Documentation: YOLO Annotation and Dataset Preparation Script

Overview

This script processes image annotations and organizes them into a format suitable for training YOLO object detection models. It performs the following key tasks:

1. **Converts annotations** from a polygon-based format to YOLO format.
2. **Organizes datasets** into training and validation sets.
3. **Generates configuration files** (e.g., data.yaml) required for YOLO training.

Code Breakdown

1. Directory and Path Setup

The script defines paths to organize the data. These include directories for annotations, images, and YOLO-specific outputs. It also creates necessary directories if they don't already exist.

```
os.makedirs(output_yolo_dir, exist_ok=True)
os.makedirs(train_images_dir, exist_ok=True)
os.makedirs(val_images_dir, exist_ok=True)
os.makedirs(train_labels_dir, exist_ok=True)
os.makedirs(val_labels_dir, exist_ok=True)
```

Defined Paths:

- output_yolo_dir: Directory for storing YOLO-formatted annotations.
- train_images_dir and val_images_dir: Directories for training and validation images.
- train_labels_dir and val_labels_dir: Directories for corresponding annotation files.

---

2. Class Label Mapping

The label_to_id dictionary maps human-readable class names to numeric IDs required by YOLO.

```
label_to_id = {
   "TYPE 2": 0,
   "TYPE 3": 1
```

```
}
```

This ensures consistency across annotation files.

---

3. Annotation Conversion

a. Polygon to Bounding Box Conversion

The `polygon_to_bbox` function calculates bounding box coordinates from a set of points representing a polygon. The bounding box is defined by its top-left corner and its width and height.

```
def polygon_to_bbox(points):
    x_coords = [p[0] for p in points]
    y_coords = [p[1] for p in points]
    min_x, min_y = min(x_coords), min(y_coords)
    max_x, max_y = max(x_coords), max(y_coords)
    return min_x, min_y, max_x, max_y
```

b. Creating YOLO Annotations

The `create_yolo_annotations` function converts polygon-based annotations to YOLO format. The YOLO format represents each object as:

- Class ID.
- Normalized center coordinates (x, y) of the bounding box.
- Normalized width and height of the bounding box.

```
center_x = (min_x + max_x) / 2 / width
center_y = (min_y + max_y) / 2 / height
box_width = (max_x - min_x) / width
box_height = (max_y - min_y) / height
```

These annotations are saved as `.txt` files in the `output_yolo_dir`.

---

4. Processing Annotations

The script iterates through JSON annotation files, processes them, and saves YOLO-formatted annotations. Missing or unreadable files are logged in an error file (`error_files.txt`).

```
for cls in classes:
    class_dir = os.path.join(base_dir, cls, cls)
    for file in os.listdir(class_dir):
        if file.endswith(".json"):
            json_path = os.path.join(class_dir, file)
            success = create_yolo_annotations(json_path, output_yolo_dir, label_to_id)
```

```
        if not success:
            error_files.append(json_path)
```

---

5. Splitting Data

The script splits the dataset into training and validation sets based on a specified ratio (`train_ratio = 0.8`). The images and their corresponding annotations are copied to the respective directories.

```
split_idx = int(train_ratio * len(all_images))

train_images = all_images[:split_idx]
val_images = all_images[split_idx:]
```

It ensures that both the images and their annotations are moved together to maintain consistency.

---

6. Generating Configuration File

The `data.yaml` file is created to store metadata about the dataset, including:

- Class names.
- Number of classes.
- Paths to training and validation datasets.

```
data_yaml = {
    'names': classes,
    'nc': len(classes),
    'train': train_images_dir,
    'train_labels': train_labels_dir,
    'val': val_images_dir,
    'val_labels': val_labels_dir
}
with open(yaml_path, 'w') as file:
    yaml.dump(data_yaml, file, default_flow_style=False)
```

The `data.yaml` file is essential for configuring YOLO training pipelines.

Key Concepts

1. YOLO Annotation Format

YOLO annotations use a simple text format where each line corresponds to an object in the image:

```
<class_id> <center_x> <center_y> <width> <height>
```

All values (except class_id) are normalized to the range [0, 1] with respect to the image dimensions.

2. Data Preprocessing

The script ensures that the data is properly formatted and split into training and validation sets, crucial steps for model training.

3. Error Handling

The script logs errors (e.g., missing images or annotations) to facilitate debugging.

Usage Instructions

1. Place the annotation JSON files in their respective directories under the base directory.
2. Update the base_dir and classes variables as needed.
3. Run the script.
4. Use the generated data.yaml and organized datasets for YOLO model training.

Conclusion

This script streamlines the preprocessing pipeline for YOLO training by automating annotation conversion, data organization, and configuration file generation. It simplifies the preparation of large datasets for object detection tasks.