夠做標準化，因此我們透過上述的程式碼，來計算出訓練樣本的平均值以及標準差，並套用在我們的標準化轉換參數中。

B. What makes your method special?

在一開始訓練時，我都是先透過 CenterCrop，先從圖片的中心剪裁圖片，接著在 resize 成指定大小，但是這樣子訓練到最後很容易過擬合。並且不管用什麼樣的方法都沒辦法解決，因此最後再查看資料集後，發現有些細胞呈現較不規則的形狀，像下面這兩張細胞圖，如果我採用中心剪裁的方式，很有可能會把這些比較不規則狀的細胞圖特徵給剪裁掉，可能會導致重要的特徵沒辦法包含在剪裁後的區域，導致模型沒辦法訓練到重要特徵，也會導致模型的泛化能力下降，因此最後我採用透過二值化遮罩的方式，裁切出我們感興趣的區域，也就是細胞本身，這樣我們不但能夠使模型訓練到重要特徵，也能提高模型的泛化能力，我認為這個預處理的操作，是我模型準確率提升非常大的一個關鍵，而透過 pytorch 轉換，包括隨機水平翻轉，隨機選轉影像等等，也能夠改善我的模型的泛化能力。

不規則狀細胞圖

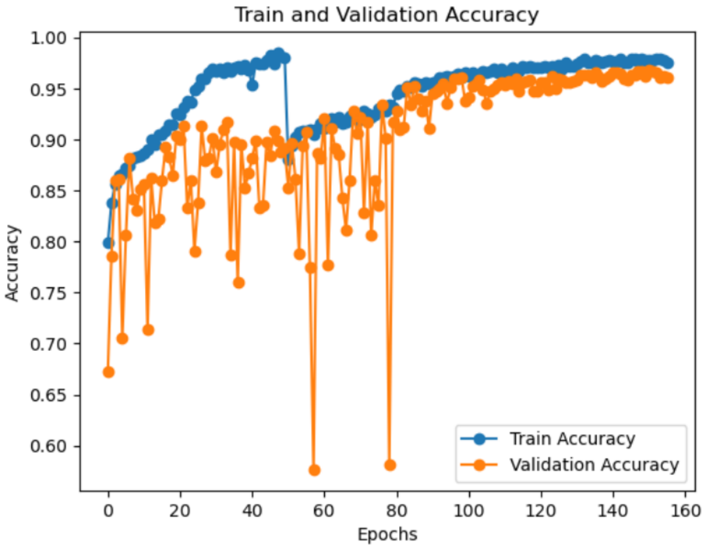
4. Experimental results

A. The highest testing accuracy

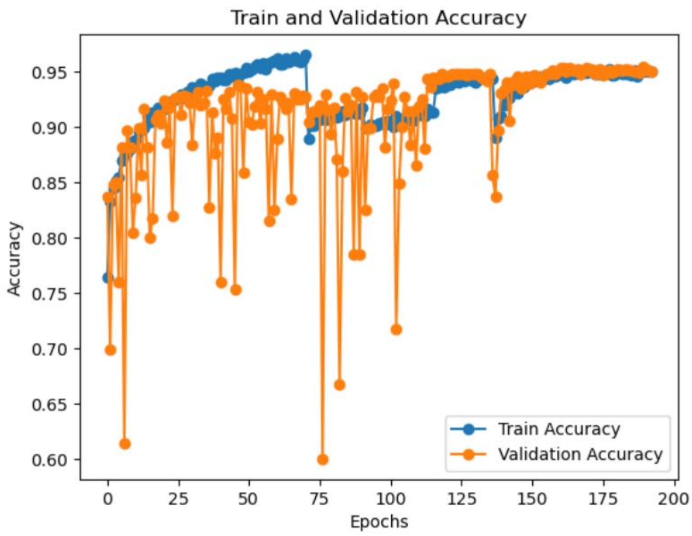
```
import torch.nn.functional as F
criterion = nn.CrossEntropyLoss()
test_loss, test_accuracy, _ = evaluate(resnet18, test_loader, criterion, device)
print(f"The best performance of the ResNet-18 model, Testing Loss: {test_loss:.4f}, Testing Accuracy: {test_accuracy:.4f}")
test_loss, test_accuracy, _ = evaluate(resnet50, test_loader, criterion, device)
print(f"The best performance of the ResNet-50 model, Testing Loss: {test_loss:.4f}, Testing Accuracy: {test_accuracy:.4f}")
test_loss, test_accuracy, _ = evaluate(resnet152, test_loader, criterion, device)
print(f"The best performance of the ResNet-152 model, Testing Loss: {test_loss:.4f}, Testing Accuracy: {test_accuracy:.4f}")

The best performance of the ResNet-18 model, Testing Loss: 0.1066, Testing Accuracy: 0.9681
The best performance of the ResNet-50 model, Testing Loss: 0.1335, Testing Accuracy: 0.9537
The best performance of the ResNet-152 model, Testing Loss: 0.1274, Testing Accuracy: 0.9568
```

The highest testing accuracy

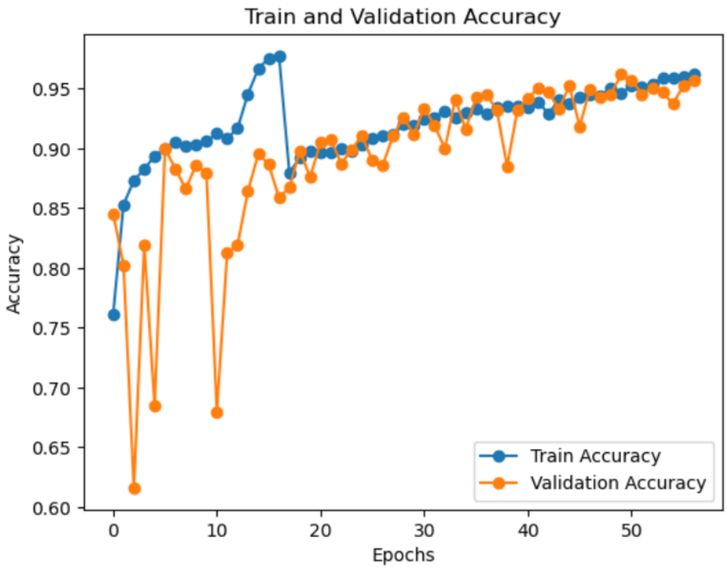
B. Comparison figures



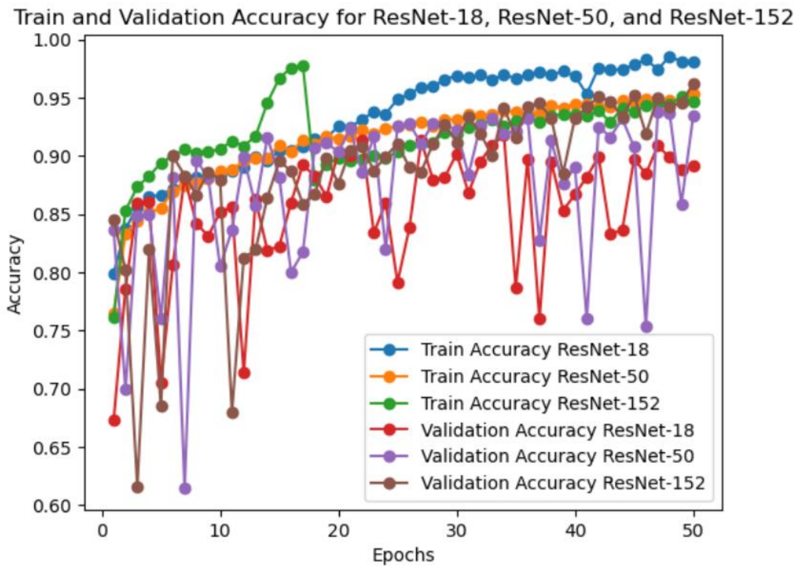
ResNet18 accuracy growth curve



ResNet50 accuracy growth curve



ResNet152 accuracy growth curve



Compare and visualize the accuracy trend between the 3 model architectures

討論:

在這裡我們可以分別看到三個不同架構的準確率成長圖，都在某一個 epoch 後，training accuracy 呈現下降的趨勢，那是因為我發現模型已經開始呈現過擬合的現象，因此我透過正則化的技術，以及 pytorch transforms 模組來增加多樣性，來提升模組的泛化能力，因此我們可以看到，在增加這些技術後，模型看起來有更好的擬合，並且提升了它的泛化能力，而我們也可以看到三個模型的準確率比較圖中，可以看到模型在評估方面的準確率表現較不穩定，這裡我在想可能是因為 batch size 設定 32 比較小的關係，導致模型梯度的更新較不穩定，但是如果將 batch size 調大，可能會導致內存不夠，因此這是我們在訓練時要權衡的。

5. Discussion

透過這次的實驗，讓我了解到 ResNet 的網路結構，以及它的設計精隨，也因為這次要透過三個模型來對我們的資料集做訓練，因為每個網路的複雜度不一樣，因此在初始化超參數的時候，是一個非常重要的關鍵，如果超參數的配置適當，我們在訓練過程中會更為順利，而我在這次的訓練中，我主要的策略是，先透過一組適當的超參數做訓練，如果練到最後模型的表現呈現過擬合的狀況，就會取消訓練，並且記錄最後訓練以及在這輪訓練中模型表現最好的參數配置，並且透過更改像是 weight decay 的值，或者透過 pytorch 的 transforms 增加數據多樣性，來提升模型的泛化能力，透過這次的實驗，除了讓我對 ResNet 有了更多的認識，並且體會到它的奧妙，也讓我對模型訓練的整個過程更加熟悉，並且能透過不同的方式來改善模型的效能。