Applied Computer Vision Intern – Assignment

1. Write a python script to convert the annotations from PascalVOC format to yolov8 format.

```
import os
import xml.etree.ElementTree as et
import argparse

def normalize_bbox(width, height, data):
    box_width = data[2] - data[0]
    box_height = data[3] - data[1]

    x_center = (data[0] + data[2]) / 2
    y_center = (data[1] + data[3]) / 2

    x_norm = x_center / width
    y_norm = y_center / height
    box_width_norm = box_width / width
    box_height_norm = box_height / height

    return x_norm, y_norm, box_width_norm, box_height_norm

def extract_annotations(xml_file, output_dir, class_map):
    print(f"Processing file: {xml_file}")
    tree = et.parse(xml_file)
    root = tree.getroot()

    size = root.find('size')
    width = int(size.find('width').text)
    height = int(size.find('height').text)

    output_file = os.path.join(output_dir,
root.find('filename').text.replace('.jpg', '.txt').replace('.png', '.txt'))

    with open(output_file, 'w') as out:
        for obj in root.iter('object'):
            name = obj.find('name').text
            if name not in class_map:
                continue
```

```
            id = class_map[name]
            box = obj.find('bndbox')
            data = (
                float(box.find('xmin').text),
                float(box.find('ymin').text),
                float(box.find('xmax').text),
                float(box.find('ymax').text)
            )
            bb = normalize_bbox(width, height, data)
            out.write(f"{id} " + " ".join([f"{a:.6f}" for a in bb]) + '\n')

def xml_to_txt(xml_dir, output_dir, class_map):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for filename in os.listdir(xml_dir):
        if filename.endswith('.xml'):
            xml_file = os.path.join(xml_dir, filename)
            extract_annotations(xml_file, output_dir, class_map)

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description="Converting data from PascalVOC
to YoloV8 format")
    parser.add_argument('--xml_dir',type=str,required=True,help="Directory
containing the PascalVOC format files")
    parser.add_argument('--output_dir',type=str,required=True,help="Directory
containing the YoloV8 format files")
    parser.add_argument('--class_path', type=str, required=True, help='Path to
the class mapping file')

    args = parser.parse_args()

    class_map = {}
    with open(args.class_path, 'r') as file:
        for id, line in enumerate(file):
            class_map[line.strip()] = id

    xml_to_txt(args.xml_dir, args.output_dir, class_map)

    print('Completed')
```

This code should be executed at the command line where three input paths should be given to transform from PascalVOC to Yolov8 format. The PascalVOC file is in .xml format which should be converted to .txt format of Yolov8. The three inputs are input label directory, output label directory and a class file which contains the number of different classes.

From each PascalVOC file, some annotations must be extracted for further detection into text file containing the bounding box coordinates along with I.D for each class. Also, these bounding boxes must be normalized before converting them into text file.
Few libraries like os, xml.etree.ElementTree and argparse are used for further processing.

2. Train yolov8 object detection model for person detection.

```
import os
import shutil
!pip install ultralytics
from ultralytics import YOLO as yolo
from sklearn.model_selection import train_test_split
import cv2
import matplotlib.pyplot as plt
```

Importing required libraries.

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounting the drive to access files.

```
weights_dir = "/content/drive/MyDrive/Colab Notebooks/Internship Works/weights"

if not os.path.exists(weights_dir):
    os.makedirs(weights_dir)
    print("Directory created")
else:
    print("Directory already created")
```

Creating a weights directory to store the weights of the model.

```
train_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/datasets/train"
val_dir = "/content/drive/MyDrive/Colab Notebooks/Internship Works/datasets/val"
test_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/datasets/test"
```

```
def dest(dest_dir):
    subdirect = ["images","labels"]
    for split in subdirect:
        full_path = os.path.join(dest_dir,split)
        if not os.path.exists(full_path):
            os.makedirs(full_path)
            print("Directories created")
        else:
            print("Directories already created")

#Source directory
images_path = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/datasets/images"
labels_path = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/datasets/labels_yolo"
images_dir = os.listdir(images_path)
labels_dir = os.listdir(labels_path)

trainval_images, test_images, trainval_labels, test_labels =
train_test_split(images_dir, labels_dir, test_size=0.2, random_state=42)
train_images, val_images, train_labels, val_labels =
train_test_split(trainval_images, trainval_labels, test_size=0.25,
random_state=42)

#Move the data
def move(files,source_dir,out_dir, subdir):
    dest_path = os.path.join(out_dir,subdir)
    for file in files:
        source_file = os.path.join(source_dir,file)
        destin_file = os.path.join(dest_path,file)
        os.rename(source_file,destin_file)
        print("moved")

dest(train_dir)
dest(val_dir)
dest(test_dir)

move(train_images,images_path,train_dir,"images")
move(train_labels,labels_path,train_dir,"labels")
move(val_images,images_path,val_dir,"images")
move(val_labels,labels_path,val_dir,"labels")
move(test_images,images_path,test_dir,"images")
move(test_labels,labels_path,test_dir,"labels")
```

Spliting the image and label data into training data, validation data and test data.

```
model = yolo("yolov8n.yaml")
model = yolo("yolov8n.pt")
model = yolo("yolov8n.yaml").load("yolov8n.pt")

results = model.train(data="/contents/drive/MyDrive/Colab Notebooks/Internship
Works/scripts/person.yaml", epochs=10, imgsz=640)
```

Train the model using the pre-trained Yolo models

```
best_wt = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/scripts/runs/detect/train12/weights/best.pt"
weigh = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/weights/person_detection.pt"
if not os.path.exists(best_wt):
    print("File not found")
else:
    shutil.move(best_wt,weigh)
```

Moving the new weights to the weight directory.

```
model = yolo(weigh)
results = model.predict("/content/drive/MyDrive/Colab Notebooks/Internship
Works/datasets/test/images/001029.jpg",save=True)
```

Testing the model.

```
predicted_bboxes = []
for result in results:
    for box in result.boxes:
        x_min, y_min, x_max, y_max = box.xyxy[0].tolist()  # Bounding box
coordinates
        confidence = box.conf[0].item()                    # Confidence score
        label = int(box.cls[0].item())                     # Class label
        predicted_bboxes.append((label, x_min, y_min, x_max, y_max, confidence))
print(predicted_bboxes)
```

Displaying the confidence score of the prediction

[(0, 70.4520263671875, 27.337087631225586, 396.0247497558594, 365.0, 0.9295008778572083)]

3. Train another yolov8 object detection model for PPE detection on cropped person Images.

```
import os
import shutil
import cv2
import xml.etree.ElementTree as et
from sklearn.model_selection import train_test_split
!pip install ultralytics
from ultralytics import YOLO as yolo
```

Importing required libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounting the drive to access files

```
def normalize(crop_width, crop_height, data):
    box_width = data[2] - data[0]
    box_height = data[3] - data[1]
    x_center = (data[0] + data[2]) / 2
    y_center = (data[1] + data[3]) / 2
    x_norm = x_center / crop_width
    y_norm = y_center / crop_height
    box_width_norm = box_width / crop_width
    box_height_norm = box_height / crop_height
    return x_norm, y_norm, box_width_norm, box_height_norm
```

```
def crop_and_adjust_annotations(image_file, annotation_file, cropped_image_dir,
new_label_dir, class_map):
    image_path = os.path.join(source_dir, image_file)
    img = cv2.imread(image_path)

    if img is None:
        print(f"Error reading image {image_file}")
        return

    if not os.path.exists(annotation_file):
        print(f"Annotation file {annotation_file} not found")
        return

    tree = et.parse(annotation_file)
    root = tree.getroot()

    person_id = 1
    for obj in root.findall("object"):
        name = obj.find("name").text
        bbox = obj.find("bndbox")

        if name == "person":
            xmin = int(float(bbox.find("xmin").text))
            ymin = int(float(bbox.find("ymin").text))
            xmax = int(float(bbox.find("xmax").text))
            ymax = int(float(bbox.find("ymax").text))

            new_img = img[ymin:ymax, xmin:xmax]  # Crop the person region
            new_img_filename =
f"person_{person_id}_{os.path.basename(image_file)}"
            new_image_path = os.path.join(cropped_image_dir, new_img_filename)
            cv2.imwrite(new_image_path, new_img)

            print(f"Person {person_id} cropped and saved at {new_image_path}")
            new_label_filename =
f"person_{person_id}_{os.path.basename(annotation_file)}"
            new_label_path = os.path.join(new_label_dir,
new_label_filename.replace(".xml", ".txt"))

            # Open label file in append mode to avoid overwriting
            with open(new_label_path, 'a') as out:
```

```
                for obj_ppe in root.findall("object"):
                    ppe_name = obj_ppe.find("name").text
                    ppe_bbox = obj_ppe.find("bndbox")
                    if ppe_name in class_map:
                        ppe_xmin = int(float(ppe_bbox.find("xmin").text))
                        ppe_ymin = int(float(ppe_bbox.find("ymin").text))
                        ppe_xmax = int(float(ppe_bbox.find("xmax").text))
                        ppe_ymax = int(float(ppe_bbox.find("ymax").text))

                        # Check if the PPE bounding box lies within the person
bounding box
                        if ppe_xmin >= xmin and ppe_xmax <= xmax and ppe_ymin >=
ymin and ppe_ymax <= ymax:
                            adjusted_xmin = ppe_xmin - xmin
                            adjusted_ymin = ppe_ymin - ymin
                            adjusted_xmax = ppe_xmax - xmin
                            adjusted_ymax = ppe_ymax - ymin

                            # Normalize the bounding box to the cropped person
image
                            data = (adjusted_xmin, adjusted_ymin, adjusted_xmax,
adjusted_ymax)
                            new_data = normalize(xmax - xmin, ymax - ymin, data)

                            # Write normalized data to the label file
                            out.write(f"{class_map[ppe_name]} " + "
".join(f"{a:.6f}" for a in new_data) + "\n")

            person_id += 1



if __name__ == "__main__":
    source_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/images"
    label_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/labels"
    cropped_image_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/cropped/images"
    new_label_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/cropped/labels"

    os.makedirs(cropped_image_dir,exist_ok=True)
```

```
    os.makedirs(new_label_dir,exist_ok=True)

    class_ppe = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/classes_ppe.txt"
    class_map = {}
    with open(class_ppe,'r') as file:
        for id, line in enumerate(file):
            class_map[line.strip()] = id

    for image_file in os.listdir(source_dir):
        annotation_file =
os.path.join(label_dir,image_file.replace(".jpg",".xml"))
        crop_and_adjust_annotations(image_file, annotation_file,
cropped_image_dir, new_label_dir, class_map)
```

Cropping each full image based on each person and for each cropped image a new annotation file is created containing other classes including ppe suit. The bounding box coordinates are adjusted for each cropped image so that all other classes are within the cropped image.

```
def count_files(directory):
    # Get a list of all items (files and directories) in the specified directory
    all_items = os.listdir(directory)

    # Filter only the files
    files = [f for f in all_items if os.path.isfile(os.path.join(directory, f))]

    return len(files)

print(count_files(cropped_image_dir))
print(count_files(new_label_dir))
```

Counting all cropped images and new annotation files so that both must be equal.

```
train_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/train"
val_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/val"
test_dir = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/test"

def dest(dest_dir):
    subdirect = ["images","labels"]
    for split in subdirect:
```

```
        full_path = os.path.join(dest_dir,split)
        if not os.path.exists(full_path):
            os.makedirs(full_path)
            print("Directories created")
        else:
            print("Directories already created")

#Source directory
images_dir = os.listdir(cropped_image_dir)
labels_dir = os.listdir(new_label_dir)

trainval_images, test_images, trainval_labels, test_labels =
train_test_split(images_dir, labels_dir, test_size=0.2, random_state=42)
train_images, val_images, train_labels, val_labels =
train_test_split(trainval_images, trainval_labels, test_size=0.25,
random_state=42)

#Move the data
def move(files,source_dir,out_dir, subdir):
    dest_path = os.path.join(out_dir,subdir)
    for file in files:
        source_file = os.path.join(source_dir,file)
        destin_file = os.path.join(dest_path,file)
        os.rename(source_file,destin_file)
        print("moved")

dest(train_dir)
dest(val_dir)
dest(test_dir)

move(train_images,cropped_image_dir,train_dir,"images")
move(train_labels,new_label_dir,train_dir,"labels")
move(val_images,cropped_image_dir,val_dir,"images")
move(val_labels,new_label_dir,val_dir,"labels")
move(test_images,cropped_image_dir,test_dir,"images")
move(test_labels,new_label_dir,test_dir,"labels")
```

Splitting each cropped image and annotation files into training data, validation data, test data.

```
model = yolo("yolov8n.yaml")
model = yolo("yolov8n.pt")
model = yolo("yolov8n.yaml").load("yolov8n.pt")
```

```
results = model.train(data="/content/drive/MyDrive/Colab Notebooks/Internship
Works/scripts/ppe.yaml", epochs=10, imgsz=640)
```

Training a new model for ppe detection

```
weigh = "/content/drive/MyDrive/Colab Notebooks/Internship
Works/weights/ppe_detection.pt"
model = yolo(weigh)
results = model.predict("/content/drive/MyDrive/Colab Notebooks/Internship
Works/ppe_datasets/test/images/person_1_001060.jpg",save=True)
```

Saving the new weights to the weights directory and testing the model.


```
predicted_bboxes = []
for result in results:
    for box in result.boxes:
        x_min, y_min, x_max, y_max = box.xyxy[0].tolist()  # Bounding box
coordinates
        confidence = box.conf[0].item()                    # Confidence score
        label = int(box.cls[0].item())                     # Class label
        predicted_bboxes.append((label, x_min, y_min, x_max, y_max, confidence))
print(predicted_bboxes)
```

Displaying the confidence scores

[(0, 12.125648498535156, 0.0, 56.050148010253906, 38.078521728515625, 0.8300375938415527), (4, 55.88288497924805, 233.68191528320312, 88.47761535644531, 249.6482391357422, 0.5699059367179871)]

4. Write the flow which will take an image directory as input, perform inference through both the models and save them in another directory (inference.py).

   And also

5. For drawing the predicted bounding boxes and confidence, please use opencv's cv2.rectangle() and cv2.puttext() and NOT yolo's inbuilt function for drawing.

```
import os
import cv2
import argparse
from ultralytics import YOLO

def draw_bounding_box(image, bbox, label, color=(0, 255, 0), thickness=2):
    """Draw bounding boxes and add labels on the image."""
    x_min, y_min, x_max, y_max = bbox
    cv2.rectangle(image, (x_min, y_min), (x_max, y_max), color, thickness)
    cv2.putText(image, label, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)

def run_person_detection(model, image):
    """Run the person detection model on the image."""
    results = model(image)
    person_bboxes = []
    for pred in results[0].boxes:  # Accessing the boxes in the first result
        if int(pred.cls[0]) == 0:  # Assuming class 0 is 'person'
            x_min, y_min, x_max, y_max = map(int, pred.xyxy[0])  # Convert to
integers
            person_bboxes.append([x_min, y_min, x_max, y_max])
    return person_bboxes

def run_ppe_detection(model, cropped_image):
    """Run the PPE detection model on the cropped person image."""
    results = model(cropped_image)
    ppe_bboxes = []
    for pred in results[0].boxes:
        label = int(pred.cls[0])  # Class label
        x_min, y_min, x_max, y_max = map(int, pred.xyxy[0])  # Bounding box
coordinates
        ppe_bboxes.append((label, [x_min, y_min, x_max, y_max]))
    return ppe_bboxes



def adjust_bbox_for_full_image(cropped_bbox, person_bbox):
    """Convert PPE bounding box from cropped image coordinates to full image
coordinates."""
```

```
    x_min_p, y_min_p, _, _ = person_bbox
    x_min, y_min, x_max, y_max = cropped_bbox
    adjusted_bbox = [x_min + x_min_p, y_min + y_min_p, x_max + x_min_p, y_max +
y_min_p]
    return adjusted_bbox

def process_inference(input_dir, output_dir, person_det_model,
ppe_detection_model):
    # Load models using the Ultralytics YOLO interface
    person_model = YOLO(person_det_model)  # Load the person detection model
    ppe_model = YOLO(ppe_detection_model)  # Load the PPE detection model

    for image_name in os.listdir(input_dir):
        image_path = os.path.join(input_dir, image_name)
        image = cv2.imread(image_path)
        if image is None:
            print(f"Error loading image: {image_path}")
            continue

        # Step 1: Detect persons in the full image
        person_bboxes = run_person_detection(person_model, image)

        for i, person_bbox in enumerate(person_bboxes):
            # Crop the person from the image
            x_min, y_min, x_max, y_max = person_bbox
            cropped_image = image[y_min:y_max, x_min:x_max]

            # Step 2: Detect PPE on the cropped person image
            ppe_bboxes = run_ppe_detection(ppe_model, cropped_image)

            # Draw bounding boxes on the full image with adjusted coordinates
            for ppe_label, ppe_bbox in ppe_bboxes:
                adjusted_bbox = adjust_bbox_for_full_image(ppe_bbox, person_bbox)
                draw_bounding_box(image, adjusted_bbox, f"PPE {ppe_label}")

        # Save the annotated image
        output_image_path = os.path.join(output_dir, image_name)
        cv2.imwrite(output_image_path, image)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Inference script for person and
PPE detection models.")
```

```
    parser.add_argument('--input_dir', type=str, required=True, help="Directory
containing input images")
    parser.add_argument('--output_dir', type=str, required=True, help="Directory
to save output images")
    parser.add_argument('--person_det_model', type=str, required=True, help="Path
to the person detection model")
    parser.add_argument('--ppe_detection_model', type=str, required=True,
help="Path to the PPE detection model")

    args = parser.parse_args()

    os.makedirs(args.output_dir, exist_ok=True)

    process_inference(args.input_dir, args.output_dir, args.person_det_model,
args.ppe_detection_model)

    print("Inference completed.")
```

Here, four inputs are present which are image input directory, output image directory, person detection model and ppe detection model. This code is being executed at command line as it uses argparse. This code infers both person detection and ppe detection where full images are used to draw bounding box for ppe detection instead of cropped images. This is first achieved by converting cropped image annotation to full image annotation. The bounding boxes are displayed on full image using cv2.rectangle() and cv2.puttext()