

Lab 3 report

1. Introduction

急性淋巴性白血病 (ALL) 是最常見的兒童癌症類型，約佔所有兒童癌症的 25%。這次實驗主要是用 resnet18、50、152 去分析急性淋巴性白血病。首先要透過 PyTorch，撰寫一個自己的 DataLoader，並設計自己的資料預處理技術。再來透過使用 ResNet 來進行急性淋巴性白血病的分類，最後將預測結果上傳到 Kaggle 的比賽中，並繪製 confusion matrix 和 accuracy curve，去評估模型的效能。

Dataset - Leukemia Classification (Kaggle):

這個 dataset 包含了 10,661 張細胞圖片，這些細胞圖片來自於顯微鏡圖像的分割，並且可以代表實際世界中的圖像，因為它們包含了一些 staining noise 和 illumination errors，儘管這些 errors 在取得圖像的過程中已經大致修正了。我們將 dataset 分成了 7,995 張的訓練資料，1,599 張的驗證資料，以及 1,067 張的測試資料。總共有 10,661 張圖片，並且分為兩個標籤的類別：

0: 正常細胞

1: 白血病母細胞

這些圖片的解析度都是一樣的，都是 $450 * 450$ 像素。

2. Implementation Details

A. The details of your model (ResNet)

```

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

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

class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion*planes, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion*planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

```

```

    def forward(self, x):
        out = nn.ReLU()(self.bn1(self.conv1(x)))
        out = nn.ReLU()(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = nn.ReLU()(out)
        return out

class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=2):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = nn.ReLU()(self.bn1(self.conv1(x)))
        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.linear(out)
        return out

def ResNet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])

```

```
def ResNet50():
    return ResNet(Bottleneck, [3, 4, 6, 3])

def ResNet152():
    return ResNet(Bottleneck, [3, 8, 36, 3])
```

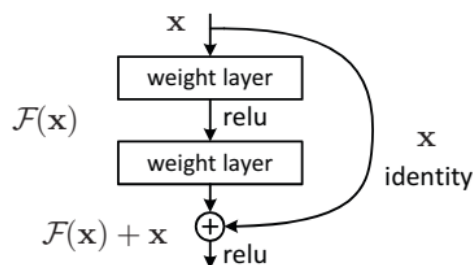
(1) 程式設計

我將 resnet 18、resnet 50、resnet 152 都打在一起，因為這三個模型在程式設計上的差異只有 resnet 50、resnet 152 是用 bottleneck。另外如果 block 的輸入 channel 數跟輸出不同，就會用 projection shortcuts 提升輸入 channel 數，再將它加到輸出，如果輸入 channel 數跟輸出相同則 shortcuts identity。下圖是 ResNet 論文的模型架構，我的 ResNet 皆是按照下面架構做出來的，只有把最後的全連接層的輸出改成 2 維。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				

(2) ResNet 概念

當模型變得特別深的時候，就有可能會發生梯度消失，就算加入一些 batch normalization，讓模型減緩梯度消失，可是只要模型變深的時候，test accuracy 一樣會變差，這不是 overfitting 的問題，因為 train accuracy 一樣在變差，所以 resnet 論文的作者，就想用 residual 的方法，讓模型較深的時候的 accuracy 不會比模型較淺的時候差。下圖是 residual 的範例圖。



從上圖可以看到 residual 跟一般的神經網路區別就在 residual 多加了一個 shortcut connections，使得輸出變成了 $F(x)+x$ ，其中 $F(x)$ 代表了層的輸出， x 是 shortcut connections 的輸入。這樣的結構讓模型在學

習過程中，不僅學習原本的輸出 $F(x)$ ，還能更容易地學習 identity mapping，即當某一層的最佳 mapping 接近 identity mapping 時，殘差 $F(x)$ 將趨近於零。這個特性解決了深度模型退化的問題。當模型層數太多或太複雜時，前面幾層可能訓練到不錯的狀態，但由於後面層數太多，可能導致梯度消失或爆炸，使整體輸出變差。通過 ResNet 的結構，即使在後面的層中，殘差也可以接近於 0（如果 identity mapping 是最佳解），從而使整個模型的輸出保持在良好的狀態。ResNet 的核心思想就是優化殘差 $F(x)$ ，透過引入 shortcut connections，使得即使模型層數很多，accuracy 也不低於層數較少的模型，因為它可以更容易地學習 identity mapping，並減緩或消除退化問題。

$$\text{其他模型: } \frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

$$\text{ResNet: } \frac{\partial f(g(x)) + g(x)}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} + \frac{\partial g(x)}{\partial x}$$

上面公式可以看出其他模型如果在 backpropagation，可能會因為乘太多小於 1 的值造成梯度消失，resnet 則會因為加上之前的輸入，導致始終保持一個比較大的值，不會造成梯度消失。

(3) ResNet18 與 ResNet50、ResNet152 的差別

ResNet18 共有 18 層、ResNet50 共有 50 層、ResNet152 共有 152 層，ResNet50 和 ResNet152 使用了一個稱為 Bottleneck 的結構，這個結構可以減少參數的數量，而不會降低網絡性能，ResNet18 則沒有使用這種結構，但是由於層數的增加，ResNet50 和 ResNet152 相對於 ResNet18 更為複雜，這代表它們的參數更多，計算量也更大，這代表通常 ResNet50 和 ResNet152 性能較好。

B. The details of your Dataloader

```
def getData(mode, model_name):
    if mode == 'train':
        df = pd.read_csv('train.csv')
        path = df['Path'].tolist()
        label = df['label'].tolist()
        return path, label

    elif mode == "valid":
        df = pd.read_csv('valid.csv')
        path = df['Path'].tolist()
        label = df['label'].tolist()
        return path, label

    else:
        df = pd.read_csv(f'{model_name}_test.csv')
        path = df['Path'].tolist()
        return path, None
```

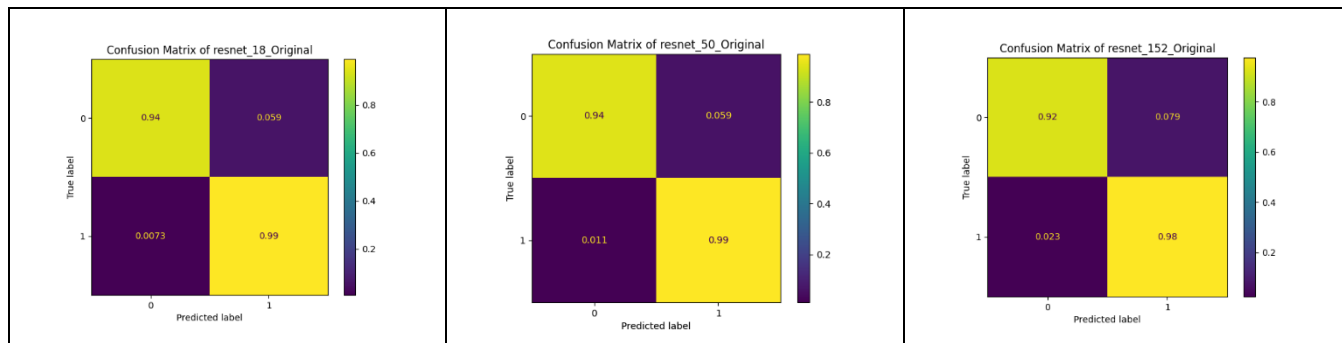
```
class LeukemiaLoader(data.Dataset):
    def __init__(self, mode, model_name):
        self.img_name, self.label = getData(mode, model_name)
        self.mode = mode
        print("> Found %d images..." % (len(self.img_name)))
        if self.mode == 'train':
            self.transform = transforms.Compose([
                transforms.RandomHorizontalFlip(),
                transforms.RandomVerticalFlip(),
                transforms.RandomRotation(20),
                transforms.CenterCrop(270),
                transforms.Resize((224, 224)),
                #GaborFilter(size=5, sigma=1.0, theta=0, lambda=1.0, gamma=0.5, psi=0),
                #HighPassFilter(size=3),
                transforms.ColorJitter(brightness=0.2, contrast=1.2, saturation=1.2, hue=0.1),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.0026741, 0.00156906, 0.00272411], std=[0.00314106, 0.00195709, 0.00304743])
            ])
        else:
            self.transform = transforms.Compose([
                transforms.CenterCrop(270),
                transforms.Resize((224, 224)),
                #GaborFilter(size=5, sigma=1.0, theta=0, lambda=1.0, gamma=0.5, psi=0),
                #HighPassFilter(size=3),
                transforms.ColorJitter(brightness=0.2, contrast=1.2, saturation=1.2, hue=0.1),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.0026741, 0.00156906, 0.00272411], std=[0.00314106, 0.00195709, 0.00304743])
            ])
    def __len__(self):
        """return the size of dataset"""
        return len(self.img_name)
    def __getitem__(self, index):
        path = self.img_name[index]
        img = cv2.imread(path, cv2.IMREAD_COLOR)
        #img = mean_filter_denoise(img)
        img = clahe_enhancement(img)
        img = unsharp_mask(img, 3, 1.5)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #img = simple_white_balance(img)
        img = Image.fromarray(img)
        img = self.transform(img)

        if self.mode == 'train' or self.mode == 'valid':
            label = self.label[index]
            return img, label
        else:
            return img
```

我首先利用函式 `getData` 讀取 `train`、`valid`、`test.csv` 檔，取出 `train`、`valid`、`test data` 每張圖片的路徑跟 `label`，再用 `__len__` 函式算出總共取出多少張圖片，最後再用 `__getitem__` 函式做資料前處理。當要載入資料到模型時，再呼叫 `class LeukemiaLoader` 就可以了，舉例：`train_loader = DataLoader(LeukemiaLoader('train',None), batch_size=64, shuffle=True, num_workers=2)`

C. Describing your evaluation through the confusion matrix

Resnet18	Resnet50	Resnet152
----------	----------	-----------



從上表可以看出我的模型比較不利於辨識出正常細胞，這可能是因為我資料前處理中主要是想要凸顯細胞中的白色區塊，讓模型可以分辨出白色區塊很多或很大塊的就是白血病患者，可能有少數正常細胞的白色區塊跟白血病患者差不多大，導致模型很難分辨，才讓一些正常細胞被模型分類錯誤。

3. Data Preprocessing

A. How you preprocessed your data?

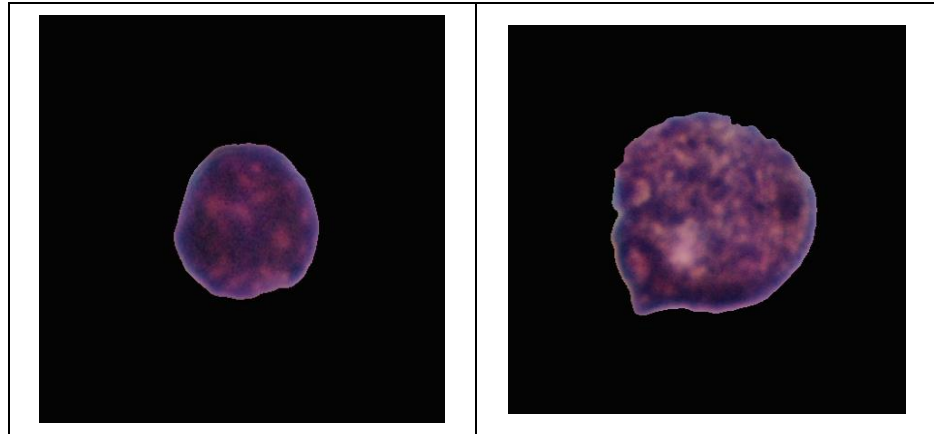
我的資料前處理方法分為以下幾個步驟

(1) clahe_enhancement

```
def clahe_enhancement(img):
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(9,9))
    channels = cv2.split(img)
    channels = list(channels) # Convert tuple to list
    for i in range(len(channels)):
        channels[i] = clahe.apply(channels[i])
    img_enhanced = cv2.merge(channels)
    return img_enhanced
```

CLAHE 是一種影像增強技術，用於改善影像的對比度和可視性，該方法在增強局部對比度和保留影像明亮和暗淡區域的細節方面特別有效。

正常細胞_原圖	白血病患者細胞_原圖
正常細胞_CLAHE	白血病患者細胞_CLAHE

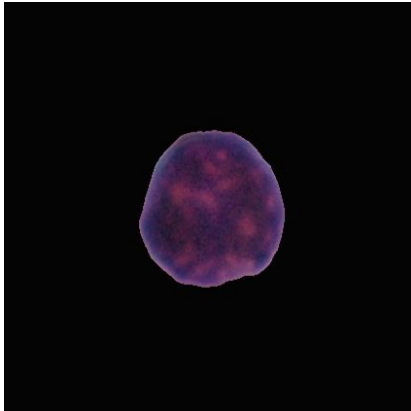
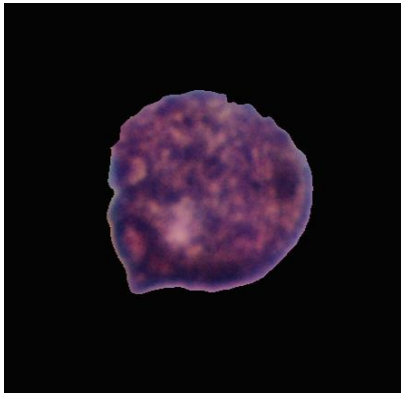


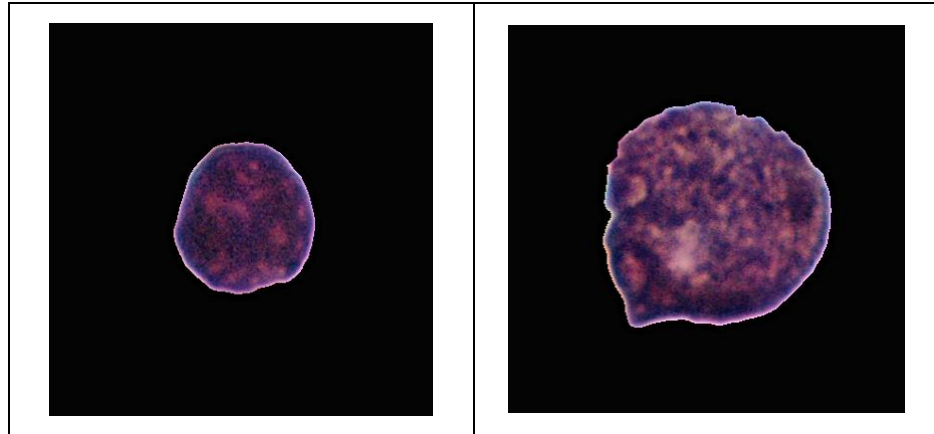
從上表可以看出白血病細胞經過 CLAHE 後，原本隱藏在細胞中的白色紋理會被凸顯出來，正常細胞則沒那麼明顯，這個白色紋理特徵可以當作分辨正常細胞跟白血病細胞的特徵之一。

(2) unsharp_mask

```
def unsharp_mask(image, sigma, strength):
    blurred = cv2.GaussianBlur(image, (0, 0), sigma)
    sharpened = cv2.addWeighted(image, 1.0 + strength, blurred, -strength, 0)
    return sharpened
```

Unsharp Mask 是另一種影像增強技術，用於增強影像中的邊緣和細節，這個方法用 GaussianBlur 作為一個 mask，可以有效地凸顯了那些有著顯著差異的區域，使得邊緣和細節更加明顯。我這邊 sigma 設 3，strength 設 1.5。

正常細胞_CLAHE	白血病細胞_CLAHE
	
正常細胞_CLAHE_Unsharp Mask	白血病細胞_CLAHE_Unsharp Mask



從上表可以看出經過 CLAHE 跟 Unsharp Mask 之後雖然增加了一些 noise，可是細胞表面白色的部分以及細胞的邊緣更明顯了，這樣更利於透過表層紋理及細胞外型，辨識是白血病細胞還是正常細胞。

(3) Data Augmentation

```
if self.mode == 'train':
    self.transform = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(20),
        transforms.CenterCrop(270),
        transforms.Resize((224, 224)),
        #GaborFilter(size=5, sigma=1.0, theta=0, lamdb=1.0, gamma=0.5, psi=0),
        #HighPassFilter(size=3),
        transforms.ColorJitter(brightness=0.2, contrast=1.2, saturation=1.2, hue=0.1),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.0026741, 0.00156906, 0.00272411], std=[0.00314106, 0.00195709, 0.00304743])
    ])
else:
    self.transform = transforms.Compose([
        transforms.CenterCrop(270),
        transforms.Resize((224, 224)),
        #GaborFilter(size=5, sigma=1.0, theta=0, lamdb=1.0, gamma=0.5, psi=0),
        #HighPassFilter(size=3),
        #transforms.ColorJitter(brightness=0.2, contrast=1.2, saturation=1.2, hue=0.1),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.0026741, 0.00156906, 0.00272411], std=[0.00314106, 0.00195709, 0.00304743])
    ])
```

Data Augmentation 主要是用來避免 overfitting。在這部分我採用了水平翻轉、垂直翻轉和角度變化等方法，來提高模型對圖片不同視角的理解。我也使用了 ColorJitte，特別是提高了對比度和飽和度的值。這樣做的原因是在我先前的前處理中，我嘗試讓細胞圖的顏色和紋理更為突出。通過增加對比度和飽和度，我能夠增加圖像中的顏色和紋理變異性，從而提升模型對這些特徵的識別能力。我也採用了 Resize、CenterCrop 和 Normalize 等技術。因為原始圖片尺寸較大，訓練模型需要較長的時間。通過 Resize 到較小的圖片，我能夠加快模型的訓練速度。由於 Resize 可能會導致圖片失去一些重要特徵，所以我先使用 CenterCrop 來裁去圖片的黑色區域，這樣在 Resize 後圖片損失的特徵就較少，進一步助於提高測試精度。最後 Normalize 有助於模型更快地收斂，改善梯度下降的性能，並減少數值計算上的不穩定性。由於我的模型是使用 64 的 batch size 進行訓練的，所以我在計算 Normalize 中的均值和標準差時也使用了 64 的批次大小。以下是我用來計算均值和標準差的程式碼。


```

dataloader = DataLoader(LeukemiaLoader('train',None), batch_size=64, shuffle=True)
def calculate_batch_mean_std(dataloader):
    total_images = 0
    batch_mean = np.zeros(3)
    batch_std = np.zeros(3)

    for batch, _ in dataloader:
        # Assumes the data is in format (batch_size, channels, height, width)
        batch = batch.permute(1, 0, 2, 3) # Change to (channels, batch_size, height, width)
        total_images += batch.shape[1]

        for channel in range(3):
            batch_mean[channel] += batch[channel, :, :, :].mean()
            batch_std[channel] += batch[channel, :, :, :].std()

    avg_mean = batch_mean / total_images
    avg_std = batch_std / total_images

    return avg_mean, avg_std

mean, std = calculate_batch_mean_std(dataloader)
print(f'Mean: {mean}, Std: {std}')

```

B. What makes your method special?

(1) 特徵凸顯

細胞圖經過 CLAHE 跟 Unsharp Mask 後，白血病細胞跟正常細胞會差異更大，白血病細胞的邊緣跟白色紋理都會很明顯，正常細胞則不會，有利於模型辨識。

(2) 模型訓練速度快速且盡可能保留更多特徵

因為有用 Resize 所以可以讓模型訓練速度更快，CenterCrop 切掉黑色區域後，會讓 Resize 後損失的特徵更少。Normalize 的部分因為是提前算好的，不用每個 epoch 都算一遍，也可以讓模型訓練速度更快，且是按照模型訓練時的 batch size 去計算的，不會跟每個 epoch 都計算一次的數值差太多。

(3) 提高模型穩定性

有使用水平翻轉、垂直翻轉和角度變化等 Data Augmentation，可以盡量減少 overfitting，在 ColorJitte 中，更是特別提高了對比度和飽和度的值，增加圖像中的顏色和紋理變異性，從而提升模型對這些特徵的識別能力。

(4) ResNet18、50、152 前處理方式有細微差別

Resnet18：原圖 resize 成 224*224、Normalize 的 mean、std 用 batch size 64 去算。

Resnet50：先將原圖 resize 成 128*128，模型訓練完 550epoch 後，將模型載入 valid accuracy 最高的權重，再將原圖 resize 成 224*224 後繼續訓練模型。兩種前處理 Normalize 的 mean、std 皆用 batch size

64 去算。

Resnet152：先將原圖 resize 成 128*128，模型訓練完 550epoch 後，將模型載入 valid accuracy 最高的權重，再將原圖 resize 成 224*224 後繼續訓練模型。第一種前處理 Normalize 的 mean、std 用 batch size 64 去算，第二種前處理 Normalize 的 mean、std 用 batch size 32 去算。

三種模型之所以有這些差別是因為如果直接用 224*224 去訓練 resnet50、resnet152，一個 epoch 就會花非常久的時間，所以我先 resize 成 128*128，雖然會損失不少資訊，但是可以把模型的權重先大致調的不錯，再用 224*224 去訓練，可以比直接用 224*224 去訓練減少不少時間，至於 Resnet152 的圖片 resize 成 224*224 後 Normalize 的 mean、std 用 batch size 32 去算，是因為用 64 的話我的 GPU 會 out of memory。

4. Experimental results

A. The highest testing accuracy

(1) Resnet18

312551093



0.97844

(2) Resnet50

312551093



0.97000

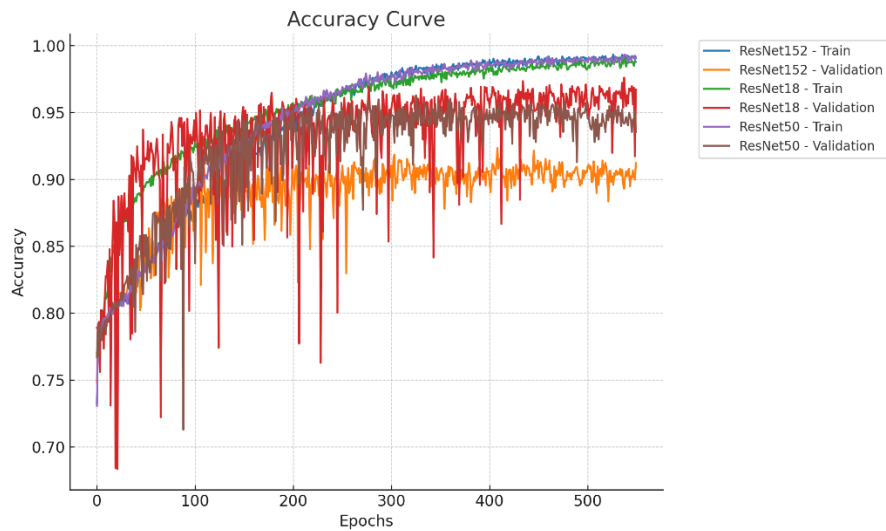
(3) Resnet152

312551093

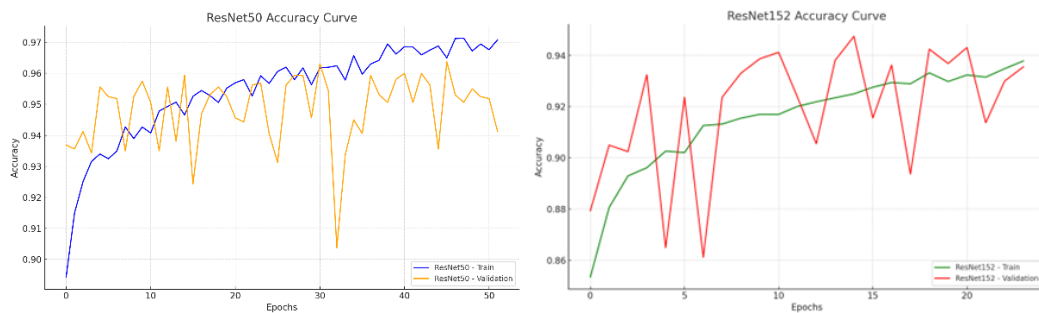


0.95313

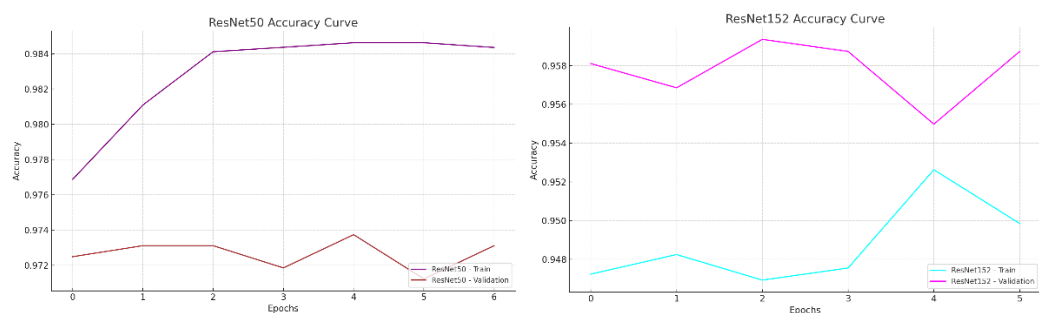
B. Comparison figures



上圖是 resnet18、50、152，跑 550epoch 的 train 跟 valid 的 accuracy curve，從上圖可以看到 resnet50、152 的 valid accuracy 並沒有像 resnet18 樣高，這是因為這部分的 resnet50、152 是用 resize 成 128*128 的圖片，因為圖片縮小太多，造成特徵損失嚴重，最後模型有點 overfitting。



上兩張圖是 resnet50、152 跑完 550epoch 後，載入最佳 valid accuracy 的模型參數，再將原始圖片 resize 成 224*224 後，訓練模型時產生的 train 跟 valid 的 accuracy curve，可以看出因為 resize 成比 128*128 更大的圖片後特徵損失減少，valid accuracy 也變得更高了。



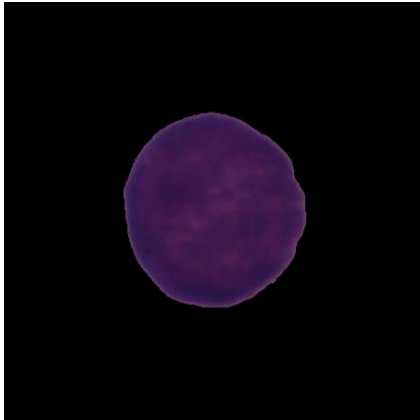
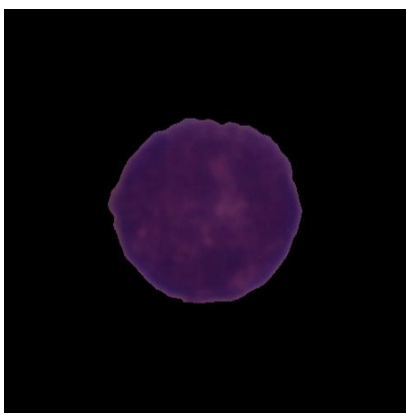
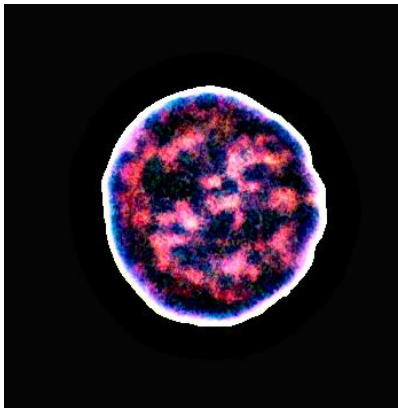
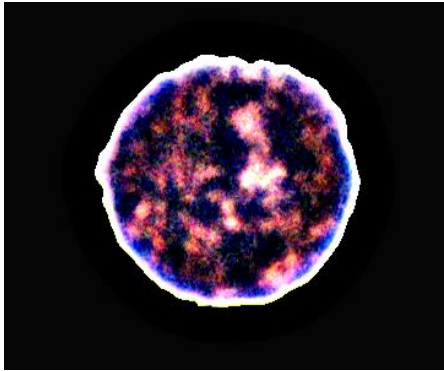
上圖是 resnet50、152 跑完上上圖的 epochs 後，將學習率調降 10 倍，就是從 0.005 調成 0.0005 的 train 跟 valid 的 accuracy curve，可以看出因為上上圖的 valid accuracy 一直在震盪，並且沒什麼在上升，所以我想調低學習率，看看能不能更容易抵達 loss 的最低點，從上圖可以看出結果顯然是蠻好的，loss 更接近最低點，

valid 的 accuracy 都比之前來的更高，這個方法是參照 resnet 論文中寫的訓練技巧。

5. Discussion

A. Anything you want to share


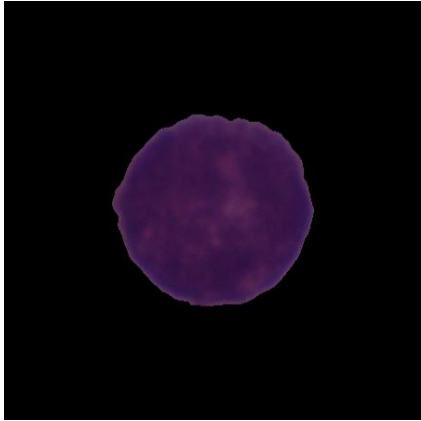
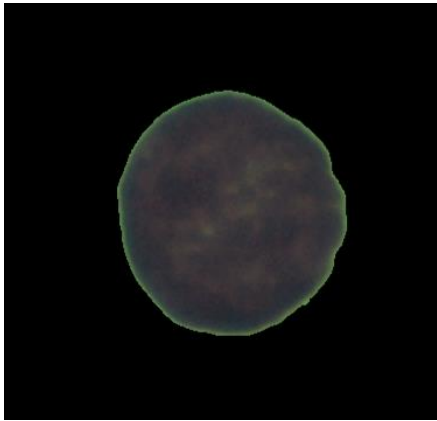
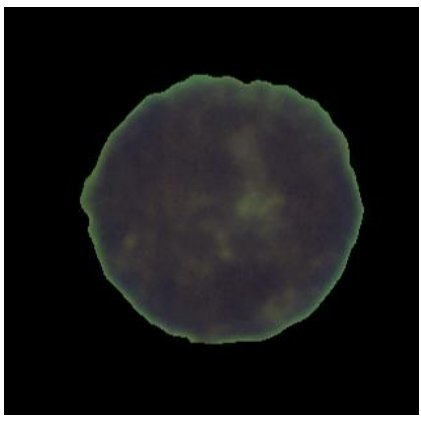
(1) 調整參數

正常細胞_原圖	白血病細胞_原圖
	
正常細胞_CLAHE_Unsharp Mask	白血病細胞_CLAHE_Unsharp Mask
	

上圖的影像處理方法跟我前處理用的完全一模一樣，只有調動參數，CLAHE 的參數設 clipLimit=4.0, tileGridSize=(9,9)，Unsharp Mask 的參數設 sigma=12, strength=5，我原本想透過這個方法，讓白血病細胞白色的部分更明顯，更有助於模型分類，可是後來發現細胞表層很多特徵都沒了，而且 noise 很多，反而不利於模型訓練。

(2) 白平衡

正常細胞_原圖	白血病細胞_原圖
---------	----------


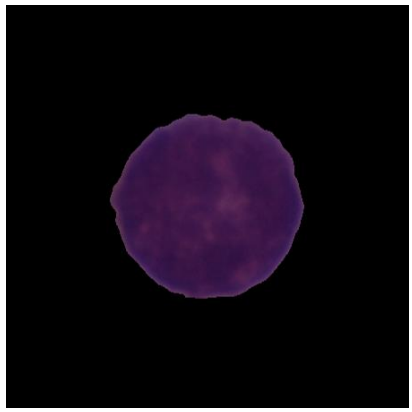
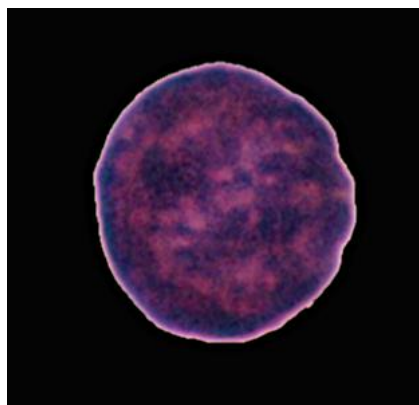
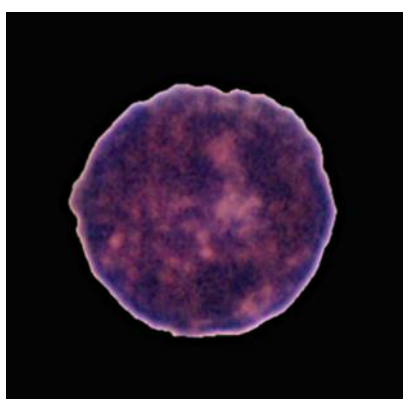
	
正常細胞_白平衡	白血病細胞_白平衡
	

我原本想用白平衡消除有些圖片有的光照錯誤，但是實驗後我發現可能是因為它讓細胞表面的顏色變得不太一樣，損失了一些原本模型可以判斷的特徵，導致更難分類了。以下是我白平衡的程式碼。

```
def simple_white_balance(img):
    avg_b = np.average(img[:, :, 0])
    avg_g = np.average(img[:, :, 1])
    avg_r = np.average(img[:, :, 2])
    avg = (avg_b + avg_g + avg_r) / 3
    scale_b = avg / avg_b
    scale_g = avg / avg_g
    scale_r = avg / avg_r
    img[:, :, 0] = cv2.multiply(img[:, :, 0], scale_b)
    img[:, :, 1] = cv2.multiply(img[:, :, 1], scale_g)
    img[:, :, 2] = cv2.multiply(img[:, :, 2], scale_r)
    return img
```

(3) Mean Filtering

正常細胞_原圖	白血病細胞_原圖
---------	----------

	
正常細胞_mean fitter_CLAHE_Unsharp Mask	白血病細胞_mean fitter _CLAHE_Unsharp Mask
	

我原本想用 mean fitter 消除一些圖片中的 noise，但結果發現他把細胞表層模糊太多，導致模型更難萃取出正確的特徵來進行分類，使得模型效能變低。我上圖的 CLAHE、Unsharp Mask 設的參數都跟我前處理的一樣，mean fitter 參數設 `cv2.boxFilter(img, -1, (3,3), normalize=True)`。