HW-6

USC ID: **4382673678**

Goal: fine-tuning an object detection model.

1. **Classification**: For each of the twenty classes, predicting presence/absence of an example of that class in the test image.
2. **Detection**: Predicting the bounding box and label of each object from the twenty target classes in the test image.

Model used:
"COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"
"COCO-Detection/retinanet_R_50_FPN_3x.yaml".

Library used: https://github.com/facebookresearch/**detectron2**, Pyyaml, cv2, TensorBoard

Dataset: Pascal VOC dataset for object detection

http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar

The twenty object classes are:

- *Person:* person
- *Animal:* bird, cat, cow, dog, horse, sheep
- *Vehicle:* aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor:* bottle, chair, dining table, potted plant, sofa, tv/monitor

```
[11/23 19:33:04 d2.data.build]: Removed 0 images with no usable annotations. 2501 images left.
[11/23 19:33:04 d2.data.build]: Distribution of instances among all 20 categories:
```

category	#instances	category	#instances	category	#instances
aeroplane	156	bicycle	202	bird	294
boat	208	bottle	338	bus	131
car	826	cat	191	chair	726
cow	185	diningtable	148	dog	271
horse	207	motorbike	193	person	2705
pottedplant	305	sheep	191	sofa	218
train	158	tvmonitor	191		
total	7844				

Qualitative and Quantitative results:

1. **Faster RCNN seems to be working better as compared to RetinaNet.**

   Evaluation on 2 different models:
   Faster RCNN 'AP50': 71.70258207440183
   RetinaNet ‘AP50’: 48.231234566

2. **Config used for training are:**

Faster RCNN:

```
cfg.DATALOADER.NUM_WORKERS = 1
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml") # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 1
```

```
cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR
cfg.SOLVER.MAX_ITER = 3000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 20
```

RetinaNet config:

```
cfg.DATALOADER.NUM_WORKERS = 1
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/retinanet_R_50_FPN_3x.yaml") # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 1
cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR
cfg.SOLVER.MAX_ITER = 3000

cfg.MODEL.RETINANET.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.RETINANET.NUM_CLASSES = 20

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

```
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/retinanet_R_50_FPN_3x.yaml") # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 1
cfg.SOLVER.BASE_LR = 0.00005
cfg.SOLVER.MAX_ITER = 2500

cfg.MODEL.RETINANET.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.RETINANET.NUM_CLASSES = 20

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

3. Training is done on Google CoLab.
4. Fine-tuning is done on the pre-trained model in both the cases.
5. Total train time taken for both the models is near to 30 mins for 3000 epochs and the inference time is close to 14 mins.

This data doesn't show one is faster than another as the difference is very less. Also, can't rely on CoLab because of its own load and provisioning strategy. Also, the number of times the experiment is done was 2-3 so it is not good to compare the time.

6. Total loss for RetinaNet is going to 0.1757. RetinaNet uses focal loss to reduce loss for harder examples but this is not evident with the result.

Faster RCNN:
<start>

```
iter: 19  total_loss: 3.899  loss_cls: 2.954  loss_box_reg: 0.9129
loss_rpn_cls: 0.006985  loss_rpn_loc: 0.01315
```

<end>
iter: 2999 total_loss: 0.6291 loss_cls: 0.1664 loss_box_reg: 0.3603
loss_rpn_cls: 0.001284 loss_rpn_loc: 0.01243

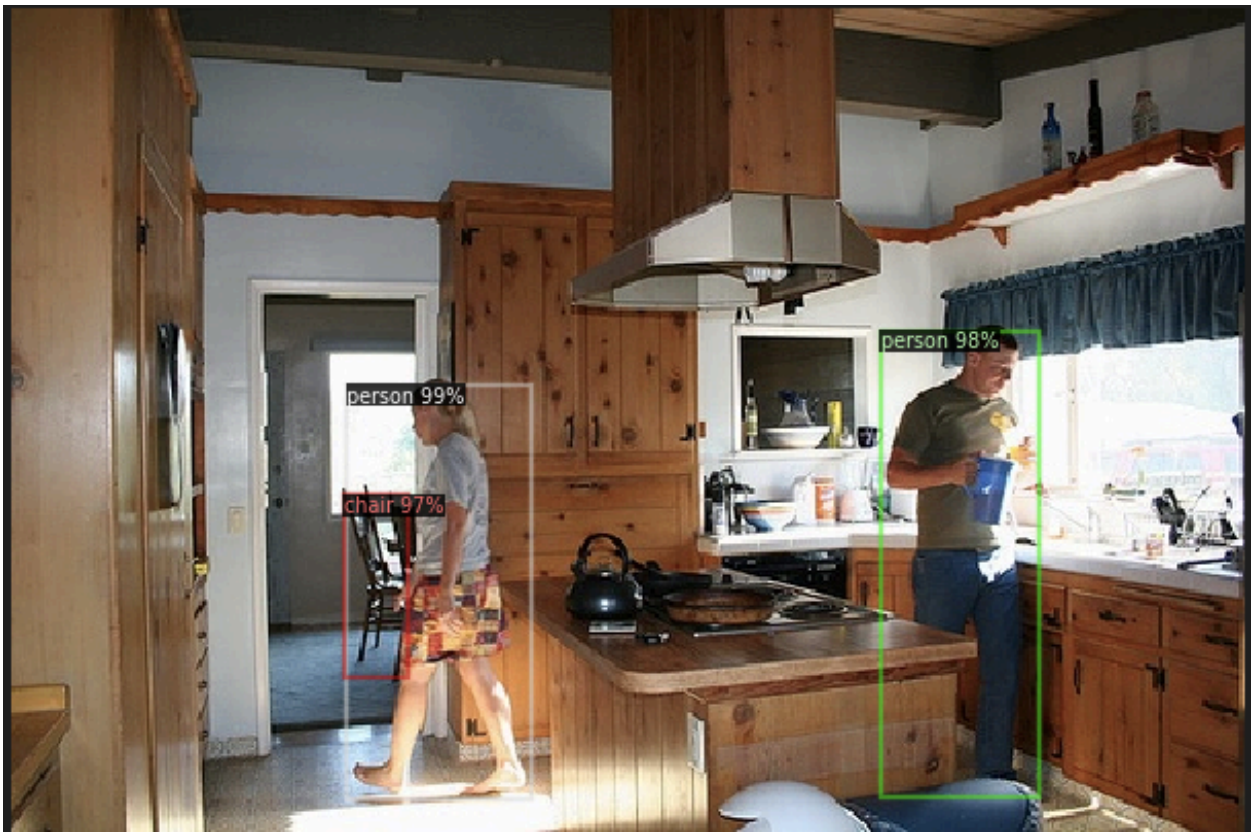
RetinaNet:
<start>

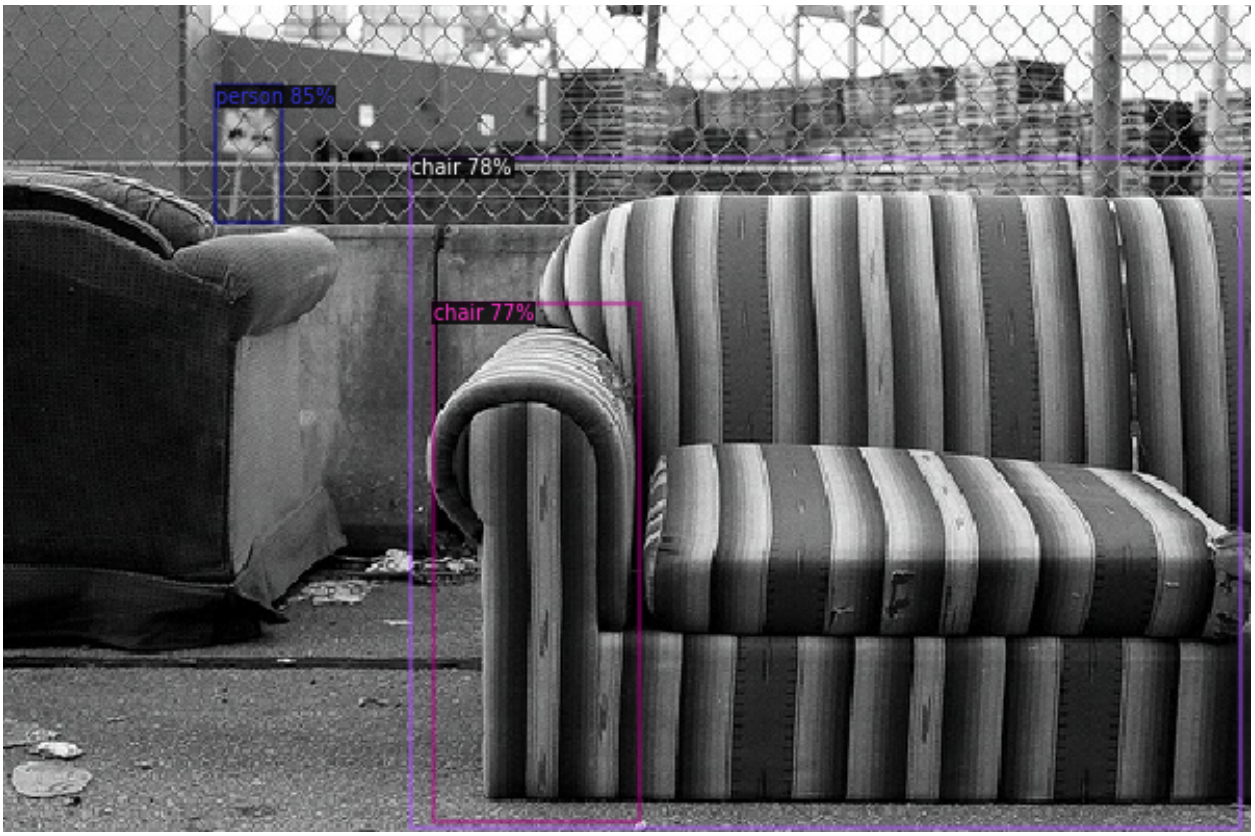
```
total_loss: 0.9713  loss_cls: 0.7708  loss_box_reg: 0.1481
```

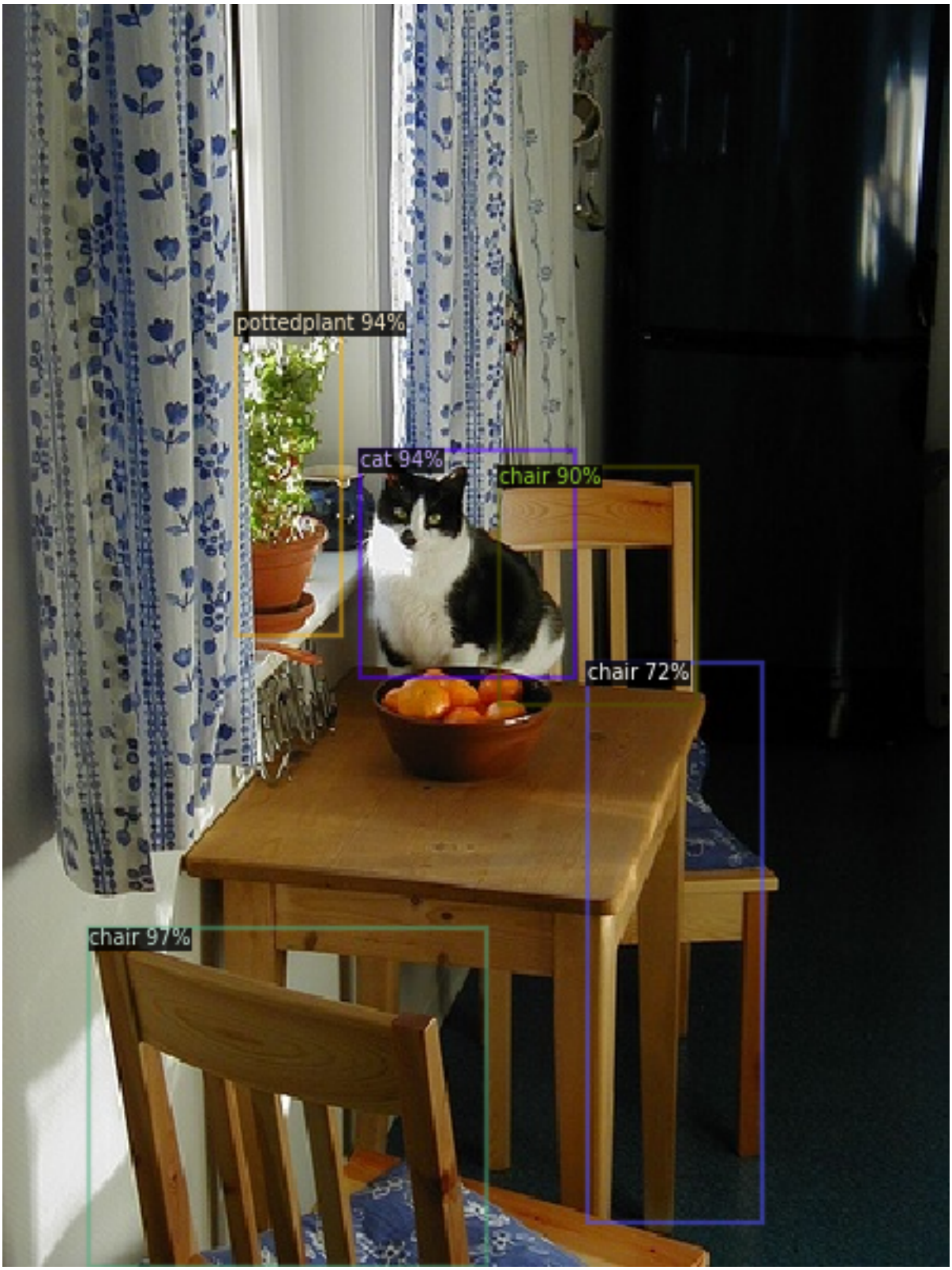
<end>
total_loss: 0.1757 loss_cls: 0.09117 loss_box_reg: 0.06593

7. Some results from Faster RCNN

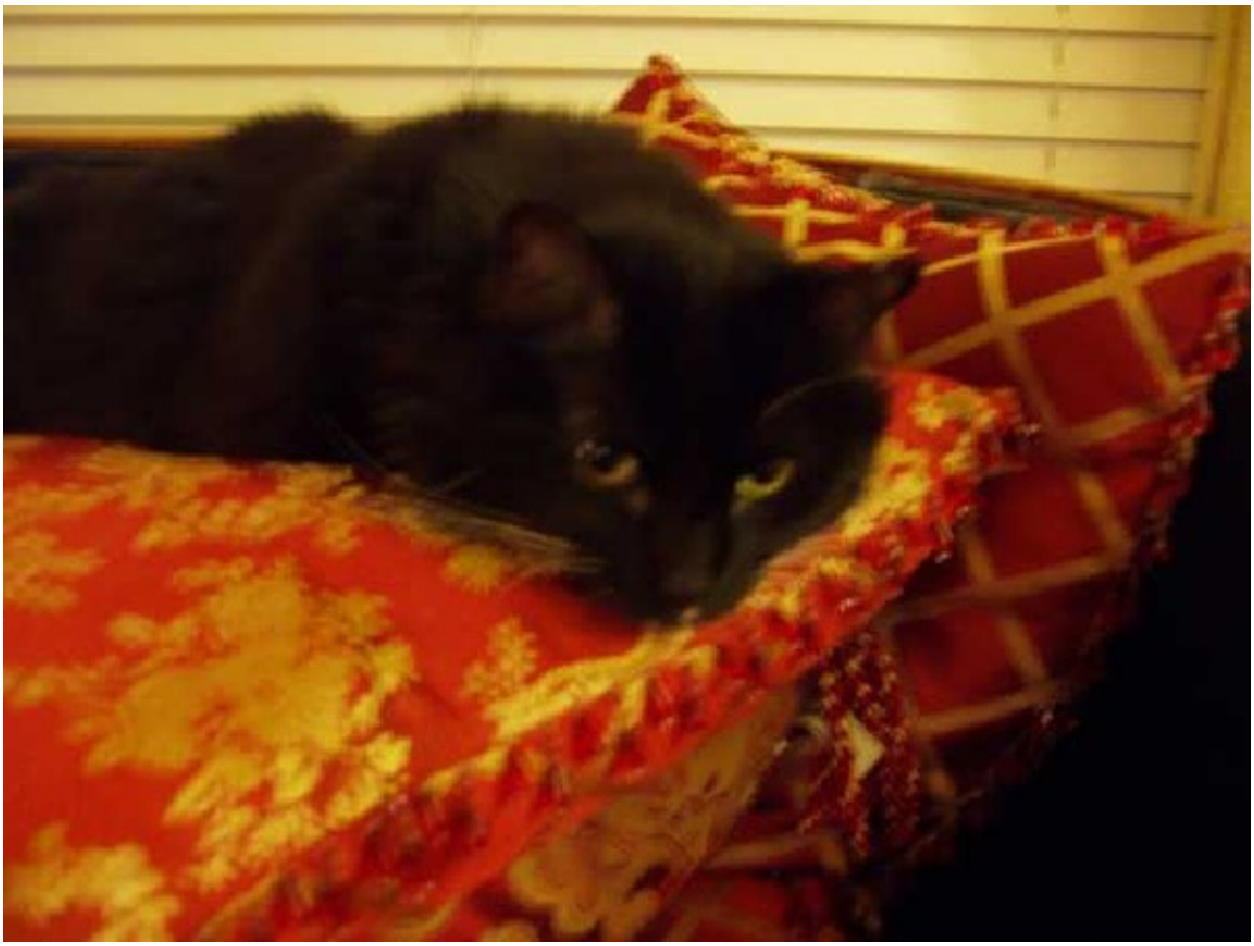

car 94%


person 99%
chair 97%
person 98%


person 85%
chair 78%
chair 77%


pottedplant 94%
cat 94%
chair 90%
chair 72%
chair 97%

Bad example: where it is not detecting the objects.



8. Some result from RetinaNet


bicycle 74%
PEUGEOT


bus 90%
4 Acomb
TAXIS
person 86%
B7 FTR


train 85%

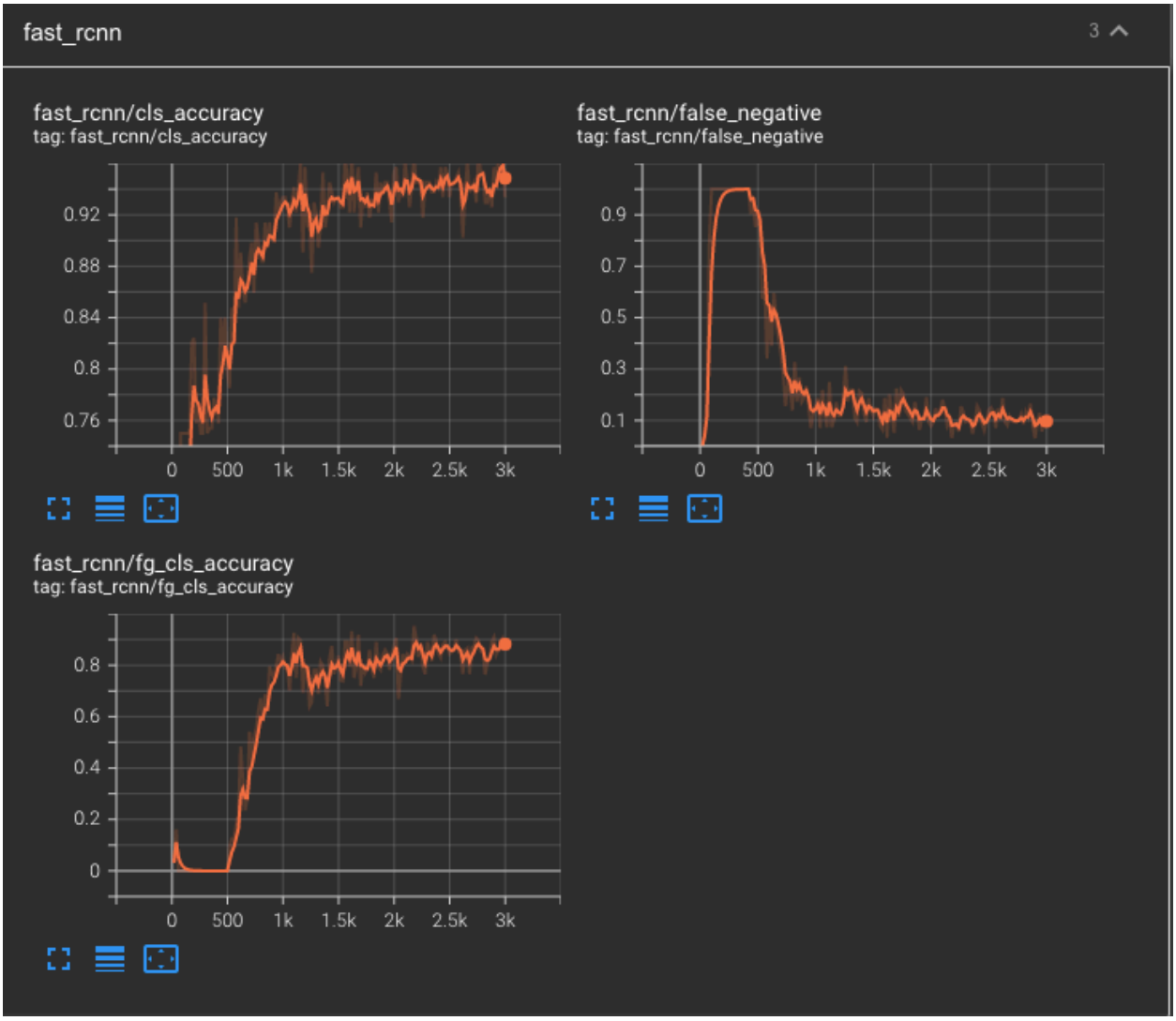

person 86%
car 73%
person 84%

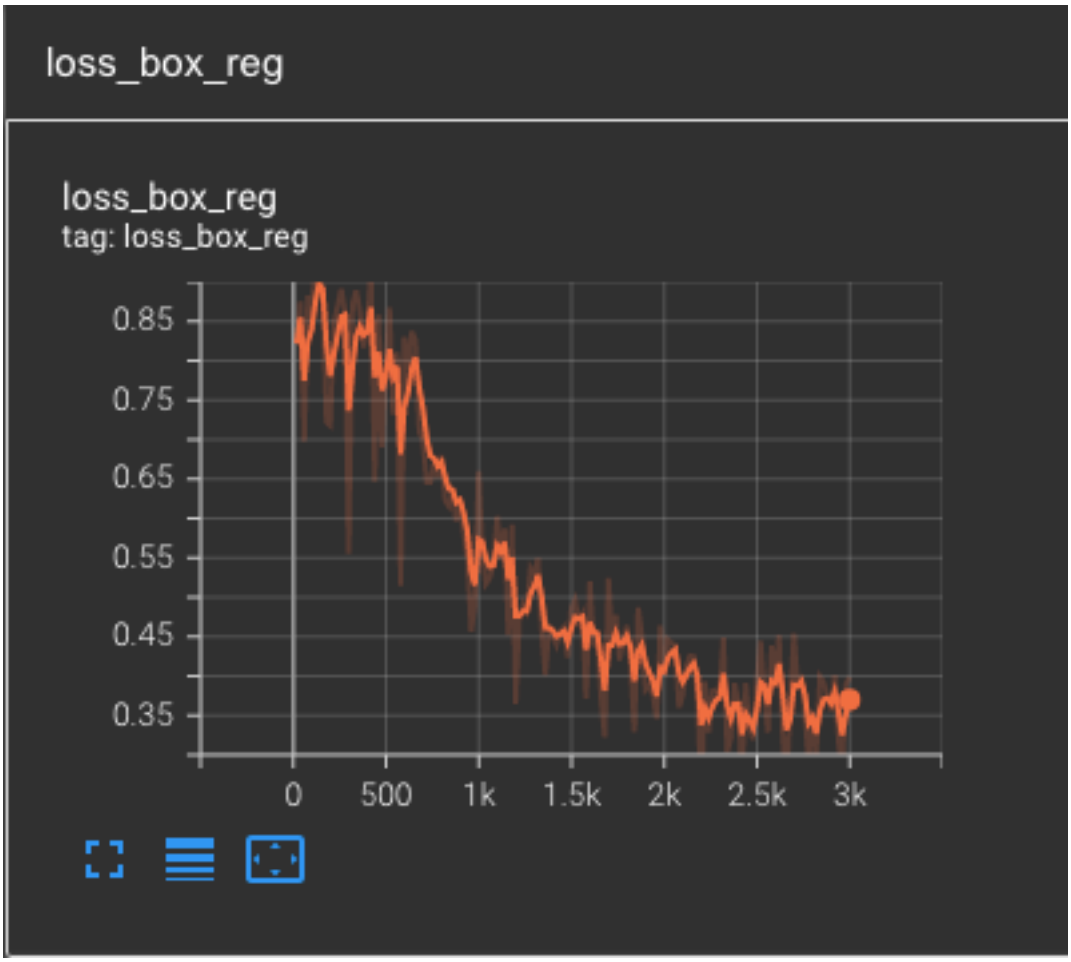
Fig: Faster RCNN is able to detect the persons as well as buses in the background. Also, the confidence seems to be very high which is impressive.

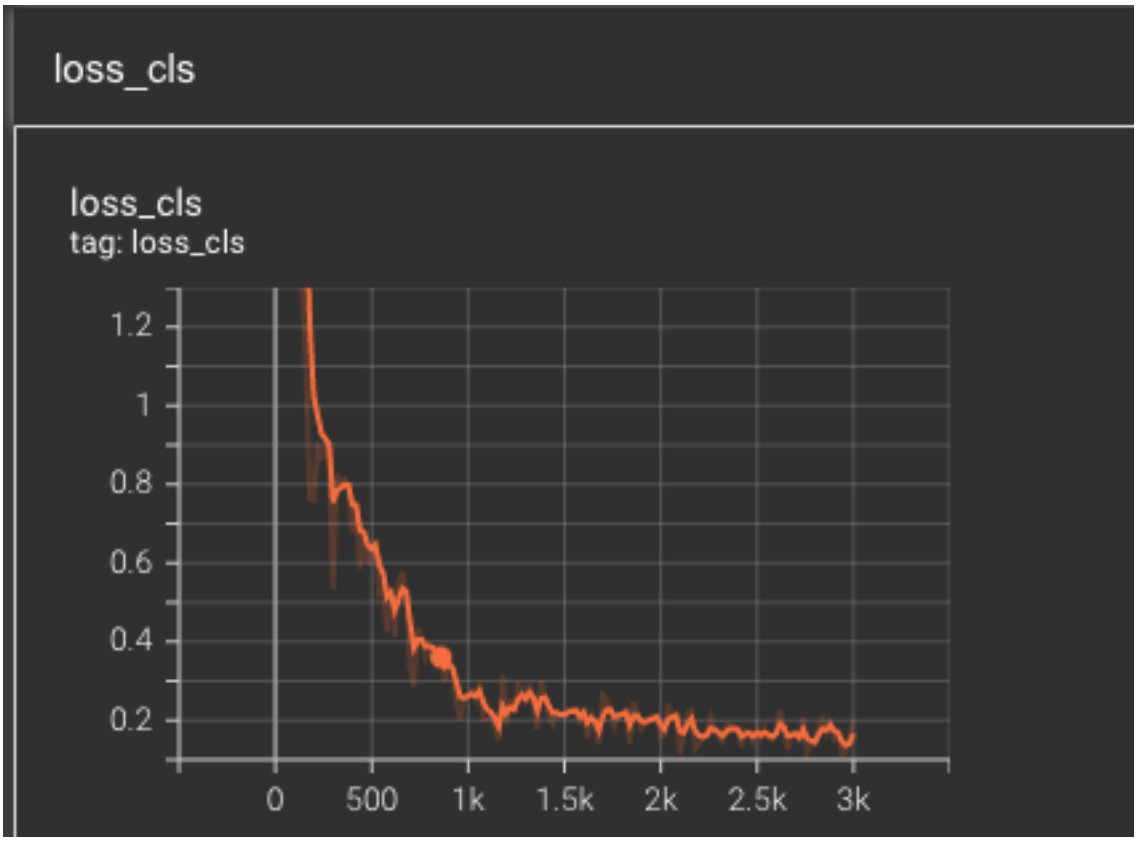

person 99%
bicycle 89%
bicycle 89%
person 95%
person 81%

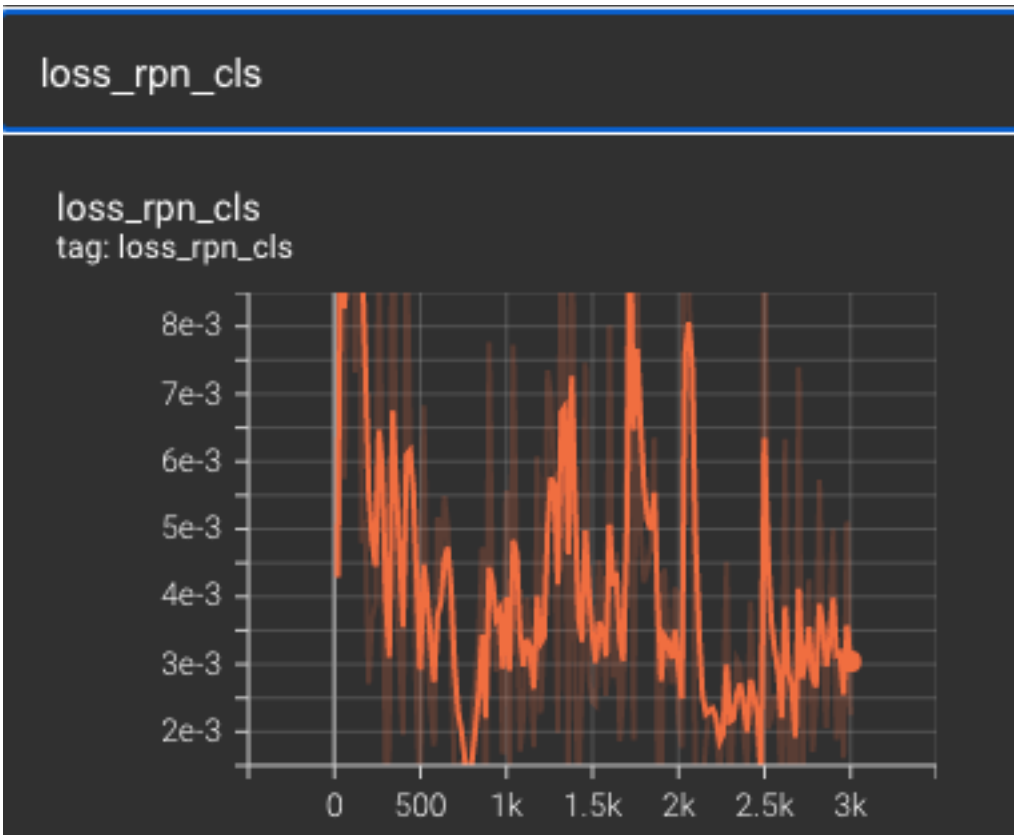
Let’s look into the graphs in terms of losses and accuracy.

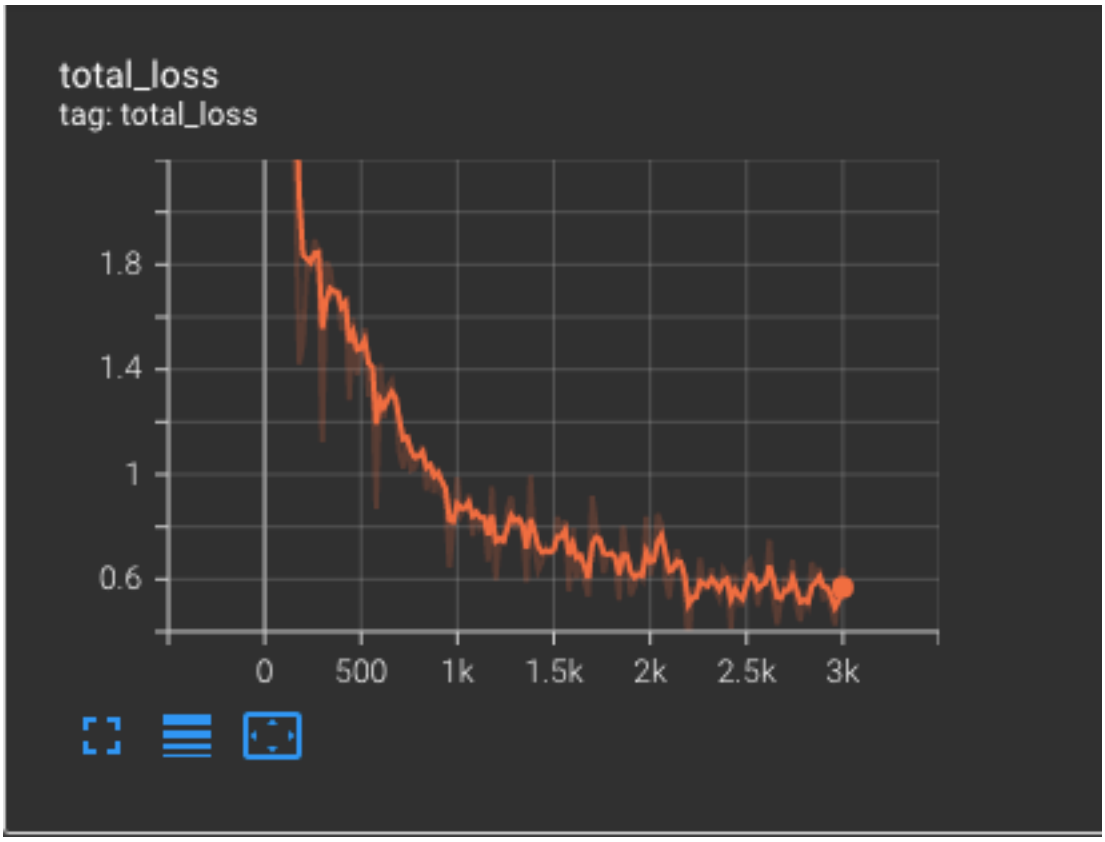
1.. Faster RCNN related graphs …



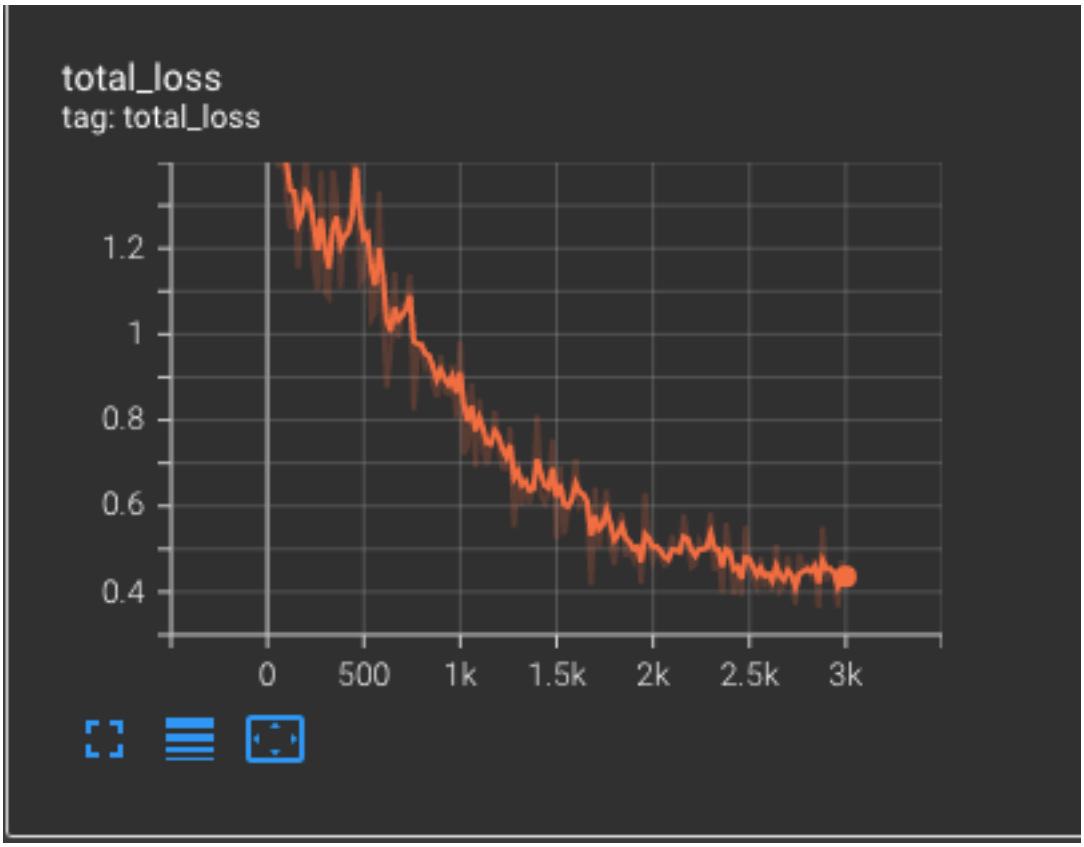

loss_box_reg
loss_box_reg
tag: loss_box_reg
0.85
0.75
0.65
0.55
0.45
0.35
0
500
1k
1.5k
2k
2.5k
3k


loss_cls
loss_cls
tag: loss_cls
1.2
1
0.8
0.6
0.4
0.2
0
500
1k
1.5k
2k
2.5k
3k

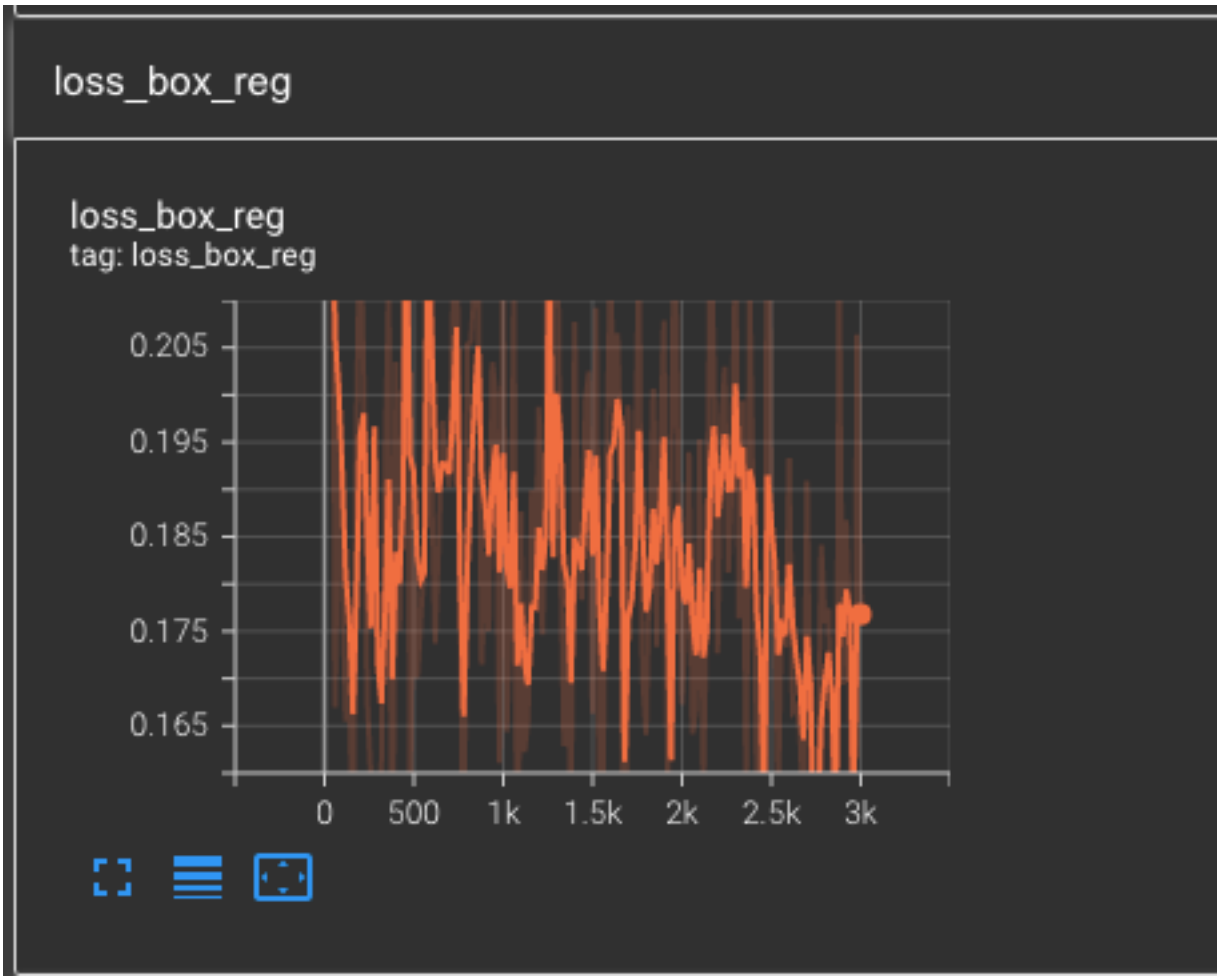

loss_rpn_cls
loss_rpn_cls
tag: loss_rpn_cls
8e-3
7e-3
6e-3
5e-3
4e-3
3e-3
2e-3
0
500
1k
1.5k
2k
2.5k
3k

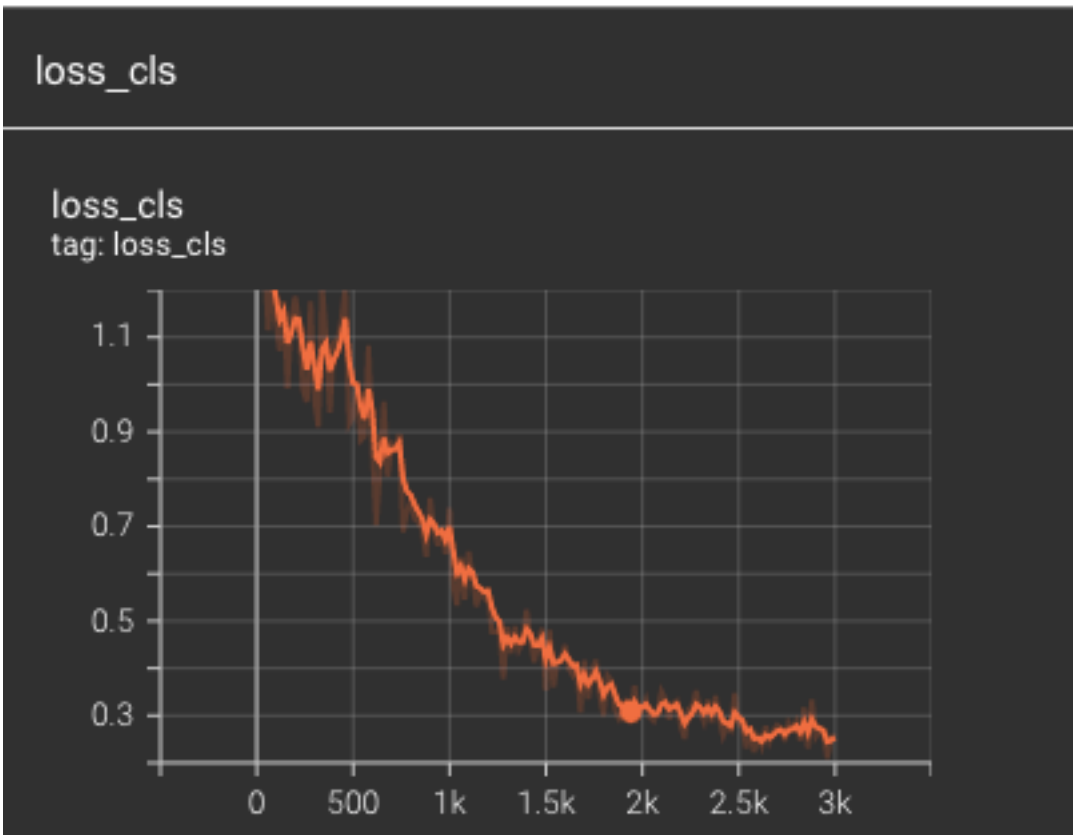

total_loss
tag: total_loss
1.8
1.4
1
0.6
0
500
1k
1.5k
2k
2.5k
3k

2.. RetinaNet related graphs …




num_pos_anchors
num_pos_anchors
tag: num_pos_anchors
82
78
74
70
0
500
1k
1.5k
2k
2.5k
3k


loss_box_reg
loss_box_reg
tag: loss_box_reg
0.205
0.195
0.185
0.175
0.165
0
500
1k
1.5k
2k
2.5k
3k


loss_cls
loss_cls
tag: loss_cls
1.1
0.9
0.7
0.5
0.3
0
500
1k
1.5k
2k
2.5k
3k

Faster RCNN Model

```
[11/23 19:33:03 d2.engine.defaults]: Model:
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (top_block): LastLevelMaxPool()
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
          3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
      )
      (res2): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv1): Conv2d(
            64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
```

```
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
      )
      (res3): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv1): Conv2d(
            256, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
```

```
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
        (3): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
      )
      (res4): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
          (conv1): Conv2d(
            512, 256, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
```

```
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (3): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (4): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
```

```
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (5): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
      )
      (res5): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
          (conv1): Conv2d(
            1024, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
```

```
        (2): BottleneckBlock(
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
      )
    )
  )
  (proposal_generator): RPN(
    (rpn_head): StandardRPNHead(
      (conv): Conv2d(
        256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
        (activation): ReLU()
      )
      (objectness_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1,
1))
      (anchor_deltas): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
    (anchor_generator): DefaultAnchorGenerator(
      (cell_anchors): BufferList()
    )
  )
  (roi_heads): StandardROIHeads(
    (box_pooler): ROIPooler(
      (level_poolers): ModuleList(
        (0): ROIAlign(output_size=(7, 7), spatial_scale=0.25,
sampling_ratio=0, aligned=True)
        (1): ROIAlign(output_size=(7, 7), spatial_scale=0.125,
sampling_ratio=0, aligned=True)
        (2): ROIAlign(output_size=(7, 7), spatial_scale=0.0625,
sampling_ratio=0, aligned=True)
        (3): ROIAlign(output_size=(7, 7), spatial_scale=0.03125,
sampling_ratio=0, aligned=True)
      )
    )
    (box_head): FastRCNNConvFCHead(
      (flatten): Flatten(start_dim=1, end_dim=-1)
      (fc1): Linear(in_features=12544, out_features=1024, bias=True)
      (fc_relu1): ReLU()
      (fc2): Linear(in_features=1024, out_features=1024, bias=True)
      (fc_relu2): ReLU()
    )
    (box_predictor): FastRCNNOutputLayers(
      (cls_score): Linear(in_features=1024, out_features=21, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=80, bias=True)
```

```
    )
  )
)
[11/23 19:33:04 d2.data.build]: Removed 0 images with no usable
annotations. 2501 images
```

RetinaNet Model Architecture…

```
RetinaNet(
  (backbone): FPN(
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (top_block): LastLevelP6P7(
      (p6): Conv2d(2048, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1))
      (p7): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1))
    )
    (bottom_up): ResNet(
      (stem): BasicStem(
        (conv1): Conv2d(
          3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
      )
      (res2): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv1): Conv2d(
            64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
```

```
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv2): Conv2d(
            64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
          )
          (conv3): Conv2d(
            64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
        )
      )
      (res3): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv1): Conv2d(
            256, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
```

```
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
        (3): BottleneckBlock(
          (conv1): Conv2d(
            512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv2): Conv2d(
            128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
          )
          (conv3): Conv2d(
            128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
        )
      )
      (res4): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
          (conv1): Conv2d(
```

```
            512, 256, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (3): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
```

```
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (4): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
        (5): BottleneckBlock(
          (conv1): Conv2d(
            1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv2): Conv2d(
            256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
          )
          (conv3): Conv2d(
            256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
          )
        )
      )
      (res5): Sequential(
        (0): BottleneckBlock(
          (shortcut): Conv2d(
            1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
          (conv1): Conv2d(
            1024, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
        (1): BottleneckBlock(
```

```
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
        (2): BottleneckBlock(
          (conv1): Conv2d(
            2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv2): Conv2d(
            512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False
            (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
          )
          (conv3): Conv2d(
            512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
            (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
          )
        )
      )
    )
  )
  (head): RetinaNetHead(
    (cls_subnet): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (1): ReLU()
      (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (3): ReLU()
      (4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (5): ReLU()
      (6): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (7): ReLU()
    )
    (bbox_subnet): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (1): ReLU()
      (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (3): ReLU()
```

```
      (4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (5): ReLU()
      (6): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (7): ReLU()
    )
    (cls_score): Conv2d(256, 180, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (bbox_pred): Conv2d(256, 36, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
  )
  (anchor_generator): DefaultAnchorGenerator(
    (cell_anchors): BufferList()
  )
)
```

Code Snippet:

1. Faster RCNN

Install detectron2

```
!pip install pyyaml==5.1

import torch
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
# Install detectron2 that matches the above pytorch version
# See https://detectron2.readthedocs.io/tutorials/install.html for instructions
!pip install detectron2 -f
https://dl.fbaipublicfiles.com/detectron2/wheels/$CUDA_VERSION/torch$TORCH_VERSION/index.html
```

Train

```


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_2007_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 1
```

```
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/faster_rcnn_R_50_FPN_3x.yaml") # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 1
cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR
cfg.SOLVER.MAX_ITER = 3000
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()
```

Evaluation

```
from detectron2.evaluation import PascalVOCDetectionEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader

evaluator = PascalVOCDetectionEvaluator("voc_2007_train")
val_loader = build_detection_test_loader(cfg, "voc_2007_train")
print(inference_on_dataset(predictor.model, val_loader, evaluator))
# another equivalent way to evaluate the model is to use `trainer.test`
```

2.RetinaNet

Train

```
from detectron2.engine import DefaultTrainer

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/retinanet_R_50_FPN_3x.yaml"))
cfg.OUTPUT_DIR = 'MyVOCTraining_retinanet'
cfg.DATASETS.TRAIN = ("voc_2007_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 1
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/retinanet_R_50_FPN_3x.yaml") # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 1
cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR
cfg.SOLVER.MAX_ITER = 3000
```

```
cfg.MODEL.RETINANET.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.RETINANET.NUM_CLASSES = 20

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

Inference for visualization

```
path = '/content/datasets/VOC2007/JPEGImages/'
img_lists = listdir(path)

for img in random.sample(img_lists,3):
    img_path = os.path.join(path,img)
    print(img_path)
    img_read = cv2.imread(img_path)
    #cv2.imshow("Image", im)
    #cv2.waitKey(0)
    outputs = predictor(img_read)
    v = Visualizer(img_read[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]),
scale=1.2)
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2_imshow(out.get_image()[:, :, ::-1])
```

References:
[1] https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5#scrollTo=h9tECBQCvMv3
[2] https://detectron2.readthedocs.io/en/latest/_modules/detectron2/model_zoo/model_zoo.html
[3] http://host.robots.ox.ac.uk/pascal/VOC/voc2007/