Computer Vision Researcher: "Vision Lab October 2024 CV Researcher Assignment Submission".

Submitted By - Hemant Dadhich (Fresh Grad. in Artificial Intelligence and Dada Science)

Objective :

To explore and experiment with DINO ("DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection"). The goal is to evaluate its pretrained models and subsequently fine-tune them effectively.

Itinerary :

- Dataset Prepration
- Repository Setup & fetching pretrained weights
- Evaluation of the Pretrained model
- Report and Analysis
- Fine-tuning
- Testing of Fine Tuning model
- Attention Map Visualization

❖ Dataset Prepration

- The dataset, provided with annotations in COCO format, was well-structured and error-free, making it suitable for training without any incorrect annotations. This ensured efficient fine-tuning.
- The dataset consisted of 201 images, with 161 images assigned for training and the remaining 40 for validation. The respective annotations (in `.json` format) were also split accordingly.
- Below is the directory structure used for the project:

```
COCODIR/
  └── annotations/
     ├── instances_train2017.json
     ├── instances_val2017.json
  ├── train2017/
  └── val2017/
```

- As part of the process, I visualized the annotations on the dataset images to ensure clarity and correctness. Reference images are provided below:

❖ Repository Setup & fetching pretrained weights

- I cloned the repository from https://github.com/IDEA-Research/DINO.git and downloaded the pretrained DINO 4-Scale weights with the ResNet50 backbone. The weights used were from the 36-epoch and 11-epoch models.

❖ Evaluation of the Pretrained model

- The evaluation was conducted by following the instructions provided in the official repository:

  ```
  !bash scripts/DINO_eval.sh
  /content/drive/MyDrive/DINO/COCODIR
  /content/drive/MyDrive/DINO/checkpoint0033_4scale.pth
  ```

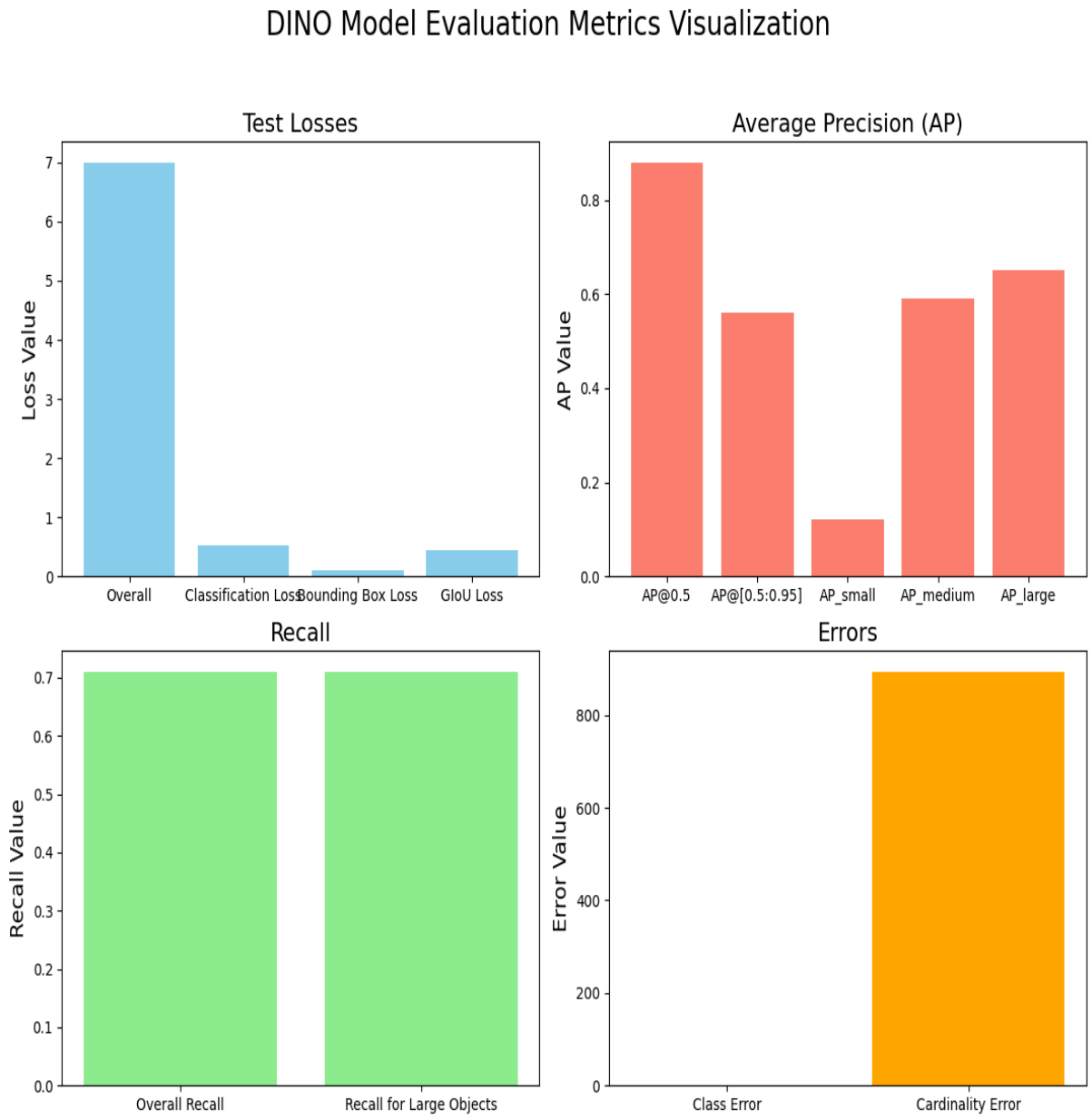
- The pretrained model was trained on 91 (80) classes of the COCO dataset with a 36-epoch setting. According to the

official repository, DINO 4-Scale achieves 50.9 AP for the 36-epoch model and 49.0 AP for the 12-epoch model.

- The results (AP, AR, and losses) obtained after running the evaluation script on the validation set aligned well with the official results mentioned in the repository.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.560
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.883
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.677
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.475
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.641
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.577
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.121
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.597
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.649
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.594
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.709
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.614
```

❖ Report and Analysis



Detailed Analysis on the Evaluation Results of Pretrained DINO 4-Scale(36epoch) Mode:

- Test Losses: Shows the overall loss and specific losses (classification, bounding box, GIoU).
- Average Precision (AP): Highlights AP values at different IoU thresholds, with a focus on small, medium, and large object performance.
- Recall: Provides an overview of overall recall and recall for large objects.
- Errors: Depicts the class error and the high cardinality error.

LOSS :

- Overall Loss (7.00) : Aggregates the classification, bounding box, and GIoU losses.
- Classification loss (Cross-entropy - 0.517): Fairly consistent classification
- Bounding Box Loss(0.100): Relatively Low

- GIoU Loss: 0.446 : {Reference : "Intersection of Union (IoU) loss measures the overlap between predicted 3D box and ground truth 3D bounding boxes."} : Can be Improved.

AVERAGE PRECISION (AP):

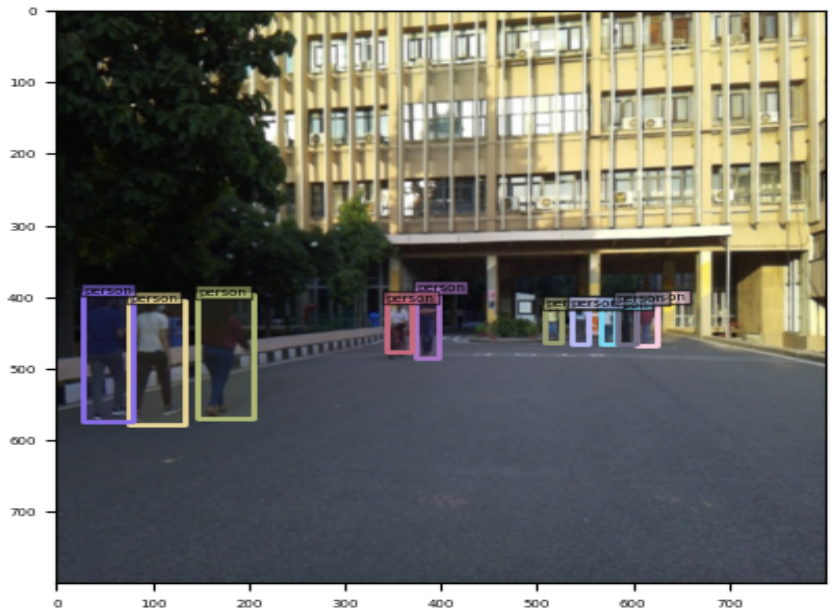
- AP@0.5 (higher IoU threshold) : 0.88 : High Precision when strict localization is required.
- AP@[0.5:0.95] (standard COCO metric): 0.56 : Decent
- AP_small: 0.12, AP_medium: 0.59, AP_large: 0.65 : The model performs better with medium to large objects but struggles with smaller objects

RECALL :

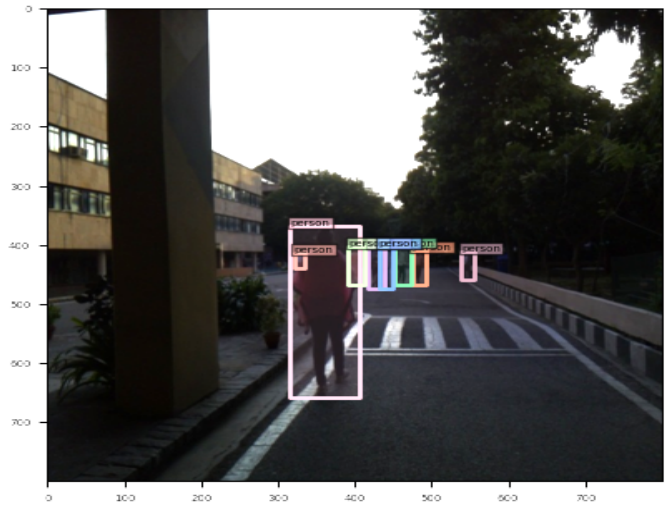
- Overall Recall: 0.71 : Moderately High.
- CLASS ERROR : Classification Error: 1.19%
- CARDINALITY ERROR: (893.55) : suggesting issues with the number of predicted objects versus actual objects in an image, This is mainly due to the large no. of classes on which the pretrained model is trained.

Visualizing Detection Results on validation dataset

- There are in total 40 images in the validation set.
- As mentioned earlier got some good results without even training the model, since it was pretrained on 36 epoch setting on coco data.
- **Happy Cases:**



- **Failure Cases:** Some of the failure cases involved, not able to detect distant objects, overlapping and multiple bounding boxes for a single person and had below average results on small objects.



❖ Fine-tuning

- For fine-tuning the model on the pretrained weights of (DENO 4-Scale model), firstly the config file of the architecture is modified :
  config.py :
- Since the no. of class is only 1 set num_classes in the config.py. The model architecture is defined to have 1 increment of the number of classes so mentioned 2 for 1 class.

```
num_classes=2
```

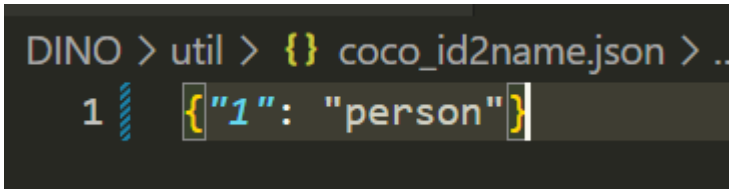
It is necessary to pass these parameters in the finetuning command when modifying num_classes:

```
  --pretrain_model_path
/content/drive/MyDrive/DINO/checkpoint0033_4scale.pth \
 -- finetune_ignore label_enc.weight class_embed \
 -- options num_classes=2
```

- **dn_labelbook_size:**
  It should be : dn_labelbook_size >= num_classes + 1

- **Epochs:**
  Epochs = 12 (also trained for 20 epochs)

- **Coco_id2name.json:** since training only on one class



- Saved Checkpoints: It can be found in Logs directory:
  Logs/dino/R50-MS4/checkpoint_best_regular.pths
  The training/finetuning logs are also present at there

❖ Attention Maps

❖ For visualisation of attention maps we need cross attention weights from the decoder layer. We can also visualize using self attention weights but they will give the generalised view of the image, while cross attention focus on the detection.

- **`MSDeformAttn class:`** To Get attention maps we need to first modify the attention weights from the forward() method of the `MSDeformAttn class`

```
class MSDeformAttn(nn.Module):
    def forward(self, query, reference_points, input_flatten, inpu
            output = output.to(torch.float16)
            output = self.output_proj(output)
            return output, attention_weights


        output = MSDeformAttnFunction.apply(
            value, input_spatial_shapes, input_level_start_index,
        output = self.output_proj(output)
        return output, attention_weights
```

- **DeformableTransformerDecoderLayer class**

```
class DeformableTransformerDecoderLayer(nn.Module):
    def forward(self,
            else:
                raise ValueError('unknown funcname {}'.format(funcname))

        return tgt, cross_attention_weights
        # return tgt, cross_attention_weights, self_attention_weights
```

Return the cross attention weights from here

- There are more changes in some other classes as well : DeformableTransformer, TransformerDecoder etc.

- Attention Map Results: