Annotation Process and Challenges

This report outlines the process of annotating images for object detection, the challenges encountered, and the methods used to address them. The dataset was created from images collected from Kaggle and Google Photos, followed by annotation, conversion to different formats, and augmentation to enhance training performance.

Annotation Process

1. **Image Collection**
    - A diverse set of images was gathered from Kaggle and Google Photos to ensure a varied dataset.
2. **Annotation with LabelMe**
    - The LabelMe tool was used for annotation, where bounding boxes were manually drawn around objects of interest.
    - LabelMe stores annotations in JSON format, which provides xmin, ymin, width, and height values for each bounding box.
3. **Conversion to COCO Format**
    - Since COCO is a widely accepted annotation format, LabelMe JSON annotations were converted to COCO format.
    - This standardization ensured compatibility with various deep-learning models and frameworks.
4. **Conversion to YOLO Format**
    - YOLO requires annotations in a different format where bounding boxes are represented as:
        - `(x_center, y_center, width, height)`, normalized between 0 and 1.
    - Conversion was done using the formula:
        - `x_center = (xmin + width / 2) / image_width`
        - `y_center = (ymin + height / 2) / image_height`
        - `width = width / image_width`
        - `height = height / image_height`
    - Handling this transformation correctly was a key challenge.

Data Augmentation

1. **Using Albumentations**
    - To increase the dataset size and improve model generalization, Albumentations was used for data augmentation.
    - Various transformations such as rotation, flipping, brightness adjustments, and scaling were applied.
    - The original 150 images were expanded into a dataset of 600 images, maintaining the COCO annotation format.
2. **Challenges in Augmentation**
    - Ensuring bounding box consistency after transformations.
    - Validating that augmented images retained correct annotations in COCO format.

Challenges Faced and Solutions

1. **Bounding Box Conversion Issues**
    - Ensuring correct calculations when transforming LabelMe JSON to YOLO format.
    - Solution: Used systematic calculations to convert bounding box coordinates properly.
2. **Handling Augmented Annotations**
    - Bounding boxes needed to be recalculated after augmentation to maintain accuracy.
    - Solution: Used Albumentations' built-in functions to transform both images and annotations together.

The annotation process was systematically executed, ensuring proper format conversions and augmentation. By overcoming challenges related to bounding box transformations and data expansion, a robust dataset was created, suitable for training the YOLOv8 model effectively.

Object Detection Model Training

This report summarizes the process of training an object detection model using the YOLOv8s architecture. The dataset was annotated in the YOLO format, and the model was fine-tuned on a custom dataset of labeled images.

Model Architecture: YOLOv8s

YOLOv8s (You Only Look Once, version 8 small) is a real-time object detection model known for its efficiency and accuracy. It consists of:

- **Backbone:** A CSPDarkNet-based feature extractor.
- **Neck:** PANet structure to enhance feature fusion.
- **Head:** YOLO detection head with anchor-free regression and classification.
- **Loss Function:** Combination of box loss, class loss, and DFL (Distribution Focal Loss) to improve localization accuracy.

Training Methodology

Dataset Preparation

- The dataset was collected from Kaggle and Google Photos.
- It was annotated using LabelMe, producing JSON annotations.
- JSON files were converted to COCO format and then to YOLO format for compatibility with YOLOv8.
- Data augmentation was applied using Albumentations, increasing the dataset from 150 to 600 images in COCO format.

Data Splitting

- The dataset was divided into **85% training** and **15% validation**.

Model Training

- A pre-trained YOLOv8s model was fine-tuned on the dataset.
- Training was conducted for **80 epochs** with optimized learning rates.
- Metrics such as precision, recall, and mean Average Precision (mAP) were tracked.

Model Performance & Evaluation

Training performance metrics include:

- **Precision (mAP@50):** Increased from **19.7% to 34.9%** over 4 epochs.
- **Recall:** Improved from **21.1% to 35.9%**.
- **mAP@50-95:** Reached **15.6%** at peak performance.
- **Validation loss:** Gradually decreased, indicating proper learning.

The YOLOv8s model demonstrated effective learning with increasing precision and recall. Further improvements can be achieved by:

- Increasing dataset size.
- Experimenting with learning rates and augmentations.
- Fine-tuning hyperparameters for better localization and classification accuracy.

This report provides an overview of the object detection model's training, performance, and key learnings from the process.

Accuracy: The overall accuracy of the model is approximately **85.6%**, calculated as the ratio of correctly classified instances to the total instances.

Precision: Precision for each class indicates the correctness of positive predictions:

- Bike: **80.0%**
- Car: **58.6%**
- Rickshaw: **85.5%**
- Background: **50.0%**

Recall: Recall measures the proportion of actual positives correctly identified:

- Bike: **66.7%**
- Car: **77.3%**
- Rickshaw: **89.0%**
- Background: **61.5%**

User Guide for Real-Time Object Detection Application

The **Real-Time Object Detection Application** is a desktop application built using PyQt6 and the YOLO object detection framework. It allows users to process videos and detect objects in real-time, displaying bounding boxes and object labels for detected objects.

Requirements

To use this application, ensure you have the following prerequisites installed on your system:

1. **Python 3.8 or higher**
2. **Required Python Libraries:**
    - PyQt6
    - OpenCV (`cv2`)
    - PyTorch
    - Ultralytics YOLO
3. **Pre-trained YOLO Model File:**
   The application requires a trained YOLO model file (`model.pt`) in the same directory as the script.

How to Use the Application

Launch the Application
Run the script by executing the following command in your terminal:

```
python object_detection_app.py
```

User Interface
The application will display a simple graphical user interface (GUI) with the following elements:

- **Select Video Button:** A button to choose the video file for processing.

Selecting a Video

- Click the **"Select Video"** button.
- A file dialog will open. Navigate to the folder containing your video file and select it. Supported video formats include `.mp4` and `.avi`.

Real-Time Object Detection

- Once a video is selected, the application will start processing the video in real-time.
- The detection results will be displayed in a new window, showing bounding boxes and labels for detected objects.

Stopping the Detection Process

- To stop the detection process, press the **'q'** key while the detection window is open.

Exiting the Application

- Close the application window to exit.

3. Research and Literature Review:

SAM 2: Segment Anything in Images and Videos

The proposed method in SAM 2 is **Promptable Visual Segmentation (PVS)**, which segments objects in both images and videos based on user prompts (points, boxes, or masks). It includes a **streaming memory module** to store object information from previous frames, enabling accurate segmentation across video sequences. This approach is supported by a new **data engine** that generates high-quality training data, resulting in the large SA-V dataset for training and evaluation.

SAM 2 is designed to handle both static image segmentation and dynamic video segmentation, **treating an image as a single-frame video**. It includes a memory module to store information about objects from earlier frames, enabling accurate real-time segmentation in videos.

The Segment Anything Model 2 (SAM 2) represents a significant advancement in the field of visual segmentation by providing a unified framework for both image and video segmentation. Unlike its predecessor, which focused solely on images, SAM 2 adapts to the complexities of video data, where objects can undergo changes in appearance due to motion, lighting, and occlusion.

A key feature of SAM 2 is its **Promptable Visual Segmentation (PVS)**, which uses user-defined prompts (e.g., clicks, boxes, or masks) to segment objects across frames. By incorporating a **streaming memory module**, the model efficiently processes video frames in real-time, utilizing previous interactions for better accuracy.

To support SAM 2, the team developed a **data engine** for annotating challenging video data, creating the **SA-V dataset**, the largest of its kind. This dataset is highly diverse, with 53× more

masks than any existing video segmentation dataset, making it suitable for segmenting any object, including occluded and small parts.

Experimental results demonstrate that SAM 2 surpasses prior methods in established benchmarks, offering better segmentation accuracy with fewer user interactions. It also performs well across zero-shot benchmarks for both images and videos, establishing itself as a versatile tool for applications like AR/VR, robotics, and video editing.

Overall, SAM 2's innovative architecture and extensive dataset mark a significant milestone in video segmentation, promising broad applications in various domains.