This is for Question 5 Part B and C using the backbone option

```
# For tips on running notebooks in Google Colab, see
# https://pytorch.org/tutorials/beginner/colab
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive') #Updated the spots where they needed the locations to change, also downloaded the PennFundan dataset and tutorial source for the test photo at the end
# Dataset: https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
# Test Photo: https://github.com/pytorch/tutorials/blob/d686b662932a380a58b7683425faa00c06bcf502/_static/img/tv_tutorial/tv_image05.png
#Source Code: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html
```

```
Mounted at /content/drive
```

TorchVision Object Detection Finetuning Tutorial

.. tip::

```
To get the most of this tutorial, we suggest using this
[Colab
Version](https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/torchvision_finetuning_instance_segmentation.ipynb)_.
This will allow you to experiment with the information presented
below.
```

For this tutorial, we will be finetuning a pre-trained Mask R-CNN_ model on the Penn-Fudan Database for Pedestrian Detection and Segmentation_. It contains 170 images with 345 instances of pedestrians, and we will use it to illustrate how to use the new features in torchvision in order to train an object detection and instance segmentation model on a custom dataset.

.. note ::

```
This tutorial works only with torchvision version >=0.16 or nightly.
If you're using torchvision<=0.15, please follow
[this tutorial
instead](https://github.com/pytorch/tutorials/blob/d686b662932a380a58b7683425faa00c06bcf502/intermediate_source/torchvision_tutorial.rst).
```

Defining the Dataset

The reference scripts for training object detection, instance segmentation and person keypoint detection allows for easily supporting adding new custom datasets. The dataset should inherit from the standard `torch.utils.data.Dataset` class, and implement `__len__` and `__getitem__`.

The only specificity that we require is that the dataset `__getitem__` should return a tuple:

- image: :class:`torchvision.tv_tensors.Image` of shape `[3, H, W]`, a pure tensor, or a PIL Image of size `(H, W)`
- target: a dict containing the following fields
  - `boxes`, :class:`torchvision.tv_tensors.BoundingBoxes` of shape `[N, 4]`: the coordinates of the `N` bounding boxes in `[x0, y0, x1, y1]` format, ranging from `0` to `W` and `0` to `H`
  - `labels`, integer :class:`torch.Tensor` of shape `[N]`: the label for each bounding box. `0` represents always the background class.
  - `image_id`, int: an image identifier. It should be unique between all the images in the dataset, and is used during evaluation
  - `area`, float :class:`torch.Tensor` of shape `[N]`: the area of the bounding box. This is used during evaluation with the COCO metric, to separate the metric scores between small, medium and large boxes.
  - `iscrowd`, uint8 :class:`torch.Tensor` of shape `[N]`: instances with `iscrowd=True` will be ignored during evaluation.
  - (optionally) `masks`, :class:`torchvision.tv_tensors.Mask` of shape `[N, H, W]`: the segmentation masks for each one of the objects

If your dataset is compliant with above requirements then it will work for both training and evaluation codes from the reference script. Evaluation code will use scripts from `pycocotools` which can be installed with `pip install pycocotools`.

.. note :: For Windows, please install `pycocotools` from gautamchitnis_ with command

```
pip install
git+https://github.com/gautamchitnis/cocoapi.git@cocodataset-
master#subdirectory=PythonAPI
```

One note on the `labels`. The model considers class `0` as background. If your dataset does not contain the background class, you should not have `0` in your `labels`. For example, assuming you have just two classes, *cat* and *dog*, you can define `1` (not `0`) to represent *cats* and `2` to represent *dogs*. So, for instance, if one of the images has both classes, your `labels` tensor should look like `[1, 2]`.

Additionally, if you want to use aspect ratio grouping during training (so that each batch only contains images with similar aspect ratios), then it is recommended to also implement a `get_height_and_width` method, which returns the height and the width of the image. If this method is not provided, we query all elements of the dataset via `__getitem__` , which loads the image in memory and is slower than if a custom method is provided.

Writing a custom dataset for PennFudan

Let's write a dataset for the PennFudan dataset. After downloading and extracting the zip file_, we have the following folder structure:

::

PennFudanPed/ PedMasks/ FudanPed00001_mask.png FudanPed00002_mask.png FudanPed00003_mask.png FudanPed00004_mask.png ... PNGImages/ FudanPed00001.png FudanPed00002.png FudanPed00003.png FudanPed00004.png

Here is one example of a pair of images and segmentation masks

So each image has a corresponding segmentation mask, where each color correspond to a different instance. Let's write a :class:`torch.utils.data.Dataset` class for this dataset. In the code below, we are wrapping images, bounding boxes and masks into `torchvision.TVTensor` classes so that we will be able to apply torchvision built-in transformations (new Transforms API) for the given object detection and segmentation task. Namely, image tensors will be wrapped by :class:`torchvision.tv_tensors.Image`, bounding boxes into :class:`torchvision.tv_tensors.BoundingBoxes` and masks into :class:`torchvision.tv_tensors.Mask`. As `torchvision.TVTensor` are :class:`torch.Tensor` subclasses, wrapped objects are also tensors and inherit the plain :class:`torch.Tensor` API. For more information about torchvision `tv_tensors` see this documentation.

```
import os
import torch

from torchvision.io import read_image
from torchvision.ops.boxes import masks_to_boxes
from torchvision import tv_tensors
from torchvision.transforms.v2 import functional as F


class PennFudanDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms):
        self.root = root
        self.transforms = transforms
        # load all image files, sorting them to
        # ensure that they are aligned
        self.imgs = list(sorted(os.listdir(os.path.join(root,
"PNGImages"))))
        self.masks = list(sorted(os.listdir(os.path.join(root,
"PedMasks"))))

    def __getitem__(self, idx):
        # load images and masks
```

```
        img_path = os.path.join(self.root, "PNGImages",
self.imgs[idx])
        mask_path = os.path.join(self.root, "PedMasks",
self.masks[idx])
        img = read_image(img_path)
        mask = read_image(mask_path)
        # instances are encoded as different colors
        obj_ids = torch.unique(mask)
        # first id is the background, so remove it
        obj_ids = obj_ids[1:]
        num_objs = len(obj_ids)

        # split the color-encoded mask into a set
        # of binary masks
        masks = (mask == obj_ids[:, None, None]).to(dtype=torch.uint8)

        # get bounding box coordinates for each mask
        boxes = masks_to_boxes(masks)

        # there is only one class
        labels = torch.ones((num_objs,), dtype=torch.int64)

        image_id = idx
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:,
0])
        # suppose all instances are not crowd
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

        # Wrap sample and targets into torchvision tv_tensors:
        img = tv_tensors.Image(img)

        target = {}
        target["boxes"] = tv_tensors.BoundingBoxes(boxes,
format="XYXY", canvas_size=F.get_size(img))
        target["masks"] = tv_tensors.Mask(masks)
        target["labels"] = labels
        target["image_id"] = image_id
        target["area"] = area
        target["iscrowd"] = iscrowd

        if self.transforms is not None:
            img, target = self.transforms(img, target)

        return img, target

    def __len__(self):
        return len(self.imgs)
```

That's all for the dataset. Now let's define a model that can perform predictions on this dataset.

Defining your model

In this tutorial, we will be using Mask R-CNN, *which is based on top of Faster R-CNN*. Faster R-CNN is a model that predicts both bounding boxes and class scores for potential objects in the image.

Mask R-CNN adds an extra branch into Faster R-CNN, which also predicts segmentation masks for each instance.

There are two common situations where one might want to modify one of the available models in TorchVision Model Zoo. The first is when we want to start from a pre-trained model, and just finetune the last layer. The other is when we want to replace the backbone of the model with a different one (for faster predictions, for example).

Let's go see how we would do one or another in the following sections.

2 - Modifying the model to add a different backbone

```
import torchvision
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator

# load a pre-trained model for classification and return
# only the features
backbone = torchvision.models.mobilenet_v2(weights="DEFAULT").features
# ``FasterRCNN`` needs to know the number of
# output channels in a backbone. For mobilenet_v2, it's 1280
# so we need to add it here
backbone.out_channels = 1280

# let's make the RPN generate 5 x 3 anchors per spatial
# location, with 5 different sizes and 3 different aspect
# ratios. We have a Tuple[Tuple[int]] because each feature
# map could potentially have different sizes and
# aspect ratios
anchor_generator = AnchorGenerator(
    sizes=((32, 64, 128, 256, 512),),
    aspect_ratios=((0.5, 1.0, 2.0),)
)

# let's define what are the feature maps that we will
# use to perform the region of interest cropping, as well as
# the size of the crop after rescaling.
# if your backbone returns a Tensor, featmap_names is expected to
# be [0]. More generally, the backbone should return an
# ``OrderedDict[Tensor]``, and in ``featmap_names`` you can choose
which
```

```
# feature maps to use.
roi_pooler = torchvision.ops.MultiScaleRoIAlign(
    featmap_names=['0'],
    output_size=7,
    sampling_ratio=2
)

# put the pieces together inside a Faster-RCNN model
model = FasterRCNN(
    backbone,
    num_classes=2,
    rpn_anchor_generator=anchor_generator,
    box_roi_pool=roi_pooler
)
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-
7ebf99e0.pth" to /root/.cache/torch/hub/checkpoints/mobilenet_v2-
7ebf99e0.pth
100%|██████████| 13.6M/13.6M [00:00<00:00, 80.2MB/s]
```

Object detection and instance segmentation model for PennFudan Dataset

In our case, we want to finetune from a pre-trained model, given that our dataset is very small, so we will be following approach number 1.

Here we want to also compute the instance segmentation masks, so we will be using Mask R-CNN:

```
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor


def get_model_instance_segmentation(num_classes):
    # load an instance segmentation model pre-trained on COCO
    model =
torchvision.models.detection.maskrcnn_resnet50_fpn(weights="DEFAULT")

    # get number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features,
num_classes)

    # now get the number of input features for the mask classifier
    in_features_mask =
model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256
```

```
    # and replace the mask predictor with a new one
    model.roi_heads.mask_predictor = MaskRCNNPredictor(
        in_features_mask,
        hidden_layer,
        num_classes
    )

    return model
```

That's it, this will make `model` be ready to be trained and evaluated on your custom dataset.

Putting everything together

In `references/detection/`, we have a number of helper functions to simplify training and evaluating detection models. Here, we will use `references/detection/engine.py` and `references/detection/utils.py`. Just download everything under `references/detection` to your folder and use them here. On Linux if you have `wget`, you can download them using below commands:

```
os.system("wget
https://raw.githubusercontent.com/pytorch/vision/main/references/detec
tion/engine.py")
os.system("wget
https://raw.githubusercontent.com/pytorch/vision/main/references/detec
tion/utils.py")
os.system("wget
https://raw.githubusercontent.com/pytorch/vision/main/references/detec
tion/coco_utils.py")
os.system("wget
https://raw.githubusercontent.com/pytorch/vision/main/references/detec
tion/coco_eval.py")
os.system("wget
https://raw.githubusercontent.com/pytorch/vision/main/references/detec
tion/transforms.py")

# Since v0.15.0 torchvision provides `new Transforms API
<https://pytorch.org/vision/stable/transforms.html>`_
# to easily write data augmentation pipelines for Object Detection and
Segmentation tasks.
#
# Let's write some helper functions for data augmentation /
# transformation:

from torchvision.transforms import v2 as T


def get_transform(train):
    transforms = []
    if train:
```

```
        transforms.append(T.RandomHorizontalFlip(0.5))
    transforms.append(T.ToDtype(torch.float, scale=True))
    transforms.append(T.ToPureTensor())
    return T.Compose(transforms)


# Testing ``forward()`` method (Optional)
# ---------------------------------------
#
# Before iterating over the dataset, it's good to see what the model
# expects during training and inference time on sample data.
import utils


model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(weights="DEFAULT"
)
dataset = PennFudanDataset('drive/MyDrive/data/PennFudanPed',
get_transform(train=True))
data_loader = torch.utils.data.DataLoader(
    dataset,
    batch_size=2,
    shuffle=True,
    num_workers=4,
    collate_fn=utils.collate_fn
)

# For Training
images, targets = next(iter(data_loader))
images = list(image for image in images)
targets = [{k: v for k, v in t.items()} for t in targets]
output = model(images, targets)  # Returns losses and detections
print(output)

# For inference
model.eval()
x = [torch.rand(3, 300, 400), torch.rand(3, 500, 400)]
predictions = model(x)  # Returns predictions
print(predictions[0])

Downloading:
"https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-
258fb6c6.pth" to
/root/.cache/torch/hub/checkpoints/fasterrcnn_resnet50_fpn_coco-
258fb6c6.pth
100%|██████████| 160M/160M [00:00<00:00, 171MB/s]
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py
:557: UserWarning: This DataLoader will create 4 worker processes in
total. Our suggested max number of worker in current system is 2,
which is smaller than what this DataLoader is going to create. Please
```

```
be aware that excessive worker creation might get DataLoader running
slow or even freeze, lower the worker number to avoid potential
slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(

{'loss_classifier': tensor(0.2291, grad_fn=<NllLossBackward0>),
'loss_box_reg': tensor(0.1100, grad_fn=<DivBackward0>),
'loss_objectness': tensor(0.0078,
grad_fn=<BinaryCrossEntropyWithLogitsBackward0>), 'loss_rpn_box_reg':
tensor(0.0105, grad_fn=<DivBackward0>)}
{'boxes': tensor([], size=(0, 4), grad_fn=<StackBackward0>), 'labels':
tensor([], dtype=torch.int64), 'scores': tensor([],
grad_fn=<IndexBackward0>)}
```

Let's now write the main function which performs the training and the validation:

```
from engine import train_one_epoch, evaluate

# train on the GPU or on the CPU, if a GPU is not available
device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

# our dataset has two classes only - background and person
num_classes = 2
# use our dataset and defined transformations
dataset = PennFudanDataset('drive/MyDrive/data/PennFudanPed',
get_transform(train=True))
dataset_test = PennFudanDataset('drive/MyDrive/data/PennFudanPed',
get_transform(train=False))

# split the dataset in train and test set
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-50])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-50:])

# define training and validation data loaders
data_loader = torch.utils.data.DataLoader(
    dataset,
    batch_size=2,
    shuffle=True,
    num_workers=4,
    collate_fn=utils.collate_fn
)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test,
    batch_size=1,
    shuffle=False,
    num_workers=4,
    collate_fn=utils.collate_fn
```

```
)

# get the model using our helper function
model = get_model_instance_segmentation(num_classes)

# move model to the right device
model.to(device)

# construct an optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(
    params,
    lr=0.005,
    momentum=0.9,
    weight_decay=0.0005
)

# and a learning rate scheduler
lr_scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.1
)

# let's train it for 5 epochs
num_epochs = 10

for epoch in range(num_epochs):
    # train for one epoch, printing every 10 iterations
    train_one_epoch(model, optimizer, data_loader, device, epoch,
print_freq=10)
    # update the learning rate
    lr_scheduler.step()
    # evaluate on the test dataset
    evaluate(model, data_loader_test, device=device)

print("That's it!")
```

```
Downloading:
"https://download.pytorch.org/models/maskrcnn_resnet50_fpn_coco-
bf2d0c1e.pth" to
/root/.cache/torch/hub/checkpoints/maskrcnn_resnet50_fpn_coco-
bf2d0c1e.pth
100%|██████████| 170M/170M [00:01<00:00, 154MB/s]

Epoch: [0]  [ 0/60]  eta: 0:09:12  lr: 0.000090  loss: 5.0986 (5.0986)
loss_classifier: 0.5976 (0.5976)  loss_box_reg: 0.1450 (0.1450)
loss_mask: 4.3445 (4.3445)  loss_objectness: 0.0105 (0.0105)
loss_rpn_box_reg: 0.0011 (0.0011)  time: 9.2085  data: 1.7603  max
mem: 2065
```

```
Epoch: [0]  [10/60]  eta: 0:01:05  lr: 0.000936  loss: 1.7777 (2.5739)
loss_classifier: 0.4427 (0.4398)  loss_box_reg: 0.2589 (0.2873)
loss_mask: 1.1316 (1.8147)  loss_objectness: 0.0235 (0.0256)
loss_rpn_box_reg: 0.0042 (0.0064)  time: 1.3122  data: 0.1694  max
mem: 2764
Epoch: [0]  [20/60]  eta: 0:00:37  lr: 0.001783  loss: 0.9253 (1.6445)
loss_classifier: 0.1694 (0.2943)  loss_box_reg: 0.2363 (0.2486)
loss_mask: 0.4028 (1.0708)  loss_objectness: 0.0174 (0.0243)
loss_rpn_box_reg: 0.0060 (0.0064)  time: 0.5337  data: 0.0097  max
mem: 3211
Epoch: [0]  [30/60]  eta: 0:00:24  lr: 0.002629  loss: 0.5503 (1.2777)
loss_classifier: 0.0942 (0.2279)  loss_box_reg: 0.2240 (0.2401)
loss_mask: 0.1915 (0.7836)  loss_objectness: 0.0093 (0.0189)
loss_rpn_box_reg: 0.0062 (0.0071)  time: 0.5428  data: 0.0086  max
mem: 3211
Epoch: [0]  [40/60]  eta: 0:00:15  lr: 0.003476  loss: 0.4641 (1.0915)
loss_classifier: 0.0733 (0.1892)  loss_box_reg: 0.2205 (0.2392)
loss_mask: 0.1874 (0.6412)  loss_objectness: 0.0037 (0.0150)
loss_rpn_box_reg: 0.0075 (0.0069)  time: 0.5498  data: 0.0110  max
mem: 3211
Epoch: [0]  [50/60]  eta: 0:00:07  lr: 0.004323  loss: 0.4495 (0.9692)
loss_classifier: 0.0539 (0.1633)  loss_box_reg: 0.2102 (0.2342)
loss_mask: 0.1874 (0.5518)  loss_objectness: 0.0021 (0.0130)
loss_rpn_box_reg: 0.0058 (0.0069)  time: 0.5793  data: 0.0148  max
mem: 3222
Epoch: [0]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.3703 (0.8745)
loss_classifier: 0.0415 (0.1442)  loss_box_reg: 0.1577 (0.2203)
loss_mask: 0.1652 (0.4921)  loss_objectness: 0.0017 (0.0112)
loss_rpn_box_reg: 0.0058 (0.0067)  time: 0.5651  data: 0.0123  max
mem: 3222
Epoch: [0] Total time: 0:00:41 (0.6973 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:28  model_time: 0.2362 (0.2362)
evaluator_time: 0.0045 (0.0045)  time: 0.5697  data: 0.3278  max mem:
3222
Test:  [49/50]  eta: 0:00:00  model_time: 0.0982 (0.1290)
evaluator_time: 0.0053 (0.0121)  time: 0.1231  data: 0.0049  max mem:
3222
Test: Total time: 0:00:07 (0.1594 s / it)
Averaged stats: model_time: 0.0982 (0.1290)  evaluator_time: 0.0053
(0.0121)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.584
```

```
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.954
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.676
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.267
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.596
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.596
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.266
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.667
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.667
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.400
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.733
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.668
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.664
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.961
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.798
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.294
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.432
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.688
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.300
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.722
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.723
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.533
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.692
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.732
Epoch: [1]  [ 0/60]  eta: 0:00:52  lr: 0.005000  loss: 0.2154 (0.2154)
loss_classifier: 0.0134 (0.0134)  loss_box_reg: 0.0351 (0.0351)
```

```
loss_mask: 0.1655 (0.1655)  loss_objectness: 0.0000 (0.0000)
loss_rpn_box_reg: 0.0014 (0.0014)  time: 0.8740  data: 0.3722  max
mem: 3222
Epoch: [1]  [10/60]  eta: 0:00:29  lr: 0.005000  loss: 0.2632 (0.3034)
loss_classifier: 0.0393 (0.0371)  loss_box_reg: 0.0935 (0.1025)
loss_mask: 0.1525 (0.1554)  loss_objectness: 0.0010 (0.0015)
loss_rpn_box_reg: 0.0036 (0.0068)  time: 0.5839  data: 0.0398  max
mem: 3222
Epoch: [1]  [20/60]  eta: 0:00:23  lr: 0.005000  loss: 0.2812 (0.3210)
loss_classifier: 0.0393 (0.0425)  loss_box_reg: 0.1028 (0.1159)
loss_mask: 0.1403 (0.1537)  loss_objectness: 0.0010 (0.0017)
loss_rpn_box_reg: 0.0061 (0.0072)  time: 0.5770  data: 0.0092  max
mem: 3222
Epoch: [1]  [30/60]  eta: 0:00:17  lr: 0.005000  loss: 0.2619 (0.2951)
loss_classifier: 0.0371 (0.0390)  loss_box_reg: 0.1001 (0.1037)
loss_mask: 0.1330 (0.1445)  loss_objectness: 0.0008 (0.0017)
loss_rpn_box_reg: 0.0057 (0.0061)  time: 0.5975  data: 0.0097  max
mem: 3499
Epoch: [1]  [40/60]  eta: 0:00:11  lr: 0.005000  loss: 0.2293 (0.2814)
loss_classifier: 0.0254 (0.0362)  loss_box_reg: 0.0697 (0.0951)
loss_mask: 0.1265 (0.1430)  loss_objectness: 0.0004 (0.0014)
loss_rpn_box_reg: 0.0037 (0.0057)  time: 0.5854  data: 0.0087  max
mem: 3499
Epoch: [1]  [50/60]  eta: 0:00:05  lr: 0.005000  loss: 0.2533 (0.2769)
loss_classifier: 0.0271 (0.0350)  loss_box_reg: 0.0671 (0.0901)
loss_mask: 0.1385 (0.1447)  loss_objectness: 0.0004 (0.0017)
loss_rpn_box_reg: 0.0040 (0.0055)  time: 0.5632  data: 0.0091  max
mem: 3499
Epoch: [1]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.2533 (0.2748)
loss_classifier: 0.0292 (0.0352)  loss_box_reg: 0.0677 (0.0869)
loss_mask: 0.1429 (0.1455)  loss_objectness: 0.0004 (0.0015)
loss_rpn_box_reg: 0.0044 (0.0058)  time: 0.5583  data: 0.0081  max
mem: 3499
Epoch: [1] Total time: 0:00:35 (0.5834 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:36  model_time: 0.2662 (0.2662)
evaluator_time: 0.0041 (0.0041)  time: 0.7247  data: 0.4532  max mem:
3499
Test:  [49/50]  eta: 0:00:00  model_time: 0.0967 (0.1095)
evaluator_time: 0.0038 (0.0059)  time: 0.1158  data: 0.0042  max mem:
3499
Test: Total time: 0:00:06 (0.1344 s / it)
Averaged stats: model_time: 0.0967 (0.1095)  evaluator_time: 0.0038
(0.0059)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.02s).
```

```
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.738
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.969
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.932
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.381
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.688
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.752
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.330
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.796
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.796
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.500
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.806
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.692
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.974
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.826
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.329
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.619
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.706
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.311
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.748
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.748
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.533
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.750
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
```

```
maxDets=100 ] = 0.753
Epoch: [2]  [ 0/60]  eta: 0:00:53  lr: 0.005000  loss: 0.1752 (0.1752)
loss_classifier: 0.0203 (0.0203)  loss_box_reg: 0.0485 (0.0485)
loss_mask: 0.1022 (0.1022)  loss_objectness: 0.0004 (0.0004)
loss_rpn_box_reg: 0.0038 (0.0038)  time: 0.8881  data: 0.3680  max
mem: 3499
Epoch: [2]  [10/60]  eta: 0:00:29  lr: 0.005000  loss: 0.2351 (0.2578)
loss_classifier: 0.0322 (0.0362)  loss_box_reg: 0.0631 (0.0793)
loss_mask: 0.1351 (0.1367)  loss_objectness: 0.0005 (0.0007)
loss_rpn_box_reg: 0.0038 (0.0048)  time: 0.5899  data: 0.0413  max
mem: 3499
Epoch: [2]  [20/60]  eta: 0:00:23  lr: 0.005000  loss: 0.2351 (0.2564)
loss_classifier: 0.0303 (0.0325)  loss_box_reg: 0.0725 (0.0793)
loss_mask: 0.1351 (0.1394)  loss_objectness: 0.0005 (0.0007)
loss_rpn_box_reg: 0.0036 (0.0046)  time: 0.5669  data: 0.0089  max
mem: 3499
Epoch: [2]  [30/60]  eta: 0:00:17  lr: 0.005000  loss: 0.2122 (0.2387)
loss_classifier: 0.0246 (0.0306)  loss_box_reg: 0.0517 (0.0711)
loss_mask: 0.1209 (0.1320)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0031 (0.0043)  time: 0.5666  data: 0.0091  max
mem: 3499
Epoch: [2]  [40/60]  eta: 0:00:11  lr: 0.005000  loss: 0.2139 (0.2401)
loss_classifier: 0.0334 (0.0315)  loss_box_reg: 0.0517 (0.0713)
loss_mask: 0.1146 (0.1319)  loss_objectness: 0.0004 (0.0007)
loss_rpn_box_reg: 0.0031 (0.0046)  time: 0.5879  data: 0.0099  max
mem: 3506
Epoch: [2]  [50/60]  eta: 0:00:05  lr: 0.005000  loss: 0.2241 (0.2373)
loss_classifier: 0.0339 (0.0311)  loss_box_reg: 0.0579 (0.0691)
loss_mask: 0.1267 (0.1320)  loss_objectness: 0.0007 (0.0007)
loss_rpn_box_reg: 0.0034 (0.0044)  time: 0.5892  data: 0.0091  max
mem: 3506
Epoch: [2]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.2103 (0.2388)
loss_classifier: 0.0257 (0.0310)  loss_box_reg: 0.0546 (0.0685)
loss_mask: 0.1331 (0.1340)  loss_objectness: 0.0005 (0.0008)
loss_rpn_box_reg: 0.0029 (0.0045)  time: 0.5629  data: 0.0074  max
mem: 3506
Epoch: [2] Total time: 0:00:34 (0.5829 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:27  model_time: 0.1925 (0.1925)
evaluator_time: 0.0037 (0.0037)  time: 0.5428  data: 0.3454  max mem:
3506
Test:  [49/50]  eta: 0:00:00  model_time: 0.0969 (0.1068)
evaluator_time: 0.0032 (0.0049)  time: 0.1137  data: 0.0037  max mem:
3506
Test: Total time: 0:00:06 (0.1284 s / it)
Averaged stats: model_time: 0.0969 (0.1068)  evaluator_time: 0.0032
(0.0049)
Accumulating evaluation results...
```

```
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.764
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.905
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.292
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.696
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.786
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.339
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.820
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.820
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.433
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.758
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.837
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.748
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.973
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.938
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.315
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.567
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.769
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.326
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.792
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.792
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.433
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.758
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.805
Epoch: [3]  [ 0/60]  eta: 0:00:55  lr: 0.000500  loss: 0.1499 (0.1499)
loss_classifier: 0.0188 (0.0188)  loss_box_reg: 0.0275 (0.0275)
loss_mask: 0.0995 (0.0995)  loss_objectness: 0.0013 (0.0013)
loss_rpn_box_reg: 0.0028 (0.0028)  time: 0.9178  data: 0.3975  max
mem: 3506
Epoch: [3]  [10/60]  eta: 0:00:30  lr: 0.000500  loss: 0.1933 (0.1984)
loss_classifier: 0.0250 (0.0240)  loss_box_reg: 0.0381 (0.0457)
loss_mask: 0.1201 (0.1236)  loss_objectness: 0.0010 (0.0020)
loss_rpn_box_reg: 0.0023 (0.0030)  time: 0.6011  data: 0.0473  max
mem: 3506
Epoch: [3]  [20/60]  eta: 0:00:23  lr: 0.000500  loss: 0.1933 (0.1941)
loss_classifier: 0.0217 (0.0244)  loss_box_reg: 0.0381 (0.0418)
loss_mask: 0.1204 (0.1229)  loss_objectness: 0.0007 (0.0018)
loss_rpn_box_reg: 0.0023 (0.0033)  time: 0.5586  data: 0.0100  max
mem: 3506
Epoch: [3]  [30/60]  eta: 0:00:17  lr: 0.000500  loss: 0.1934 (0.1968)
loss_classifier: 0.0248 (0.0255)  loss_box_reg: 0.0394 (0.0434)
loss_mask: 0.1204 (0.1232)  loss_objectness: 0.0004 (0.0016)
loss_rpn_box_reg: 0.0032 (0.0032)  time: 0.5505  data: 0.0101  max
mem: 3506
Epoch: [3]  [40/60]  eta: 0:00:11  lr: 0.000500  loss: 0.1934 (0.1993)
loss_classifier: 0.0257 (0.0261)  loss_box_reg: 0.0436 (0.0445)
loss_mask: 0.1213 (0.1236)  loss_objectness: 0.0006 (0.0017)
loss_rpn_box_reg: 0.0034 (0.0034)  time: 0.5638  data: 0.0108  max
mem: 3506
Epoch: [3]  [50/60]  eta: 0:00:05  lr: 0.000500  loss: 0.1763 (0.1969)
loss_classifier: 0.0241 (0.0258)  loss_box_reg: 0.0361 (0.0436)
loss_mask: 0.1137 (0.1223)  loss_objectness: 0.0007 (0.0019)
loss_rpn_box_reg: 0.0025 (0.0032)  time: 0.5697  data: 0.0088  max
mem: 3506
Epoch: [3]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1844 (0.1977)
loss_classifier: 0.0217 (0.0261)  loss_box_reg: 0.0374 (0.0444)
loss_mask: 0.1123 (0.1220)  loss_objectness: 0.0004 (0.0017)
loss_rpn_box_reg: 0.0027 (0.0034)  time: 0.5720  data: 0.0080  max
mem: 3506
Epoch: [3] Total time: 0:00:34 (0.5757 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:26  model_time: 0.2132 (0.2132)
evaluator_time: 0.0038 (0.0038)  time: 0.5280  data: 0.3096  max mem:
3506
Test:  [49/50]  eta: 0:00:00  model_time: 0.1036 (0.1098)
evaluator_time: 0.0050 (0.0060)  time: 0.1220  data: 0.0047  max mem:
3506
Test: Total time: 0:00:06 (0.1367 s / it)
```

```
Averaged stats: model_time: 0.1036 (0.1098)  evaluator_time: 0.0050
(0.0060)
Accumulating evaluation results...
DONE (t=0.03s).
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.791
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.977
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.923
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.372
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.703
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.812
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.348
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.850
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.850
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.567
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.866
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.754
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.977
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.937
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.368
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.523
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.773
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.331
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.803
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
```

```
maxDets=100 ] = 0.803
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.600
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.812
Epoch: [4]  [ 0/60]  eta: 0:01:20  lr: 0.000500  loss: 0.2263 (0.2263)
loss_classifier: 0.0418 (0.0418)  loss_box_reg: 0.0667 (0.0667)
loss_mask: 0.1081 (0.1081)  loss_objectness: 0.0019 (0.0019)
loss_rpn_box_reg: 0.0078 (0.0078)  time: 1.3454  data: 0.6900  max
mem: 3506
Epoch: [4]  [10/60]  eta: 0:00:31  lr: 0.000500  loss: 0.1923 (0.1878)
loss_classifier: 0.0294 (0.0274)  loss_box_reg: 0.0394 (0.0393)
loss_mask: 0.1081 (0.1172)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0016 (0.0031)  time: 0.6336  data: 0.0698  max
mem: 3506
Epoch: [4]  [20/60]  eta: 0:00:24  lr: 0.000500  loss: 0.1878 (0.1919)
loss_classifier: 0.0271 (0.0274)  loss_box_reg: 0.0394 (0.0438)
loss_mask: 0.1124 (0.1166)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0022 (0.0034)  time: 0.5651  data: 0.0086  max
mem: 3506
Epoch: [4]  [30/60]  eta: 0:00:17  lr: 0.000500  loss: 0.1777 (0.1855)
loss_classifier: 0.0248 (0.0265)  loss_box_reg: 0.0356 (0.0394)
loss_mask: 0.1126 (0.1156)  loss_objectness: 0.0002 (0.0009)
loss_rpn_box_reg: 0.0017 (0.0030)  time: 0.5693  data: 0.0113  max
mem: 3506
Epoch: [4]  [40/60]  eta: 0:00:11  lr: 0.000500  loss: 0.1566 (0.1846)
loss_classifier: 0.0217 (0.0255)  loss_box_reg: 0.0259 (0.0393)
loss_mask: 0.1126 (0.1161)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0014 (0.0028)  time: 0.5630  data: 0.0128  max
mem: 3506
Epoch: [4]  [50/60]  eta: 0:00:05  lr: 0.000500  loss: 0.1653 (0.1845)
loss_classifier: 0.0224 (0.0254)  loss_box_reg: 0.0278 (0.0396)
loss_mask: 0.1163 (0.1159)  loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0020 (0.0028)  time: 0.5632  data: 0.0133  max
mem: 3506
Epoch: [4]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1707 (0.1862)
loss_classifier: 0.0241 (0.0255)  loss_box_reg: 0.0333 (0.0407)
loss_mask: 0.1126 (0.1164)  loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0029 (0.0030)  time: 0.5827  data: 0.0109  max
mem: 3506
Epoch: [4] Total time: 0:00:35 (0.5868 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:26  model_time: 0.1612 (0.1612)
evaluator_time: 0.0042 (0.0042)  time: 0.5203  data: 0.3537  max mem:
3506
Test:  [49/50]  eta: 0:00:00  model_time: 0.0973 (0.1117)
```

```
evaluator_time: 0.0033 (0.0072)  time: 0.1171  data: 0.0053  max mem:
3506
Test: Total time: 0:00:06 (0.1387 s / it)
Averaged stats: model_time: 0.0973 (0.1117)  evaluator_time: 0.0033
(0.0072)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.802
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.913
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.335
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.700
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.825
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.354
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.849
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.849
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.868
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.758
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.945
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.313
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.527
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.780
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.334
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
```

```
10 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.815
Epoch: [5]  [ 0/60]  eta: 0:01:01  lr: 0.000500  loss: 0.1311 (0.1311)
loss_classifier: 0.0145 (0.0145)  loss_box_reg: 0.0197 (0.0197)
loss_mask: 0.0942 (0.0942)  loss_objectness: 0.0002 (0.0002)
loss_rpn_box_reg: 0.0024 (0.0024)  time: 1.0227  data: 0.4592  max
mem: 3506
Epoch: [5]  [10/60]  eta: 0:00:27  lr: 0.000500  loss: 0.1704 (0.1697)
loss_classifier: 0.0172 (0.0197)  loss_box_reg: 0.0356 (0.0336)
loss_mask: 0.1118 (0.1133)  loss_objectness: 0.0006 (0.0010)
loss_rpn_box_reg: 0.0024 (0.0021)  time: 0.5558  data: 0.0470  max
mem: 3506
Epoch: [5]  [20/60]  eta: 0:00:22  lr: 0.000500  loss: 0.1704 (0.1813)
loss_classifier: 0.0216 (0.0224)  loss_box_reg: 0.0365 (0.0369)
loss_mask: 0.1150 (0.1184)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0026 (0.0028)  time: 0.5431  data: 0.0072  max
mem: 3506
Epoch: [5]  [30/60]  eta: 0:00:17  lr: 0.000500  loss: 0.1792 (0.1873)
loss_classifier: 0.0254 (0.0234)  loss_box_reg: 0.0371 (0.0392)
loss_mask: 0.1242 (0.1211)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0027 (0.0028)  time: 0.5843  data: 0.0089  max
mem: 3506
Epoch: [5]  [40/60]  eta: 0:00:11  lr: 0.000500  loss: 0.1743 (0.1817)
loss_classifier: 0.0224 (0.0225)  loss_box_reg: 0.0323 (0.0373)
loss_mask: 0.1098 (0.1183)  loss_objectness: 0.0003 (0.0010)
loss_rpn_box_reg: 0.0017 (0.0027)  time: 0.5693  data: 0.0092  max
mem: 3506
Epoch: [5]  [50/60]  eta: 0:00:05  lr: 0.000500  loss: 0.1669 (0.1854)
loss_classifier: 0.0195 (0.0239)  loss_box_reg: 0.0334 (0.0392)
loss_mask: 0.1054 (0.1183)  loss_objectness: 0.0006 (0.0011)
loss_rpn_box_reg: 0.0024 (0.0029)  time: 0.5706  data: 0.0094  max
mem: 3506
Epoch: [5]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1669 (0.1838)
loss_classifier: 0.0234 (0.0236)  loss_box_reg: 0.0390 (0.0390)
loss_mask: 0.1064 (0.1172)  loss_objectness: 0.0003 (0.0011)
loss_rpn_box_reg: 0.0025 (0.0029)  time: 0.5905  data: 0.0086  max
mem: 3506
Epoch: [5] Total time: 0:00:34 (0.5787 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:32  model_time: 0.1621 (0.1621)
evaluator_time: 0.0040 (0.0040)  time: 0.6579  data: 0.4907  max mem:
```

```
3506
Test:  [49/50]  eta: 0:00:00  model_time: 0.0967 (0.1065)
evaluator_time: 0.0041 (0.0050)  time: 0.1136  data: 0.0037  max mem:
3506
Test: Total time: 0:00:06 (0.1310 s / it)
Averaged stats: model_time: 0.0967 (0.1065)  evaluator_time: 0.0041
(0.0050)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.809
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.973
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.913
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.348
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.706
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.831
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.356
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.854
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.854
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.783
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.872
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.760
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.954
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.313
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.562
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.781
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.334
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.815
Epoch: [6]  [ 0/60]  eta: 0:00:59  lr: 0.000050  loss: 0.1643 (0.1643)
loss_classifier: 0.0163 (0.0163)  loss_box_reg: 0.0271 (0.0271)
loss_mask: 0.1181 (0.1181)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0027 (0.0027)  time: 0.9919  data: 0.4327  max
mem: 3506
Epoch: [6]  [10/60]  eta: 0:00:29  lr: 0.000050  loss: 0.1693 (0.1709)
loss_classifier: 0.0187 (0.0200)  loss_box_reg: 0.0339 (0.0332)
loss_mask: 0.1108 (0.1150)  loss_objectness: 0.0003 (0.0004)
loss_rpn_box_reg: 0.0023 (0.0024)  time: 0.5922  data: 0.0479  max
mem: 3506
Epoch: [6]  [20/60]  eta: 0:00:23  lr: 0.000050  loss: 0.1793 (0.1807)
loss_classifier: 0.0242 (0.0241)  loss_box_reg: 0.0339 (0.0363)
loss_mask: 0.1108 (0.1167)  loss_objectness: 0.0004 (0.0007)
loss_rpn_box_reg: 0.0023 (0.0028)  time: 0.5668  data: 0.0090  max
mem: 3506
Epoch: [6]  [30/60]  eta: 0:00:17  lr: 0.000050  loss: 0.1793 (0.1808)
loss_classifier: 0.0242 (0.0245)  loss_box_reg: 0.0327 (0.0368)
loss_mask: 0.1136 (0.1161)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0025 (0.0028)  time: 0.5922  data: 0.0084  max
mem: 3506
Epoch: [6]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1654 (0.1841)
loss_classifier: 0.0206 (0.0245)  loss_box_reg: 0.0300 (0.0384)
loss_mask: 0.1133 (0.1173)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0025 (0.0030)  time: 0.5777  data: 0.0092  max
mem: 3506
Epoch: [6]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1654 (0.1815)
loss_classifier: 0.0191 (0.0243)  loss_box_reg: 0.0300 (0.0377)
loss_mask: 0.1091 (0.1157)  loss_objectness: 0.0003 (0.0010)
loss_rpn_box_reg: 0.0022 (0.0029)  time: 0.5662  data: 0.0090  max
mem: 3780
Epoch: [6]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1691 (0.1797)
loss_classifier: 0.0206 (0.0238)  loss_box_reg: 0.0285 (0.0370)
loss_mask: 0.1148 (0.1152)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0020 (0.0028)  time: 0.5693  data: 0.0076  max
mem: 3780
Epoch: [6] Total time: 0:00:35 (0.5842 s / it)
creating index...
```

```
index created!
Test:  [ 0/50]  eta: 0:00:28  model_time: 0.1869 (0.1869)
evaluator_time: 0.0060 (0.0060)  time: 0.5776  data: 0.3833  max mem:
3780
Test:  [49/50]  eta: 0:00:00  model_time: 0.0973 (0.1077)
evaluator_time: 0.0033 (0.0052)  time: 0.1150  data: 0.0036  max mem:
3780
Test: Total time: 0:00:06 (0.1305 s / it)
Averaged stats: model_time: 0.0973 (0.1077)  evaluator_time: 0.0033
(0.0052)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.808
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.918
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.348
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.704
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.831
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.355
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.855
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.855
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.783
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.873
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.759
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.954
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.318
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
```

```
maxDets=100 ] = 0.528
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.780
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.334
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.814
Epoch: [7]  [ 0/60]  eta: 0:01:28  lr: 0.000050  loss: 0.1616 (0.1616)
loss_classifier: 0.0205 (0.0205)  loss_box_reg: 0.0288 (0.0288)
loss_mask: 0.1102 (0.1102)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0021 (0.0021)  time: 1.4667  data: 0.6278  max
mem: 3780
Epoch: [7]  [10/60]  eta: 0:00:32  lr: 0.000050  loss: 0.1826 (0.1937)
loss_classifier: 0.0241 (0.0265)  loss_box_reg: 0.0363 (0.0411)
loss_mask: 0.1177 (0.1229)  loss_objectness: 0.0004 (0.0007)
loss_rpn_box_reg: 0.0021 (0.0025)  time: 0.6578  data: 0.0641  max
mem: 3780
Epoch: [7]  [20/60]  eta: 0:00:24  lr: 0.000050  loss: 0.1834 (0.1907)
loss_classifier: 0.0241 (0.0258)  loss_box_reg: 0.0402 (0.0413)
loss_mask: 0.1172 (0.1201)  loss_objectness: 0.0004 (0.0007)
loss_rpn_box_reg: 0.0026 (0.0029)  time: 0.5638  data: 0.0084  max
mem: 3780
Epoch: [7]  [30/60]  eta: 0:00:17  lr: 0.000050  loss: 0.1867 (0.1909)
loss_classifier: 0.0240 (0.0260)  loss_box_reg: 0.0443 (0.0416)
loss_mask: 0.1058 (0.1197)  loss_objectness: 0.0004 (0.0006)
loss_rpn_box_reg: 0.0026 (0.0030)  time: 0.5671  data: 0.0102  max
mem: 3780
Epoch: [7]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1575 (0.1866)
loss_classifier: 0.0200 (0.0247)  loss_box_reg: 0.0340 (0.0401)
loss_mask: 0.1044 (0.1184)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0020 (0.0028)  time: 0.5652  data: 0.0095  max
mem: 3780
Epoch: [7]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1498 (0.1816)
loss_classifier: 0.0164 (0.0240)  loss_box_reg: 0.0250 (0.0378)
loss_mask: 0.1044 (0.1164)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0017 (0.0027)  time: 0.5425  data: 0.0088  max
mem: 3780
Epoch: [7]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1522 (0.1790)
loss_classifier: 0.0212 (0.0239)  loss_box_reg: 0.0261 (0.0367)
loss_mask: 0.1036 (0.1150)  loss_objectness: 0.0002 (0.0006)
loss_rpn_box_reg: 0.0024 (0.0028)  time: 0.5749  data: 0.0091  max
```

```
mem: 3780
Epoch: [7] Total time: 0:00:35 (0.5869 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:25  model_time: 0.1866 (0.1866)
evaluator_time: 0.0039 (0.0039)  time: 0.5075  data: 0.3155  max mem:
3780
Test:  [49/50]  eta: 0:00:00  model_time: 0.1186 (0.1148)
evaluator_time: 0.0055 (0.0068)  time: 0.1294  data: 0.0048  max mem:
3780
Test: Total time: 0:00:06 (0.1398 s / it)
Averaged stats: model_time: 0.1186 (0.1148)  evaluator_time: 0.0055
(0.0068)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.810
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.919
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.348
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.700
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.834
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.356
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.856
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.856
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.876
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.761
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.954
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.318
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.531
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.782
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.337
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.802
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.815
Epoch: [8]  [ 0/60]  eta: 0:01:06  lr: 0.000050  loss: 0.1370 (0.1370)
loss_classifier: 0.0163 (0.0163)  loss_box_reg: 0.0187 (0.0187)
loss_mask: 0.0959 (0.0959)  loss_objectness: 0.0004 (0.0004)
loss_rpn_box_reg: 0.0057 (0.0057)  time: 1.1106  data: 0.4501  max
mem: 3780
Epoch: [8]  [10/60]  eta: 0:00:29  lr: 0.000050  loss: 0.1865 (0.2064)
loss_classifier: 0.0221 (0.0261)  loss_box_reg: 0.0353 (0.0489)
loss_mask: 0.1295 (0.1263)  loss_objectness: 0.0002 (0.0008)
loss_rpn_box_reg: 0.0027 (0.0044)  time: 0.5999  data: 0.0469  max
mem: 3780
Epoch: [8]  [20/60]  eta: 0:00:23  lr: 0.000050  loss: 0.1865 (0.1967)
loss_classifier: 0.0244 (0.0263)  loss_box_reg: 0.0408 (0.0450)
loss_mask: 0.1164 (0.1205)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0030 (0.0043)  time: 0.5597  data: 0.0091  max
mem: 3780
Epoch: [8]  [30/60]  eta: 0:00:17  lr: 0.000050  loss: 0.1610 (0.1818)
loss_classifier: 0.0210 (0.0243)  loss_box_reg: 0.0292 (0.0389)
loss_mask: 0.0997 (0.1145)  loss_objectness: 0.0003 (0.0005)
loss_rpn_box_reg: 0.0026 (0.0036)  time: 0.5602  data: 0.0101  max
mem: 3780
Epoch: [8]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1455 (0.1775)
loss_classifier: 0.0183 (0.0239)  loss_box_reg: 0.0238 (0.0373)
loss_mask: 0.0973 (0.1124)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0021 (0.0032)  time: 0.5544  data: 0.0081  max
mem: 3780
Epoch: [8]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1653 (0.1799)
loss_classifier: 0.0229 (0.0248)  loss_box_reg: 0.0294 (0.0371)
loss_mask: 0.1098 (0.1143)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0018 (0.0030)  time: 0.5777  data: 0.0088  max
mem: 3780
Epoch: [8]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1663 (0.1786)
```

```
loss_classifier: 0.0229 (0.0245)  loss_box_reg: 0.0291 (0.0367)
loss_mask: 0.1133 (0.1139)  loss_objectness: 0.0002 (0.0007)
loss_rpn_box_reg: 0.0018 (0.0029)  time: 0.5682  data: 0.0086  max
mem: 3780
Epoch: [8] Total time: 0:00:34 (0.5768 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:42  model_time: 0.3319 (0.3319)
evaluator_time: 0.0050 (0.0050)  time: 0.8421  data: 0.5039  max mem:
3780
Test:  [49/50]  eta: 0:00:00  model_time: 0.0978 (0.1148)
evaluator_time: 0.0033 (0.0058)  time: 0.1142  data: 0.0036  max mem:
3780
Test: Total time: 0:00:07 (0.1426 s / it)
Averaged stats: model_time: 0.0978 (0.1148)  evaluator_time: 0.0033
(0.0058)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.806
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.919
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.348
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.700
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.830
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.354
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.853
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.853
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.872
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.761
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
```

```
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.954
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.318
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.529
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.782
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.337
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.803
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.803
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.816
Epoch: [9]  [ 0/60]  eta: 0:00:53  lr: 0.000005  loss: 0.1116 (0.1116)
loss_classifier: 0.0113 (0.0113)  loss_box_reg: 0.0150 (0.0150)
loss_mask: 0.0852 (0.0852)  loss_objectness: 0.0000 (0.0000)
loss_rpn_box_reg: 0.0002 (0.0002)  time: 0.8908  data: 0.3519  max
mem: 3780
Epoch: [9]  [10/60]  eta: 0:00:29  lr: 0.000005  loss: 0.1478 (0.1777)
loss_classifier: 0.0200 (0.0243)  loss_box_reg: 0.0245 (0.0380)
loss_mask: 0.0980 (0.1113)  loss_objectness: 0.0003 (0.0013)
loss_rpn_box_reg: 0.0020 (0.0028)  time: 0.5810  data: 0.0399  max
mem: 3780
Epoch: [9]  [20/60]  eta: 0:00:23  lr: 0.000005  loss: 0.1664 (0.1761)
loss_classifier: 0.0231 (0.0243)  loss_box_reg: 0.0284 (0.0357)
loss_mask: 0.1047 (0.1119)  loss_objectness: 0.0003 (0.0017)
loss_rpn_box_reg: 0.0020 (0.0025)  time: 0.5605  data: 0.0095  max
mem: 3780
Epoch: [9]  [30/60]  eta: 0:00:17  lr: 0.000005  loss: 0.1671 (0.1812)
loss_classifier: 0.0244 (0.0245)  loss_box_reg: 0.0301 (0.0375)
loss_mask: 0.1119 (0.1150)  loss_objectness: 0.0005 (0.0014)
loss_rpn_box_reg: 0.0025 (0.0029)  time: 0.5652  data: 0.0098  max
mem: 3780
Epoch: [9]  [40/60]  eta: 0:00:11  lr: 0.000005  loss: 0.1601 (0.1795)
loss_classifier: 0.0228 (0.0243)  loss_box_reg: 0.0293 (0.0375)
loss_mask: 0.1107 (0.1135)  loss_objectness: 0.0005 (0.0013)
loss_rpn_box_reg: 0.0029 (0.0029)  time: 0.5756  data: 0.0094  max
mem: 3780
Epoch: [9]  [50/60]  eta: 0:00:05  lr: 0.000005  loss: 0.1601 (0.1808)
loss_classifier: 0.0220 (0.0245)  loss_box_reg: 0.0309 (0.0381)
loss_mask: 0.1089 (0.1141)  loss_objectness: 0.0003 (0.0011)
```

```
loss_rpn_box_reg: 0.0029 (0.0029)  time: 0.5913  data: 0.0095  max
mem: 3780
Epoch: [9]  [59/60]  eta: 0:00:00  lr: 0.000005  loss: 0.1621 (0.1808)
loss_classifier: 0.0219 (0.0247)  loss_box_reg: 0.0326 (0.0377)
loss_mask: 0.1089 (0.1143)  loss_objectness: 0.0002 (0.0011)
loss_rpn_box_reg: 0.0033 (0.0029)  time: 0.5792  data: 0.0088  max
mem: 3780
Epoch: [9] Total time: 0:00:34 (0.5811 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:40  model_time: 0.3076 (0.3076)
evaluator_time: 0.0055 (0.0055)  time: 0.8024  data: 0.4880  max mem:
3780
Test:  [49/50]  eta: 0:00:00  model_time: 0.1000 (0.1120)
evaluator_time: 0.0034 (0.0052)  time: 0.1157  data: 0.0037  max mem:
3780
Test: Total time: 0:00:06 (0.1375 s / it)
Averaged stats: model_time: 0.1000 (0.1120)  evaluator_time: 0.0034
(0.0052)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.806
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.919
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.348
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.700
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.830
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.354
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.853
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.853
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.872
IoU metric: segm
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.761
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.975
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.954
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.313
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.529
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.782
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.337
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.803
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.803
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.467
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.816
That's it!
```

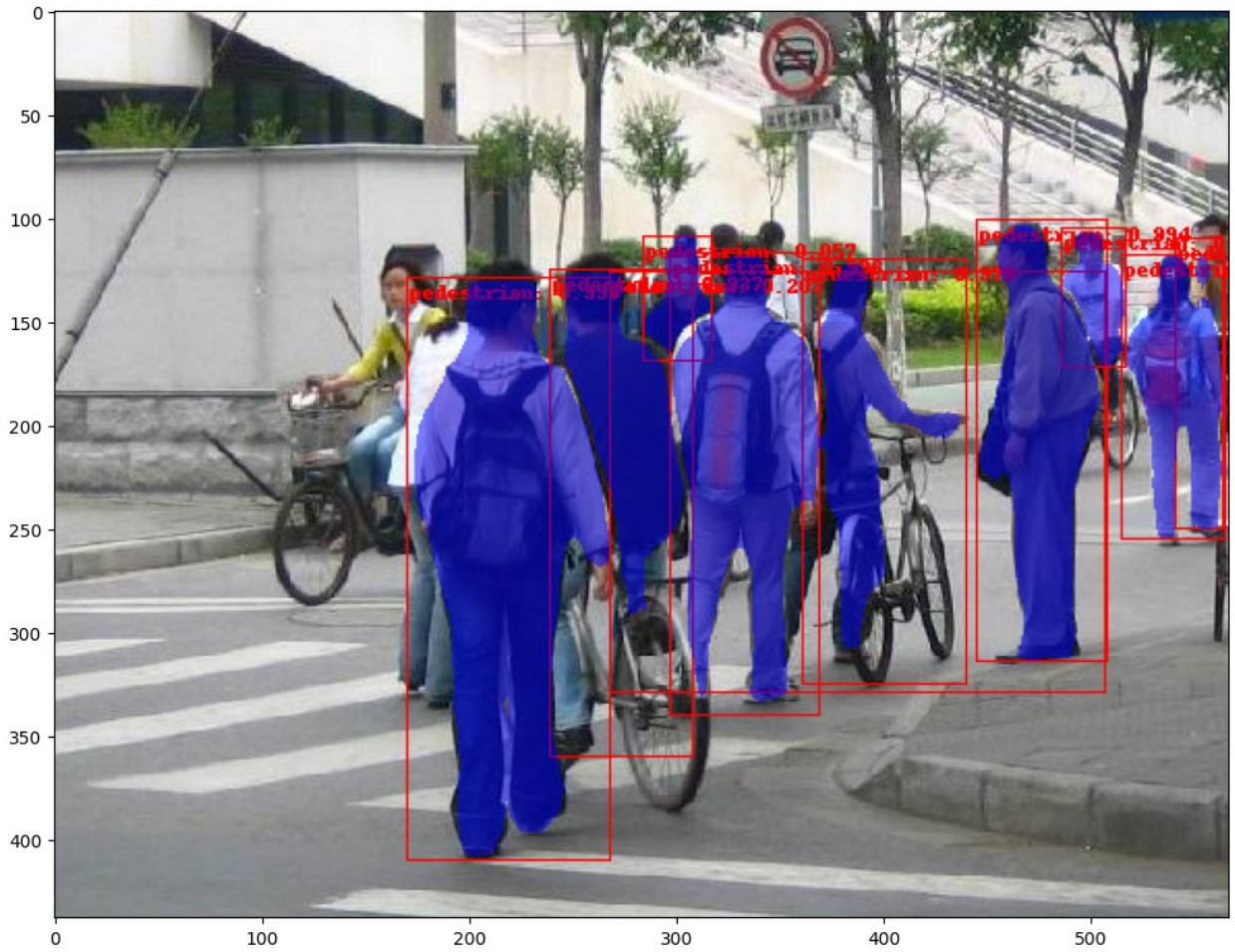
So after one epoch of training, we obtain a COCO-style mAP > 50, and a mask mAP of 65.

But what do the predictions look like? Let's take one image in the dataset and verify

```
import matplotlib.pyplot as plt

from torchvision.utils import draw_bounding_boxes,
draw_segmentation_masks


image =
read_image("drive/MyDrive/_static/img/tv_tutorial/tv_image05.png")
eval_transform = get_transform(train=False)

model.eval()
with torch.no_grad():
    x = eval_transform(image)
    # convert RGBA -> RGB and move to device
    x = x[:3, ...].to(device)
    predictions = model([x, ])
    pred = predictions[0]
```

```
image = (255.0 * (image - image.min()) / (image.max() -
image.min())).to(torch.uint8)
image = image[:3, ...]
pred_labels = [f"pedestrian: {score:.3f}" for label, score in
zip(pred["labels"], pred["scores"])]
pred_boxes = pred["boxes"].long()
output_image = draw_bounding_boxes(image, pred_boxes, pred_labels,
colors="red")

masks = (pred["masks"] > 0.7).squeeze(1)
output_image = draw_segmentation_masks(output_image, masks, alpha=0.5,
colors="blue")


plt.figure(figsize=(12, 12))
plt.imshow(output_image.permute(1, 2, 0))
```

```
<matplotlib.image.AxesImage at 0x7c62d689c3d0>
```



```
pred_labels
```

```
['pedestrian: 0.996',
 'pedestrian: 0.996',
 'pedestrian: 0.994',
 'pedestrian: 0.987',
 'pedestrian: 0.976',
 'pedestrian: 0.937',
 'pedestrian: 0.203',
 'pedestrian: 0.181',
 'pedestrian: 0.119',
 'pedestrian: 0.057']
```

```
pred_boxes
```

```
tensor([[170, 129, 268, 410],
        [297, 117, 369, 340],
        [445, 101, 508, 314],
        [515, 118, 565, 255],
        [361, 120, 440, 325],
        [239, 125, 308, 360],
        [268, 126, 507, 329],
        [486, 105, 517, 172],
        [541, 110, 564, 250],
        [284, 109, 317, 169]], device='cuda:0')
```

The results look good!

Wrapping up

In this tutorial, you have learned how to create your own training pipeline for object detection models on a custom dataset. For that, you wrote a `torch.utils.data.Dataset` class that returns the images and the ground truth boxes and segmentation masks. You also leveraged a Mask R-CNN model pre-trained on COCO train2017 in order to perform transfer learning on this new dataset.

For a more complete example, which includes multi-machine / multi-GPU training, check `references/detection/train.py`, which is present in the torchvision repository.

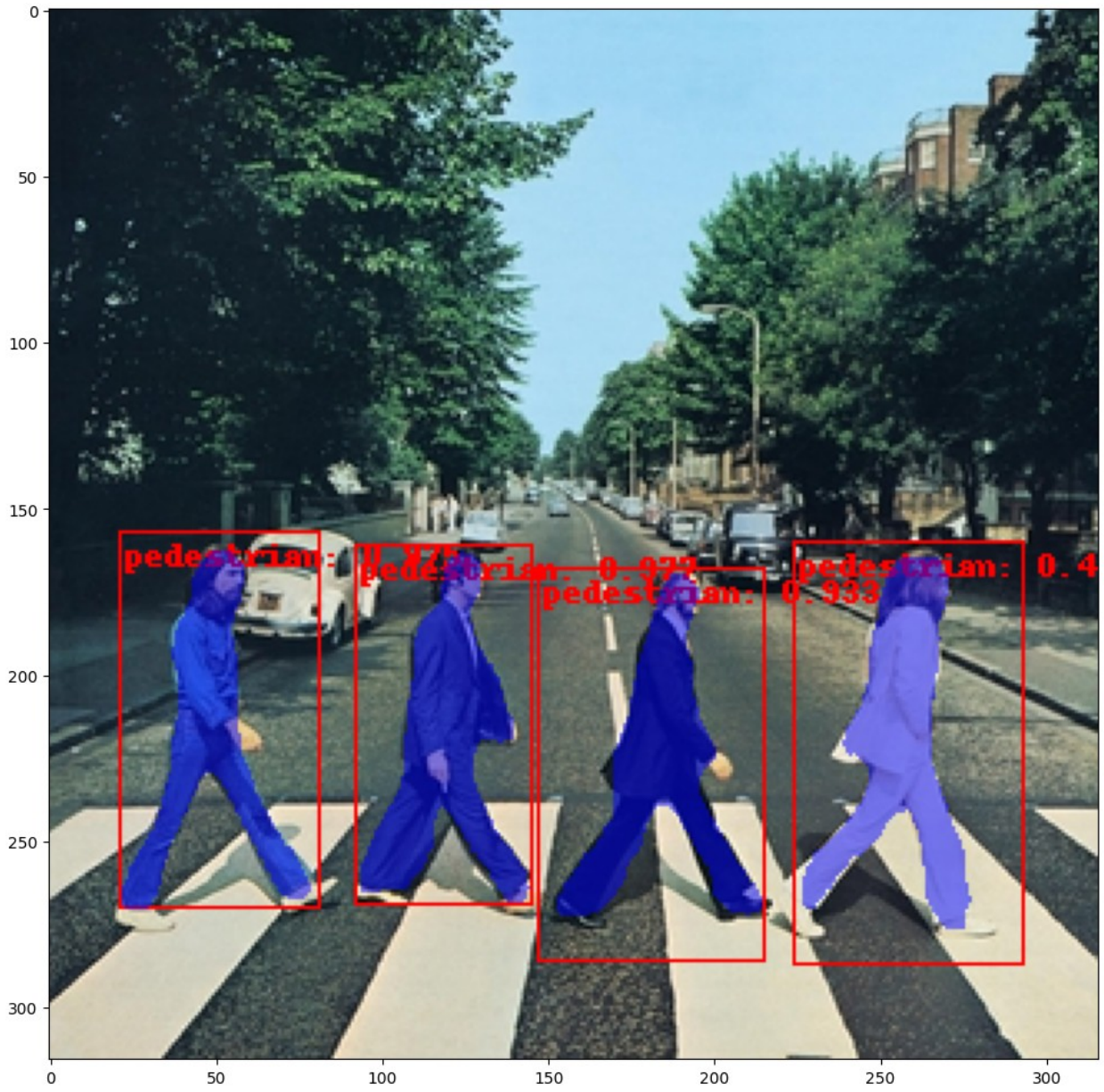
You can download a full source file for this tutorial here_.

```
image = read_image("drive/MyDrive/Beatles_-_Abbey_Road.jpg")
eval_transform = get_transform(train=False)

model.eval()
with torch.no_grad():
    x = eval_transform(image)
    # convert RGBA -> RGB and move to device
    x = x[:3, ...].to(device)
    predictions = model([x, ])
```

```
    pred = predictions[0]


image = (255.0 * (image - image.min()) / (image.max() -
image.min())).to(torch.uint8)
image = image[:3, ...]
pred_labels = [f"pedestrian: {score:.3f}" for label, score in
zip(pred["labels"], pred["scores"])]
pred_boxes = pred["boxes"].long()
output_image = draw_bounding_boxes(image, pred_boxes, pred_labels,
colors="red", font_size=2)

masks = (pred["masks"] > 0.7).squeeze(1)
output_image = draw_segmentation_masks(output_image, masks, alpha=0.5,
colors="blue")


plt.figure(figsize=(12, 12))
plt.imshow(output_image.permute(1, 2, 0))
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/utils.py:223:
UserWarning: Argument 'font_size' will be ignored since 'font' is not
set.
  warnings.warn("Argument 'font_size' will be ignored since 'font' is
not set.")
```

```
<matplotlib.image.AxesImage at 0x7c62d68430a0>
```




```
pred_labels
```

```
['pedestrian: 0.977',
 'pedestrian: 0.975',
 'pedestrian: 0.933',
 'pedestrian: 0.496']
```

```
pred_boxes
```

```
tensor([[ 92, 161, 145, 269],
        [ 21, 157,  81, 270],
        [147, 168, 215, 286],
        [224, 160, 293, 287]], device='cuda:0')
```