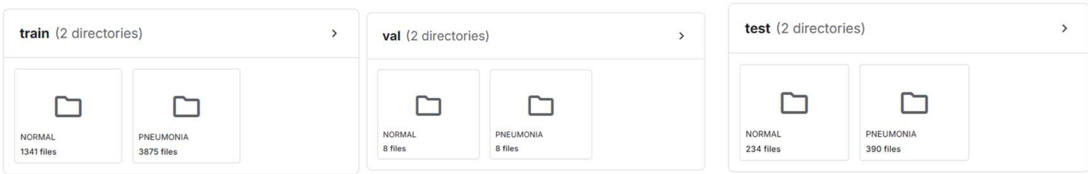
Artificial Intelligence on Medical Imaging Lab 1

314553020 許良亦

1. Introduction

本次報告實作了三種影像分類模型：ResNet-18、ResNet-50，以及採用 ImageNe 預訓練權重的 DenseNet-121，並用於 Chest X-Ray Images (Pneumonia) 資料集的(肺炎 vs.正常)二分類。由圖一可見，類別分布約為：訓練集 3:1、驗證集 1:1、測試集 1.5:1。實驗中以自訂的 DataLoader 進行資料前處理，最終比較三種模型的預測準確率與混淆矩陣表現。



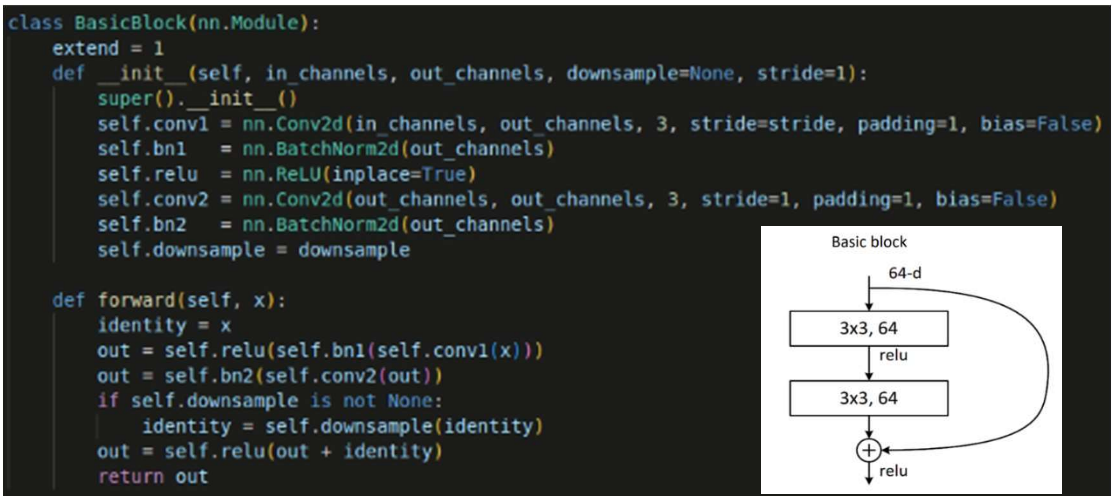
▲ 圖一、Chest X-Ray dataset

2. Experiment setups

A. The detail of your model

ResNet 以殘差（residual）設計緩解深層網路的退化問題；其殘差模組分為兩型：Basic Block 與 Bottleneck。前者多用於較淺的 ResNet-18，後者則配置於較深的 ResNet-50（如下圖二、三）。

在一個 residual block 裡，除了主幹的卷積堆疊，還有一條 shortcut 把輸入直接傳到輸出端。這條捷徑可視為 identity 分支，會跨越一層或多層，最後與主幹分支學到的映射相加，形成 (y＝F(x)＋x)。透過這個操作，深層模型至少能保有與淺層相當的表現，同時更容易在既有特徵 (x) 的基礎上只學差值（殘差），讓優化更穩定、梯度傳遞更順暢，也更有機會得到更好的整體性能。

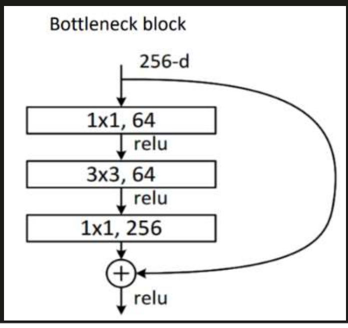


```python
class BasicBlock(nn.Module):
    extend = 1
    def __init__(self, in_channels, out_channels, downsample=None, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, 3, stride=stride, padding=1, bias=False)
        self.bn1   = nn.BatchNorm2d(out_channels)
        self.relu  = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, 3, stride=1, padding=1, bias=False)
        self.bn2   = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        if self.downsample is not None:
            identity = self.downsample(identity)
        out = self.relu(out + identity)
        return out
```


▲ 圖二、Basic block

Bottleneck block 先用 1×1 卷積壓縮通道，再以 3×3 卷積在較低通道數下提取空間特徵，最後用 1×1 卷積擴張通道。

```python
class Bottleneck(nn.Module):
    extend = 4
    def __init__(self, in_channels, out_channels, downsample=None, stride=1):
        super().__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels * self.extend, 1, bias=False),
            nn.BatchNorm2d(out_channels * self.extend),
        )
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        out = self.block(x)
        if self.downsample is not None:
            identity = self.downsample(x)
        out = self.relu(out + identity)
        return out
```



▲ 圖三、Bottleneck block

```python
class ResNet(nn.Module):
    def __init__(self, Block, layers, num_classes, num_channels=3, dropout=0.3):
        super().__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(num_channels, 64, 7, stride=2, padding=3, bias=False)
        self.bn1   = nn.BatchNorm2d(64)
        self.relu  = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.layer1 = self._make_layer(Block, layers[0], base=64,  stride=1)
        self.layer2 = self._make_layer(Block, layers[1], base=128, stride=2)
        self.layer3 = self._make_layer(Block, layers[2], base=256, stride=2)
        self.layer4 = self._make_layer(Block, layers[3], base=512, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        feat_dim = 512 * Block.extend
        self.fc = nn.Sequential(
            nn.Linear(feat_dim, 50),
            nn.ReLU(inplace=True),
            nn.Dropout(dropout),
            nn.Linear(50, num_classes)
        )

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

    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
```

▲ 圖四、Resnet 架構

```python
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)
    )
```

▲ 圖五、DownSample

- **Resnet-18**

```python
def ResNet18(num_classes: int, in_ch: int = 3, dropout: float = 0.09, pretrained: bool = False):
    return ResNet(BasicBlock, [2, 2, 2, 2], num_classes=num_classes, num_channels=in_ch, dropout=dropout)
```

▲ 圖六、ResNet-18

除了剛剛提及的 BasicBlock 之外，主體分成四個 stage：[64, 128, 256, 512] 通道，各自堆 [2, 2, 2, 2] 個 BasicBlock。每個 stage 的第一個 block 會用 stride=2 做降採樣（stage1 例外，stride=1），同時若通道數或步幅改變，shortcut 分支用 1×1 的 DownSample 把 shape 對齊，之後的 block 全用 stride=1。尾端接 AdaptiveAvgPool2d(1×1) 做全域平均池化，flatten 後進入一個小的 FC head（Linear→ReLU→Dropout→Linear 到 num_classes）。因為 BasicBlock 的擴張係數 extend=1，最後特徵維度是 512×1。

- **Resnet-50**

```python
def ResNet50(num_classes: int, in_ch: int = 3, dropout: float = 0.09, pretrained: bool = False):
    return ResNet(Bottleneck, [3, 4, 6, 3], num_classes=num_classes, num_channels=in_ch, dropout=dropout)
```

▲ 圖七、ResNet-50

把 BasicBlock 改為 Bottleneck，分四個 stage：通道基底一樣是 [64, 128, 256, 512]，但每個 stage 的 block 數是 [3, 4, 6, 3]。每個 stage 的首個 bottleneck 以 stride=2 降採樣（stage1 依然 stride=1），並用 1×1 DownSample 對齊捷徑；其餘 bottleneck 保持 stride=1。單個 bottleneck 內部是 1×1→3×3→1×1：先壓縮通道、在較低通道做 3×3 特徵抽取、最後再用 1×1 擴回(圖三)； extend=4，因此最後的特徵維度是 512×4=2048。收尾同樣是 Global AvgPool → FC（Linear→ReLU→Dropout→Linear）。

B. The detail of your Dataloader

```
path,label
chest_xray/train/NORMAL/NORMAL2-IM-0626-0001.jpeg,0
chest_xray/train/NORMAL/NORMAL2-IM-1152-0001.jpeg,0
chest_xray/train/NORMAL/NORMAL2-IM-1276-0001.jpeg,0
chest_xray/train/NORMAL/NORMAL2-IM-0814-0001.jpeg,0
chest_xray/train/NORMAL/IM-0549-0001-0002.jpeg,0
chest_xray/train/NORMAL/IM-0501-0001-0001.jpeg,0
chest_xray/train/NORMAL/IM-0618-0001-0001.jpeg,0
chest_xray/train/NORMAL/NORMAL2-IM-0425-0001.jpeg,0
chest_xray/train/NORMAL/NORMAL2-IM-0634-0001.jpeg,0
chest_xray/train/NORMAL/NORMAL2-IM-0907-0001.jpeg,0
chest_xray/train/NORMAL/NORMAL2-IM-1260-0001.jpeg,0
```

▲ 圖八、csv data

```python
class ChestXRayCSVDataset(Dataset):
    def __init__(
        self,
        spec: CSVSpec,
        transform: Optional[torch.nn.Module] = None,
        equalize: bool = True,
        to_3ch: bool = True,
    ):
        self.data_root = Path(spec.data_root)
        csv_path = self.data_root / spec.csv_name
        if not csv_path.exists():
            raise FileNotFoundError(f"CSV not found: {csv_path}")

        df = pd.read_csv(csv_path)
        path_col = None
        for c in ["path", "filepath", "img"]:
            if c in df.columns:
                path_col = c
                break
        if path_col is None or "label" not in df.columns:
            raise ValueError(f"CSV must contain columns: (path|filepath|img), label — got {df.columns.tolist()}")

        self.paths = df[path_col].astype(str).str.strip().tolist()
        self.labels = df["label"].astype(int).tolist()
```

⮝ 圖八、get csv data

讀入對應的 CSV，將影像路徑與標籤標準化為兩個列表（paths、labels）

```python
def build_transforms(
    img_size: int = 224,
    train_aug: bool = True,
    imagenet_norm: bool = True,
    mild_rot_deg: int = 7,
) -> Tuple[torch.nn.Module, torch.nn.Module]:
    resize = KeepAspectSquareResize(img_size=img_size, fill=0)

    if imagenet_norm:
        normalize = T.Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225])
    else:
        normalize = T.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])

    if train_aug:
        train_tf = T.Compose([
            resize,
            T.RandomApply([T.GaussianBlur(kernel_size=3, sigma=(0.1, 0.6))], p=0.05),
            T.RandomAffine(
                degrees=mild_rot_deg, translate=(0.02, 0.02), scale=(0.98, 1.02),
                fill=0, interpolation=T.InterpolationMode.BILINEAR
            ),
            T.ColorJitter(brightness=0.04, contrast=0.04),
            T.ToTensor(),
            RandomGaussianNoise(p=0.10, std=0.005),
            normalize,
        ])
    else:
        train_tf = T.Compose([
            resize,
            T.ToTensor(),
            normalize,
        ])

    val_tf = T.Compose([
        resize,
        T.ToTensor(),
        normalize,
    ])

    return train_tf, val_tf
```

```python
def __len__(self) -> int:
    return len(self.paths)

def __getitem__(self, idx: int):
    rel = Path(self.paths[idx])
    img_path = rel if rel.is_absolute() else (self.data_root / rel)
    if not img_path.exists():
        raise FileNotFoundError(f"Image not found: {img_path}\n")

    img = Image.open(img_path).convert("L")

    box = img.getbbox()
    if box is not None:
        img = img.crop(box)

    if self.equalize:
        img = ImageOps.equalize(img)

    if self.to_3ch:
        img = To3Channels()(img)

    if self.transform is not None:
        img = self.transform(img)

    label = self.labels[idx]
    return img, label

def get_labels(self) -> List[int]:
    return self.labels
```

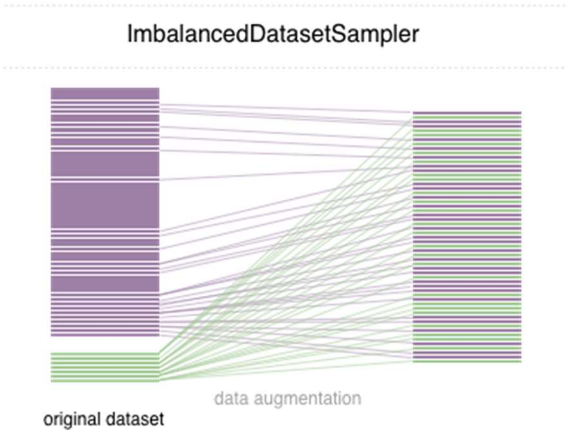
⮝ 圖九、data transform

```python
train_sampler = None
if sampler == "imbalanced":
    train_sampler = ImbalancedDatasetSampler(train_ds)
elif sampler == "weighted":
    labels = torch.tensor(train_ds.get_labels())
    class_counts = torch.bincount(labels)
    class_weights = 1.0 / class_counts.float().clamp_min(1)
    sample_weights = class_weights[labels]
    train_sampler = WeightedRandomSampler(
        weights=sample_weights, num_samples=len(sample_weights), replacement=True
    )
```

```python
class ImbalancedDatasetSampler(Sampler[int]):

    def __init__(
        self,
        dataset,
        indices: Optional[Sequence[int]] = None,
        num_samples: Optional[int] = None,
        replacement: bool = True,
        smoothing: float = 0.0,
    ):
        self.dataset = dataset
        self.indices = list(range(len(dataset))) if indices is None else list(indices)
        self.num_samples = len(self.indices) if num_samples is None else int(num_samples)
        self.replacement = bool(replacement)

        all_labels = _infer_all_labels(dataset)
        labels = [int(all_labels[i]) for i in self.indices]

        counts = Counter(labels)
        self.weights = torch.tensor(
            [1.0 / (counts[l] + float(smoothing)) for l in labels],
            dtype=torch.double,
        )

    def __iter__(self):
        chosen = torch.multinomial(self.weights, self.num_samples, replacement=self.replacement)
        return (self.indices[i] for i in chosen.tolist())

    def __len__(self) -> int:
        return self.num_samples
```

▲ 圖十、imbalanced

前處理包含裁除純黑邊（getbbox）、直方圖等化以提升對比，以及將灰階影像複寫為 3 通道（便於沿用 ImageNet 正規化與預訓練權重）；幾何與強度處理則使用自訂的 KeepAspectSquareResize 等比縮放並補邊到指定的正方形大小。訓練階段的 transform 採輕量高斯模糊、隨機仿射（小角度旋轉/平移/縮放）、亮度/對比抖動、轉為 Tensor、加入少量高斯雜訊並進行正規化；驗證/測試階段僅進行縮放、轉 Tensor 與正規化。為因應類別不平衡，訓練可選用 ImbalancedDatasetSampler（依類別頻率重採樣）或 WeightedRandomSampler（由 labels 計算類別權重加權抽樣），未指定時則隨機打亂。最終為 train/val/test 各建立一個 DataLoader，設定 batch_size、num_workers、pin_memory 等參數，以批次輸出 (image_tensor, label)；並提供 get_labels() 介面便於後續加權與採樣統計的計算。



為了緩解類別不平衡，我們在訓練階段進行重取樣，讓批次中的正常及肺炎

比例更接近均衡，避免模型偏向多數類別。ImbalancedDatasetSampler 會依各類別的出現頻率自動為樣本分配機率（少數類別給更高機率），以隨機但平衡的方式抽樣每個 batch；相比單純的 under-/over-sampling，它較不會丟失多樣性或因重複樣本導致過度擬合。

3. Experiment result and Discussion

A. Highest testing accuracy and F1-score

以下結果為訓練 50 epoch，batch-size 設為 64 、learning rate 設為、 optimizer 使用 AdamW(weight decay 為 1e-4)、dropout 為 0.09、loss 為 CrossEntropyLoss。

<table>
<tr><th>ResNet-18</th><th>ResNet-50</th><th>Densenet-121</th></tr>
<tr>
<td>precision=0.8868, recall=0.9846, f1=0.9332, acc=91.19%
precision recall f1-score support
0 - NORMAL 0.97 0.79 0.87 234
1 - PNEUMONIA 0.89 0.98 0.93 390
accuracy 0.91 624
macro avg 0.93 0.89 0.90 624
weighted avg 0.92 0.91 0.91 624</td>
<td>precision=0.8942, recall=0.9538, f1=0.9231, acc=90.06%
precision recall f1-score support
0 - NORMAL 0.91 0.81 0.86 234
1 - PNEUMONIA 0.89 0.95 0.92 390
accuracy 0.90 624
macro avg 0.90 0.88 0.89 624
weighted avg 0.90 0.90 0.90 624</td>
<td>precision=0.8871, recall=0.9667, f1=0.9252, acc=90.22%
precision recall f1-score support
0 - NORMAL 0.93 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 0.90 0.90 0.90 624</td>
</tr>
<tr><td>Acc = 91.19%, F1 = 0.93</td><td>Acc = 90.06%, F1 = 0.92</td><td>Acc = 90.22%, F1 = 0.92</td></tr>
</table>

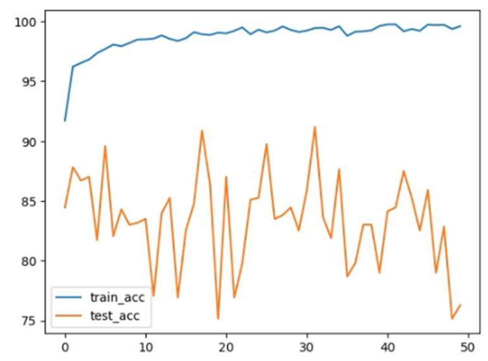
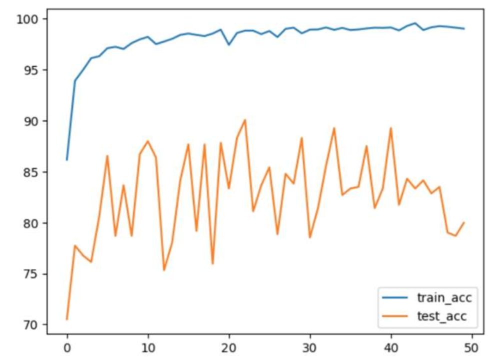
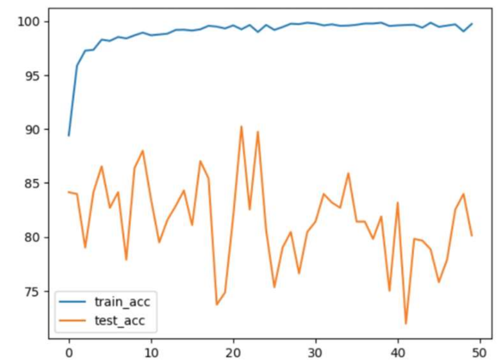
⮝ 表一、Testing accuracy & F1-score

根據表一，三個模型的整體表現接近，最佳為 **ResNet-18：91.19%**，**ResNet-50：90.06%**，**DenseNet-121：90.22%**。從分類報告可見共同的型態：對 **PNEUMONIA** 的 **recall** 很高，而 **NORMAL** 的 **recall** 較低，代表模型傾向於多抓到肺炎，但會多標一些正常胸腔為陽性。就效益而言，**ResNet-18** 在最輕量參數量的同時拿到最高準確率與 **F1（0.91）**。

進一步觀察二類的 precision/recall，可發現對 PNEUMONIA 而言，模型的 **recall 普遍高於 precision**；對 NORMAL 則相反。臨床上最不希望把肺炎判成正常，因此 **PNEUMONIA 的 recall** 相當重要，同時也需兼顧 precision 以避免過多誤報。在本次設定下，三個 model 在測試集對 PNEUMONIA 的表現皆**接近 90% 的 precision** 並有 約 **0.95–0.98 的 recall**，代表該抓的大多能抓到，但仍會有少量正常被誤判為肺炎的情形。

B. Ploting the comparison figures

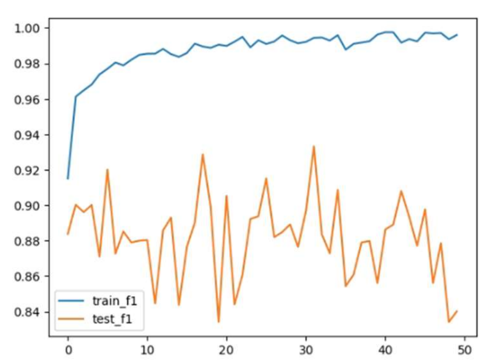
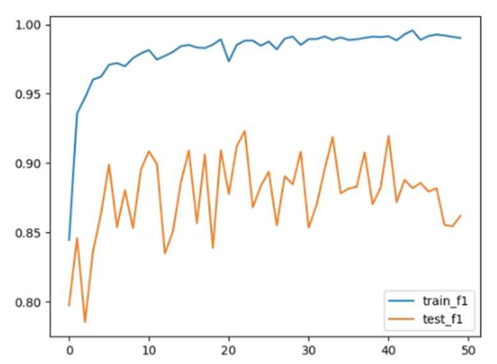
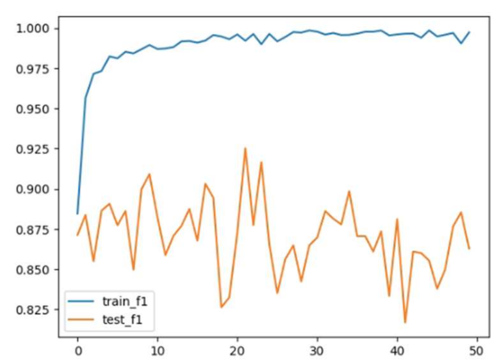
- **Accuracy figure**

ResNet-18	ResNet-50	Densenet-121
		

▲ 表二、Training & Testing accuracy

三個模型的 train_acc 在前 5–10 個 epoch 內即快速攀升至 99–100%，圖中可見訓練集的正確率穩定上升並逐步飽和；相對地，測試集的正確率出現較大幅度的震盪。其主為只對訓練集做了重取樣以平衡 NORMAL／PNEUMONIA 的比例；而測試集則維持真實分佈（肺炎樣本佔比超過一半），再加上評估集規模較小、每次權重更新都可能微調決策邊界，容易讓指標在各個 epoch 間波動。

- **F1-score figure**

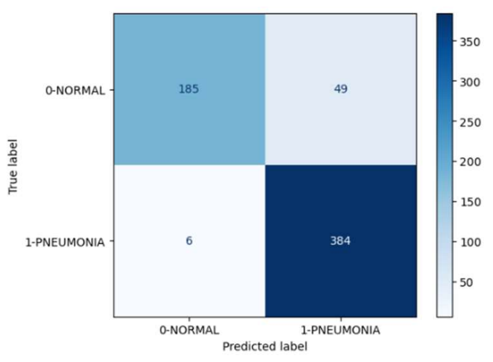
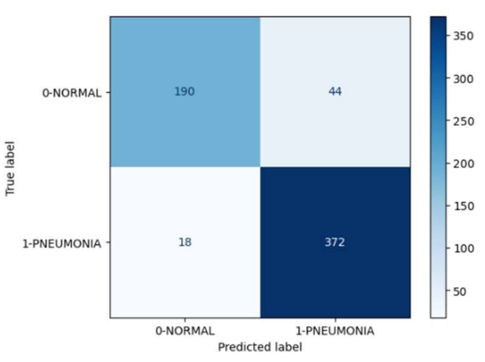
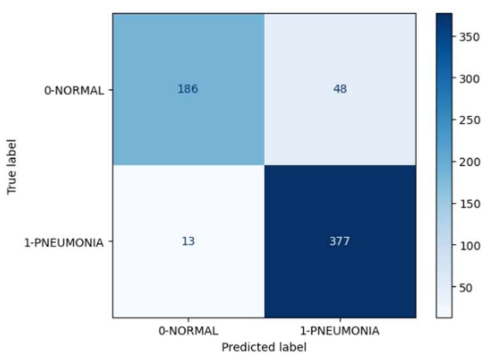
ResNet-18	ResNet-50	Densenet-121
		

▲ 表三、Training & Testing F1-score

三個模型的 train_f1 在前幾個 epoch 近 0.98–1.00 並飽和；而 test_f1 落在約 0.83–0.92 間明顯震盪(受測試集規模較小、且僅訓練集做了重取樣的影響)。ResNet-18 的 test_f1 相對較穩、DenseNet-121 變動最大、ResNet-50 介於兩者之間，最終三者的峰值彼此接近。

- **Confusion Matrix**

Confusion Matrix 以真實標籤及預測標籤構成（對二分類為 2×2 表），對角線為正確分類（TP、TN），非對角線為錯誤分類（FP、FN）。特別是在不平衡資料下，觀察右上角的 FP 與左下角的 FN 能直觀判斷模型是否把兩類互相混淆，以及錯誤偏向哪一邊；同時，混淆矩陣也是後續計算 precision、recall、F1-score 的基礎，便於評估模型在不同錯誤成本下的實際表現。

ResNet-18	ResNet-50	Densenet-121
		

▲ 表四、Testing Confusion Matrix

從表四來看，三者都呈現 FN 很少、FP 較多的型態，對肺炎敏感度高、但會把部分正常誤判為肺炎，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。ResNet-18 的取向較合適；以減少誤報來看的話，ResNet-50 略優。

C. Anything you want to present

- **高斯濾波**

```python
class RandomGaussianNoise(torch.nn.Module):
    """Additive Gaussian noise with small std, applied with given p."""
    def __init__(self, p: float = 0.25, std: float = 0.01):
        super().__init__()
        self.p = p
        self.std = std

    def forward(self, tensor: torch.Tensor) -> torch.Tensor:
        if random.random() < self.p:
            noise = torch.randn_like(tensor) * self.std
            return tensor + noise
        return tensor
```

▲ 圖十一、高斯 noise

本次 lab 加入兩種輕量級高斯處理以提升泛化，先 Gaussian Blur 以小機率套用 3×3 高斯濾波，模擬影像取得過程中的輕微模糊，減少對高頻噪點與邊緣細節的過度擬合，讓模型更具穩健性，接著加入 Gaussian Noise 以機率 p 於張量上加高斯白噪，使模型學到對隨機像素擾動的不變性。

- **Resize preprocessing**

```python
class KeepAspectSquareResize:
    """Resize to square (img_size x img_size) while keeping aspect ratio by padding."""
    def __init__(self, img_size: int, fill: int = 0):
        self.img_size = img_size
        self.fill = fill

    def __call__(self, img: Image.Image) -> Image.Image:
        w, h = img.size
        if w == 0 or h == 0:
            return img
        scale = self.img_size / max(w, h)
        new_w, new_h = int(round(w * scale)), int(round(h * scale))
        img = img.resize((new_w, new_h), resample=Image.BILINEAR)
        pad_left = (self.img_size - new_w) // 2
        pad_top = (self.img_size - new_h) // 2
        pad_right = self.img_size - new_w - pad_left
        pad_bottom = self.img_size - new_h - pad_top
        if any(p > 0 for p in (pad_left, pad_top, pad_right, pad_bottom)):
            img = ImageOps.expand(img, border=(pad_left, pad_top, pad_right, pad_bottom), fill=self.fill)
        return img
```

本次實作 KeepAspectSquareResize 以影像長邊為基準等比縮放到目標邊長，再對短邊左右/上下對稱補邊，將內容置中，補邊填值為 0(黑邊)，插值採 bilinear。透過這個方法可以同時滿足 CNN 需要固定大小的要求，又避免直接拉伸造成的幾何變形，不會改變肺部結構的相對比例；之後再接 ToTensor/Normalize 與輕量增強。

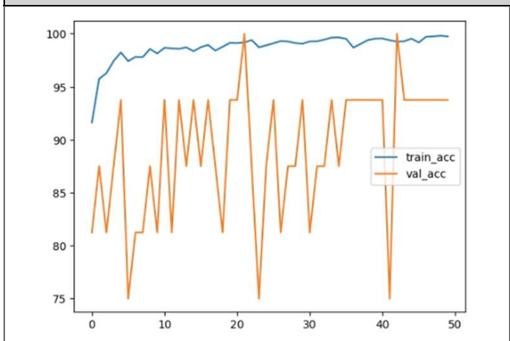
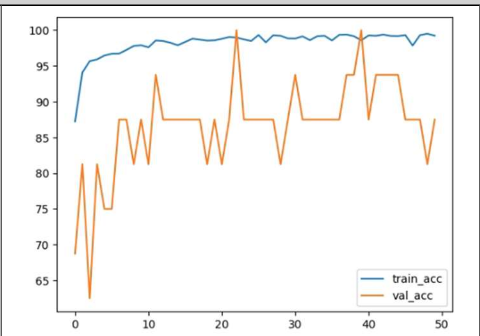
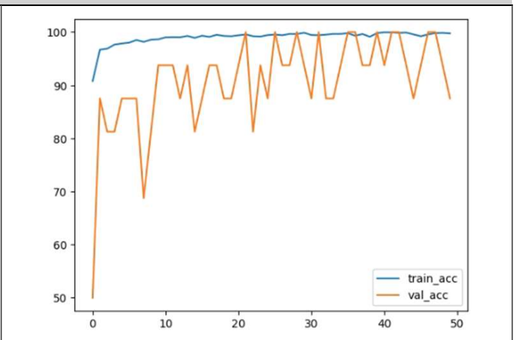
- **影像增強**

```python
class To3Channels:
    def __call__(self, img: Image.Image) -> Image.Image:
        if img.mode != "L":
            img = img.convert("L")
        return Image.merge("RGB", (img, img, img))
```

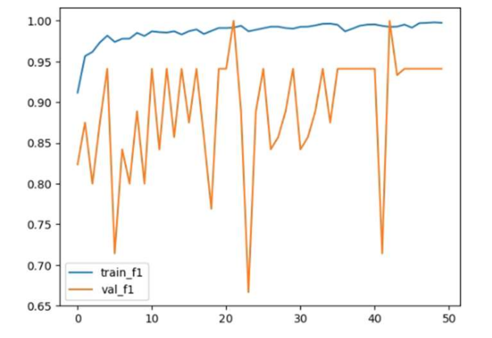
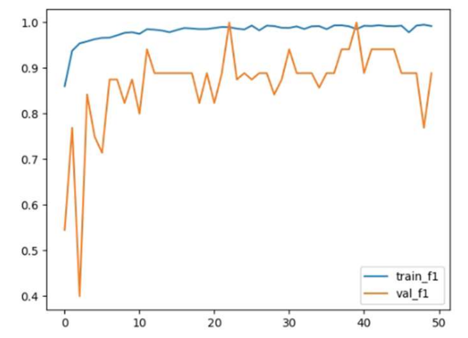
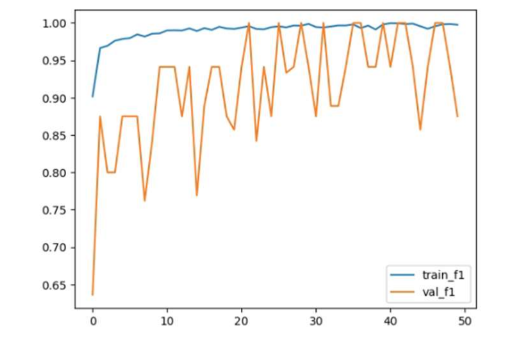
```python
T.ColorJitter(brightness=0.04, contrast=0.04),
```

本次實作也將影像先轉成 3 通道再套用 ColorJitter，亮度與對比會對所有通道等比例調整；在訓練階段隨機做約 ±4% 的微幅變動，用來模擬不同曝光造成的成像差異，提升模型對光照與對比變化的穩健性。

- **Validation**

ResNet-18	ResNet-50	Densenet-121
		

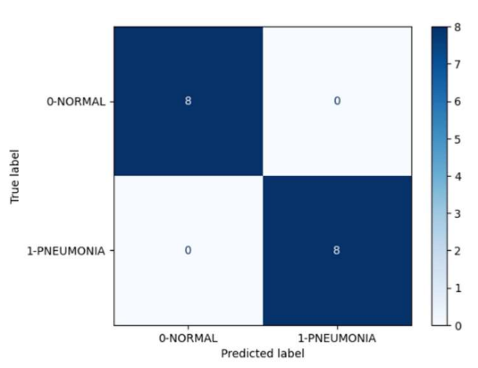
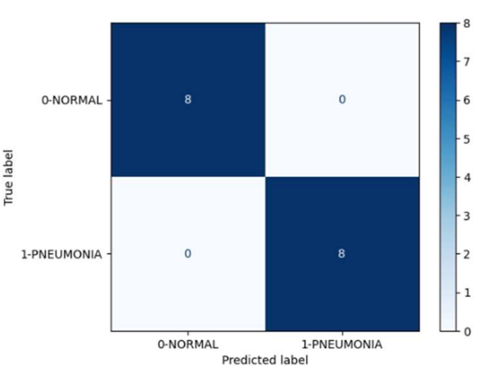
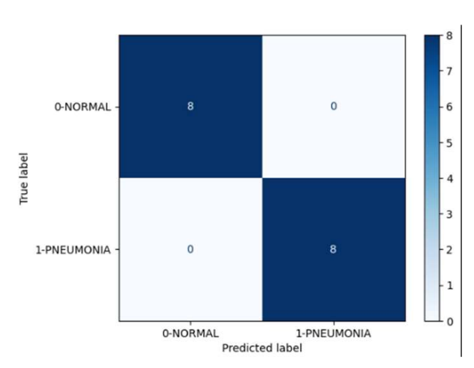
▲ 表五、Training & Validation Accuracy

ResNet-18	ResNet-50	Densenet-121
		

▲ 表六、Training & Validation F1-score

ResNet-18	ResNet-50	Densenet-121
precision=1.0000, recall=1.0000, f1=1.0000, acc=100.00% precision recall f1-score support 0 - NORMAL 1.00 1.00 1.00 8 1 - PNEUMONIA 1.00 1.00 1.00 8 accuracy 1.00 16 macro avg 1.00 1.00 1.00 16 weighted avg 1.00 1.00 1.00 16	precision=1.0000, recall=1.0000, f1=1.0000, acc=100.00% precision recall f1-score support 0 - NORMAL 1.00 1.00 1.00 8 1 - PNEUMONIA 1.00 1.00 1.00 8 accuracy 1.00 16 macro avg 1.00 1.00 1.00 16 weighted avg 1.00 1.00 1.00 16	precision=1.0000, recall=1.0000, f1=1.0000, acc=100.00% precision recall f1-score support 0 - NORMAL 1.00 1.00 1.00 8 1 - PNEUMONIA 1.00 1.00 1.00 8 accuracy 1.00 16 macro avg 1.00 1.00 1.00 16 weighted avg 1.00 1.00 1.00 16

▲ 表七、Training & Validation Overall

ResNet-18	ResNet-50	Densenet-121
		

▲ 表七、Validation Confusion Matrix

- **Densenet121**

```python
class DenseNet121(nn.Module):

    def __init__(self, num_classes: int = 2, in_ch: int = 3,
                 dropout: float = 0.0, pretrained: bool = False):
        super().__init__()
        weights = None
        if pretrained:
            try:
                weights = models.DenseNet121_Weights.DEFAULT
            except Exception:
                weights = None

        backbone = models.densenet121(weights=weights, drop_rate=0.0)
        if in_ch != 3:
            old = backbone.features.conv0
            backbone.features.conv0 = nn.Conv2d(
                in_ch, old.out_channels,
                kernel_size=old.kernel_size, stride=old.stride,
                padding=old.padding, bias=False
            )

        self.features = backbone.features
        self.num_features = backbone.classifier.in_features
        self.dropout_p = float(dropout)
        self.classifier = nn.Linear(self.num_features, num_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = F.relu(x, inplace=True)
        x = F.adaptive_avg_pool2d(x, (1, 1))
        x = torch.flatten(x, 1)
        if self.dropout_p > 0:
            x = F.dropout(x, p=self.dropout_p, training=self.training)
        x = self.classifier(x)
        return x
```

我以 torchvision.models.densenet121 建立 DenseNet-121：使用 pretrained=True；為了支援灰階或任意通道輸入，當 in_ch != 3 時，僅替換首層 features.conv0 為同幾何配置的新 Conv2d(in_ch, old.out_channels, kernel_size/stride/padding 與原層一致)，其餘層維持預訓練權重不變。接著保留 DenseNet 的特徵抽取部分 features，讀取原分類器的輸入維度 in_features，並用一個 Linear(in_features → num_classes) 取代原分類頭，同時提供可調 dropout。影像經 features → ReLU → AdaptiveAvgPool2d → flatten →dropout → 新分類器輸出。

4. Github Link

https://github.com/Ianuyu/AIMI/tree/main