22-Sept-24

PORTFOLIO

WEEK 6 - DOCUMENTATION

Tran Duc Anh Dang | 103995439
STUDIO CLASS: STUDIO 1-3

Abstract

This portfolio submission presents the development and evaluation of a deep learning model using YOLO v5 for graffiti detection in images and real time video data. The project demonstrates a comprehensive understanding of data preprocessing, model training, evaluation metrics, iterative optimization, and deployment using PyTorch. The submission includes labeled datasets, model outcomes, and source code, organized as follows:

- Annotation Conversion Function: A custom function convert_annotations is implemented to transform the provided annotation format in training labels to the YOLO annotation format. This function ensures that bounding boxes are correctly normalized and formatted, facilitating seamless integration with the YOLO v5 training pipeline.
- YOLO v5 Model Training: The YOLO v5 model is trained using 400 randomly selected images from the training dataset. This training process leverages the converted annotations to enable the model to accurately detect graffiti within images. The training is performed iteratively, with each iteration refining the model's performance. The trained models from each iteration are saved as best.pt files within the **week-06-portfolio/train/runs/train/graffiti_detection_iter_X/weights/** directories, where X denotes the iteration number.
- IoU Computation and Evaluation: 40 randomly selected images from the test dataset are used to evaluate the model's performance. For each test image, the Intersection over Union (IoU) is computed to assess the accuracy of the detected bounding boxes against the ground truth. The evaluation results, comprising image_name, confidence_value, and IoU_value, are compiled into CSV files. Images with no detected graffiti are assigned an IoU value of 0. These results are stored within the **train/evaluation_images_iter_X** directories, corresponding to each training iteration.
- Iterative Training and Optimization: An iterative training process is employed where the YOLO v5 model is retrained with new sets of 400 training and 40 test images in each iteration. This process continues until 80% of the test images achieve an IoU over 90%, or all images have been utilized for training and testing. Each iteration uses the model from the previous step as the pretrained model, facilitating progressive learning and performance enhancement. The outcomes of each iteration, including CSV files and sample annotated images, are organized within their respective iteration folders under the train directory.
- Real Time Video Detection: The final optimized YOLO v5 model is deployed to detect graffiti in real time video data. The model processes various video inputs, identifying and annotating graffiti instances with bounding boxes and confidence scores. Example video sources from Pexels are utilized to demonstrate the model's real-time detection capabilities. The detection results are saved within the results directory, organized into subfolders such **as track, track2, etc**., corresponding to each video track.

Repository Structure and Access
All project requirements, documentation, source code, and results are organized within the week-06-portfolio repository on GitHub. The following links provide access to each component:

Requirements: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/requirements

Documentation: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/docs

Source Code: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/code

YAML Config: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/train/yaml

YOLO v5 Model Training and Results (train): https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/train

Evaluation Images: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/train/evaluation_images_iter_**X** **(I have 30 iter you may want to replace X with number from 0 to 30)**

Evaluation Results CSV: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/train/evaluation_results_iter_X.csv **(I have 30 iter you may want to replace X with number from 0 to 30)**

YOLO v5 Best Model on each Iteration: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/train/runs/train/graffiti_detection_iter_**X**/weights **(I have 30 iter you may want to replace X with number from 0 to 30)**

Detection Results (results): https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-06-portfolio/results

Table of Contents

Deep Learning Using Yolo Models

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

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


```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

To train and create a YOLO model using 400 randomly selected training images for graffiti detection, my code follows these steps:

1. Selecting Random Training Image

week-06-portfolio-soln.ipynb

```
# 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)
```

Snipped

2. Creating the YAML Configuration File

week-06-portfolio-soln.ipynb

```
# 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)
```

Snipped

3. Training the YOLO Model

week-06-portfolio-soln.ipynb

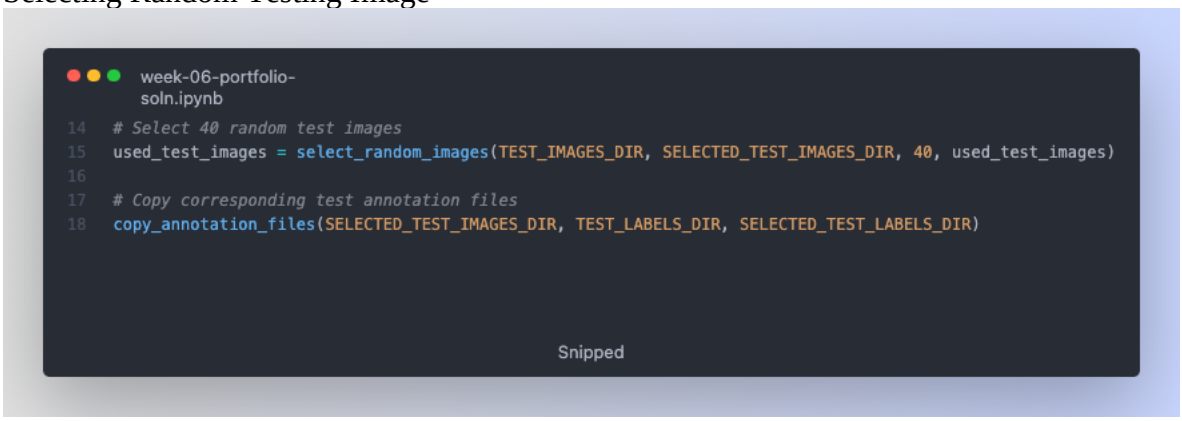
```
# Load a pretrained YOLOv5s model
model = YOLO('week-06-portfolio/models/yolov5su.pt')

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

Snipped

3) Randomly take 40 images from test data and compute IoU for each and generate a 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 evaluate the trained YOLO model's performance on 40 randomly selected test images and compute the Intersection over Union (IoU) for each, your code performs the following:

1. Selecting Random Testing Image



2. Evaluating the Model and Computing IoU

week-06-portfolio-soln.ipynb

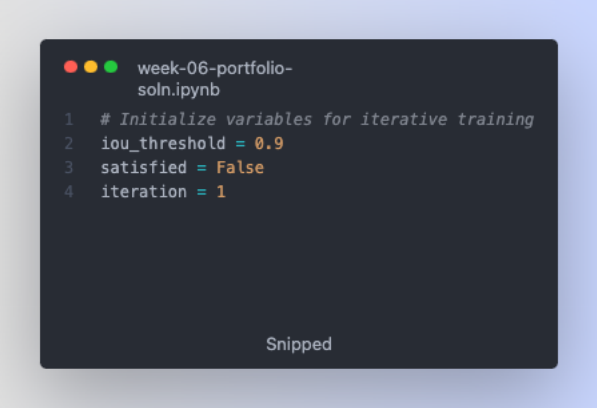
```python
def compute_iou(pred_boxes, true_boxes):
    # Convert boxes to tensors
    pred_boxes = torch.tensor(pred_boxes)
    true_boxes = torch.tensor(true_boxes)

    # 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) Until IoU value of 80% images in your test data is over 90% or all images are utillised for training and testing purpose, you need to iteratively train and test the model with a new set of 400 training and 40 test images. Make sure you use the model of previous iteration as the pre-trained model for new iteration.

To iteratively train and test the YOLO model until at least 80% of the test images achieve an IoU greater than 90%, I implements a while-loop that continues this process until the stopping criteria are met. Here's a comprehensive breakdown of how this is achieved:

1. Initialization the threshold



2. Iterative Training and Evaluation Loop

week-06-portfolio-soln.ipynb

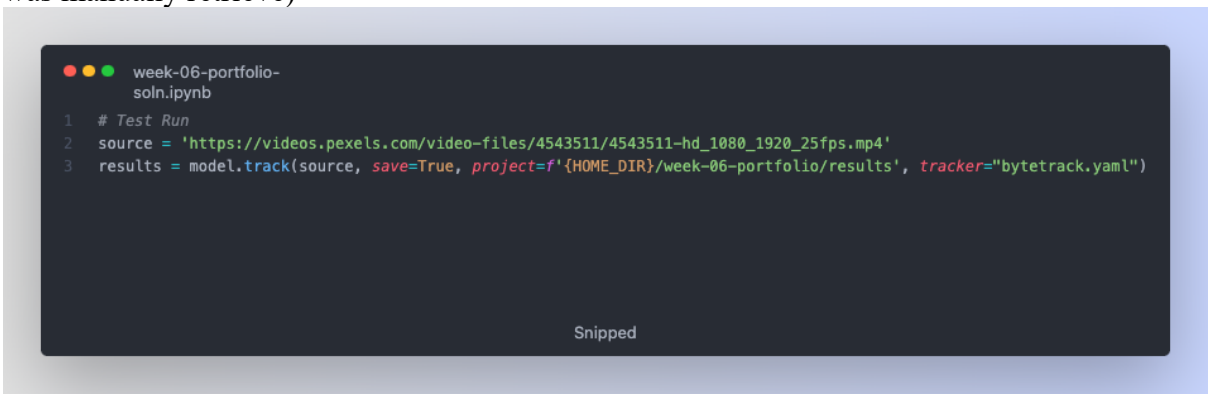
```
while not satisfied:
    print(f"\nStarting iteration {iteration}")

    # 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
```

Snipped

5) Use your final model to detect graffiti in real-time video data.
To deploy the final trained YOLO model for real-time graffiti detection in video data, my code performs the following steps:

1. Load the trained model

2. Testing on sample video from pexels providing in the requirements (the URL in here was manually retrieve)



3. Retrieve video from Pexels through its API
    a. Function to extracting video id from provided URL in requirements

b. Function to retrieve the video stream url from Pexels

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. Retrieve video and use model to predict and tracking

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