SCHOOL OF SCIENCE, COMPUTING AND ENGINEERING TECHNOLOGIES

COS40007

Artificial Intelligence for Engineering

Portfolio Assessment-5: "Deep learning using YOLO v5'"

Studio 6 – Portfolio 6

Students' Name: Duc Thinh Pham - 104169675

Lecturer: Dr. Trung Luu

Date: October 20th, 2024

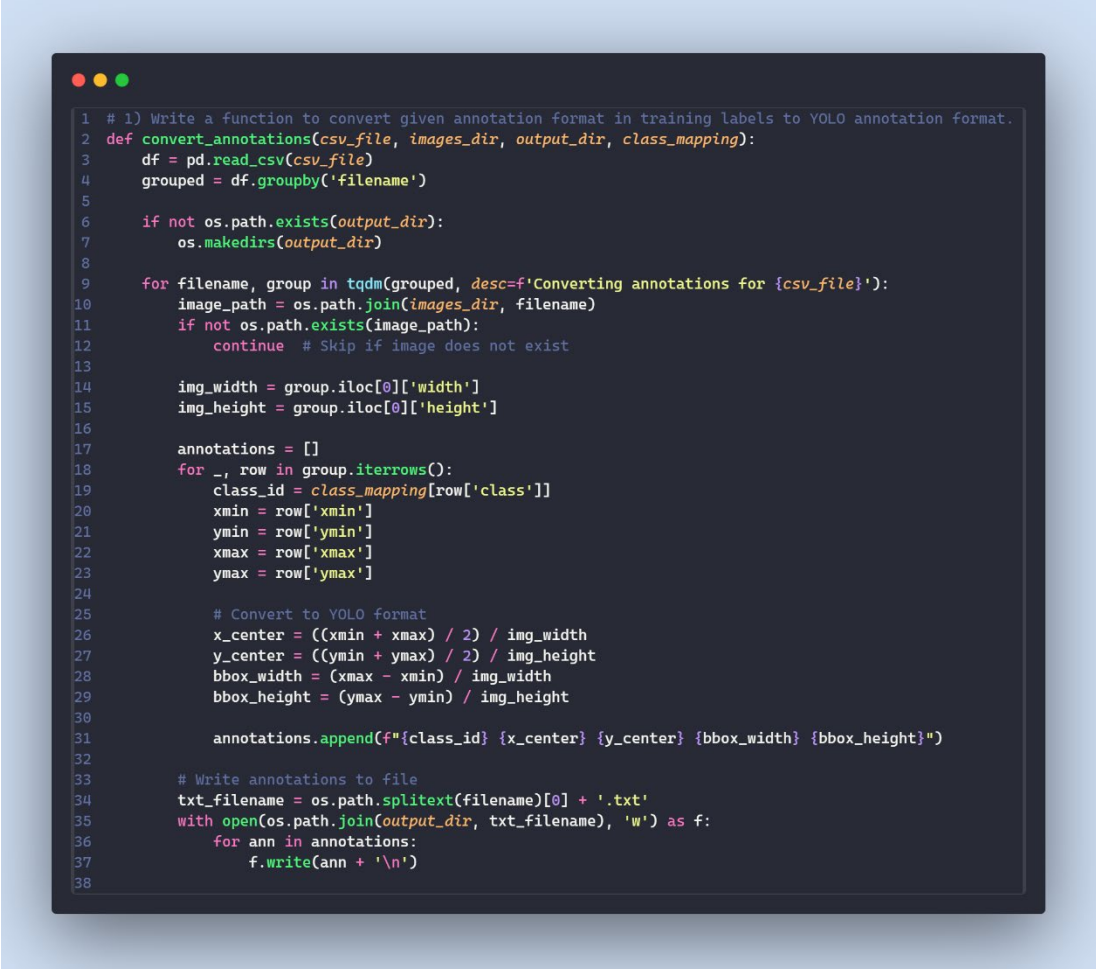
Contents

Abstract

This portfolio demonstrates the development of a deep learning model using YOLOv5 and PyTorch for graffiti detection. The primary goal is to iteratively train a YOLO model using a subset of images to detect graffiti while optimizing performance through the Intersection over Union (IoU) metric. The model is trained using 400 random images from the training set in each iteration, and tested on 40 random images, with a CSV file generated to record image names, confidence values, and IoU values. The process is repeated until 80% of test images achieve an IoU value over 90%, or until the entire dataset is utilized. Furthermore, the final model is applied to real-time video data for graffiti detection. The portfolio includes code for data annotation conversion, iterative model training, and evaluation, along with detection results on video files.

Developing deep learning model using YOLO v5 and Pytorch

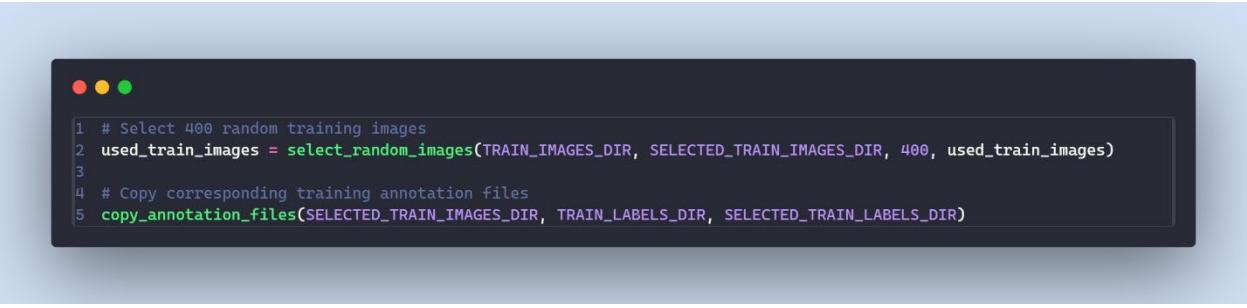
1. **Write a function to convert given annotation format in training labels to YOLO annotation format.**



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

2.1. Select randomly 400 images from train data



```python
# 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)
```

2.2. Generate a YAML configuration file for a YOLO training process.

```python
def create_yaml_file(file_path, train_images, val_images, nc, class_names):
    data = {
        'train': train_images,
        'val': val_images,
        'nc': nc,
        'names': class_names
    }
    with open(file_path, 'w') as f:
        for key, value in data.items():
            if isinstance(value, list):
                value_str = str(value).replace("'", "")
                f.write(f"{key}: {value_str}\n")
            else:
                f.write(f"{key}: {value}\n")
```

2.3. Train YOLO model

```python
# Load a pretrained YOLOv5s model
model = YOLO(f'{WEEK06_DIR}/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. 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.

3.1. Select randomly 40 images from test data

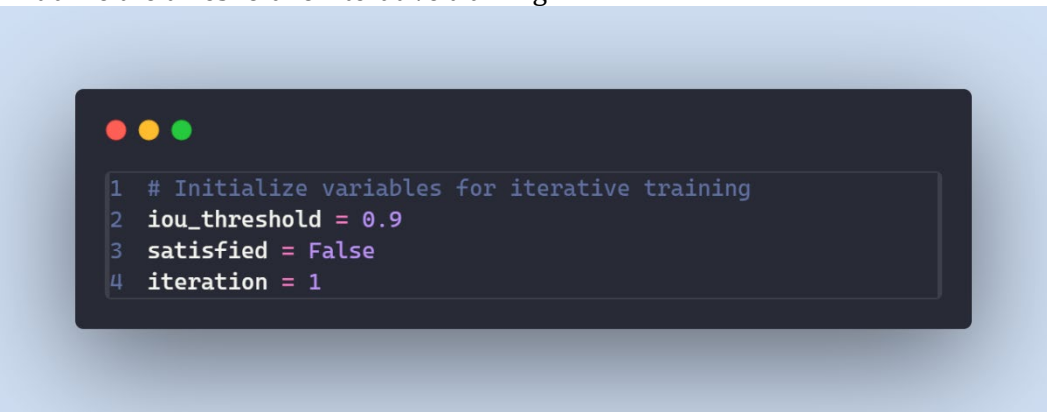
```python
# Select 40 random test images
used_test_images = select_random_images(TEST_IMAGES_DIR, SELECTED_TEST_IMAGES_DIR, 40, used_test_images)

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

3.2. Evaluate YOLO model and compute IoU

```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)

```

4. **Until IoU value of 80% images in your test data is over 90% or all images are utilized 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.**

4.1. Initialize the threshold for iterative training



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

4.2 Train and evaluate loop

```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 = os.path.join(WEEK06_DIR, f'images/train_selected_iter_{iteration}')
    selected_train_labels_dir = os.path.join(WEEK06_DIR, f'labels/train_selected_iter_{iteration}')

    os.makedirs(selected_train_images_dir, exist_ok=True)
    os.makedirs(selected_train_labels_dir, exist_ok=True)

    # 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 paths
    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 = os.path.join(WEEK06_DIR, 'yaml')
    os.makedirs(yaml_dir, exist_ok=True)

    yaml_file_path_iter = os.path.join(WEEK06_DIR, f'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 the previous iteration or start with the pre-trained model
    if iteration == 1:
        model = YOLO(os.path.join(WEEK06_DIR, 'models', 'yolov5su.pt'))
# Start with pre-trained YOLOv5su model
    else:
        previous_model_path = os.path.join(WEEK06_DIR, 'runs', 'train', f'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=os.path.join(WEEK06_DIR, 'runs', 'train'),
        name=f'graffiti_detection_iter_{iteration}',
        device=device
    )

    # Evaluate the model
    output_images_dir_iter = os.path.join(WEEK06_DIR, f'evaluation_images_iter_{iteration}')
    os.makedirs(output_images_dir_iter, exist_ok=True)

    df_results = evaluate_model(model, SELECTED_TEST_IMAGES_DIR, SELECTED_TEST_LABELS_DIR,
output_images_dir_iter)
    df_results.to_csv(os.path.join(WEEK06_DIR, f'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(os.path.join(WEEK06_DIR, 'models', f'yolov5s_graffiti_iter_{iteration}.pt'))
        satisfied = True
    else:
        print(f"IoU threshold not met in iteration {iteration}")

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

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

5.1. Load the best trained model

```
model = YOLO(f'{WEEK06_DIR}/train/graffiti_detection_iter_30/weights/best.pt')
```

5.2. Fetch video from Pexels using its API

```
# 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
```

5.3. Fetch the video and apply the model for prediction

```
# 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/"
]


# Predict 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'{WEEK06_DIR}/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}")
```

Appendix

My GitHub repository: thinhpham1807/COS40007---Artificial-Intelligence-for-Engineering (github.com)

Submission folder:
https://drive.google.com/drive/folders/1CVvGfAItkBKUBpUSdN3uE5NhuYuFgymw?usp=drive_link

1. Code for step 1 -
https://drive.google.com/drive/folders/1lsUKSxrDJk85IDKP9ILPCwxdoXRDGqvs?usp=drive_link

2. The best.pt model of each iteration -
https://drive.google.com/drive/folders/1cDfL7MiMDe92b0k0kBVe1OoxK9RdTUW7?usp=drive_link

3. The CSV file of outcome for each iteration, and 2 good sample of detected images with

bounding box. Separate by folder for each iteration -
https://drive.google.com/drive/folders/1Ka5FvUi1Dz5W8wJr11dPELNErkWVE2j?usp=drive_link

4. Detection outcomes of 5 videos in (5) - https://drive.google.com/drive/folders/1RrWrYXF_PsQN3-aWQePZceHPmlbGpC6r?usp=drive_link