DIP Final Project – Object Detection

姓名:陳韋澎 班級:智能系統研究所 學號: 411581008

****完成程式碼與實驗結果請見:**

https://github.com/IAII-411581008/dip_final_proj/tree/main

一、資料集介紹

COCO 資料集是一個大規模的影像資料集常用於物件檢測與語義分割等任務。對於從事目標檢測的學者和開發人員而言，這是一個必不可少的數據集。其具有超過 20 萬張的影像資料、80 個種類的資料共大型影像模型進行權重訓練。因此，本次實驗採用 COCO 資料集的所有物件檢測資料進行神經網路模型訓練。測試資料集將以課堂所提供之影片(person_dog.mp4)進行模型預測。

二、實驗過程

i) 作業系統環境建置

考量到實驗室與個人資源有限，並且受到桌電記憶體與效能的限制，因此最終採用 WSL2 (Windows Subsystem for Linux)而非傳統雙系統的方式進行作業系統建置。WSL2 的安裝非常簡單，僅需於 cmd 內輸入 wsl--install 並下載 Ubuntu LTS 22.04 即可。然而，在安裝 CUDA 時需要特別注意的是 NVIDIA CUDA 的驅動器僅需安裝於 Windows 端即可，WSL2 端無需另外安裝 CUDA 驅動器。考量到 CUDA 與 NVIDIA RTX 3080 顯示卡的支援度，本次實驗採用 CUDA 11.2 與 CUDNN 8.5.0 用以加速模型矩陣運算。詳細開發環境與套件版本規格，如: Python3.10，Matlab R2022B, Cmake 3.22.1，Pytorch，Scipy 等如下表所示:

圖表 1、開發環境之軟體與硬體版本與規格

硬體/軟體套件	規格/版本
GPU	NVIDIA RTX 3080 (10GB)
CPU	Intel i9-12900K@3.20GHz
DRAM	64GB
CUDA	11.2
CUDNN	8.5.0
Python	3.10
Matlab	R2022b
Cmake	3.22.1
Matplotlib	3.3.0
Numpy	1.22.2
Opencv-python	4.6.0

Pillow	7.1.2
Pyyaml	5.3.1
Requests	2.23.0
Scipy	1.4.1
Torch	1.8.0
Torchvision	0.9.0
Tqdm	4.64.0

ii) 建立程式集開發架構

完整程式集開發架構如下圖所示:

```
[ultralytics]/
    docker/
    docs/
    examples/
    path/
    runs/
        detect/
            yolov8x_final/ # final prediction results
                frame2video_pre_frames/ # predicted frames
                    0.png
                    1.png
                    ...

                labels/ # predicted annotations
                    frame2video_pre_1.txt
                    frame2video_pre_1.txt
                    ...
                frame2video_pre.avi # raw predicted video

        frame2video_pre/ # convert into video
            frame2video_pre.avi
        img2video/
            person_dog.avi # predicted video w/o preprocessing
            person_dog_yolov8x_final # final predicted video w/ preprocessing
        plot_count_label/ # convert into frames
            xx.png
            ...
        TA_result/ # convert into frames
            xx.png
            ...
        video2frame_pre/ # convert into frames
            xx.png
            ...
        video2images/ # convert into frames
            xx.png
            ...

    tests/ # for env testing
    ultralytics/ # build model utils

    person_dog.mp4 # raw testing video
    count_num_classes.py # overall process of labels' counting
    img2video.py # convert into video
    plot_count_label.py # count each label
    video2img.py # convert into frames
    video_preprocessing.mlx # exp img preprecessing
    yolov8x.pt # YOLOv8X pretrained weights
    requirements.txt # install required toolkits
```

圖 1、程式集開發架構圖

iii) 測試影片前處理

我們觀察到在原始影像的光線明亮度降低(如圖 2 所示)，且部分人物與背景的對比對低。於圖 2(右)原始直方圖所示，原始影像整體的 intensities 集中分佈於 150 以下，這使得神經網路模型難以辨識出遠處樹林旁對比度與亮度降低的人類(如圖 3(左)所示)。因此，我們對於原始影像進行

了一系列處理。首先，我們觀察到影像中的人正好都穿著深色系的服裝，並且正好都露出其小腿，深色服裝與皮膚的顏色在影像強度上具有一定的差異性，由此我們希望透過 Laplacian Filtering 的方式去強化其邊緣部分，使得這些特征得以被模型提取。我們透過一 4-neighborhood 的 Laplacian 濾波器，設定其 filter coefficients 為[0, 1, 0; 1, -4, 1; 0, 1, 0]進行空間濾波，提取其高頻資訊，在將這些高頻資訊加入原始影像中，以此強化物體邊緣。然而，考量到影像中大部分的高頻成分將集中在草地的部分，為了避免過度強化草的邊緣進而惡化模型的檢測結果，在反復進行試驗後，我們最終將高頻遮罩的 coefficient 設定為 0.3。接著，從原始影像直方圖可觀察到影像整體強度較低，而 Power-law Transformation 的方式恰好適用於擴大低強度範圍並且壓縮高強度範圍。因此，我們利用了 Gamma Correction 技術，將其 Gamma Coefficient 設定為 0.8 以此改善影像中部分對比度較低的現象。透過對比圖 3 影像處理前與處理後的結果，可觀察到樹林旁的二人與其背景的對比度明顯提升了。最後，我們對於影片中的 181 個 Frames 進行影像前處理，以此強化模型的特征提取結果。影像前處理的完整程式碼如圖 6 所示。

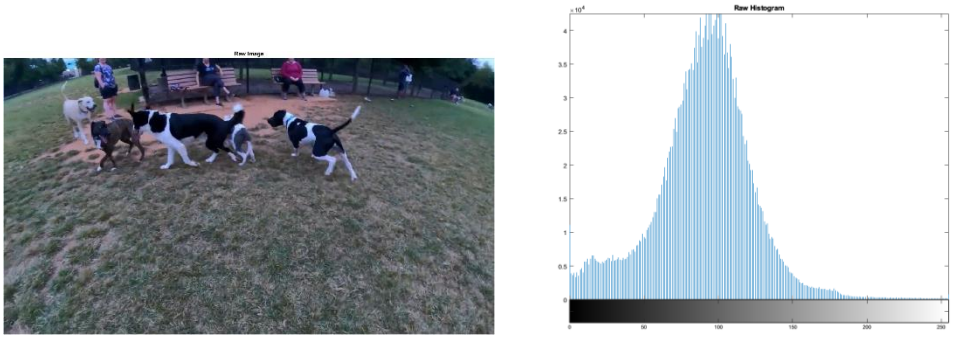


圖 2、原始第 168 Frame 的影像(左)與其原始 Histogram。

圖 3、影像處理前(左) v.s. 影像處理後(右)的明亮與對比度

圖 4、Laplacian Filtering 前(左) v.s. Laplacian 邊緣偵測(右)之影像

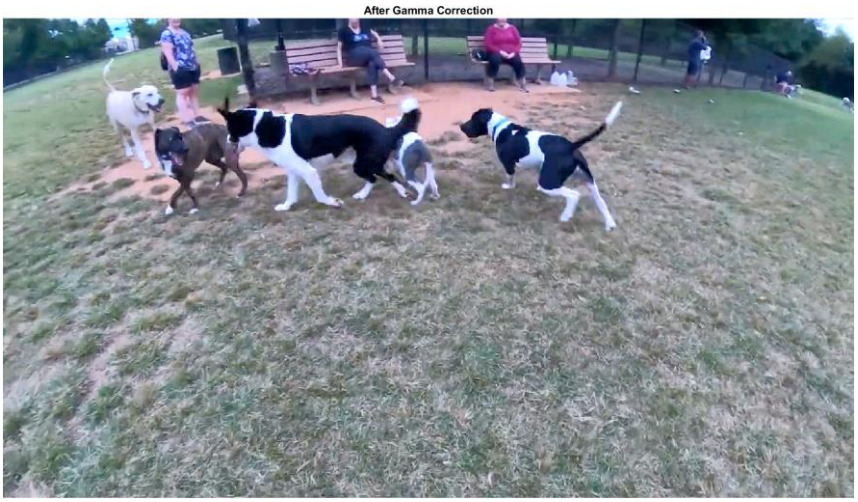


圖 5、第 168 Frame 影像前處理後的結果

```
% Specify the path to the video file
videoFilePath = 'person_dog.mp4';

% Create a VideoReader object
videoReader = VideoReader(videoFilePath);

% Create a folder to save the frames
outputFolder = 'runs/video2frame_pre';
if ~exist(outputFolder, 'dir')
    mkdir(outputFolder);
end

% Loop through each frame and save it as an image
frameCount = 1;
while hasFrame(videoReader)
    % Read the frame
    frame = readFrame(videoReader);

    % Convert the input image to double datatframeype
    double_img = im2double(frame);

    % Use the Laplacian filter to highlight edges
    L_filter = fspecial('laplacian', 0.3);  % Defining Laplacian filter
    filtering = imfilter(double_img, L_filter, 'replicate');    % Applying Laplacian filter to image
    filtered_img = double_img - filtering;

    imshow(filtering*3)

    % Selecting a scaling factor for high boost filtering
    S = 0.3;

    % Applying high boost filtering to Laplacian filtered image
    sharpen_img = double_img + S * filtered_img;

    % Applying Gamma Correction
    gammaImg = imadjust(sharpen_img,[],[],0.8);

%     imshow(gammaImg)
%     title('After Gamma Correction')
%
%     imhist(gammaImg)
%     title('Histogram After Gamma Correction')

    % Create the filename for the current frame
    filename = sprintf('frame_%04d.png', frameCount);

    % Save the frame as an image
    imwrite(J, fullfile(outputFolder, filename));

    % Increment the frame count
    frameCount = frameCount + 1;
end

disp('Frames extracted successfully.');
```

圖 6、影像前處理程式碼

iv) 測試影片後處理

完成測試樣本預測後，我們需在每一幀均附上各個種類成功檢測數量，具體流程如下:

1. 將原始影片 person_dog.mp4 以 30 FPS 分解成 181 個 Frames
2. 透過 YOLOv8X 的預測結果，計算各個 Frame's Annotation 中物品種類的數量
3. 將學號、Person 及 Dog 數量以綠色標籤 plot 與各個 Frame 左上角
4. 最後，將這些標籤好的 Frames 重新轉檔回影片 .avi 檔

plot_num_classes.py > plot_label_counts_on_image

You, 3 hours ago | 1 author (You)

```python
import cv2
import os

''' video2images '''
def video_to_images(video_path, output_folder):
    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Open the video file
    cap = cv2.VideoCapture(video_path)

    # Get the frames per second (fps) and frame width/height
    fps = cap.get(cv2.CAP_PROP_FPS)
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Get the total number of frames in the video
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    print(f"FPS: {fps}, Width: {width}, Height: {height}, Total Frames: {total_frames}")

    # Loop through each frame and save it as an image
    for frame_num in range(total_frames):
        ret, frame = cap.read()

        if not ret:
            break

        # Save the frame as an image
        image_path = os.path.join(output_folder, f"person_dog_{frame_num:04d}.png")
        cv2.imwrite(image_path, frame)

        # Display progress
        if frame_num % 100 == 0:
            print(f"Processing frame {frame_num}/{total_frames}")

    # Release the video capture object
    cap.release()

    print(f"Frames saved to {output_folder}")

''' count each label '''
def count_labels(annotation_file):
    # Read the annotation file and extract labels
    with open(annotation_file, 'r') as file:
        lines = [line.strip().split() for line in file]

    # Extract labels from the first column
    labels = [line[0] for line in lines]

    # Count occurrences of labels "0" and "16"
    label_counts = {"Person": labels.count("0"), "Dog": labels.count("16")}

    # Print the counts
    for label, count in label_counts.items():
        print(f"{label}: {count}")

```

```python
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw, ImageFont

def plot_label_counts_on_image(image_path, annotation_file,frameIdx):
    # Read the annotation file and extract labels
    with open(annotation_file, 'r') as file:
        lines = [line.strip().split() for line in file]      You, 3 hours ago • preprocessing

    # Extract labels from the first column
    labels = [line[0] for line in lines]

    # Count occurrences of labels "0" and "16"
    label_counts = {"Person": labels.count("0"), "Dog": labels.count("16")}

    # Create an image with the counts displayed in the top left corner
    original_image = Image.open(image_path)

    # Create an ImageDraw object to add text to the image
    draw = ImageDraw.Draw(original_image)

    # Specify the font and size for the text (increased font size)
    font_size = 30
    font = ImageFont.truetype("arial.ttf", font_size)

    # Add text indicating the count of each label in the top left corner with bold, red text and line breaks
    text = ""
    draw.text((10, 10), f"ID: {411581008} \n", font=font, fill=(0, 255, 0))
    text += "\n"
    for label, count in label_counts.items():
        # Render the text twice for a bold effect
        if label == "Person":
            draw.text((10, 60), f"{label}: {count}", font=font, fill=(0, 255, 0))
        else:
            draw.text((10, 110), f"{label}: {count}", font=font, fill=(0, 255, 0))
        # draw.text((11, 11), f"{label}: {count}", font=font, fill=(255, 0, 0))

    final_image_path = os.path.join("runs\plot_count_label", f"person_dog_{frameIdx:04d}.png")
    original_image.save(final_image_path)

    # Display the final image
    # original_image.show()

''' images2video '''
def frames_to_video(frames_folder, output_video_path, fps):
    # Get the list of image files in the frames folder
    image_files = [f for f in os.listdir(frames_folder) if f.endswith(('.png', '.jpg', '.jpeg'))]

    if not image_files:
        print("No image files found in the specified folder.")
        return

    # Sort the image files based on their numerical names
    image_files.sort(key=lambda x: int(''.join(filter(str.isdigit, x))))

    # Get the dimensions of the first image
    first_image = cv2.imread(os.path.join(frames_folder, image_files[0]))
    height, width, _ = first_image.shape

    # Create a VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    video_writer = cv2.VideoWriter(output_video_path, fourcc, fps, (width, height))

    # Iterate through image files and write frames to the video
    for image_file in image_files:
        frame = cv2.imread(os.path.join(frames_folder, image_file))
        video_writer.write(frame)
        # print(image_file)
    # Release the video writer
    video_writer.release()

    print(f"Video successfully created: {output_video_path}")
```

```python
if __name__ == "__main__":
    ## video2frames
    # Specify the path to the input video and output folder
    video_path = r"runs\detect\yolov8x_final\frame2video_pre.avi"
    output_folder = r"runs\video2images"

    # Convert the video to images
    video_to_images(video_path, output_folder)

    ## count each label
    annotationDir = r"E:\research_proj\dip_final_proj\ultralytics\runs\detect\yolov8x_final\labels"
    imgDir = r"E:\research_proj\dip_final_proj\ultralytics\runs\video2images"

    annotationFiles = os.listdir(annotationDir)
    # Sort the image files based on their numerical names
    annotationFiles.sort(key=lambda x: int(''.join(filter(str.isdigit, x))))

    imgFilenames = os.listdir(imgDir)

    for fileIdx in range(0,len(annotationFiles)):
        annotationFile = os.path.join(annotationDir, annotationFiles[fileIdx])
        imgFile = os.path.join(imgDir, imgFilenames[fileIdx])
        # Specify the path to your annotation file
        # annotation_file = r"E:\research_proj\dip_final_proj\ultralytics\runs\detect\predict11\labels\perso

        # Count and print the occurrences of labels "0" and "16"
        count_labels(annotationFile)

        # Plot the label counts on the image
        plot_label_counts_on_image(imgFile, annotationFile,fileIdx)

    ## convert back to video
    frames_folder = r"runs\plot_count_label"
    output_video_path = r"runs\imgs2video\person_dog_yolov8x_final.mp4"
    fps = 30  # Specify the desired frames per second (fps) for the output video

    # Replace "path/to/frames" with the path to your folder containing frames
    frames_to_video(frames_folder, output_video_path, fps)
```

圖 7、影像後處理程式碼

三、預訓練模型與超參數設定

本次實驗採用 YOLOv8X 進行物件檢測。YOLO 系列的模型常用於物件檢測任務，其具有非常好檢測結果的同時，整體運算速度也相較於 RCNN-based 模型來的快。因此，我們最終採用最新版 YOLO 進行物件檢測。YOLOv8 自行開發了 C2f 模塊，並替換了 Darknet-53 模型中的 C3 模塊。損失函數的部分，其采用了 TaskAlignedAssigner 的方式進行標籤分配與 Distribution Focal Loss DIOU NMS Loss 與 CIOU Loss 進行權重調整。具體模型架構可參考圖 8。

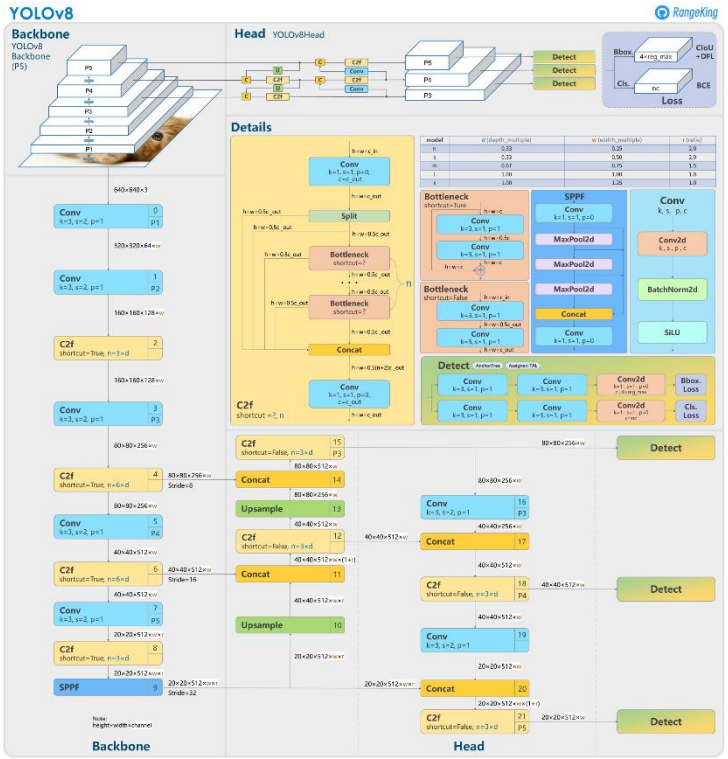


圖 8、Yolov8 架構圖 (Source: Roboflow)

圖表 2 訓練超參數設定

超參數種類	參數值
Model Weights	yolo8x.pt
Confidence Score Threshold	0.2
IoU Threshold	0.5
Classes	0, 1, 16
Image Size	640
Max_det	300
Vid_stride	False
Agnostic_nms	False
Augment	False

四、實驗結果與討論

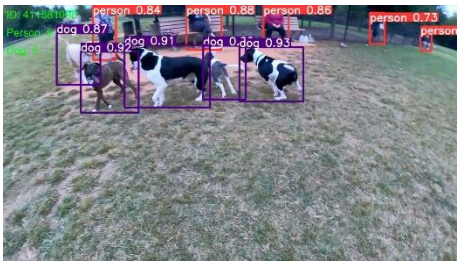
實驗結果的部分，我們劃分成兩個部分進行討論: (1)對比使用影像處理後對於模型檢測結果的影響; (2)與助教的實驗結果進行比較。圖 9 為 YOLOv8X 在相同模型權重與超參數設定下，對於影像處理前與處理後的檢測結果。透過對比圖 9 的十張圖片可發現，右上角樹林旁的二人在經過 Laplacian Sharpening 與 Power-law Transformation 後更容易被 YOLOv8X 檢測出來。其主要的原因是這兩個人的腿部特征受到 High-pass Filtering 後，其邊緣部分可明顯與背景劃分開來，並且在 Gamma Correction 的幫助下擴大 low-level intensities range，使得這兩個人深色衣服的特征與陰暗背景的對比度得以進一步提升，進而促使這兩個人的特征得以成功被模型提取。另外，我們亦可對比影像處理前與處理後人物的 Confidence Score 差異性。圖八的結果所示，經影像處理後，人物整體的 Confidence Scores 明顯高於原始影像的結果，尤其是在右上角樹林下的兩個人的 Confidence Scores。然而，影像處理後亦可能帶來負面的影響，如: 狗狗整體的 Confidence Scores 在進行影像處理後的表現略低於原始影像，這可能是因為 Laplacian 銳利化了草地的特征，進而導致不必要的“雜訊”被放大，影像模型的檢測結果。因此，我們在設計 Laplacian Filtering 時特意限制了其 Mask’s Coefficient，避免過量的高頻資訊惡化模型的檢測結果。但透過對比左半部分與右半部分的結果，影像處理對於狗狗的影響並未非常明顯，整體 Confidence Scores 依舊保持在 80%以上。最後，圖 9(i)、(j)的結果所示，經影像處理後 YOLOv8X 得以成功檢測出所有人物與狗狗，特別是右上角的二人與狗狗。

第二部分我們同樣的任意挑選個五張圖片與助教的實驗結果進行對比。透過對比左半部分與右半部分的結果可以發現，我們的研究方法對於人物的檢測結果更加的敏銳，如圖 10(d)的結果所示(中間凳子上的二人)，本研究方法得僅需部分人物特征(i.e., 小腿與鞋子的特征)便可成功檢測出相關人物。其可能原因有幾種: (1)本次實驗使用的影像處理技術得以有效地強化膚色、鞋子與背景的對比度，使得這些特征得以成功被模型提取; (2)本次實驗所采用的模型為最新版 YOLO 模型，模型整體權重與提取技術由於過往 YOLO 模型; (3)本次實驗所設定的 Confidence Score Threshold”可能”低於助教的預設值，因此這些物件標籤得以在較低的 Confidence Score 下被檢測出來。透過圖 10(f)、(g)的實驗結果並結合圖 9 的實驗結果得以證實上述(1)、(2)的推理，我們在採用了 YOLOv8X 模型後亦可檢測出樹林下的人物(右上角第二個人)，並且在進行影像強化後，模型最終得以檢測出右上角最右邊的人物及狗狗並同時提升了人物整體的 Confidence Score。因此，本研究方法整體對於右上角樹林下的二人及狗狗檢測結果較為精準。另外，本研究方式對於重疊物體的檢測率較由於助教的方式，如圖 10(g)、(h)的實驗結果所示(中間偏左的三隻狗狗與一個人)，助教的研究結果並未檢測出後面的三隻狗狗及人物，而本研究方式不僅能檢測出人物前的兩只狗狗，對於人物後的狗狗亦能精確地成功檢測。最後，圖 10(i)、(j)所示為 5 人 6 狗的影像，助教之研究結果漏檢了中間重疊的一隻狗及右上角遠處的狗狗及人物，而本研究方式得以成功檢測出所有人物與狗狗。由此可見，在使用 YOLOv8X 並進行影像強化後得以提升模型整體的檢測結果。

(a)

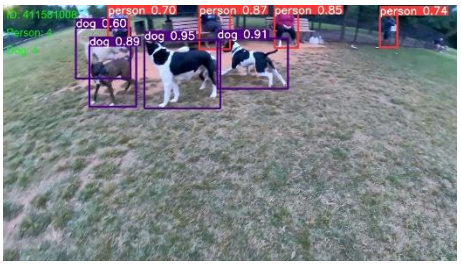
(b)

(c)

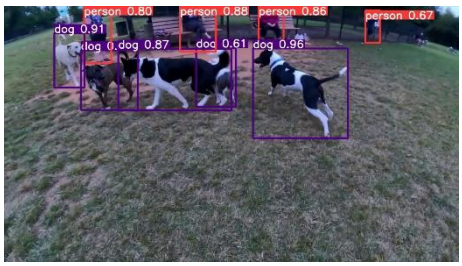


(d)

(e)

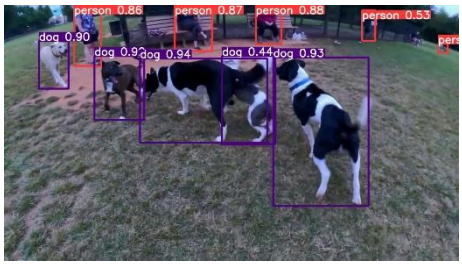


(f)

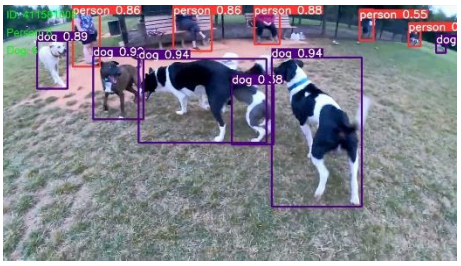


(g)

(h)

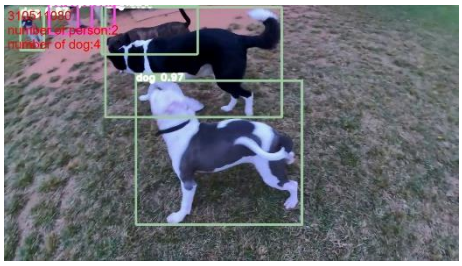


(i)

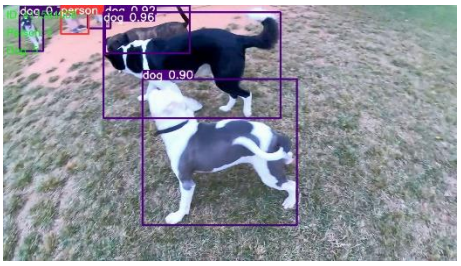


(j)

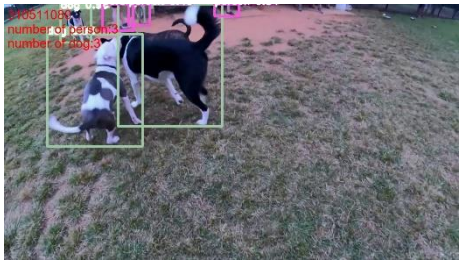
圖 9、YOLOv8X 對於 person_dog.mp4 物件檢測結果。(a)、(c)、(e)、(g)、(i)為採用原始影像的檢測結果; (b)、(d)、(f)、(h)、(j)為採用影像處理後的檢測結果



(a)

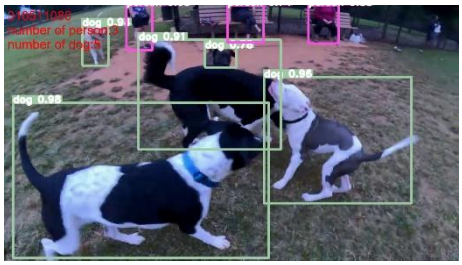


(b)

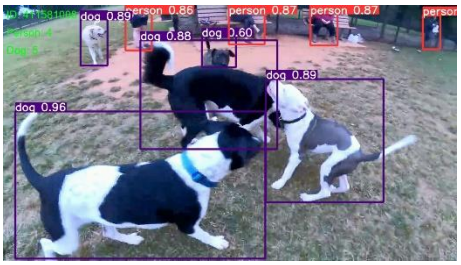


(c)

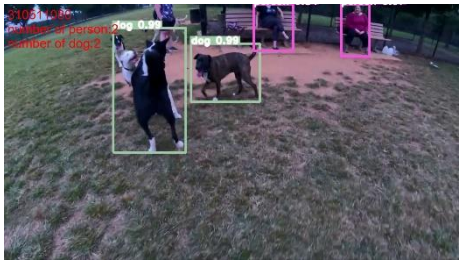
(d)



(e)

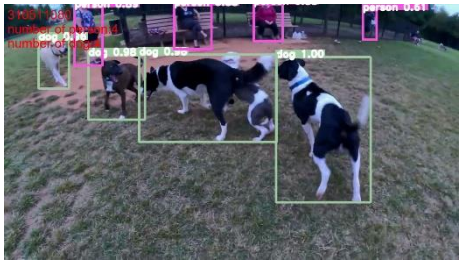


(f)

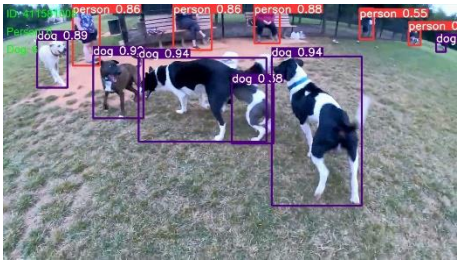


(g)

(h)



(i)



(j)

圖 10、person_dog.mp4 物件檢測結果。(a)、(c)、(e)、(g)、(i)為助教的檢測結果; (b)、(d)、(f)、(h)、(j)為本次實驗的檢測結果