Documentation for YOLO Image Processing Application

Overview

This application is designed to process images in real-time using the YOLO (You Only Look Once) object detection model. It detects specific object types in images, annotates them, and organizes the images into appropriate directories based on detected object classes. This is particularly useful for scenarios requiring automated image categorization and annotation, such as quality control, research, or surveillance tasks.

Libraries Used

1. **os**: Handles file and directory operations such as path manipulations and folder creation.
2. **shutil**: Moves files between directories.
3. **cv2 (OpenCV)**: Reads and processes image files.
4. **ultralytics (YOLO)**: Runs object detection on images using a pre-trained YOLO model.
5. **watchdog**: Monitors directories for changes, enabling real-time processing.
6. **time**: Introduces delays in program execution and supports continuous monitoring.

Code Breakdown

1. Importing Required Libraries

```
import os
import shutil
import cv2
from ultralytics import YOLO
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import time
```

These libraries provide the functionality needed for file handling, image processing, object detection, and real-time directory monitoring.

2. Loading the YOLO Model

```
model = YOLO(r'D:\My Laptop\Me\Programming\Machine Learning\Internships\Egyptian Space
Agency\YOLO\yolov11\small_model2\weights\best.pt')
```

This line loads a pre-trained YOLO model, allowing the application to detect objects in the images. Replace the path with the actual location of the model weights file on your system.

3. Defining Directory Paths

```
both_dir = 'both'
type2_dir = 'Type_2'
type3_dir = 'Type_3'
annotated_dir = 'annotated'
input_dir = 'data'
no_bursts = 'no_bursts'
```

Specifies the directories where categorized images and annotations will be saved.

4. Creating Output Directories

```
for folder in [both_dir, type2_dir, type3_dir, annotated_dir, no_bursts]:
    if not os.path.exists(folder):
        os.makedirs(folder)
```

Ensures the directories for output images and annotations exist, creating them if necessary.

5. Image Processing Function

```
def process_image(image_path):
    image_name = os.path.basename(image_path)
    img = cv2.imread(image_path)
    if img is None:
        print(f"Could not read image: {image_path}")
        return
```

Reads an image from the specified path. If the image cannot be read, it skips further processing.

```
result = model(img)
detected_classes = [model.names[int(cls)] for cls in result[0].boxes.cls]
bboxes = result[0].boxes.xywhn
class_ids = result[0].boxes.cls
```

Runs the YOLO model on the image and extracts detected object classes, bounding boxes, and class indices.

```
annotation_path = os.path.join(annotated_dir, f"{os.path.splitext(image_name)[0]}.txt")
with open(annotation_path, 'w') as f:
    for bbox, cls_id in zip(bboxes, class_ids):
        cls_id = int(cls_id)
        bbox_str = ' '.join(map(str, bbox.tolist()))
        f.write(f"{cls_id} {bbox_str}\n")
```

Generates YOLO-style annotations for each detected object and saves them in the `annotated` directory.

```
if 'TYPE 2' in detected_classes and 'TYPE 3' in detected_classes:
    destination_path = os.path.join(both_dir, image_name)
```

```
elif 'TYPE 2' in detected_classes:
    destination_path = os.path.join(type2_dir, image_name)
elif 'TYPE 3' in detected_classes:
    destination_path = os.path.join(type3_dir, image_name)
else:
    destination_path = os.path.join(no_bursts, image_name)
shutil.move(image_path, destination_path)
```

Categorizes the image based on detected object classes and moves it to the corresponding directory.

6. Processing Existing Images

```
for image_name in os.listdir(input_dir):
    image_path = os.path.join(input_dir, image_name)
    process_image(image_path)
```

Processes all images already present in the input directory when the application starts.

7. Real-Time Monitoring

```
class NewImageHandler(FileSystemEventHandler):
    def on_created(self, event):
        if not event.is_directory:
            process_image(event.src_path)
```

Defines a custom handler to process new images added to the input directory.

```
observer = Observer()
event_handler = NewImageHandler()
observer.schedule(event_handler, input_dir, recursive=False)
```

Sets up an observer to monitor the input directory for new files.

```
try:
    observer.start()
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    observer.stop()
observer.join()
```

Starts the observer for real-time monitoring and gracefully shuts it down upon user interruption.

Key Functionalities

1. **Batch Processing**: Processes all existing images in the input directory at startup.
2. **Real-Time Monitoring**: Automatically processes new images added to the input directory.

3. **Annotation Generation**: Creates YOLO-style annotations for each processed image.
4. **Image Categorization**: Moves images to predefined directories based on detected object classes.

Usage Instructions

1. Ensure that the required libraries are installed: pip install opencv-python ultralytics watchdog.
2. Place the pre-trained YOLO weights file at the specified path.
3. Add input images to the data directory.
4. Run the script. Processed images and annotations will be saved in the respective directories.
5. To stop the application, use Ctrl+C in the terminal.
