統計學習與深度學習 Mini Project 2

B11705051 陳奕廷

專案內容：

此份專案主要是針對 HW3 的影像分類問題的延伸，Image 資料集裡面，訓練資料集有 37866 張照片，而要被預測的資料有 8784 張。

和作業不同的地方是，我們觀察到主要的難關是照片可以是多標籤的，因此需要使用另外的 loss 計算方式以及 label 的方法，詳細後續會再補充。而標籤總共有 79 個，格式以下。有很多就不列舉了。

```
01A: 人物-男人
01B: 人物-女人
01C: 人物-小孩/嬰兒
01D: 人物-人物組合
01E: 人物-性別不明
```

資料前處理：

由於我認為圖片已經相對非常清晰好看了，因此我也沒有多做什麼 transform 的變化，採用比較少變化的方法以免模型訓練時資料複雜度太高。

```
# Basic data augmentation
transform = transforms.Compose([
    transforms.RandomResizedCrop(256, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225]),
])
```

這裡僅採用隨機裁剪、翻轉、旋轉等操作增加數據多樣性，然後把 image vector 正規化，好讓模型去做訓練。

接著是 label，只要訓練資料有的 label 會再 label vector 上面改成 1 其餘不是的都是 0。偏向 One Hot 作法，只不過他現在是多標籤，因此有可能擁有多個 1。

```
mlb = MultiLabelBinarizer()
train_labels = mlb.fit_transform(train_df['labels'])
num_classes = len(mlb.classes_)
```

在訓練資料製作完成的同時，我把它 split 成訓練以及驗證資料 validation，驗證資料總共有 1000 個。目的是要檢測 mAP 有沒有持續下降，若停滯那就 early stop，以免 epochs 過多導致的 overfit。

```python
from torch.utils.data import random_split

train_dataset = MultiLabelDataset(train_df, local_train_dir, labels=train_labels, transform=transform)
validation_size = 1000
train_size = len(train_dataset) - validation_size
train_dataset, validation_dataset = random_split(train_dataset, [train_size, validation_size])
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
validation_loader = DataLoader(validation_dataset, batch_size=batch_size, shuffle=False)

# 測試集保持不變
test_dataset = MultiLabelDataset(test_df, local_test_dir, labels=None, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

print(f"Training samples: {train_size}, Validation samples: {validation_size}")
```

```
cuda
Training samples: 36866, Validation samples: 1000
```

模型選擇：

這邊我使用的是 EfficientNet 家族的模型，EfficientNetB4。這邊會選擇使用 B4 的原因是，我事先跑過訓練更快的 B0，發現在 validation set 上，early stop 時，mAp 在 0.59 左右，上傳 Kaggle 也大概落在 0.52，因此我猜測使用更深更複雜一點的同家族模型，我的表現也會相當不錯。而訓練時第一個 epoch 也史無前例的出現 0.4 (先前 B0 開頭是 0.25)。最後修改分類層 classifier，使其適配多標籤分類的輸出數量 79 個。

```python
# Load EfficientNet-B0 model with pretrained weights
weights = EfficientNet_B4_Weights.IMAGENET1K_V1
model = efficientnet_b4(weights=weights)
```

一些幫助自己的方法：

由於訓練時間較長，也可能常常遇到失敗，網路不穩，Kernel Dead 等等的情況，因此我在每一個 epoch 都把好的結果的 model 儲存下來，也寫了一套讀取 model 的方法。

```python
# 設定模型文件的路徑
model_path = '/kaggle/input/model-for-sldl/pytorch/default/1/best_model.pth'  # 根據實際文件名稱設置

# 檢查該路徑是否存在
if os.path.exists(model_path):
    print(f"從 {model_path} 載入最佳模型")

    # 載入模型檔案
    checkpoint = torch.load(model_path)

    # 載入模型和優化器的 state_dict
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    epoch = checkpoint['epoch']
    best_mAP = checkpoint['mAP']  # 恢復最佳 mAP

    print(f"從 epoch {epoch+1} 繼續訓練，最佳 mAP: {best_mAP:.4f}")
else:
    print(f"沒有找到模型檔案，從頭開始訓練。")
    epoch = 0  # 從第一個 epoch 開始
    best_mAP = 0.0  # 尚未有最佳模型
```

```python
# Early stopping logic
if mAP > best_mAP:
    best_mAP = mAP
    early_stop_counter = 0
    # Save the best model
    best_model_path = f'/kaggle/working/best_model.pth'
    torch.save({
        'epoch': epoch + 1,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': avg_loss,
        'mAP': mAP,
    }, best_model_path)
    print(f"New best model saved to {best_model_path}")
else:
    early_stop_counter += 1
    print(f"No improvement in mAP. Early stop counter: {early_stop_counter}/{patience}")

# Check if early stopping condition is met
if early_stop_counter >= patience:
    print(f"Early stopping triggered. Best mAP: {best_mAP:.4f}")
    break
```

超參數的選擇：

```python
# Focal loss for imbalanced datasets
class FocalLoss(nn.Module):
    def __init__(self, alpha=1, gamma=2, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.reduction = reduction

    def forward(self, inputs, targets):
        BCE_loss = nn.BCEWithLogitsLoss(reduction='none')(inputs, targets)
        pt = torch.exp(-BCE_loss)
        focal_loss = self.alpha * (1 - pt) ** self.gamma * BCE_loss
        if self.reduction == 'mean':
            return focal_loss.mean()
        elif self.reduction == 'sum':
            return focal_loss.sum()
        else:
            return focal_loss

criterion = FocalLoss()
optimizer = optim.AdamW(model.parameters(), lr=learning_rate, weight_decay=1e-4)
scheduler = optim.lr_scheduler.OneCycleLR(optimizer, max_lr=learning_rate, epochs=num_epochs, steps_per_epoch=len(train_loader))
```

這裡使用 Binary Cross-Entropy（BCE），是因為它很適合多標籤分類的評估，並採用 Focal Loss 是 BCE 的變體，設計用來解決類別不平衡問題，特別是在檢測或分類問題中，負樣本往往遠多於正樣本。

Optimizer 使用 AdamW，是改進版的 Adam 優化器，它能更好地處理權重衰減（weight decay），在深度學習模型中常用於穩定訓練並加快收斂。

並採用動態調整的學習率，使用已經生成的套件幫助模型挑選合適的學習率，並在每一個 epoch 後調整。

```python
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

觀察現象：

這裡我發現，有切出 validation set 並使用 early stop 非常重要，我一開始認為直接訓練並輸出結果應該就有好分數了吧，但可以發現 loss 非常快就變很小，那對於使用者來說，loss 的變動這麼低，模型收斂了嗎，或者 overfit 了嗎。所以採用 validation & early stop 後，我發現丟上 Kaggle 的分數也好看很多，我同時也知道該在什麼時候停，即使每一個 epoch 的 loss 都非常非常小。下圖是讀取已經訓練三個 epoch 的模型繼續訓練，因為中間斷線。

```
Epoch 1: 100%|██████████| 1153/1153 [20:10<00:00,  1.05s/it, Batch Loss=0.0269]
Epoch [1/10], Loss: 0.0132
Epoch [1/10], mAP: 0.6133
New best model saved to /kaggle/working/best_model.pth
Epoch 2: 100%|██████████| 1153/1153 [16:52<00:00,  1.14it/s, Batch Loss=0.0229]
Epoch [2/10], Loss: 0.0114
Epoch [2/10], mAP: 0.6176
New best model saved to /kaggle/working/best_model.pth
Epoch 3: 100%|██████████| 1153/1153 [16:51<00:00,  1.14it/s, Batch Loss=0.0192]
Epoch [3/10], Loss: 0.0099
Epoch [3/10], mAP: 0.6590
New best model saved to /kaggle/working/best_model.pth
Epoch 4: 100%|██████████| 1153/1153 [16:52<00:00,  1.14it/s, Batch Loss=0.016]
Epoch [4/10], Loss: 0.0087
Epoch [4/10], mAP: 0.6743
New best model saved to /kaggle/working/best_model.pth
Epoch 5: 100%|██████████| 1153/1153 [16:49<00:00,  1.14it/s, Batch Loss=0.0439]
Epoch [5/10], Loss: 0.0077
Epoch [5/10], mAP: 0.6777
New best model saved to /kaggle/working/best_model.pth
Epoch 6: 100%|██████████| 1153/1153 [16:56<00:00,  1.13it/s, Batch Loss=0.016]
Epoch [6/10], Loss: 0.0069
Epoch [6/10], mAP: 0.7000
New best model saved to /kaggle/working/best_model.pth
Epoch 7: 100%|██████████| 1153/1153 [17:02<00:00,  1.13it/s, Batch Loss=0.00849]
Epoch [7/10], Loss: 0.0062
Epoch [7/10], mAP: 0.7028
New best model saved to /kaggle/working/best_model.pth
Epoch 8: 100%|██████████| 1153/1153 [16:47<00:00,  1.14it/s, Batch Loss=0.0167]
Epoch [8/10], Loss: 0.0055
Epoch [8/10], mAP: 0.6920
No improvement in mAP. Early stop counter: 1/1
Early stopping triggered. Best mAP: 0.7028
```

成效：

從上圖可以看到，自己的 validation set 的結果是 0.7028，而我丟上 Kaggle 後的分數是 0.65-0.66。差距不會說非常的巨大，是有估計價值的。

程式碼的使用：

在 ipynb 最後下面，有讀取 best model 相關的程式碼，除此之外從上往下執行程式碼即可讀取資料並訓練，最後生成預測。