

NYCU DLP Lab3 – Leukemia Classification

312551086 張紀睿

1. Introduction

本次作業中，透過 pytorch 實作 ResNet18, ResNet50 以及 ResNet152，了解 ResNet 的模型架構以及 residual connection 對於模型訓練的幫助，並實作 dataloader(Dataset)用以輸入資料集，對輸入圖片進行 preprocessing，比如轉換為 tensor，以及各種 data augmentation。最後，透過分析 confusion matrix 了解模型表現，並調整訓練方式。

2. Implementation Details

A. The details of your model (ResNet)

ResNet 18

在 ResNet18 的部分，我先建構了 BasicBlock 的 class: ResBlock18，可傳入輸入與輸出 channel，根據傳入的 downsample 決定是否加入 downsample 的 layer 來使 residual 的 shape 與輸出符合。建立好 basicblock 之後，再堆疊 basicblock 以建構出 ResNet18。

```
class ResBlock18(nn.Module):
    def __init__(self, input_size, output_size, stride = 1, downsample = False):
        super().__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(input_size, output_size, kernel_size = 3, stride = stride, padding = 1, bias = False),
            nn.BatchNorm2d(output_size),
            nn.ReLU(),
            nn.Conv2d(output_size, output_size, kernel_size = 3, stride = 1, padding = 1, bias = False),
            nn.BatchNorm2d(output_size)
        )
        if downsample:
            self.downsample = nn.Sequential(
                nn.Conv2d(input_size, output_size, kernel_size = 1, stride = stride, bias = False),
                nn.BatchNorm2d(output_size)
            )
        else:
            self.downsample = None

        self.relu = nn.ReLU()

    def forward(self, x):
        if self.downsample is not None:
            residual = self.downsample(x)
        else:
            residual = x
        x = self.conv_layers(x)
        x = x + residual
        x = self.relu(x)
        return x
```

```

class ResNet18(nn.Module):
    def __init__(self):
        super().__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()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride = 2, padding = 1)
        self.conv2_x = nn.Sequential(
            ResBlock18(64, 64, 1, False),
            ResBlock18(64, 64, 1, False)
        )
        self.conv3_x = nn.Sequential(
            ResBlock18(64, 128, 2, True),
            ResBlock18(128, 128, 1, False)
        )
        self.conv4_x = nn.Sequential(
            ResBlock18(128, 256, 2, True),
            ResBlock18(256, 256, 1, False)
        )
        self.conv5_x = nn.Sequential(
            ResBlock18(256, 512, 2, True),
            ResBlock18(512, 512, 1, False)
        )

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, 2)
    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.conv2_x(x)
        x = self.conv3_x(x)
        x = self.conv4_x(x)
        x = self.conv5_x(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

```

ResNet 50, 152

在 ResNet50 與 ResNet152 的部分，我先建構了 BottleneckBlock 的 class: ResBlock50，同樣可傳入輸入與輸出 channel，並根據傳入的 downsample 決定是否加入 downsample 的 layer 來使 residual 的 shape 與輸出符合。建立好 bottleneckblock 之後，再堆疊 bottleneckblock 以建構出 ResNet50 與 152。由於 ResNet152 的 layer 所需要的 block 數較多，建構時以 for loop 來堆疊，ResNet50 則直接手動建構。

```

class ResBlock50(nn.Module):
    def __init__(self, input_size, first_size, output_size, stride = 1, downsample = False):
        super().__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(input_size, first_size, kernel_size = 1, stride = 1, padding = 0, bias = False),
            nn.BatchNorm2d(first_size),
            nn.Conv2d(first_size, first_size, kernel_size = 3, stride = stride, padding = 1, bias = False),
            nn.BatchNorm2d(first_size),
            nn.Conv2d(first_size, output_size, kernel_size = 1, stride = 1, padding = 0, bias = False)
        )
        if downsample:
            self.downsample = nn.Sequential(
                nn.Conv2d(input_size, output_size, kernel_size = 1, stride = stride, bias = False),
                nn.BatchNorm2d(output_size)
            )
        else:
            self.downsample = None

        self.relu = nn.ReLU()

    def forward(self, x):
        if self.downsample is not None:
            residual = self.downsample(x)
        else:
            residual = x
        x = self.conv_layers(x)
        x = x + residual
        x = self.relu(x)
        return x

```

```

class ResNet50(nn.Module):
    def __init__(self):
        super().__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()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride = 2, padding = 1)
        self.conv2_x = nn.Sequential(
            ResBlock50(64, 64, 256, 1, True),
            ResBlock50(256, 64, 256, 1, False),
            ResBlock50(256, 64, 256, 1, False)
        )
        self.conv3_x = nn.Sequential(
            ResBlock50(256, 128, 512, 2, True),
            ResBlock50(512, 128, 512, 1, False),
            ResBlock50(512, 128, 512, 1, False),
            ResBlock50(512, 128, 512, 1, False)
        )
        self.conv4_x = nn.Sequential(
            ResBlock50(512, 256, 1024, 2, True),
            ResBlock50(1024, 256, 1024, 1, False),
            ResBlock50(1024, 256, 1024, 1, False),
            ResBlock50(1024, 256, 1024, 1, False),
            ResBlock50(1024, 256, 1024, 1, False),
            ResBlock50(1024, 256, 1024, 1, False)
        )
        self.conv5_x = nn.Sequential(
            ResBlock50(1024, 512, 2048, 2, True),
            ResBlock50(2048, 512, 2048, 1, False),
            ResBlock50(2048, 512, 2048, 1, False)
        )
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(2048, 2)

```

```

class ResNet152(nn.Module):
    def __init__(self):
        super().__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()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride = 2, padding = 1)
        conv2_block = []
        conv2_block.append(ResBlock50(64, 64, 256, 1, True))
        for i in range(2):
            conv2_block.append(ResBlock50(256, 64, 256, 1, False))
        self.conv2_x = nn.Sequential(*conv2_block)

        conv3_block = []
        conv3_block.append(ResBlock50(256, 128, 512, 2, True))
        for i in range(7):
            conv3_block.append(ResBlock50(512, 128, 512, 1, False))
        self.conv3_x = nn.Sequential(*conv3_block)

        conv4_block = []
        conv4_block.append(ResBlock50(512, 256, 1024, 2, True))
        for i in range(35):
            conv4_block.append(ResBlock50(1024, 256, 1024, 1, False))
        self.conv4_x = nn.Sequential(*conv4_block)

        conv5_block = []
        conv5_block.append(ResBlock50(1024, 512, 2048, 2, True))
        for i in range(2):
            conv5_block.append(ResBlock50(2048, 512, 2048, 1, False))
        self.conv5_x = nn.Sequential(*conv5_block)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(2048, 2)

    def forward(self, x): ...

```

B. The details of your Dataloader

```
class LeukemiaLoader(data.Dataset):
    def __init__(self, root, mode, model = "18", aug = True):
        self.root = root
        self.img_name, self.label = getData(mode, model)
        self.mode = mode
        if 'train' in mode and aug:
            self.transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.RandomRotation(degrees = (0,90)),
                transforms.RandomHorizontalFlip(p = 0.5),
                transforms.RandomVerticalFlip(p = 0.5),
                transforms.Normalize((0.0,), (255.0,))
            ])
        else:
            self.transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize((0.0,), (255.0,))
            ])

        #print("> Found %d images..." % (len(self.img_name)))

    def __len__(self):
        """return the size of dataset"""
        return len(self.img_name)

    def __getitem__(self, index):
        """something you should implement here"""
        img_path = self.root + self.img_name[index]
        img = Image.open(img_path).convert('RGB')
        img = self.transform(img)
        if self.mode == "test":
            return img
        label = torch.tensor(self.label[index]).float()
        return img, label
```

```
def getData(mode, model = "18"):
    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

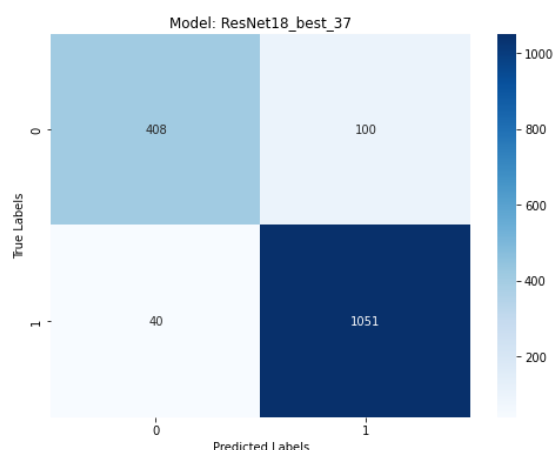
    elif mode == "train_all":
        df = pd.read_csv('train.csv')
        path = df['Path'].tolist()
        label = df['label'].tolist()
        df = pd.read_csv('valid.csv')
        path += df['Path'].tolist()
        label += df['label'].tolist()
        return path, label
```

- 在 getData 部分，除了原有的 train, valid, test mode 以外，新增了“train_all”。此模式下會將 train 與 valid.csv 的資料合併在一起，用意是使用所有有 label 的資料訓練模型，以獲得更好的預測效果。
- 在 transform 的部分，原本考慮加入 colorjitter、random crop、gaussian blur 等轉換，但觀察訓練資料認為若加入這些雖然能使訓練資料更豐富，但可能反而學習到一些對於預測沒有幫助的特徵，因此最後僅使用隨機翻轉與旋轉。另外，可以將 aug 設為 False 來關閉隨機翻轉的 augmentation。

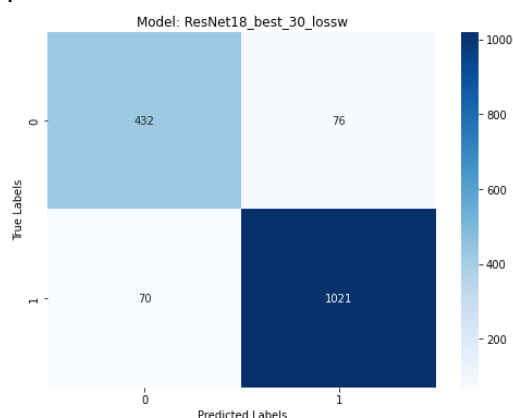
C. Describing your evaluation through the confusion matrix

由於訓練時間有限，此部分主要針對 ResNet18 進行調整，找到表現較佳的設定後再用以訓練 ResNet50, ResNet152，並根據模型微調參數。

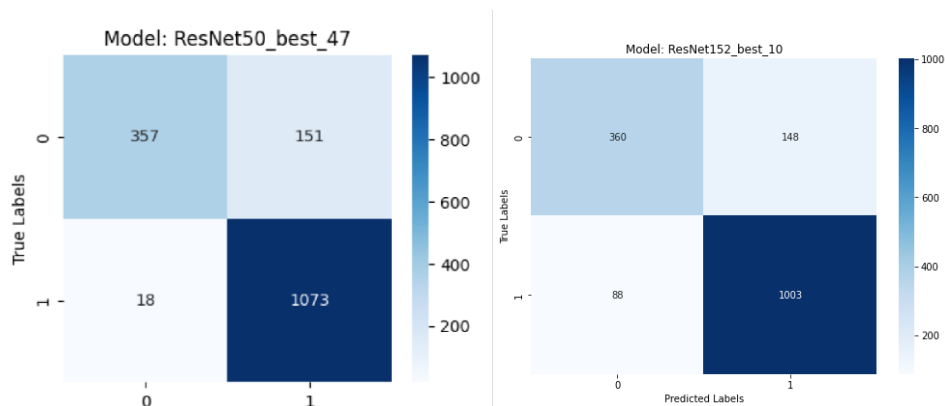
最初使用 nn.CrossEntropy 訓練得到以下 confusion matrix，觀察後發現由於資料集中有 data imbalance 的問題，導致模型對於 label 0 的預測效果較差，導致在遇到 label 0 時模型仍容易預測為 1。



因此，我調整了 loss function 設定，將 weights 設為[1.2, 1.0]以提升 label 0 的權重(原本設為[1.5,0.7])，但發現會使訓練更不穩定)。調整設定訓練完成後，得到以下 confusion matrix，雖然 FN 略為增加，但由於減少了 FP 整體 performance 得到了進步，證明了此次調整的成效。



附上 ResNet50 與 ResNet152 的 confusion matrix，推測由於所需訓練時間較長，因此表現反而略差於 ResNet18。



另外，由於最終用來上傳 test 的模型會再用 train 與 valid set 結合而成的資料 finetune，因此最終表現會再比 confusion matrix 顯示的進步一些。

3. Data Preprocessing

A. How you preprocessed your data?

用來訓練的資料會先轉換為 tensor，再經過隨機角度的旋轉(0~90 度)，並有一半的機率會水平翻轉，一半的機率會垂直翻轉，最後再經過 normalize 以讓模型更好做訓練。若 mode 選擇 train_all 的話，會載入 train 與 valid set 中的所有圖片用來做訓練；另外若將 aug 設為 false 或者資料並非用於訓練時，則只會經過 to tensor 與 normalize 兩項 transform。




B. What makes your method special?

在 Augmentation 方面並沒有加入過多要素，希望保留原資料集影像的特徵，因此僅做了角度的旋轉。另外，我認為有將 train 與 valid set 都載入訓練資料的選項，使模型在最終的表現有了再進步的空間。

4. Experimental results



A. The highest testing accuracy

Screenshots

	resnet18_output.csv Complete · 4d ago · 15	0.97469
	resnet50_output.csv Complete · 1d ago · finetune20	0.95688
	resnet152_output.csv Complete · 2d ago · 152loadbest	0.92596

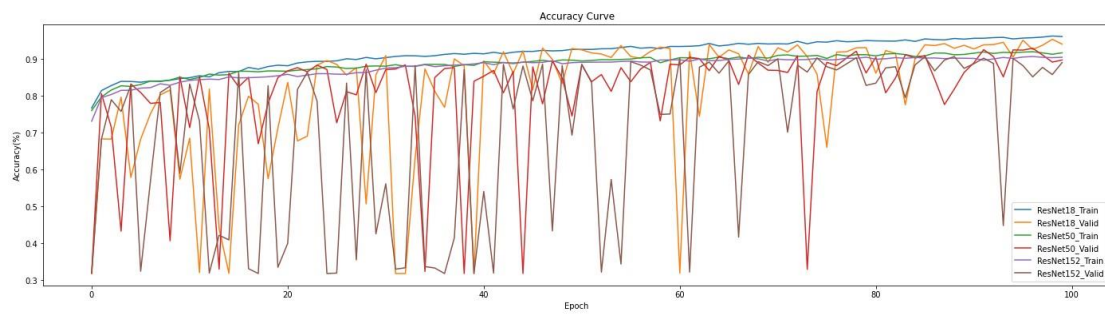
雖然在我所得到的結果中，模型越深反而得到了越低的 test score，但我認為與訓練不足有所關係，較深的模型訓練一個 epoch 所需時間較長，加上我最初也是以 ResNet18 為基準去調整訓練方向(Loss Weight 等等)，因此 ResNet18 才有高於其他兩個模型的結果。

另外，下圖為 ResNet18 與 ResNet50 在沒有經過 train + valid set finetune 時的最佳表現，

	resnet18_output.csv Complete · 5d ago · 95pa no valid	0.95876
	resnet50_output.csv Complete · 1d ago	0.94657

比較最佳結果發現，經過 finetune 後 ResNet18 進步了大約 1.6%，ResNet50 也有了 1%的進步，顯示了在訓練後期準確率提升緩慢的情況下，加入 valid set 是一個可行的做法。可惜的是 ResNet152 沒有足夠的時間進行 finetune，因此在此處無法加入比較。

B. Comparison figures



模型訓練設置如下。

Optimizer: Adam, Loss function: CrossEntropy (loss weight: 1.2, 1.0),

Batch size: 16, 16, 8 (ResNet18, ResNet50, ResNet152),

Learning Rate: [5e-3, 1e-3], [1e-3, 1e-4], [1e-3, 1e-4] · (ResNet18,

ResNet50, ResNet152, [前 50 epoch, 後 50 epoch])

關於 Valid 表現與 LR 調整的關係會在 discussion 部分深入探討。

5. Discussion

- 是否加入 Data Augmentation: 若不使用 Data Augmentation，每個 epoch 約可以減少 1 分鐘的訓練時間。然而，在我使用 ResNet18 實際測試訓練同樣 50 epoch 並比較 test score 時，使用 Data Augmentation 的 test score 進步了大約 2%，推測由於 Data Augmentation 會使模型學習到更豐富的資訊而有了較好的表現，因此雖然會使訓練時間略為增加，最終還是決定保留 Data Augmentation。
- Loss Function 的調整: 在 part2 的 C 有提到為了解決 data imbalance 導致模型預測不平衡的問題，我調整了 Loss function 中 label 的權重，然而可以從 ResNet50 與 152 的 confusion matrix 看出此改動對於這兩個模型並沒有明顯效果。因此，若時間充裕，我認為仍應該根據不同模型去做實驗與調整，而非將 ResNet18 測試出來的結果直接套用在其他模型上。
- Valid Accuracy 與 LR 調整: 起初由於考量到訓練時間有限，因此我將 learning rate 調的比較高，然而觀察 valid accuracy 就會發現，模型表現非常不穩定。由於此次使用的模型都是較深的模型，因此在 lr 太高的情況下，若是學習的方向不正確就很容易造成模型整個學到錯誤的資訊，而造成此種波動的結果。因此，在後 50 個 epoch 時，我將 lr 調低為原先的 1/5 或 1/10，以減少這種狀況，從 accuracy curve 可以看出雖然偶爾還是有這種狀況發生，但已經減少很多，valid accuracy 的起伏也變得較為平緩。
- 用所有資料 Finetune 的問題: 雖然將 train set 與 valid set 結合起來用於訓練可以讓模型學習到更多的資訊，但由於沒有 valid accuracy 來評斷模型訓練狀況，會不知道究竟參數是否適合，也無從得知是否 overfit 或是 underfit。因此，或許仍須留下一定比例的 valid data 來做驗證。