HW2 - Long-Tailed Object Detection 報告

1. Model Description

本專案採用 Ultralytics YOLOv11l 作為目標偵測模型，以 yolo11l.yaml 從零初始化訓練（非載入預訓練權重），並在推論階段使用訓練得到的 best.pt 完成測試集偵測與 Kaggle 風格提交檔生成。輸入解析度設定為 960×960。

Backbone：C2f/Conv 疊層 + SPPF（多尺度池化）
Neck：PAN-FPN 自頂向下 / 自底向上融合，強化多尺度特徵
Head：解耦頭（分類 / 回歸分支）+ DFL（Distribution Focal Loss）以提升邊界框回歸精度
Post-process：NMS（非極大值抑制）

2. Implementation Details

2.1 資料與前處理

程式首先把原始資料（data/train/img/, data/train/gt.txt, data/test/img/）轉為 YOLO 可讀格式（data_yolo/）：

gt.txt 格式：frame_id, l, t, w, h（像素座標）。

影像匹配：

透過 smart_find_image_by_frame_id(...) 以 零填充 / 檔名數字抽取 等多策略匹配 frame_id → 圖檔，極大降低命名不一致導致的漏配。

座標轉換：

(l,t,w,h) → YOLO 正規化格式 (cx/W, cy/H, w/W, h/H)，並在邊界外做裁剪保護；MAKE_EMPTY_LABEL_FOR_NOBOX=True 可為無框圖也建立空標籤檔（利於訓練穩定）。

資料切分：

VAL_RATIO=0.10，RANDOM_SEED=42，隨機切分 train/val；並將 test 影像複製至 images/test/。

處理 long tail 分佈問題：

生成 train_upsampled.txt 列出訓練圖片列表，讓稀有類別圖片重複出現，生成 dataset_upsampled.yaml 仅將 train 指向上面的 txt，其他不变。

產出：

data_yolo/images/{train,val,test}/、data_yolo/labels/{train,val}/
data_yolo/dataset.yaml（路徑與類別名 names: {0: "object"}）
data_yolo/train_upsampled.txt
dataset_upsampled.yaml

前處理：Ultralytics 於 Dataset pipeline 內自動完成 Letterbox 到 640×640、像素正規化與通道處理（BGR→RGB）。

2.2 訓練設定

```
单击以添加断点 lytics import YOLO

# Load a model: 从 YAML 初始化（随机权重）
model = YOLO("yolo11l.yaml")

# Train the model with custom parameters
results = model.train(
    data="data_yolo/dataset_upsampled.yaml",
    epochs=200,          # 你可根据时间调
    imgsz=980,           # 显存吃紧可降到 640/768
    batch=1,             # 12GB 建议 8~16
    device=0,            # 或"cpu"
    pretrained=False,    # 禁用预训练
    optimizer="SGD",
    lr0=0.01,
    lrf=0.1,
    momentum=0.937,
    weight_decay=5e-4,
    warmup_epochs=3.0,
    cos_lr=True,
    project="runs/train",
    name="hw2",
)
```

主要超參數設定

2.3 驗證

驗證集

```
from ultralytics import YOLO
from pathlib import Path

ckpt = Path("/content/runs/train/hw210/weights/best.pt") # Point to the best.pt file
assert ckpt.exists(), f"未找到模型权重：{ckpt}"

model = YOLO(str(ckpt))  # load a custom model（刚训练出来的）
metrics = model.val(data="data_yolo/dataset_upsampled.yaml", imgsz=1920)
metrics  # mAP 等指标
```

Ultralytics 會在 runs/detect/val* 產生相應輸出（曲線圖、混淆矩陣等）

2.4 提交檔(滿足 Kaggle 格式要求)生成

```python
# ======= 可调参数 =======
MAIN_CONF = 0.01      # 想要的主阈值
PRED_CONF = 0.001     # 推理时使用的低阈值，用于抓全候选，后续再按 MAIN_CONF 过滤
IOU_THRES = 0.7
IMG_SIZE  = 960
DEVICE    = 0         # 或"cpu"

# 载入刚训练的权重
ckpt = Path("/content/runs/train/hw210/weights/best.pt")
assert ckpt.exists(), f"未找到模型权重：{ckpt}"
model = YOLO(str(ckpt))

# 收集 test 文件并确定 Image_ID（按文件名排序→1-based）
IMAGES_TEST = Path("data_yolo/images/test")
img_exts = {'.jpg','.jpeg','.png','.bmp','.tif','.tiff','.webp'}
test_files = sorted([p for p in IMAGES_TEST.iterdir() if p.suffix.lower() in img_exts], key=lambda p: p.name)
assert len(test_files) > 0, f"测试目录为空：{IMAGES_TEST}"
id_map = {p.name: i+1 for i, p in enumerate(test_files)}  # 1-based
```

```python
# 用较低 conf 推理，确保能拿到“top1 兜底候选”
pred_list = list(model.predict(
    source=str(IMAGES_TEST),
    imgsz=IMG_SIZE,
    conf=PRED_CONF,     # 低阈值，保留尽可能多候选
    iou=IOU_THRES,
    device=DEVICE,
    stream=False,
    save=True,
    verbose=False
))
```

由於 test 中存在一些圖片只有一個 object，如果 conf 設定太高則可能產生 NULL VALUE.設定太低則可能導致抓取錯誤 object.所以採用先用低 conf 抓取多物體，再用高 conf 過濾，同時設置條件保證每張圖片至少有抓取到一個框。

以最終訓練得到的 best.pt 在指定測試集上推論，並將 results 轉為 Image_ID, PredictionString：

PredictionString 以 置信度降序串接，單框格式：conf l t w h cls（其中座標以像素為單位，且對邊界做裁剪保護）。

完整 CSV 寫至 SUB_CSV_PATH。

可視化圖片輸出在 runs/detect/predict/，便於抽樣質檢。

3. Result Analysis

results.csv

1 to 10 of 10 entries Filter

epoch	time	train/box_loss	train/cls_loss	train/dfl_loss	metrics/precision(B)	metrics/recall(B)	metrics/mAP50(B)	metrics/mAP50-95(B)	val/box_loss	val/cls_loss	val/dfl_loss	lr/pg0	lr/pg1	lr/pg2
1	245.797	5.05242	11.5523	4.54751	0.00099	0.0048	0.0005	0.00015	4.67764	37.3231	5.84143	0.0700351	0.00332943	0.00332943
2	476.925	4.9374	8.52457	4.26319	0.00115	0.00747	0.00067	0.00016	4.69726	7.35216	4.30545	0.0400347	0.0066624	0.0066624
3	708.118	4.29644	6.79035	3.63343	0.00516	0.05269	0.00301	0.00078	3.99752	128.359	3.3834	0.0100329	0.00999388	0.00999388
4	936.643	3.48832	4.86014	2.84099	0.02236	0.06187	0.01327	0.00313	3.8321	20.8617	3.03119	0.009995	0.009995	0.009995
5	1165.15	3.14006	3.69589	2.50383	0.0216	0.14222	0.01751	0.00587	2.9907	6.61946	2.43742	0.00999112	0.00999112	0.00999112
6	1393.38	2.91492	3.14268	2.24349	0.84542	0.08302	0.088	0.02713	2.85407	2.47281	2.17725	0.00998613	0.00998613	0.00998613
7	1621.43	2.78163	2.78465	2.16932	0.44732	0.14676	0.15272	0.04852	2.66867	2.06784	2.04078	0.00998003	0.00998003	0.00998003
8	1849.71	2.65701	2.44834	2.03116	0.50834	0.17167	0.17584	0.05692	2.57131	2.06658	1.95517	0.00997282	0.00997282	0.00997282
9	2077.7	2.56802	2.39567	1.98452	0.53617	0.17655	0.17948	0.06084	2.50773	1.95152	1.90172	0.00996452	0.00996452	0.00996452
10	2305.54	2.50737	2.2238	1.95235	0.61261	0.20529	0.21473	0.07866	2.38227	1.83396	1.8227	0.00995511	0.00995511	0.00995511

Show 10 per page

分析訓練日誌（前 10 epoch）可得：

訓練損失：train/box_loss 5.05→2.5 、train/cls_loss 11.5→2.2、train/dfl_loss 4.54→1.95，單調下降，收斂正常。

驗證損失：val/box_loss 4.67→2.38 val/cls_loss 37.3→1.83 val/dfl_loss 5.84→1.83

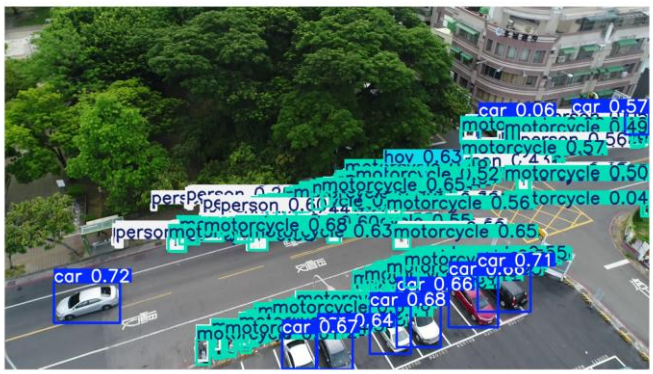
精度/召回：Precision 約 0→0.0.61 穩定上升，Recall 0.005→0.21 持續提升。

mAP：mAP@50 0→0.21 穩定上升；mAP@50:95 0→0.078

由前十個 epoch 可看出訓練效果正在穩定提升，經過 200 個 epoch 訓練以後，會在 150 個 epoch 左右收斂到穩定，此時 mAP@50:95 約為 0.25.

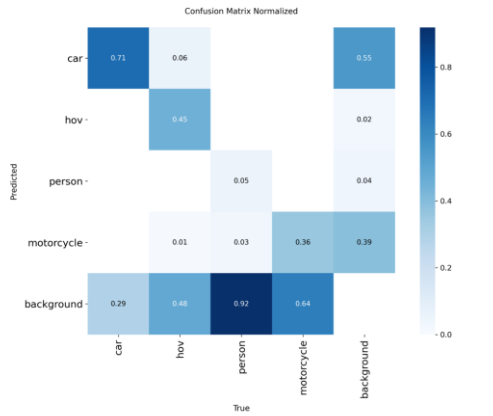
Example detection：

img0007.jpg



Confusion matrix：

confusion_matrix_normalized.png



4. Short Conclusion

本最終版本以 yolo11l.yaml 架構 從零訓練 配合 強資料擴增 與自訂 超參數 / 損失權重，建立完整的 資料→訓練→驗證→推論→提交 流程。此設計有利於在資料分佈與標註風格與通用預訓練差距較大時獲得更佳泛化；但也更依賴充足訓練輪數、合適學習率與乾淨標註。

訓練過程中我發現模型在 kaggle 上的分數都要比 val set 上的分數略高出 0.01 左右，分析後認為很有可能是由於 long tail 分佈，使得 test set 和 val set 資料分佈有差異，分數略微有波動。