a. How did you address the problems of imbalanced data and insufficient sample size? (5 pts)

第二部分沒有 imbalanced data 所以沒有操作

```
train_data_label_dist = pd.DataFrame(
    {
        "label" : train_data.iloc[:, 1].value_counts().sort_index().index.astype(str),
        "counts" : train_data.iloc[:, 1].value_counts().sort_index().values
    }
)
train_data_label_dist
```

[6]

	label	counts
0	0	40
1	1	40

b. Reproductivity of the results (the first part 2 pts and the second part 4 pts)

因為 gpu 的運算，所以不是每次都一樣，但是結果大致相同

c. Number of parameters: Please write the parameter count of your final selected model to the Kaggle competition (2 pts)

Parameters of Late fusion model

+ Code + Markdown

[19]:

```
model_late = Late_fusion(num_classes = 2)
total_params = sum(p.numel() for p in model_late.parameters())
print("總參數量:", total_params)
del model_late
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:19<00:00, 27.7MB/s]
總參數量: 60855106
```

60855106

d. The difficulty during training (the first part 3 pts and the second part 8 pts)

第二部分的模型每個都好大，我光是一個 epoch 就要花 10 分鐘，所以沒辦法用自己的電腦跑，搞 kaggle 反而比弄模型本身麻煩

e. Briefly explain the structures of the models you are using for the second part: You are required to do analyses of single slice, late fusion, early fusion, and 3D CNNs (16 pts)

1. Single Slice 模型

   結構：使用預訓練的 VGG16，但移除分類層。從單張切片中提取空間特徵。

   自訂分類器：全連接層，將提取的特徵對應到分類結果。使用 Softmax 輸出機率分布。

   損失函數：NLLLoss（模型輸出 log 機率）。

   核心概念：每張切片獨立處理，假設單張切片的資訊足夠進行分類。

```python
class Single_slice(nn.Module):
    def __init__(self, num_classes, input_size=(3, 50, 50), features_grad=False):
        super().__init__()
        # Initialize VGG16 as the feature extractor
        vgg16 = models.vgg16(weights='IMAGENET1K_V1', progress=True)
        vgg16.classifier = nn.Identity()
        # Freeze or unfreeze VGG16 feature layers based on `features_grad`
        for param in vgg16.features.parameters():
            param.requires_grad = features_grad
        self.backend = vgg16
        # Define a global average pooling layer to reduce feature map dimensions
        self.global_pool = nn.AdaptiveAvgPool2d(1)  # Outputs a 1×1 feature map per channel
        # Define a classifier for the extracted features
        self.classifier = nn.Linear(512, num_classes)  # 512 is the number of output channels from VGG16
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):  # x: list of 22 tensors, each of shape (batch_size, 3, 256, 256)
        softmax_outputs = []
        for i, slice_tensor in enumerate(x):
            # Extract feature map from VGG16
            feature_map = self.backend(slice_tensor)
            # Reshape the feature map back to [batch_size, 512, 7, 7] before pooling
            feature_map = feature_map.view(-1, 512, 7, 7)
            # Apply global average pooling to reduce feature map to shape (batch_size, 512)
            pooled_features = self.global_pool(feature_map).squeeze(-1).squeeze(-1)  # Removes the 1×1 dimensions
            # Ensure the shape is (batch_size, 512) before passing to classifier
            assert pooled_features.shape[1] == 512, f"Expected shape (batch_size, 512), got {pooled_features.shape}"
            # Pass feature map through classifier and apply softmax
            softmax_output = self.softmax(self.classifier(pooled_features))
            softmax_outputs.append(softmax_output)
        # Average the softmax outputs across all slices
        output = torch.stack(softmax_outputs).mean(dim=0)
        # Take the log of the averaged output for compatibility with NLLLoss
        log_output = torch.log(output + 1e-10)  # Adding epsilon to avoid log(0)
```

2. Late Fusion 模型

結構：使用 VGG16 分別處理每張切片。每張切片經 VGG16 提取特徵。把所有切片的特徵進行融合（例如取平均或串接）。

自訂分類器：全連接層將融合後的特徵對應到分類結果。使用 Softmax 輸出機率分布。損失函數：CrossEntropyLoss。

核心概念：切片獨立提取特徵，最後再結合所有切片的結果進行分類。

```python
class Late_fusion(nn.Module):
    def __init__(self, num_classes, input_size = (3, 256, 256), features_grad = False):
        super().__init__()
        # 取出vgg16中的特徵層
        vgg16 = models.vgg16(weights='IMAGENET1K_V1', progress = True)
        # 將 VGG16 的 classifier 層替換為 nn.Identity()，丟棄預設的分類層。
        vgg16.classifier = nn.Identity()
        # 可調整 feature map 的大小，可以改變參數量，但注意到會影響分類層
        # vgg16.avgpool = nn.AdaptiveAvgPool2d(output_size=(,))
        vgg16.avgpool = nn.AdaptiveAvgPool2d(output_size=(2, 2))  # Adjustable
        # 固定/不固定特徵層的參數值
        # 若 features_grad 設置為 FALSE，使用預訓練的 VGG16 特徵參數值，表示在訓練時參數將不會被更新
        # 若 features_grad 設置為 True，將利用預訓練的權重作為起點，然後讓模型根據本次的資料集進行調整。
        for param in vgg16.features.parameters():
            param.requires_grad = features_grad
        self.backend = vgg16

        # vgg16 最後一層通過 avgpool 後，每張 2D影像產生一個 (512, 7, 7) 的 feature map，共 22 個
        # 另外我們會對 22 張 feature map 進行拼接再攤平，得到一個長度為 512 * 22 * 7 * 7 的特徵向量
        # 增加分類層
        self.classifier = nn.Sequential(
            nn.Linear(512 * 22 * 2 * 2, 1024),  # Adjusted input dimension
            nn.ReLU(),
            nn.Dropout(0.5),  # Add Dropout
            nn.Linear(1024, num_classes)  # Final classification layer
        )
    # 在 PyTorch 中，forward 函數在 nn.Module 類的子類中是用來定義模型前向傳播過程的函數，這個函數描述了當 input 輸入模型時，如何一步一步地經過各層，最後輸出 outputs
    def forward(self, x): # x: 22 個 (batch_size, 3, 256, 256) 的 list
        feature_map_list = []
        for i in range(len(x)):
            feature_map_list.append(self.backend(x[i]))
        feature_map_concentrate = torch.cat(feature_map_list, dim=1)  # shape: (batch_size, 512 * 22, 7, 7)
        feature_map_concentrate = feature_map_concentrate.view(feature_map_concentrate.size(0), -1)  # shape: (batch_size, 512 * 22 * 7 * 7)
        outputs = self.classifier(feature_map_concentrate)
```

3. Early Fusion 模型

結構： 把切片堆疊在一起作為模型的輸入。修改 VGG16 的第一層卷

積層，接受多通道輸入。同時學習每張切片間的關聯和空間特徵。
自訂分類器：全連接層將提取的 3D 特徵對應到分類結果。使用 Softmax 輸出機率分布。
核心概念：所有切片從一開始就被視為一個整體，模型會學習它們的關聯性。

```python
class Early_fusion(nn.Module):
    def __init__(self, num_classes, input_size=(3, 256, 256), features_grad=False):
        super().__init__()
        # 加載預訓練的 VGG16 模型，並移除分類層
        vgg16 = models.vgg16(weights='IMAGENET1K_V1', progress=True)
        vgg16.classifier = nn.Identity()  # 去除 VGG16 的分類層，保留特徵提取部分
        # 鎖定/解鎖特徵層的參數
        for param in vgg16.features.parameters():
            param.requires_grad = features_grad
        self.backend = vgg16
        # 定義新的分類層
        self.classifier = nn.Sequential(
            nn.Linear(512, 256),  # VGG16 最後一層特徵為 512 維
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        # Convert list to tensor if necessary
        if isinstance(x, list):
            x = torch.stack(x)
        # Process each slice individually through VGG16
        features = [self.backend(slice).squeeze(0) for slice in x.view(-1, *x.shape[2:])]
        # features = [self.avgpool(f).view(-1, 512) for f in features]
        # Stack and reshape features back to [batch_size, 512]
        features = torch.stack(features).view(4, -1, 512).mean(dim=1)
        # features = torch.stack(features).view(22, -1, 512).mean(dim=1)
        # Pass through classifier
        log_output = self.classifier(features)

        return log_output
```

4. 3D CNN 模型

結構：
使用 3D 卷積層，同時學習空間和深度（切片間）特徵。包含：3D 卷積層、3D 池化層。最後的特徵壓平成全連接層進行分類。
自訂分類器：全連接層對應到分類結果。使用 Softmax 輸出機率分布。損失函數：NLLLoss。
核心概念：直接對完整 MRI 體積資料進行建模，能同時捕捉空間和深度資訊。

```python
class Resnet3d(nn.Module):
    def __init__(self, num_classes, features_grad = False):
        super().__init__()
        # 用上方定義 generate_model 函數生成一個預訓練的 3D ResNet50 模型。
        resnet50_3d = generate_model(model_type='resnet', model_depth=50,
                                     input_W=256, input_H=256, input_D=22, shortcut_type='B',
                                     pretrain_path = "/kaggle/input/resnet50/pretrain/resnet_50.pth",
                                     nb_class=2)
        resnet50_3d.classifier = nn.Identity()
        for param in resnet50_3d.parameters():
            param.requires_grad = features_grad
        # 更改成 3 channel
        temp = [param for param in resnet50_3d.conv1.parameters()][0] # (64, 1, 7, 7, 7)
        temp = torch.cat((temp, temp, temp), dim = 1) # (64, 3, 7, 7, 7)
        resnet50_3d.conv1 = nn.Conv3d(3, 64, kernel_size=(7, 7, 7), stride=(2, 2, 2), padding=(3, 3, 3), bias=False)
        for param in resnet50_3d.conv1.parameters():
            param = temp
        self.backend = resnet50_3d
        self.classifier = nn.Sequential(
            nn.Linear(2048, 512),
            nn.ReLU(),
            nn.BatchNorm1d(512),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )
    def forward(self, x): # x 是一個 (batchsize, 3, 22, 256, 256) 的 tensor
        if type(x) == list:
            x = torch.stack(x, dim=0)
        output = self.backend(x)
        outputs = self.classifier(output)
        return outputs
```

f. You should submit compiled HTML file and ipynb notebook with name