Mid-term Project: Implementing Object Detection on a Dataset

Overview: Traffic Sign Using YOLOv8

This project aims to develop a traffic sign and signal detection model using the YOLOv8 object detection algorithm. By leveraging a dataset containing various traffic signs and signals, such as speed limits, stop signs, and traffic lights, this system provides a foundation for autonomous driving applications, aiding in real-time recognition of road indicators for safer navigation.

Dataset: Traffic Sign Detection

The dataset used in this project is the "Car Detection" dataset from Kaggle. This dataset was chosen because it contains annotated images of various traffic-related objects, such as traffic lights, stop signs, and speed limit signs. These objects are essential for training an object detection model in real-world autonomous driving scenarios. The dataset includes diverse images captured under different lighting conditions, perspectives, and distances, providing a robust foundation for model generalization.

Components of the Dataset:

Images

- The dataset contains a collection of images depicting various traffic signs, signals, and vehicle-related objects. Each image is formatted for compatibility with object detection tasks, capturing various angles, lighting conditions, and real-world scenarios to improve the model's generalizability.

Annotations (Bounding Boxes)

- Each image has corresponding annotation files that include labeled bounding boxes. These annotations provide the location and class of each object (e.g., traffic signs and signals) within the image. This structure allows the model to learn both the object's identity and its precise position.
- The annotations are in .txt format, where each line contains information such as the class label, x and y coordinates, width, and height of each bounding box in a normalized format.

Class Labels

- The dataset includes a set of 15 classes, such as "Green Light," "Red Light," "Speed Limit 20," "Stop," etc. These class labels correspond to various traffic signs and signals commonly encountered in real-world driving scenarios.

Data Split for Training, Validation, and Testing

- The dataset is structured into three subsets: train, val, and test, which are defined in

the data.yaml configuration file. This split allows for training the model on one portion, validating its performance on a separate portion, and finally testing on an unseen dataset.

Configuration File (data.yaml)

- The data.yaml file is used by YOLOv8 to organize and read the dataset correctly. This file specifies the paths for the train, val, and test image directories, the number of classes (nc: 15), and the class labels. The data.yaml file is integral for setting up the dataset correctly in YOLO and ensuring the model knows where to find each dataset component.

Algorithm: YOLOv8

YOLO (You Only Look Once) v8 is the latest iteration of the YOLO series, known for its real-time object detection capabilities. YOLOv8 combines high detection speed with impressive accuracy, making it a powerful choice for applications where quick and accurate recognition is essential, such as in autonomous driving and traffic sign detection.

Why YOLOv8 for Traffic Sign Detection?

- YOLOv8 is particularly suited for this project due to its high accuracy in identifying small, distinct objects (like traffic signs) at varying distances and under different lighting conditions. Its real-time processing capability aligns well with the needs of autonomous driving, where rapid and precise detection is crucial for safe navigation.

Dataset Preparation:

Image Resizing

- Each image in the dataset was resized to a target size of 416 x 416 pixels, which is a common input size for YOLO models. This resizing step standardizes the images, ensuring they fit the input requirements of the model and reducing computational load during training.

```
# Define directory and target image size
image_dir = '/content/drive/MyDrive/car/train/images'
target_size = (416, 416)  # Resize to 416x416
```

Pixel Normalization

- Each image's pixel values were normalized to a range of 0 to 1 by dividing by 255.0. Normalization helps stabilize training, making the model less sensitive to variations in lighting and contrast across images.

```
# Normalize pixel values (0 to 1)
image_normalized = image_resized / 255.0
```

Bounding Boxes and Labels

- The dataset includes bounding box annotations for each object in the form of label files (.txt format) where each line represents an object with the class label, normalized x and y coordinates, and width and height of the bounding box.

This preprocessing technique ensures the dataset is standardized for YOLOv8's requirements, improves computational efficiency, and enhances model performance by normalizing images and organizing bounding box labels.

Model Implementation:

Model Initialization and Configuration

- The model is constructed using the YOLO class from the ultralytics library, which is specifically designed for implementing YOLO models easily.

MODEL BUIDLING

```
[ ]  model = YOLO("yolov8n.pt")
```

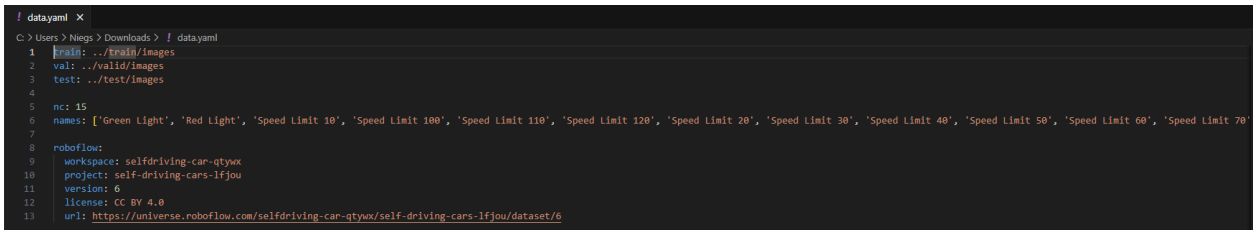
Training Configuration

- The train() function is used to fine-tune the model with custom configurations such as dataset path, number of epochs, batch size, and optimizer.

```
# Training The Final Model
Result_Final_model = Final_model.train(data="/content/drive/MyDrive/car/data.yaml",epochs = 30, batch = -1, optimizer = 'auto')
```

Dataset Configuration

- YOLOv8 reads the dataset configuration from data.yaml, which specifies paths for training, validation, and testing data, as well as the class names. This helps the model to map each bounding box to the correct traffic sign label during training.



- This setup ensures the model correctly associates each bounding box with one of the 15 traffic-related classes

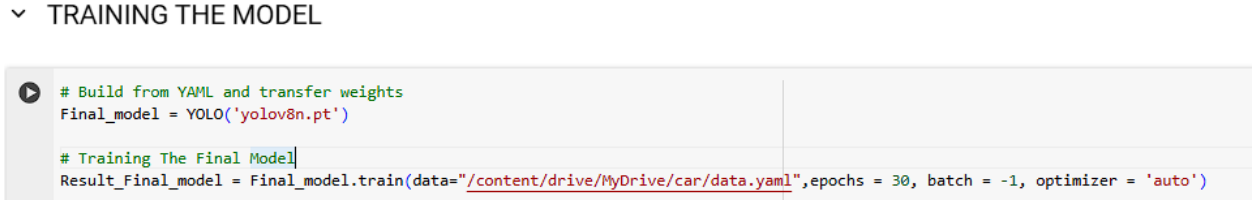
Training the Model:

Dataset Splitting

- The dataset is divided into three subsets: train, val, and test, which is defined in the data.yaml configuration file. These subsets ensure the model is trained on one portion, validated on another to fine-tune hyperparameters, and finally evaluated on unseen data in the test set. This split prevents overfitting and gives an accurate assessment of the model's real-world performance.

Training Process

- To train the model, the following code is used, with key hyperparameters set to optimize learning:



- Epochs: 30 epochs to balance model learning and training duration.
- Batch Size: Auto-adjusted to 75 based on GPU capacity, optimizing resource usage.
- Optimizer: Auto-selected as AdamW, with adaptive learning rate adjustments.

Testing and Evaluation:

Testing

- The model was tested using the test dataset to evaluate its ability to generalize and accurately detect objects in new, unseen images.

Results:

Evaluation Metrics

Precision (mAP@50)

- Indicates the accuracy of the model's detections, specifically how well the model correctly identifies relevant objects without false positives.
- Precision close to 1.0 reflects strong model reliability, as seen in this case where precision was approximately 0.934.

Recall

- Shows the model's ability to detect all relevant objects. Higher recall means the model has a low rate of false negatives (missed detections).
- Recall near 1.0, as seen in the model's results (0.892), indicates that it captures nearly all objects in the images.

Mean Average Precision (mAP@50 and mAP@50-95)

- **mAP@50**: The average precision at an Intersection over Union (IoU) threshold of 0.50. High mAP@50 (0.956) demonstrates that the model is highly accurate at

identifying object boundaries.

- **mAP@50-95**: The average mAP across multiple IoU thresholds (0.5 to 0.95), showing overall robustness in various detection scenarios. With an mAP@50-95 of 0.833, the model handles different IoU thresholds well, indicating high localization accuracy.

Inference Speed

- Measured in milliseconds per image, this includes preprocessing, inference, and postprocessing times. This metric helps gauge the model's suitability for real-time applications.
- Average speeds were approximately:
    - Preprocessing: 0.2 ms
    - Inference: 2.4 ms
    - Postprocessing: 1.6 ms
- These low times make the model appropriate for real-time scenarios, such as live traffic sign detection.

Discussion of Challenges:

Data Preprocessing and Dataset Consistency

- One challenge was ensuring consistency in the dataset, as object detection models like YOLO are sensitive to variations in image resolution, annotation quality, and class balance. For instance, different traffic signs may vary significantly in size or visibility depending on lighting and camera angle.
- **Solution:** We preprocessed the images by resizing and normalizing the pixel and verified annotation quality.

Learning

- Working with object detection needs accurate annotations. Even small inconsistencies in bounding boxes or class labels can impact performance. We learned that high-quality data is foundational for training object detection models.

Conclusion:

This project successfully implemented a YOLOv8-based model for real-time traffic sign detection, achieving high accuracy and recall with an mAP@50 of 0.956 and mAP@50-95 of 0.833. The model demonstrated strong precision and reliable object localization, making it

suitable for applications in autonomous driving. In testing, the model detected objects accurately and quickly, with low inference times that enable real-time deployment. Compared to traditional methods like HOG-SVM, YOLOv8 proved to be a more effective choice for complex, multi-object detection tasks, offering both efficiency and accuracy.