HW1 Object Detection

> 111360205 謝進權

程式碼說明

- 讓 colab 與 google drive 進行 mount，存取雲端硬碟中的檔案。

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

- 匯入必要的套件＆設定資料集的路徑

```
import os
import torch
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.transforms import functional as F
from torch.utils.data import Dataset, DataLoader, random_split
from PIL import Image
import pandas as pd
# from tqdm import tqdm

# ------------------------------
# 1. 資料集路徑
# ------------------------------
TRAIN_IMG_DIR = '/content/drive/MyDrive/taica-cvpdl-2025-hw-1/train/img'
TRAIN_GT_FILE = '/content/drive/MyDrive/taica-cvpdl-2025-hw-1/train/gt.txt'
TEST_IMG_DIR  = '/content/drive/MyDrive/taica-cvpdl-2025-hw-1/test/img'

print("✅ Dataset paths loaded")
```

```
✅ Dataset paths loaded
```

- Dataset 實作。將圖片中的 gt.txt 標註檔案，轉換成訓練用的 Tensor 格式

```python
# ------------------------------
# 2. Dataset
# ------------------------------
import cv2
import albumentations as A
from albumentations.pytorch import ToTensorV2

def get_train_transforms():
    return A.Compose([
        A.HorizontalFlip(p=0.5), # 50% 機率水平翻轉
        A.RandomBrightnessContrast(p=0.2), # 隨機調整亮度對比
        A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), # 標準化
        ToTensorV2(), # 轉換成 Tensor
    ], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['labels'])) # 關鍵! 告訴 Albumentations 如何處理 box

class PigDataset(Dataset):
    def __init__(self, img_dir, gt_file=None, transforms=None):
        self.img_dir = img_dir
        self.transforms = transforms
        self.imgs = sorted(os.listdir(img_dir))

        self.boxes = {}
        if gt_file:
            with open(gt_file) as f:
                for line in f:
                    line = line.strip().split(',')
                    if len(line) < 5:
                        continue
                    img_id = line[0]
                    x, y, w, h = map(float, line[1:5])
                    if w <= 0 or h <= 0:
                        continue
                    if img_id not in self.boxes:
                        self.boxes[img_id] = []
                    self.boxes[img_id].append([x, y, x + w, y + h])

        # 排除掉沒有任何標註框的圖片
        all_imgs = sorted(os.listdir(img_dir))
        self.imgs = []
        for img_name in all_imgs:
            img_id = str(int(os.path.splitext(img_name)[0]))
            if img_id in self.boxes and self.boxes[img_id]:
                self.imgs.append(img_name)

    def __len__(self):
        return len(self.imgs)

    def __getitem__(self, idx):
        img_name = self.imgs[idx]
        img_path = os.path.join(self.img_dir, img_name)

        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        img_id = str(int(os.path.splitext(img_name)[0]))
        boxes = self.boxes.get(img_id)
        labels = [1] * len(boxes)

        if self.transforms:
            # 直接將 python list 傳入，不要先轉 tensor
            transformed = self.transforms(image=img, bboxes=boxes, labels=labels)

            img = transformed['image']
            transformed_boxes = transformed['bboxes']

            # 如果增強後所有的豬都被裁掉了，我們需要處理這種情況
            # 為了簡單起見，可以選擇遞迴呼叫來取下一張圖片
            if not transformed_boxes:
                return self.__getitem__((idx + 1) % len(self.imgs))

            target = {
                "boxes": torch.tensor(transformed_boxes, dtype=torch.float32),
                "labels": torch.ones(len(transformed_boxes), dtype=torch.int64) # label 永遠是 1
            }
        else:
            # 如果沒有 transform，才需要手動處理
            target = {
                "boxes": torch.tensor(boxes, dtype=torch.float32),
                "labels": torch.ones(len(boxes), dtype=torch.int64)
            }
            # 需要從 torchvision.transforms import functional as F
            # img = F.to_tensor(img)

        # 修正 return 的變數
        return img, target, img_name

```

- 將 Dataset 物件包裝成 DataLoader

```python
# ------------------------------
# 3. DataLoader + Validation Split
# ------------------------------

# 1. 先定義好給驗證集用的 transform
def get_val_transforms():
    return A.Compose([
        A.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ToTensorV2(),
    # 注意：即使驗證集也要傳 bbox_params，因為 Albumentations 的 pipeline 要求
    ], bbox_params=A.BboxParams(format='pascal_voc', label_fields=['labels']))

# 2. 建立一個不帶任何 transform 的初始 dataset
full_dataset = PigDataset(TRAIN_IMG_DIR, TRAIN_GT_FILE, transforms=None)

# 3. 按照原樣切分
n_total = len(full_dataset)
n_val = int(0.2 * n_total)
n_train = n_total - n_val
train_dataset, val_dataset = random_split(full_dataset, [n_train, n_val])

# 4. 關鍵步驟：為切分後的兩個子集分別賦予不同的 transform
train_dataset.dataset.transforms = get_train_transforms()
val_dataset.dataset.transforms = get_val_transforms()

# 5. DataLoader 維持不變
def collate_fn(batch):
    # 雖然 Dataset 內部已經處理了 None，但這裡的遞迴可能會產生 None，所以保留過濾是好的
    batch = list(filter(lambda x: x is not None and x[0] is not None, batch))
    if not batch: return (torch.empty(0), torch.empty(0)) # 處理整個 batch 都被過濾掉的極端情況
    return tuple(zip(*batch))

train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True, collate_fn=collate_fn, num_workers=2, pin_memory=True) # pin_memory=True讓資料從CPU到GPU可以更快
val_loader   = DataLoader(val_dataset, batch_size=4, shuffle=False, collate_fn=collate_fn, num_workers=2, pin_memory=True)

print(f"Train size: {len(train_dataset)} | Val size: {len(val_dataset)}")
```

```
Train size: 1013 | Val size: 253
/usr/local/lib/python3.12/dist-packages/albumentations/core/composition.py:331: UserWarning: Got processor for bboxes, but no transform to process it.
  self._set_keys()
```

- 模型設定

```python
# --------------------------------
# 4. Faster R-CNN + pretrained weights
# --------------------------------
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

from torchvision.models.detection import fasterrcnn_resnet50_fpn_v2
model = fasterrcnn_resnet50_fpn_v2(weights="COCO_V1")

num_classes = 2
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
model.to(device)

```

- 參數設定

```python
# --------------------------------
# 5. 訓練設定
# --------------------------------
from torch.optim.lr_scheduler import CosineAnnealingLR

params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.AdamW(params, lr=1e-4, weight_decay=1e-4)
num_epochs = 10

scheduler = CosineAnnealingLR(optimizer, T_max=num_epochs, eta_min=1e-6)

best_val_loss = float('inf') # 追蹤歷史最低的驗證損失
MODEL_SAVE_PATH = '/content/drive/MyDrive/Colab Notebooks/5-電腦視覺與深度學習/best_model.pth' # 設定儲存檔案的路徑和名稱
```

- （訓練＋驗證＋模型儲存）loop

```python
# ------------------------------
# 6. 訓練 + 驗證 Loop
# ------------------------------
from tqdm import tqdm  # 匯入 tqdm，用於產生進度條
from torchmetrics.detection.mean_ap import MeanAveragePrecision

for epoch in range(num_epochs):
    print(f"\nEpoch {epoch+1}/{num_epochs}")
    model.train()
    train_loss_tracker = {'total_loss': 0.0, 'loss_classifier': 0.0, 'loss_box_reg': 0.0, 'loss_objectness': 0.0, 'loss_rpn_box_reg': 0.0}
    running_loss = 0.0

    if epoch == 9:
        optimizer = torch.optim.Adam(model.parameters(), lr=5e-5)
        print("Learning Rate scaled down")

    # 訓練迴圈加入進度條
    for imgs, targets, _ in tqdm(train_loader, desc=f"Training"):
        imgs = [img.to(device) for img in imgs]
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        loss_dict = model(imgs, targets)
        losses = sum(loss for loss in loss_dict.values())

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

        # 記錄詳細損失
        train_loss_tracker['total_loss'] += losses.item()
        for k, v in loss_dict.items():
            train_loss_tracker[k] += v.item()

    # 在每個 epoch 訓練結束後，更新學習率
    scheduler.step()


    avg_train_loss = running_loss / len(train_loader)
    print(f"Train Loss: {avg_train_loss:.4f}")

    # ------------------------------
    # Validation
    # ------------------------------
    # 初始化 mAP 計算器
    metric = MeanAveragePrecision(box_format='xyxy').to(device)

    # 驗證時需要切換到 eval 模式來獲取預測結果
    model.eval()
    with torch.no_grad():
        for imgs, targets, _ in tqdm(val_loader, desc=f"Validating"):
            imgs = [img.to(device) for img in imgs]
            targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

            predictions = model(imgs) # 在 eval 模式下，模型輸出預測結果
            metric.update(predictions, targets) # 更新 mAP 計算器

    # 計算並打印 mAP
    results = metric.compute()
    val_map = results['map'].item()
    print(f"Validation mAP: {val_map:.4f}")

    # ------------------------------
    # 模型儲存邏輯 (改用 mAP 作為標準)
    # ------------------------------
    if 'best_map' not in locals(): best_map = -1.0

    if val_map > best_map:
        print(f"Validation mAP Improved ({best_map:.4f} -> {val_map:.4f}). Saving model...")
        best_map = val_map
        torch.save(model.state_dict(), MODEL_SAVE_PATH) # 推論時通常只需要模型權重
    else:
        print("Validation mAP did not improve.")
```

- 得出的結果：若測試集的 mAP 值更高，就儲存模型。

```
Epoch 9/10
Training: 100%|██████████| 254/254 [07:05<00:00,  1.68s/it]
Train Loss: 0.0000
Validating: 100%|██████████| 64/64 [00:46<00:00,  1.38it/s]
Validation mAP: 0.8482
Validation mAP Improved (0.8465 -> 0.8482). Saving model...

Epoch 10/10
Learning Rate scaled down
Training: 100%|██████████| 254/254 [07:05<00:00,  1.67s/it]
Train Loss: 0.0000
Validating: 100%|██████████| 64/64 [00:46<00:00,  1.38it/s]
Validation mAP: 0.8351
Validation mAP did not improve.
```

- 套用訓練好的模型，並且進行預測，匯出 submission.csv 檔案

```python
# ------------------------------
# 7. 預測 & submission
# ------------------------------
model.eval()
test_imgs = sorted(os.listdir(TEST_IMG_DIR))
predictions = []

with torch.no_grad():
    # for img_name in tqdm(test_imgs, desc="Testing"):
    for img_name in test_imgs:
        img_path = os.path.join(TEST_IMG_DIR, img_name)
        img = Image.open(img_path).convert("RGB")
        img_tensor = F.to_tensor(img).to(device)
        pred = model([img_tensor])[0]

        img_id = int(os.path.splitext(img_name)[0])
        parts = []
        for score, box in zip(pred['scores'], pred['boxes']):
            if score < 0.3:
                continue
            x_min, y_min, x_max, y_max = box.tolist()
            w, h = x_max - x_min, y_max - y_min
            parts.append(f"{score:.6f} {x_min:.2f} {y_min:.2f} {w:.2f} {h:.2f} 0")

        pred_str = " ".join(parts)
        predictions.append([img_id, pred_str])

submission = pd.DataFrame(predictions, columns=['Image_ID', 'PredictionString'])
submission.to_csv('submission.csv', index=False)
print("✅ Submission saved: submission.csv")
```

```
✅ Submission saved: submission.csv
```

問題與解決方式

1. 資料增強不同步

> Data Augmentation

- **問題說明**：一開始程式碼只有對圖片進行資料增強，但是沒有對應調整Bounding Box的座標，造成模型學到錯誤的「圖片-標籤」組合，讓模型無法有效學習。
- **解決方式**：引入 `Albumentations` 函式庫，讓圖片與 Bounding Box 可以進行一致的幾何變換。

2. 固定的學習率導致收斂效果不佳

> Fixed Learning Rate

- **問題說明**：最初只有設定一個固定的學習率。在訓練後期，會因為較大的學習率在最佳解附近震盪，較難收斂。
- **解決方式**：引入學習率排程器 `CosineAnnealingLR` ，讓學習率可以從初始值動態下降。並且將 `Optimizer` 更換為 `AdamW` ，防止 overfit ，提升模型表現。

Kaggle 分數截圖

Private Score