Data Distillation of WayMo dataset in PyTorch

(Apurva Patel – amp2365)

Dataset description

The Waymo Open Dataset is a large-scale dataset developed by Waymo, a self-driving technology company owned by Alphabet Inc. (formerly Google's self-driving car project). A detailed overview of various aspects of the Waymo dataset:

1. Data Modalities:

a. LiDAR Data:

- Waymo's vehicles are equipped with LiDAR (Light Detection and Ranging) sensors that capture 3D point cloud data.
- LiDAR data provides detailed information about the surrounding environment, including the positions and shapes of objects, distances, and geometry.
- It helps in understanding the scene's structure and enables accurate localization and mapping.

b. Camera Data:

- The dataset includes data from high-resolution cameras mounted on Waymo's vehicles.
- Camera images provide visual information about the environment in the form of RGB images.
- These images are crucial for tasks such as object detection, semantic segmentation, and scene understanding.

c. Radar Data:

- Waymo vehicles also use radar sensors to capture additional information about the surrounding objects.
- Radar data provides velocity, range, and angle information about objects, including vehicles, pedestrians, and cyclists.
- It helps in detecting and tracking moving objects and understanding their motion patterns.

2. Annotation:

a. Object Detection and Tracking:

- The dataset is annotated with bounding boxes around various objects of interest, including vehicles, pedestrians, cyclists, and traffic signs.
- These annotations provide ground truth information for training object detection and tracking algorithms.

b. Semantic Segmentation:

- Semantic segmentation annotations label each pixel in the camera images with a category label, indicating the object or scene class it belongs to.
- This information is useful for understanding the scene's semantic structure and identifying different types of objects and road markings.

c. Motion Forecasting:

- Waymo dataset also includes annotations for motion forecasting, which predict the future trajectories of surrounding objects.
- These annotations help in planning and decision-making tasks by anticipating the behavior of other road users.

3. Scenarios:

a. Urban Environments:

- The dataset covers various urban driving scenarios, including city streets, intersections, and residential areas.
- Urban environments present complex challenges such as traffic congestion, pedestrian crossings, and diverse road geometries.

b. Highway Driving:

- Waymo dataset includes data from highway driving scenarios, where vehicles travel at higher speeds with fewer interactions.
- Highway driving presents challenges such as lane changes, merging, and maintaining safe distances from other vehicles.

c. Challenging Conditions:

- The dataset captures data in challenging weather and lighting conditions, including rain, fog, and nighttime driving.
- Handling adverse weather conditions is crucial for ensuring the robustness and safety of autonomous driving systems.

4. Data Volume:

- The Waymo dataset comprises terabytes of data collected over millions of miles of driving in various locations and conditions.
- It includes data from diverse sources such as LiDAR, cameras, and radar sensors, providing rich and comprehensive information for research and development.

5. Usage:

- The Waymo dataset is widely used in industry for training and evaluating autonomous driving algorithms.
- It serves as a benchmark for testing the performance of perception, prediction, planning, and control algorithms in simulated and real-world environments.
- We are using this data to get/obtain and generate a control sequence for an autonomous vehicle.

6. Availability (Our biggest concern):

- While portions of the Waymo dataset are publicly available for research purposes, access to the entire dataset may be restricted due to privacy and proprietary concerns.
- Researchers and developers interested in using the Waymo dataset may need to request access or collaborate directly with Waymo to obtain the data for their projects.
- Waymo also provides tools, documentation, and support for us working with the dataset, enabling them to make meaningful contributions to the field of autonomous driving.

Key Takeaways:

1. Waymo introduces a new large-scale, diverse dataset consisting of 1150 scenes captured across urban and suburban geographies.

2. The dataset includes well-synchronized and calibrated high-quality LiDAR and camera data.

3. The dataset is 15x more diverse than the largest camera+LiDAR dataset available based on a proposed diversity metric.

4. The data is exhaustively annotated with 2D (camera image) and 3D (LiDAR) bounding boxes, with consistent identifiers across frames.

5. Strong baselines for 2D and 3D detection and tracking tasks are provided.

6. The effects of dataset size and generalization across geographies on 3D detection methods are studied.

7. The LiDAR Spherical coordinate system and Evaluation Metrics

8. The LiDAR Spherical coordinate system is based on the Cartesian coordinate system in the LiDAR sensor frame.
9. The dataset provides baselines for detection and tracking of vehicles and pedestrians, with the same method applicable to other object types.

10. Evaluation of tracking results can be done in both 2D image view and 3D vehicle-centric coordinates.

11. The multiple object tracking (MOT) metric is used to evaluate tracking performance, consolidating detection, localization, and tracking into a single metric.

12. The dataset is organized into sequences, each 20 seconds long, with data sampled at 10Hz.

13. Each object in the dataset is annotated with a unique identifier consistent across sequences.

14. IoU thresholds of 0.7 for vehicles and 0.5 for pedestrians are used when computing metrics for all tasks.

Existing Research using this dataset:

Paper 1: Scalable Scene Flow from Point Clouds in the Real World

Key Takeaways:

This paper describes the FastFlow3D model and its application in scene flow estimation from LiDAR point clouds in autonomous vehicles (AVs).

1. **FastFlow3D Model Overview**: FastFlow3D is a model designed for real-time scene flow estimation from LiDAR point clouds in AVs. It consists of three main components:
    - **Scene Encoder**: This utilizes a variant of Point Pillars for encoding point clouds at different spatial resolutions.
    - **Decoder**: This utilizes a 2D convolutional U-Net architecture to fuse contextual information from both frames and obtain point-wise flow predictions.
    - **Shared MLP**: Used to regress an embedding onto point-wise motion predictions.

2. **Training and Dataset**: FastFlow3D is trained using supervised learning with the average L2 loss between the final prediction and ground truth flow annotations. The model is trained on a large-scale dataset derived from the Waymo Open Dataset, comprising 800 training scenes and 200 validation scenes, each recorded at 10 Hz.

3. **Performance Evaluation**: The performance of FastFlow3D is evaluated extensively, showing significant improvements over previous models. The dataset's large size allows for better generalization, and the model demonstrates reasonable predictive performance for various classes of objects, such as vehicles, pedestrians, and cyclists.

4. **Scalability and Generalization**: The model's scalability is highlighted, showing nearly linear growth with a small constant, making it suitable for real-time operation. Additionally, the model's ability to generalize to unlabeled moving objects is explored, revealing some limitations that could be addressed in future work.

5. It suggests that the dataset could serve as a baseline for exploring self-supervised and semi-supervised methods for scene flow estimation in AVs. This is the use case we are working on and it will be very helpful to incorporate some of these features in our modelling.

- FastFlow3D model represents a significant advancement in scene flow estimation from LiDAR point clouds, offering improved scalability, performance, and potential for further research in autonomous driving applications.

<u>Working of FastFlow3D model:</u>

1. **Input Data**:
    - The model takes two successive LiDAR point clouds as input. Each point cloud represents the 3D environment captured by LiDAR sensors mounted on an autonomous vehicle at different time instances.

2. **Scene Encoder**:
    - The scene encoder processes each point cloud to extract meaningful features. It uses a variant of Point Pillars, which aggregates points within fixed vertical columns (pillars) and applies a 2D convolutional network to reduce spatial resolution.
    - Each pillar is parameterized by its center coordinates (cx, cy, cz) and the offset (Δx, Δy, Δz) from the pillar center to the points in the pillar. Additionally, laser features (l0, l1) are appended to the encoding.

- Dynamic voxelization is employed to compute a linear transformation and aggregate all points within a pillar, enhancing efficiency.
- The featurized points within each pillar are summed, which outperforms max-pooling operations used in previous works.

3. **Decoder**:
    - The decoder fuses contextual information from both frames using a 2D convolutional U-Net architecture.
    - It concatenates the embeddings of both encoders at each spatial resolution and applies 2D convolutions to obtain contextual information.
    - The resulting feature map represents a grid-structured flow embedding, which is used for point-wise flow prediction.

4. **Shared MLP**:
    - The shared multi-layer perceptron (MLP) regresses an embedding onto point-wise motion predictions.
    - It takes as input the concatenated point feature and the corresponding flow embedding grid cell and computes the flow vector.

5. **Training**:
    - The model is trained using supervised learning with the average L2 loss between the final prediction for each LiDAR return and the corresponding ground truth flow annotation.
    - Training data is derived from a large-scale dataset obtained from the Waymo Open Dataset (full data not available publicly), which provides tracked 3D objects and diverse scene data.

6. **Performance Evaluation**:
    - The model's performance is evaluated extensively using metrics such as mean pointwise L2 error, percentage of points with error below certain thresholds, precision, and recall.
    - Evaluation is conducted on various subsets of the dataset to analyze predictive performance for different classes of objects and varying densities of point clouds.

7. **Scalability and Generalization**:
    - The model exhibits favorable scaling behavior, making it suitable for real-time operation even with large datasets.
    - Generalization to unlabeled moving objects is explored, revealing limitations and opportunities for future research.

- In a general manner, FastFlow3D model processes LiDAR point clouds through a scene encoder, decoder, and shared MLP to predict point-wise motion flow.

- It is trained using supervised learning on a large-scale dataset and evaluated extensively to analyze its performance, scalability, and generalization capabilities.

Limitations which we can improve upon:

1. The dataset is limited to urban and suburban geographies, which may not fully capture the diversity of real-world driving scenarios.

2. The paper does not provide information on the specific challenges or limitations faced during the data collection process.

3. The evaluation of tracking performance is based on the multiple object tracking (MOT) metric, which may not capture all aspects of tracking systems' performance.

4. The paper focuses on 2D and 3D detection and tracking tasks but does not explore other important aspects of autonomous driving, such as decision-making or path planning.

5. The paper does not discuss the potential biases or limitations in the annotation process, which could impact the accuracy and generalizability of the results.

Paper 2: Evaluating Validity of Synthetic Data in Perception Tasks for Autonomous Vehicles

Key Takeaways:

The paper explores the efficacy of utilizing both simulated and real-world data for training object detection models in the context of autonomous vehicles. It investigates the performance of the YOLOv3 object detection algorithm when trained on datasets derived from simulated environments as well as those collected from real-world scenarios. The study provides insights into the challenges, advantages, and limitations of using simulated data versus real-world data for training autonomous vehicle perception systems.

The introduction outlines the significance of perception systems in autonomous vehicles and the necessity of large and diverse datasets for training robust object detection models. It highlights the emergence of simulation environments as a cost-effective alternative to collecting real-world data but acknowledges the need to evaluate their effectiveness.

Methodology: The paper employs the YOLOv3 object detection algorithm and trains it on datasets obtained from both simulated and real-world environments. It describes the datasets used, including the LGSVL Automotive Simulator (Sim1 and Sim2), the KITTI Vision Benchmark Suite, and the Waymo Open Dataset. The training process involves preprocessing annotations and hyper-parameter tuning using Google Cloud Platform virtual machines with NVIDIA GPUs. The evaluation metrics include Mean Average Precision (mAP) and precision-recall curves.

Results: The study presents detailed results of training and testing the YOLOv3 model on different datasets. It compares the performance of models trained on simulated data versus real-world data and analyzes factors influencing model generalization, such as dataset diversity, graphical fidelity, and location specificity. Key findings include the superior performance of models trained on the LGSVL Sim1 dataset, challenges faced in detecting objects in nighttime scenes, and limitations of models trained on datasets with different anchor sizes.

Working/Methodology in this paper:

Training:

1. **Dataset Preprocessing:** The training process begins with preprocessing the annotations from all datasets into the required format. Annotations from each dataset are aggregated into a single text file, where each row describes the ground truth bounding boxes for a single image. The format of each row includes the path to the image and the coordinates of bounding boxes along with the label.

2. **Model Selection and Implementation:** The YOLOv3 object detection algorithm is chosen for its balance between speed and accuracy. An open-source version of YOLOv3 written using keras with a TFlow backend is used for training.

3. **Hyper-parameter Tuning:** Hyper-parameters such as batch size and learning rate are tuned to achieve optimal loss values for each dataset. Initially, the DarkNet layers are frozen, and training starts with a batch size of 16 and a learning rate of 10^-3. For fine-tuning, the learning rate is reduced to 10^-4, all layers are unfrozen, and the batch size is reduced to 2. This process ensures that the model converges to low stable losses.

4. **Training Infrastructure:** Trainings are performed on a combination of Google Cloud Platform (GCP) virtual machines with NVIDIA P100 GPUs and desktop computers with NVIDIA RTX 2080 GPUs. The choice of infrastructure allows for efficient training and utilization of GPU resources.

Testing:

1. **Test Setup:** The trained models are tested on separate test sets derived from the same datasets used for training. The performance of each model is evaluated using Mean Average Precision (mAP) and precision-recall curves.

2. **Evaluation Metrics:** The primary evaluation metric used is mAP, which measures the average precision across different classes of objects. The mAP is calculated by determining the precision and recall values for each class at different IoU thresholds (from 0.5 to 0.95), and then averaging the precision values for each class. Precision-recall curves are also plotted to visualize the model's performance across different confidence score thresholds.

3. **Analysis of Results:** The results of testing are analyzed to assess the performance of models trained on simulated versus real-world data. Factors such as dataset diversity, graphical fidelity, and location specificity are considered in understanding the models' generalization capabilities. Insights are drawn from the comparison of mAP values and precision-recall curves across different datasets.

4. **Identification of Limitations:** Any limitations or challenges encountered during testing, such as difficulties in nighttime detection or issues with cross-traffic detection, are identified and discussed. These insights contribute to a deeper understanding of the strengths and weaknesses of the trained models and the datasets used for training and testing.

- In all, training and testing process involves a systematic approach to optimizing model performance, evaluating its effectiveness across different datasets, and gaining insights into the factors influencing object detection in autonomous vehicles.

Mathematical Evaluation metrics:

1. **Intersection over Union (IoU):**
   IoU is a measure commonly used in object detection tasks to evaluate the overlap between predicted bounding boxes and ground truth bounding boxes. It quantifies how well a predicted bounding box aligns with the true position of the object in the image.

   Mathematically, IoU is defined as the ratio of the area of intersection to the area of union between two bounding boxes:

$$\text{IoU} = \frac{Area\ of\ Intersection}{Area\ of\ \text{Union}}$$

   - **Area of Intersection (A_intersect):** This represents the region where the predicted bounding box and ground truth bounding box overlap.
   - **Area of Union (A_union):** This is the combined area covered by both bounding boxes.

   Given the coordinates of the bounding boxes, IoU is computed using the following steps:

   1. Calculate the coordinates of the intersection area by finding the overlap between the two bounding boxes.
   2. Compute the area of intersection using the coordinates obtained in step 1.
   3. Calculate the area of each bounding box.
   4. Determine the area of union by adding the areas of both bounding boxes and subtracting the area of intersection.

5. Finally, divide the area of intersection by the area of union to obtain the IoU score.

2. **Mean Average Precision (mAP):**
mAP is a metric used to assess the performance of object detection models. It calculates the average precision across multiple classes or object categories. The process involves computing the precision-recall curve for each class, which is then used to calculate the average precision (AP) for that class. The mAP is the mean of the AP values across all classes.

$$\text{mAP} = \frac{1}{Number\ of\ classes} \sum_{i=1}^{number\ of\ classes} AP_i$$

- *APi* represents the average precision for class i.
- The average precision (AP) for each class is calculated by integrating the precision-recall curve and averaging the precision values at different recall levels.

The precision-recall curve is generated by varying the confidence threshold for object detection. It plots the precision (fraction of true positive detections among all positive predictions) against recall (fraction of true positive detections among all ground truth instances) at different threshold levels.

To compute the average precision for a specific class, the area under the precision-recall curve is divided by the total number of ground truth instances for that class.

By averaging the AP values across all classes, the mAP provides a comprehensive measure of the model's overall performance in detecting objects across different categories.

These mathematical concepts form the basis for evaluating and comparing the effectiveness of object detection algorithms and models. This will enable us to quantitatively assess the accuracy and reliability of the systems.

Limitations which we wil improve upon:

1. **Dataset Bias:** One limitation is dataset bias, where models trained on a specific dataset may exhibit poor generalization to unseen data due to differences in dataset characteristics such as scene complexity, object density, and environmental conditions.

2. **Localization Errors:** Object detection models may suffer from localization errors, leading to inaccurate bounding box predictions. Factors such as occlusions, perspective distortion, and variations in object scale can contribute to these errors.

3. **Limited Dataset Diversity:** Simulated datasets often lack the diversity of real-world scenarios, leading to models that are proficient in simulated environments but struggle to generalize to diverse real-world conditions.

4. **Anchor Box Selection:** The choice of anchor box sizes in YOLOv3 significantly impacts model performance. Inaccurate anchor box selection, especially across datasets with different aspect ratios and object scales, can lead to suboptimal detection results.

5. **Simulation Fidelity:** While simulated datasets aim to replicate real-world scenarios, they may fail to capture the complexity and variability of real-world environments accurately. Limited fidelity in simulation environments can hinder model generalization and real-world applicability.

6. **Scenario-Specific Performance:** Models trained on datasets from specific locales may exhibit performance discrepancies when applied to different geographic regions or driving scenarios, highlighting the importance of locale-specific training and scenario coverage.

7. **Algorithmic Limitations:** The selected object detection algorithm, YOLOv3, is exhibiting limitations such as poor performance for overlapping objects and challenges in detecting small or distant objects, necessitating exploration of alternative algorithms for improved performance.

Addressing these limitations will require a holistic approach, including the acquisition of diverse and representative datasets (suggested: WayMo cross-validation), refinement of simulation environments for increased fidelity, optimization of training methodologies, and exploration of advanced object detection algorithms tailored to specific application domains for generating a control sequence of the AV.

My Implementation:

1. **Data Preprocessing**:
    - The first step is to preprocess the Waymo dataset. This involves loading the dataset, which includes front camera images .
    - The dataset is then split into training and testing sets. This splitting allows us to train the model on a subset of the data and evaluate its performance on unseen data.

2. **Neural Network Architecture**:
    - We define a convolutional neural network (CNN) architecture using PyTorch. CNNs are commonly used for image-related tasks due to their ability to capture spatial hierarchies in data.
    - The CNN architecture consists of several convolutional layers followed by fully connected layers. The convolutional layers extract features from the input images, while the fully connected layers perform the final classification or regression tasks.

3. **Training Loop**:
    - The training loop involves iterating over batches of training data. In each iteration:
        - We input the images into the neural network and obtain predictions for the control signals.
        - We compute the loss between the predicted control signals and the ground truth control signals.
        - We backpropagate the loss through the network to update the weights using an optimizer (e.g., Adam optimizer).
    - This process is repeated for multiple epochs until the model converges and the loss stabilizes.

4. **Evaluation**:
    - Once the model is trained, we evaluate its performance on the testing dataset.
    - We measure metrics such as mean squared error between the predicted and ground truth control signals to assess the model's accuracy.

5. **Fine-tuning and Optimization**:
    - We experiment with different network architectures, hyperparameters, and optimization strategies to improve the model's performance.
    - Fine-tuning involves adjusting the model's parameters based on the performance on the testing dataset.
    - We may also fine-tune the model on additional data if available to further improve its performance.

6. **Prediction**:
    - After the model is trained and evaluated, we can use it to predict control signals for new data.
    - We input new images into the trained model and obtain predictions for the corresponding control signals.
    - These predictions can then be used to control the vehicle in real-time based on the model's outputs.

Problems I am facing:

1. Currently I have completed the architecture, and it runs in training as I am utilizing jupyter notebook.
2. Also the dataset storage in a bucket is expensinve.
3. Also can you suggest some checkpointing mechanism for auto reload of results.