Executive Summary:

This project implements a multi-stage object detection system using YOLOv8 for person detection and Personal Protective Equipment (PPE) identification. The system processes images through two separate models: one for detecting persons in full images, and another for identifying PPE items on cropped person images.

Key achievements include:

1. Development of a Python script to convert PascalVOC annotations to YOLOv8 format.
2. Training of two YOLOv8 models: one for person detection and another for PPE detection.
3. Implementation of a custom logic to handle annotations for multiple persons and their corresponding PPE.
4. Creation of an inference pipeline that processes images through both models and reconstructs final predictions.
5. Utilization of OpenCV for drawing prediction results.

The system demonstrates robust performance in identifying persons and various PPE items, including hard-hats, gloves, boots, vests and PPE-suits This solution has potential applications in workplace safety monitoring and compliance checking.

Project Objectives:

1. Develop a data preprocessing pipeline to convert existing annotations from PascalVOC format to YOLOv8 format, ensuring compatibility with the chosen model architecture.
2. Train a YOLOv8 object detection model specifically for person detection in full images, laying the foundation for subsequent PPE analysis.
3. Implement a custom annotation conversion logic to transform full-image PPE annotations into annotations for cropped person images.
4. Train a second YOLOv8 object detection model for PPE detection, focusing on identifying nine specific items: hard-hats, gloves, masks, glasses, boots, vests, PPE-suits, ear-protectors, and safety-harnesses.
5. Create an end-to-end inference pipeline that processes input images through both models sequentially, handling the necessary coordinate transformations between full and cropped images.
6. Implement custom visualization of detection results using OpenCV, ensuring fine-grained control over the output presentation.
7. Evaluate the performance of both models using appropriate metrics and analyze the system's effectiveness in real-world scenarios.
8. Document the entire process, including methodologies, challenges faced, solutions implemented, and potential areas for future improvement.

Methodologies

1. PascalVOC to YOLO Annotation Conversion:

Introduction:

The Python script is designed to convert annotations in the PascalVOC format to the YOLO Format. This conversion is essential for training object detection models using the YOLO framework.

The script leverages the **ARGPARSE** module for command-line argument parsing, **XMl.etree** for **XML** parsing, and **OS** and **GLOB** for file and directory operations.

Approach:

1. **Argument Parsing:**
    a. Use ARGPARSE to define command-line arguments for input and output directories.
    b. Parse the command-line arguments to obtain user-specified paths.
2. **Class Definition:**
    a. Read the classes.txt file from the input directory to obtain a list of class names.
3. **Annotation Conversion:**
    a. For each XML file in the annotations subdirectory:
        i. Parse the XML file using xml.etree.ElementTree.
        ii. Extract the image size and bounding box information.
        iii. Convert the bounding box coordinates to YOLO format using the convert_bbox function.
        iv. Write the class label and normalized bounding box coordinates to a text file in the output directory.
4. **Error Handling:**
    a. Check for the existence of input and output directories, and handle errors gracefully.
    b. Provide informative messages for missing files or invalid annotations.

2. Yolo v8 Person Detection Model:

Approach:

1. Dataset Splitting (Function: split_dataset_multiclass):
    a. Creates directories for training and validation sets of two object classes: "person" and "PPE".
    b. Splits the original image dataset into training and validation sets based on a specified ratio (default: 80/20).
    c. Processes the corresponding label files for each class:
        i. Separates labels for "person" and "PPE" objects.
        ii. Adjusts class IDs for "PPE" by subtracting 1 to maintain unique class labels.
    d. Copies images and filtered label files to their respective training and validation directories.

2. Model Training (YOLO Model with ultralytics):
    a. Loads a pre-trained YOLOv8 model (e.g., "yolov8n.pt").
    b. Defines the dataset configuration file (.yaml) containing training parameters.
    c. Trains the model for a specified number of epochs.
    d. Applies data augmentation techniques like image rotation, translation, and scaling during training.
    e. Monitors training progress and saves the final model.

3. Model Validation:
    a. Evaluates the trained model's performance on the validation set using metrics like mean Average Precision (mAP).
    b. Prints the mAP50-95, mAP50, and mAP75 values to assess object detection accuracy.

4. Inference and Visualization:
    a. Loads a test image for object detection.
    b. Runs inference on the test image using the trained model.
    c. Visualizes the detected objects and bounding boxes directly on the image.
    d. Saves the visualized image with detections.

Detailed Logic:

1. Dataset Splitting (Function: split_dataset_multiclass):

This function splits a source dataset containing images and their corresponding labels into separate training and validation sets. Here's the logic breakdown:

- **Directory Creation:** Creates necessary directories for training and validation images and labels for both "person" and "PPE" categories.

- **Image Selection:**
    - Identifies all image files within the source directory.
    - Shuffles the list of images for randomization.
    - Splits the list based on a specified ratio (default 80%) to create training and validation sets.

- **Label Processing:**
    - Defines a function process_label_file to handle label files for each image.
    - Opens the label file and reads its content line by line.
    - Filters labels based on class ID
        - Class ID 0 (assumed to be "person") is copied directly to the person training and validation directories.

        - Other class IDs are considered PPE. Their class IDs are decremented by 1 to account for the separate "person" class. These lines are then copied to the respective PPE training and validation directories.

- **File Copying:**
    - Iterates through the training and validation image lists.
    - Copies each image file to the corresponding training or validation directory (both "person" and "PPE").
    - Copies the corresponding label file after processing it with process_label_file.

2. Object Detection Training and Evaluation:

This section demonstrates training a YOLOv8 model and evaluating its performance.

- **Model Loading:** Loads the pre-trained YOLOv8 model "yolov8n.pt" using the ultralytics library.
- **Data Configuration:** Defines the path to a YAML file (data_yaml) containing information about the training data like image paths and class labels.
- **Model Training:**
    - Trains the model using the train method of the YOLO object. The provided code snippet shows various hyperparameters used for training, which can be adjusted based on your dataset size and desired performance.
    - You can experiment with parameters like epochs, batch size, learning rate, and data augmentation settings.
- **Model Validation:**
    - After training, the val method is used to evaluate the model's performance on the validation set.
    - It calculates and prints metrics like mAP50-95 (mean Average Precision at different Intersections over Union (IoU) thresholds).

- **Model Saving:**
    - The trained model with the best performance is saved as "final_model.pt".
- **Inference on Test Image:**
    - Loads a test image (test_image) using OpenCV.
    - Performs object detection using the trained model (model(test_image)).
- **Visualization:**
    - Visualizes the detected objects and bounding boxes directly on the image using the plot method of the detection result.
    - Converts the image from BGR (OpenCV format) to RGB for proper display in Colab.
    - Displays the visualized image using cv2_imshow.
    - Saves the visualized image as "result.jpg".

Overall, this code demonstrates a practical workflow for creating a custom object detection model with YOLOv8. By splitting your dataset and fine-tuning hyperparameters, you can train a model tailored to your specific object detection task (e.g., identifying people and PPE in images).

3. Image Cropping and Annotation Adjustment for PPE detection model

Introduction

This Python script is designed to crop images containing persons and adjust their corresponding annotations to reflect the new image dimensions. The script leverages OpenCV for image processing, os for file operations, and glob for file path matching.

Approach :

1. Image and Annotation Loading:
    a. Use glob to find all image files in the input directory.
    b. For each image, locate the corresponding annotation file based on the image filename.
2. Image Cropping:
    a. Read the image using Open CV.
    b. Extract the bounding box coordinates of persons from the annotation file.
    c. Crop the image based on the person bounding box coordinates.
3. Annotation Adjustment:
    a. Calculate the new bounding box coordinates relative to the cropped image.
    b. Adjust the annotation file to reflect the new coordinates.
4. Saving Results:
    a. Save the cropped images and adjusted annotation files to the output directory.

4. Yolo v8 PPE detection model on cropped person images.

Introduction:

This Colab notebook demonstrates a workflow for training a YOLOv8 model to detect Personal Protective Equipment (PPE) in images. It leverages the ultralytics library for streamlined model training and inference.

Approach :

1. Dataset Splitting:
    a. Splits the original "CROPPED" dataset directory into training and validation sets based on an 80/20 ratio.
    b. Filters label files to keep only the desired classes representing different types of PPE (hard-hat, glasses, vest, etc.).
    c. Copies filtered images and labels to their respective training and validation directories.

2. Label Counting (Function: count_labels)

    a. Counts the occurrences of each class label in the training and validation sets.
    b. Helps verify the distribution of classes and identify potential imbalances.

3. Dataset Configuration (YAML File):

    a. Defines the training data paths (train and validation sets).
    b. Assigns human-readable names to class IDs for better interpretation.

4. Model Training:

    a. Loads the pre-trained YOLOv8 "yolov8n.pt" model for efficient training.

b. Defines hyperparameters for training: epochs, batch size, optimizer, learning rate, and data augmentation settings.
c. Uses a smaller image size (416) and reduced mosaic augmentation since the input images are cropped.
d. Trains the model for 50 epochs and saves the best performing model as "best.pt".

5. Model Evaluation (Implicit):

a. While not explicitly shown, the training process likely involves validation during each epoch to evaluate the model's performance on unseen data.

6. Inference and Visualization:

a. Loads a test image from the "cropped_1.jpg" path.
b. Runs model inference to detect PPE objects in the image.
c. Visualizes the detected objects and bounding boxes directly on the image.
d. Converts the image from BGR (OpenCV format) to RGB for proper display in Colab.
e. Saves the visualized image as "result.jpg".

5. Inferencing Through The Models.

Introduction

This Python script demonstrates a workflow for detecting persons and Personal Protective Equipment (PPE) within images. It takes image directory, person detection model and PPE Detection model as an input performs the inferencing and saves them in another directory.

Approach

1) Model Loading:
    a) Load the person detection model and the PPE detection model using `YOLO`.
2) Image Processing:
    a) For each image in the input directory:
        i) Read the image using OpenCV.
        ii) Perform person detection using the person detection model.
        iii) For each detected person:
            (1) Crop the region of interest containing the person.
            (2) Perform PPE detection on the cropped region using the PPE detection model.
            (3) Adjust the PPE bounding box coordinates to match the original image.
3) Visualization:
    a) Draw bounding boxes and labels for detected persons and PPE items on the original image.
4) Output Saving:
    a) Save the processed image with the visualizations to the output directory.

Conclusion

This script provides a robust solution for detecting persons and PPE within images. It can be adapted to various use cases where identifying and localizing these objects is essential. The script's modular design allows for easy integration into larger applications.

Key Learnings and Findings

1. YOLOv8 for Object Detection:

   A. YOLOv8 is a powerful and efficient object detection model.
   B. It can be used effectively for tasks like person and PPE detection.
   C. The ultralytics library provides a user-friendly interface for training and using YOLOv8 models.

2. Multi-Stage Detection:

   A. The approach of first detecting persons and then performing PPE detection on cropped regions is a common strategy for improving accuracy.
   B. This can be especially useful when the objects of interest (PPE) are relatively small and might be missed by a single-stage detector.

3. Data Preprocessing:

   A. The script demonstrates the importance of preprocessing data for object detection tasks.
   B. This includes creating training and validation sets, filtering labels, and adjusting bounding box coordinates.

4. Model Selection:

   A. The choice of YOLOv8 model (e.g., "yolov8n.pt") depends on factors like the desired accuracy, speed, and computational resources available.
   B. Experimenting with different models can help find the best fit for a specific task.

5. Hyperparameter Tuning:

   A. The script's training parameters can be fine-tuned to improve model performance.
   B. This includes adjusting parameters like learning rate, batch size, and data augmentation settings.

6. Visualization:

A. Visualizing the detected objects and bounding boxes is essential for understanding the model's performance and identifying potential issues.
B. The script uses OpenCV to draw bounding boxes and labels on the images.

7. Efficiency:

A. The script demonstrates the efficiency of YOLOv8, which can perform real-time object detection on many images.
B. This makes it suitable for applications that require fast and accurate object detection.

Conclusion

This project has successfully developed a multi-stage object detection system for person and PPE detection using YOLOv8 models. We have achieved our primary objectives of creating a data preprocessing pipeline, training specialized models for person and PPE detection, and implementing an end-to-end inference system.

Key achievements include:

1. Successful conversion of PascalVOC annotations to YOLOv8 format.
2. Training of a YOLOv8 model for person detection.
3. Development of a custom annotation conversion logic for PPE items .
4. Training of a YOLOv8 model for PPE detection.
5. Implementation of an inference pipeline that effectively combines both models .
6. Custom visualization of results using OpenCV. The system demonstrates robust performance in identifying persons and various PPE items in diverse scenarios.