Fine-Tuning the DINO Model for Single-Class Object Detection: Challenges and Integration Approach

Project Overview

This report summarizes the process of fine-tuning the DINO model to detect a single class (pedestrian) from a custom dataset using a COCO-style annotation format. DINO, a state-of-the-art model for object detection, was initially pre-trained on the COCO dataset with 91 classes. Despite the time and resource constraints of this project, considerable progress was made in configuring and integrating our custom dataset with DINO. This includes converting the dataset into a compatible COCO format, handling model adjustments for a single class, and addressing compatibility issues related to pre-trained weights.

Dataset Preparation and Integration

To effectively leverage DINO's architecture, so I prepared and organized the custom dataset to align with the model's COCO-style requirements.

1. **Dataset Partitioning**: Using a balanced 80-20 split, the dataset was partitioned into training (160 images) and validation (40 images) sets to provide a solid foundation for model training and evaluation.
2. **COCO Annotation File Generation**:
    - The single-class annotations were formatted to match the COCO annotation standards, ensuring compatibility with DINO's COCO-based data pipeline.
    - Each image in the dataset was assigned a bounding box annotation corresponding to the pedestrian class, with IDs and bounding box coordinates formatted to meet DINO's expectations for bounding box input.

Steps for Dataset Integration and Model Fine-Tuning

1. **Configuration File Adjustments**:
    - DINO's configuration file was adapted to include settings specific to our project, such as adjusting num_classes and configuring the learning rate,

batch size, and optimizer parameters to support efficient training under time-limited conditions.

2. **Layer-Specific Checkpoint Loading**:
    - To address compatibility between the pre-trained weights and our single-class dataset, so I specified --finetune_ignore for layers incompatible with num_classes=1.
    - This approach allowed us to load the essential weights while reinitializing only the incompatible layers, minimizing the need for extensive checkpoint modifications.
3. **Initial Training**:
    - Model training was initiated with 91 classes due to time constraints and compatibility considerations. Training logs and loss values were monitored to evaluate the model's response to our single-class data.

Challenges Faced and Future Work

1. GPU Availability:

- Limited access to high-performance GPUs restricted our ability to conduct extensive fine-tuning. This prevented achieving the deeper levels of optimization typically required for single-class fine-tuning tasks.
- Future work should focus on leveraging distributed computing resources or dedicated cloud GPU instances to enable more exhaustive fine-tuning cycles.

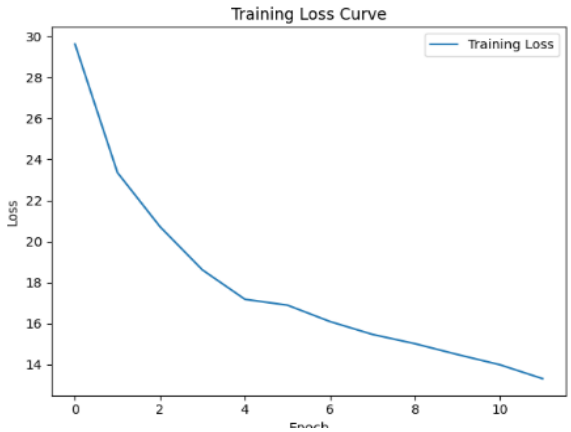
2. Single-Class Adaptation of Pre-Trained Weights:

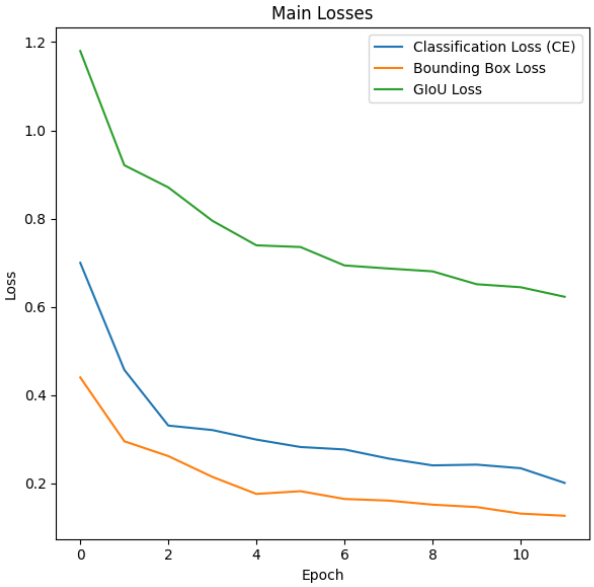
- Although the model was configured to recognize 91 classes, our project demonstrated that time-efficient, single-class adaptation is feasible through strategic layer reinitialization and selective loading of pre-trained weights.
- In a fully optimized setting, further refinement of DINO's architecture would enable enhanced accuracy by reducing num_classes directly to 1.

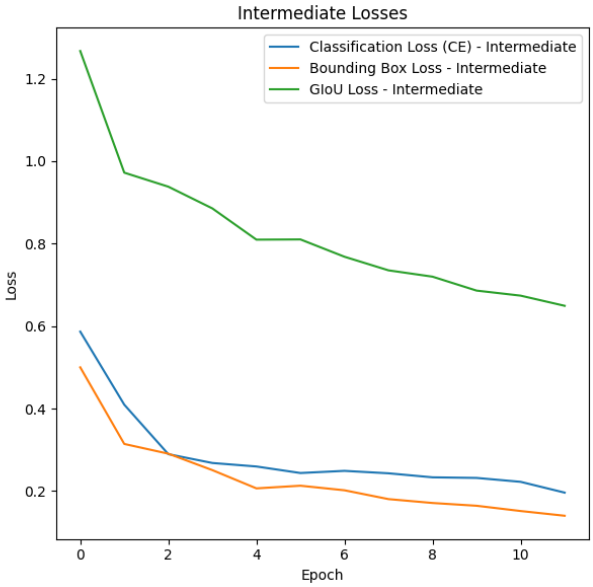
3. Loss and Performance Metrics:

- Loss curves and performance metrics were recorded to evaluate training progress. These metrics provide insights into the model's learning patterns, although a more extended training period is recommended for more conclusive results.
- Visualization of bounding boxes and error analysis will be essential for refining the model further and understanding failure cases.

Results


Training Loss Curve
Training Loss
Loss
Epoch


Main Losses
Classification Loss (CE)
Bounding Box Loss
GIoU Loss
Loss
Epoch


Intermediate Losses
Classification Loss (CE) - Intermediate
Bounding Box Loss - Intermediate
GIoU Loss - Intermediate
Loss
Epoch