Syook assignment

1. Introduction

This project involves creating an inference pipeline to detect persons and their associated personal protective equipment (PPE) in images using two separate YOLOv8 models. The workflow involves detecting persons in full images, cropping these images based on detected person bounding boxes, and then running a PPE detection model on these cropped images. Finally, the results from both models are combined and visualized on the original images.

---

2. Approach

Step 1: Person Detection

- **Model**: The person detection is handled by a YOLOv8 model pre-trained or fine-tuned specifically for detecting persons.
- **Process**:
  - The model takes a full image as input and performs inference.
  - Bounding boxes for detected persons are extracted, including their coordinates and confidence scores.

Step 2: Image Cropping

- **Purpose**: The bounding boxes from the person detection model are used to crop the image around each detected person.
- **Process**:
  - For each detected person, a region of interest (ROI) is created by cropping the original image according to the bounding box coordinates.

Step 3: PPE Detection

- **Model**: The cropped images are passed to a second YOLOv8 model trained specifically for PPE detection.
- **Process**:

- The model performs inference on the cropped images.
- Bounding boxes for detected PPE items are extracted.

Step 4: Mapping Cropped Detections Back to Full Image

- **Logic**:
  - The PPE detections (from cropped images) are mapped back to their corresponding locations in the original full image.
  - This is done by adjusting the coordinates of the PPE bounding boxes relative to the original bounding box of the person.

Step 5: Visualization

- **Process**:
  - Bounding boxes and confidence scores from both models are drawn on the original images.
  - OpenCV's `cv2.rectangle()` and `cv2.putText()` functions are used for this purpose.

Step 6: Output

- **Process**:
  - The images with drawn bounding boxes and labels are saved in the specified output directory.

---

3. Learning

Model Selection:

- **YOLOv8** was selected due to its state-of-the-art performance in object detection tasks, offering a good balance between accuracy and speed.

Bounding Box Mapping:

- Mapping the cropped image's detections back to the full image required careful handling of coordinates, ensuring that the relative positions were accurately maintained.

Optimization:

- Running the inference on full images first and then on cropped images helped in reducing the computational load, as the PPE model only needed to process relevant portions of the image.

---

4. Evaluation Metrics

Accuracy:

- Evaluated based on the correct detection of persons and correct classification of PPE items.

Precision and Recall:

- Precision and recall metrics were used to evaluate the performance of both the person detection and PPE detection models.

Inference Time:

- The time taken to run inference on a batch of images was measured to ensure the pipeline was efficient.

Code Logic Explanation

---

1. Inference Pipeline

```
import os
```

```
import cv2
```

```
import argparse
```

```
from ultralytics import YOLO
```

- These imports bring in necessary libraries. `os` and `cv2` are used for file operations and image processing, `argparse` for command-line arguments, and `YOLO` for the model.

```
def run_inference(model, image):
```

- **Purpose**: This function performs inference using the provided YOLO model.
- **Details**:
  - `model(image)`: Runs inference on the input image.

- `return results`: Returns the results from the inference.

```
def draw_boxes(image, results, label_map):
```

- **Purpose**: This function draws bounding boxes and labels on the image.
- **Details**:
  - Iterates over detected results and boxes.
  - Extracts coordinates, confidence scores, and class labels.
  - Draws rectangles and text on the image using OpenCV functions `cv2.rectangle()` and `cv2.putText()`.

```
def main(input_dir, output_dir, person_model_path, ppe_model_path):
```

- **Purpose**: This function manages the overall inference and processing workflow.
- **Details**:
  - Loads YOLO models for person and PPE detection.
  - Ensures the output directory exists.
  - Defines a label map to translate class IDs to human-readable labels.
  - Processes each image in the input directory:
    - Runs person detection on the full image.
    - Crops the image based on detected person bounding boxes.
    - Runs PPE detection on the cropped images.
    - Maps PPE detections back to the full image and draws bounding boxes.
    - Saves the annotated image to the output directory.

```
if __name__ == "__main__":
```

- **Purpose**: This block runs the `main()` function with command-line arguments.
- **Details**:
  - Uses `argparse` to parse input arguments.
  - Calls `main()` with the parsed arguments.

---

2. Pascal VOC to YOLO Conversion

`import os`

`import xml.etree.ElementTree as ET`

`import argparse`

- Imports required for file operations, XML parsing, and argument handling.

`def convert_pascal_voc_to_yolo(label_folder, output_folder):`

- **Purpose**: Converts Pascal VOC annotations to YOLO format.
- **Details**:
  - Reads XML files from the label folder.
  - Parses each XML file to extract bounding box coordinates.
  - Converts coordinates to YOLO format: class ID, center coordinates, width, and height.
  - Writes the YOLO formatted annotations to text files in the output folder.

`if __name__ == '__main__':`

- **Purpose**: Executes the conversion script with command-line arguments.
- **Details**:
  - Parses arguments using `argparse`.
  - Calls `convert_pascal_voc_to_yolo()` with the parsed arguments.

---

3. PPE Dataset Generation

`import os`

`import cv2`

`import xml.etree.ElementTree as ET`

`import argparse`

`from ultralytics import YOLO`

- Imports necessary libraries for file handling, image processing, XML parsing, and YOLO model usage.

`def parse_xml_annotation(xml_file):`

- **Purpose**: Parses XML files to extract object annotations.

- **Details**:
  - Reads and parses XML files.
  - Extracts bounding box coordinates and class names.
  - Returns annotations in a list.

```
def save_yolo_annotation(filename, annotations, img_width, img_height):
```

- **Purpose**: Saves annotations in YOLO format.
- **Details**:
  - Converts bounding box coordinates to YOLO format.
  - Writes annotations to a text file.

```
def process_images(image_dir, annotation_dir, model, output_image_dir, output_annotation_dir):
```

- **Purpose**: Processes images and annotations using the YOLO model.
- **Details**:
  - Loads images and their corresponding XML annotations.
  - Runs YOLO inference to detect persons.
  - Crops images around detected persons and performs PPE detection on the cropped images.
  - Adjusts and saves the PPE annotations in YOLO format.

```
if __name__ == "__main__":
```

- **Purpose**: Runs the dataset generation script with command-line arguments.
- **Details**:
  - Uses `argparse` to parse input arguments.
  - Calls `process_images()` with parsed arguments and required directories.

---

6. Conclusion

This project successfully implements an efficient and accurate pipeline for detecting persons and their PPE in images. By leveraging the strengths of YOLOv8 for both person and PPE detection and optimizing the processing steps, the

pipeline achieves reliable performance in real-world scenarios where PPE compliance is critical.