Portfolio Task 6

Arnob Ghosh
103494114

OneDrive File Link: week-06-portfolio

Abstract

The creation and assessment of a deep learning model for real-time video and image graffiti detection using YOLO v5 is presented in this portfolio submission. The project exhibits a thorough comprehension of PyTorch deployment, iterative optimization, evaluation metrics, model training, and data preprocessing. The submission is structured as follows and contains labeled datasets, model results, and source code:

- Annotation Conversion Function: To convert the given annotation format in training labels to the YOLO annotation format, a custom function called convert_annotations is implemented. By guaranteeing that bounding boxes are properly formatted and normalized, this function makes it easier to integrate with the YOLO v5 training pipeline.

- YOLO v5 Model Training: 400 randomly chosen photos from the training dataset are used to train the YOLO v5 model. Through the use of the converted annotations, the model is trained to identify graffiti in images with accuracy. The model's performance is improved with each iteration of the training process. The best trained models are saved from each iteration.files. The iteration number is indicated by X in the week-06-portfolio/train/runs/train/graffiti_detection_iter_X/weights/ directories.

- IoU Computation and Evaluation: The effectiveness of the model is assessed using 40 randomly chosen images from the test dataset. To evaluate the detected bounding box accuracy against the ground truth, the Intersection over Union (IoU) is computed for every test image. The evaluation results are combined into CSV files and include image_name, confidence_value, and IoU_value. An IoU value of 0 is assigned to images where graffiti is not detected. These files, which correspond to every training iteration, are kept in the train and evaluation_images_iter_X directories.

- • Iterative Training and Optimization: The YOLO v5 model is retrained using fresh sets of 400 training and 40 test images for each iteration of the training process. This process is repeated until all test and training images have been used, or 80% of the test images have an IoU greater than 90%. The pretrained model from the previous step is used in each iteration to enable progressive learning and improved performance. The products of every iteration, such as CSV files and sample annotated images, are arranged in the train directory under the corresponding iteration folders.

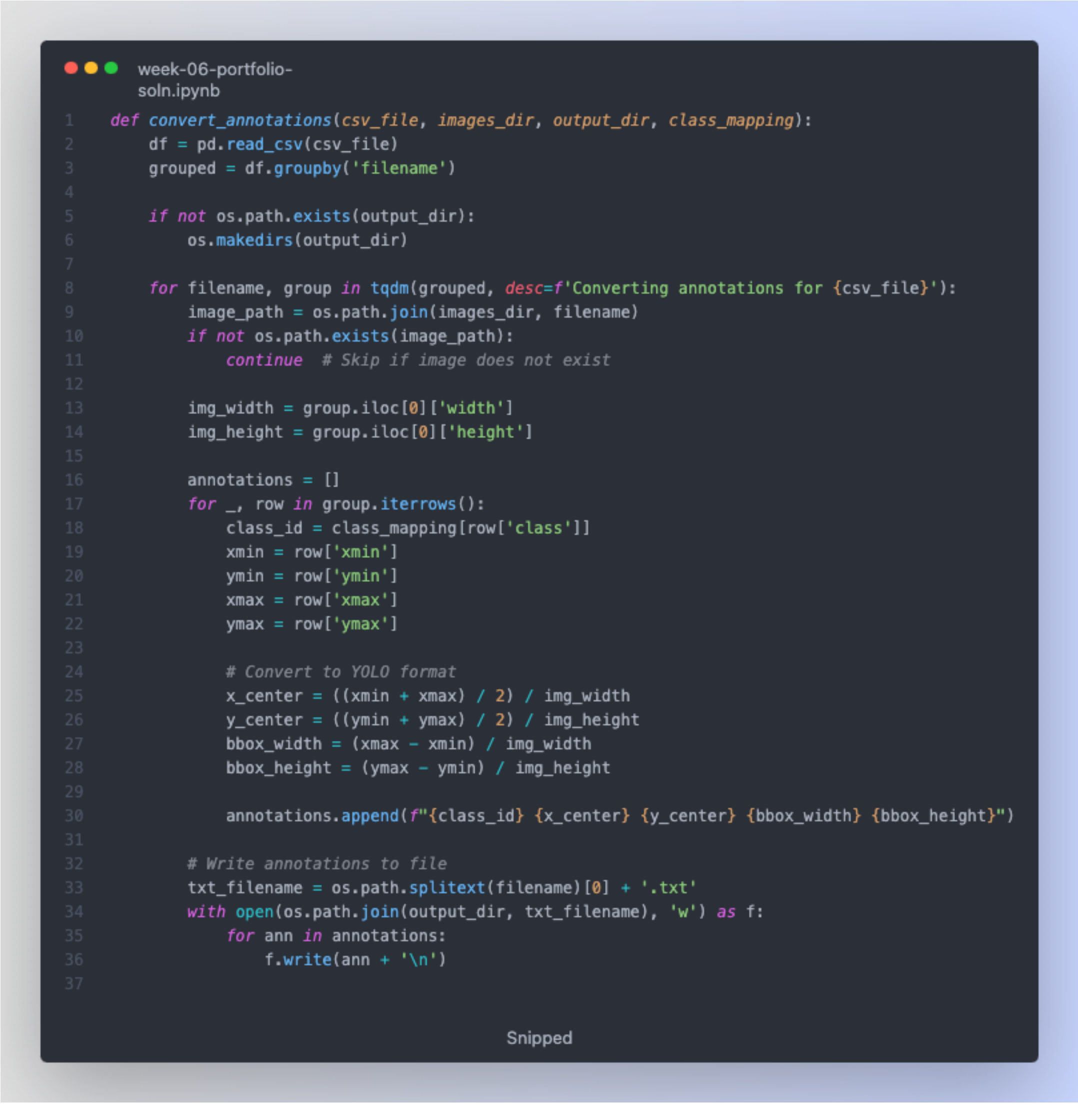
- Real-Time Video Detection: To identify graffiti in real-time video data, the fully optimized YOLO v5 model is used. After processing different types of video inputs, the model finds and labels instances of graffiti with bounding boxes and confidence scores. The Pexels example video sources are used to illustrate the real-time detection capabilities of the model. The detection results are stored in the results directory and are arranged according to each video track in subfolders called track, track2, etc.

Table of Contents

Deep Learning Using Yolo Models

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

I have included the convert_annotations function in your code to convert the given annotation format in training labels to the YOLO annotation format.

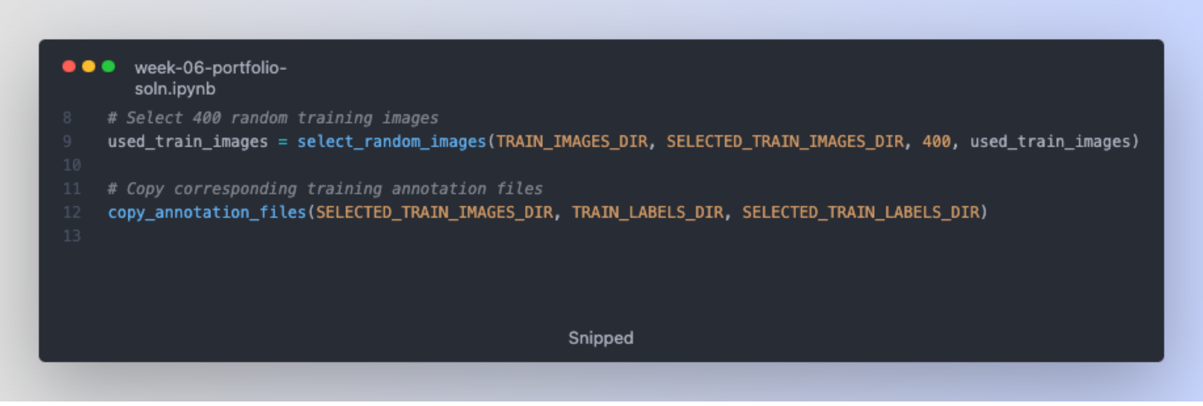


```python
def convert_annotations(csv_file, images_dir, output_dir, class_mapping):
    df = pd.read_csv(csv_file)
    grouped = df.groupby('filename')

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

    for filename, group in tqdm(grouped, desc=f'Converting annotations for {csv_file}'):
        image_path = os.path.join(images_dir, filename)
        if not os.path.exists(image_path):
            continue  # Skip if image does not exist

        img_width = group.iloc[0]['width']
        img_height = group.iloc[0]['height']

        annotations = []
        for _, row in group.iterrows():
            class_id = class_mapping[row['class']]
            xmin = row['xmin']
            ymin = row['ymin']
            xmax = row['xmax']
            ymax = row['ymax']

            # Convert to YOLO format
            x_center = ((xmin + xmax) / 2) / img_width
            y_center = ((ymin + ymax) / 2) / img_height
            bbox_width = (xmax - xmin) / img_width
            bbox_height = (ymax - ymin) / img_height

            annotations.append(f"{class_id} {x_center} {y_center} {bbox_width} {bbox_height}")

        # Write annotations to file
        txt_filename = os.path.splitext(filename)[0] + '.txt'
        with open(os.path.join(output_dir, txt_filename), 'w') as f:
            for ann in annotations:
                f.write(ann + '\n')
```

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

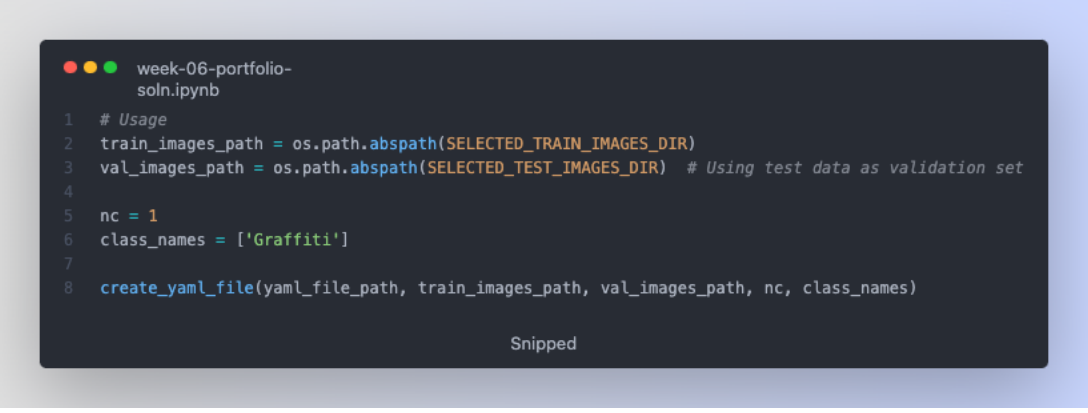
My code takes the following actions to train and build a YOLO model for graffiti detection using 400 randomly chosen training images:

1. Picking a Random Training Picture



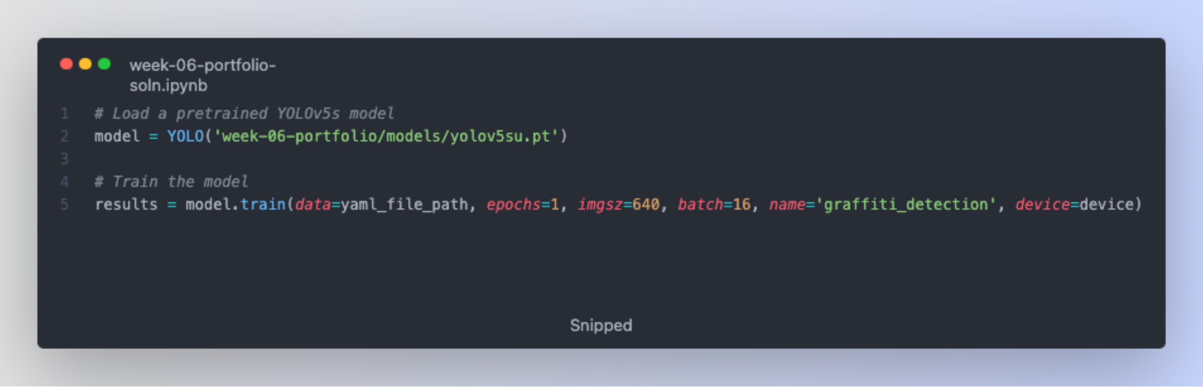
```
# Select 400 random training images
used_train_images = select_random_images(TRAIN_IMAGES_DIR, SELECTED_TRAIN_IMAGES_DIR, 400, used_train_images)

# Copy corresponding training annotation files
copy_annotation_files(SELECTED_TRAIN_IMAGES_DIR, TRAIN_LABELS_DIR, SELECTED_TRAIN_LABELS_DIR)
```

2. Producing the File with YAML Configuration.



```
# Usage
train_images_path = os.path.abspath(SELECTED_TRAIN_IMAGES_DIR)
val_images_path = os.path.abspath(SELECTED_TEST_IMAGES_DIR)  # Using test data as validation set

nc = 1
class_names = ['Graffiti']

create_yaml_file(yaml_file_path, train_images_path, val_images_path, nc, class_names)
```

3. Using the YOLO Model in Training

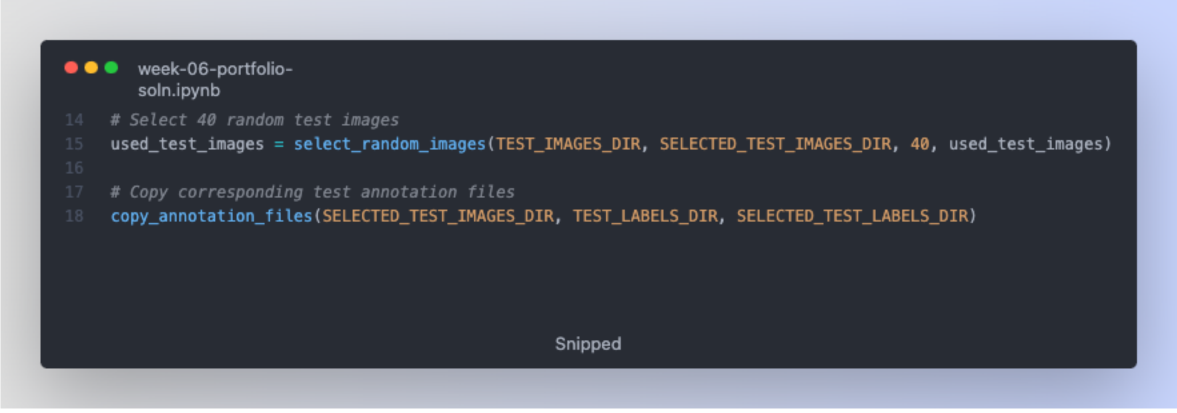


```
# Load a pretrained YOLOv5s model
model = YOLO('week-06-portfolio/models/yolov5su.pt')

# Train the model
results = model.train(data=yaml_file_path, epochs=1, imgsz=640, batch=16, name='graffiti_detection', device=device)
```

3) Using the test data, select 40 images at random, calculate the IoU for each, and create a CSV file with three columns: image name, confidence value, and IoU value. An image's IoU will be 0.3 if no graffiti is found on it.Making the YOLO Model Better

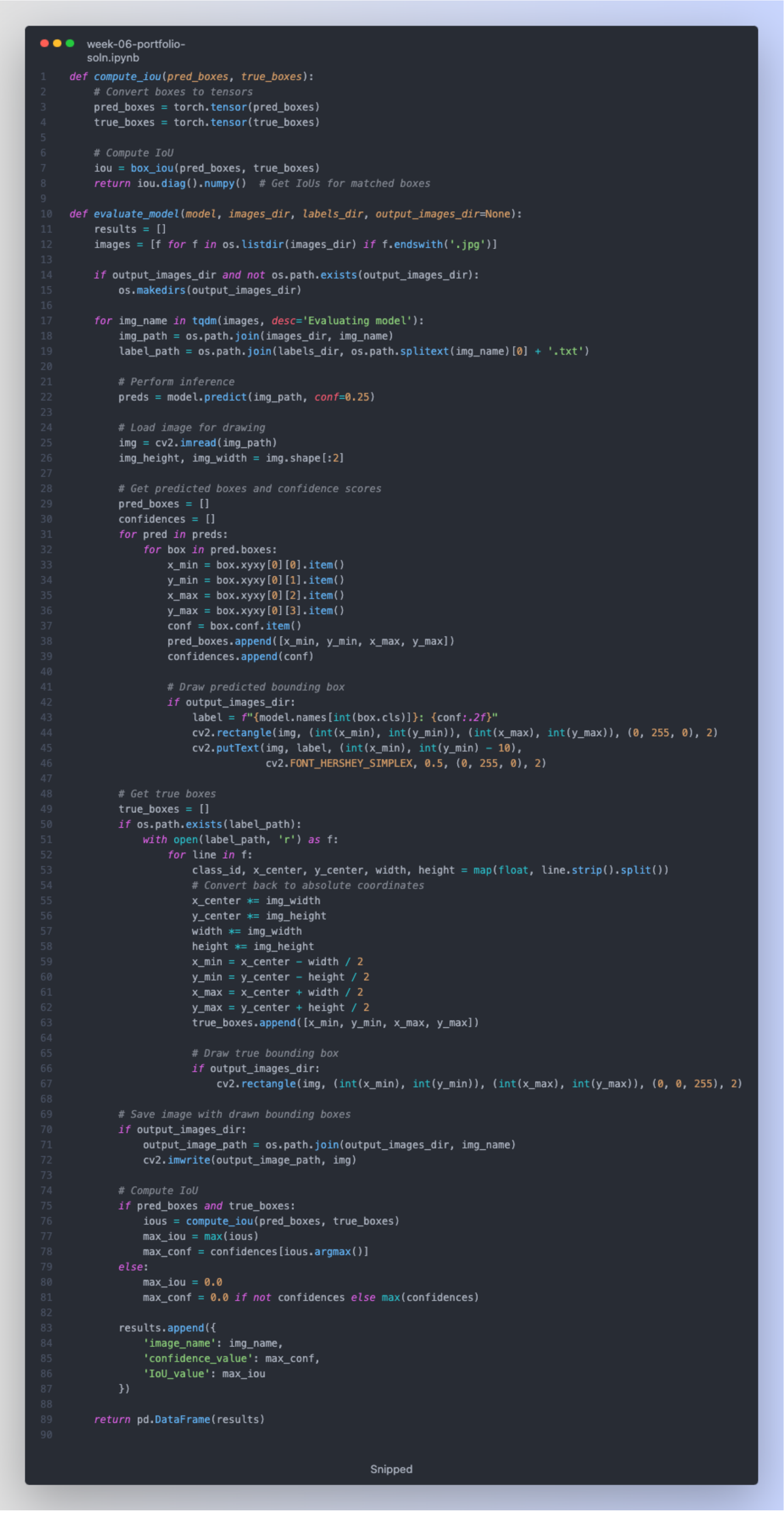
Your code does the following to calculate the Intersection over Union (IoU) for each of the 40 randomly chosen test images and assess the performance of the trained YOLO model on them:

1. Choosing an Image for Random Testing



```
# Select 40 random test images
used_test_images = select_random_images(TEST_IMAGES_DIR, SELECTED_TEST_IMAGES_DIR, 40, used_test_images)

# Copy corresponding test annotation files
copy_annotation_files(SELECTED_TEST_IMAGES_DIR, TEST_LABELS_DIR, SELECTED_TEST_LABELS_DIR)
```

2. Assessing the Framework and Calculating IoU



week-06-portfolio-soln.ipynb

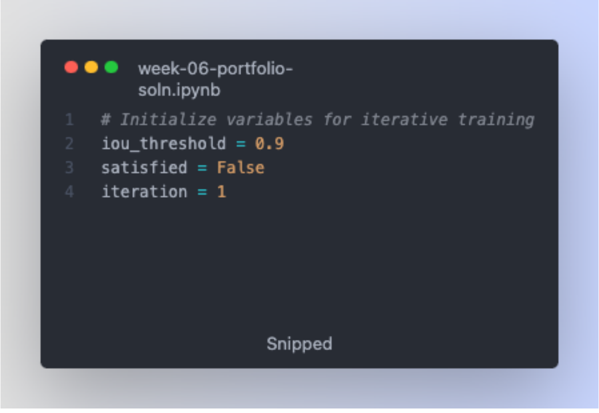
```python
def compute_iou(pred_boxes, true_boxes):
    # Convert boxes to tensors
    pred_boxes = torch.tensor(pred_boxes)
    true_boxes = torch.tensor(true_boxes)

    # Compute IoU
    iou = box_iou(pred_boxes, true_boxes)
    return iou.diag().numpy()  # Get IoUs for matched boxes

def evaluate_model(model, images_dir, labels_dir, output_images_dir=None):
    results = []
    images = [f for f in os.listdir(images_dir) if f.endswith('.jpg')]

    if output_images_dir and not os.path.exists(output_images_dir):
        os.makedirs(output_images_dir)

    for img_name in tqdm(images, desc='Evaluating model'):
        img_path = os.path.join(images_dir, img_name)
        label_path = os.path.join(labels_dir, os.path.splitext(img_name)[0] + '.txt')

        # Perform inference
        preds = model.predict(img_path, conf=0.25)

        # Load image for drawing
        img = cv2.imread(img_path)
        img_height, img_width = img.shape[:2]

        # Get predicted boxes and confidence scores
        pred_boxes = []
        confidences = []
        for pred in preds:
            for box in pred.boxes:
                x_min = box.xyxy[0][0].item()
                y_min = box.xyxy[0][1].item()
                x_max = box.xyxy[0][2].item()
                y_max = box.xyxy[0][3].item()
                conf = box.conf.item()
                pred_boxes.append([x_min, y_min, x_max, y_max])
                confidences.append(conf)

                # Draw predicted bounding box
                if output_images_dir:
                    label = f"{model.names[int(box.cls)]}: {conf:.2f}"
                    cv2.rectangle(img, (int(x_min), int(y_min)), (int(x_max), int(y_max)), (0, 255, 0), 2)
                    cv2.putText(img, label, (int(x_min), int(y_min) - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        # Get true boxes
        true_boxes = []
        if os.path.exists(label_path):
            with open(label_path, 'r') as f:
                for line in f:
                    class_id, x_center, y_center, width, height = map(float, line.strip().split())
                    # Convert back to absolute coordinates
                    x_center *= img_width
                    y_center *= img_height
                    width *= img_width
                    height *= img_height
                    x_min = x_center - width / 2
                    y_min = y_center - height / 2
                    x_max = x_center + width / 2
                    y_max = y_center + height / 2
                    true_boxes.append([x_min, y_min, x_max, y_max])

                    # Draw true bounding box
                    if output_images_dir:
                        cv2.rectangle(img, (int(x_min), int(y_min)), (int(x_max), int(y_max)), (0, 0, 255), 2)

        # Save image with drawn bounding boxes
        if output_images_dir:
            output_image_path = os.path.join(output_images_dir, img_name)
            cv2.imwrite(output_image_path, img)

        # Compute IoU
        if pred_boxes and true_boxes:
            ious = compute_iou(pred_boxes, true_boxes)
            max_iou = max(ious)
            max_conf = confidences[ious.argmax()]
        else:
            max_iou = 0.0
            max_conf = 0.0 if not confidences else max(confidences)

        results.append({
            'image_name': img_name,
            'confidence_value': max_conf,
            'IoU_value': max_iou
        })

    return pd.DataFrame(results)
```

Snipped

4) MUST iteratively train and test the model using a fresh set of 400 training and 40 test images until the IoU value of 80% of the images in test data is over 90% or all images are used for training and testing purposes. Make sure the pre-trained model for the new iteration is the one from the previous iteration.

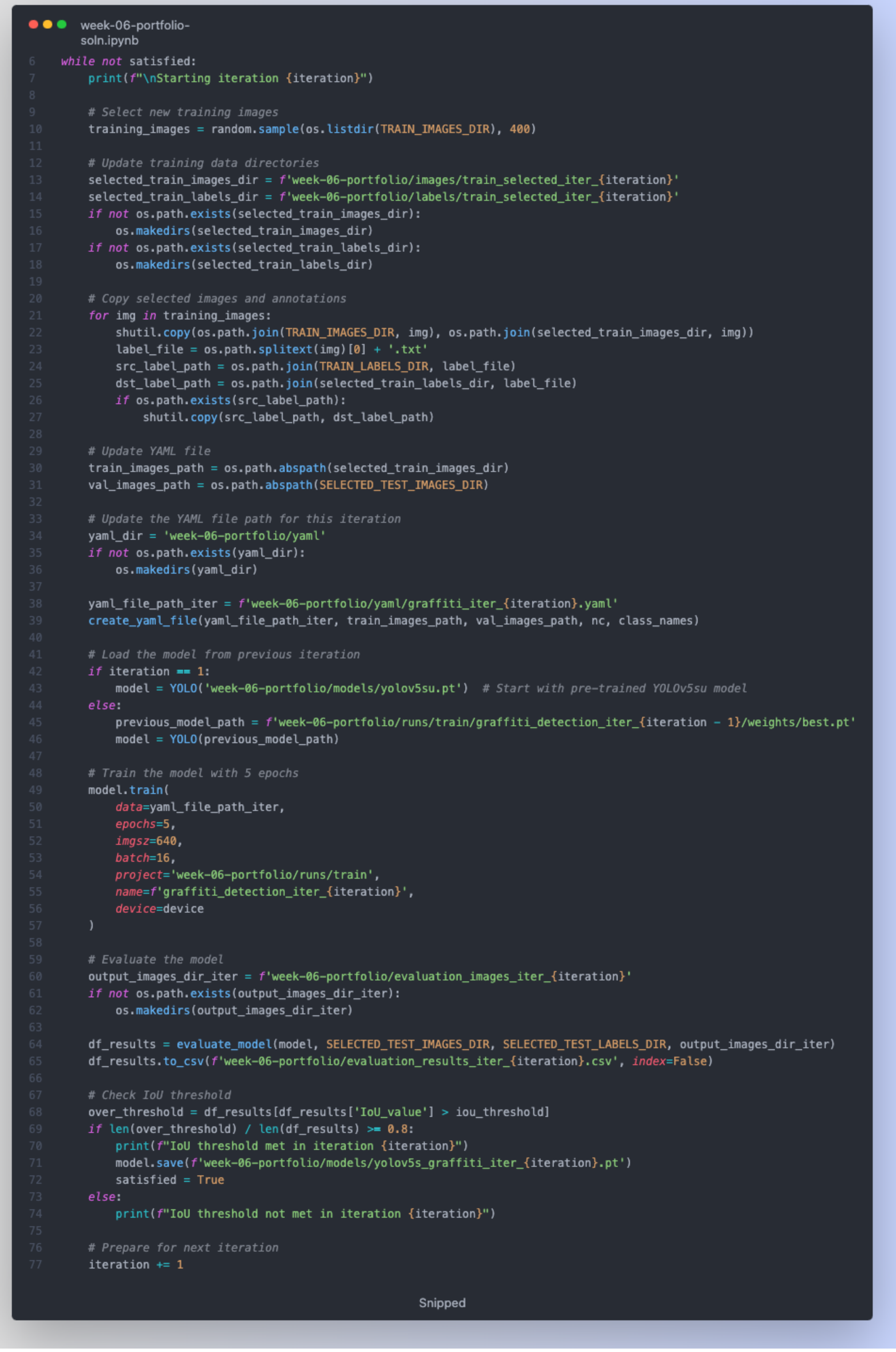
I implement a while-loop that keeps going until the stopping criteria are met in order to iteratively train and test the YOLO model until at least 80% of the test images achieve an IoU greater than 90%. Here's a detailed explanation of how this is accomplished:

1. Setting the threshold initial



```
# Initialize variables for iterative training
iou_threshold = 0.9
satisfied = False
iteration = 1
```

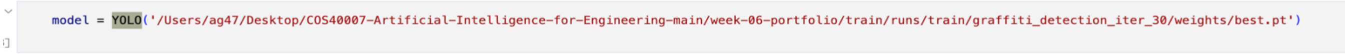
2. Loop for Iterative Training and Assessment



```python
while not satisfied:
    print(f"\nStarting iteration {iteration}")

    # Select new training images
    training_images = random.sample(os.listdir(TRAIN_IMAGES_DIR), 400)

    # Update training data directories
    selected_train_images_dir = f'week-06-portfolio/images/train_selected_iter_{iteration}'
    selected_train_labels_dir = f'week-06-portfolio/labels/train_selected_iter_{iteration}'
    if not os.path.exists(selected_train_images_dir):
        os.makedirs(selected_train_images_dir)
    if not os.path.exists(selected_train_labels_dir):
        os.makedirs(selected_train_labels_dir)

    # Copy selected images and annotations
    for img in training_images:
        shutil.copy(os.path.join(TRAIN_IMAGES_DIR, img), os.path.join(selected_train_images_dir, img))
        label_file = os.path.splitext(img)[0] + '.txt'
        src_label_path = os.path.join(TRAIN_LABELS_DIR, label_file)
        dst_label_path = os.path.join(selected_train_labels_dir, label_file)
        if os.path.exists(src_label_path):
            shutil.copy(src_label_path, dst_label_path)

    # Update YAML file
    train_images_path = os.path.abspath(selected_train_images_dir)
    val_images_path = os.path.abspath(SELECTED_TEST_IMAGES_DIR)

    # Update the YAML file path for this iteration
    yaml_dir = 'week-06-portfolio/yaml'
    if not os.path.exists(yaml_dir):
        os.makedirs(yaml_dir)

    yaml_file_path_iter = f'week-06-portfolio/yaml/graffiti_iter_{iteration}.yaml'
    create_yaml_file(yaml_file_path_iter, train_images_path, val_images_path, nc, class_names)

    # Load the model from previous iteration
    if iteration == 1:
        model = YOLO('week-06-portfolio/models/yolov5su.pt')  # Start with pre-trained YOLOv5su model
    else:
        previous_model_path = f'week-06-portfolio/runs/train/graffiti_detection_iter_{iteration - 1}/weights/best.pt'
        model = YOLO(previous_model_path)

    # Train the model with 5 epochs
    model.train(
        data=yaml_file_path_iter,
        epochs=5,
        imgsz=640,
        batch=16,
        project='week-06-portfolio/runs/train',
        name=f'graffiti_detection_iter_{iteration}',
        device=device
    )

    # Evaluate the model
    output_images_dir_iter = f'week-06-portfolio/evaluation_images_iter_{iteration}'
    if not os.path.exists(output_images_dir_iter):
        os.makedirs(output_images_dir_iter)

    df_results = evaluate_model(model, SELECTED_TEST_IMAGES_DIR, SELECTED_TEST_LABELS_DIR, output_images_dir_iter)
    df_results.to_csv(f'week-06-portfolio/evaluation_results_iter_{iteration}.csv', index=False)

    # Check IoU threshold
    over_threshold = df_results[df_results['IoU_value'] > iou_threshold]
    if len(over_threshold) / len(df_results) >= 0.8:
        print(f"IoU threshold met in iteration {iteration}")
        model.save(f'week-06-portfolio/models/yolov5s_graffiti_iter_{iteration}.pt')
        satisfied = True
    else:
        print(f"IoU threshold not met in iteration {iteration}")

    # Prepare for next iteration
    iteration += 1
```

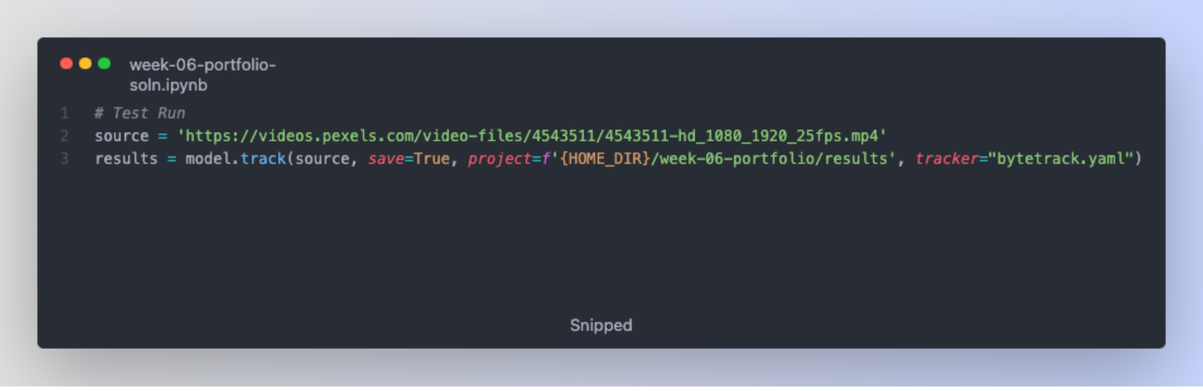
5) final model to detect graffiti in real-time video data.
My code carries out the following actions in order to implement the final trained YOLO model for real-time graffiti detection in video data:

1. Put the learned model on.



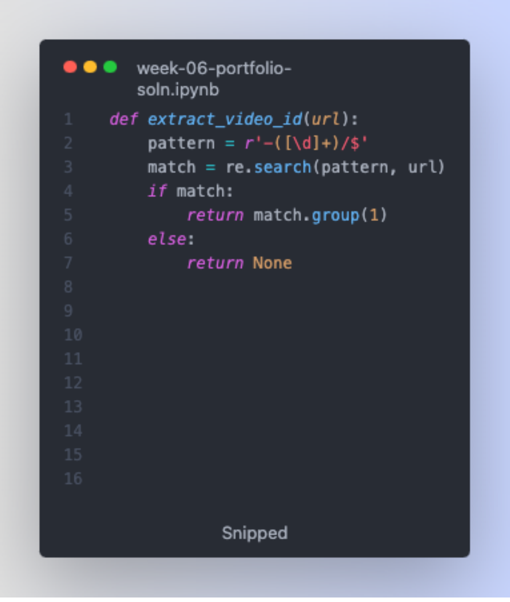
```
model = YOLO('/Users/ag47/Desktop/COS40007-Artificial-Intelligence-for-Engineering-main/week-06-portfolio/train/runs/train/graffiti_detection_iter_30/weights/best.pt')
```

2. Testing using a Pexels sample video that satisfies the requirements (the URL was manually retrieved).



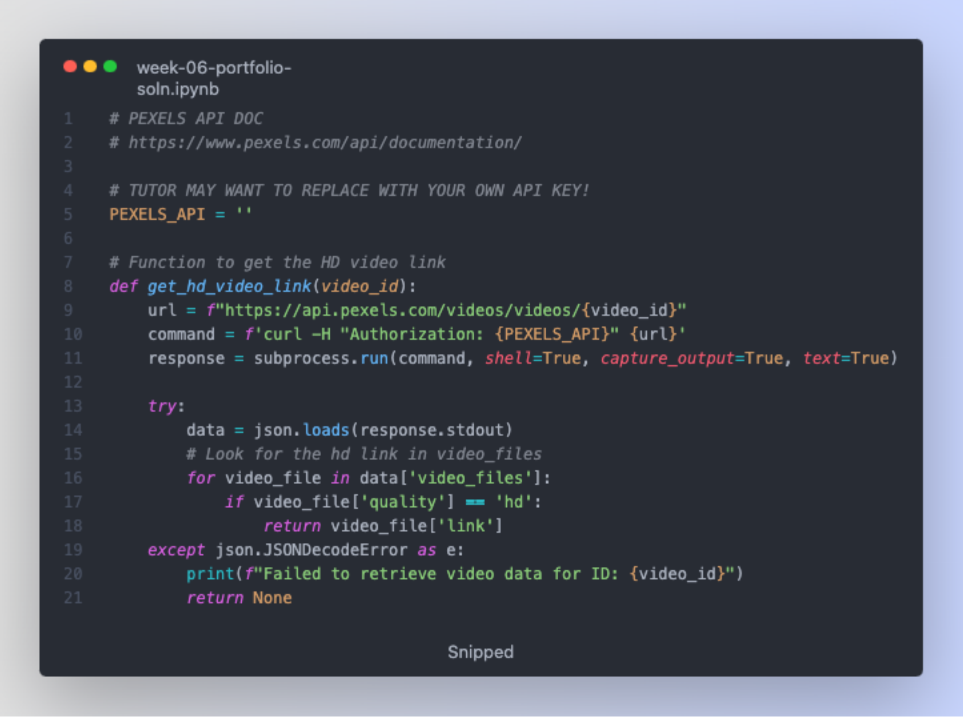
```
# Test Run
source = 'https://videos.pexels.com/video-files/4543511/4543511-hd_1080_1920_25fps.mp4'
results = model.track(source, save=True, project=f'{HOME_DIR}/week-06-portfolio/results', tracker="bytetrack.yaml")
```

3. **Get video from Pexels by utilizing its API.:**The ability to extract the video ID from the requirements' supplied URL

   a. 

```
def extract_video_id(url):
    pattern = r'-([\d]+)/$'
    match = re.search(pattern, url)
    if match:
        return match.group(1)
    else:
        return None
```

b. The ability to obtain the Pexels video stream URL

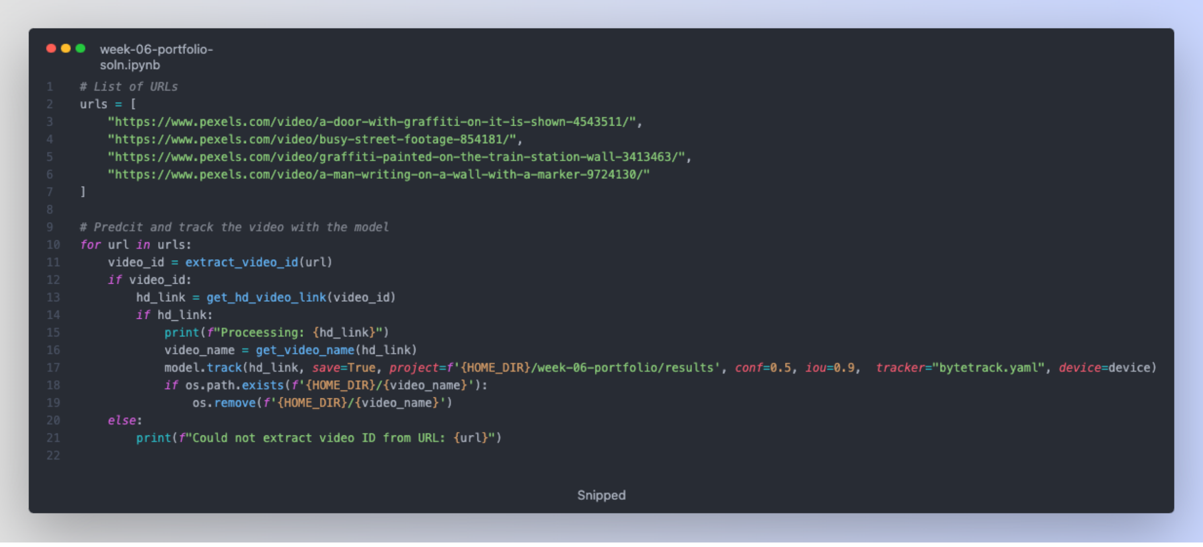


week-06-portfolio-soln.ipynb

```python
# PEXELS API DOC
# https://www.pexels.com/api/documentation/

# TUTOR MAY WANT TO REPLACE WITH YOUR OWN API KEY!
PEXELS_API = ''

# Function to get the HD video link
def get_hd_video_link(video_id):
    url = f"https://api.pexels.com/videos/videos/{video_id}"
    command = f'curl -H "Authorization: {PEXELS_API}" {url}'
    response = subprocess.run(command, shell=True, capture_output=True, text=True)

    try:
        data = json.loads(response.stdout)
        # Look for the hd link in video_files
        for video_file in data['video_files']:
            if video_file['quality'] == 'hd':
                return video_file['link']
    except json.JSONDecodeError as e:
        print(f"Failed to retrieve video data for ID: {video_id}")
        return None
```

Snipped

4. Get video and utilize a model for tracking and prediction.



week-06-portfolio-soln.ipynb

```python
# List of URLs
urls = [
    "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/video/a-man-writing-on-a-wall-with-a-marker-9724130/"
]

# Predcit and track the video with the model
for url in urls:
    video_id = extract_video_id(url)
    if video_id:
        hd_link = get_hd_video_link(video_id)
        if hd_link:
            print(f"Proceessing: {hd_link}")
            video_name = get_video_name(hd_link)
            model.track(hd_link, save=True, project=f'{HOME_DIR}/week-06-portfolio/results', conf=0.5, iou=0.9,  tracker="bytetrack.yaml", device=device)
            if os.path.exists(f'{HOME_DIR}/{video_name}'):
                os.remove(f'{HOME_DIR}/{video_name}')
    else:
        print(f"Could not extract video ID from URL: {url}")

```

Snipped