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.

```python
def convert_csv_to_yolo(csv_file, images_dir, yolo_labels_dir):
    if not os.path.exists(yolo_labels_dir):
        os.makedirs(yolo_labels_dir)

    # Load the CSV file
    df = pd.read_csv(csv_file)

    for image_file in df['filename'].unique():
        image_annotations = df[df['filename'] == image_file]
        image_path = os.path.join(images_dir, image_file)

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

        # Create YOLO label file
        yolo_label_file = os.path.join(yolo_labels_dir, os.path.splitext(image_file)[0] + '.txt')

        with open(yolo_label_file, 'w') as f:
            for _, row in image_annotations.iterrows():
                # Convert VOC to YOLO format
                xmin, ymin, xmax, ymax = row['xmin'], row['ymin'], row['xmax'], row['ymax']
                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

                # Write the YOLO format
                f.write(f"0 {x_center:.6f} {y_center:.6f} {bbox_width:.6f} {bbox_height:.6f}\n")

        print(f"Converted {image_file} to YOLO format.")
```

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:

```python
# Function to select random images from source_dir and copy them to target_dir
def select_random_images(source_dir, target_dir, num_images=400):
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)
    if not os.path.exists("selected_train_images/labels"): # Target dir for annotations
        os.makedirs("selected_train_images/labels")

    images = [f for f in os.listdir(source_dir) if f.endswith('.jpg') or f.endswith('.JPG')]

    # Check if the number of available images is less than the requested number
    if len(images) < num_images:
        print(f"Only {len(images)} images available, selecting all of them.")
        num_images = len(images)
    selected_images = random.sample(images, num_images)

    for image in selected_images:
        # Copy the image to the target directory
        shutil.copy(os.path.join(source_dir, image), os.path.join(target_dir, image))

        # Check for the corresponding annotation file (YOLO format .txt)
        annotation_file = os.path.splitext(image)[0] + ".txt"
        annotation_src_path = os.path.join("labels/train", annotation_file)                # annotations src dir
        annotation_dst_path = os.path.join("selected_train_images/labels", annotation_file) # annotations dst dir

        if os.path.exists(annotation_src_path):
            shutil.copy(annotation_src_path, annotation_dst_path)
        else:
            print(f"Warning: Annotation file not found for {image}. Skipping {annotation_file}.")
    print(f"Copied {num_images} images to {target_dir} and {num_images} annotations to selected_train_images/labels")
```

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:

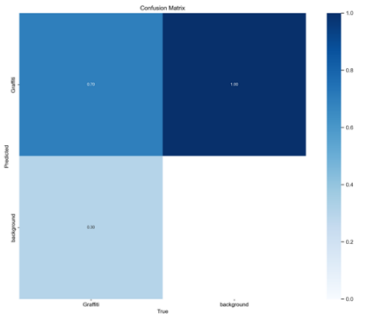
```yaml
train: ../selected_train_images/images  # Path to 400 selected training images
val: ../images/test                     # Path to testing/validation images (used for validation during training)

# Classes
nc: 1               # Number of classes
names: ['Graffiti']  # Class name
```

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

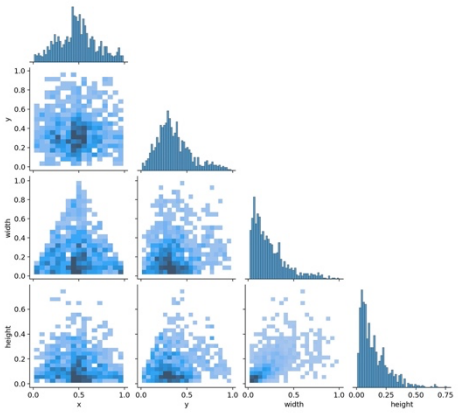
EDA Evaluations:

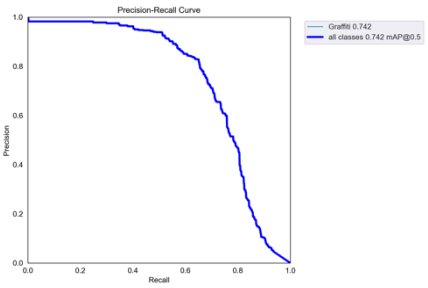
These are some of the EDA results to be evaluated from the training session with yolov5m.pt:

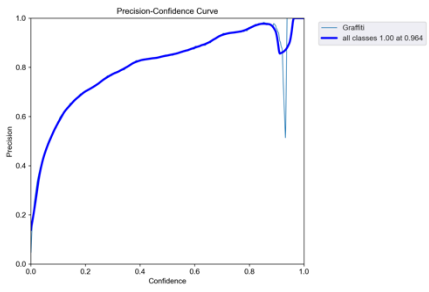
1. confusion_matrix.png

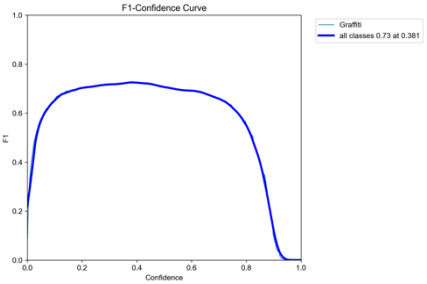


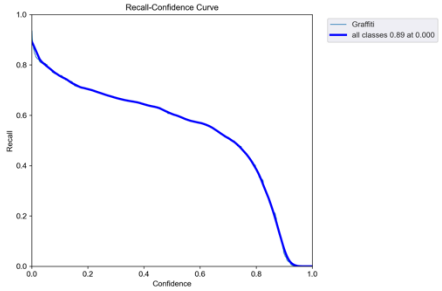
2. labels_correlogram.jpg
3. Confidence level - curves:



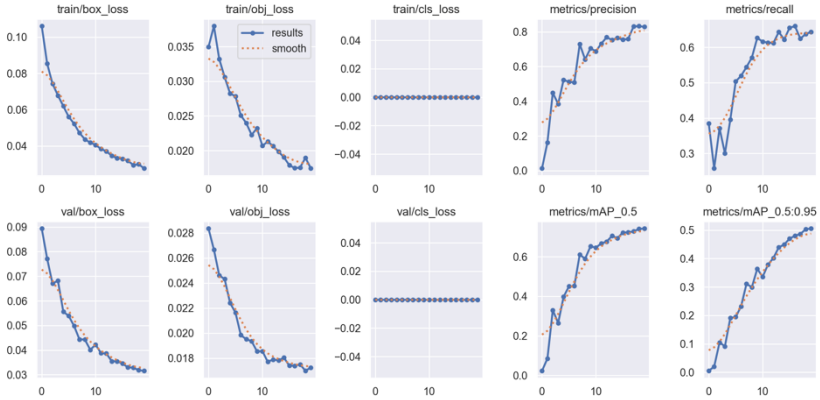




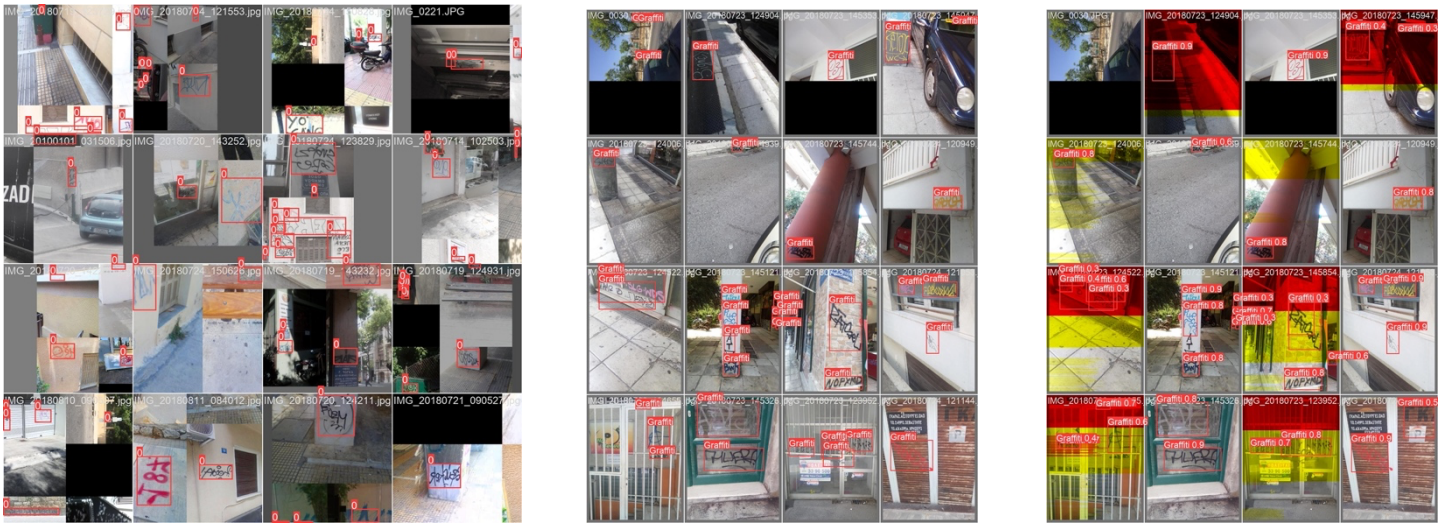




4. Results:



5. Batch 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.

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      14/19         0G    0.03315    0.01903          0         59        640: 100%|██████████| 25/25 [18:30<00
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 7/7
                   all        209        450      0.765      0.622      0.693      0.449

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      15/19         0G     0.0329    0.01789          0         51        640: 100%|██████████| 25/25 [16:43<00
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 7/7
                   all        209        450      0.754      0.655      0.722       0.47

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      16/19         0G    0.03176    0.01748          0         72        640: 100%|██████████| 25/25 [20:04<00
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 7/7
                   all        209        450      0.759       0.66      0.723       0.48

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      17/19         0G    0.02944    0.01755          0         66        640: 100%|██████████| 25/25 [19:05<00
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 7/7
                   all        209        450      0.831      0.624      0.728      0.487

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      18/19         0G    0.02984    0.01896          0         87        640: 100%|██████████| 25/25 [19:10<00
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 7/7
                   all        209        450      0.833      0.638       0.74      0.503

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19         0G    0.02759    0.01744          0         42        640: 100%|██████████| 25/25 [19:35<00
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 7/7
                   all        209        450      0.828      0.644      0.742      0.505

20 epochs completed in 7.754 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 42.2MB
Optimizer stripped from runs/train/exp/weights/best.pt, 42.2MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 212 layers, 20852934 parameters, 0 gradients, 47.9 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 7/7
                   all        209        450      0.829      0.646      0.742      0.506
Results saved to runs/train/exp
```

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:

```python
# Load the YOLOv5 model
model = torch.hub.load('ultralytics/yolov5', 'custom', path='runs/train/exp/weights/best.pt')  # Using pretrained model from step 2

def compute_iou(boxA, boxB):
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    iou = interArea / float(boxAArea + boxBArea - interArea)
    return iou
```

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

2. 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.

```python
# Load YOLO ground truth annotations (test txt files)
def load_yolo_annotation(yolo_label_file, img_width, img_height):
    with open(yolo_label_file, 'r') as file:
        boxes = []
        for line in file.readlines():
            _, x_center, y_center, width, height = map(float, line.strip().split())
            xmin = (x_center - width / 2) * img_width
            ymin = (y_center - height / 2) * img_height
            xmax = (x_center + width / 2) * img_width
            ymax = (y_center + height / 2) * img_height
            boxes.append([xmin, ymin, xmax, ymax])
    return boxes
```

3. 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.

```python
# Evaluate the model on 40 randomly sampled test images
def evaluate_test_images(test_dir, yolo_labels_dir, output_csv, model):
    images = [f for f in os.listdir(test_dir) if f.endswith('.jpg') or f.endswith('.JPG')]
    results = []

    # Convert the unique filenames to a list and randomly sample 40 images
    test_images = random.sample(images, 40)  # Randomly sample 40 test images
    for img_name in test_images:
        img_path = os.path.join(test_dir, img_name)
        img = cv2.imread(img_path)
        img_height, img_width = img.shape[:2]

        # Perform inference using the model
        results_inference = model(img)
        pred_boxes = results_inference.xyxy[0].cpu().numpy()[:, :4]  # Get the predicted boxes
        pred_scores = results_inference.xyxy[0].cpu().numpy()[:, 4]  # Get the confidence scores

        # Get the YOLO ground truth for the image from labels/test
        yolo_label_file = os.path.join(yolo_labels_dir, os.path.splitext(img_name)[0] + '.txt')
        if os.path.exists(yolo_label_file):
            gt_boxes = load_yolo_annotation(yolo_label_file, img_width, img_height)
        else:
            gt_boxes = []  # No annotations if the label file doesn't exist
```

```python
        # Calculate IoU for each prediction
        if len(pred_boxes) == 0:  # No detections
            iou = 0
            confidence = 0
        else:
            ious = []
            confidences = []
            for pred_box, confidence in zip(pred_boxes, pred_scores):
                if gt_boxes:  # Only compute IoU if ground-truth boxes are available
                    ious.append(max([compute_iou(pred_box, gt_box) for gt_box in gt_boxes]))
                else:
                    ious.append(0)  # If no ground truth, IoU is 0
                confidences.append(confidence)

            iou = max(ious)               # Take the maximum IoU
            confidence = max(confidences)  # Take the maximum confidence
        # Append the result
        results.append([img_name, confidence, iou])

    # Save results to CSV
    output_df = pd.DataFrame(results, columns=['image_name', 'confidence_value', 'iou_value'])
    output_df.to_csv(output_csv, index=False)
```

4. Iteration Trainings:

The 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.

```python
# Iterative training and testing until 80% IoU > 90%
def iterative_training_and_testing():
    threshold_met = False
    iteration = 1

    while not threshold_met:
        print(f"Iteration {iteration}: Training and Testing")

        # Step a: Call the script to retrain the model using a new set of training images
        # Make sure the weights of the last model are used for training the next one.
        # Training with 400 images, image size 640x640, batch size 32 (improve generalization), 20 epochs
        # Trained model will be saved at runs/train/exp{n+1}/weights with n to be the iteration ID ranging from exp2
        os.system("python3 train.py --img 640 --batch 32 --epochs 20 --data data.yaml --weights runs/train/exp/weights/best.pt --cache")

        # Step b: Call the evaluation function for 40 test images
        evaluate_test_images("images/test", "labels/test", f"iou_results_iter_{iteration}.csv", model)

        # Step c: Check if 80% of the images have an IoU value > 90%
        iou_results = pd.read_csv(f"iou_results_iter_{iteration}.csv")
        iou_over_90 = (iou_results['iou_value'] > 0.9).sum()
        total_images = len(iou_results)
        percentage_iou_over_90 = (iou_over_90 / total_images) * 100

        print(f"Iteration {iteration}: {percentage_iou_over_90}% images have IoU > 90%")

        # Step d: If 80% of the images have IoU > 90%, stop training
        if percentage_iou_over_90 >= 80:
            iou_results.to_csv("final_model.csv", index=False) # save current model as final model
            print("Threshold met. Stopping further iterations.")
            threshold_met = True
        else:
            print("Threshold not met. Continue the iterations.")
            iteration += 1

# Start the iterative training and testing process
iterative_training_and_testing()
```

5. 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.

6. Model selection:

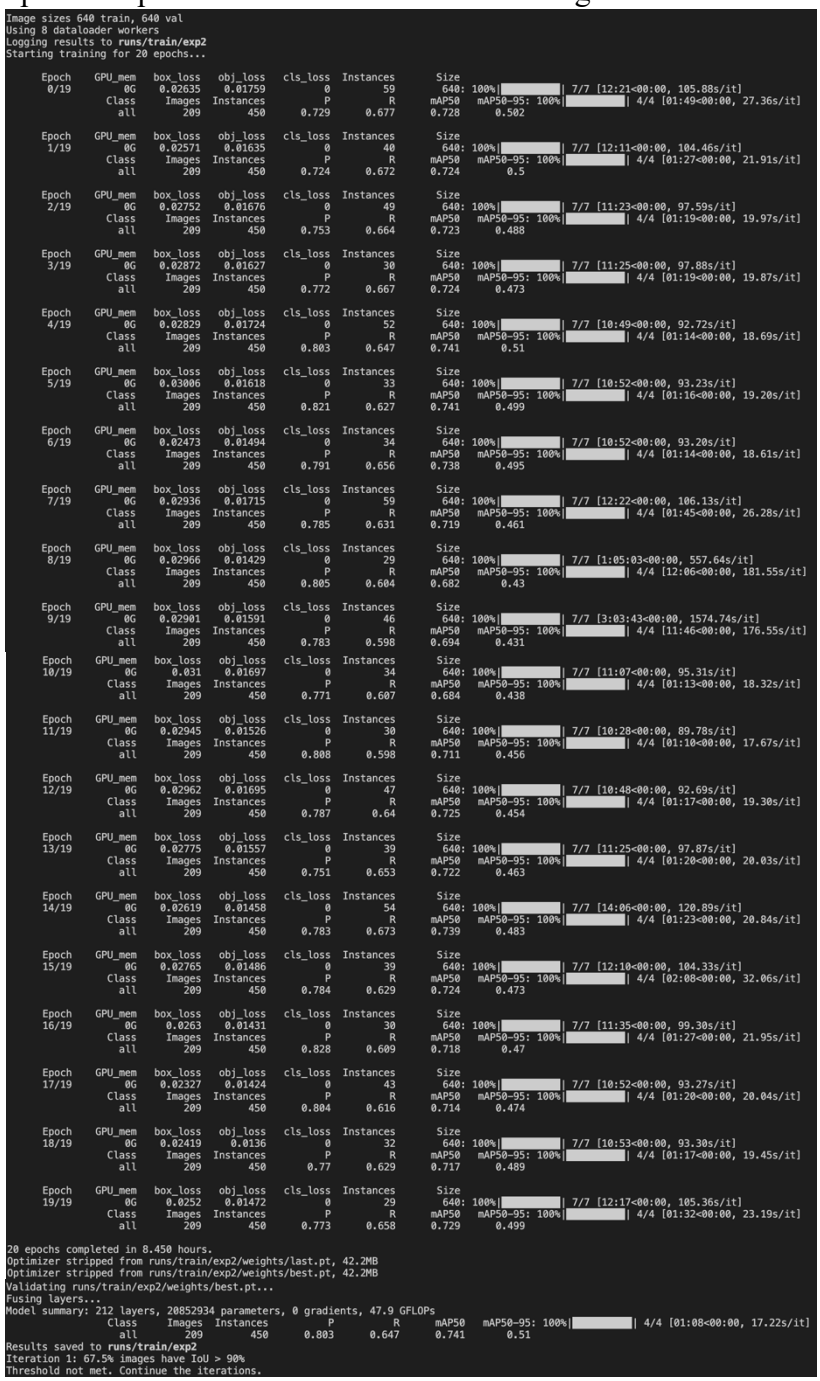
Due to limitation on time resource and CPU health issue, unfortunately, I couldn't iterate the training infinitely and eventually stopped at the 8th iteration, with the total runtime recorded for STEP 3 reached approximately 39 hours.

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:



```
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to runs/train/exp2
Starting training for 20 epochs...

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       0/19         0G    0.02635    0.01759          0         59        640: 100%|██████████| 7/7 [12:21<00:00, 105.88s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:49<00:00, 27.36s/it]
                   all        209        450      0.729      0.677      0.728      0.502

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       1/19         0G    0.02571    0.01635          0         40        640: 100%|██████████| 7/7 [12:11<00:00, 104.46s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:27<00:00, 21.91s/it]
                   all        209        450      0.724      0.672      0.724        0.5

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       2/19         0G    0.02752    0.01676          0         49        640: 100%|██████████| 7/7 [11:23<00:00, 97.59s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:19<00:00, 19.97s/it]
                   all        209        450      0.753      0.664      0.723      0.488

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       3/19         0G    0.02872    0.01627          0         30        640: 100%|██████████| 7/7 [11:25<00:00, 97.88s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:19<00:00, 19.87s/it]
                   all        209        450      0.772      0.667      0.724      0.473

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       4/19         0G    0.02829    0.01724          0         52        640: 100%|██████████| 7/7 [10:49<00:00, 92.72s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:14<00:00, 18.69s/it]
                   all        209        450      0.803      0.647      0.741       0.51

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       5/19         0G    0.03006    0.01618          0         33        640: 100%|██████████| 7/7 [10:52<00:00, 93.23s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:16<00:00, 19.20s/it]
                   all        209        450      0.821      0.627      0.741      0.499

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       6/19         0G    0.02473    0.01494          0         34        640: 100%|██████████| 7/7 [10:52<00:00, 93.20s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:14<00:00, 18.61s/it]
                   all        209        450      0.791      0.656      0.738      0.495

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       7/19         0G    0.02936    0.01715          0         59        640: 100%|██████████| 7/7 [12:22<00:00, 106.13s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:45<00:00, 26.28s/it]
                   all        209        450      0.785      0.631      0.719      0.461

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       8/19         0G    0.02966    0.01429          0         29        640: 100%|██████████| 7/7 [1:05:03<00:00, 557.64s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [12:06<00:00, 181.55s/it]
                   all        209        450      0.805      0.604      0.682       0.43

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
       9/19         0G    0.02901    0.01591          0         46        640: 100%|██████████| 7/7 [3:03:43<00:00, 1574.74s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [11:46<00:00, 176.55s/it]
                   all        209        450      0.783      0.598      0.694      0.431

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      10/19         0G      0.031    0.01697          0         34        640: 100%|██████████| 7/7 [11:07<00:00, 95.31s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:13<00:00, 18.32s/it]
                   all        209        450      0.771      0.607      0.684      0.438

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      11/19         0G    0.02945    0.01526          0         30        640: 100%|██████████| 7/7 [10:28<00:00, 89.78s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:10<00:00, 17.67s/it]
                   all        209        450      0.808      0.598      0.711      0.456

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      12/19         0G    0.02962    0.01695          0         47        640: 100%|██████████| 7/7 [10:48<00:00, 92.69s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:17<00:00, 19.30s/it]
                   all        209        450      0.787       0.64      0.725      0.454

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      13/19         0G    0.02775    0.01557          0         39        640: 100%|██████████| 7/7 [11:25<00:00, 97.87s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:20<00:00, 20.03s/it]
                   all        209        450      0.751      0.653      0.722      0.463

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      14/19         0G    0.02619    0.01458          0         54        640: 100%|██████████| 7/7 [14:06<00:00, 120.89s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:23<00:00, 20.84s/it]
                   all        209        450      0.783      0.673      0.739      0.483

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      15/19         0G    0.02765    0.01486          0         39        640: 100%|██████████| 7/7 [12:10<00:00, 104.33s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [02:08<00:00, 32.06s/it]
                   all        209        450      0.784      0.629      0.724      0.473

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      16/19         0G     0.0263    0.01431          0         30        640: 100%|██████████| 7/7 [11:35<00:00, 99.30s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:27<00:00, 21.95s/it]
                   all        209        450      0.828      0.609      0.718       0.47

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      17/19         0G    0.02327    0.01424          0         43        640: 100%|██████████| 7/7 [10:52<00:00, 93.27s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:20<00:00, 20.04s/it]
                   all        209        450      0.804      0.616      0.714      0.474

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      18/19         0G    0.02419     0.0136          0         32        640: 100%|██████████| 7/7 [10:53<00:00, 93.30s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:17<00:00, 19.45s/it]
                   all        209        450       0.77      0.629      0.717      0.489

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19         0G     0.0252    0.01472          0         29        640: 100%|██████████| 7/7 [12:17<00:00, 105.36s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:32<00:00, 23.19s/it]
                   all        209        450      0.773      0.658      0.729      0.499

20 epochs completed in 8.450 hours.
Optimizer stripped from runs/train/exp2/weights/last.pt, 42.2MB
Optimizer stripped from runs/train/exp2/weights/best.pt, 42.2MB
Validating runs/train/exp2/weights/best.pt...
Fusing layers...
Model summary: 212 layers, 20852934 parameters, 0 gradients, 47.9 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 4/4 [01:08<00:00, 17.22s/it]
                   all        209        450      0.803      0.647      0.741       0.51
Results saved to runs/train/exp2
Iteration 1: 67.5% images have IoU > 90%
Threshold not met. Continue the iterations.
```

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**.

2. **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.

3. **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.

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

# Get video frame width, height, and frames per second (fps)
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = cap.get(cv2.CAP_PROP_FPS)

# Define video writer to save output video
out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width, frame_height))

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

    # Perform inference on the current frame
    results = model(frame)
    # Draw bounding boxes and labels on the frame
    results.render()
    frame_with_detections = results.imgs[0]
    # Write the frame with detections to the output video
    out.write(frame_with_detections)
```

4. **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.

```
# Display the frame (optional for real-time visualization)
cv2.imshow('Graffiti Detection', frame_with_detections)

# Exit if 'q' is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

5. **Output**:

Processed videos with graffiti detections are saved in the output_videos folder.

```
# Process each video in the live_video directory
for video_file in os.listdir(video_dir):
    if video_file.endswith(('.mp4', '.avi', '.mov')): # Handling different video extension formats
        video_path = os.path.join(video_dir, video_file)
        output_path = os.path.join(output_dir, f"output_{video_file}")
        print(f"Processing {video_file}...")
        detect_graffiti_in_video(video_path, output_path)
        print(f"Finished processing {video_file}. Saved to {output_path}")

print("All videos processed. Graffiti detection complete.")
```

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.

Example 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