MOT 模型介紹

本次作業採用 ByteTrack 作為 MOT 模型。

ByteTrack 是一種 多目標追蹤演算法（Multi-Object Tracking, MOT），其目標是在每一幀影像中持續追蹤同一物體的軌跡。它的基本概念是：

- 使用 YOLO 類的物件偵測器（如 YOLOv5）作為前端，先偵測出所有物體。
- 保留低分數的偵測框，並善用它們以提升追蹤穩定性。
- 不需外部外觀特徵提取器，靠簡單幾何資訊即可達到高準確率。

ByteTrack 的追蹤流程可分為以下步驟：

1. 物件偵測（Detection）：每一幀使用偵測器偵測所有物體，並根據信心分數將結果分為：
    - 高分數偵測（High-score detections）
    - 低分數偵測（Low-score detections）
2. 高分數追蹤配對（First Association）：使用 Kalman Filter 預測物體位置，再透過 IoU 計算進行資料關聯，將高分數偵測框配對至現有追蹤軌跡。
3. 低分數追蹤補充（Second Association）：對於第一階段未配對成功的軌跡，ByteTrack 再嘗試使用低分數偵測框進行配對。這能避免因暫時遮擋或檢測失誤導致的追蹤中斷。
4. 軌跡管理（Track Management）：
    - 若某條軌跡連續幾幀都無法配對成功，則會被刪除。
    - 對於新出現且穩定的偵測框，則會啟動新的追蹤 ID。

選擇使用 ByteTrack 的理由主要是它的結構和使用方式較簡單，容易實作和整合，需要更換偵測器的修改幅度也較小。也不需要額外資料或複雜的訓練流程。且基於網路上搜尋到的資料， ByteTrack 在追蹤上可兼顧精度和運算效率。

環境架設說明

以下為本地配置，以此環境測試 ByteTrack 搭配 YOLOv8 已檢測 Fisheye8K 和 LOAF 資料集。

硬體：

- 處理器：12th Gen Intel(R) Core(TM) i5-12600KF 3.70 GHz
- 記憶體：32 GB
- 顯卡：NVIDIA GeForce RTX 3050
- 作業系統：Windows 11 23H2

軟體：

- WSL 2.4.13.0
- Linux 5.15
- Ubuntu 22.04
- cuda 版本：12.8.90
    - 到官網下載安裝程式。
- pyTorch 版本：2.4.1

- 會在後續安裝
- torchvision==0.19.1

MOT 模型運行流程說明

Fisheye8K & LOAF

Fisheye8K 和 LOAF 在本地端測試。

以下是在本地端建立 MOT 模型和測試 Fisheye8K 和 LOAF 資料集的流程。

1. 建立作業專用資料夾

```
mkdir ~/DIP_HW3
cd ~/DIP_HW3
```

2. 建立 Python 的虛擬環境，此處使用 pyenv 搭配 Python 3.8.10 ，主要原因是 3.6 ~ 3.9 是 ByteTrack 所建議的版本。

```
pyenv install -v 3.8.10
pip3 install virtualenv
virtualenv -p ~/.pyenv/versions/3.8.10/bin/python hw3
source ./hw3/bin/activate
```

3. 下載 ByteTrack 。注意到官網提供的 requirements.txt 存在一個問題，他預設安裝最新版的 numpy ，但是大部分程式使用到 `numpy.float` 這個型別，他在 numpy 1.24 中被移除，必須降低版本。此處 1.23.5 是測試過後不會產生衝突且可使用 `numpy.float` 的版本，不過仍有警告訊息。

```
git clone https://github.com/ifzhang/ByteTrack.git
cd ByteTrack
pip3 install -r requirements.txt
pip3 install --upgrade numpy==1.23.5
python3 setup.py develop
pip3 install cython; pip3 install
'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
pip3 install cython_bbox
pip3 install -v -e .
```

4. 下載 YOLOv8 的模型和其他所需套件。此處並不是使用 ByteTrack 提供的模型。此外，需要降低 protobuf 套件的版本， 3.20.0 是一個測試之後不會與其他套件衝突的版本。

```
wget https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8x.pt
pip3 install ultralytics supervision onemetric
pip3 install --upgrade protobuf==3.20.0
```

5. 把用於 Fishey8K 和 LOAF 的 Python 程式 `fisheye8k.py` 和 `loaf.py` 移動到 ByteTrack/ 資料夾底下。此外把兩個資料集的資料夾命名為 Fisheye8K/ 和 LOAF/ ，並移動到 ByteTrack/ 資料夾底下。此時相關檔案的路徑應該如下，且 pwd 仍在 ByteTrack/ 底下：

```
ByteTrack/
+- Fisheye8K/
|  +- test/
|  |  +- images/
|  |     +- *.png
|  +- train/
|     +- images/
|        +- *.png
+- LOAF/
|  +- test/
|  |  +- *.jpg
|  +- train/
|     +- *.jpg
+- fisheye8k.py
+- loaf.py
+- yolox_x.pth
```

6. 透過以下程式分別執行 Python 檔案。如果想測試單一場景，須給定該場景的路徑，並用參數 `--name <scene id>` 來執行。

```
# Test Fisheye8K
python fisheye8k.py \
    --path Fisheye8K\
    --device cuda \
    --yolo_model yolov8x.pt \
    --fps 15 \
    --record_fake all

# Test LOAF
python loaf.py \
    --path LOAF \
    --device cuda \
    --conf 0.3 \
    --track_thresh 0.1 \
    --match_thresh 0.4 \
    --track_buffer 0 \
    --fps 15 \
    --min_box_area 0 \
    --record_fake all
```

7. 此時在 ByteTrack/ 路徑底下便會有兩個資料夾： track_result/ 和 video/ ，分別存放 csv 檔案和 mp4 檔案。

輸出結果說明

Fisheye8K & LOAF

MOT 運行結果

Fisheye8K 與 LOAF 兩個資料集皆使用 ByteTrack，搭配 YOLOv8 模型作為偵測器進行多目標追蹤。追蹤結果已儲存為指定路徑下的 CSV 與 MP4 檔案。

從輸出影片中可以觀察到，大部分的人與車輛在 YOLOv8 的偵測下皆能正確辨識。然而在 LOAF 資料集中，因鏡頭裝設方式接近俯視，且為魚眼鏡頭，畫面下半部的物件在幾何上產生明顯扭曲與上下翻轉，導致 YOLOv8 難以正確辨識。整體來看，魚眼鏡頭的影像變形對物件偵測造成了明顯挑戰，也進一步影響追蹤的穩定性。且兩個資料集皆為每秒 1 張影像（1 FPS）的低頻率輸出。如此低的取樣頻率使得物體在不同幀之間的位移過大，進一步降低追蹤的連貫性與準確性。

物件追蹤參考了助教建議的做法，除了使用原本的 ByteTrack 以外，也採用了資料集中的 annotations 資料來協助。不過實際測試仍然存在著難度，因為這麼做變相的是必須自己實作，將未被追蹤到的物件綁定至 annotations 中的資料，並在每一幀中嘗試進行追蹤，難度自然提升不少。事實上，在 Fisheye8K 中有一些物件有被成功綁定，但是在 LOAF 資料集中，這麼做的成功率非常低，和原先 ByteTrack 的表現類似。因此最後這兩個資料集的結果， Fisheye8K 有採用 annotations 作為協助，但 LOAF 則沒有。

此外，我也進行了一個測試，在 Fisheye8K 的資料集中，將 annotations 中的 box 輸入至 ByteTrack 中，結果和我原本未經任何修改的輸出很類似，追蹤結果不太好。

綜合而言，追蹤表現會受到鏡頭視角、幀率與環境光源等多種因素影響。在目前的測試中，接近水平視角、光照良好的白天場景下，ByteTrack 的追蹤結果最為穩定與準確。

實作中遇到怎樣的困難

實作中遇到的第一個困難就是硬體限制，畢竟不是每個人都有足夠好的顯卡。因此必須花費更多時間在等待處理結果。在此提供個人實際操作所得到的約略數據：在使用本機的設定進行實驗時，Fisheye8k 大約花了 14 分鐘，而 LOAF 資料集則花費約 1 小時。如果參數調整或是程式設計不當，便要浪費大量時間。此外，數十 GB 的測資更是佔據大量硬碟空間。

此外，原本 ByteTrack 建議使用的模型是 `bytetrack_x_mot17.pth.tar` ，這個模型只能辨識行人，因此後續又更換其他模型。率先嘗試的是 yolox ，但是它無法很好的應對魚眼鏡頭。因此後續又改為 YOLOv8 ，偵測結果尚可。

最後一個很重大的困難是教材。有關 MOT 的教材不少，但是大多不是很完整，且似乎假設使用者已具備基礎知識。此處主要參考官方提供的 demo 程式，和一個「YOLOv8+ByteTrack多目標跟蹤(行人車輛計數與越界識別)」的教學。後續想嘗試針對魚眼鏡頭的改善，也幾乎沒有看到針對魚眼鏡頭的 MOT 教學，少數教學僅有物件偵測的部分。

結果的哪部分表現不佳

表現不佳主要分成兩個部分：物件偵測和物件追蹤。

物件偵測的表現在使用 YOLOv8 之後，相較先前的 yolox 有明顯改善，不過在鏡頭邊緣而導致圖片扭曲，或是俯視視角導致物件上下顛倒這兩個部分，偵測的準確度函式略低。如果為了增加物件被辨識的機率而降低信心值，反而會導致誤認。

物件追蹤的表現完全仰賴 ByteTrack 的精度和結果，雖然有接受到偵測物件，但無法很好的進行匹配。以魚眼鏡頭拍攝的的 Fisheye 和 LOAF 資料集追蹤果很差。我們把可能的原因列出：

1. 魚眼鏡頭導致圖片扭曲，因此物件的外觀與比例變形，雖然 YOLO 可能有偵測到物件，但是變形的物體仍會影響追蹤的準確度。
2. 俯視視角使物體外觀、大小、距離都被壓縮，對追蹤階段（卡爾曼濾波與 IoU 配對）不利。
3. 由於幀與幀的時間間隔過大（1 FPS），物體移動幅度大，導致 ByteTrack 誤認為是新物體，導致 ID 改變，或是視為新的追蹤目標。
4. ByteTrack 不使用 ReID 模組，僅依靠幾何資訊進行配對，容易導致混淆。

可以如何改進

針對物件偵測的部分，可以嘗試的改進如下：

1. 嘗試使用魚眼素材訓練的模型。網路上有查到一款名為「YOLOv9-FishEye」的專案，似乎有使用魚眼素材來進行訓練，以克服扭曲和顛倒的問題。
2. 有其他文章指出，可以透過幾何運算來修正物體被顛倒的問題，如果偵測時均使用方向相同的素材，則偵測的準確率也可以提高。

針對物件追蹤的部分，可以嘗試的改進如下：

1. 加入魚眼影像的幾何校正，使扭曲的部分盡量轉換為接近常規透視影像。
2. 加入 ReID 功能，例如 Deepsort 的外觀遷入向量作法（當然也可以嘗試其他 MOT 模型）。這可以在幾何資訊不足以區分相似目標時發揮效果。我在嘗試 LOAF 資料集時有嘗試為每個物件都建立簡單的色彩直方圖，利用這個直方圖做為簡單的特徵，不過這個做法會將多個不同目標混淆在一起，因此沒有被採用。
3. 使用時間差值（如補幀）或運動預測（非簡單線性的卡爾曼濾波）補償低幀率的問題，改善中間軌跡的連貫性。
4. 調整 ByteTrack 的參數來適應場景特性（不過我陸續測試了幾組參數，差異都不是特別明顯。）

我後來找到一篇「Fisheye Multiple Object Tracking by Learning Distortions Without Dewarping」的報告，其中實作了包含「引入 Re-ID」、「使用 HDS 模組來試圖補償大幅位移」、「根據魚眼鏡頭下的視角進行資料關聯」，以及「針對魚眼資料進行訓練與微調」等等。實際上這篇報告不僅來自交大，採用的資料也是 Fisheye8K ，因此上述的改善方式如果可以實現，應該可以對當前的追蹤效果提升許多。

環境架設說明2

以下為本地配置，以此環境測試 ByteTrack 搭配 YOLOv8 以檢測 MOT17 資料集，因無GPU，故僅使用CPU運算。

硬體：

- 處理器：12th Gen Intel(R) Core(TM) i7-1260P 2.10 GHz
- 記憶體：32 GB
- 顯卡：IIntel Iris Xe Graphics(無法使用cuda)
- 作業系統：Windows 11 24H2(OS 組件26100.4061)

軟體：

- Anaconda3 2024.10-1(conda 24.11.3)
- python 3.7.16

MOT 模型運行流程說明

MOT17

MOT17 在本地端CPU測試。

以下是在本地端建立 MOT 模型和測試 MOT17 資料集的流程。

1. 建立 作業專用資料夾

```
mkdir ~/DIP_HW3
cd ~/DIP_HW3
```

2. 在anaconda中建立 虛擬環境 name: DIP, python:3.7.16(github上作者建議的環境)
3. 打開anaconda prompt，並進入虛擬環境DIP --> $ conda activate DIP
4. 下載byteTrack

```
git clone -q https://github.com/ifzhang/ByteTrack.git
```

5. 安裝 requirements

```
cd ByteTrack
pip install -r requirements.txt
pip install ultralytics
pip install cython; pip3 install
git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI
pip install cython_bbox
pip install faiss-cpu
pip install onemetric
```

5. 按github上的installation執行 $python setup.py -q develop
6. 下載並解壓縮MOT17 資料集至 ~/DIP_HW3/ByteTrack/MOT17/
7. 下載byteTrack_x_mot17.pth.tar模型("https://drive.google.com/uc?id=1P4mY0Yyd3PPTybgZkjMYhFri88nTmJX5")放至 ~/DIP_HW3/ByteTrack/pretrained/
8. 把用於MOT17的程式 `mot17_tracker.py` 移動到 ~/DIP_HW3/ 資料夾底下。此時相關檔案的路徑應該如下：

```
~/DIP_HW3/
+- ByteTrack/
+- MOT17/
|  +- test/
```

```
|  |  +- MOT17-[01-14]-.*/
|  |     +- img1/
|  |        +- *.jpg
|  |     +- det/
|  |        +- det.txt
|  |     +- seqinfo.ini
|  +- train/
|  |  +- MOT17-[01-14]-.*/
|  |     +- img1/
|  |        +- *.jpg
|  |     +- det/
|  |        +- det.txt
|  |     +- gt/
|  |        +- gt.txt
|  |     +- seqinfo.ini
+- mot17_tracker.py
+- yolov8x.pt
```

9. 透過以下程式執行 Python 檔案。如果想測試單一場景，須給定該場景的名稱，場景需在test資料夾中，並用參數 `--seq <scene id>` 來執行。 參數num_workers可自行調整

```
# Test part of MOT17(about 1 hour)
python mot17_tracker.py \
    --path ./MOT17 \
    --seq MOT17-01-DPM \
    --device cpu \
    --yolo_model yolov8x.pt \
    --num_workers 6
```

```
# Test all MOT17
python mot17_tracker.py \
    --path ./MOT17 \
    --device cpu \
    --yolo_model yolov8x.pt \
    --num_workers 6
```

10. 此時在 ~/DIP_HW3/ 路徑底下便會有兩個資料夾： MOT17_track_result/ 和 MOT17_videos/ ，分別存放 csv 檔案和 mp4 檔案。

輸出結果說明

MOT17

運行結果

MOT17資料集使用 ByteTrack，搭配 YOLOv8 模型作為偵測器進行多目標追蹤。追蹤結果已儲存為指定路徑下的 CSV 與 MP4 檔案。

從輸出影片中可以觀察到，大部分的移動中的行人 YOLOv8 的偵測下皆能正確辨識。目前我的實作中路徑track的結果不好。

實作中遇到怎樣的困難

- 實作中遇到的第一個困難就是硬體上的限制，畢竟不是每個資工系的學生電腦都有可以用cuda的顯卡，因此我一開始是想要在google colab上跑，但google colab的環境是python 3.11.12, 和bytetrack的numpy版本不相容，在降到numpy 1.23.5後發現numpy 1.23.5和bytetrack需要的onnxruntime版本不相容，且沒有其相容的版本(colab 屍體1: https://colab.research.google.com/drive/1adYUxtdzriXhivrDyfOiMrfMdfrrrrw-?usp=sharing)。而後嘗試將colab環境改成3.7.16，發現colab的wheel無法安裝。後嘗試安裝miniconda，建立虛擬環境，安裝requirements時都很順利，但執行bytetrack時發先系統的python import路徑和虛擬環境的python路徑不相同，導致import 失敗(colab 屍體2，部分已被焚毀: https://colab.research.google.com/drive/1FCMlmucVBpAZUxN20QFQNhvGZFCD5zCz?usp=sharing)。在嘗試了無數種方法後，在deadline的壓力下，決定先在本機cpu上跑，雖然速度慢了 **億** 點點，但至少可以跑出結果。

- 實作嘗試中的第二個問題是colab上session之間的環境和設定、資料無法共通，且因為沒有買google drive的空間，資料集、bytetrack和輸出結果無法都存到google drive，導致每次都要重新解壓縮資料集(約7分鐘)和安裝bytetrack以及對應的requirement(約20分鐘)，且colab的session時間有限制(因為免費版的T4 GPU 算力配額)，導致需要切換帳號(然後又要重來...)。- 後來在本機上跑時，發現ByteTrack的預設模型是`bytetrack_x_mot17.pth.tar`，這個模型只能辨識行人，後來更換成yolov8x的模型，但對行人以外的物件還是偵測不佳，因剩餘的時間已不足再更換模型，所以就只能先這樣了。

- 最後一個很重大的困難是教學資源。有關 MOT 的教材不少，但是大多不是很完整或是版本不同，且似乎假設使用者已具備基礎知識。此處主要參考隊友的loaf.py以及網路上的官方和其他人的教學，和官方 github issue。

- 關於環境設定的部分最後在bytetrack issue #334中 hgquannn指向的issue #349中 ricky-696指向的 AICUP_Baseline_BoT-SORT issue #10中找到可行的環境版本設定(但最後在colab中仍然無法成功建立環境)

結果的哪部分表現不佳

表現不佳主要分成兩個部分：物件偵測和物件追蹤。

物件偵測的表現在使用 yolov8x 之後，相較先前的 bytetrack_x_mot17 有少許改善，但不顯著，這部分我沒有對其他scence驗證，因為cpu運算時間實在太長，在切換到畚箕計算後剩餘的時間不夠(sorry, my fault)。物件追蹤的表現完全仰賴 ByteTrack 的精度和結果，雖然有接受到偵測物件，但無法很好的進行匹配。

可以如何改進

針對物件偵測的部分，可以嘗試的改進如下： 物件偵測改用其他模型(待尋找)， track的結果需要進一步的處理

分工表

Fisheye8K & LOAF ：陳均凱
MOT17: 周哲煒

完整檔案: https://e.pcloud.link/publink/show?code=XZNYJtZa1UBJoOXeM0COEciB1e9P8lzDkGy