Name: Trung Kien Nguyen

Student ID: 104053642

Studio: 1- 3

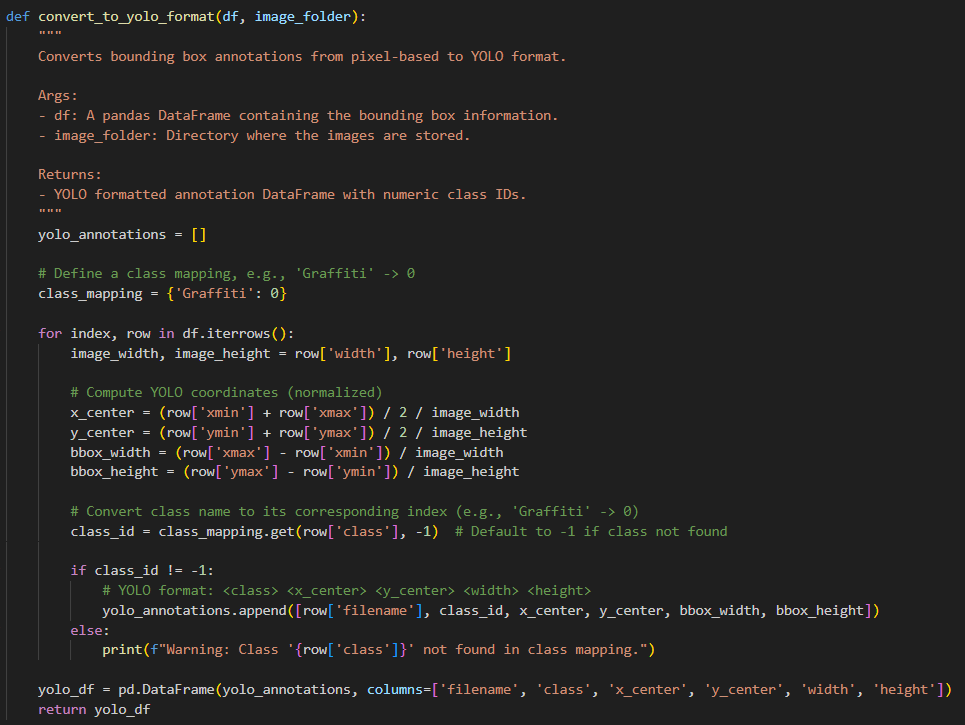
PORTFOLIO – WEEK 6

There is only a single implementation file for the whole Week 6 Portfolio, which can be retrieved at the following link:

# Step 1: Convert to YOLO annotation format

Firstly, I have extract the 3238357.zip file, and save the extracted bounding\_boxes and images folders in the dataset folder. This is also where I put all sampled data files in the following tasks, which can be retrieved via the link:

I have then created a method to convert to YOLO annotation format:



Basically, this method of convert\_to\_yolo\_format take a dataframe as a parameter, which is the bounding box dataset (train/test). This file has features of:

* filename: The name of the image file.
* class: The class label for the object (e.g., 'Graffiti').
* xmin, ymin: The coordinates of the top-left corner of the bounding box.
* xmax, ymax: The coordinates of the bottom-right corner of the bounding box.
* width, height: The width and height of the image in pixels.

The method then iteratively traverse through all rows in this dataset, I create the corresponding YOLO annotation format for each row, including some components:

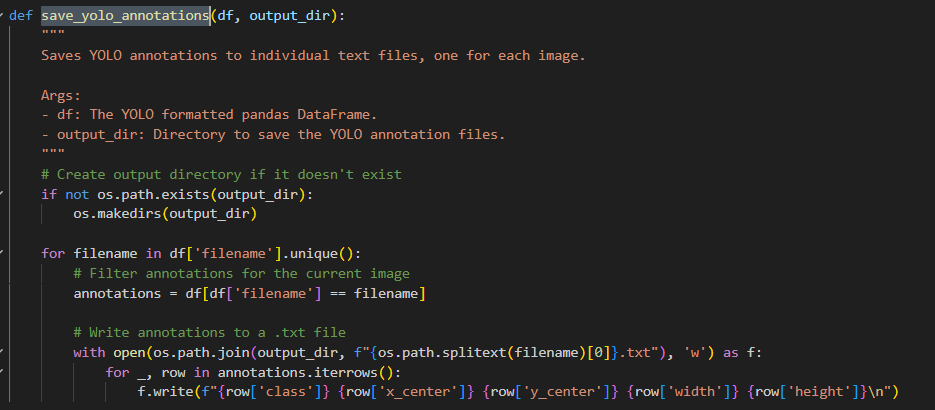
* The x\_center and y\_center coordinates of the center of the bounding box.
* Size of bounding box: The width is computed by subtracting xmin from xmax, and the height by subtracting ymin from ymax. These values are also normalized by dividing by the image dimensions.

Also, the id of class is converted from the corresponding class namem which is Graffiti, using the class\_mapping dictionary

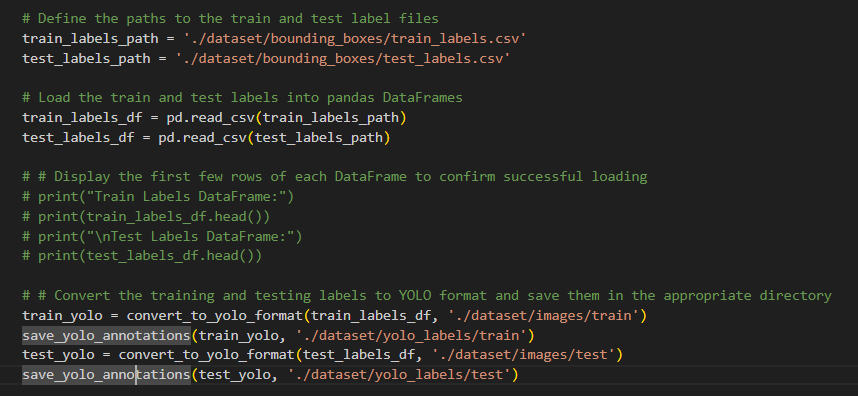
Finally, after all the iterations, the DataFrame output is returned with the following features:

* filename: The name of the image.
* class: The class ID of the object.
* x\_center: The normalized x-coordinate of the bounding box center.
* y\_center: The normalized y-coordinate of the bounding box center.
* width: The normalized width of the bounding box.
* height: The normalized height of the bounding box.

Next, I then create a method to save annotation files, based on the formatted yolo annotation dataframe, which is named “save\_yolo\_annotations”:

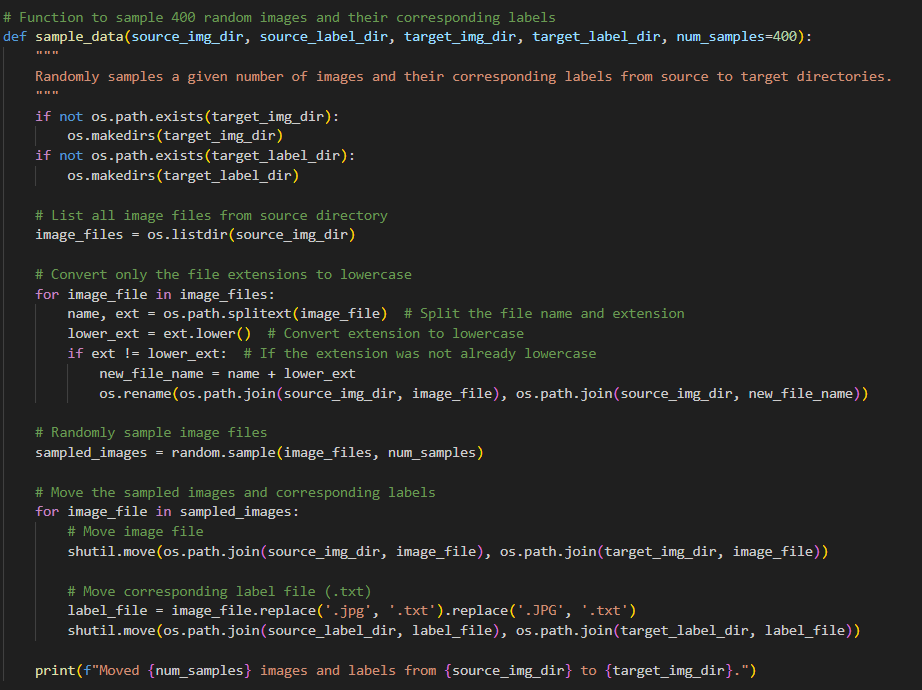


The annotation text files will be saved in a folder located in the path “output\_dir”. For each row in the annotation dataset, the method will save corresponding text file with this format: <class> <x\_center> <y\_center> <width> <height>. The name of the text file will be based on the name of the corresponding image file.

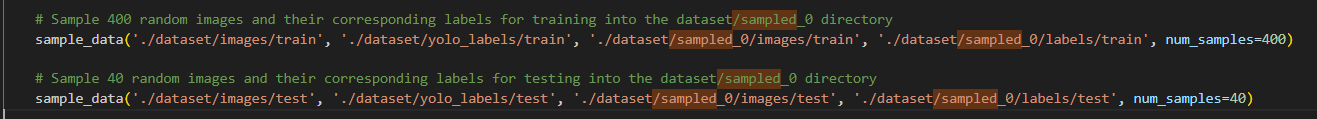


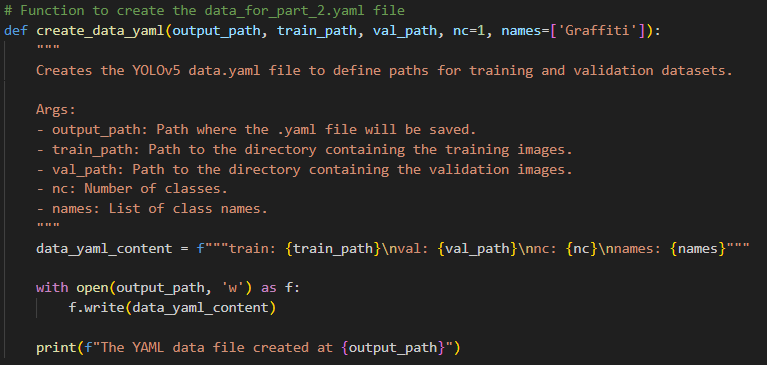
In the main method, I run the save\_yolo\_annotations method for both test and train dataset, saving the corresponding labels text files in the yolo\_labels folder. Here is the link to this folder:

# Step 2: Sample some images and perform training process

First, I have created a method to sample the number of dataset needed for the both training and testing process, which is sample\_data: 

This method will check the images and corresponding lables in source\_img\_dir and source\_label\_dir respectively, taking a number of “num\_samples” samples, then move them into target\_img\_dir and target\_label\_dir. Initially, I take 400 pairs of images-labels from the training dataset (dataset/images/train and dataset/yolo\_labels/train) and 40 pairs from the testing dataset (dataset/images/test and dataset/yolo\_labels/test), gather them and move to the dataset/sample\_0 folder (link: )

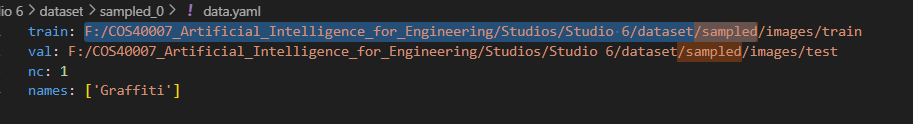


Next, I implemented a method to generate appropriate YAML data file for the training/testing process: 

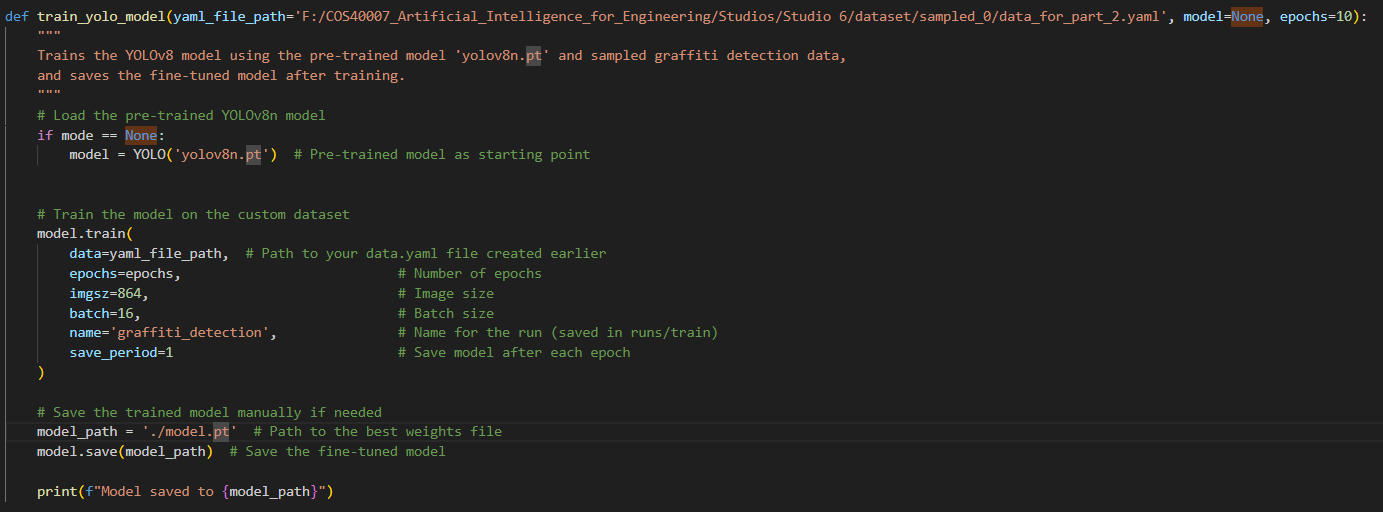
The data.yaml file is saved into the sample\_0 folder, created with the path to sampled\_0/images/train and sample\_0/images/test.



Here is the output of that YAML file



and the link to retrieve this file:

After that, I created a separate method for the training purpose, which is train\_yolo\_model: 

This take the path for the saved yaml file as a paramater, then use its content for retrieving the training data. The model use for this training will be passed as a parameter. By default, which is also at this step, it will be null, and if so, it uses the pre-trained yolov8n.pt model (link: )



Additionally, I have included some parameters for the training process, including:

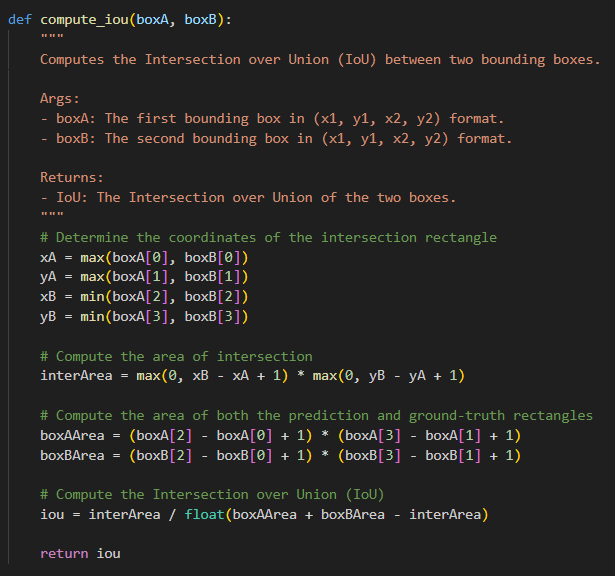
* Number of epochs: 50
* Batch size: 16
* Image size: 864. This is determined manually as the average image size in the dataset is about 720x960

The trained model is saved as the name model.pt. This will be reused in the following steps of further training and testing. Due to its massive size, I will provide the link of the final model (after Step 4)

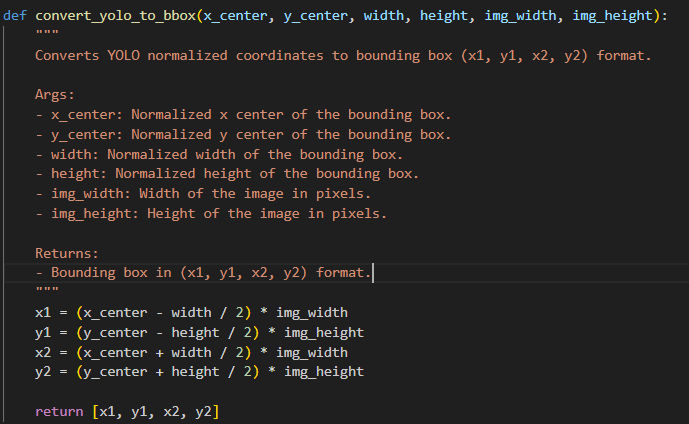
# Step 3: Test and evaluate the trained model

Previously while sampling the 400 images-labels pairs for the training process, I have also sampling 40 pairs for the testing process as well, the output is saved in the folder dataset/images/test and dataset/yolo\_labels/test respectively.

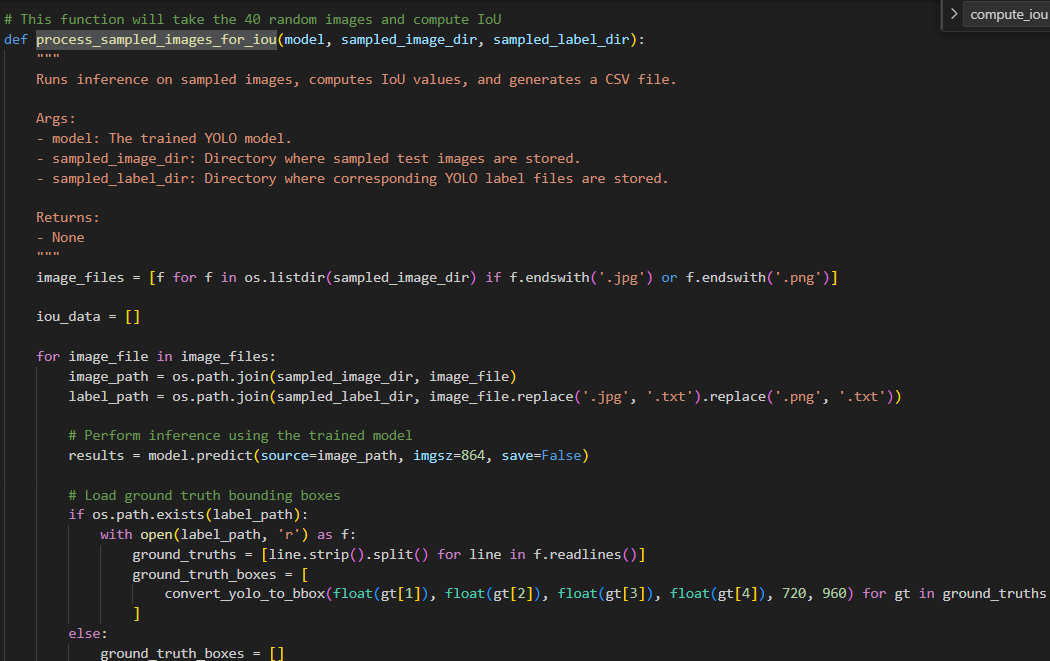
Now, I created a method to compute the IoU value, given the parameters of two bounding box

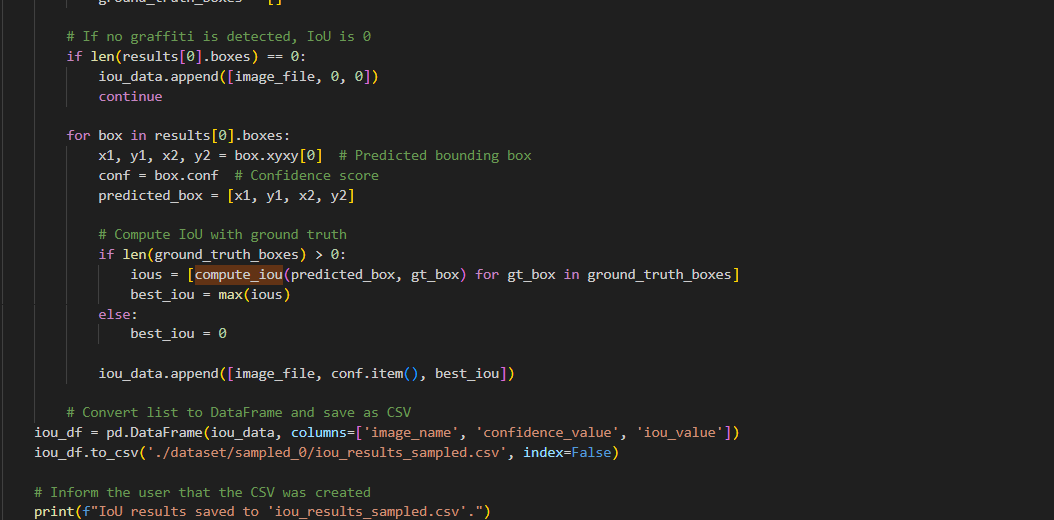


Next, I created a method to convert read data from yolo annotation text file back to a bounding box format, which is [x1, y1, x2, y2]

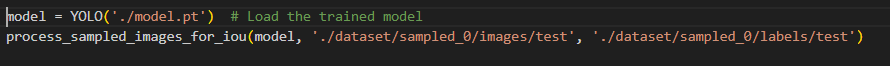


Then is to evaluate the model with the testing dataset (dataset/sampled\_0/images/test and dataset/sampled\_0/labels/test) via the process\_sampled\_images\_for\_iou method:





This method will use the trained “model.pt” model to evaluate:



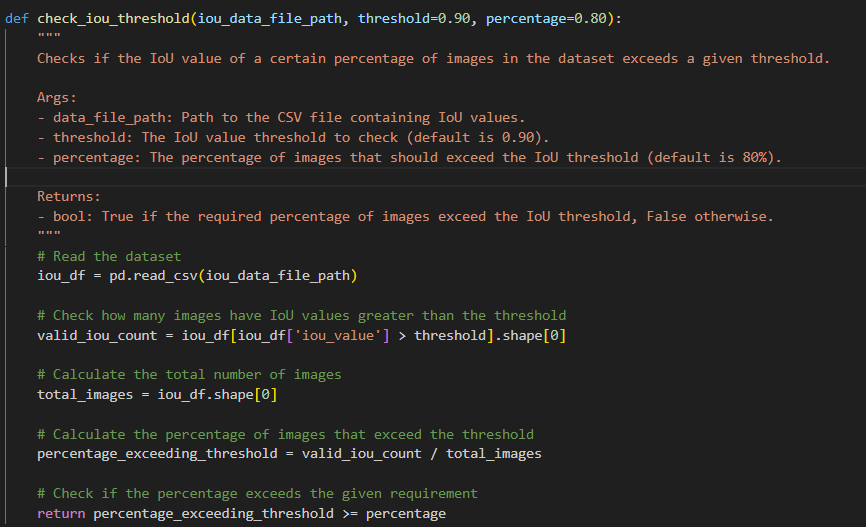
Here is the breakdown of evaluation process:

* Load and Predict: It iterate through each image in the specified directory, uses the YOLO model to predict bounding boxes, and fetches the corresponding ground-truth boxes.
* IoU Calculation: For each prediction, the function calculates the IoU with the ground-truth boxes using the compute\_iou() function. If no graffiti is detected in the image, the IoU is set to 0.
* Store Results: The image name, confidence score, and best IoU value for each prediction are stored in a list.
* Save to CSV: The result (image name, confidence, IoU) is converted into a DataFrame and saved to a CSV file (iou\_results\_sampled.csv). You can retrieved this file in the link:

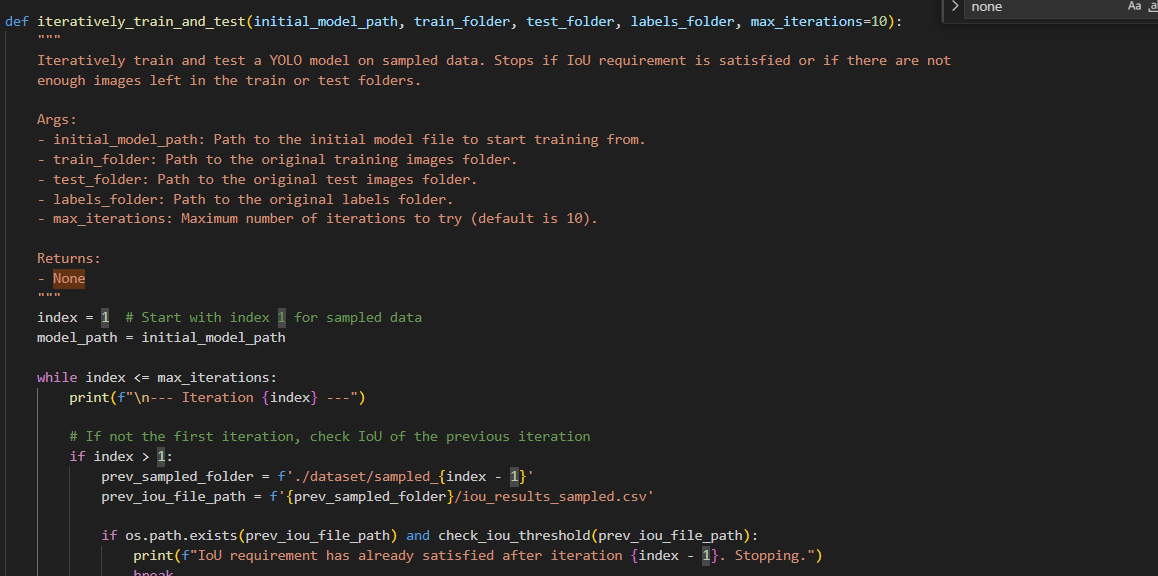
# Step 4: Iterative training, testing and evaluating

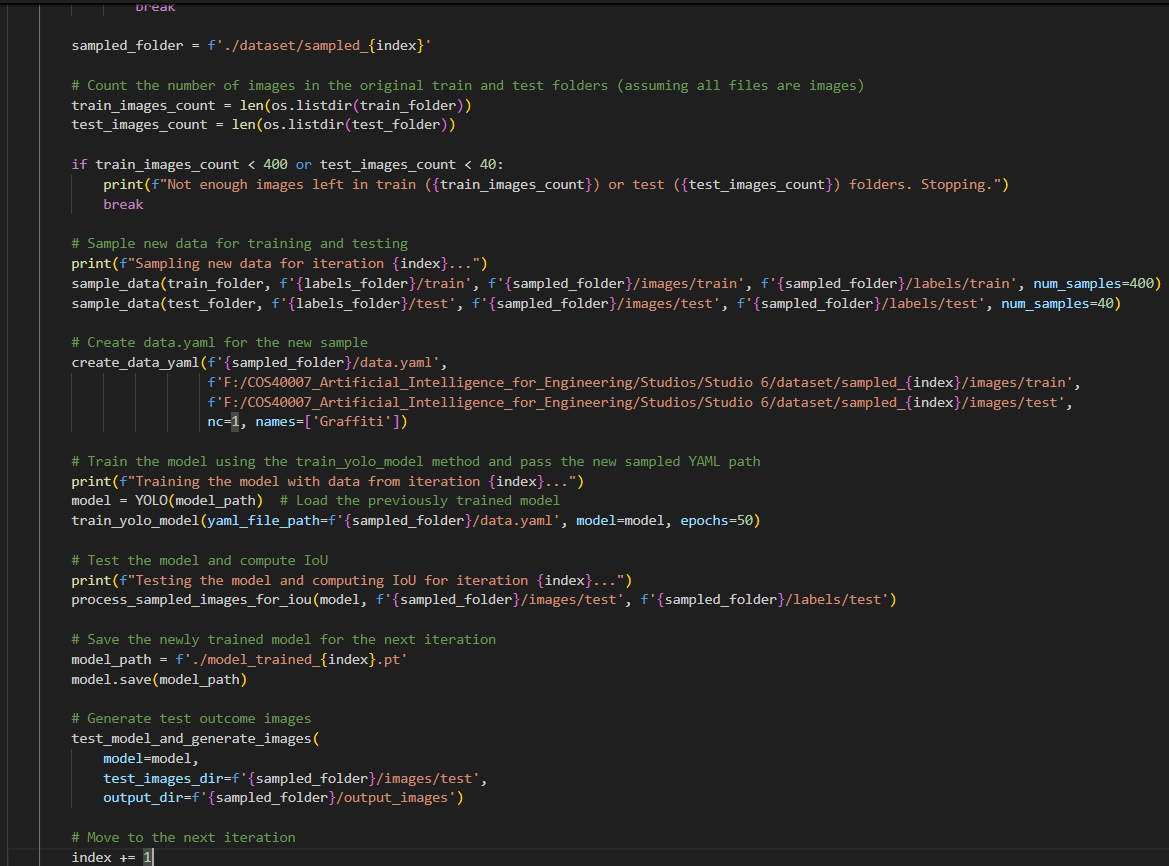
I have to train and test the model iteratively using a sampled set of 400 training and 40 test images (and corresponding labels) in this step.

Firstly, I created a method to check the iteration termination condition, which is callled “check\_iou\_threshold”. This method taking the IoU result data, check if there are more than a certain ratio of test samples has succesfully surpass the IoU threshold. According to the requirement, the ratio threshold is 80% and the IoU threshold is 90%:



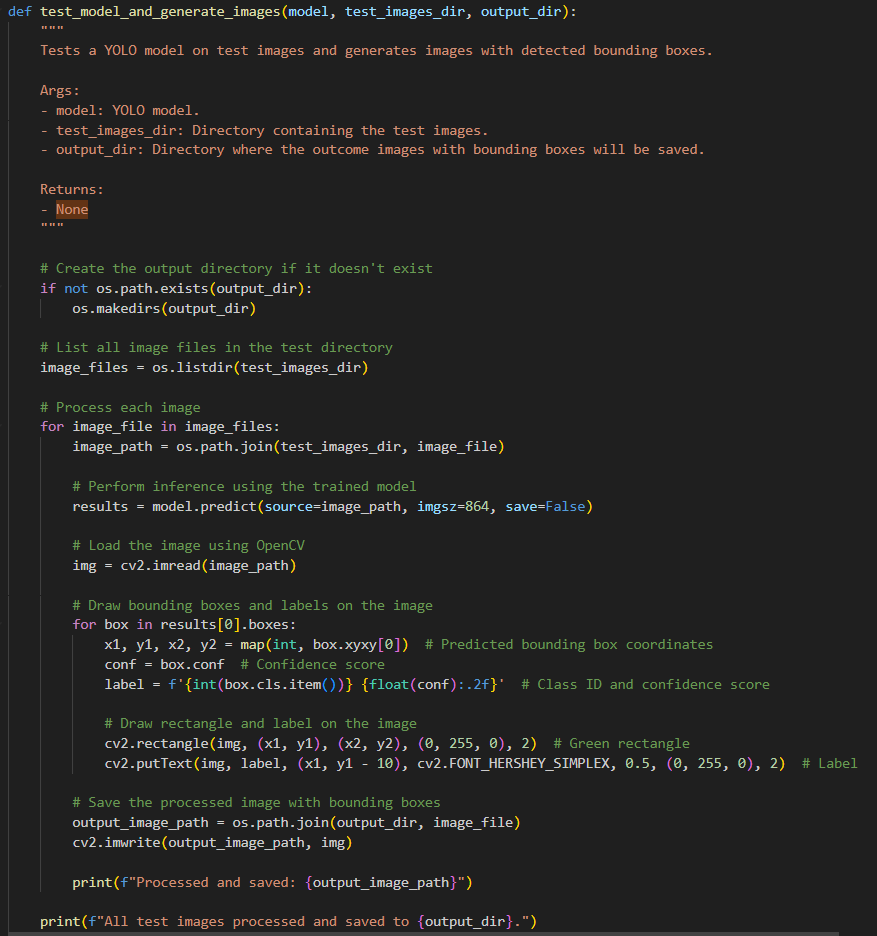
Then, I create a method named “iteratively\_train\_and\_test” for the iteration requirement with a “while” format.



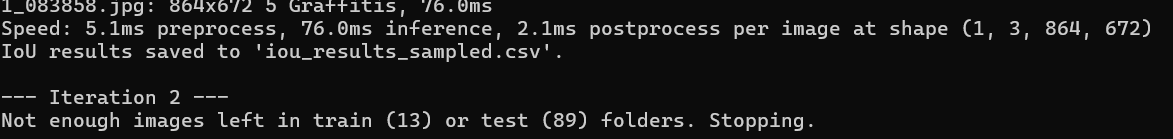


Basically, for each iteration:

* There is an index determine the number of iteration that has been done. This index is also used to name the sampled data folder (“sampled\_{index}”), which is used to stored each set of 400 training samples and 40 testing samples, the corresponding YAML data file, and the IoU Evaluation result CSV file. In the above step, I have saved every output to the “sampled\_0” folder, so that the method can start from the index of 1 smoothly.
* At the beginning of the iteration, I will use “check\_iou\_threshold” method to check if the IoU result from the last sampled folder (sampled\_{index - 1}) has satisfied the termination condition.
* Then, I also check in the original dataset folders (dataset/images and dataset/yolo\_labels) if there are still enough 400 samples for training and 40 samples for testing
* If at least one of the above condition is met, then the loop will end
* Next, I use the “sample\_data” method to sample randomly a set of 400 training samples and 40 testing samples, saved it in “sampled\_{index}” folder. The corresponding YAML data file will also be generated in this folder
* Then, I train the model from the previous iteration, or the model after step 3 (“initail\_model\_path”) if it is the first iteration, using the sampled training set
* After training, I continue to use “process\_sampled\_images\_for\_iou” method to generate the evaluation IoU CSV file, preparing for the next iteration
* The model after each will be saved in a separated file named “model\_trained\_{index}.pt” in the sampled folder, and will be used for the next iteration
* Finally, I implement the method to generate test outcome images for each iteration, using the package cv2. The output images is saved in the sampled folder of the correspoding iteration as well.



After the loops end, since there is only a total of 813 images in the training dataset, and 400 had already been taken in Step 2, there is only one iteration run in this loop:



In the Iteration 1, this is some outcome test images with detected bounding boxes as follow:

You can find more such outcome images in the iteration 1 via the link below.

Here is the link to:

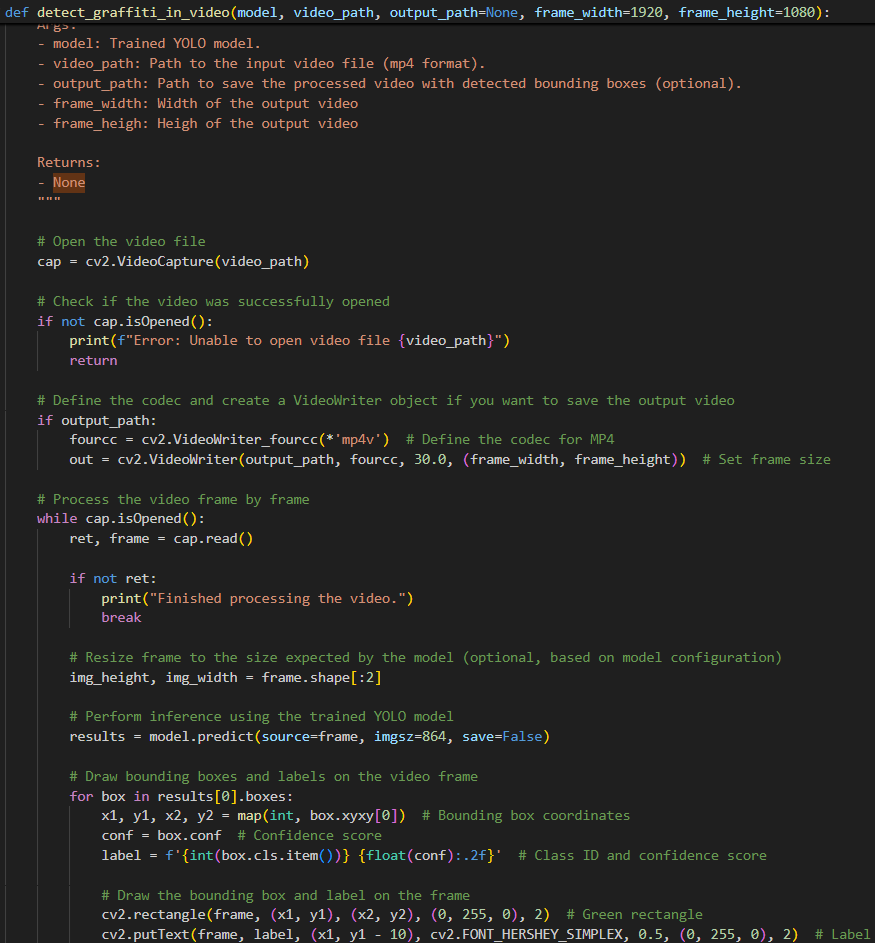
* sampled\_1 folder:
* Evaluation IoU CSV file for Iteration 1:
* Model after Iteration 1, which is also the final model:
* Test outcome images using that model:

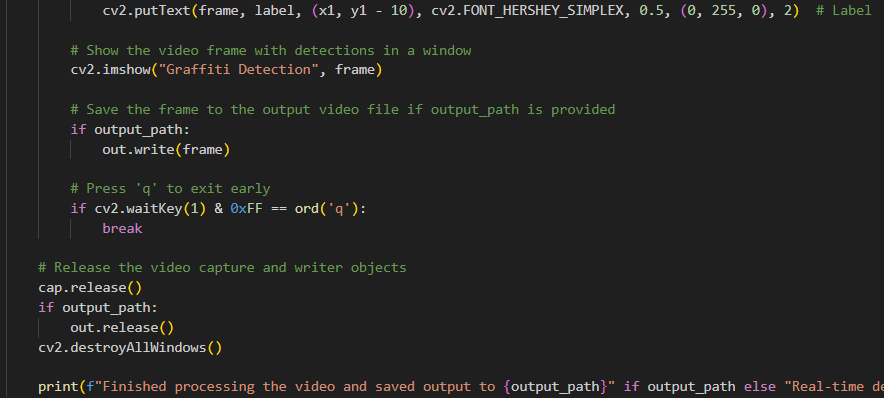
# Step 5: Test with real-time video data

I have downloaded videos from the 5 given links:

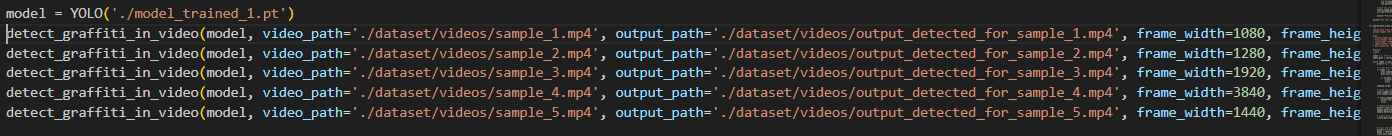
* <https://www.pexels.com/video/a-door-with-graffiti-on-it-is-shown-4543511/>
* <https://www.pexels.com/video/busy-street-footage-854181/>
* <https://www.pexels.com/video/graffiti-painted-on-the-train-station-wall-3413463/>
* <https://www.pexels.com/photo/two-men-posing-outside-teleboutique-storefront-28587032/> (The fourth given link lead to a page of a photographer named Pat Whelen (<https://www.pexels.com/@pat-whelen-2913248/>), in which I couldn’t find any videos about graffiti, so I had to find the new one)
* <https://www.pexels.com/video/a-man-writing-on-a-wall-with-a-marker-9724130/>

These are saved in the folder of dataset/videos, respectively named as sample\_1.mp4, sample\_2.mp4, sample\_3.mp4, sample\_4.mp4, sample\_5.mp4

Then I created a method to generate video with detected graffiti, using cv2 package (VideoCapture class) and the model after Step 4:   




Here is the use of that method:



The 5 output videos with detection of graffiti are also located in the dataset/videos folder, which can be retrieved from these 5 corresponding links: