**Student Name:** Dang Khoa Le

**Student ID:** 103844421

**Tutorial Session:**  Tuesday 6.30-8.30 PM

**Portfolio Week 6 Assessment**

**Assessment Project Structure**

The following directory and file structure was used for this project:

* **annotations/**: Contains train\_labels.csv and test\_labels.csv – the annotations for the training and test datasets in CSV format (not in YOLO format).
* **images/train/**: Contains the images used for training.
* **images/test/**: Contains the images used for testing.
* **labels/train/**: Contains the YOLO-format annotations (generated in STEP 1) for the training dataset.
* **labels/test/**: Contains the YOLO-format annotations (generated in STEP 1) for the test dataset.
* **selected\_train\_images/**: Contains a subset of 400 randomly sampled images and corresponding YOLO annotation files used for training.
  + images/: Contains the selected training images.
  + labels/: Contains the corresponding YOLO annotation files for the selected images.
* **data.yaml**: Configuration file for training in YOLOv5, specifying class names, paths to the dataset, and training parameters.
* **train.py**: Script from the YOLOv5 GitHub repository used to train the model.
* **yolov5m.pt**: Pre-trained model used as the starting point for training (STEP 2).
* **runs/train/exp/weights/**: Stores the model weights (best.pt and last.pt) after training is completed (STEP 2).
* **iou\_results\_iter\_{i}.csv**: CSV files generated at each iteration (STEP 3) containing the image name, confidence value, and IoU value for 40 images randomly sampled from the test dataset. ‘i’ is the iteration ID made in STEP 3.
* **runs/train/exp{n+1}/weights/**: Stores the model weights (best.pt and last.pt) after training is completed. ‘n’ is the iteration ID made in STEP 3 (starting from exp2).
* **live\_video/**: Directory containing video files for STEP 4.
* **output\_videos/:** Directory to save output videos with graffiti detections in STEP 4.

**STEP 1: Convert Annotations to YOLO Format**

The dataset annotations provided in train\_labels.csv and test\_labels.csv follow a non-YOLO format. YOLO requires annotations in the following format:

<object\_class> <x\_center> <y\_center> <width> <height>

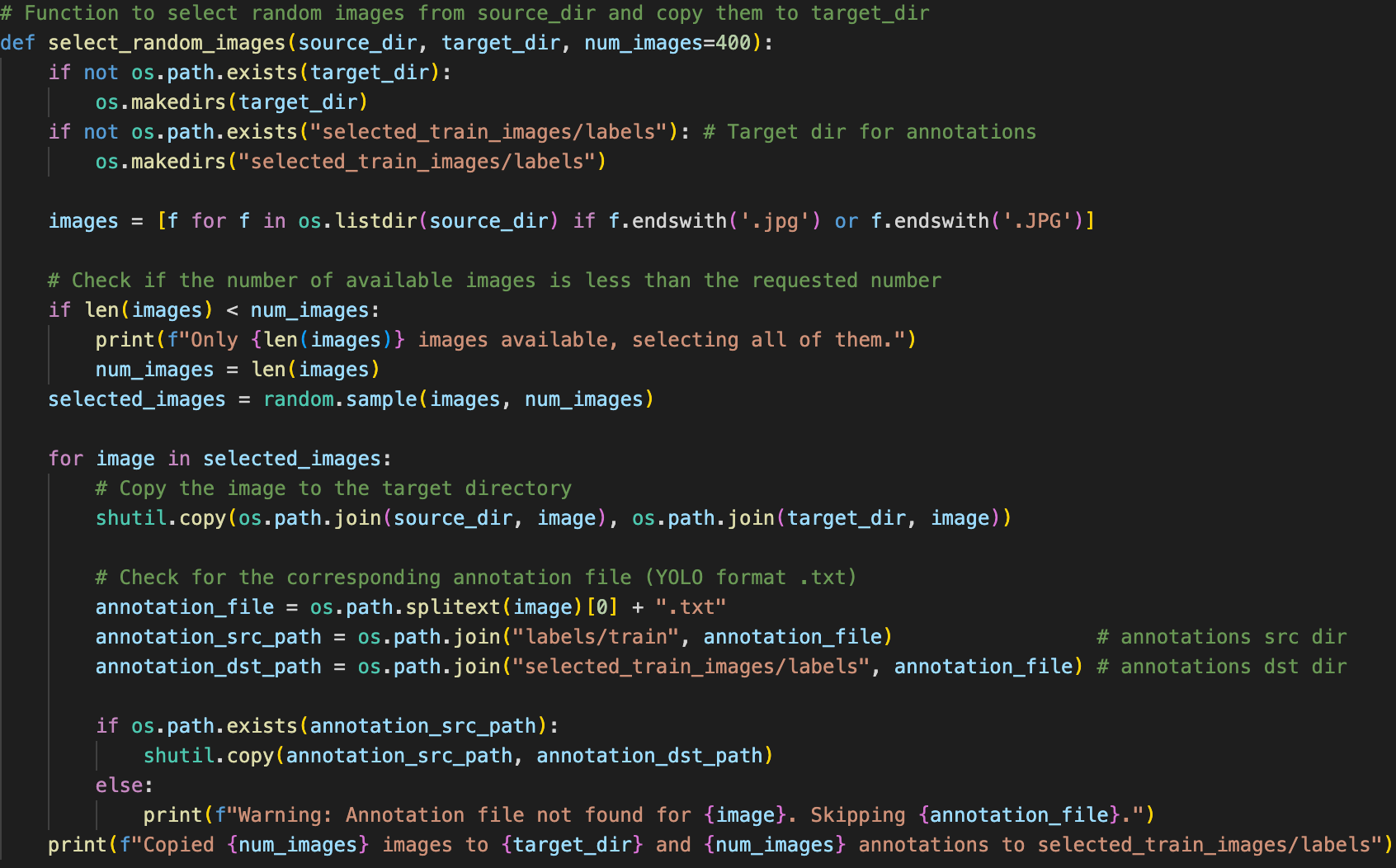
The following function converts the annotations to YOLO format and saves the output as .txt files for each image.



**STEP 2: Train the YOLO Model**

Once the dataset was converted to YOLO format, the next step was to randomly select 400 images from the training dataset and use them to train the YOLOv5 model.

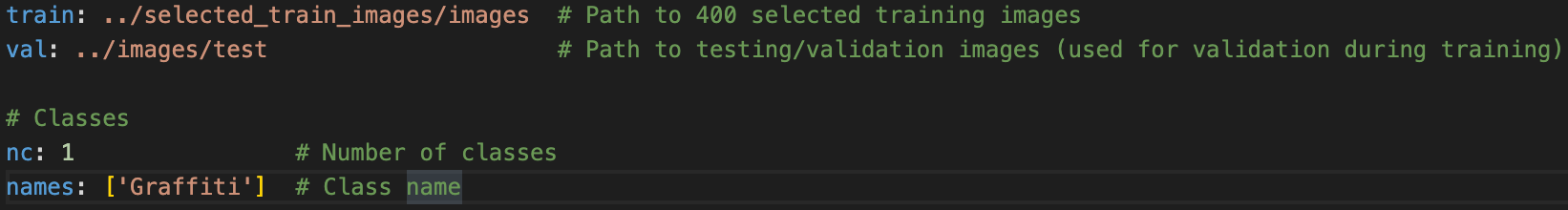
The select\_random\_images function randomly selects 400 images from the images/train/ directory and copies the corresponding annotations from labels/train/ to the selected\_train\_images/ folder.

**Training the YOLO Model**:

The following command was used to train the model with the selected 400 images:

python3 train.py --img 640 --batch 16 --epochs 50 --data data.yaml --weights yolov5m.pt –cache

Which is the bash script to train data using YOLOv5 train.py model, with 400 images, image size 640x640, 50 epochs and utilising the medium pre-trained model weights from yolov5m.pt

The configuration data.yaml file are shown as below:

The trained model was saved to runs/train/exp/weights/best.pt.

**EDA Evaluations:**

These are some of the EDA results to be evaluated from the training session with yolov5m.pt:

1. A blue squares with white text

   Description automatically generatedconfusion\_matrix.png
2. labels\_correlogram.jpg
3. A group of blue and white graphs

   Description automatically generated with medium confidenceA graph with a blue line

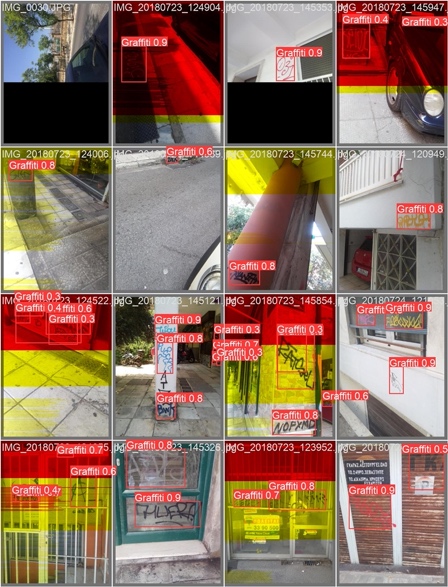
   Description automatically generatedA graph of a graph

   Description automatically generatedA graph with a blue line

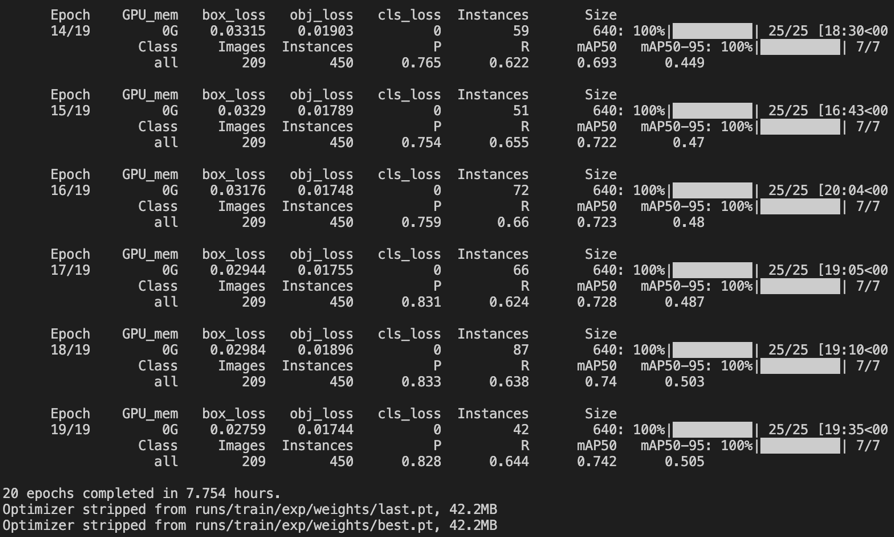
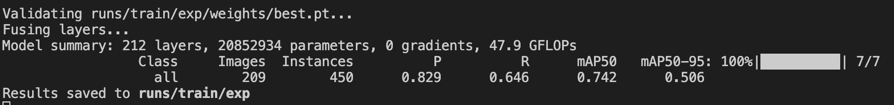
   Description automatically generatedA graph of a graph

   Description automatically generatedConfidence level - curves:
4. A graph of loss and loss

   Description automatically generated with medium confidenceResults:
5. A collage of images of street signs

   Description automatically generatedBatch files (train batch, value (test) batch labels and value (test) batch prediction)

**Performance**: The model performs well with a good balance between precision and recall, achieving around 73% in F1-score and 74.2% mean average precision (mAP). However, the recall tends to drop as the confidence increases, indicating that the model might miss some graffiti instances when it is more confident.

This training process took 7.754 hours from training with 20 epochs on a Macbook Pro M1 CPU.

**STEP 3: Compute IoU for Test Data and Evaluations**

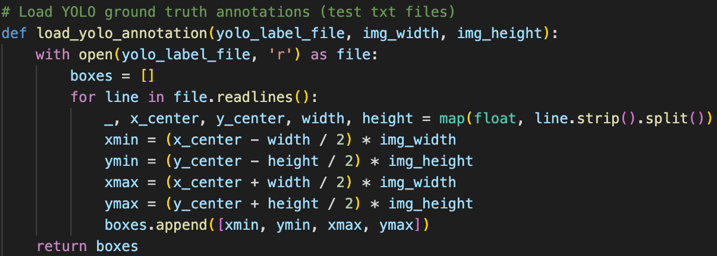
After training, we evaluated the model on 40 randomly sampled images from the test dataset images/test directory.. For each image, we computed the IoU (Intersection over Union) between the predicted and ground-truth bounding boxes. The results were saved in a CSV file (iou\_results\_iter\_{i}.csv, with ‘i’ to be the ID of the iteration), as well as the trained model saved (best.pt and last.pt models saved at directory runs/train/exp{n+1}/weights with n to be the iteration ID, starting from exp2).

The following function computes IoU and stores the results:

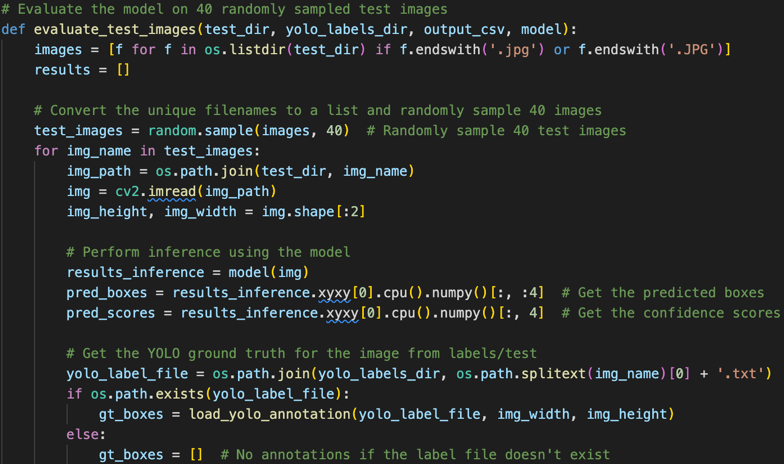
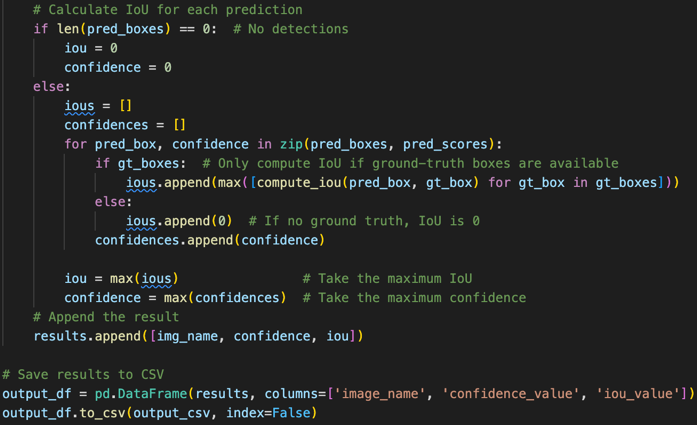
1. Compute IoU result:

The IoU is computed for each predicted bounding box against the ground truth boxes extracted from the YOLO annotations.

1. YOLO annotation loading:

The function load\_yolo\_annotation() reads YOLO annotation .txt files from the labels/test directory. These annotations are used to obtain the ground truth bounding boxes for each test images.

1. Random sampling and Evaluations:

Directly sample 40 images from the test image directory images/test and evaluate the model's predictions using the ground truth YOLO annotations from labels/test.

1. Iteration Trainings:

A computer screen shot of text

Description automatically generatedThe IoU results are saved in CSV files (iou\_results\_iter\_{i}.csv), where i is the iteration ID number. The evaluation process is repeated iteratively until expecting to have 80% of the images have an IoU > 90%. The final model’s result will be also saved as final\_model.csv.

1. Run time:

Average runtime approximately 7.5 hours with model iteration trained with 20 epochs, and 2.3 hours for training model with 10 epochs (totally 4 iterations trained with 20 epochs and 4 iterations trained with 10 epochs). These task are executed on a Macbook Pro M1 CPU.

1. Model selection:

Due to limitation on time resource and CPU health issue, unfortunately, I couldn’t iterate the training infinitely and eventually stopped at the 8th iteration, with the total runtime recorded for STEP 3 reached approximately 39 hours.

Hence, I decided to intervene and stop the iteration loop at the 8th iteration and decide to set the final training model to be the one with highest population of images seizing IoU value to be over 90%.

Taken from the record (can also be viewed at aw6results.rtf):

Iteration 1: 67.5% images have IoU > 90%

Iteration 2: 50.0% images have IoU > 90%

Iteration 3: 60.0% images have IoU > 90%

Iteration 4: 55.0% images have IoU > 90%

Iteration 5: 50.0% images have IoU > 90%

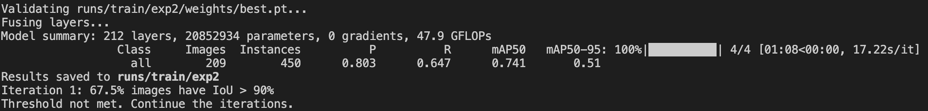
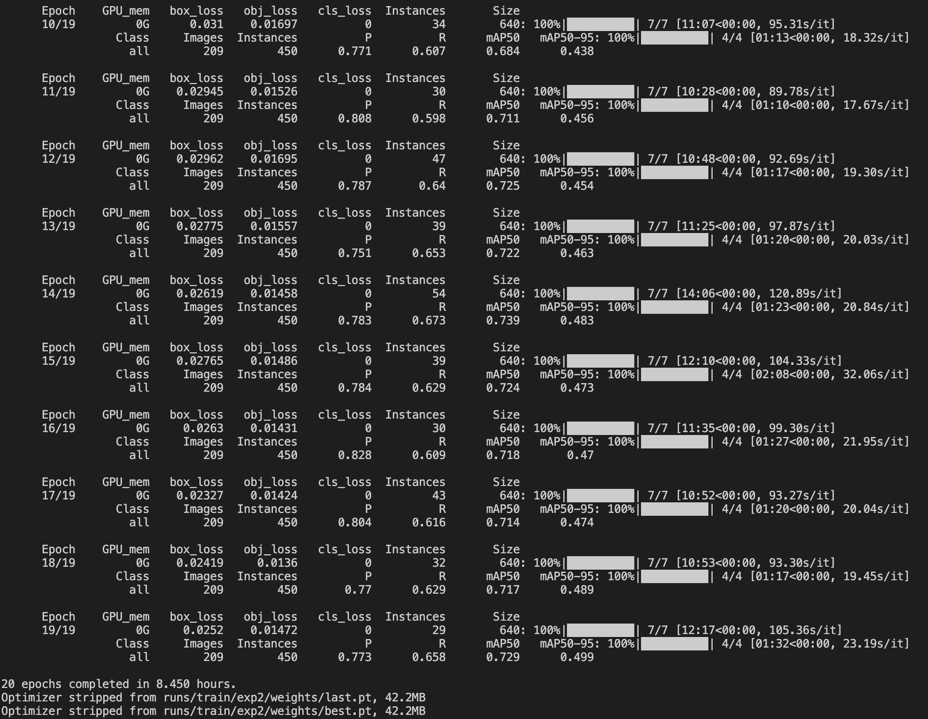
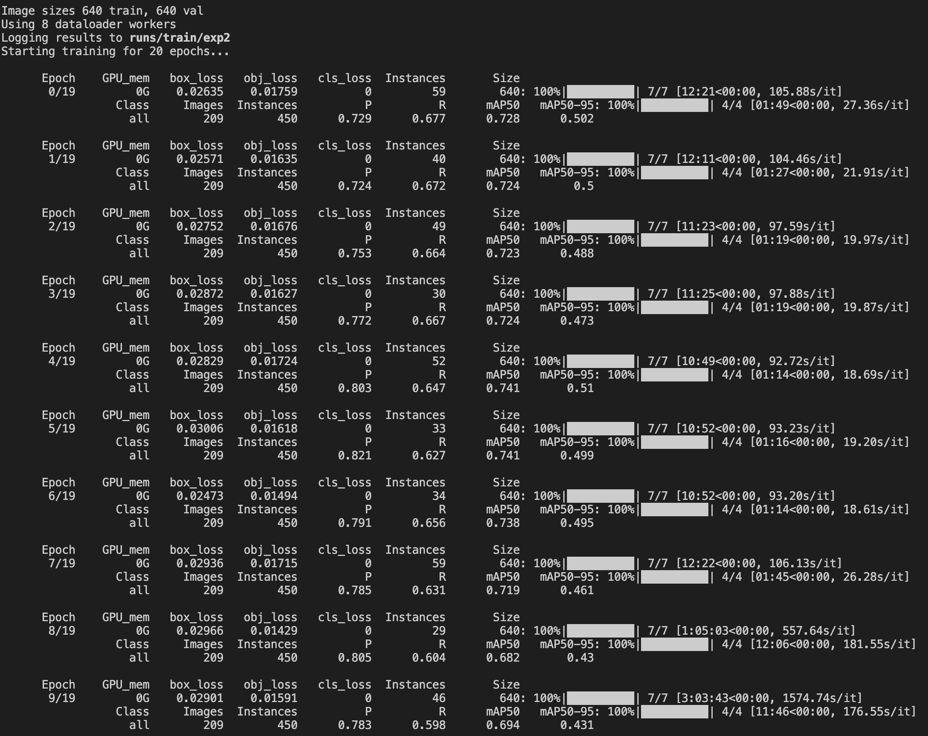
Iteration 6: 47.5% images have IoU > 90%

Iteration 7: 52.5% images have IoU > 90%

Iteration 8: 57.5% images have IoU > 90%

We conclude the highest population value reached with the first iteration, with “67.5% images have IoU > 90%”, hence, we will utilise its weight model (runs/train/exp2/weights/best.pt) as the final model to be used for detecting graffiti with live video records.

Iteration 1 is trained with 20 epochs, its results are saved as iou\_results\_iter\_1.csv. These are the terminal output and epochs’ evaluations from its training:



**STEP 4: Real-Time Graffiti Detection on Video**

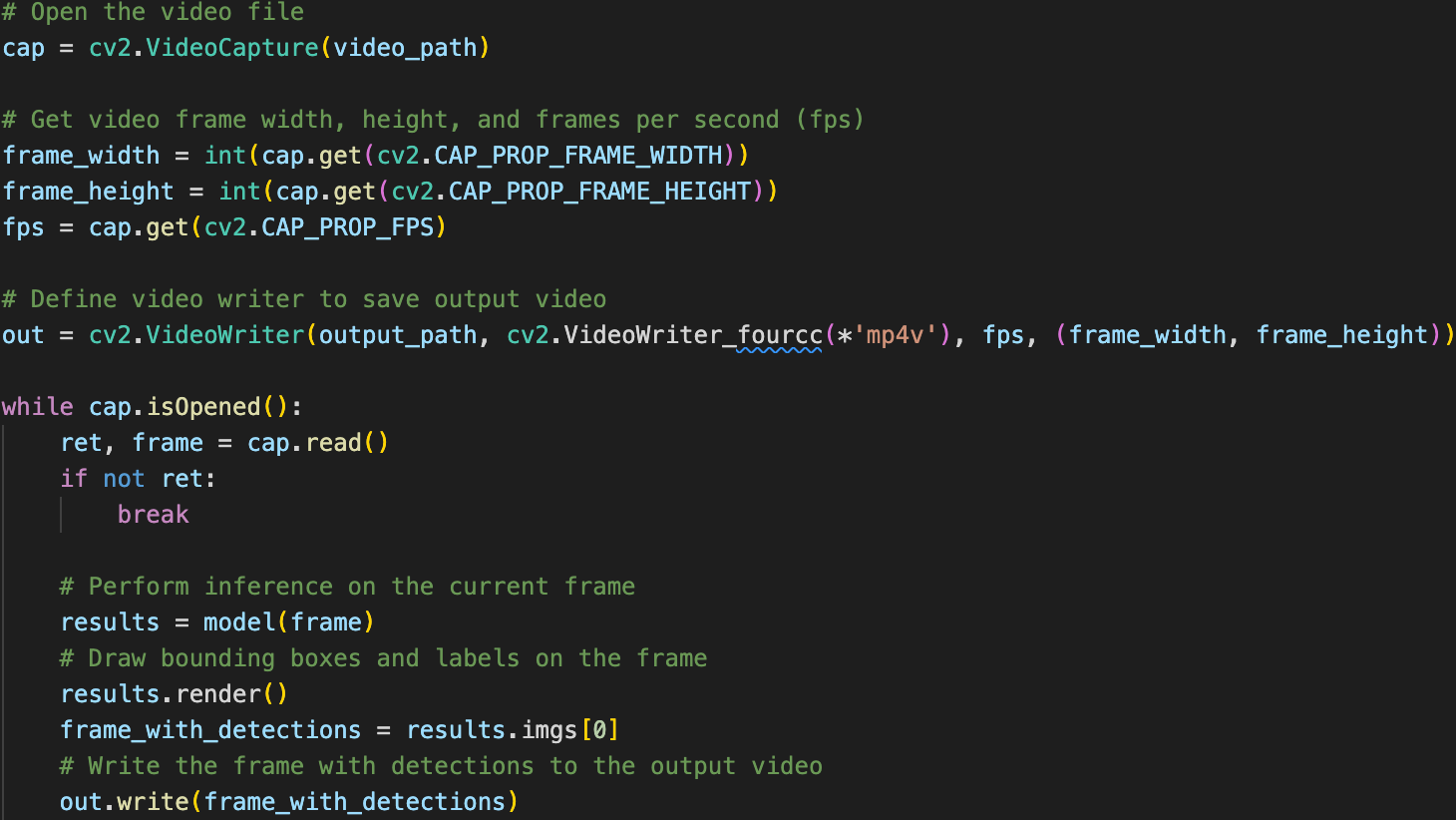
1. **Loading the Model**:

We load the trained YOLOv5 model from the path runs/train/exp2/weights/best.pt, which is the best model from **Iteration 1**.

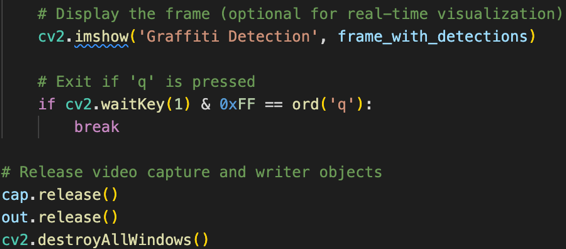
1. **Video Processing**:

* The script processes videos in the live\_video directory. It reads each frame from the video, performs inference using the YOLO model, and draws bounding boxes and labels for graffiti detections.
* The frames with detections are written to an output video in the output\_videos directory.

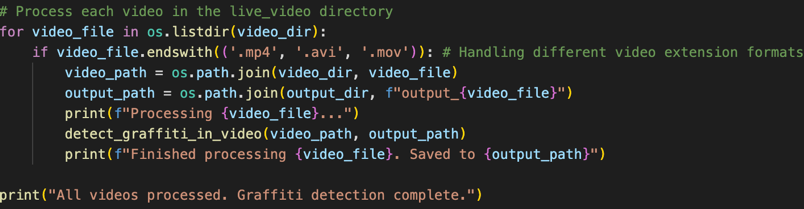
1. **Video Capture and Output**:

* The cv2.VideoCapture object reads each video, and the cv2.VideoWriter is used to save the processed video with detected graffiti.
* For each frame, graffiti detections are rendered using the results.render() method.

1. **Real-Time Display** (Optional):

The script shows the video with graffiti detections in real-time using cv2.imshow(). You can quit this by pressing 'q'. These can be commented-out if you do not wish to show graffiti detections in real-time.

1. **Output**:

Processed videos with graffiti detections are saved in the output\_videos folder.

The 4 output videos are named as output\_{video\_file} with video\_file is the original name of the video source from ‘live\_video’ directory.

A graffiti on a wall

Description automatically generatedExample of how graffities are notated on live video with bounding boxes and confidence score (e.g., a score of 0.86 means the model is 86% confident that the detected object is graffiti), here is 1 frame of the video ‘output\_3413463-hd\_1920\_1080\_30fps.mp4’, with graffiti detections notated.

**Appendix**

The Python script file for this assessment are named as ‘a5.py’.

All files and directory setups are stored and can be accessed via this Google Drive URL (allow accessibility for everyone with the link).

<https://drive.google.com/drive/folders/1_1Pqe_hjmVr31KUgTKUuSiwht_ADvLS4?usp=sharing>