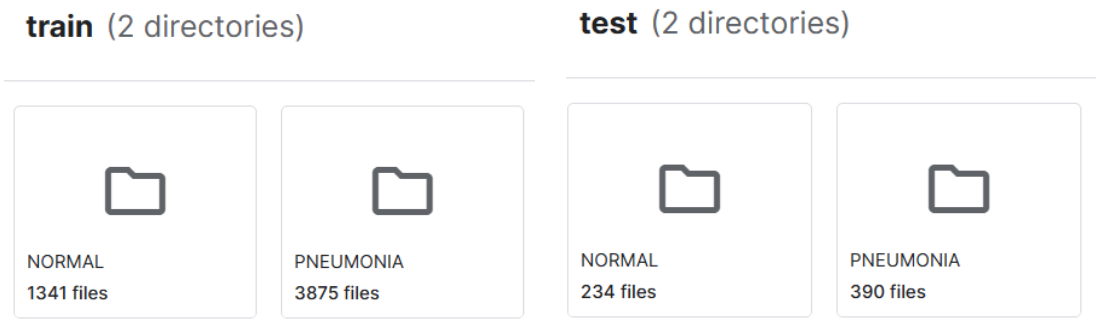
Artificial Intelligence on Medical Imaging Lab 1

312551089 彭筱竹

1. Introduction

本次報告實作 ResNet18、ResNet50 兩個神經網路架構，用以對 Chest X-Ray Images (Pneumonia) dataset 進行肺炎與正常肺部的分類，從下圖一可以觀察到，肺炎與正常肺部在訓練集與測試集的數量分布大約是 3：1 與 1.5：1，在實驗中，將自定義 DataLoader 用以對資料進行前處理，最後對兩種 ResNet 架構的預測準確率以及 Confusion Matrix 進行比較。



▲圖一、Chest X-Ray dataset

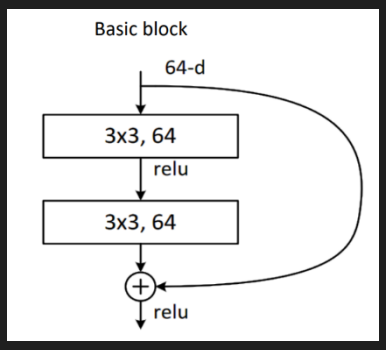
2. Implementation Details

A. The details of model (ResNet)

ResNet 利用 residual block 的設計來解決深度神經網路的退化問題，下圖二、三為其兩種 residual block 架構：Basic block、Bottleneck block，前者用於淺層網路 ResNet18，後者用於深層網路 ResNet50。

在一個 residual block 中，除了原本神經網路的主架構，另外還擁有一個捷徑連結，其中 identity 複製了淺層網路的特徵，並透過 Shortcut Connections 跳過一層或多層的連結，將上層的輸入加上神經網路主架構的輸出，使深層網路應該至少和淺層網路的性能一樣，另一方面能夠讓深層網路在輸入特徵的基礎上學習到新特徵，從而使神經網路擁有更好的性能。

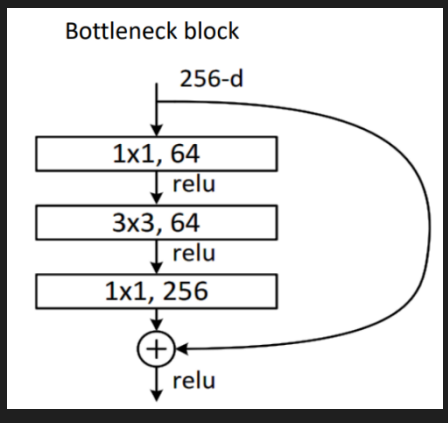
```python
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, downsample_stride):
        super(BasicBlock, self).__init__()

        if downsample_stride == None:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
            self.downsample = None
        else:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=2, padding=1, bias=False)
            self.downsample = DownSample(in_channels, out_channels, downsample_stride)

        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, inputs):
        identity = inputs

        x = self.bn1(self.conv1(inputs))
        x = self.relu(x)
        x = self.bn2(self.conv2(x))

        if self.downsample != None:
            identity = self.downsample(identity)
        x = self.relu(x + identity)

        return x
```



▲圖二、Basic block

Bottleneck block 是針對較深層的神經網路設計，其透過 1x1 的卷積把 256 維降至 64 維後，再通過 1x1 卷積升維恢復，以此來降低網路的參數量。

```python
class Bottleneck(nn.Module):
    extend = 4
    def __init__(self, in_channels, out_channels, downsample=None, stride=1):
        super(Bottleneck, self).__init__()

        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(out_channels, out_channels*self.extend, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(out_channels*self.extend)
        )
        self.relu = nn.ReLU()
        self.downsample = downsample

    def forward(self, inputs):

        identity = inputs
        x = self.block(inputs)

        if self.downsample != None:
            identity = self.downsample(inputs)

        x += identity
        x = self.relu(x)

        return x
```



▲圖三、Bottleneck block

◆ **ResNet18**

如下圖四所示，利用 BasicBlock 物件可以快速建立 ResNet18。

```
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()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = nn.Sequential(BasicBlock(64, 64, None), BasicBlock(64, 64, None))
        self.layer2 = nn.Sequential(BasicBlock(64, 128, (2, 2)), BasicBlock(128, 128, None))
        self.layer3 = nn.Sequential(BasicBlock(128, 256, (2, 2)), BasicBlock(256, 256, None))
        self.layer4 = nn.Sequential(BasicBlock(256, 512, (2, 2)), BasicBlock(512, 512, None))

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, 2)

    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 = self.fc(x.reshape(x.shape[0], -1))
        return x
```

▲圖四、ResNet18

◆ **ResNet50**

由於 ResNet50 架構的實現與 ResNet18 相比起來更為巨大，這裡將基本架構建立於圖五的 ResNet 物件內，再利用圖六的 make layer 重複生成 Bottleneck block 建構 ResNet50，而圖七的 DownSample 則在跨卷積層通道數變化時使用。

```
class ResNet(nn.Module):
    def __init__(self, Block, layer, num_classes, num_channels=3):
        super(ResNet, self).__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(num_channels, 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.layer1 = self.make_layer(Block, layer[0], base=64)
        self.layer2 = self.make_layer(Block, layer[1], base=128, stride=2)
        self.layer3 = self.make_layer(Block, layer[2], base=256, stride=2)
        self.layer4 = self.make_layer(Block, layer[3], base=512, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512*Block.extend, num_classes)

    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 = self.fc(x.reshape(x.shape[0], -1))
        return x
```

▲圖五、ResNet

```
def make_layer(self, Block, blocks, base, stride=1):
    downsample = None
    layers = []

    if stride != 1 or self.in_channels != base*Block.extend:
        downsample = DownSample(self.in_channels, base*Block.extend, stride)

    layers.append(Block(self.in_channels, base, downsample=downsample, stride=stride))
    self.in_channels = base*Block.extend

    for _ in range(blocks-1):
        layers.append(Block(self.in_channels, base))

    return nn.Sequential(*layers)
```

▲圖六、make layer

```
def DownSample(in_channels, out_channels, stride):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
        nn.BatchNorm2d(out_channels))
```

▲圖七、DownSample

B. The details of Dataloader

圖八的 getData 根據給定的模式讀取 csv 檔，並回傳資料集的圖片路徑及對應的標籤。

```
def getData(mode, netnum=None):
    df = pd.read_csv(f'{mode}.csv')
    path = df['path'].tolist()
    label = df['label'].tolist()
    return path, label
```

▲圖八、getData

Chest X-Ray dataset Loader 如圖九所示，通過繼承 torch.utils.data.Dataset 來自定義 Dataloader，__len__返回資料集的大小，__init__內宣告了包含 torchvision 提供的 RandomHorizontalFlip 等 transforms 方便進行資料增強，__getitem__從 index 獲取對應的路徑，並執行前處理對圖片進行變換，最後回傳修改後的資料和對應的標籤。

```python
class ChestLoader(data.Dataset):
    def __init__(self, root, mode, netnum=None):

        self.root = root
        self.mode = mode
        self.netnum = netnum

        self.img_name, self.label = getData(mode)

        if self.mode == 'train':
            self.transformations = trans.Compose([
                                                  trans.Resize((256, 256)),
                                                  trans.CenterCrop((224, 224)),
                                                  trans.RandomHorizontalFlip(),
                                                  trans.RandomRotation(20),
                                                  trans.ToTensor(),
                                                  trans.Normalize(mean=[0.5], std=[0.5])])
        else:
            self.transformations = trans.Compose([
                                                  trans.Resize((256, 256)),
                                                  trans.CenterCrop((224, 224)),
                                                  trans.ToTensor(),
                                                  trans.Normalize(mean=[0.5], std=[0.5])])
                                                  #trans.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])

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

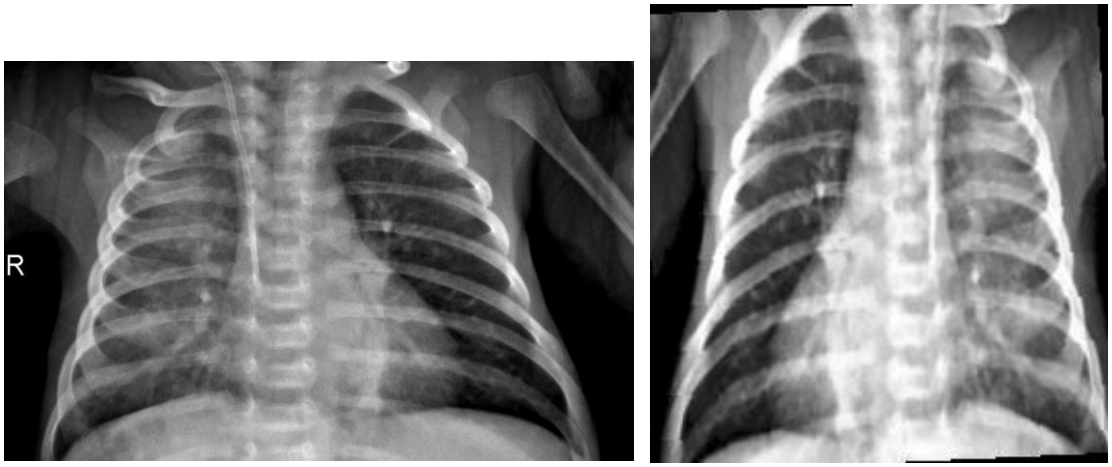
```python
def __getitem__(self, index):

    path = os.path.join(self.root, self.img_name[index])
    img = Image.open(path).convert('L')

    box = img.getbbox()
    region = img.crop((box[0], box[1], box[2], box[3]))

    # region = region.convert('HSV')
    # img = region.filter(ImageFilter.EDGE_ENHANCE)
    img = ImageOps.equalize(region)

    img = self.transformations(img)

    label = self.label[index]
    return img, label
```

▲圖九、ChestLoader

C. Data Preprocessing

在讀取圖片後，轉換成單通道的灰階影像，並剪裁掉多餘的黑邊，黑邊中可能包含會影響模型學習的雜訊，最大保留整個胸腔。ImageOps.equalize 可對整張影像作直方圖均衡化，能藉提高對比度的方式更加凸顯肺炎異常混濁的病灶區域。

接著利用事先宣告的 transforms 把原本圖片的尺寸調整為 256 x 256 的大小，經過前面的剪裁，必須重新將圖片調整到一致的大小，同時能縮小資料量，之後的 CenterCrop 可以切除不重要的手臂、腹部、頸部。接著作隨機的水平翻轉、角度旋轉，透過資料增強隨機產生相似但不同的資料，擴大資料規模，降低模型對某些特徵的依賴性，從而提高泛化能力，最後將圖片轉換成 Tensor 的資料型態並進行標準化，將特徵資料依照原本分布按比例縮放，讓資料落在特定區間，減小離群值對於模型的影響。



(a) (b)

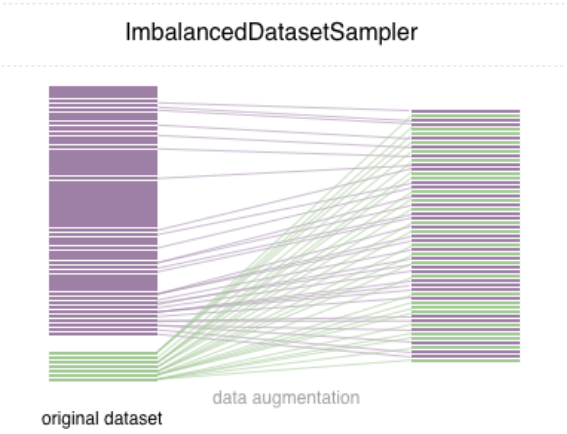
▲圖十、前處理：(a)處理前、(b)處理後

除此之外，在載入 DataLoader 時，使用 ImbalancedDatasetSampler 進行 resampling 並且計算 sampling 的權重。

```
# distribution of classes in the dataset
df = pd.DataFrame()
df["label"] = self._get_labels(dataset) if labels is None else labels
df.index = self.indices
df = df.sort_index()

label_to_count = df["label"].value_counts()

weights = 1.0 / label_to_count[df["label"]]

self.weights = torch.DoubleTensor(weights.to_list())
```

▲圖十一、ImbalancedDatasetSampler

為了解決資料不平衡的問題，進行 resampling 重新平衡正常肺部與肺炎的資料分佈，盡量確保訓練的模型不會偏向於具有更多資料的類別。ImbalancedDatasetSampler 會根據不同類別的數量去自動計算 sampling 的權重，這樣可以避免單純 under-sampling 或 over-sampling 丟失資訊和 overfitting 的問題。



3. Experimental results & Discussion

下列實驗結果均是訓練了 50 個 epoch。Batch size 設定為 64、Learning rate 設定為 1e-5、optimizer 設定為 Adam、loss 為 CrossEntropyLoss。

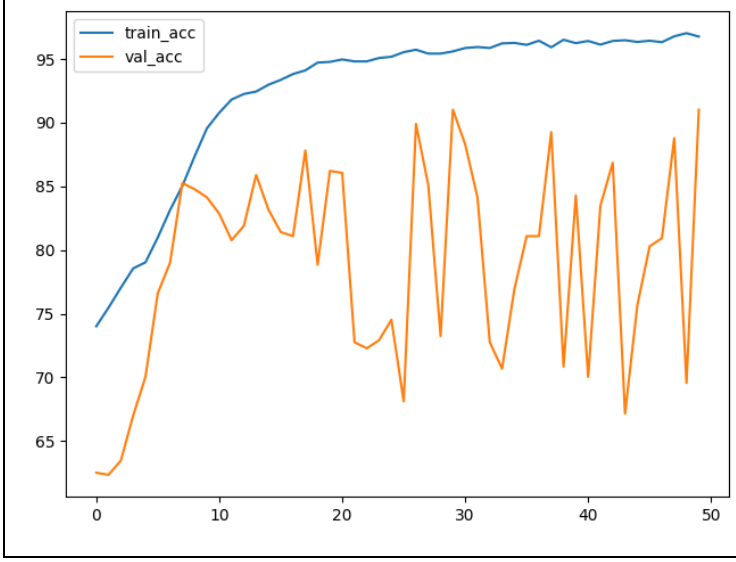
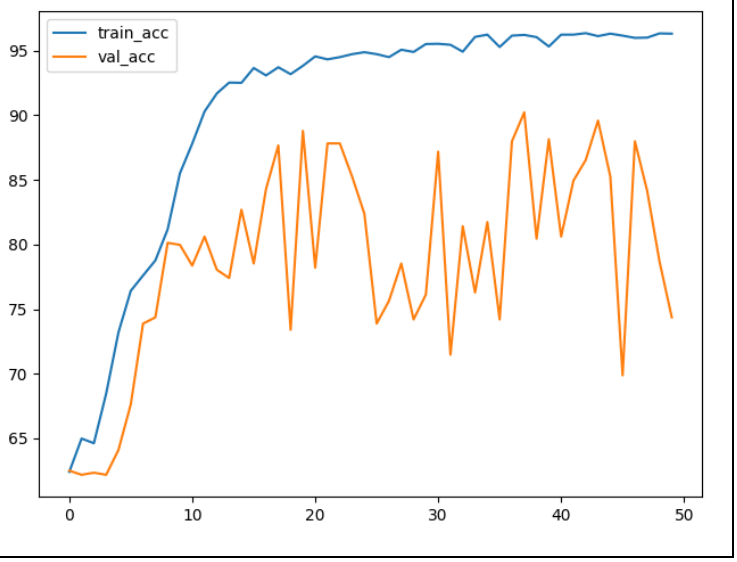
A. The highest testing accuracy

ResNet18	ResNet50
↳ Recall: 0.9308, Precision: 0.9260, F1-score: 0.9284 ↳ Test Acc.(%): 91.03% precision recall f1-score support 0 - NORMAL 0.88 0.88 0.88 234 1 - PNEUMONIA 0.93 0.93 0.93 390 accuracy 0.91 624 macro avg 0.90 0.90 0.90 624 weighted avg 0.91 0.91 0.91 624	↳ Recall: 0.9692, Precision: 0.8852, F1-score: 0.9253 ↳ Test Acc.(%): 90.22% precision recall f1-score support 0 - NORMAL 0.94 0.79 0.86 234 1 - PNEUMONIA 0.89 0.97 0.93 390 accuracy 0.90 624 macro avg 0.91 0.88 0.89 624 weighted avg 0.91 0.90 0.90 624
91.03%	**90.22%**

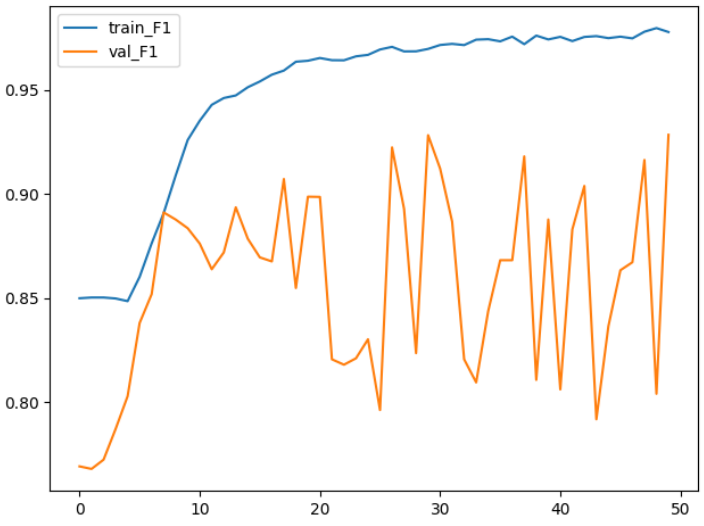
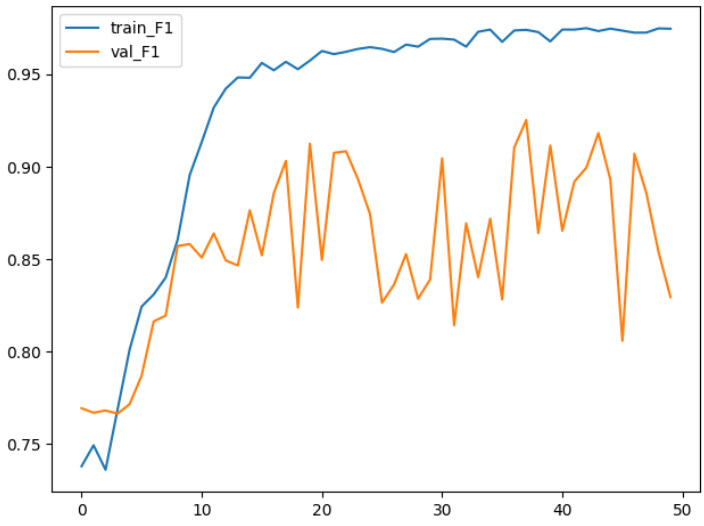
可以發現，對於 Pneumonia，recall 皆高於 precision，而對於 Normal cell，precision 皆高於 recall，我們最不希望 Pneumonia 被判斷為 Normal 的情況發生，因此 Pneumonia 的 precision 非常重要，所以就整體來說訓練了 50 個 epoch 的兩種 ResNet 架構，ResNet18 在 testing data 中 Pneumonia 的 precision 和 recall 都有達到 90%以上，幾乎該抓的都有抓到，但少數時候會抓錯。

B. Plotting the comparison figures

從圖中可以看到訓練資料的正確率穩定上升，但驗證資料卻出現震盪的現象，另外訓練資料平衡類別間的數量差異，但驗證資料依據真實情況肺炎的數量依舊占了一半以上。

ResNet18	ResNet50
	

▲training and validation accuracy

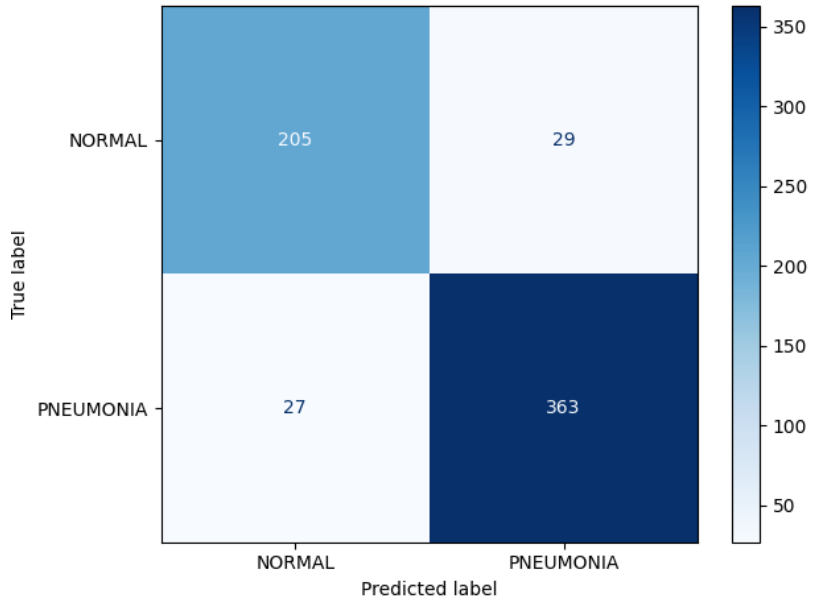
ResNet18	ResNet50
	

▲training and validation F1 score

C. Confusion matrix

confusion matrix 由 true label 和 predicted label 兩個維度組成，右上與左下的 False Positive 與 False Negative 對於分析模型對於不平衡 dataset 的分類有很大的幫助，通過 confusion matrix 可以方便看出模型是否將兩個不同的類別混淆，同時以利計算後續的 precision、recall 及 F1-score

從 confusion matrix 可以觀察到，TP 大於 TN、FP 大於 FN，模型更傾向於將圖片預測為 Pneumonia，由於 Pneumonia 的數量大於 Normal 的數量，對模型來說是更為安全的選擇。



• compute_class_weight

sklearn 提供的 compute_class_weight 能夠計算類別之間的權重，並在 loss 宣告時指定 weight 參數，數量較少的類別權重較高，使模型在訓練時 loss 能根據類別權重去計算，然而實驗出來的效果並不佳。

```
class_weights = compute_class_weight('balanced', np.unique(getData('valid')[1]), np.array(getData('valid')[1]))
class_weights = torch.tensor(class_weights, dtype=torch.float).to(device)
loss = nn.CrossEntropyLoss(weight=class_weights, reduction='mean')
```

▲compute_class_weight

• 前處理

除了上述 2.C，本次報告也實驗了使用高斯濾波去除胸腔內的雜訊，以及顏色反轉、增強對比度的方法，推測前者可能過濾掉了重要的資訊，而後者使雜訊被強調出來，兩者訓練出來的正確率皆不高。

• 黑邊裁剪

本次報告也試著實現尋找圖片上下左右的第一個不為零的像素去做黑邊裁剪，不過發現使用 numpy 的效率不佳，遂改為 PIL 的 getbbox，使計算的速度有所提升。

```
box = np.nonzero(img)
obj = img[min(box[0]):max(box[0]), min(box[1]):max(box[1])]
```

▲numpy crop

4.Github Link

https://github.com/pengemma/AIMI/tree/main/LAB1