CSCI 677 –Advance Computer Vision- Fall 2024 - HW 4

USC ID: 1154164561
NAME: Srija Madarapu
EMAIL: madarapu@usc.edu

Code:

https://colab.research.google.com/drive/1-N8h_piZMOTYz_-5Ng5_ZbX8HH9uDSPJ?usp=sharing

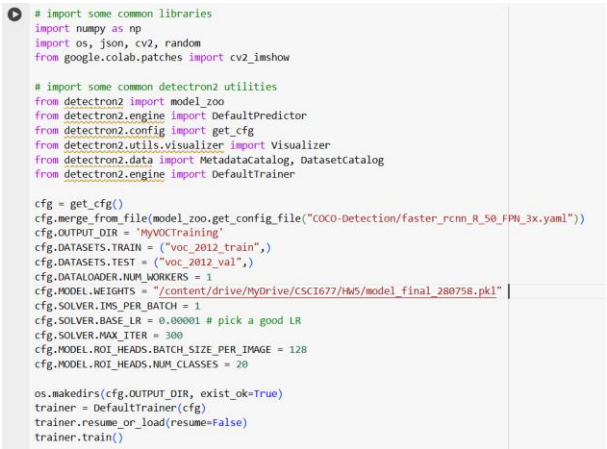
1. The code you have written (in report PDF, .py file or .ipynb file, first is preferred for grading), including brief descriptions/comments of each function/training block.

R-CNN:

1. Repository Cloning and Installation: Git Clone: The first step is cloning the Detectron2 repository from GitHub. Dependencies Installation: Using distutils to run the setup.py of Detectron2 and then installing its dependencies using pip.
2. Dataset Preparation: Unregistering Datasets: Ensures that if the VOC2012 dataset is already registered, it will be removed and re-registered. Parsing VOC XML Annotations: Defines a function parse_voc_xml that parses the VOC XML annotation files. It extracts information such as:
    - Image filename
    - Image dimensions (width, height)
    - Object annotations (category, bounding box coordinates)

Dataset Dictionary: The function get_voc_dicts converts the XML annotations into a dictionary format that Detectron2 can use. This includes the file paths of images, bounding boxes, and associated class labels.
3. Dataset Registration: VOC2012 dataset is registered for training and validation using DatasetCatalog.register. The dataset consists of 20 object classes (like "aeroplane", "car", "cat", etc.) which are mapped to category IDs using category_id_map. Metadata for visualization is also set, such as class names, which is used when visualizing the data with Detectron2.
4. Data Visualization: Visualizer Setup: A few random images from the VOC dataset are selected and visualized with bounding boxes drawn around the objects. Using Visualizer: The Visualizer class from Detectron2 is used to overlay annotations onto the images. The cv2_imshow function is used to display images in Google Colab.
5. Model Configuration and Training: Configuration Setup: The code sets up the model configuration (cfg) using a pre-trained model (Faster RCNN “R50-FPN, 3x” model, ID# 137849458) from the Detectron2 model zoo.
    - Configures training parameters such as batch size, learning rate, number of iterations, and model weights (initialized from a pre-trained model).

Trainer: The DefaultTrainer is used for training the model. It handles the setup of data loaders, model, optimizer, etc. Training: The training process is initiated with trainer.train() which trains the model for 3000 iterations.



```
# import some common libraries
import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
from detectron2.engine import DefaultTrainer

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.OUTPUT_DIR = 'MyVOCTraining'
cfg.DATASETS.TRAIN = ("voc_2012_train",)
cfg.DATASETS.TEST = ("voc_2012_val",)
cfg.DATALOADER.NUM_WORKERS = 1
cfg.MODEL.WEIGHTS = "/content/drive/MyDrive/CSCI677/HW5/model_final_280758.pkl"
cfg.SOLVER.IMS_PER_BATCH = 1
cfg.SOLVER.BASE_LR = 0.00001 # pick a good LR
cfg.SOLVER.MAX_ITER = 300
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 20

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

6. Model Testing: Post-Training: After training, the model weights are saved, and the model is loaded for inference. Prediction Setup: The DefaultPredictor is used to make predictions on images from the dataset using the trained model. A custom threshold for detection (0.7) is applied, meaning detections with a confidence score below 0.7 are discarded. Inference and Visualization: For each selected image, the trained model makes predictions, and these are visualized with bounding boxes and class labels using the Visualizer.

```
[ ]  # cfg now already contains everything we've set previously. We changed it a little bit for infer
     cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "/content/MyVOCTraining/model_final.pth")  # pa
     cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set a custom testing threshold
     predictor = DefaultPredictor(cfg)
```

```
[ ]  from detectron2.utils.visualizer import ColorMode
     import cv2
     from detectron2.engine import DefaultPredictor
     from detectron2.utils.visualizer import Visualizer
     from google.colab.patches import cv2_imshow


     dataset_dicts = get_voc_dicts("/content/datasets/VOC2012/JPEGImages", "/content/datasets/VOC2012/Annotations")
     for d in selected_filenames:
         im = cv2.imread(d)
         print("Selected filenames:", d)
         outputs = predictor(im)  # format is documented at https://detectron2.readthedocs.io/tutorials/models.html#model-output-format
         v = Visualizer(im[:, :, ::-1],
                        metadata=voc_metadata,
                        scale=0.5,
                          # remove the colors of unsegmented pixels. This option is only available for segmentation models
         )
         out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
         cv2_imshow(out.get_image()[:, :, ::-1])
```

DETR:

1. Clone DETR repository: The script starts by cloning the official DETR GitHub repository, which contains the necessary code for training and evaluation.

2. Preparing Annotations: The script gathers XML annotation files from the VOC 2012 dataset, specifically from the folder /content/VOCdevkit/VOC2012/Annotations. These annotations are in the VOC format, which needs to be converted to COCO format for compatibility with DETR.

3. Writing Annotation Paths: The paths of all the XML annotation files are written to a file called anno_path_list.txt, which will later be used for annotation processing.

4. Defining COCO Class Labels: A list of COCO categories (COCO_CLASSES) is defined, which includes various object classes like 'person', 'car', 'dog', etc. Some class names are mapped to new names using a TRANSLATE dictionary (e.g., "motorcycle" to "motorbike") to ensure compatibility with the COCO dataset format.

5. Creating Label Mapping: A coco_labels.txt file is created, which contains the class labels used by COCO, and any "N/A" entries are replaced with generic names (e.g., empty0, empty1).

6. Creating Directory Structure: Directories are created for storing the COCO format dataset, including subdirectories for training and validation images (train2017, val2017) and annotations (annotations).

7. Converting VOC Annotations to COCO Format:A conversion script is provided (voc2coco.py) that converts the VOC XML annotations to COCO JSON format. This script performs several tasks: Parsing XML Files: It reads the VOC XML annotations to extract information about images and objects. Creating COCO Annotations: For each object in an annotation, a corresponding entry is created in the COCO JSON format, including bounding box coordinates and category IDs. Generating COCO JSON: After processing all annotations, a COCO-format JSON file (output.json) is generated with fields like images, annotations, and categories.

8. Copying Images: The script copies the images from the VOC dataset's JPEGImages directory into the corresponding COCO dataset directories (train2017 and val2017).

9. Training the DETR Model: The DETR model is trained using the converted COCO-format dataset. The training process is initiated using the main.py script from the DETR repository: The script specifies various hyperparameters for the training, such as the batch size, learning rate, number of epochs, and the path to the pre-trained DETR model.The model is fine-tuned using the VOC dataset converted into the COCO format.

```
[ ]  !python /content/detr/main.py \
       --batch_size 2 \
       --resume https://dl.fbaipublicfiles.com/detr/detr-r50-e632da11.pth \
       --coco_path /content/VOC_coco_format \
       --output_dir /content/VOC_coco_format \
       --lr 0.0001 \
       --lr_backbone 0.00001 \
       --epochs 1
```

10. Evaluating the DETR Model: After training, the model is evaluated using the same main.py script, and the evaluation results are logged.

```
[ ]  !python /content/detr/main.py \
       --batch_size 2 \
       --resume https://dl.fbaipublicfiles.com/detr/detr-r50-e632da11.pth \
       --eval \
       --no_aux_loss \
       --coco_path /content/VOC_coco_format
```

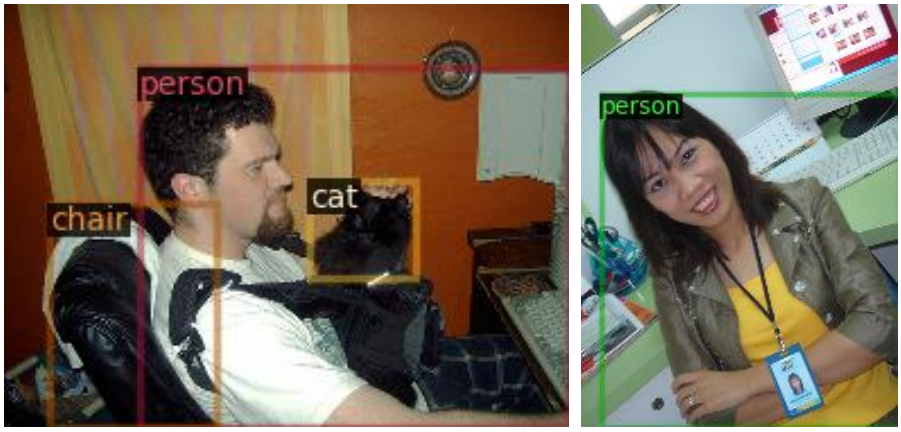
2. Show qualitative and quantitative results of the two models and provide a comparison between the two. Include training curves for the two methods. Also compare the performance of the models before and after fine tuning.

R-CNN

Qualitative:

LR = 0.00025, Iteration = 3000

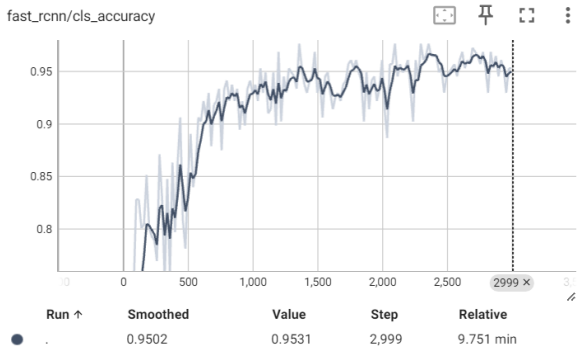
Before Training



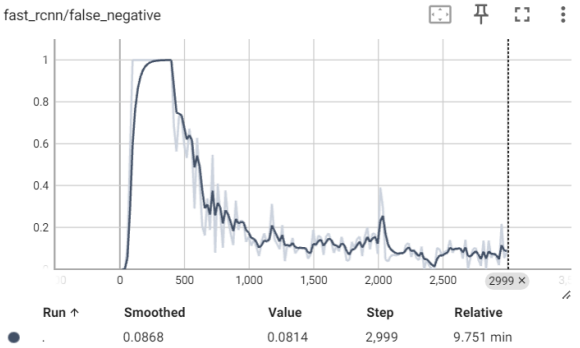
After Training

Quantitative:

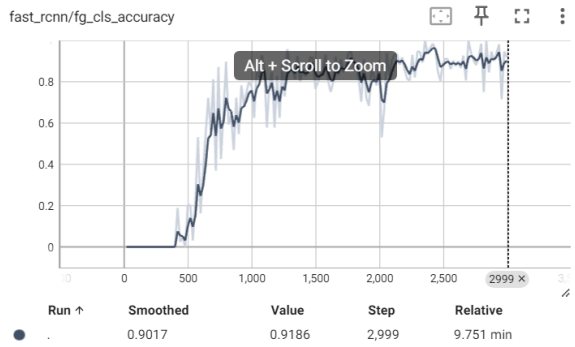
AP: 51.006899745498444, 'AP50': 81.36832538507295, 'AP75': 58.59825322089813



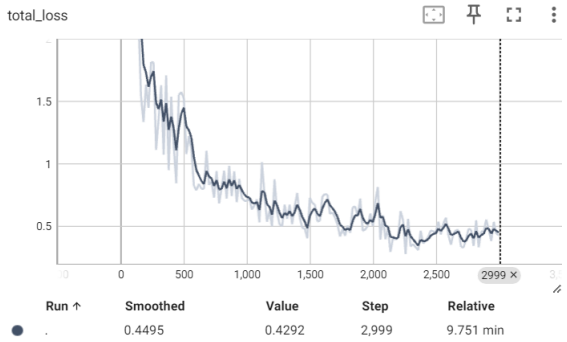
The curve shows a steady increase in classification accuracy over training steps, with some fluctuations.The final accuracy is around 0.95, indicating high performance in object classification.



This curve shows a rapid decrease initially, indicating a significant reduction in missed detections. The curve then plateaus, suggesting that the model has reached a limit in its ability to reduce false negatives.



This curve likely represents the classification accuracy for foreground objects (objects of interest). It shows a similar trend to the overall classification accuracy, but with slightly lower values.



This curve shows the overall training loss, which combines various components like classification loss, bounding box regression loss, and others.The decreasing trend indicates that the model is learning and improving over time.

Performance after fine tuning:

Losses:

- Total Loss: The total loss fluctuates throughout the iterations but generally decreases from 3.92 (iteration 19) to 2.50 (iteration 299), indicating progress in training.
- Loss Components:
    - Box Regression Loss (loss_box_reg): This starts around 0.67 and decreases steadily to 0.76 by the final iteration.
    - Classification Loss (loss_cls): This decreases from 3.20 at iteration 19 to 1.69 at iteration 299, which is a positive sign of improving model performance.
    - RPN Classification Loss (loss_rpn_cls): This fluctuates slightly around 0.01, contributing minimally to the total loss.
    - RPN Localization Loss (loss_rpn_loc): This fluctuates between 0.00 and 0.01, also contributing minimally.

Accuracy Metrics:

- Fast R-CNN Classification Accuracy (cls_accuracy): This metric starts at 0.0% and improves gradually, reaching 79.3% by the final iteration.
- False Negatives: The model starts with no false negatives and sees some false negatives towards the later stages, peaking at 100% for several iterations, then gradually reducing towards the end.
- Foreground Classification Accuracy (fg_cls_accuracy): This metric remains 0.0 throughout the training, suggesting that foreground classification may be underperforming.

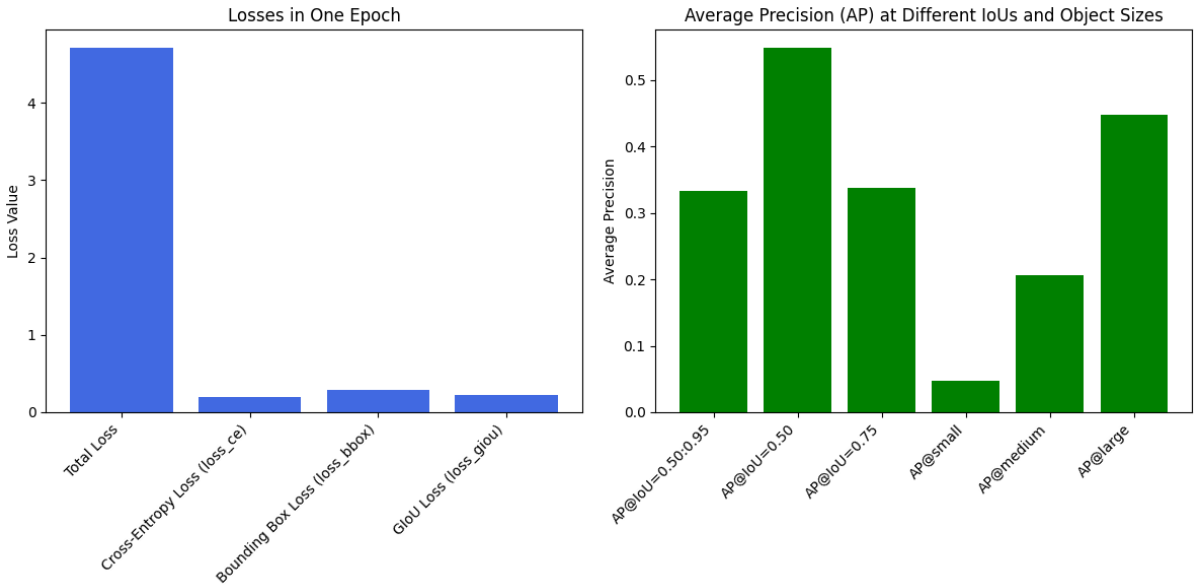
DETR

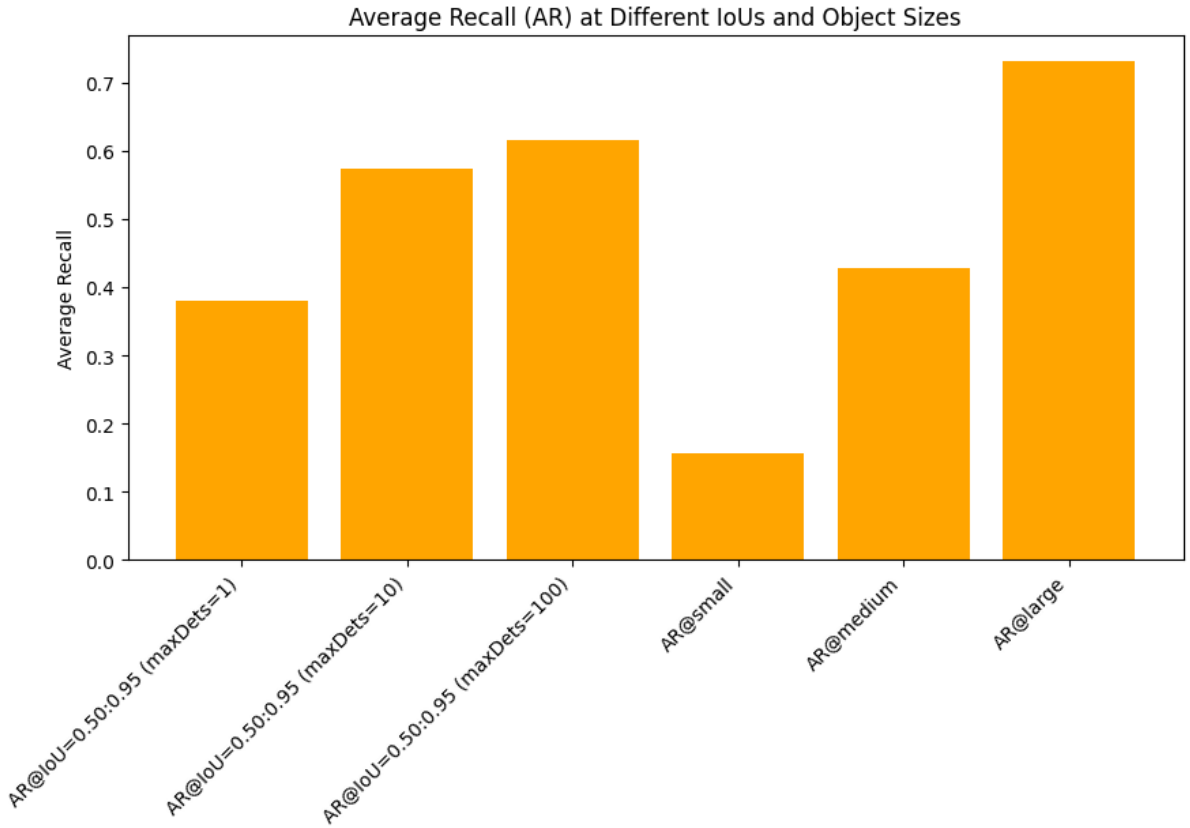
Qualitative:

Quantitative:

Averaged stats: class_error: 50.00 loss: 4.7068 (5.8375) loss_ce: 0.1964 (0.2555) loss_bbox: 0.2882 (0.3391) loss_giou: 0.2245 (0.3637) loss_ce_0: 0.2322 (0.3146) loss_bbox_0: 0.3111 (0.3650) loss_giou_0: 0.2901 (0.3893) loss_ce_1: 0.2380 (0.2931) loss_bbox_1: 0.3035 (0.3314) loss_giou_1: 0.2653 (0.3558) loss_ce_2: 0.2170 (0.2689) loss_bbox_2: 0.2455 (0.3224) loss_giou_2: 0.2173 (0.3478) loss_ce_3: 0.2152 (0.2577) loss_bbox_3: 0.2515 (0.3243) loss_giou_3: 0.2103 (0.3514) loss_ce_4: 0.1899 (0.2570) loss_bbox_4: 0.2435 (0.3379) loss_giou_4: 0.2118 (0.3627) loss_ce_unscaled: 0.1964 (0.2555) class_error_unscaled: 25.0000 (26.4463) loss_bbox_unscaled: 0.0576 (0.0678) loss_giou_unscaled: 0.1123 (0.1818) cardinality_error_unscaled: 1.5000 (2.2824) loss_ce_0_unscaled: 0.2322 (0.3146) loss_bbox_0_unscaled: 0.0622 (0.0730) loss_giou_0_unscaled: 0.1451 (0.1947) cardinality_error_0_unscaled: 3.5000 (2.8254) loss_ce_1_unscaled: 0.2380 (0.2931) loss_bbox_1_unscaled: 0.0607 (0.0663) loss_giou_1_unscaled: 0.1327 (0.1779) cardinality_error_1_unscaled: 1.0000 (2.1909) loss_ce_2_unscaled: 0.2170 (0.2689) loss_bbox_2_unscaled: 0.0491 (0.0645) loss_giou_2_unscaled: 0.1086 (0.1739) cardinality_error_2_unscaled: 1.5000 (2.3620) loss_ce_3_unscaled: 0.2152 (0.2577) loss_bbox_3_unscaled: 0.0503 (0.0649) loss_giou_3_unscaled: 0.1051 (0.1757) cardinality_error_3_unscaled: 2.0000 (2.7428) loss_ce_4_unscaled: 0.1899 (0.2570) loss_bbox_4_unscaled: 0.0487 (0.0676) loss_giou_4_unscaled: 0.1059 (0.1813) cardinality_error_4_unscaled: 1.5000 (2.2013)
IoU metric: bbox
Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.333
Average Precision (AP) @[IoU=0.50 | area= all | maxDets=100] = 0.548
Average Precision (AP) @[IoU=0.75 | area= all | maxDets=100] = 0.337
Average Precision (AP) @[IoU=0.50:0.95 | area= small | maxDets=100] = 0.047
Average Precision (AP) @[IoU=0.50:0.95 | area=medium | maxDets=100] = 0.206
Average Precision (AP) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.447
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 1] = 0.381
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 10] = 0.574
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.616
Average Recall (AR) @[IoU=0.50:0.95 | area= small | maxDets=100] = 0.156
Average Recall (AR) @[IoU=0.50:0.95 | area=medium | maxDets=100] = 0.428
Average Recall (AR) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.732

Training Curves:





Performance before fine tuning:

1. Loss Breakdown

- Total Loss: 0.5375 (average over all iterations), with a standard deviation of 0.6904.
  - This is the combined loss, which includes various components like classification, bounding box prediction, and Generalized IoU (GIoU).
- Loss for Classification (loss_ce): 0.2664 (average), with a standard deviation of 0.2609.
  - This is the loss from misclassifying the objects (the cross-entropy loss).
- Loss for Bounding Boxes (loss_bbox): 0.0945 (average), with a standard deviation of 0.2040.
  - This represents how well the model predicts the bounding box coordinates.
- Loss for Generalized IoU (loss_giou): 0.1353 (average), with a standard deviation of 0.2255.

    - This metric evaluates how well the predicted boxes overlap with the ground truth boxes in terms of IoU, with better overlap resulting in a smaller loss.

Unscaled Losses:

- Unscaled losses are the individual components of the loss function (before any scaling) and the errors associated with bounding boxes or classification. For example, the unscaled loss_ce_unscaled represents the classification error before any adjustments are made.
- Cardinality Error Unscaled: 4.0000 (average), with a standard deviation of 6.4572.
    - This is the error due to the model predicting a different number of objects than the ground truth (cardinality refers to the number of objects detected).

2. Performance Metrics (AP and AR)

- Average Precision (AP): This is the primary metric used to evaluate object detection models, showing how well the model can classify and localize objects. AP is calculated at different Intersection-over-Union (IoU) thresholds.
    - IoU = 0.50:0.95 (all sizes): 0.539
    - IoU = 0.50: 0.705 (better performance at the lower IoU threshold).
    - IoU = 0.75: 0.582 (the model performs well at this higher IoU threshold as well).
    - IoU = 0.50:0.95 (small objects): 0.125 (lower performance on small objects).
    - IoU = 0.50:0.95 (medium objects): 0.376 (moderate performance on medium-sized objects).
    - IoU = 0.50:0.95 (large objects): 0.664 (good performance on large objects).

The AP values at IoU = 0.50 (often used for evaluation) are much higher than those for higher IoU thresholds, indicating that the model tends to perform better at identifying objects in general (i.e., it can detect more objects roughly) but struggles with more precise localization (IoU of 0.75).

3. Average Recall (AR)

- AR at IoU = 0.50:0.95:
    - maxDets=1: 0.501 (the recall is roughly 50% when only one object is allowed to be detected per image).
    - maxDets=10: 0.740 (this improves significantly when up to 10 objects can be detected per image).
    - maxDets=100: 0.774 (the recall increases further when up to 100 objects can be detected).
- AR for different object sizes:
    - Small: 0.354 (low recall for small objects).
    - Medium: 0.633 (decent recall for medium objects).
    - Large: 0.870 (excellent recall for large objects).

The model performs well at recall when detecting multiple objects (maxDets=10 and maxDets=100), and is especially strong with large objects. It struggles with small objects, which is a common issue for many object detection models.

Performance after fine tuning:

1. Classification and Localization Losses

- Class Error: The overall classification error is 50.00%, which is relatively high. This could be due to a poor model fit, insufficient training, or difficulty in distinguishing between classes.
- Loss Components:
    - Total Loss: 4.7068 (5.8375), which seems to have a reduction from the previous iteration (5.8375), indicating some progress, but still relatively high.
    - Cross-Entropy Loss (loss_ce): 0.1964 (0.2555), a reasonable reduction. Cross-entropy loss quantifies the classification accuracy and the reduction suggests improved classification.
    - Bounding Box Loss (loss_bbox): 0.2882 (0.3391), indicating some progress in localization accuracy.
    - Generalized Intersection over Union (loss_giou): 0.2245 (0.3637), showing a moderate improvement in bounding box overlap.
    - The unscaled values of the losses show smaller variations, but the trend of improvement in the scaled losses indicates some progress.

2. Individual Class Losses

The loss is broken down across multiple classes (from 0 to 4). For instance:

- loss_ce_0 to loss_ce_4: The cross-entropy losses across classes seem to be decreasing, with loss_ce_4 showing the smallest value (0.1899), implying that the model is having better performance on the last class.

- loss_bbox and loss_giou for each class indicate similar trends where the losses are reducing slightly, reflecting some improvements in localization and IoU metrics.

3. Evaluation Metrics (AP and AR):

The average precision (AP) and average recall (AR) metrics provide insights into how well the model is performing on object detection tasks:

- AP@IoU=0.50:0.95 (overall): 0.333 — This is a reasonable value, showing that the model performs moderately well across different IoU thresholds (from 0.5 to 0.95).
- AP@IoU=0.50: 0.548 — This is a good result, suggesting that the model can detect objects well when using a lower IoU threshold.
- AP@IoU=0.75: 0.337 — This drop indicates a significant decline in performance when the IoU threshold is increased, which suggests that the model may struggle with more precise localization.
- AP for Small, Medium, and Large objects:
    - Small objects: 0.047, which is very low. This suggests that the model struggles to detect small objects reliably.
    - Medium objects: 0.206, indicating moderate success in detecting medium-sized objects.
    - Large objects: 0.447, which is good, suggesting that the model is more effective at detecting larger objects.
- AR Metrics:
    - AR@IoU=0.50:0.95 (overall): 0.616 — A reasonable recall score, showing that the model identifies a good portion of the objects.
    - AR for Small, Medium, and Large objects: The recall for small objects (0.156) is notably low, while it performs much better for medium (0.428) and large (0.732) objects. This further emphasizes the model's difficulty in detecting small objects.

4. Error Analysis

- The model's class error (50%) and cardinality error (around 1.5 to 3) suggest that while the model is identifying some objects, it is likely missing or misclassifying others. The cardinality error indicates how well the model is predicting the correct number of objects in the scene.
- The cardinality error is particularly high in some cases (for example, class 0 has a cardinality error of 3.5), which suggests that the model might be over-predicting or under-predicting the number of instances for certain classes.

Compare the performance of the models before and after fine tuning:

1. Loss Breakdown

The total loss has substantially increased in the after scenario, which may indicate some instability or underperformance after a change in the model's configuration, training process, or data handling.

2. Averaged Precision (AP)

After modifications, the model's performance has dropped across all AP metrics, especially for small and large objects. The drop in performance might be due to overfitting, improper training, or a data mismatch.

3. Average Recall (AR)

The recall has decreased after the changes, particularly for small and medium objects. The ability to detect large objects has also decreased, which might suggest the model is generalizing poorly or suffering from some form of underfitting or improper training.

4.Class Error and Cardinality Error

The class error has doubled, indicating that the model is making more mistakes in terms of classification after the changes.The cardinality error has improved, suggesting the model may have a better understanding of how many objects should be in the image, though this doesn't compensate for the increase in class error.

Comparison between two models:

Before:

After R-CNN:

After DETR:

Overall Performance

Both R-CNN and DETR seem to perform reasonably well in object detection, accurately identifying cars and people in the image. However, DETR appears to be more confident in its predictions, as indicated by the higher confidence scores.

Specific Differences

1. Object Detection:
    - R-CNN identifies a larger number of objects, including multiple cars.
    - DETR, on the other hand, identifies fewer objects but with higher confidence scores. This suggests that DETR might be more selective in its predictions, focusing on objects it is highly certain about.
2. Object Classification:
    - Both models correctly classify the objects as cars and people.
    - DETR assigns higher confidence scores to its classifications, indicating greater certainty in its predictions.

3. Bounding Box Accuracy:
    - R-CNN's bounding boxes seem to be more accurate, closely fitting the contours of the objects.
    - DETR's bounding boxes are slightly less precise, especially for the car in the top left corner.

3. As the models may be slow to train, number of training epochs may be limited; however, you are still encouraged to try some variations if possible.

DETR:

lr 0.0003 \lr_backbone 0.00003 \epochs 2\

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.399
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.603
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.423
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.049
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.222
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.513
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.404
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.604
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.665
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.151
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.482
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.788
```

1. Average Precision (AP):
    - AP@IoU=0.50:0.95 (mAP): This is the mean Average Precision computed over different Intersection over Union (IoU) thresholds ranging from 0.50 to 0.95. It gives a general measure of model performance across different levels of precision.
        - AP = 0.399: This indicates moderate performance. A typical goal for object detection models is to have an mAP around 0.40 to 0.50 for standard datasets like COCO. This result suggests the model has decent performance but could benefit from further fine-tuning or improvements.
2. AP at IoU=0.50:
    - AP@IoU=0.50: This metric evaluates precision when the IoU threshold is set at 0.50, meaning a predicted bounding box is considered correct if it overlaps with the ground truth by at least 50%.
        - AP = 0.603: This is a good value. Higher precision at a lower IoU threshold typically indicates that the model is better at detecting objects when the overlap requirement is less strict. An AP over 0.60 at this threshold is a strong result.
3. AP at IoU=0.75:
    - AP@IoU=0.75: This measures the precision when the IoU threshold is higher (0.75), making the evaluation stricter.
        - AP = 0.423: This shows the model is also performing well at higher precision levels, which suggests it's not just detecting objects but doing so with a higher degree of overlap with the ground truth.
4. AP for Different Object Sizes:
    - Small Objects: AP = 0.049 at IoU=0.50:0.95
    - Medium Objects: AP = 0.222 at IoU=0.50:0.95
    - Large Objects: AP = 0.513 at IoU=0.50:0.95
    - These values indicate that the model is performing much better at detecting larger objects than small ones, which is typical for many object detection models. Small object detection is usually more challenging due to the resolution of features at smaller scales.
        - Small Objects: Low performance, as expected (AP=0.049). This suggests the model struggles with small objects in the dataset. Improving this could require adjustments to the model architecture, data augmentation, or more fine-tuning.
        - Medium Objects: Slightly better (AP=0.222), but still not as strong as the performance on large objects.
        - Large Objects: The model performs much better on large objects (AP=0.513), which is a typical behavior, as larger objects cover more pixels and tend to have clearer features.
5. Average Recall (AR):

- AR@IoU=0.50:0.95 (AR): This metric measures how many of the true objects are detected by the model (recall). It's computed for a range of IoU thresholds.
    - AR = 0.665: This means the model is able to recall approximately 66.5% of the ground truth objects at the IoU thresholds between 0.50 and 0.95. This is a relatively good result.

6. AR at IoU=0.50:
    - AR@IoU=0.50: Measures recall when the IoU threshold is 0.50, indicating how many true objects are detected with at least 50% overlap.
        - AR = 0.404: This is lower than the mAP, indicating that while the model is detecting a decent number of objects, the precision (AP) at this threshold is better than recall.
7. AR at IoU=0.75:
    - AR@IoU=0.75: Measures recall when the IoU threshold is stricter (0.75).
        - AR = 0.423: This shows a drop in recall at the higher IoU threshold, which is expected as more strict criteria lead to fewer detections.
8. AR for Different Object Sizes:
    - Small Objects: AR = 0.151 at IoU=0.50:0.95
    - Medium Objects: AR = 0.482 at IoU=0.50:0.95
    - Large Objects: AR = 0.788 at IoU=0.50:0.95
    - These results show that the model recalls many more large objects (AR=0.788) compared to small objects (AR=0.151), which is expected. Improving recall for small objects would likely require addressing the challenges that small object detection presents.