This is for Question 5 Part B and C using the finetuning 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.

1 - Finetuning from a pretrained model

Let's suppose that you want to start from a model pre-trained on COCO and want to finetune it for your particular classes. Here is a possible way of doing it:

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

# load a model pre-trained on COCO
model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(weights="DEFAULT"
)

# replace the classifier with a new one, that has
# num_classes which is user-defined
num_classes = 2  # 1 class (person) + background
# 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)
```

```
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 [00:01<00:00, 157MB/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])
```

```
/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(0.0989, grad_fn=<NllLossBackward0>),
'loss_box_reg': tensor(0.0605, grad_fn=<DivBackward0>),
'loss_objectness': tensor(0.0056,
grad_fn=<BinaryCrossEntropyWithLogitsBackward0>), 'loss_rpn_box_reg':
tensor(0.0073, 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, 92.5MB/s]

Epoch: [0]  [ 0/60]  eta: 0:09:22  lr: 0.000090  loss: 4.0232 (4.0232)
loss_classifier: 0.6608 (0.6608)  loss_box_reg: 0.1815 (0.1815)
loss_mask: 3.1586 (3.1586)  loss_objectness: 0.0194 (0.0194)
loss_rpn_box_reg: 0.0029 (0.0029)  time: 9.3771  data: 1.6565  max
mem: 2596
Epoch: [0]  [10/60]  eta: 0:01:07  lr: 0.000936  loss: 1.7460 (2.3481)
loss_classifier: 0.4496 (0.4475)  loss_box_reg: 0.3494 (0.3553)
loss_mask: 1.0930 (1.5102)  loss_objectness: 0.0304 (0.0277)
loss_rpn_box_reg: 0.0061 (0.0074)  time: 1.3599  data: 0.1567  max
mem: 2978
Epoch: [0]  [20/60]  eta: 0:00:39  lr: 0.001783  loss: 1.0728 (1.5656)
loss_classifier: 0.2308 (0.3128)  loss_box_reg: 0.2986 (0.3006)
loss_mask: 0.4108 (0.9232)  loss_objectness: 0.0153 (0.0225)
loss_rpn_box_reg: 0.0058 (0.0066)  time: 0.5580  data: 0.0099  max
mem: 2978
Epoch: [0]  [30/60]  eta: 0:00:25  lr: 0.002629  loss: 0.5781 (1.2421)
loss_classifier: 0.1208 (0.2424)  loss_box_reg: 0.1701 (0.2828)
loss_mask: 0.2195 (0.6921)  loss_objectness: 0.0083 (0.0175)
loss_rpn_box_reg: 0.0041 (0.0072)  time: 0.5665  data: 0.0111  max
mem: 3066
Epoch: [0]  [40/60]  eta: 0:00:15  lr: 0.003476  loss: 0.5432 (1.0732)
loss_classifier: 0.0734 (0.2017)  loss_box_reg: 0.1987 (0.2805)
loss_mask: 0.2021 (0.5695)  loss_objectness: 0.0044 (0.0144)
loss_rpn_box_reg: 0.0059 (0.0072)  time: 0.5562  data: 0.0098  max
mem: 3066
Epoch: [0]  [50/60]  eta: 0:00:07  lr: 0.004323  loss: 0.4499 (0.9406)
loss_classifier: 0.0452 (0.1706)  loss_box_reg: 0.1744 (0.2575)
loss_mask: 0.1770 (0.4932)  loss_objectness: 0.0026 (0.0123)
loss_rpn_box_reg: 0.0052 (0.0070)  time: 0.5357  data: 0.0101  max
mem: 3066
Epoch: [0]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.3485 (0.8478)
```

```
loss_classifier: 0.0368 (0.1514)  loss_box_reg: 0.1203 (0.2363)
loss_mask: 0.1702 (0.4425)  loss_objectness: 0.0015 (0.0108)
loss_rpn_box_reg: 0.0044 (0.0069)  time: 0.5412  data: 0.0090  max
mem: 3066
Epoch: [0] Total time: 0:00:42 (0.7033 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:27  model_time: 0.2204 (0.2204)
evaluator_time: 0.0211 (0.0211)  time: 0.5590  data: 0.3109  max mem:
3066
Test:  [49/50]  eta: 0:00:00  model_time: 0.1112 (0.1196)
evaluator_time: 0.0061 (0.0107)  time: 0.1321  data: 0.0052  max mem:
3066
Test: Total time: 0:00:07 (0.1491 s / it)
Averaged stats: model_time: 0.1112 (0.1196)  evaluator_time: 0.0061
(0.0107)
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 ] = 0.662
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.954
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.849
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.357
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.559
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.680
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.338
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.727
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.727
 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 ] = 0.713
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.736
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.693
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
```

```
maxDets=100 ] = 0.960
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.868
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.369
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.459
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.713
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.353
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.750
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.751
 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 ] = 0.754
Epoch: [1]  [ 0/60]  eta: 0:01:02  lr: 0.005000  loss: 0.3659 (0.3659)
loss_classifier: 0.0570 (0.0570)  loss_box_reg: 0.1407 (0.1407)
loss_mask: 0.1630 (0.1630)  loss_objectness: 0.0016 (0.0016)
loss_rpn_box_reg: 0.0036 (0.0036)  time: 1.0337  data: 0.4756  max
mem: 3066
Epoch: [1]  [10/60]  eta: 0:00:30  lr: 0.005000  loss: 0.3129 (0.3190)
loss_classifier: 0.0385 (0.0399)  loss_box_reg: 0.1143 (0.1172)
loss_mask: 0.1380 (0.1558)  loss_objectness: 0.0005 (0.0009)
loss_rpn_box_reg: 0.0049 (0.0052)  time: 0.6191  data: 0.0496  max
mem: 3066
Epoch: [1]  [20/60]  eta: 0:00:24  lr: 0.005000  loss: 0.2991 (0.3156)
loss_classifier: 0.0334 (0.0396)  loss_box_reg: 0.0931 (0.1061)
loss_mask: 0.1543 (0.1634)  loss_objectness: 0.0007 (0.0016)
loss_rpn_box_reg: 0.0048 (0.0050)  time: 0.5884  data: 0.0081  max
mem: 3066
Epoch: [1]  [30/60]  eta: 0:00:18  lr: 0.005000  loss: 0.2847 (0.3090)
loss_classifier: 0.0421 (0.0408)  loss_box_reg: 0.0901 (0.1063)
loss_mask: 0.1517 (0.1552)  loss_objectness: 0.0015 (0.0017)
loss_rpn_box_reg: 0.0043 (0.0049)  time: 0.5998  data: 0.0097  max
mem: 3464
Epoch: [1]  [40/60]  eta: 0:00:12  lr: 0.005000  loss: 0.2822 (0.3030)
loss_classifier: 0.0421 (0.0400)  loss_box_reg: 0.0887 (0.1028)
loss_mask: 0.1373 (0.1540)  loss_objectness: 0.0007 (0.0014)
loss_rpn_box_reg: 0.0038 (0.0048)  time: 0.6153  data: 0.0090  max
mem: 3464
Epoch: [1]  [50/60]  eta: 0:00:06  lr: 0.005000  loss: 0.2812 (0.3023)
loss_classifier: 0.0341 (0.0400)  loss_box_reg: 0.0887 (0.1020)
loss_mask: 0.1392 (0.1538)  loss_objectness: 0.0005 (0.0014)
```

```
loss_rpn_box_reg: 0.0038 (0.0051)  time: 0.6257  data: 0.0098  max
mem: 3464
Epoch: [1]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.2764 (0.2969)
loss_classifier: 0.0402 (0.0396)  loss_box_reg: 0.0861 (0.0994)
loss_mask: 0.1343 (0.1512)  loss_objectness: 0.0006 (0.0014)
loss_rpn_box_reg: 0.0057 (0.0052)  time: 0.6134  data: 0.0094  max
mem: 3464
Epoch: [1] Total time: 0:00:36 (0.6161 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:39  model_time: 0.2075 (0.2075)
evaluator_time: 0.0281 (0.0281)  time: 0.7880  data: 0.5507  max mem:
3464
Test:  [49/50]  eta: 0:00:00  model_time: 0.0969 (0.1079)
evaluator_time: 0.0032 (0.0060)  time: 0.1103  data: 0.0038  max mem:
3464
Test: Total time: 0:00:06 (0.1350 s / it)
Averaged stats: model_time: 0.0969 (0.1079)  evaluator_time: 0.0032
(0.0060)
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 ] = 0.777
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.966
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.916
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.367
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.581
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.801
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.396
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.827
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.827
 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 ] = 0.725
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.846
IoU metric: segm
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.743
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.971
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.921
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.354
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.532
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.761
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.377
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.787
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.788
 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 ] = 0.750
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.798
Epoch: [2]  [ 0/60]  eta: 0:00:57  lr: 0.005000  loss: 0.2624 (0.2624)
loss_classifier: 0.0328 (0.0328)  loss_box_reg: 0.0768 (0.0768)
loss_mask: 0.1485 (0.1485)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0043 (0.0043)  time: 0.9614  data: 0.3912  max
mem: 3464
Epoch: [2]  [10/60]  eta: 0:00:29  lr: 0.005000  loss: 0.2395 (0.2361)
loss_classifier: 0.0326 (0.0314)  loss_box_reg: 0.0622 (0.0653)
loss_mask: 0.1212 (0.1340)  loss_objectness: 0.0004 (0.0006)
loss_rpn_box_reg: 0.0043 (0.0047)  time: 0.5913  data: 0.0433  max
mem: 3464
Epoch: [2]  [20/60]  eta: 0:00:23  lr: 0.005000  loss: 0.2164 (0.2381)
loss_classifier: 0.0306 (0.0326)  loss_box_reg: 0.0592 (0.0659)
loss_mask: 0.1212 (0.1343)  loss_objectness: 0.0005 (0.0006)
loss_rpn_box_reg: 0.0042 (0.0047)  time: 0.5783  data: 0.0097  max
mem: 3464
Epoch: [2]  [30/60]  eta: 0:00:17  lr: 0.005000  loss: 0.2076 (0.2267)
loss_classifier: 0.0270 (0.0301)  loss_box_reg: 0.0574 (0.0631)
loss_mask: 0.1179 (0.1286)  loss_objectness: 0.0004 (0.0006)
loss_rpn_box_reg: 0.0031 (0.0042)  time: 0.6047  data: 0.0093  max
mem: 3464
Epoch: [2]  [40/60]  eta: 0:00:11  lr: 0.005000  loss: 0.2118 (0.2347)
loss_classifier: 0.0270 (0.0311)  loss_box_reg: 0.0595 (0.0677)
loss_mask: 0.1246 (0.1308)  loss_objectness: 0.0002 (0.0005)
loss_rpn_box_reg: 0.0037 (0.0046)  time: 0.5985  data: 0.0094  max
mem: 3464
```

```
Epoch: [2]  [50/60]  eta: 0:00:05  lr: 0.005000  loss: 0.2745 (0.2435)
loss_classifier: 0.0311 (0.0325)  loss_box_reg: 0.0837 (0.0720)
loss_mask: 0.1364 (0.1328)  loss_objectness: 0.0005 (0.0007)
loss_rpn_box_reg: 0.0054 (0.0054)  time: 0.5857  data: 0.0099  max
mem: 3464
Epoch: [2]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.2684 (0.2462)
loss_classifier: 0.0331 (0.0332)  loss_box_reg: 0.0837 (0.0742)
loss_mask: 0.1263 (0.1328)  loss_objectness: 0.0006 (0.0008)
loss_rpn_box_reg: 0.0035 (0.0052)  time: 0.5700  data: 0.0085  max
mem: 3464
Epoch: [2] Total time: 0:00:35 (0.5971 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:27  model_time: 0.1593 (0.1593)
evaluator_time: 0.0136 (0.0136)  time: 0.5565  data: 0.3816  max mem:
3464
Test:  [49/50]  eta: 0:00:00  model_time: 0.0989 (0.1055)
evaluator_time: 0.0030 (0.0047)  time: 0.1097  data: 0.0037  max mem:
3464
Test: Total time: 0:00:06 (0.1273 s / it)
Averaged stats: model_time: 0.0989 (0.1055)  evaluator_time: 0.0030
(0.0047)
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 ] = 0.746
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.981
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.919
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.352
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.579
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.760
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.385
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.788
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.788
 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 ] = 0.798
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.739
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.890
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.394
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.495
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.751
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.376
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.781
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.783
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.667
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.725
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.792
Epoch: [3]  [ 0/60]  eta: 0:00:59  lr: 0.000500  loss: 0.2168 (0.2168)
loss_classifier: 0.0278 (0.0278)  loss_box_reg: 0.0707 (0.0707)
loss_mask: 0.1120 (0.1120)  loss_objectness: 0.0021 (0.0021)
loss_rpn_box_reg: 0.0043 (0.0043)  time: 0.9965  data: 0.3444  max
mem: 3464
Epoch: [3]  [10/60]  eta: 0:00:30  lr: 0.000500  loss: 0.2418 (0.2369)
loss_classifier: 0.0279 (0.0308)  loss_box_reg: 0.0656 (0.0665)
loss_mask: 0.1386 (0.1345)  loss_objectness: 0.0007 (0.0010)
loss_rpn_box_reg: 0.0036 (0.0041)  time: 0.6190  data: 0.0402  max
mem: 3464
Epoch: [3]  [20/60]  eta: 0:00:25  lr: 0.000500  loss: 0.2288 (0.2314)
loss_classifier: 0.0299 (0.0303)  loss_box_reg: 0.0602 (0.0632)
loss_mask: 0.1326 (0.1333)  loss_objectness: 0.0005 (0.0009)
loss_rpn_box_reg: 0.0032 (0.0037)  time: 0.6124  data: 0.0094  max
mem: 3464
Epoch: [3]  [30/60]  eta: 0:00:18  lr: 0.000500  loss: 0.2084 (0.2180)
loss_classifier: 0.0284 (0.0304)  loss_box_reg: 0.0457 (0.0558)
loss_mask: 0.1106 (0.1270)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0030 (0.0038)  time: 0.6178  data: 0.0095  max
mem: 3464
Epoch: [3]  [40/60]  eta: 0:00:12  lr: 0.000500  loss: 0.1870 (0.2101)
loss_classifier: 0.0254 (0.0296)  loss_box_reg: 0.0372 (0.0535)
```

```
loss_mask: 0.1054 (0.1226)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0022 (0.0035)  time: 0.5934  data: 0.0098  max
mem: 3464
Epoch: [3]  [50/60]  eta: 0:00:06  lr: 0.000500  loss: 0.1886 (0.2074)
loss_classifier: 0.0208 (0.0285)  loss_box_reg: 0.0435 (0.0524)
loss_mask: 0.1120 (0.1220)  loss_objectness: 0.0004 (0.0010)
loss_rpn_box_reg: 0.0020 (0.0035)  time: 0.6067  data: 0.0106  max
mem: 3464
Epoch: [3]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1886 (0.2092)
loss_classifier: 0.0252 (0.0288)  loss_box_reg: 0.0435 (0.0524)
loss_mask: 0.1151 (0.1234)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0031 (0.0036)  time: 0.6042  data: 0.0099  max
mem: 3464
Epoch: [3] Total time: 0:00:36 (0.6153 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:28  model_time: 0.1925 (0.1925)
evaluator_time: 0.0142 (0.0142)  time: 0.5605  data: 0.3520  max mem:
3464
Test:  [49/50]  eta: 0:00:00  model_time: 0.1048 (0.1132)
evaluator_time: 0.0047 (0.0065)  time: 0.1237  data: 0.0090  max mem:
3464
Test: Total time: 0:00:07 (0.1403 s / it)
Averaged stats: model_time: 0.1048 (0.1132)  evaluator_time: 0.0047
(0.0065)
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 ] = 0.828
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.981
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.950
 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 ] = 0.650
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.846
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.419
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.866
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.866
 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 ] = 0.825
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.880
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 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.919
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.378
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.517
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.772
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.382
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.797
 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 ] = 0.567
 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 ] = 0.812
Epoch: [4]  [ 0/60]  eta: 0:01:03  lr: 0.000500  loss: 0.1764 (0.1764)
loss_classifier: 0.0218 (0.0218)  loss_box_reg: 0.0406 (0.0406)
loss_mask: 0.1091 (0.1091)  loss_objectness: 0.0004 (0.0004)
loss_rpn_box_reg: 0.0045 (0.0045)  time: 1.0649  data: 0.3697  max
mem: 3464
Epoch: [4]  [10/60]  eta: 0:00:31  lr: 0.000500  loss: 0.2057 (0.2016)
loss_classifier: 0.0229 (0.0288)  loss_box_reg: 0.0416 (0.0444)
loss_mask: 0.1205 (0.1236)  loss_objectness: 0.0006 (0.0006)
loss_rpn_box_reg: 0.0044 (0.0043)  time: 0.6200  data: 0.0414  max
mem: 3464
Epoch: [4]  [20/60]  eta: 0:00:25  lr: 0.000500  loss: 0.2057 (0.2106)
loss_classifier: 0.0295 (0.0301)  loss_box_reg: 0.0496 (0.0495)
loss_mask: 0.1205 (0.1261)  loss_objectness: 0.0007 (0.0008)
loss_rpn_box_reg: 0.0037 (0.0041)  time: 0.6055  data: 0.0097  max
mem: 3700
Epoch: [4]  [30/60]  eta: 0:00:18  lr: 0.000500  loss: 0.1907 (0.2052)
loss_classifier: 0.0274 (0.0292)  loss_box_reg: 0.0480 (0.0478)
loss_mask: 0.1135 (0.1238)  loss_objectness: 0.0005 (0.0007)
loss_rpn_box_reg: 0.0029 (0.0036)  time: 0.6023  data: 0.0097  max
```

```
mem: 3700
Epoch: [4]  [40/60]  eta: 0:00:12  lr: 0.000500  loss: 0.1797 (0.2047)
loss_classifier: 0.0246 (0.0302)  loss_box_reg: 0.0437 (0.0480)
loss_mask: 0.1135 (0.1223)  loss_objectness: 0.0005 (0.0007)
loss_rpn_box_reg: 0.0027 (0.0034)  time: 0.5845  data: 0.0109  max
mem: 3700
Epoch: [4]  [50/60]  eta: 0:00:06  lr: 0.000500  loss: 0.1739 (0.1996)
loss_classifier: 0.0253 (0.0290)  loss_box_reg: 0.0385 (0.0468)
loss_mask: 0.1050 (0.1199)  loss_objectness: 0.0005 (0.0008)
loss_rpn_box_reg: 0.0023 (0.0032)  time: 0.5931  data: 0.0120  max
mem: 3700
Epoch: [4]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1693 (0.1997)
loss_classifier: 0.0275 (0.0292)  loss_box_reg: 0.0385 (0.0467)
loss_mask: 0.1055 (0.1197)  loss_objectness: 0.0005 (0.0009)
loss_rpn_box_reg: 0.0023 (0.0032)  time: 0.6095  data: 0.0095  max
mem: 3700
Epoch: [4] Total time: 0:00:36 (0.6121 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:26  model_time: 0.1775 (0.1775)
evaluator_time: 0.0206 (0.0206)  time: 0.5346  data: 0.3344  max mem:
3700
Test:  [49/50]  eta: 0:00:00  model_time: 0.0987 (0.1066)
evaluator_time: 0.0026 (0.0048)  time: 0.1097  data: 0.0037  max mem:
3700
Test: Total time: 0:00:06 (0.1277 s / it)
Averaged stats: model_time: 0.0987 (0.1066)  evaluator_time: 0.0026
(0.0048)
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 ] = 0.840
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.981
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.950
 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 ] = 0.674
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.859
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.426
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.881
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.881
 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 ] = 0.825
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.897
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.757
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.913
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.378
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.530
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.770
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.385
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.799
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.801
 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 ] = 0.810
Epoch: [5]  [ 0/60]  eta: 0:00:59  lr: 0.000500  loss: 0.2363 (0.2363)
loss_classifier: 0.0346 (0.0346)  loss_box_reg: 0.0638 (0.0638)
loss_mask: 0.1342 (0.1342)  loss_objectness: 0.0002 (0.0002)
loss_rpn_box_reg: 0.0035 (0.0035)  time: 0.9918  data: 0.4285  max
mem: 3700
Epoch: [5]  [10/60]  eta: 0:00:30  lr: 0.000500  loss: 0.1874 (0.1939)
loss_classifier: 0.0242 (0.0280)  loss_box_reg: 0.0449 (0.0424)
loss_mask: 0.1151 (0.1199)  loss_objectness: 0.0004 (0.0011)
loss_rpn_box_reg: 0.0021 (0.0025)  time: 0.6078  data: 0.0489  max
mem: 3700
Epoch: [5]  [20/60]  eta: 0:00:24  lr: 0.000500  loss: 0.1931 (0.2105)
loss_classifier: 0.0356 (0.0326)  loss_box_reg: 0.0477 (0.0501)
loss_mask: 0.1145 (0.1234)  loss_objectness: 0.0004 (0.0014)
loss_rpn_box_reg: 0.0021 (0.0030)  time: 0.5918  data: 0.0103  max
mem: 3700
Epoch: [5]  [30/60]  eta: 0:00:17  lr: 0.000500  loss: 0.1931 (0.2008)
loss_classifier: 0.0284 (0.0289)  loss_box_reg: 0.0477 (0.0472)
```

```
loss_mask: 0.1142 (0.1206)  loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0028 (0.0030)  time: 0.5921  data: 0.0098  max
mem: 3700
Epoch: [5]  [40/60]  eta: 0:00:11  lr: 0.000500  loss: 0.1741 (0.1982)
loss_classifier: 0.0214 (0.0286)  loss_box_reg: 0.0373 (0.0459)
loss_mask: 0.1122 (0.1197)  loss_objectness: 0.0003 (0.0010)
loss_rpn_box_reg: 0.0026 (0.0030)  time: 0.5814  data: 0.0102  max
mem: 3700
Epoch: [5]  [50/60]  eta: 0:00:05  lr: 0.000500  loss: 0.1935 (0.1971)
loss_classifier: 0.0248 (0.0283)  loss_box_reg: 0.0424 (0.0465)
loss_mask: 0.1114 (0.1183)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0034 (0.0031)  time: 0.6012  data: 0.0099  max
mem: 3700
Epoch: [5]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1741 (0.1930)
loss_classifier: 0.0218 (0.0271)  loss_box_reg: 0.0325 (0.0444)
loss_mask: 0.1061 (0.1175)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0026 (0.0031)  time: 0.6125  data: 0.0085  max
mem: 3700
Epoch: [5] Total time: 0:00:36 (0.6051 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:28  model_time: 0.1591 (0.1591)
evaluator_time: 0.0116 (0.0116)  time: 0.5638  data: 0.3914  max mem:
3700
Test:  [49/50]  eta: 0:00:00  model_time: 0.1025 (0.1087)
evaluator_time: 0.0039 (0.0052)  time: 0.1167  data: 0.0041  max mem:
3700
Test: Total time: 0:00:06 (0.1351 s / it)
Averaged stats: model_time: 0.1025 (0.1087)  evaluator_time: 0.0039
(0.0052)
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 ] = 0.837
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.979
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.930
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.388
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.629
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.858
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.423
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.875
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.875
 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 ] = 0.812
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.892
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.762
 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 ] = 0.403
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.544
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.385
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.804
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.806
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.633
 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 ] = 0.813
Epoch: [6]  [ 0/60]  eta: 0:01:24  lr: 0.000050  loss: 0.2115 (0.2115)
loss_classifier: 0.0332 (0.0332)  loss_box_reg: 0.0507 (0.0507)
loss_mask: 0.1236 (0.1236)  loss_objectness: 0.0005 (0.0005)
loss_rpn_box_reg: 0.0036 (0.0036)  time: 1.4092  data: 0.6883  max
mem: 3700
Epoch: [6]  [10/60]  eta: 0:00:33  lr: 0.000050  loss: 0.2112 (0.1921)
loss_classifier: 0.0329 (0.0284)  loss_box_reg: 0.0381 (0.0406)
loss_mask: 0.1210 (0.1200)  loss_objectness: 0.0002 (0.0004)
loss_rpn_box_reg: 0.0027 (0.0027)  time: 0.6647  data: 0.0694  max
mem: 3700
Epoch: [6]  [20/60]  eta: 0:00:25  lr: 0.000050  loss: 0.1861 (0.1938)
loss_classifier: 0.0253 (0.0281)  loss_box_reg: 0.0354 (0.0426)
loss_mask: 0.1169 (0.1195)  loss_objectness: 0.0003 (0.0005)
loss_rpn_box_reg: 0.0027 (0.0032)  time: 0.6113  data: 0.0086  max
```

```
mem: 3700
Epoch: [6]  [30/60]  eta: 0:00:18  lr: 0.000050  loss: 0.1861 (0.1884)
loss_classifier: 0.0226 (0.0266)  loss_box_reg: 0.0354 (0.0416)
loss_mask: 0.1085 (0.1164)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0024 (0.0031)  time: 0.6113  data: 0.0095  max
mem: 3700
Epoch: [6]  [40/60]  eta: 0:00:12  lr: 0.000050  loss: 0.1644 (0.1829)
loss_classifier: 0.0184 (0.0256)  loss_box_reg: 0.0317 (0.0396)
loss_mask: 0.1084 (0.1143)  loss_objectness: 0.0002 (0.0005)
loss_rpn_box_reg: 0.0022 (0.0029)  time: 0.5842  data: 0.0091  max
mem: 3700
Epoch: [6]  [50/60]  eta: 0:00:06  lr: 0.000050  loss: 0.1649 (0.1882)
loss_classifier: 0.0205 (0.0266)  loss_box_reg: 0.0322 (0.0410)
loss_mask: 0.1118 (0.1170)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0018 (0.0029)  time: 0.5804  data: 0.0091  max
mem: 3700
Epoch: [6]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1811 (0.1904)
loss_classifier: 0.0275 (0.0268)  loss_box_reg: 0.0370 (0.0415)
loss_mask: 0.1207 (0.1183)  loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0027 (0.0030)  time: 0.5921  data: 0.0083  max
mem: 3700
Epoch: [6] Total time: 0:00:36 (0.6128 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:40  model_time: 0.1714 (0.1714)
evaluator_time: 0.0194 (0.0194)  time: 0.8045  data: 0.6114  max mem:
3700
Test:  [49/50]  eta: 0:00:00  model_time: 0.1001 (0.1105)
evaluator_time: 0.0025 (0.0057)  time: 0.1101  data: 0.0037  max mem:
3700
Test: Total time: 0:00:07 (0.1411 s / it)
Averaged stats: model_time: 0.1001 (0.1105)  evaluator_time: 0.0025
(0.0057)
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 ] = 0.836
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.979
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.941
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.355
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.655
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
```

```
maxDets=100 ] = 0.858
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.420
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.874
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.874
 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 ] = 0.812
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.892
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.763
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.917
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.403
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.540
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.386
 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 ] = 0.805
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.633
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.787
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.811
Epoch: [7]  [ 0/60]  eta: 0:00:58  lr: 0.000050  loss: 0.1546 (0.1546)
loss_classifier: 0.0144 (0.0144)  loss_box_reg: 0.0233 (0.0233)
loss_mask: 0.1152 (0.1152)  loss_objectness: 0.0002 (0.0002)
loss_rpn_box_reg: 0.0015 (0.0015)  time: 0.9699  data: 0.3689  max
mem: 3700
Epoch: [7]  [10/60]  eta: 0:00:31  lr: 0.000050  loss: 0.1793 (0.1782)
loss_classifier: 0.0260 (0.0260)  loss_box_reg: 0.0386 (0.0390)
loss_mask: 0.1070 (0.1103)  loss_objectness: 0.0002 (0.0003)
loss_rpn_box_reg: 0.0030 (0.0026)  time: 0.6248  data: 0.0409  max
mem: 3700
Epoch: [7]  [20/60]  eta: 0:00:24  lr: 0.000050  loss: 0.1793 (0.1823)
```

```
loss_classifier: 0.0254 (0.0261)  loss_box_reg: 0.0397 (0.0411)
loss_mask: 0.1049 (0.1116)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0030 (0.0029)  time: 0.6018  data: 0.0095  max
mem: 3700
Epoch: [7]  [30/60]  eta: 0:00:17  lr: 0.000050  loss: 0.1716 (0.1806)
loss_classifier: 0.0230 (0.0257)  loss_box_reg: 0.0397 (0.0396)
loss_mask: 0.1068 (0.1118)  loss_objectness: 0.0003 (0.0005)
loss_rpn_box_reg: 0.0027 (0.0029)  time: 0.5863  data: 0.0099  max
mem: 3700
Epoch: [7]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1733 (0.1817)
loss_classifier: 0.0228 (0.0256)  loss_box_reg: 0.0364 (0.0394)
loss_mask: 0.1077 (0.1133)  loss_objectness: 0.0003 (0.0005)
loss_rpn_box_reg: 0.0028 (0.0030)  time: 0.5703  data: 0.0087  max
mem: 3700
Epoch: [7]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1733 (0.1853)
loss_classifier: 0.0205 (0.0257)  loss_box_reg: 0.0364 (0.0405)
loss_mask: 0.1171 (0.1155)  loss_objectness: 0.0004 (0.0006)
loss_rpn_box_reg: 0.0031 (0.0030)  time: 0.5830  data: 0.0088  max
mem: 3700
Epoch: [7]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1799 (0.1869)
loss_classifier: 0.0235 (0.0261)  loss_box_reg: 0.0418 (0.0411)
loss_mask: 0.1171 (0.1161)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0028 (0.0030)  time: 0.5975  data: 0.0084  max
mem: 3700
Epoch: [7] Total time: 0:00:36 (0.6016 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:27  model_time: 0.1995 (0.1995)
evaluator_time: 0.0102 (0.0102)  time: 0.5437  data: 0.3323  max mem:
3700
Test:  [49/50]  eta: 0:00:00  model_time: 0.0981 (0.1083)
evaluator_time: 0.0026 (0.0047)  time: 0.1100  data: 0.0036  max mem:
3700
Test: Total time: 0:00:06 (0.1294 s / it)
Averaged stats: model_time: 0.0981 (0.1083)  evaluator_time: 0.0026
(0.0047)
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 ] = 0.839
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.979
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.941
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.355
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.659
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.860
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.424
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.879
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.879
 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 ] = 0.825
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.896
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.763
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.918
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.403
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.534
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.775
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.386
 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 ] = 0.805
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.633
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.787
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.811
Epoch: [8]  [ 0/60]  eta: 0:01:05  lr: 0.000050  loss: 0.1549 (0.1549)
loss_classifier: 0.0242 (0.0242)  loss_box_reg: 0.0308 (0.0308)
loss_mask: 0.0977 (0.0977)  loss_objectness: 0.0003 (0.0003)
loss_rpn_box_reg: 0.0020 (0.0020)  time: 1.0998  data: 0.4827  max
mem: 3700
Epoch: [8]  [10/60]  eta: 0:00:29  lr: 0.000050  loss: 0.1596 (0.1569)
loss_classifier: 0.0237 (0.0219)  loss_box_reg: 0.0263 (0.0260)
loss_mask: 0.1095 (0.1062)  loss_objectness: 0.0003 (0.0005)
```

```
loss_rpn_box_reg: 0.0020 (0.0022)  time: 0.5801  data: 0.0547  max
mem: 3700
Epoch: [8]  [20/60]  eta: 0:00:23  lr: 0.000050  loss: 0.1797 (0.1906)
loss_classifier: 0.0262 (0.0277)  loss_box_reg: 0.0315 (0.0408)
loss_mask: 0.1151 (0.1184)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0025 (0.0029)  time: 0.5693  data: 0.0113  max
mem: 3700
Epoch: [8]  [30/60]  eta: 0:00:18  lr: 0.000050  loss: 0.1824 (0.1826)
loss_classifier: 0.0262 (0.0254)  loss_box_reg: 0.0394 (0.0391)
loss_mask: 0.1151 (0.1145)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0028 (0.0029)  time: 0.6157  data: 0.0108  max
mem: 3700
Epoch: [8]  [40/60]  eta: 0:00:12  lr: 0.000050  loss: 0.1793 (0.1847)
loss_classifier: 0.0236 (0.0260)  loss_box_reg: 0.0394 (0.0401)
loss_mask: 0.1075 (0.1150)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0025 (0.0029)  time: 0.6154  data: 0.0103  max
mem: 3700
Epoch: [8]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1794 (0.1860)
loss_classifier: 0.0266 (0.0254)  loss_box_reg: 0.0393 (0.0395)
loss_mask: 0.1112 (0.1176)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0025 (0.0029)  time: 0.5890  data: 0.0097  max
mem: 3700
Epoch: [8]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1794 (0.1867)
loss_classifier: 0.0276 (0.0260)  loss_box_reg: 0.0365 (0.0408)
loss_mask: 0.1053 (0.1166)  loss_objectness: 0.0002 (0.0006)
loss_rpn_box_reg: 0.0026 (0.0028)  time: 0.5954  data: 0.0087  max
mem: 3700
Epoch: [8] Total time: 0:00:36 (0.6055 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:28  model_time: 0.1800 (0.1800)
evaluator_time: 0.0126 (0.0126)  time: 0.5767  data: 0.3823  max mem:
3700
Test:  [49/50]  eta: 0:00:00  model_time: 0.1195 (0.1192)
evaluator_time: 0.0047 (0.0068)  time: 0.1389  data: 0.0058  max mem:
3700
Test: Total time: 0:00:07 (0.1499 s / it)
Averaged stats: model_time: 0.1195 (0.1192)  evaluator_time: 0.0047
(0.0068)
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 ] = 0.840
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.979
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
```

```
maxDets=100 ] = 0.931
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.355
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.659
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.861
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.425
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.878
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.878
 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 ] = 0.825
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.895
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.765
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.918
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.403
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.535
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.777
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.385
 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 ] = 0.805
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.633
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.787
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.811
Epoch: [9]  [ 0/60]  eta: 0:01:36  lr: 0.000005  loss: 0.1179 (0.1179)
loss_classifier: 0.0076 (0.0076)  loss_box_reg: 0.0123 (0.0123)
loss_mask: 0.0972 (0.0972)  loss_objectness: 0.0002 (0.0002)
loss_rpn_box_reg: 0.0006 (0.0006)  time: 1.6021  data: 0.8491  max
mem: 3700
```

```
Epoch: [9]  [10/60]  eta: 0:00:34  lr: 0.000005  loss: 0.1728 (0.1847)
loss_classifier: 0.0242 (0.0255)  loss_box_reg: 0.0357 (0.0386)
loss_mask: 0.1076 (0.1176)  loss_objectness: 0.0002 (0.0007)
loss_rpn_box_reg: 0.0020 (0.0023)  time: 0.6896  data: 0.0844  max
mem: 3700
Epoch: [9]  [20/60]  eta: 0:00:26  lr: 0.000005  loss: 0.1840 (0.1879)
loss_classifier: 0.0292 (0.0284)  loss_box_reg: 0.0365 (0.0414)
loss_mask: 0.1125 (0.1150)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0026 (0.0025)  time: 0.6100  data: 0.0099  max
mem: 3700
Epoch: [9]  [30/60]  eta: 0:00:19  lr: 0.000005  loss: 0.1834 (0.1829)
loss_classifier: 0.0286 (0.0269)  loss_box_reg: 0.0356 (0.0394)
loss_mask: 0.1061 (0.1132)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0031 (0.0026)  time: 0.6119  data: 0.0100  max
mem: 3778
Epoch: [9]  [40/60]  eta: 0:00:12  lr: 0.000005  loss: 0.1670 (0.1854)
loss_classifier: 0.0234 (0.0268)  loss_box_reg: 0.0321 (0.0401)
loss_mask: 0.1090 (0.1150)  loss_objectness: 0.0002 (0.0006)
loss_rpn_box_reg: 0.0032 (0.0028)  time: 0.5948  data: 0.0085  max
mem: 3778
Epoch: [9]  [50/60]  eta: 0:00:06  lr: 0.000005  loss: 0.1898 (0.1883)
loss_classifier: 0.0244 (0.0267)  loss_box_reg: 0.0388 (0.0407)
loss_mask: 0.1180 (0.1174)  loss_objectness: 0.0002 (0.0006)
loss_rpn_box_reg: 0.0032 (0.0029)  time: 0.5872  data: 0.0101  max
mem: 3778
Epoch: [9]  [59/60]  eta: 0:00:00  lr: 0.000005  loss: 0.1793 (0.1891)
loss_classifier: 0.0253 (0.0269)  loss_box_reg: 0.0378 (0.0417)
loss_mask: 0.1127 (0.1169)  loss_objectness: 0.0004 (0.0006)
loss_rpn_box_reg: 0.0029 (0.0030)  time: 0.5977  data: 0.0096  max
mem: 3778
Epoch: [9] Total time: 0:00:37 (0.6213 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:42  model_time: 0.2411 (0.2411)
evaluator_time: 0.0180 (0.0180)  time: 0.8438  data: 0.5829  max mem:
3778
Test:  [49/50]  eta: 0:00:00  model_time: 0.0998 (0.1116)
evaluator_time: 0.0029 (0.0051)  time: 0.1125  data: 0.0038  max mem:
3778
Test: Total time: 0:00:06 (0.1391 s / it)
Averaged stats: model_time: 0.0998 (0.1116)  evaluator_time: 0.0029
(0.0051)
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 ] = 0.839
```

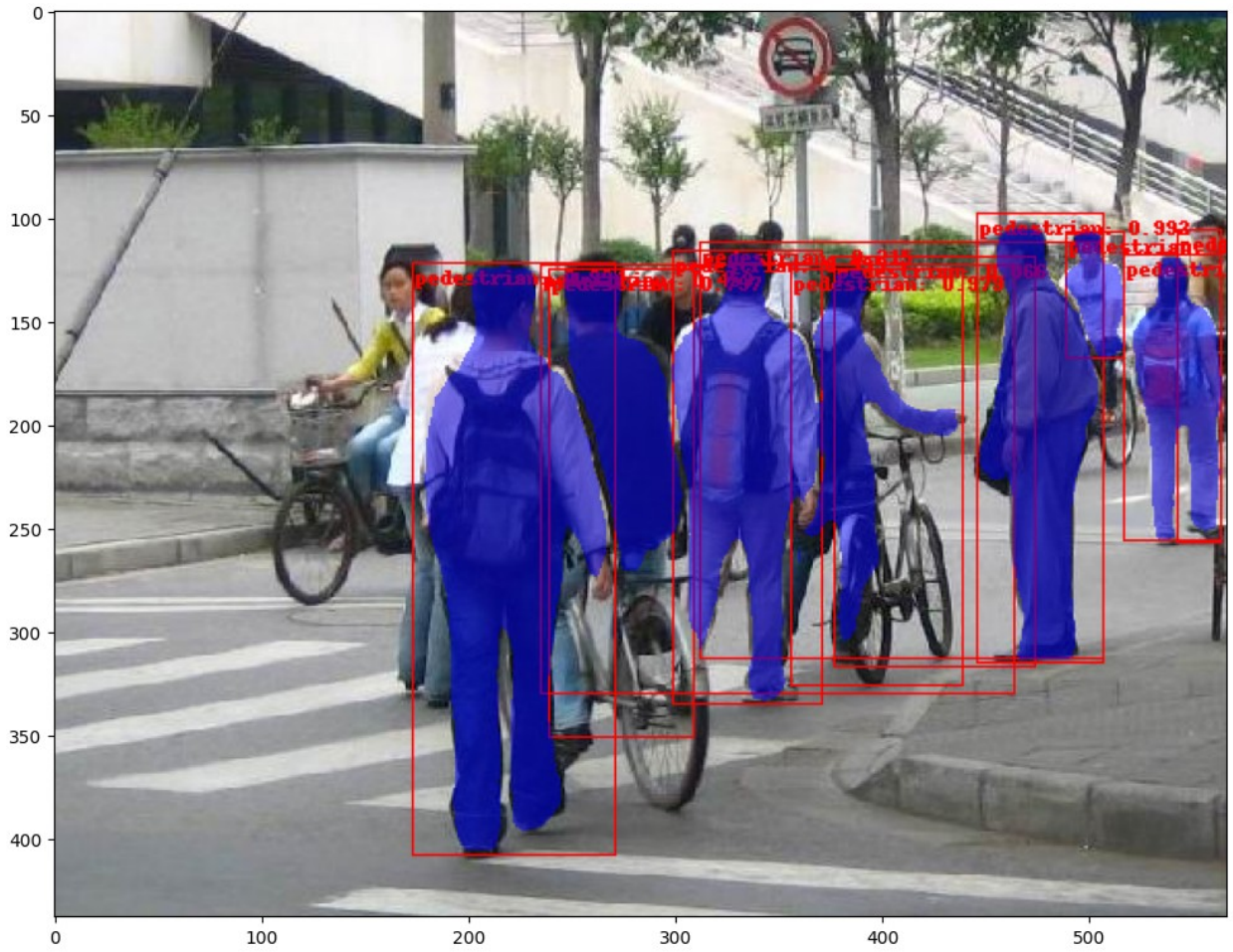
```
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.979
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.931
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.355
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.659
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.860
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.424
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.877
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.877
 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 ] = 0.825
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.894
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.767
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.918
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.403
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.535
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.779
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.386
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.804
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.806
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.633
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.787
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.812
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 0x7ef574e209a0>
```




```
pred_labels
```

```
['pedestrian: 0.995',
 'pedestrian: 0.994',
 'pedestrian: 0.992',
 'pedestrian: 0.988',
 'pedestrian: 0.979',
 'pedestrian: 0.797',
 'pedestrian: 0.453',
 'pedestrian: 0.215',
 'pedestrian: 0.089',
 'pedestrian: 0.066',
 'pedestrian: 0.055']
```

```
pred_boxes
```

```
tensor([[299, 116, 371, 335],
        [173, 122, 271, 408],
        [446,  98, 507, 315],
        [517, 118, 564, 256],
        [356, 125, 439, 326],
```

```
        [239, 125, 309, 351],
        [235, 123, 464, 330],
        [312, 112, 507, 313],
        [489, 107, 517, 168],
        [377, 119, 474, 317],
        [543, 106, 564, 257]], device='cuda:0')
```

The results look good!

Wrapping up

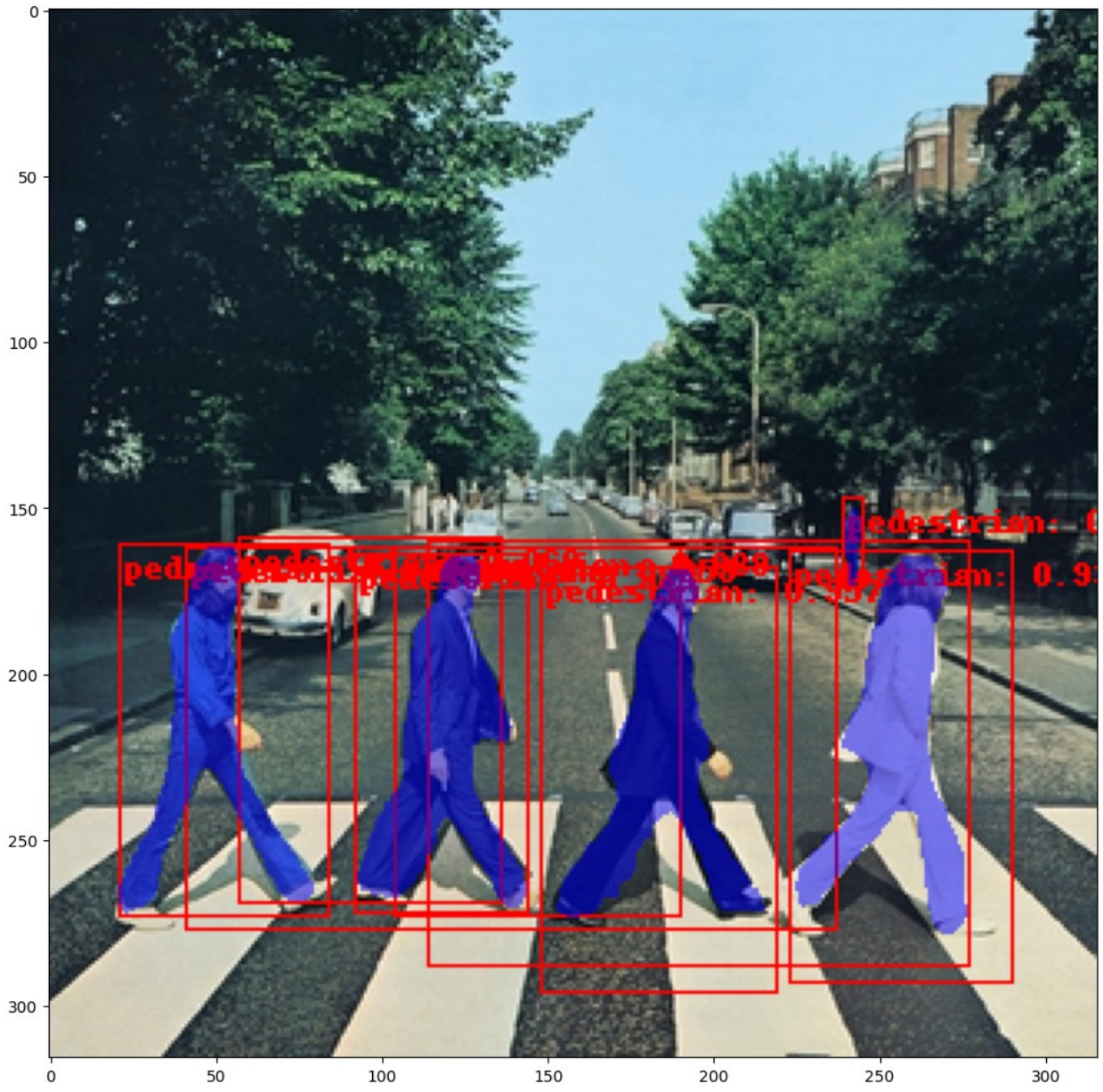
In this tutorial, you have learned how to create your own training pipeline for object detection models on a custom dataset. For that, you wrote a `torch.utils.data.Dataset` class that returns the images and the ground truth boxes and segmentation masks. You also leveraged a Mask R-CNN model pre-trained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

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 0x7ef4a2a48cd0>
```




```
pred_labels
```

```
['pedestrian: 0.987',
 'pedestrian: 0.985',
 'pedestrian: 0.957',
 'pedestrian: 0.939',
 'pedestrian: 0.103',
 'pedestrian: 0.093',
 'pedestrian: 0.089',
```

```
 'pedestrian: 0.060',
 'pedestrian: 0.059']

pred_boxes

tensor([[ 21, 161,  84, 273],
        [ 92, 164, 144, 272],
        [148, 168, 219, 296],
        [223, 163, 290, 293],
        [ 41, 162, 237, 277],
        [239, 147, 245, 174],
        [114, 160, 277, 288],
        [ 57, 159, 136, 269],
        [104, 162, 190, 273]], device='cuda:0')
```