Project Report: GT-YOLOv8 Tree Detection

Executive Summary:

This report details the GT-YOLOv8 project, a tree detection system implemented using the YOLOv8 object detection framework. The project utilizes a custom dataset for training and validation, resulting in multiple trained models stored within the repository. The report covers the project's architecture, implementation, performance, and future development plans. The primary objective was to build and evaluate a robust tree detection model, showcasing different training runs and their respective performance metrics.

Project Overview:

The GT-YOLOv8 project aims to develop an accurate and efficient system for detecting trees in images. The project leverages the Ultralytics YOLOv8 library, a pre-trained model (`yolov8n.pt`), and a custom dataset located in the `datasets/trainingdata` directory. The training process generated multiple model versions, each representing different training runs or hyperparameter adjustments, stored in the `runs/detect` directory. The project demonstrates a complete workflow, from data preparation to model training and evaluation, producing various performance metrics and visualization outputs.

Technical Architecture:

The system follows a standard object detection pipeline. The architecture consists of three primary components:

1. **Data Management:** The custom dataset is structured with training and validation sets, each containing images and corresponding annotation files (.txt). The `data.yaml` file configures the dataset paths and class labels.

2. **Model Training:** The YOLOv8 model (`yolov8n.pt`) is fine-tuned using the provided dataset. The `main.py` script manages the training process, utilizing the `data.yaml` configuration. Multiple training runs are evident in the `runs/detect` directory, suggesting experimentation with different hyperparameters or data augmentation techniques.

3. **Inference and Evaluation:** The trained models generate detection results, visualized in output images and evaluated using metrics like precision, recall, F1-score, and confusion matrices, which are all stored in the respective `runs/detect` subdirectories.

Features and Functionality:

* **Tree Detection:** The core functionality is the accurate detection and localization of trees within images.
* **Multiple Training Runs:** The project showcases multiple training runs, allowing for comparison of different model configurations and performance.
* **Performance Evaluation:** Comprehensive performance metrics (Precision-Recall curves, F1-score, confusion matrices) are generated and saved for each run.
* **Model Persistence:** Trained models are saved (`best.pt` and `last.pt`) for each run, facilitating easy deployment and reuse.

Installation and Setup:

The project relies on Python and the Ultralytics YOLOv8 library. The `requirements.txt` file lists all necessary dependencies. Installation can be achieved by cloning the repository and executing:

```bash
pip install -r requirements.txt
```

Before running `main.py`, ensure that the paths within `data.yaml` accurately reflect the location of your dataset.

Technology Stack:

* **Programming Language:** Python
* **Object Detection Framework:** Ultralytics YOLOv8
* **Deep Learning Framework:** PyTorch

Performance and Scalability:

The performance of each training run is assessed through the provided metrics within the respective `runs/detect` subdirectories. The `results.csv` and associated visualizations (e.g., `results.png`, PR curves) provide a detailed performance overview for each run. A quantitative analysis of these results would be required for a complete performance evaluation and comparison between runs. Scalability would depend on the size of the dataset and computational resources available. The use

of `yolov8n.pt` suggests an emphasis on efficiency, especially considering the number of training runs performed.

Future Roadmap:

* **Hyperparameter Optimization:** Systematic hyperparameter tuning to optimize model performance.
* **Data Augmentation:** Explore advanced data augmentation techniques to improve robustness.
* **Model Compression:** Investigate model compression techniques to reduce model size and inference time.
* **Real-time Inference:** Optimize the model for real-time performance on embedded systems.
* **Deployment:** Deploy the best-performing model to a practical application (e.g., web application, mobile app).

Challenges and Solutions:

* **Dataset Size:** The dataset size might limit the model's generalization ability. A solution would be to expand the dataset with more diverse images and annotations.
* **Computational Resources:** Training deep learning models can be computationally expensive. Utilizing cloud computing resources (e.g., Google Colab, AWS) could mitigate this challenge.
* **Hyperparameter Tuning:** Finding optimal hyperparameters requires experimentation and potentially advanced techniques like Bayesian optimization.

This report provides a high-level overview of the GT-YOLOv8 project. A deeper dive into specific training runs and performance analysis would require a more detailed examination of the results within the `runs/detect` directory. The numerous `tree_detectionX` runs suggest an iterative approach to model development, highlighting the practical challenges and solutions involved in building a robust object detection system.