Student name: Nguyen Anh Tuan
Student ID: 103805949

Swinburne University of Technology

COS40007-Artificial Intelligence for Engineering

Portfolio 6

Nguyen Anh Tuan - 103805949

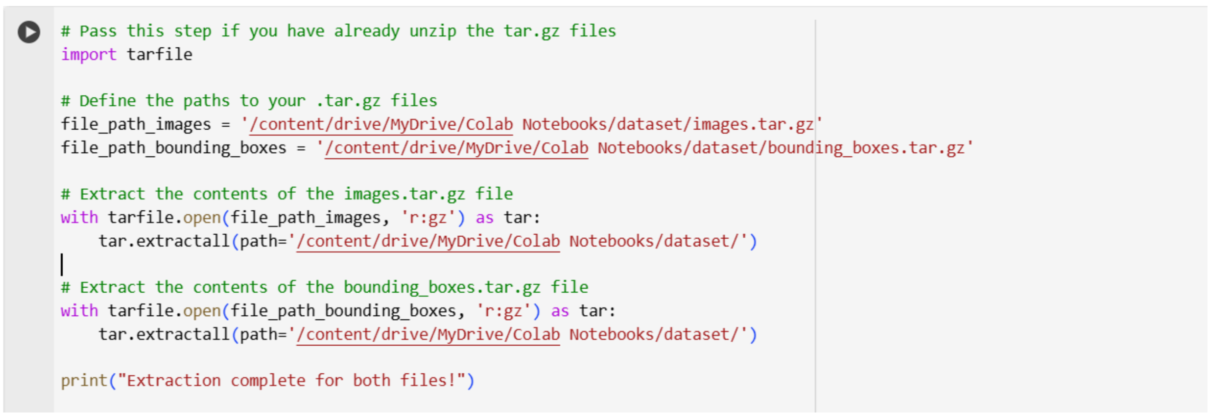
Studio class: Studio 1-1

Hanoi, Vietnam

Student name: Nguyen Anh Tuan
Student ID: 103805949

1. Write a function to convert given annotation format in training labels to YOLO annotation format

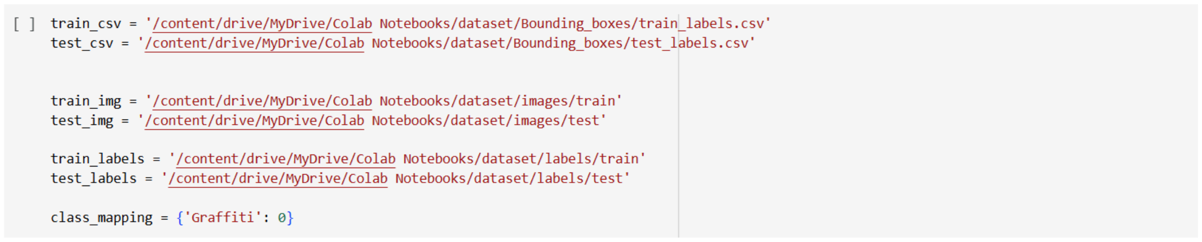
To begin with, I unzip .tar.gz files and put them in the “ /content/drive/MyDrive/Colab Notebooks/dataset/” path.



```
# Pass this step if you have already unzip the tar.gz files
import tarfile

# Define the paths to your .tar.gz files
file_path_images = '/content/drive/MyDrive/Colab Notebooks/dataset/images.tar.gz'
file_path_bounding_boxes = '/content/drive/MyDrive/Colab Notebooks/dataset/bounding_boxes.tar.gz'

# Extract the contents of the images.tar.gz file
with tarfile.open(file_path_images, 'r:gz') as tar:
    tar.extractall(path='/content/drive/MyDrive/Colab Notebooks/dataset/')

# Extract the contents of the bounding_boxes.tar.gz file
with tarfile.open(file_path_bounding_boxes, 'r:gz') as tar:
    tar.extractall(path='/content/drive/MyDrive/Colab Notebooks/dataset/')

print("Extraction complete for both files!")
```

Then, I define file paths for the training and testing data. These include train_csv and test_csv, which point to the CSV files containing the annotations for training and testing, respectively. It also specifies train_img and test_img directories where the images for training and testing are stored. Additionally, train_labels and test_labels are the output directories where the converted YOLO annotation files will be saved.

The code includes a class mapping dictionary, class_mapping, which maps the object class names (such as 'Graffiti') to numeric class IDs. This mapping is necessary because YOLO uses numeric class IDs rather than names. In this case, the class 'Graffiti' is mapped to ID 0, which ensures that the annotations are in the correct format for YOLO training.

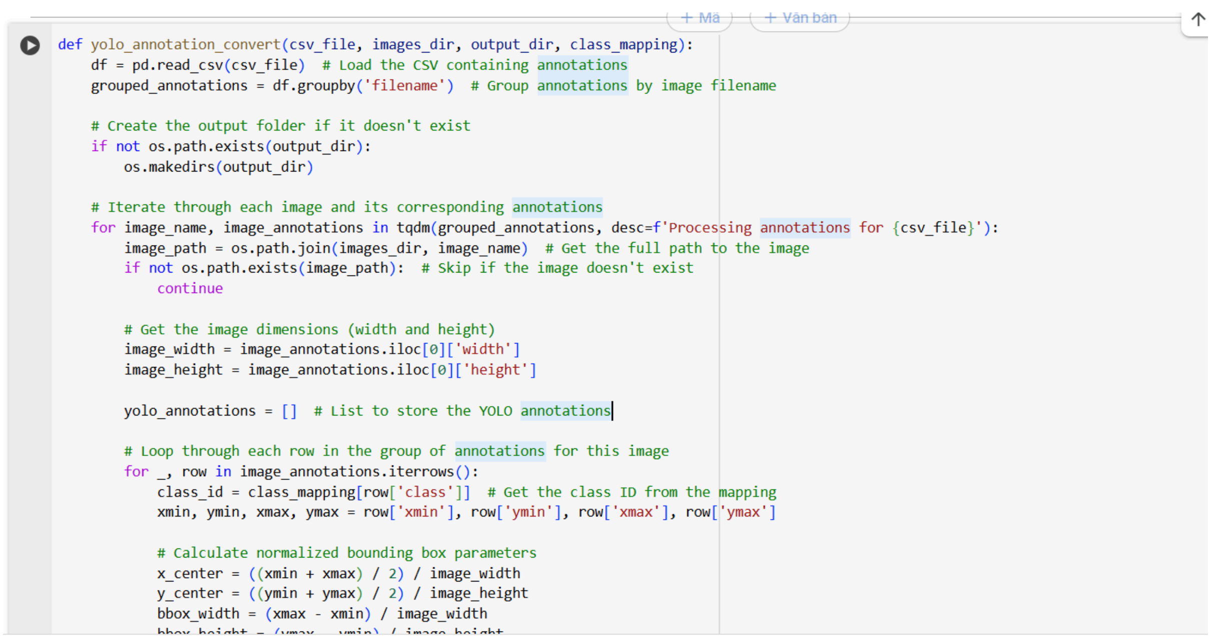


```
train_csv = '/content/drive/MyDrive/Colab Notebooks/dataset/Bounding_boxes/train_labels.csv'
test_csv = '/content/drive/MyDrive/Colab Notebooks/dataset/Bounding_boxes/test_labels.csv'


train_img = '/content/drive/MyDrive/Colab Notebooks/dataset/images/train'
test_img = '/content/drive/MyDrive/Colab Notebooks/dataset/images/test'

train_labels = '/content/drive/MyDrive/Colab Notebooks/dataset/labels/train'
test_labels = '/content/drive/MyDrive/Colab Notebooks/dataset/labels/test'

class_mapping = {'Graffiti': 0}
```

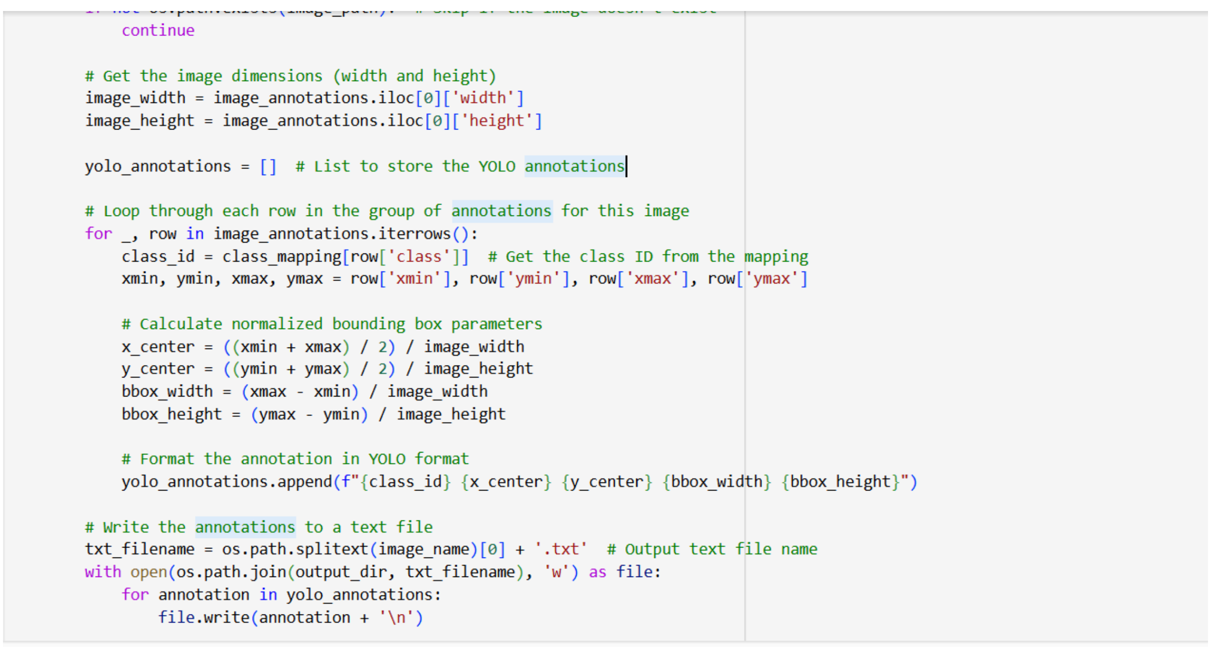
The function yolo_annotation_convert processes each image by reading its annotations from the CSV, calculates the normalized bounding box coordinates based on the image size, and writes the YOLO annotations into a .txt file. This file is saved in the specified output directory.

Student name: Nguyen Anh Tuan
Student ID: 103805949

The function is called for both the training and testing datasets, generating YOLO-compatible annotations for model training.

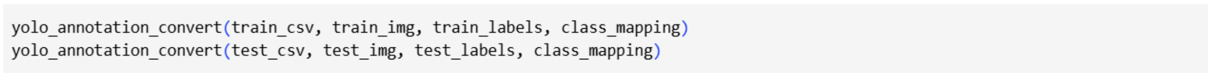


```python
def yolo_annotation_convert(csv_file, images_dir, output_dir, class_mapping):
    df = pd.read_csv(csv_file)  # Load the CSV containing annotations
    grouped_annotations = df.groupby('filename')  # Group annotations by image filename

    # Create the output folder if it doesn't exist
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Iterate through each image and its corresponding annotations
    for image_name, image_annotations in tqdm(grouped_annotations, desc=f'Processing annotations for {csv_file}'):
        image_path = os.path.join(images_dir, image_name)  # Get the full path to the image
        if not os.path.exists(image_path):  # Skip if the image doesn't exist
            continue

        # Get the image dimensions (width and height)
        image_width = image_annotations.iloc[0]['width']
        image_height = image_annotations.iloc[0]['height']

        yolo_annotations = []  # List to store the YOLO annotations

        # Loop through each row in the group of annotations for this image
        for _, row in image_annotations.iterrows():
            class_id = class_mapping[row['class']]  # Get the class ID from the mapping
            xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']

            # Calculate normalized bounding box parameters
            x_center = ((xmin + xmax) / 2) / image_width
            y_center = ((ymin + ymax) / 2) / image_height
            bbox_width = (xmax - xmin) / image_width
            bbox_height = (ymax - ymin) / image_height
```



```python
        if not os.path.exists(image_path):  # Skip if the image doesn't exist
            continue

        # Get the image dimensions (width and height)
        image_width = image_annotations.iloc[0]['width']
        image_height = image_annotations.iloc[0]['height']

        yolo_annotations = []  # List to store the YOLO annotations

        # Loop through each row in the group of annotations for this image
        for _, row in image_annotations.iterrows():
            class_id = class_mapping[row['class']]  # Get the class ID from the mapping
            xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']

            # Calculate normalized bounding box parameters
            x_center = ((xmin + xmax) / 2) / image_width
            y_center = ((ymin + ymax) / 2) / image_height
            bbox_width = (xmax - xmin) / image_width
            bbox_height = (ymax - ymin) / image_height
```

```python
            # Format the annotation in YOLO format
            yolo_annotations.append(f"{class_id} {x_center} {y_center} {bbox_width} {bbox_height}")

        # Write the annotations to a text file
        txt_filename = os.path.splitext(image_name)[0] + '.txt'  # Output text file name
        with open(os.path.join(output_dir, txt_filename), 'w') as file:
            for annotation in yolo_annotations:
                file.write(annotation + '\n')
```

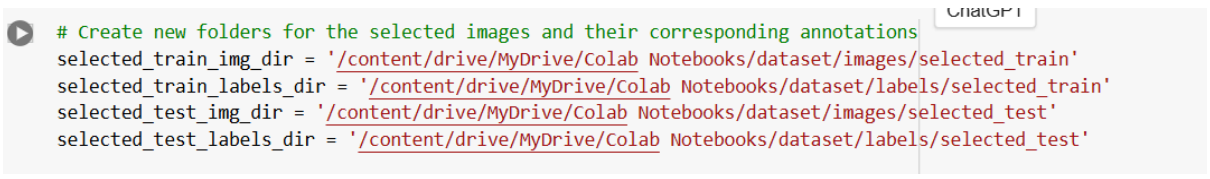


```python
yolo_annotation_convert(train_csv, train_img, train_labels, class_mapping)
yolo_annotation_convert(test_csv, test_img, test_labels, class_mapping)
```

Student name: Nguyen Anh Tuan
Student ID: 103805949

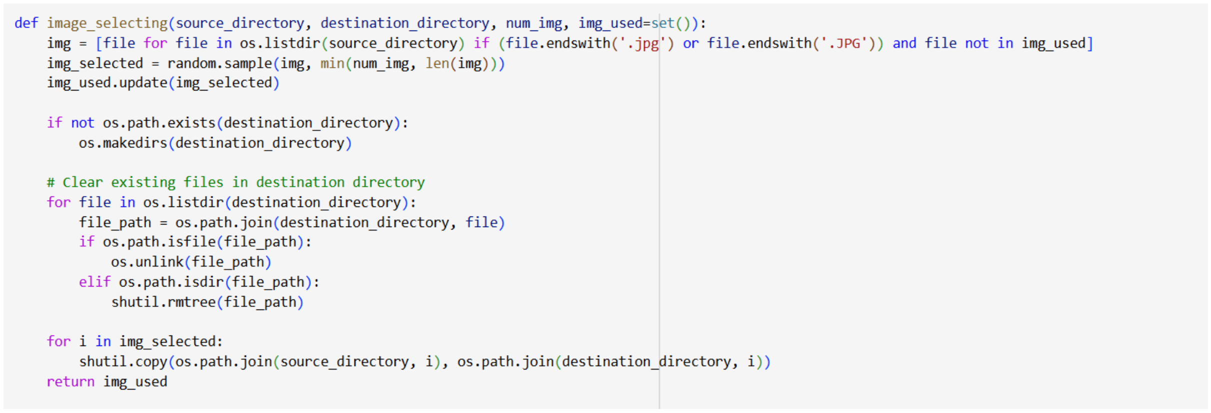
2. Train and create YOLO model by randomly taking 400 images from the train data which can detect graffiti in the image

For this part, first, I define new directories where the selected images and their annotations will be stored: selected_train_img_dir, selected_train_labels_dir, selected_test_img_dir, and selected_test_labels_dir.



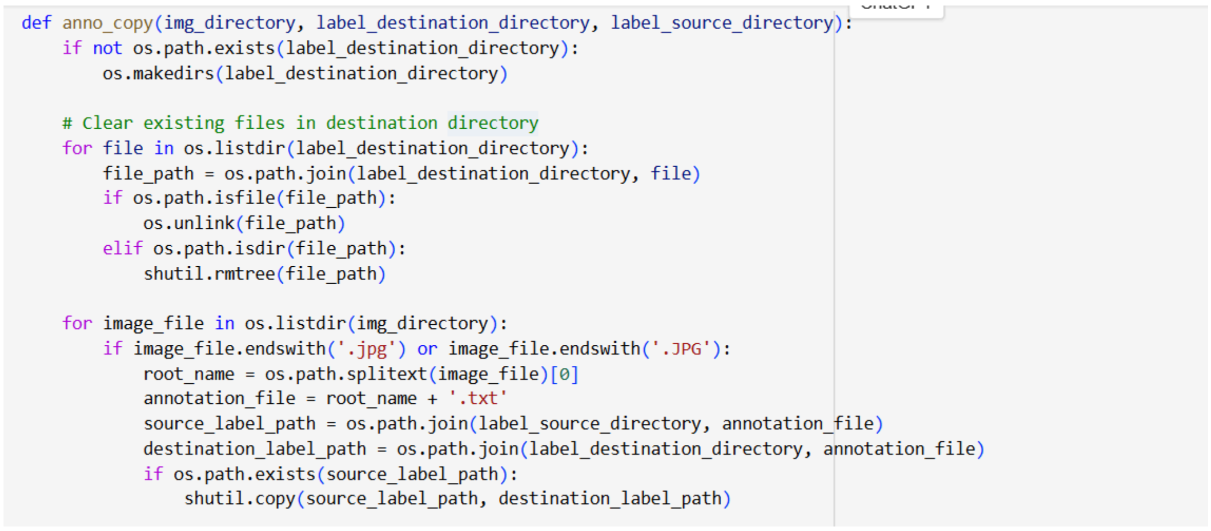
```python
# Create new folders for the selected images and their corresponding annotations
selected_train_img_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/images/selected_train'
selected_train_labels_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/labels/selected_train'
selected_test_img_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/images/selected_test'
selected_test_labels_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/labels/selected_test'
```

The function image_selecting selects a specified number of images from a source directory, ensuring that no image is selected more than once. It clears any existing files in the destination directory, creates it if it doesn't exist, and copies the selected images into the destination directory.



```python
def image_selecting(source_directory, destination_directory, num_img, img_used=set()):
    img = [file for file in os.listdir(source_directory) if (file.endswith('.jpg') or file.endswith('.JPG')) and file not in img_used]
    img_selected = random.sample(img, min(num_img, len(img)))
    img_used.update(img_selected)

    if not os.path.exists(destination_directory):
        os.makedirs(destination_directory)

    # Clear existing files in destination directory
    for file in os.listdir(destination_directory):
        file_path = os.path.join(destination_directory, file)
        if os.path.isfile(file_path):
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)

    for i in img_selected:
        shutil.copy(os.path.join(source_directory, i), os.path.join(destination_directory, i))
    return img_used
```

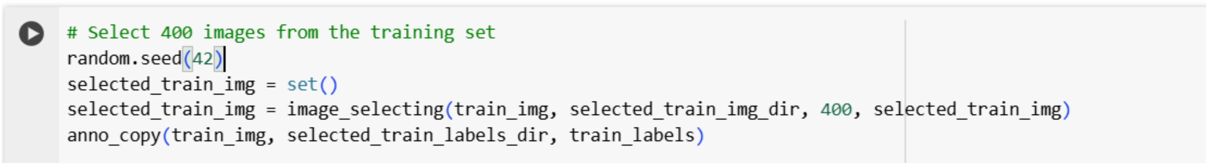
The function anno_copy copies the annotation files (in .txt format) from the source label directory to the destination label directory, ensuring that the corresponding annotations are transferred along with the selected images.



```python
def anno_copy(img_directory, label_destination_directory, label_source_directory):
    if not os.path.exists(label_destination_directory):
        os.makedirs(label_destination_directory)

    # Clear existing files in destination directory
    for file in os.listdir(label_destination_directory):
        file_path = os.path.join(label_destination_directory, file)
        if os.path.isfile(file_path):
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)

    for image_file in os.listdir(img_directory):
        if image_file.endswith('.jpg') or image_file.endswith('.JPG'):
            root_name = os.path.splitext(image_file)[0]
            annotation_file = root_name + '.txt'
            source_label_path = os.path.join(label_source_directory, annotation_file)
            destination_label_path = os.path.join(label_destination_directory, annotation_file)
            if os.path.exists(source_label_path):
                shutil.copy(source_label_path, destination_label_path)
```

Student name: Nguyen Anh Tuan
Student ID: 103805949

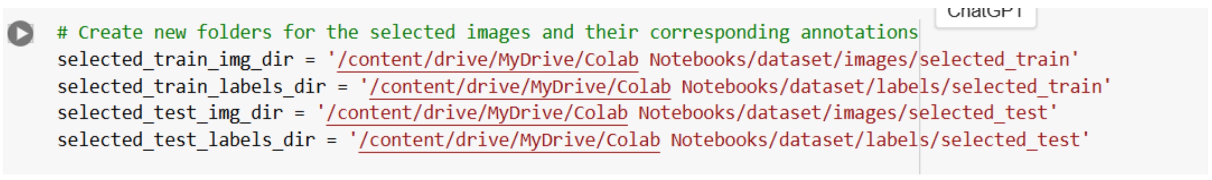
The code then selects 400 random images from the training set. It uses the anno_copy function to copy the relevant annotation files for the selected images into the appropriate directories.



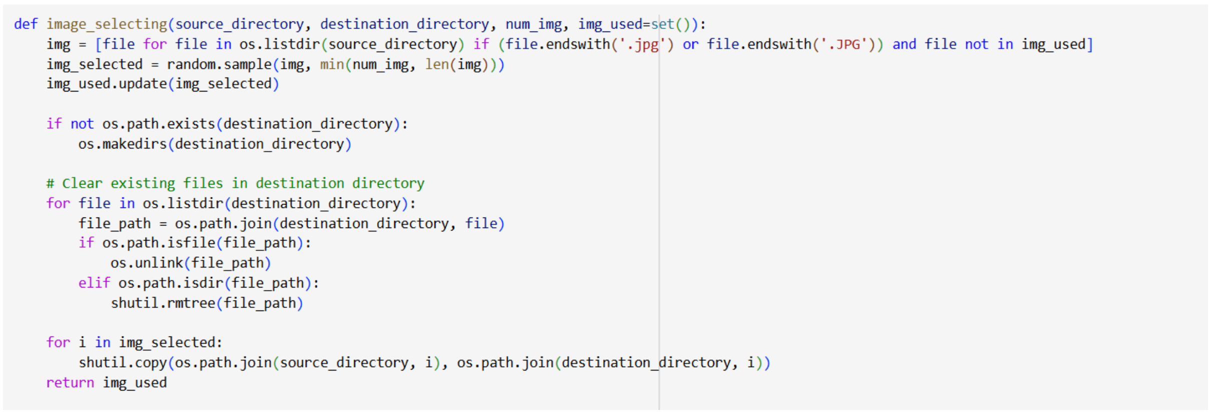
```
# Select 400 images from the training set
random.seed(42)
selected_train_img = set()
selected_train_img = image_selecting(train_img, selected_train_img_dir, 400, selected_train_img)
anno_copy(train_img, selected_train_labels_dir, train_labels)
```

3. Randomly take 40 images from test data and compute IoU for each and generate CSV file containing 3 columns [image_name, confidence value, IoU value]. If no graffiti is detected for an image then its IoU will be 0

To take 40 images from the test data, I also do the same thing similar to the previous part.

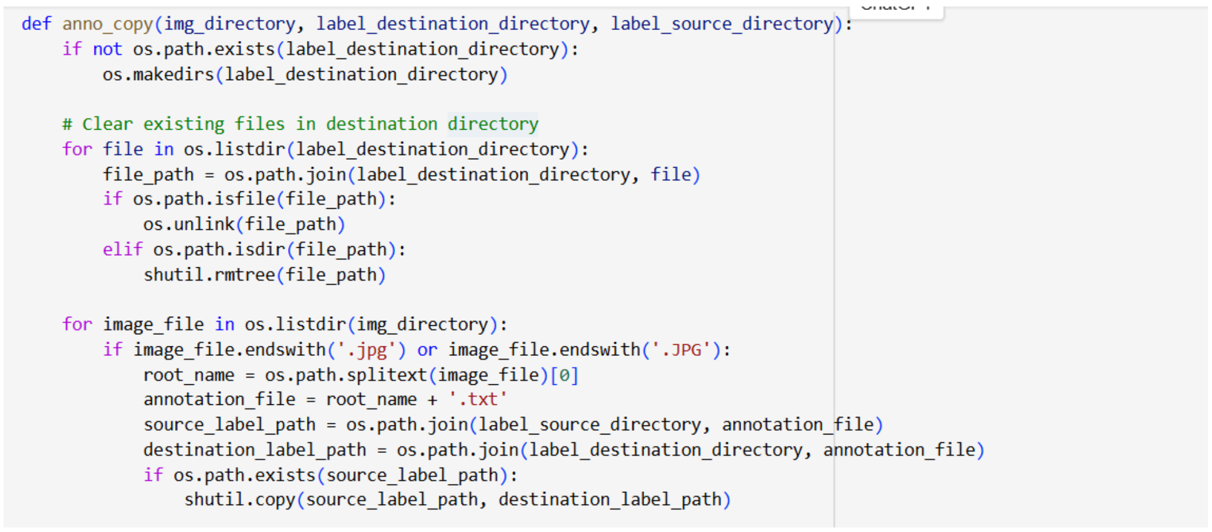


```
# Create new folders for the selected images and their corresponding annotations
selected_train_img_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/images/selected_train'
selected_train_labels_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/labels/selected_train'
selected_test_img_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/images/selected_test'
selected_test_labels_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/labels/selected_test'
```

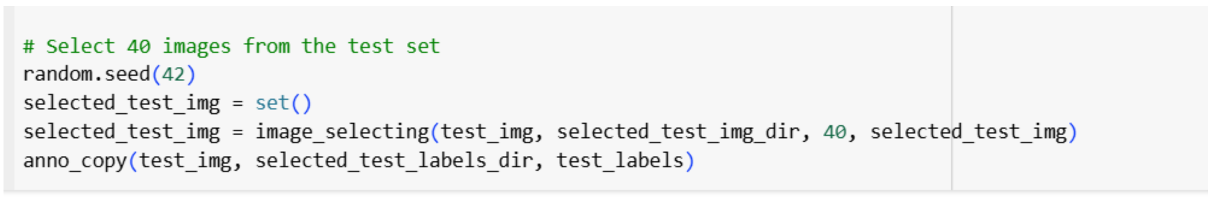


```
def image_selecting(source_directory, destination_directory, num_img, img_used=set()):
    img = [file for file in os.listdir(source_directory) if (file.endswith('.jpg') or file.endswith('.JPG')) and file not in img_used]
    img_selected = random.sample(img, min(num_img, len(img)))
    img_used.update(img_selected)

    if not os.path.exists(destination_directory):
        os.makedirs(destination_directory)

    # Clear existing files in destination directory
    for file in os.listdir(destination_directory):
        file_path = os.path.join(destination_directory, file)
        if os.path.isfile(file_path):
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)

    for i in img_selected:
        shutil.copy(os.path.join(source_directory, i), os.path.join(destination_directory, i))
    return img_used
```

Student name: Nguyen Anh Tuan
Student ID: 103805949

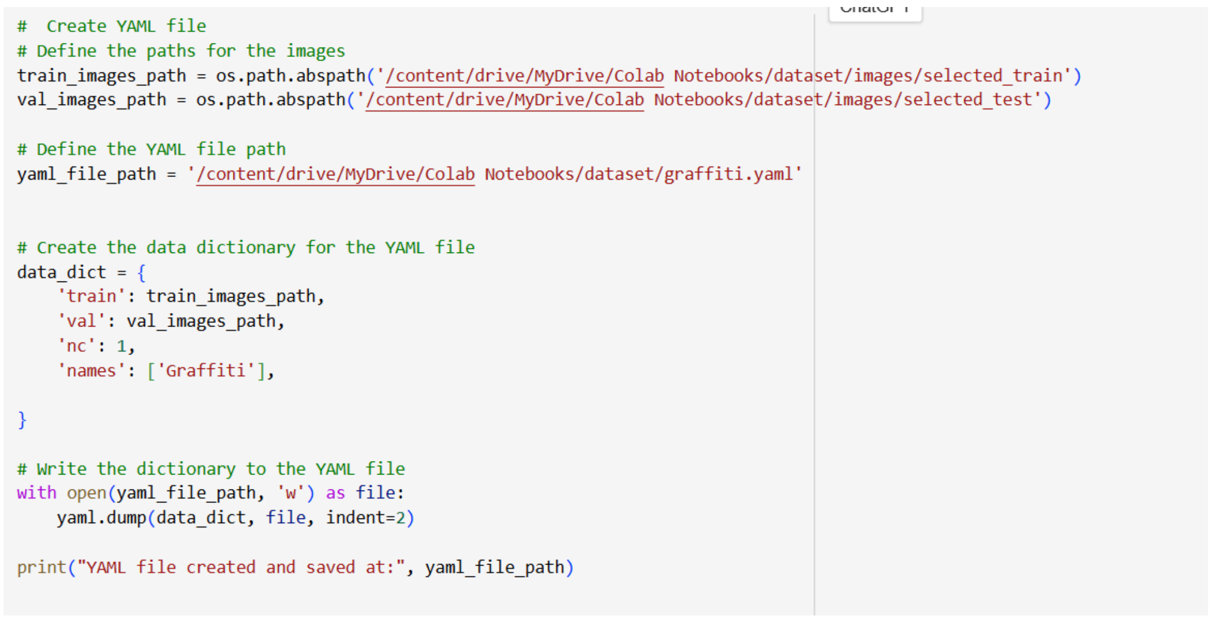


```python
def anno_copy(img_directory, label_destination_directory, label_source_directory):
    if not os.path.exists(label_destination_directory):
        os.makedirs(label_destination_directory)

    # Clear existing files in destination directory
    for file in os.listdir(label_destination_directory):
        file_path = os.path.join(label_destination_directory, file)
        if os.path.isfile(file_path):
            os.unlink(file_path)
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)

    for image_file in os.listdir(img_directory):
        if image_file.endswith('.jpg') or image_file.endswith('.JPG'):
            root_name = os.path.splitext(image_file)[0]
            annotation_file = root_name + '.txt'
            source_label_path = os.path.join(label_source_directory, annotation_file)
            destination_label_path = os.path.join(label_destination_directory, annotation_file)
            if os.path.exists(source_label_path):
                shutil.copy(source_label_path, destination_label_path)
```

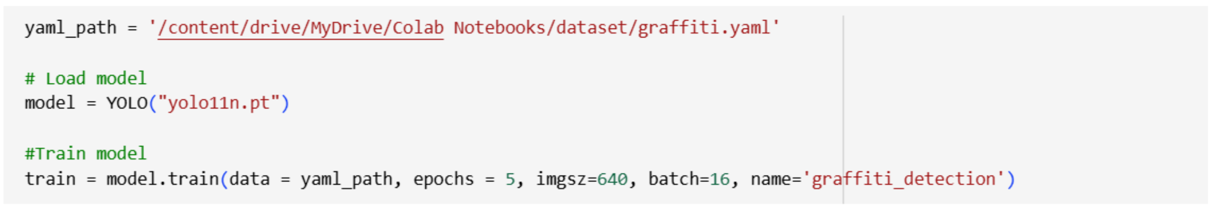


```python
# Select 40 images from the test set
random.seed(42)
selected_test_img = set()
selected_test_img = image_selecting(test_img, selected_test_img_dir, 40, selected_test_img)
anno_copy(test_img, selected_test_labels_dir, test_labels)
```

After that, I define paths for the training and validation image directories and the YAML file. It creates a dictionary with paths to the images, the number of classes, and class names, then writes this information to the YAML file. Next, a pre-trained YOLO model is loaded, and training begins with 5 epochs, a batch size of 16, and an image size of 640 pixels. The training results are saved under the name 'graffiti_detectio



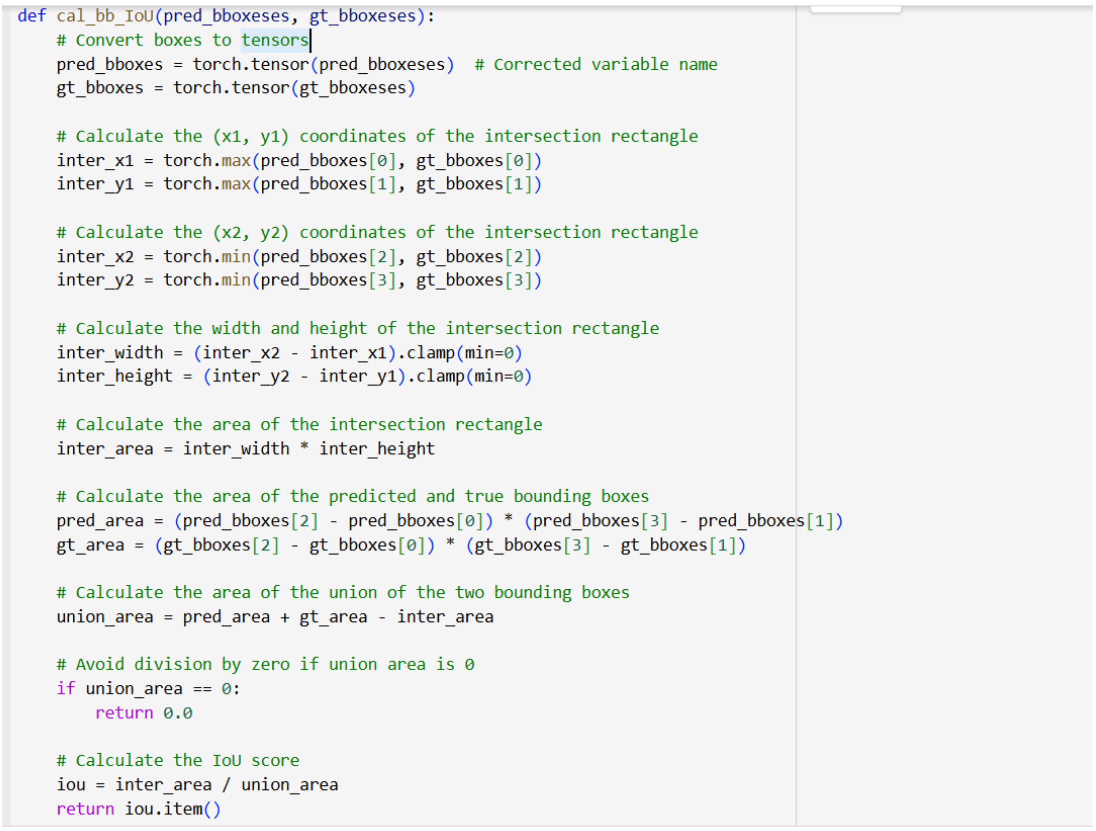
```python
#  Create YAML file
# Define the paths for the images
train_images_path = os.path.abspath('/content/drive/MyDrive/Colab Notebooks/dataset/images/selected_train')
val_images_path = os.path.abspath('/content/drive/MyDrive/Colab Notebooks/dataset/images/selected_test')

# Define the YAML file path
yaml_file_path = '/content/drive/MyDrive/Colab Notebooks/dataset/graffiti.yaml'


# Create the data dictionary for the YAML file
data_dict = {
    'train': train_images_path,
    'val': val_images_path,
    'nc': 1,
    'names': ['Graffiti'],

}

# Write the dictionary to the YAML file
with open(yaml_file_path, 'w') as file:
    yaml.dump(data_dict, file, indent=2)

print("YAML file created and saved at:", yaml_file_path)
```



```python
yaml_path = '/content/drive/MyDrive/Colab Notebooks/dataset/graffiti.yaml'

# Load model
model = YOLO("yolo11n.pt")

#Train model
train = model.train(data = yaml_path, epochs = 5, imgsz=640, batch=16, name='graffiti_detection')
```

Student name: Nguyen Anh Tuan
Student ID: 103805949

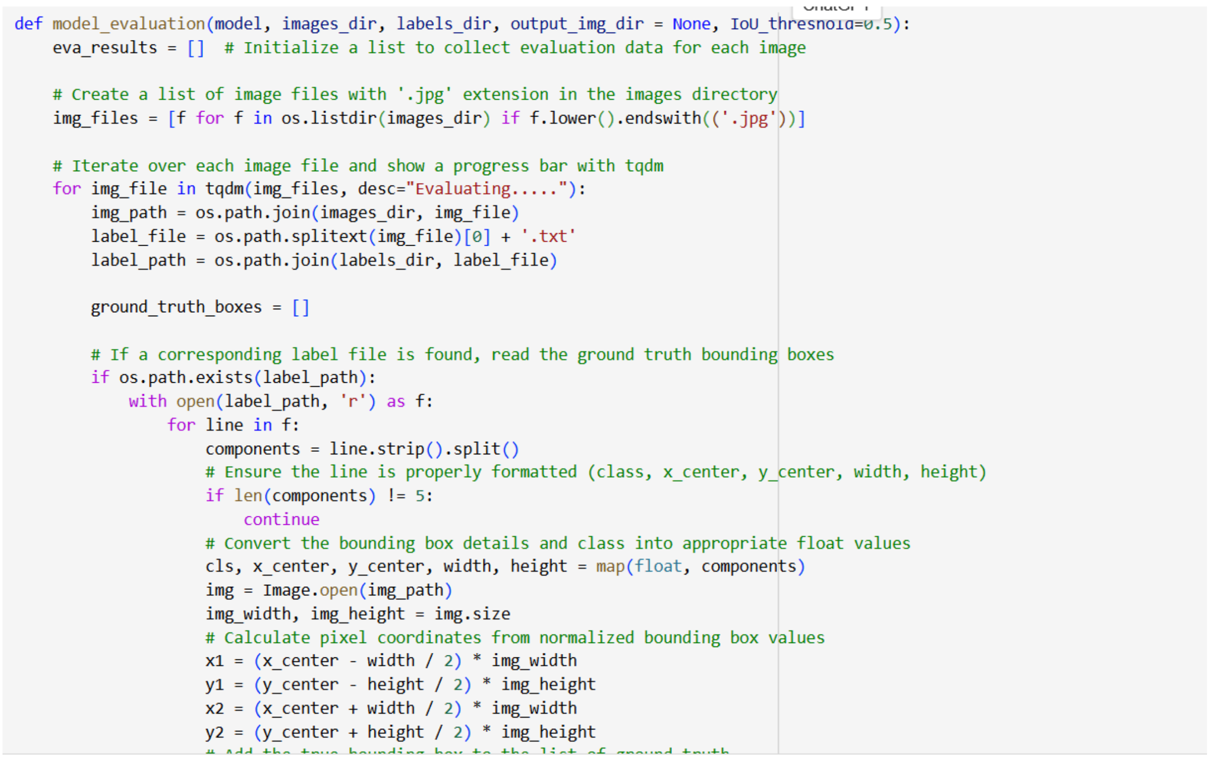
For the IoU computation, first, I initialize the cal_bb_IoU function computing the IoU between a predicted bounding box and a ground truth bounding box. It calculates the intersection area, the union area, and returns the IoU score, which is used to evaluate the model's predictions.



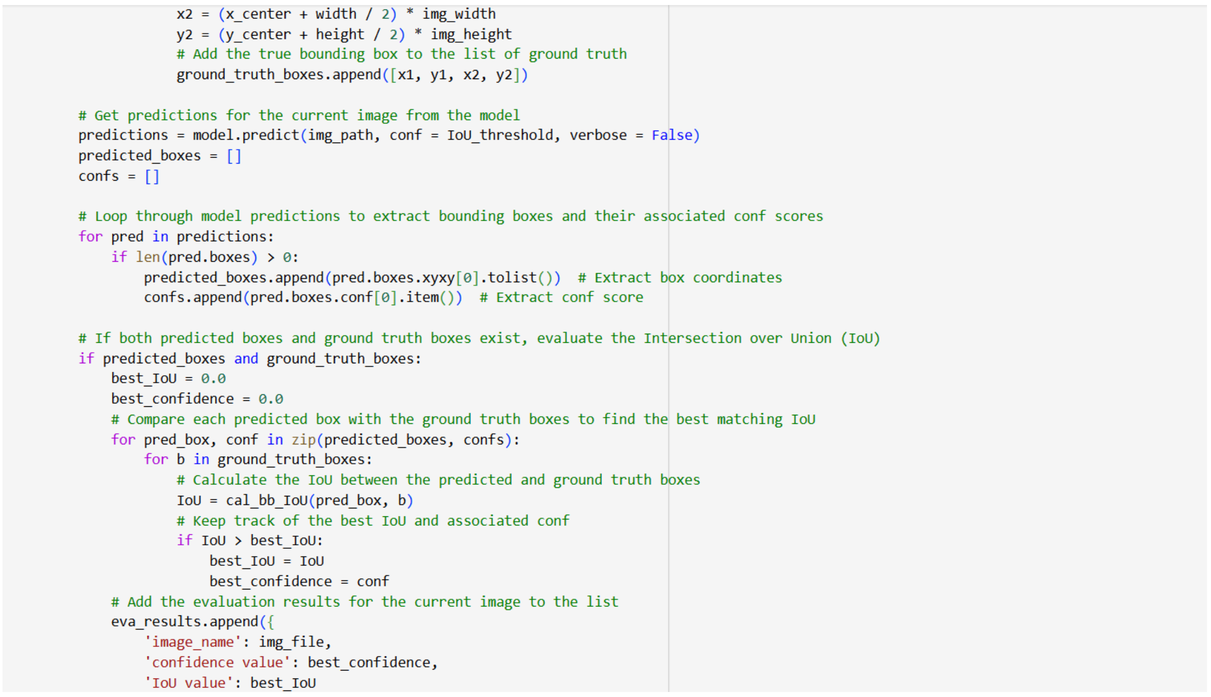
```python
def cal_bb_IoU(pred_bboxeses, gt_bboxeses):
    # Convert boxes to tensors
    pred_bboxes = torch.tensor(pred_bboxeses)  # Corrected variable name
    gt_bboxes = torch.tensor(gt_bboxeses)

    # Calculate the (x1, y1) coordinates of the intersection rectangle
    inter_x1 = torch.max(pred_bboxes[0], gt_bboxes[0])
    inter_y1 = torch.max(pred_bboxes[1], gt_bboxes[1])

    # Calculate the (x2, y2) coordinates of the intersection rectangle
    inter_x2 = torch.min(pred_bboxes[2], gt_bboxes[2])
    inter_y2 = torch.min(pred_bboxes[3], gt_bboxes[3])

    # Calculate the width and height of the intersection rectangle
    inter_width = (inter_x2 - inter_x1).clamp(min=0)
    inter_height = (inter_y2 - inter_y1).clamp(min=0)

    # Calculate the area of the intersection rectangle
    inter_area = inter_width * inter_height

    # Calculate the area of the predicted and true bounding boxes
    pred_area = (pred_bboxes[2] - pred_bboxes[0]) * (pred_bboxes[3] - pred_bboxes[1])
    gt_area = (gt_bboxes[2] - gt_bboxes[0]) * (gt_bboxes[3] - gt_bboxes[1])

    # Calculate the area of the union of the two bounding boxes
    union_area = pred_area + gt_area - inter_area

    # Avoid division by zero if union area is 0
    if union_area == 0:
        return 0.0

    # Calculate the IoU score
    iou = inter_area / union_area
    return iou.item()
```

The model_evaluation function processes each test image by loading the ground truth boxes, making predictions with the model, and calculating the IoU for each predicted box. It stores the best IoU and confidence and handles cases of false positives (predictions with no ground truth) or false negatives (no predictions).

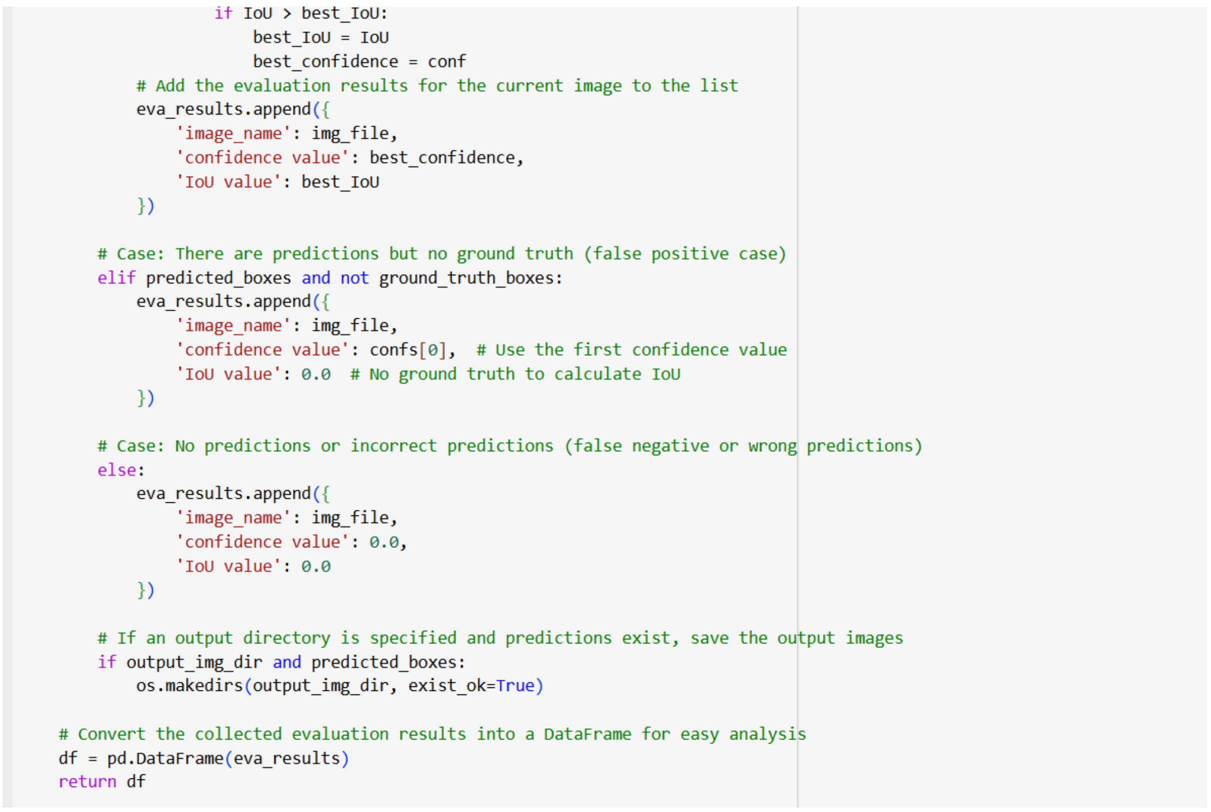
Student name: Nguyen Anh Tuan
Student ID: 103805949



```python
def model_evaluation(model, images_dir, labels_dir, output_img_dir = None, IoU_threshold=0.5):
    eva_results = []  # Initialize a list to collect evaluation data for each image

    # Create a list of image files with '.jpg' extension in the images directory
    img_files = [f for f in os.listdir(images_dir) if f.lower().endswith(('.jpg'))]

    # Iterate over each image file and show a progress bar with tqdm
    for img_file in tqdm(img_files, desc="Evaluating....."):
        img_path = os.path.join(images_dir, img_file)
        label_file = os.path.splitext(img_file)[0] + '.txt'
        label_path = os.path.join(labels_dir, label_file)

        ground_truth_boxes = []

        # If a corresponding label file is found, read the ground truth bounding boxes
        if os.path.exists(label_path):
            with open(label_path, 'r') as f:
                for line in f:
                    components = line.strip().split()
                    # Ensure the line is properly formatted (class, x_center, y_center, width, height)
                    if len(components) != 5:
                        continue
                    # Convert the bounding box details and class into appropriate float values
                    cls, x_center, y_center, width, height = map(float, components)
                    img = Image.open(img_path)
                    img_width, img_height = img.size
                    # Calculate pixel coordinates from normalized bounding box values
                    x1 = (x_center - width / 2) * img_width
                    y1 = (y_center - height / 2) * img_height
                    x2 = (x_center + width / 2) * img_width
                    y2 = (y_center + height / 2) * img_height
```

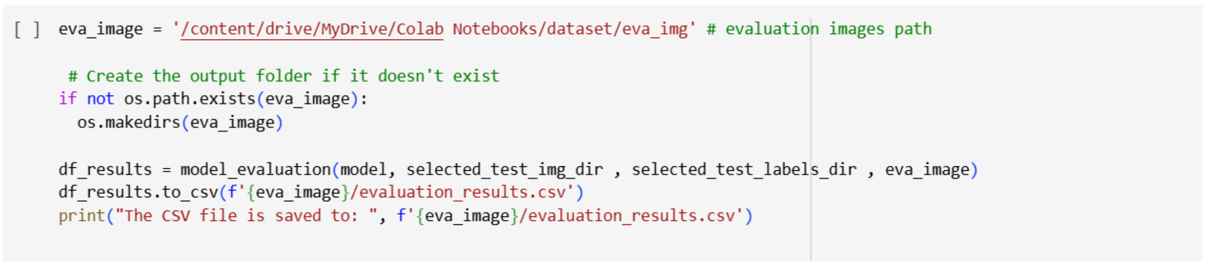


```python
                    x2 = (x_center + width / 2) * img_width
                    y2 = (y_center + height / 2) * img_height
                    # Add the true bounding box to the list of ground truth
                    ground_truth_boxes.append([x1, y1, x2, y2])

        # Get predictions for the current image from the model
        predictions = model.predict(img_path, conf = IoU_threshold, verbose = False)
        predicted_boxes = []
        confs = []

        # Loop through model predictions to extract bounding boxes and their associated conf scores
        for pred in predictions:
            if len(pred.boxes) > 0:
                predicted_boxes.append(pred.boxes.xyxy[0].tolist())  # Extract box coordinates
                confs.append(pred.boxes.conf[0].item())  # Extract conf score

        # If both predicted boxes and ground truth boxes exist, evaluate the Intersection over Union (IoU)
        if predicted_boxes and ground_truth_boxes:
            best_IoU = 0.0
            best_confidence = 0.0
            # Compare each predicted box with the ground truth boxes to find the best matching IoU
            for pred_box, conf in zip(predicted_boxes, confs):
                for b in ground_truth_boxes:
                    # Calculate the IoU between the predicted and ground truth boxes
                    IoU = cal_bb_IoU(pred_box, b)
                    # Keep track of the best IoU and associated conf
                    if IoU > best_IoU:
                        best_IoU = IoU
                        best_confidence = conf
            # Add the evaluation results for the current image to the list
            eva_results.append({
                'image_name': img_file,
                'confidence value': best_confidence,
                'IoU value': best_IoU
```



```
                if IoU > best_IoU:
                    best_IoU = IoU
                    best_confidence = conf
            # Add the evaluation results for the current image to the list
            eva_results.append({
                'image_name': img_file,
                'confidence value': best_confidence,
                'IoU value': best_IoU
            })

        # Case: There are predictions but no ground truth (false positive case)
        elif predicted_boxes and not ground_truth_boxes:
            eva_results.append({
                'image_name': img_file,
                'confidence value': confs[0],  # Use the first confidence value
                'IoU value': 0.0  # No ground truth to calculate IoU
            })

        # Case: No predictions or incorrect predictions (false negative or wrong predictions)
        else:
            eva_results.append({
                'image_name': img_file,
                'confidence value': 0.0,
                'IoU value': 0.0
            })

        # If an output directory is specified and predictions exist, save the output images
        if output_img_dir and predicted_boxes:
            os.makedirs(output_img_dir, exist_ok=True)

    # Convert the collected evaluation results into a DataFrame for easy analysis
    df = pd.DataFrame(eva_results)
    return df
```

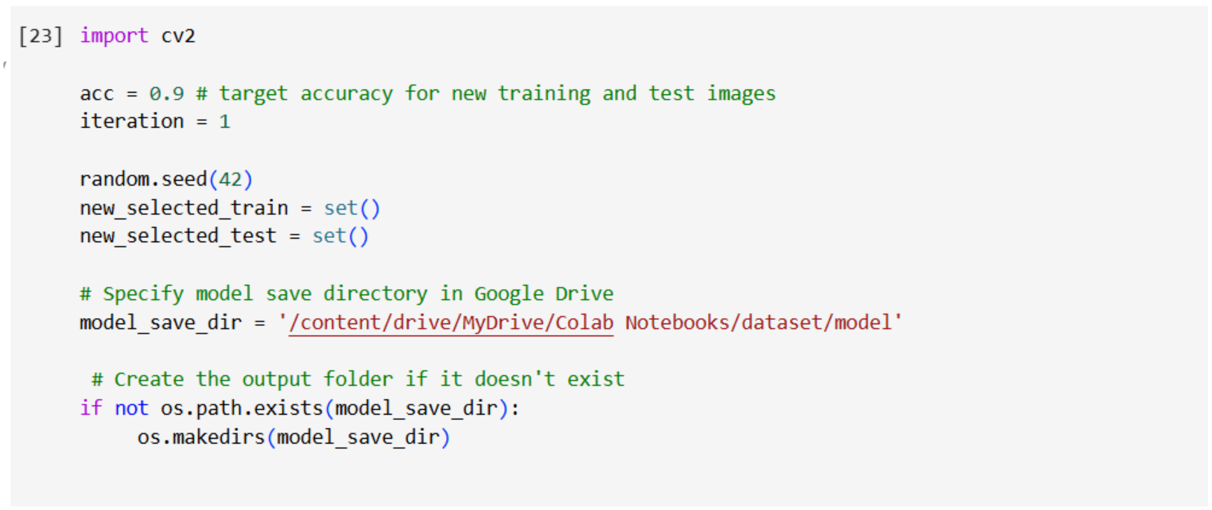
I also declare the path eva_image for the folder where evaluation images are stored. It checks if the folder exists. If not, it creates the folder using os.makedirs(). The model is then evaluated using the model_evaluation function, which inputs the model, image directory, label directory, and output folder. It returns the evaluation results in a DataFrame (df_results). The results are saved as a CSV file named evaluation_results.csv in the eva_image folder. Lastly, it prints a message confirming the location of the saved CSV file.



```
[ ]  eva_image = '/content/drive/MyDrive/Colab Notebooks/dataset/eva_img' # evaluation images path

      # Create the output folder if it doesn't exist
     if not os.path.exists(eva_image):
       os.makedirs(eva_image)

     df_results = model_evaluation(model, selected_test_img_dir , selected_test_labels_dir , eva_image)
     df_results.to_csv(f'{eva_image}/evaluation_results.csv')
     print("The CSV file is saved to: ", f'{eva_image}/evaluation_results.csv')
```

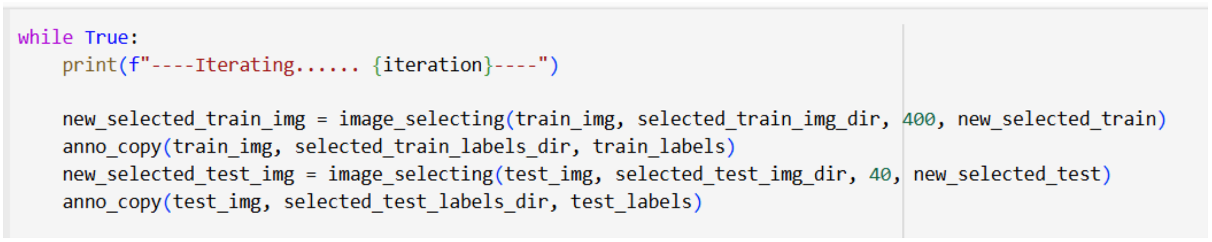
4. Iteratively train and test the model with new set of 400 training and 40 test images

In this part, first, I initialize key parameters, such as the target accuracy (acc), and set the first iteration (iteration = 1). I also ensure the necessary directories for saving the model are in place by checking if they exist and creating them if they don't.



```
[23] import cv2

     acc = 0.9 # target accuracy for new training and test images
     iteration = 1

     random.seed(42)
     new_selected_train = set()
     new_selected_test = set()

     # Specify model save directory in Google Drive
     model_save_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/model'

      # Create the output folder if it doesn't exist
     if not os.path.exists(model_save_dir):
         os.makedirs(model_save_dir)
```

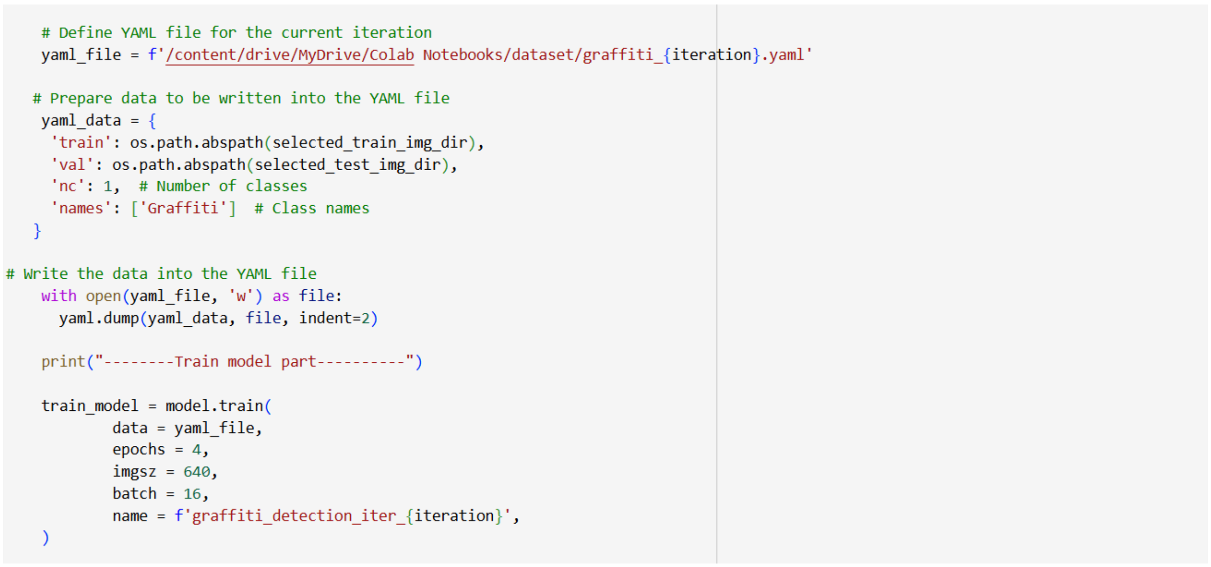
In each iteration, I select new images for training and testing. The image_selecting function selectsa set number of images from the training and test datasets. The corresponding annotations (labels) for these images are copied using the anno_copy function.



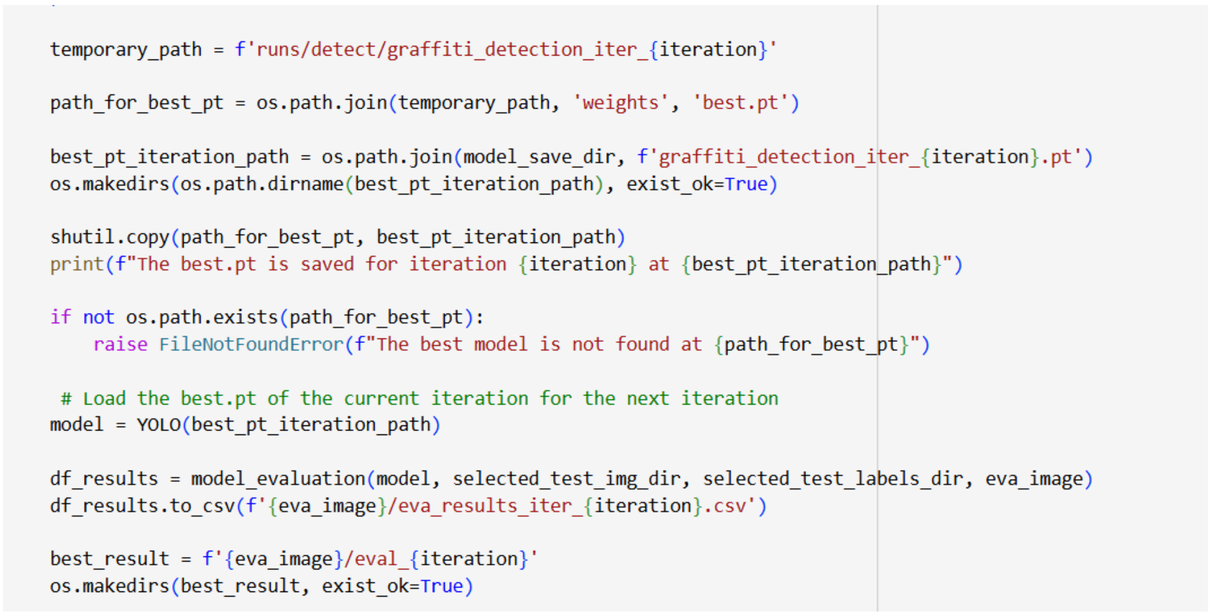
```
while True:
    print(f"----Iterating...... {iteration}----")

    new_selected_train_img = image_selecting(train_img, selected_train_img_dir, 400, new_selected_train)
    anno_copy(train_img, selected_train_labels_dir, train_labels)
    new_selected_test_img = image_selecting(test_img, selected_test_img_dir, 40, new_selected_test)
    anno_copy(test_img, selected_test_labels_dir, test_labels)
```

At the start of each iteration, a YAML configuration file is created. This file contains important information, such as the paths to the selected images, and metadata, such as the number of classes and class names. The YAML file is then written and saved to be used in training. The YOLO model is then trained with the selected training images, as specified in the YAML file. The training process runs for a specified number of epochs, and the best model weights are saved to a predefined directory for future use.

Student name: Nguyen Anh Tuan
Student ID: 103805949



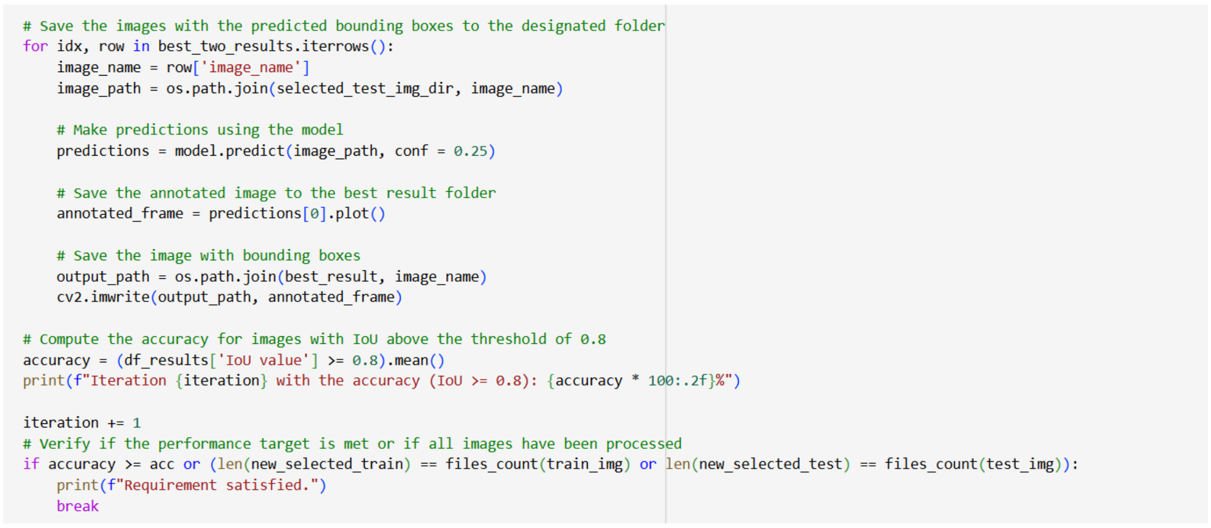
```
    # Define YAML file for the current iteration
    yaml_file = f'/content/drive/MyDrive/Colab Notebooks/dataset/graffiti_{iteration}.yaml'

   # Prepare data to be written into the YAML file
    yaml_data = {
     'train': os.path.abspath(selected_train_img_dir),
     'val': os.path.abspath(selected_test_img_dir),
     'nc': 1,  # Number of classes
     'names': ['Graffiti']  # Class names
   }

# Write the data into the YAML file
    with open(yaml_file, 'w') as file:
      yaml.dump(yaml_data, file, indent=2)

    print("--------Train model part----------")

    train_model = model.train(
            data = yaml_file,
            epochs = 4,
            imgsz = 640,
            batch = 16,
            name = f'graffiti_detection_iter_{iteration}',
    )
```



```
temporary_path = f'runs/detect/graffiti_detection_iter_{iteration}'

path_for_best_pt = os.path.join(temporary_path, 'weights', 'best.pt')

best_pt_iteration_path = os.path.join(model_save_dir, f'graffiti_detection_iter_{iteration}.pt')
os.makedirs(os.path.dirname(best_pt_iteration_path), exist_ok=True)

shutil.copy(path_for_best_pt, best_pt_iteration_path)
print(f"The best.pt is saved for iteration {iteration} at {best_pt_iteration_path}")

if not os.path.exists(path_for_best_pt):
    raise FileNotFoundError(f"The best model is not found at {path_for_best_pt}")

 # Load the best.pt of the current iteration for the next iteration
model = YOLO(best_pt_iteration_path)

df_results = model_evaluation(model, selected_test_img_dir, selected_test_labels_dir, eva_image)
df_results.to_csv(f'{eva_image}/eva_results_iter_{iteration}.csv')

best_result = f'{eva_image}/eval_{iteration}'
os.makedirs(best_result, exist_ok=True)
```

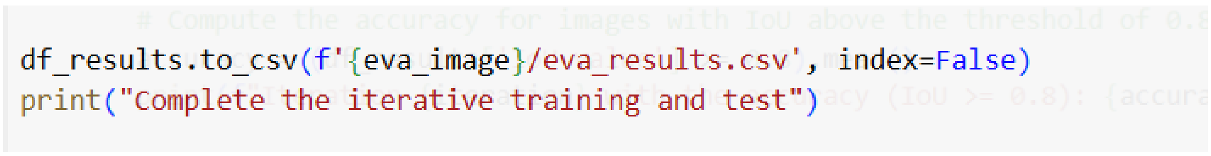
Once the model is trained, the code is evaluated on the test set. The evaluation step involves identifying the two images with the highest Intersection over Union (IoU) values. The model makes predictions on these images, and the results (including bounding boxes) are saved to a designated folder. After each evaluation, the code calculates the accuracy based on IoU values greater than or equal to 0.8. If the target accuracy has not been met, or if there are more images to process, the iteration continues. Otherwise, the loop breaks.

Student name: Nguyen Anh Tuan
Student ID: 103805949



```
# Save the images with the predicted bounding boxes to the designated folder
for idx, row in best_two_results.iterrows():
    image_name = row['image_name']
    image_path = os.path.join(selected_test_img_dir, image_name)

    # Make predictions using the model
    predictions = model.predict(image_path, conf = 0.25)

    # Save the annotated image to the best result folder
    annotated_frame = predictions[0].plot()

    # Save the image with bounding boxes
    output_path = os.path.join(best_result, image_name)
    cv2.imwrite(output_path, annotated_frame)

# Compute the accuracy for images with IoU above the threshold of 0.8
accuracy = (df_results['IoU value'] >= 0.8).mean()
print(f"Iteration {iteration} with the accuracy (IoU >= 0.8): {accuracy * 100:.2f}%")

iteration += 1
# Verify if the performance target is met or if all images have been processed
if accuracy >= acc or (len(new_selected_train) == files_count(train_img) or len(new_selected_test) == files_count(test_img)):
    print(f"Requirement satisfied.")
    break
```

Finally, after all iterations, the results are saved in CSV files, and the process ends, having either met the target accuracy or processed all available images.



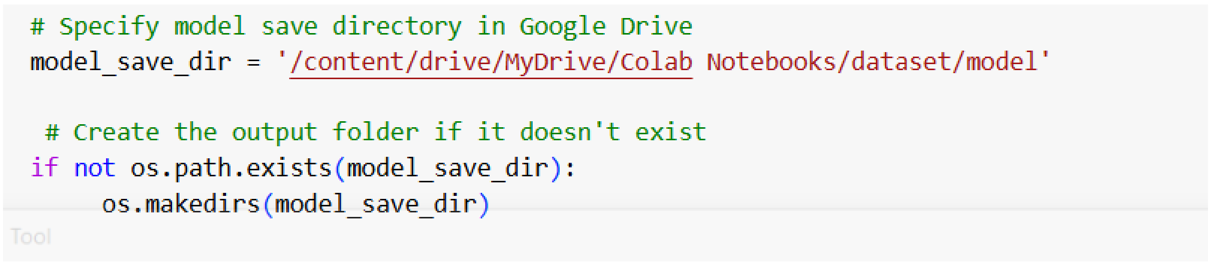
```
df_results.to_csv(f'{eva_image}/eva_results.csv', index=False)
print("Complete the iterative training and test")
```

About the output for this part, I store the CSV file of outcome for each iteration, and 2 good samples of detected images with bounding box in this link below.

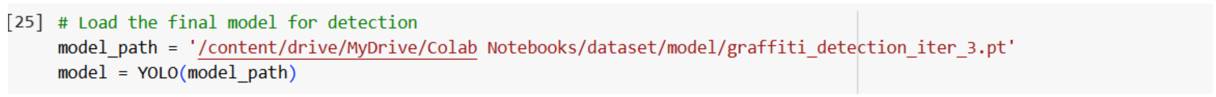
- Link: https://drive.google.com/drive/u/1/folders/1O56rMlWf_NudnjAN8dCbPxGSeO73ZYTx

5. Use your final model to detect graffiti in real-time video data.

I start this task by setting up the model's save directory on Google Drive and checking if the folder exists. If it doesn't, the folder is created. The final model for graffiti detection is then loaded from the specified path (graffiti_detection_iter_3.pt) using the YOLO framework.



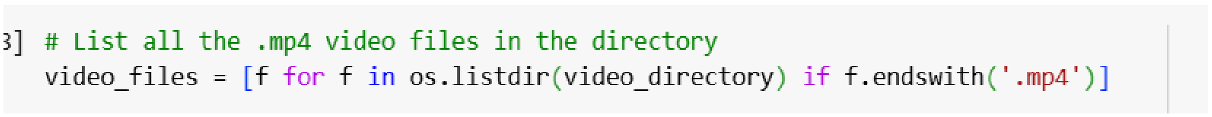
```
# Specify model save directory in Google Drive
model_save_dir = '/content/drive/MyDrive/Colab Notebooks/dataset/model'

 # Create the output folder if it doesn't exist
if not os.path.exists(model_save_dir):
     os.makedirs(model_save_dir)
```



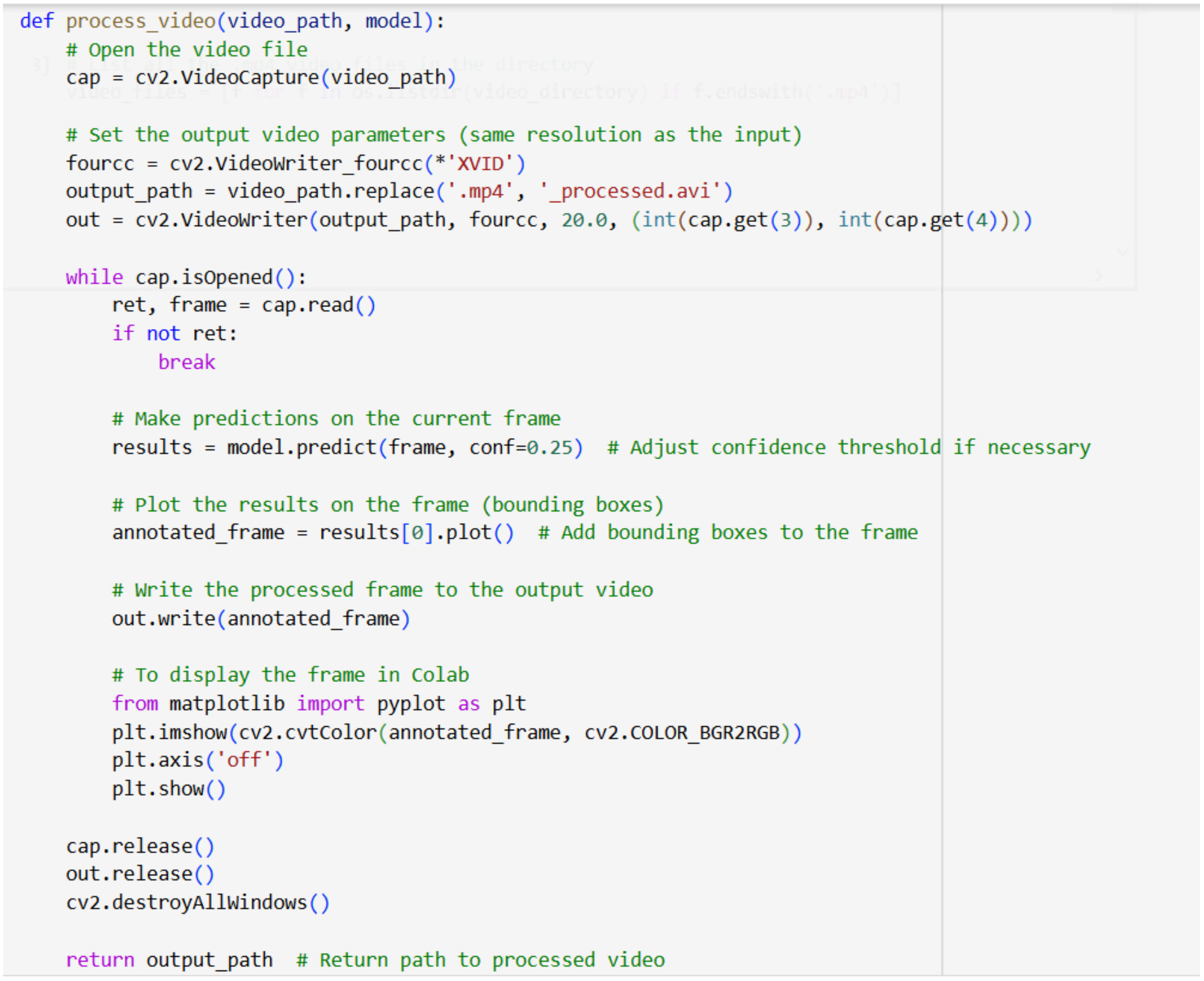
```
[25] # Load the final model for detection
     model_path = '/content/drive/MyDrive/Colab Notebooks/dataset/model/graffiti_detection_iter_3.pt'
     model = YOLO(model_path)
```

Student name: Nguyen Anh Tuan
Student ID: 103805949

Next, I specify the directory where five videos are stored on Google Drive and list all .mp4 video files in that directory. A function called process_video is defined to process these videos. It opens each video, sets up output video parameters (such as resolution), and creates an output file to save the processed frames. For each frame in the video, the model makes predictions and draws bounding boxes around detected graffiti. These annotated frames are written to the output video file. After processing all frames in a video, the function releases the video resources and returns the path to the processed video. The main loop processes each video in the directory, calling the process_video function and saving the processed videos. The paths to these processed videos are stored in a list



```
3] # List all the .mp4 video files in the directory
   video_files = [f for f in os.listdir(video_directory) if f.endswith('.mp4')]
```



```
def process_video(video_path, model):
    # Open the video file
    cap = cv2.VideoCapture(video_path)

    # Set the output video parameters (same resolution as the input)
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    output_path = video_path.replace('.mp4', '_processed.avi')
    out = cv2.VideoWriter(output_path, fourcc, 20.0, (int(cap.get(3)), int(cap.get(4))))

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        # Make predictions on the current frame
        results = model.predict(frame, conf=0.25)  # Adjust confidence threshold if necessary

        # Plot the results on the frame (bounding boxes)
        annotated_frame = results[0].plot()  # Add bounding boxes to the frame

        # Write the processed frame to the output video
        out.write(annotated_frame)

        # To display the frame in Colab
        from matplotlib import pyplot as plt
        plt.imshow(cv2.cvtColor(annotated_frame, cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.show()

    cap.release()
    out.release()
    cv2.destroyAllWindows()

    return output_path  # Return path to processed video
```

Finally, I use the files.download() method from Google Colab to provide download links for each processed video and prompt the user to download the videos directly. The 5 detection outcomes of the 5 videos are the video files with the “video(number)_processed.avi” format (e.g: video2_processed.avi)
You can access the link below to see the video outcomes.

- Link: https://drive.google.com/drive/u/1/folders/1Zsi51MAHAn7jDF6TKg-zR4z98vOzqVzu

6. Appendix:

- Source code: https://colab.research.google.com/drive/1eFc5JmM6Gf5rtp_RF1dzDIY_jelijFnM?usp=sharing
- Link for the dataset: https://drive.google.com/drive/u/1/folders/1Wbam7iygHwjR6lsXMZRtof1pQFQPRxfn