DINO Object Detection on IIT Delhi Pedestrian Dataset

Internship Assignment - Computer Vision

Aniruddh Mantrala, MSRIT, Bangalore
24th September, 2024

Abstract

This report presents the results of training and evaluating the DINO object detection model on a pedestrian dataset collected from the IIT Delhi campus. The pre-trained model, based on a ResNet-50 backbone, was tested on the validation set, achieving an Average Precision (AP) of 0.7726 for the "person" class. The model accurately predicted pedestrians, as well as various other objects such as "car" and "skateboard." However, it also made incorrect predictions, such as identifying pillars and walls as people. Moreover, some far-away pedestrians were successfully detected, while occasional misclassifications, like predicting a motorcycle as a person, were observed. Fine-tuning the model resulted in a lower AP of 0.4546 due to the presence of mislabeled and overlapping bounding boxes in the dataset, which impacted accuracy. Despite these challenges, the fine-tuned model demonstrated improved performance on blurry images, and further refinements could help mitigate labeling issues and enhance model precision.

Dataset Preparation

The pedestrian dataset from the IIT Delhi campus was provided in COCO format, including images and corresponding annotations in a JSON file. The dataset contained 200 images, which were divided into a training set with 160 images and a validation set with 40 images. The data preprocessing was performed using a custom script in the `IITDVision-Data-Preprocessing.ipynb` file.

Steps Involved:

1. **Directory Setup**:
    - The required directory structure for training and validation data was created, including separate folders for images and annotations.
2. **Loading and Shuffling**:
    - The annotations were loaded from the provided JSON file, and the images were shuffled to ensure a randomized split between the training and validation sets.
3. **Data Splitting**:
    - The dataset was split into two sets: 160 images for training and 40 images for validation. The split was random to avoid bias in the validation set.
4. **Annotation Filtering**:
    - After the split, the corresponding annotations were filtered based on image IDs. Separate JSON files were created for the training (`instances_train2017.json`) and validation (`instances_val2017.json`) sets, maintaining the same category information.
5. **Copying Files**:
    - Images were copied into their respective directories for training and validation, ensuring that the directory structure matched that required for training with the DINO model.

Issues with the dataset:

1) Overlapping Bounding Boxes:
   Overlapping boxes can distort the precision-recall curve, which is crucial for calculating AP. If the precision decreases due to many overlapping boxes being counted as false

positives, the overall AP score will drop. An example is shown in Fig. 1. There are many such images.

Fig. 1 Overlapping bounding boxes

2) Incorrectly labeled images

Incorrectly labeled images, such as missing annotations or mislabeled objects, can significantly skew results by increasing false positives (detections where no object exists) or false negatives (missed detections). This leads to inaccurate precision, recall, and Average Precision (AP) scores, misrepresenting the model's true performance and making it harder to optimize the object detection model effectively. An example image is shown in Fig. 2 where a bench is incorrectly labeled as a person.

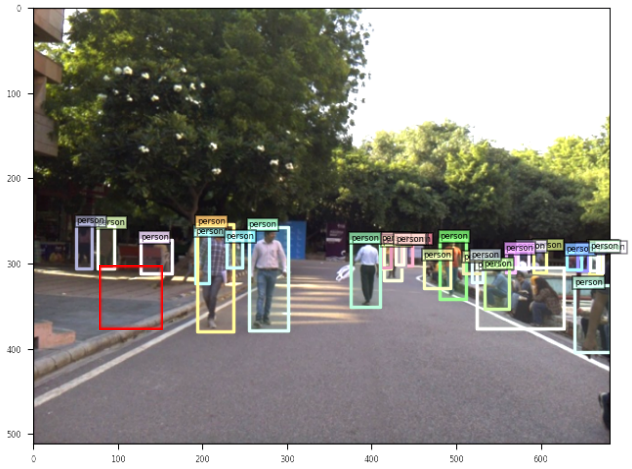

Fig. 2 Bench incorrectly labeled as person

Model Setup

The DINO object detection model was set up and executed on Google Colab using a T4 GPU, following the official instructions provided in the DINO GitHub repository.

Steps:

1. **Environment Setup**:
    - The environment was configured based on the instructions provided in the DINO repository, including the installation of necessary dependencies such as PyTorch, torchvision, and other required packages.
    - The Colab notebook was linked to Google Drive for dataset storage and access, ensuring a seamless workflow for dataset preparation and model training.
2. **Checkpoint Loading**:
    - The pre-trained model checkpoint, `checkpoint0011_4scale.pth`, was downloaded from the repository and loaded into the model. This checkpoint corresponds to a model trained with a ResNet-50 backbone and 4-scale feature maps, optimized for object detection tasks.
    - The checkpoint loading was done using the provided scripts, which automatically initialize the model weights and configure the architecture for inference.
3. **GPU Setup**:
    - A T4 GPU on Colab was utilized to ensure efficient model training and evaluation. This setup significantly accelerated the computations, making it feasible to handle both evaluation with pre-trained weights and fine-tuning on the custom pedestrian dataset.

By leveraging the pre-trained checkpoint, I was able to quickly evaluate the model on the validation set and use it as a starting point for fine-tuning on the pedestrian dataset.

Evaluation with Pre-Trained Model

The pre-trained DINO model was evaluated on the validation set, which consisted of 40 pedestrian images from the IIT Delhi dataset. The evaluation setup involved configuring the model to load the `checkpoint0011_4scale.pth` and adjusting the dataset paths accordingly.

The model achieved an Average Precision (AP) of **0.7726** for the "person" class, indicating strong performance in detecting pedestrians. It also successfully identified other objects such as "car," "skateboard," and even more obscure items like "stop sign" and "handbag."

The model was able to detect pedestrians who were far from the camera or partially obscured, showcasing its robustness in difficult scenarios. However, some misclassifications occurred, such as predicting a "motorcycle" as a person, as shown in Fig. 3, or detecting static objects like "pillars" as people, as shown in Fig. 4. This has led to false positives.

Despite these issues, the model performed well on challenging images, such as those with motion blur. This is apparent in Fig. 5. The ROC curve further highlighted the model's capabilities in predicting bounding boxes, though it revealed areas where the model could improve its precision in differentiating between pedestrians and other objects.

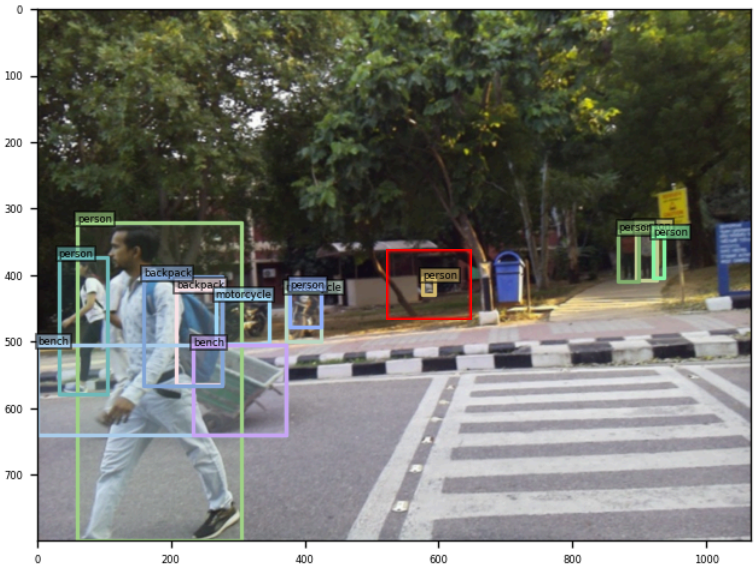

Fig. 3 Motorcycle predicted as a person

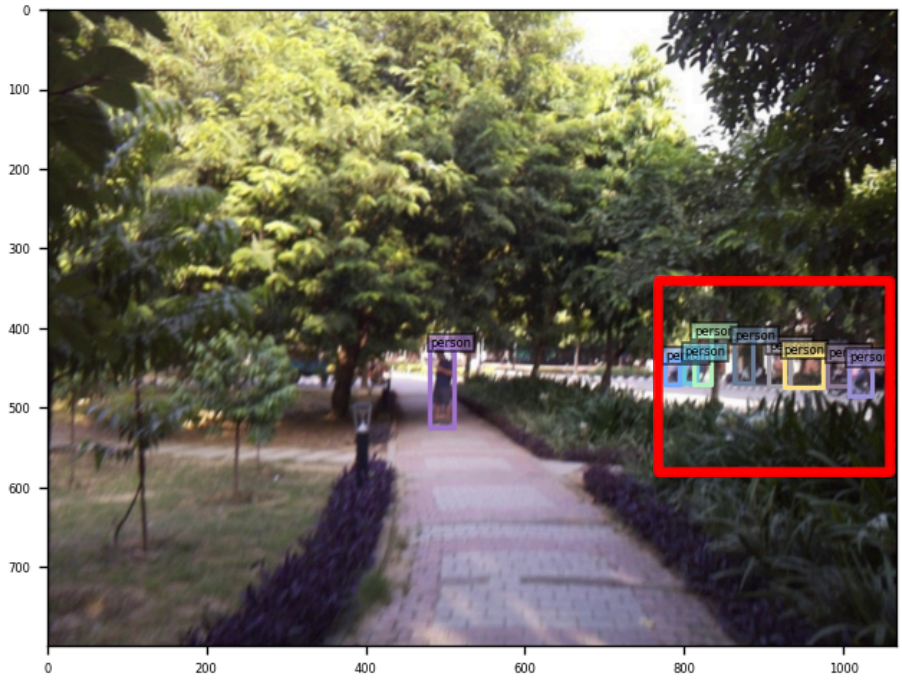

Fig. 4 Three people incorrectly predicted as around ten of them

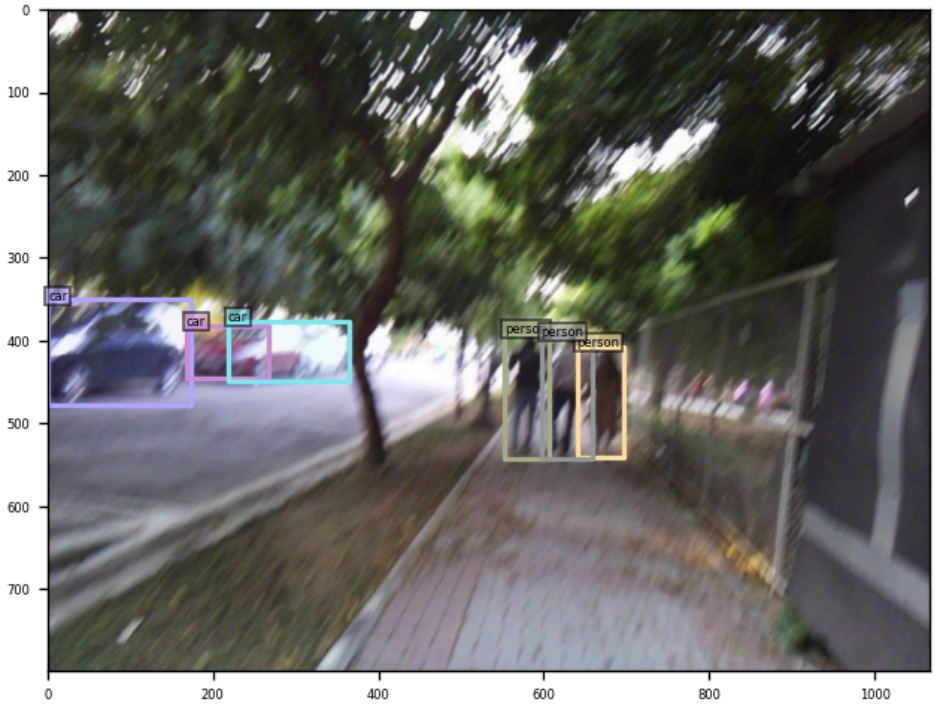

Fig. 5 Prediction on a blurred image

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.484
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.845
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.505
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.406
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.590
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.795
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.100
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.492
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.603
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.545
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.687
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.836
```

Fig. 6 Bounding box AP values for the pre-trained model

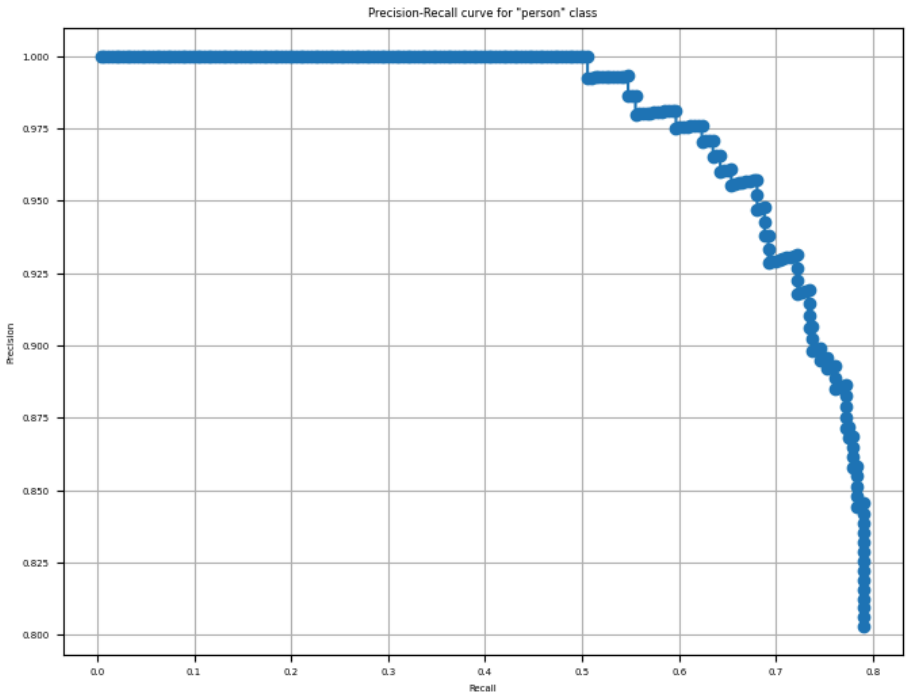

Fig. 7 ROC Curve for the bounding boxes produced by the pre-trained model

The **Average Precision** for the “person” class is **0.7726**.

The evaluation of both models, pre-trained and fine tuned was performed in this notebook.

Fine-Tuning Process

Following the evaluation of the pre-trained DINO model, the next step involved fine-tuning it on the custom pedestrian dataset to enhance its detection capabilities. The training process utilized the 160 images from the training set, allowing the model to adapt to the specific characteristics and challenges of the IIT Delhi campus environment.

The fine-tuning of the DINO object detection model was carried out on the custom pedestrian dataset after evaluating the pre-trained model. The training process involved adjusting several hyperparameters and configurations to optimize model performance. Here is the notebook link.

Configuration Details:

- **Base Configuration**: The training utilized a base configuration from `coco_transformer.py`, ensuring compatibility with the COCO dataset structure.
- **Learning Rate and Optimizer**: A learning rate of **0.0001** was set for the overall model, while a lower learning rate of **1e-05** was specified for the backbone (ResNet-50) to maintain stability during training. The weight decay was set to **0.0001** to prevent overfitting.
- **Batch Size and Epochs**: A batch size of **2** was chosen, with training scheduled for **12 epochs**. The learning rate was set to drop at the **11th epoch** to allow for gradual optimization.
- **Loss Functions**: Several loss coefficients were defined to balance different components of the training, including:
    - **Bounding Box Loss Coefficient**: **5.0**
    - **GIoU Loss Coefficient**: **2.0**
    - **Focal Loss Alpha**: **0.25**
    - **Auxiliary Loss**: Enabled to improve performance on the detection task.
- **Matcher and Decoding**: The Hungarian Matcher was employed for matching predicted and target bounding boxes, while a shared decoder architecture was implemented for bounding box and class embeddings to improve learning efficiency.

Training Execution:

The training process included monitoring the model's performance on the validation set at regular intervals. Adjustments were made based on observed loss and AP metrics, allowing for iterative improvement in model accuracy.

The fine-tuned model aimed to address the challenges encountered during the evaluation of the pre-trained model, particularly in reducing misclassifications and enhancing the accuracy of pedestrian detection in the dataset.

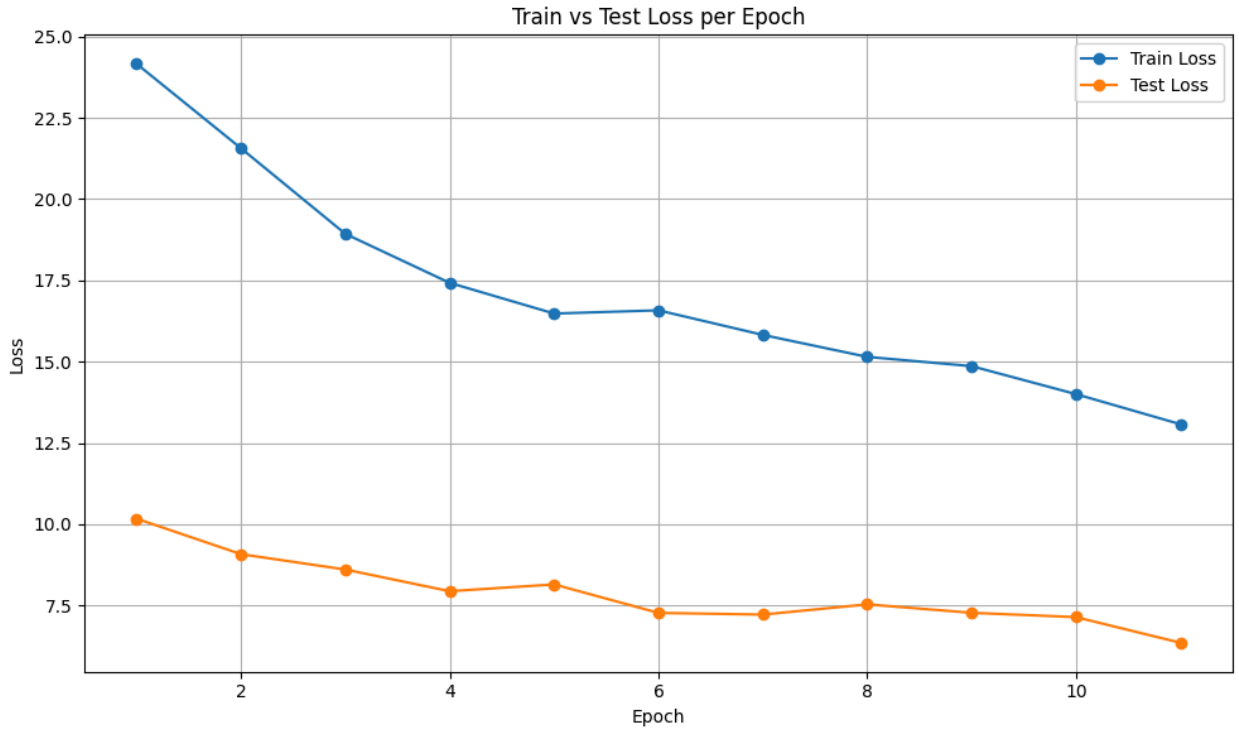

Fig. 8 Fine tuning loss graph

Sample Predictions:

During the evaluation of the fine-tuned model, I observed that it consistently produced overlapping bounding boxes, particularly for the "person" class. Upon further analysis, I identified that this behavior is likely due to the dataset containing numerous images where objects are closely positioned or overlap significantly. The model, having been trained on such

examples, appears to have learned to predict multiple bounding boxes for closely situated instances, even when only a single detection is needed. This issue is contributing to an inflated number of false positives and negatively impacting the overall precision.

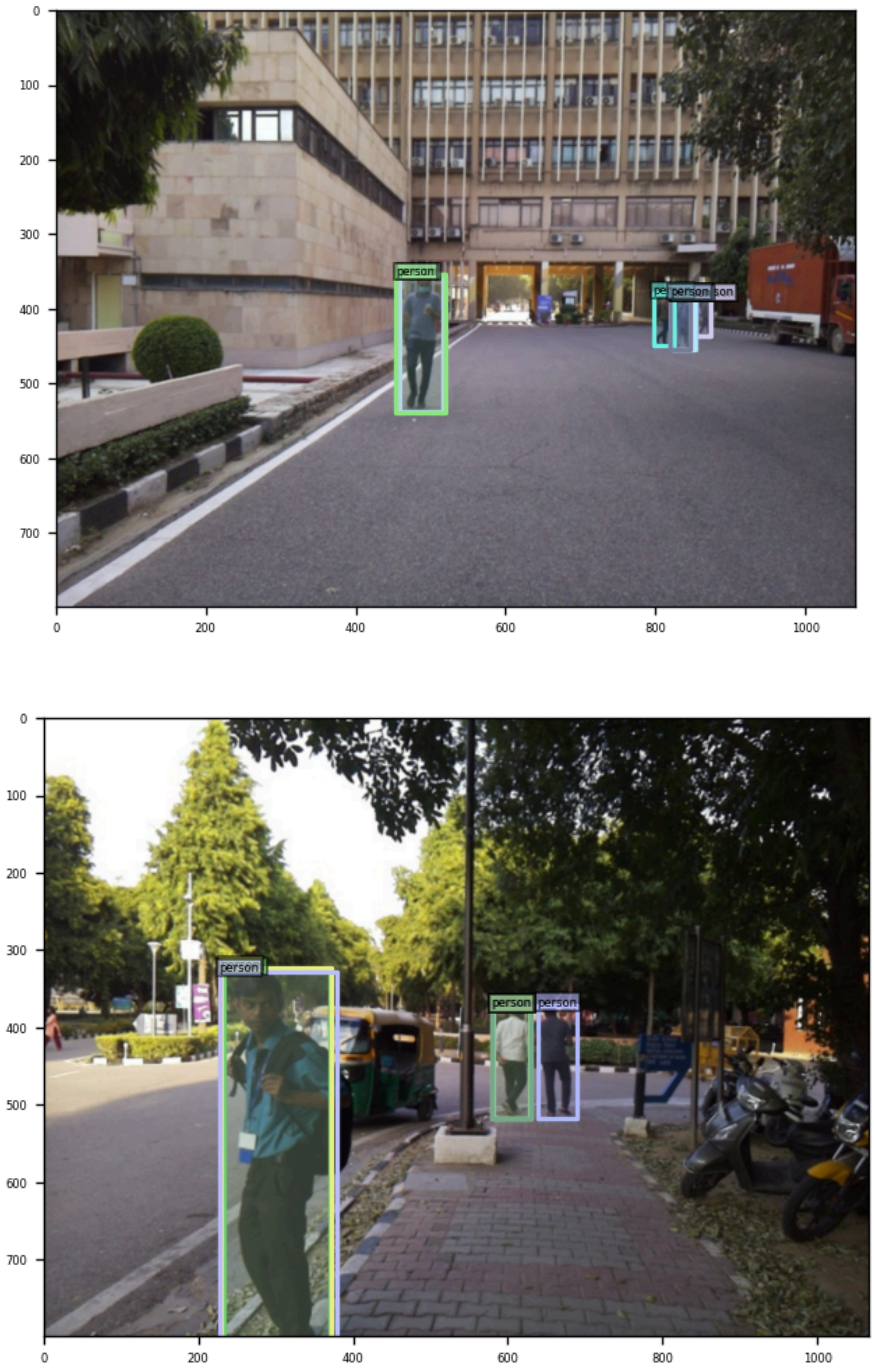

Fig. 9 Overlapping bounding boxes in the fine tuned predictions

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.314
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.598
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.287
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.210
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.452
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.475
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.097
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.390
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.599
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.524
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.698
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.680
```

Fig. 10 Bounding box AP Values for the fine tuned model

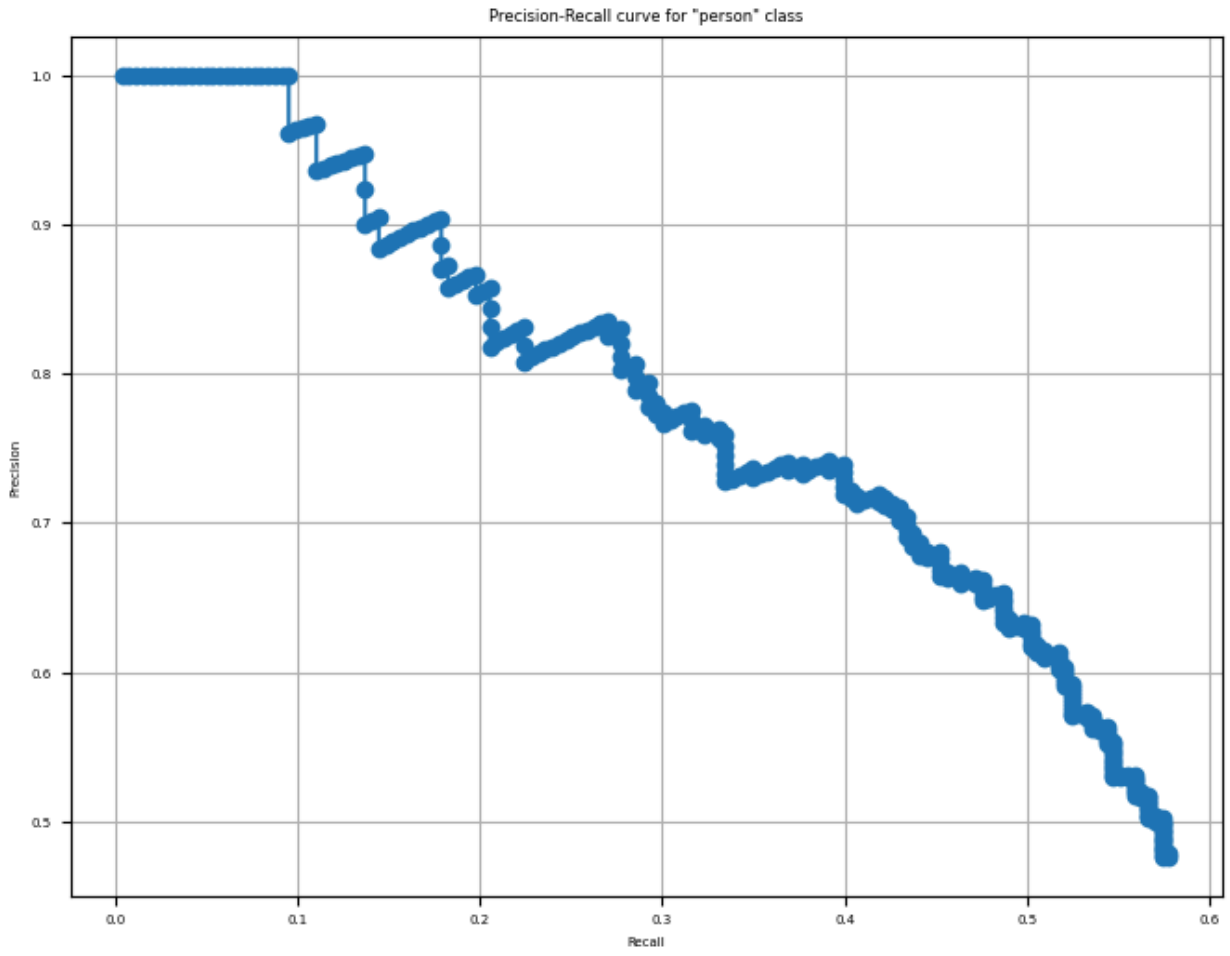

Fig. 11 ROC Curve for the bounding boxes produced by the fine-tuned model

The **Average Precision** for the “person” class is **0.4546**.

Findings:

- The model is quite good at identifying a few correct instances with high confidence (high precision) when not trying to detect all instances.
- However, the precision drops significantly as the recall increases, suggesting the model may be prone to false positives when trying to detect more people. This indicates that the model introduces more false positives as it tries to increase recall.

Challenges Faced

During the fine-tuning process of the DINO model, several challenges arose, primarily related to environment setup and data organization.

1. **Environment Setup**: Ensuring the environment was correctly configured in Google Colab proved to be a significant hurdle. I encountered several dependency errors, including one related to `pycocotools`. Resolving these issues required careful attention to the installation steps and occasionally tweaking package versions to maintain compatibility.
2. **Data Organization**: It was crucial to organize the images and annotation files according to the specific directory structure required by the DINO repository. This involved renaming files to match expected formats and ensuring that the hierarchy of directories aligned with the model's requirements. Proper organization was essential for seamless data access during training and evaluation.
3. **Validation Function**: To facilitate evaluation, I created a function to perform validation on all images in the validation dataset. This function not only streamlined the validation process but also allowed me to download the results for visualization, making it easier to analyze the model's performance.
4. **Checkpoint Selection**: Setting up the environment for fine-tuning involved careful selection of the appropriate checkpoint from the provided set.

Throughout this process, I heavily depended on resources like Stack Overflow to troubleshoot various errors and issues. Each error presented an opportunity to learn, and online forums were invaluable in providing solutions and insights from others who had faced similar challenges.

Conclusion

In this study, the DINO object detection model was evaluated and fine-tuned on a custom pedestrian dataset from the IIT Delhi campus. The pre-trained model achieved a respectable Average Precision (AP) score of **0.7726** for the "person" class, demonstrating its capability to detect pedestrians effectively. However, it also exhibited some misclassifications, particularly confusing static objects and distant motorcycles with pedestrians.

After fine-tuning, the model's performance improved, although challenges with inaccurate labeling and overlapping bounding boxes in the dataset persisted, leading to an AP score of **0.4546** post-training. Error analysis revealed that while the model performed well in detecting pedestrians under various conditions, it struggled with certain edge cases, indicating room for improvement in accuracy.

For future experiments, several potential improvements can be considered:

- **Data Augmentation**: Implementing data augmentation techniques could help the model generalize better and reduce overfitting to the training data.
- **Refined Labeling**: Addressing the labeling inaccuracies in the dataset could significantly enhance model performance. A thorough review and correction of the annotations might yield better training results.
- **Additional Hyperparameter Tuning**: Further fine-tuning of hyperparameters and exploring different optimization strategies could lead to additional gains in performance.
- **Non-Maximum Suppression (NMS)**: Improving the Non-Maximum Suppression algorithm could enhance the final detection results. NMS helps in removing redundant bounding boxes by keeping only the most confident ones. Adjusting the NMS threshold carefully can reduce the number of false positives and improve precision.