Deep Learning Assignment 2

Name: Santosh Srinivas Ravichandran

Net ID: sr6411

1.

a)

$$L = 0.5 * (a * w_1^2 + b * w_2^2)$$

$$\frac{\partial L}{\partial w_1} = 0.5 * a * 2 * w_1 = a * w_1$$

$$\frac{\partial L}{\partial w_2} = 0.5 * b * 2 * w_2 = b * w_2$$

$$\nabla L(w) = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \end{bmatrix}$$

$$\nabla L(w) = \begin{bmatrix} a * w_1 \\ b * w_2 \end{bmatrix}$$

To derive the weights where the function is minimum, we set the gradient to 0 and find the weights where this occurs.

Setting $\frac{\partial L}{\partial w_1} = 0$ and $\frac{\partial L}{\partial w_2} = 0$

$\frac{\partial L}{\partial w_1} = a(w^*_1) = 0$

$=> w^*_1 = 0$

$\frac{\partial L}{\partial w_2} = b(w^*_2) = 0$

$=> w^*_2 = 0$

Thus the weights, w^*_1 and w^*_2 that achieve the minimum value of L are 0 and 0.

$w^* = (w^*_1, w^*_2) = (0, 0)$ **are the weights that achieve the minimum value of L.**

B)

<u>Gradient Descent formula:</u>

$w_i(t+1) = w_i(t) - \alpha * \frac{\partial L}{\partial w_i(t)}$ where α is the learning rate.

Thus for w_1, that is for i=1

$w_1(t+1) = w_1(t) - \alpha * \frac{\partial L}{\partial w_1(t)}$

Substituting $\frac{\partial L}{\partial w_1(t)} = a * w_1(t)$ in the above equation

$w_1(t+1) = w_1(t) - \alpha * a * w_1(t)$

$w_1(t+1) = (1 - \alpha * a) * w_1(t)$

$w_1(t + 1) = \rho_1 * w_1(t)$ where $\rho_1 = (1 - \alpha * a)$

Similarly for w_2,

$w_2(t + 1) = w_2(t) - \alpha * \frac{\partial L}{\partial w_2(t)}$

Substituting $\frac{\partial L}{\partial w_2(t)} = b * w_2(t)$ in the above equation

$w_2(t + 1) = w_2(t) - \alpha * b * w_2(t)$

$w_2(t + 1) = (1 - \alpha * b) * w_2(t)$

$w_2(t + 1) = \rho_2 * w_2(t)$ where $\rho_2 = (1 - \alpha * b)$

C)

Since $L = 0.5 * (a * w_1^2 + b * w_2^2)$

This is a convex shaped function like a bowl, with its minima at (0,0)

w1 updations is like this: $w_1(t + 1) = \rho_1 * w_1(t)$ where $\rho_1 = (1 - \alpha * a)$

w1 updations is like this: $w_2(t + 1) = \rho_2 * w_2(t)$ where $\rho_2 = (1 - \alpha * b)$

Thus if $0 < \rho_1 < 1$ and $0 < \rho_2 < 1$, each update for both w1 and w2 will bring us closer to the minima directly. This is because multiplying $w_1(t)$, $w_2(t)$ by a number less than 1 (ρ_1, ρ_2) will iteratively take it zero which is the minima.

$=> 0 < 1 - \alpha * a < 1 \, and \, 0 < 1 - \alpha * b < 1$ —---equations 1

$=> 0 < \alpha * a < 1 \, and \, 0 < \alpha * b < 1$

$=> 0 < \alpha < 1/a \; and \; 0 < \alpha < 1/b$ (assuming a,b are non-negative)

Whichever of 1/a and 1/b is lesser is chosen as the upper bound for the learning rate while 0 is the lower bound.

$=> 0 < \alpha < min(1/a, 1/b)$

Thus when a and b are non negative, $0 < \alpha < min(1/a, 1/b)$ does the job of convergence.

If a is negative, $=> 1/a < \alpha < 0$ from equation 1 (reversing inequality due to negative sign)

This means mathematically learning rate of negative value can do the job of directly converging. But in machine learning negative learning rates are not the most intuitive. Thus when a or b is negative, $0 < \alpha < min(1/a, 1/b)$ doesn't hold as the learning rate becomes negative.

However, learning rates of positive value such as between 0.0001 and 1 can lead to convergence. The convergence may not be direct as shown in the case where a and b are positive while we set learning rate to $0 < \alpha < min(1/a, 1/b)$ but with several iterations it is shown to work.

In addition, learning rates can be changed with time like in adaptive scheduling, where

$\alpha = k/(t)^{0.5}$ where t is the time

Thus a learning rate value initialized with 1 can become 0.0001 with several iterations.

As shown in the demo for quadratic loss functions:

Newton's Method can have learning rate of 0.5 to lead to convergence.
For momentum, a learning rate of 1.2 can lead to convergence effectively.
For Adam, a learning rate of 1.6 can lead to convergence.

Thus, to summarize, a learning rate of 0.0001(approx) to 1.5(approx) can lead to convergence for our loss function depending on the optimisation algorithm. A very large value such as 100 etc will definitely not lead to convergence as the weights overshoot the optimum continuously.

D)

A scenario where a slow convergence can occur is when the a/b ratio is very large.

Since $\frac{\partial L}{\partial w_1} = a(w^*_1)$ and $\frac{\partial L}{\partial w_2} = b(w^*_2)$ The imbalance in the values of a and b means that the optimization algorithm is highly sensitive to changes in w1 but not as sensitive to changes in w2.

Thus this can lead to oscillations in the optimization process, where the algorithm continually overshoots and undershoots the optimal w1 value, leading to a slow and oscillatory convergence.

2.

a)

<u>Sobel Filters:</u>

G_X =

-1	0	1
-2	0	2
-1	0	1

AND

G_Y =

1	2	1
0	0	0
-1	-2	-1

The filter with weights G_X above is used for vertical edge detection and outputs a high value when the image has a vertical edge. This is because to the left and right side of the vertical edge, there is a gradient in pixel values as the edge separates the foreground pixels from the background pixels. This is detected with the weights G_X as the right most column is [1,2,1] and the leftmost column is [-1,-2,-1]. When there is no edge in the image, that is when there is uniform pixel intensity, the filter outputs a low value.

The filter with weights G_Y above is used for horizontal edge detection and outputs a high value when the image has a horizontal edge. This is because to the top and bottom side of the horizontal edge, there is a gradient in pixel values as the edge separates the foreground pixels from the background pixels. This is detected with the weights G_Y as the top most row is [1,2,1] and the bottom most row is [-1,-2,-1]. When there is no edge in the image, that is when there is uniform pixel intensity, the filter outputs a low value.

b)

<u>Gaussian Filter:</u>

W =

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

To create a blurring filter for a 2D image, the weights must emphasize on the smoothing of pixel values across the image. The Gaussian filter provides a weighted average of the pixel values within the filter region with the property that they give higher weights to pixels closer to the center and lower weights to pixels farther away, which effectively smooths out the finer details of the image making it appear blurred.

C)

<u>Sobel Filter:</u>

W =

1	2	1
0	0	0
-1	-2	-1

This filter can work for horizontal sharpening because after convolving with the filter, the horizontal edges and features in the images are emphasized. This is because the pixel values to the top and bottom of a horizontal edge differ (foreground and background separation) and this filter captures this difference due to positive weights on the topmost row and negative weights on the bottom most row. Thus, this design aims to enhance the contrast between pixel values along horizontal lines or edges resulting in horizontal sharpening.

D)

<u>Gaussian Filter:</u>

W =

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

The above Gaussian filter has the property of providing weighted averaging of pixel values within a local neighborhood around each pixel. The Gaussian distribution ensures that nearby pixels receive higher weights, while more distant pixels receive lower weights. This weighted averaging helps to reduce the impact of noisy pixel values. Noise, which is often characterized by high-frequency variations in pixel values, gets smoothed out as the filter emphasizes the nearby, more important pixel values.

3.A)

The IoU (Intersection over Union) metric between two bounding boxes can be defined as the ratio of the area of their intersection to the area of their union.

The area of intersection and the area of union, both are non negative.

Thus, IoU = (area of intersection/area of union) >=0

The area of intersection is always less or equal to the area of Union.

Thus, IoU = (area of intersection/area of union) <=1

When two bounding boxes don’t overlap at all, IoU is 0 as area of intersection is 0

When two bounding boxes fully overlap, IoU is 1 as area of intersection = area of union

When two bounding boxes partially overlap, IoU is greater than 0 but less than 1 as area of intersection < area of union

Thus IoU will be a real number in the range [0,1] b)

Consider the IoU metric expressed as a function of the coordinates of the bounding boxes.

When the top-left corner of one of the bounding boxes changes, the top-left corner of the intersection bounding box may or may not change, depending on the new position of the bounding box. This introduces a non-differentiable, piecewise behavior.

If the top-left corner of one bounding box moves in a way that it doesn't affect the overlap with the other box, the IoU remains the same. However, if the movement causes the boxes to overlap, the IoU starts to increase from 0 to a small value and then continues to increase to 1 as the overlap becomes larger. After reaching 1, if the movement continues, the IoU decreases again, eventually reaching 0 as the two bounding boxes no longer overlap. This piecewise behavior is characteristic of non-differentiability. This non-differentiability arises due to the sudden changes in the IoU value as the relative positions of the bounding boxes change.

Similar arguments can be applied to the other three corners of the bounding boxes, and the non-differentiability issue persists across all corners and edges.

Because of its non-differentiability, it's challenging to use the IoU directly as a loss function for training neural networks. This is because gradient-based optimization methods, such as backpropagation, require the loss function to be differentiable, as gradients are used to update model parameters during training.

AlexNet

In this problem, you are asked to train a deep convolutional neural network to perform image classification. In fact, this is a slight variation of a network called *AlexNet*. This is a landmark model in deep learning, and arguably kickstarted the current (and ongoing, and massive) wave of innovation in modern AI when its results were first presented in 2012. AlexNet was the first real-world demonstration of a *deep* classifier that was trained end-to-end on data and that outperformed all other ML models thus far.

We will train AlexNet using the CIFAR10 dataset, which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. The classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

A lot of the code you will need is already provided in this notebook; all you need to do is to fill in the missing pieces, and interpret your results.

Warning : AlexNet takes a good amount of time to train (~1 minute per epoch on Google Colab). So please budget enough time to do this homework.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import _LRScheduler
import torch.utils.data as data

import torchvision.transforms as transforms
import torchvision.datasets as datasets

from sklearn import decomposition
from sklearn import manifold
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np

import copy
import random
import time

SEED = 1234

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

Loading and Preparing the Data

Our dataset is made up of color images but three color channels (red, green and blue), compared to MNIST's black and white images with a single color channel. To normalize our data we need to calculate the means and standard deviations for each of the color channels independently, and normalize them.

```
ROOT = '.data'
train_data = datasets.CIFAR10(root = ROOT,
                              train = True,
                              download = True)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
.data/cifar-10-python.tar.gz
```

```
100%|██████████| 170498071/170498071 [00:10<00:00, 16095863.59it/s]
```

```
Extracting .data/cifar-10-python.tar.gz to .data
```

```
# Compute means and standard deviations along the R,G,B channel

means = train_data.data.mean(axis = (0,1,2)) / 255
stds = train_data.data.std(axis = (0,1,2)) / 255
```

Next, we will do data augmentation. For each training image we will randomly rotate it (by up to 5 degrees), flip/mirror with probability 0.5, shift by +/-1 pixel. Finally we will normalize each color channel using the means/stds we calculated above.

```
train_transforms = transforms.Compose([
                           transforms.RandomRotation(5),
                           transforms.RandomHorizontalFlip(0.5),
```

```
                           transforms.RandomCrop(32, padding = 2),
                           transforms.ToTensor(),
                           transforms.Normalize(mean = means,
                                                std = stds)
                       ])

test_transforms = transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize(mean = means,
                                                std = stds)
                       ])
```

Next, we'll load the dataset along with the transforms defined above.

We will also create a validation set with 10% of the training samples. The validation set will be used to monitor loss along different epochs, and we will pick the model along the optimization path that performed the best, and report final test accuracy numbers using this model.

```
train_data = datasets.CIFAR10(ROOT,
                              train = True,
                              download = True,
                              transform = train_transforms)

test_data = datasets.CIFAR10(ROOT,
                             train = False,
                             download = True,
                             transform = test_transforms)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
VALID_RATIO = 0.9

n_train_examples = int(len(train_data) * VALID_RATIO)
n_valid_examples = len(train_data) - n_train_examples

train_data, valid_data = data.random_split(train_data,
                                           [n_train_examples,
n_valid_examples])

valid_data = copy.deepcopy(valid_data)
valid_data.dataset.transform = test_transforms
```

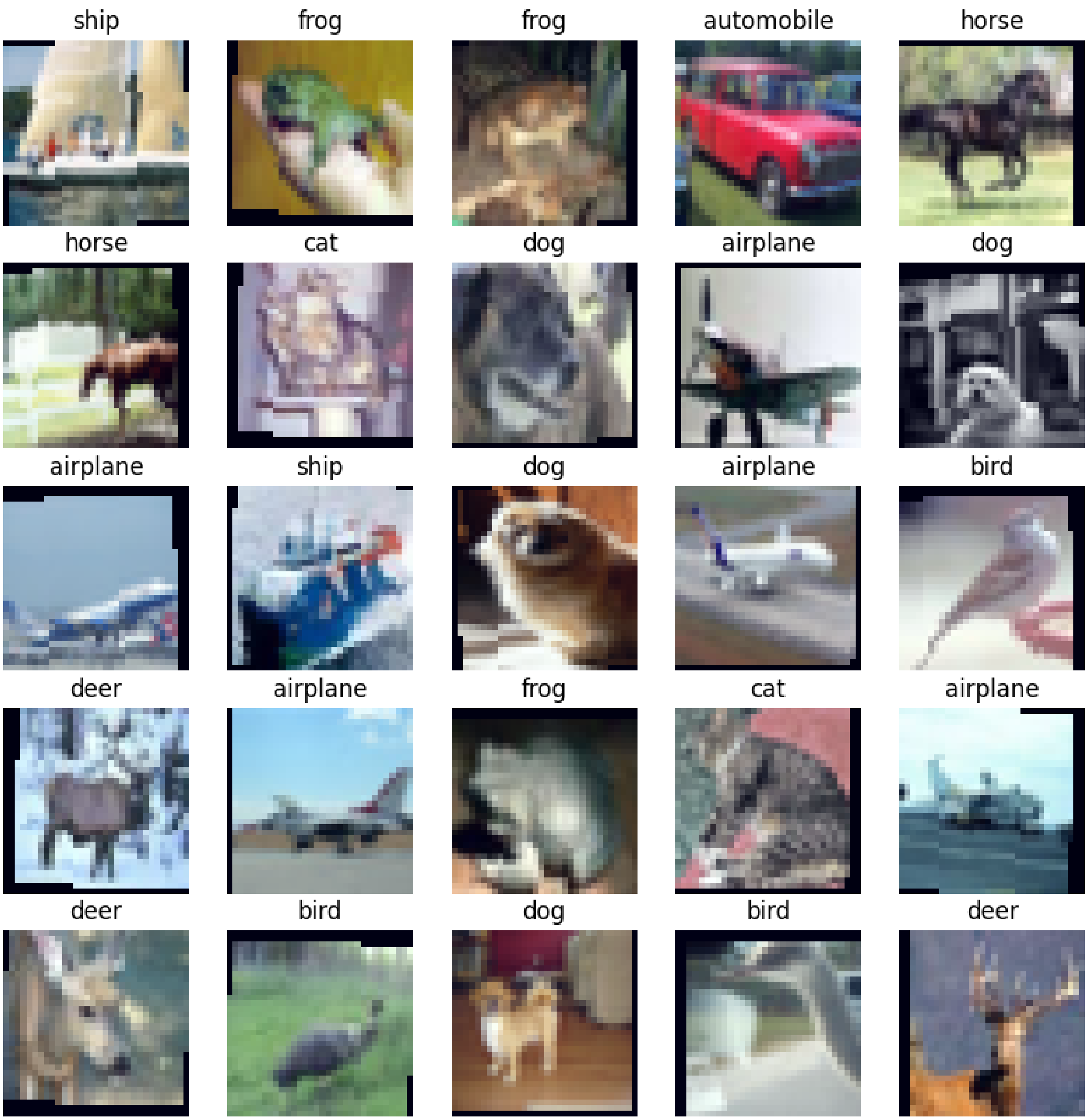
Now, we'll create a function to plot some of the images in our dataset to see what they actually look like.

Note that by default PyTorch handles images that are arranged `[channel, height, width]`, but `matplotlib` expects images to be `[height, width, channel]`, hence we need to permute the dimensions of our images before plotting them.

```python
def plot_images(images, labels, classes, normalize = False):

    n_images = len(images)

    rows = int(np.sqrt(n_images))
    cols = int(np.sqrt(n_images))

    fig = plt.figure(figsize = (10, 10))

    for i in range(rows*cols):

        ax = fig.add_subplot(rows, cols, i+1)

        image = images[i]

        if normalize:
            image_min = image.min()
            image_max = image.max()
            image.clamp_(min = image_min, max = image_max)
            image.add_(-image_min).div_(image_max - image_min + 1e-5)

        ax.imshow(image.permute(1, 2, 0).cpu().numpy())
        ax.set_title(classes[labels[i]])
        ax.axis('off')
```

One point here: `matplotlib` is expecting the values of every pixel to be between $[0, 1]$, however our normalization will cause them to be outside this range. By default `matplotlib` will then clip these values into the $[0, 1]$ range. This clipping causes all of the images to look a bit weird - all of the colors are oversaturated. The solution is to normalize each image between [0,1].

```python
N_IMAGES = 25

images, labels = zip(*[(image, label) for image, label in
                           [train_data[i] for i in range(N_IMAGES)]])

classes = test_data.classes

plot_images(images, labels, classes, normalize = True)
```



We'll be normalizing our images by default from now on, so we'll write a function that does it for us which we can use whenever we need to renormalize an image.

```
def normalize_image(image):
    image_min = image.min()
    image_max = image.max()
    image.clamp_(min = image_min, max = image_max)
    image.add_(-image_min).div_(image_max - image_min + 1e-5)
    return image
```

The final bit of the data processing is creating the iterators. We will use a large. Generally, a larger batch size means that our model trains faster but is a bit more susceptible to overfitting.

```python
# Q1: Create data loaders for train_data, valid_data, test_data
# Use batch size 256

#import utils



BATCH_SIZE = 256

train_iterator = torch.utils.data.DataLoader(
    train_data,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=4,
)

valid_iterator = torch.utils.data.DataLoader(
    valid_data,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=4,
)

test_iterator = torch.utils.data.DataLoader(
    test_data,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=4,
)
```

Defining the Model

Next up is defining the model.

AlexNet will have the following architecture:

- There are 5 2D convolutional layers (which serve as *feature extractors*), followed by 3 linear layers (which serve as the *classifier*).
- All layers (except the last one) have `ReLU` activations. (Use `inplace=True` while defining your ReLUs.)
- All convolutional filter sizes have kernel size 3 x 3 and padding 1.
- Convolutional layer 1 has stride 2. All others have the default stride (1).
- Convolutional layers 1,2, and 5 are followed by a 2D maxpool of size 2.
- Linear layers 1 and 2 are preceded by Dropouts with Bernoulli parameter 0.5.

- For the convolutional layers, the number of channels is set as follows. We start with 3 channels and then proceed like this:
  - $3 \to 64 \to 192 \to 384 \to 256 \to 256$

  In the end, if everything is correct you should get a feature map of size $ 2\times2 \times 256 = 1024$.
- For the linear layers, the feature sizes are as follows:
  - $1024 \to 4096 \to 4096 \to 10$.

  (The 10, of course, is because 10 is the number of classes in CIFAR-10).

```python
class AlexNet(nn.Module):
    def __init__(self, output_dim):
        super().__init__()



        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.MaxPool2d(kernel_size=2),
            nn.ReLU(inplace=True),

            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.MaxPool2d(kernel_size=2),
            nn.ReLU(inplace=True),

            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),

            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),

            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.MaxPool2d(kernel_size=2),
            nn.ReLU(inplace=True),
        )

        # Classifier (Linear Layers)
        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, output_dim)
        )
```

```
    def forward(self, x):
        x = self.features(x)
        h = x.view(x.shape[0], -1)
        x = self.classifier(h)
        return x, h
```

We'll create an instance of our model with the desired amount of classes.

```
OUTPUT_DIM = 10
model = AlexNet(OUTPUT_DIM)
```

Training the Model

We first initialize parameters in PyTorch by creating a function that takes in a PyTorch module, checking what type of module it is, and then using the `nn.init` methods to actually initialize the parameters.

For convolutional layers we will initialize using the *Kaiming Normal* scheme, also known as *He Normal*. For the linear layers we initialize using the *Xavier Normal* scheme, also known as *Glorot Normal*. For both types of layer we initialize the bias terms to zeros.

```
def initialize_parameters(m):
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight.data, nonlinearity = 'relu')
        nn.init.constant_(m.bias.data, 0)
    elif isinstance(m, nn.Linear):
        nn.init.xavier_normal_(m.weight.data, gain = 
nn.init.calculate_gain('relu'))
        nn.init.constant_(m.bias.data, 0)
```

We apply the initialization by using the model's `apply` method. If your definitions above are correct you should get the printed output as below.

```
model.apply(initialize_parameters)

AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 
1))
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
    (5): ReLU(inplace=True)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (12): ReLU(inplace=True)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=1024, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

We then define the loss function we want to use, the device we'll use and place our model and criterion on to our device.

```
optimizer = optim.Adam(model.parameters(), lr = 1e-3)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
criterion = nn.CrossEntropyLoss()

model = model.to(device)
criterion = criterion.to(device)

# This is formatted as code
```

We define a function to calculate accuracy...

```
def calculate_accuracy(y_pred, y):
    top_pred = y_pred.argmax(1, keepdim = True)
    correct = top_pred.eq(y.view_as(top_pred)).sum()
    acc = correct.float() / y.shape[0]
    return acc
```

As we are using dropout we need to make sure to "turn it on" when training by using `model.train()`.

```
def train(model, iterator, optimizer, criterion, device):

    epoch_loss = 0
    epoch_acc = 0
```

```
    model.train()

    for (x, y) in iterator:

        x = x.to(device)
        y = y.to(device)

        optimizer.zero_grad()

        y_pred, _ = model(x)

        loss = criterion(y_pred, y)

        acc = calculate_accuracy(y_pred, y)

        loss.backward()

        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

We also define an evaluation loop, making sure to "turn off" dropout with `model.eval()`.

```
def evaluate(model, iterator, criterion, device):

    epoch_loss = 0
    epoch_acc = 0

    model.eval()

    with torch.no_grad():

        for (x, y) in iterator:

            x = x.to(device)
            y = y.to(device)

            y_pred, _ = model(x)

            loss = criterion(y_pred, y)

            acc = calculate_accuracy(y_pred, y)

            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

Next, we define a function to tell us how long an epoch takes.

```
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

Then, finally, we train our model.

Train it for 25 epochs (using the train dataset). At the end of each epoch, compute the validation loss and keep track of the best model. You might find the command `torch.save` helpful.

At the end you should expect to see validation losses of ~76% accuracy.

```
# Q3: train your model here for 25 epochs.
# Print out training and validation loss/accuracy of the model after
each epoch
# Keep track of the model that achieved best validation loss thus far.

EPOCHS = 25

# Fill training code here

best_valid_loss = float('inf')  # Initialize with a large value
best_model = None

for epoch in range(EPOCHS):
    start_time = time.time()

    # Training
    train_loss, train_acc = train(model, train_iterator, optimizer,
criterion, device)

    # Validation
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion,
device)

    end_time = time.time()

    # Calculate the time for the current epoch
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    # Print the results for the current epoch
    print(f'Epoch: {epoch + 1:02}')
    print(f'\tTime: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc:
{train_acc*100:.2f}%')
    print(f'\tValid Loss: {valid_loss:.3f} | Valid Acc:
{valid_acc*100:.2f}%')
```

```python
    # Check if this model has the best validation loss
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        best_model = model.state_dict()  # Save the model's state_dict

# After training, you can use 'best_model' to load the best model.
```

```
Epoch: 01
     Time: 0m 15s
     Train Loss: 2.349 | Train Acc: 22.82%
     Valid Loss: 1.567 | Valid Acc: 42.05%
Epoch: 02
     Time: 0m 7s
     Train Loss: 1.519 | Train Acc: 43.57%
     Valid Loss: 1.342 | Valid Acc: 51.10%
Epoch: 03
     Time: 0m 7s
     Train Loss: 1.358 | Train Acc: 50.59%
     Valid Loss: 1.258 | Valid Acc: 55.01%
Epoch: 04
     Time: 0m 8s
     Train Loss: 1.256 | Train Acc: 54.89%
     Valid Loss: 1.159 | Valid Acc: 59.07%
Epoch: 05
     Time: 0m 7s
     Train Loss: 1.187 | Train Acc: 57.47%
     Valid Loss: 1.117 | Valid Acc: 59.93%
Epoch: 06
     Time: 0m 7s
     Train Loss: 1.111 | Train Acc: 60.45%
     Valid Loss: 1.044 | Valid Acc: 63.01%
Epoch: 07
     Time: 0m 7s
     Train Loss: 1.056 | Train Acc: 63.03%
     Valid Loss: 0.997 | Valid Acc: 65.51%
Epoch: 08
     Time: 0m 7s
     Train Loss: 1.015 | Train Acc: 64.42%
     Valid Loss: 0.952 | Valid Acc: 66.54%
Epoch: 09
     Time: 0m 7s
     Train Loss: 0.969 | Train Acc: 66.04%
     Valid Loss: 0.902 | Valid Acc: 68.51%
Epoch: 10
     Time: 0m 7s
     Train Loss: 0.928 | Train Acc: 67.56%
     Valid Loss: 0.870 | Valid Acc: 70.22%
Epoch: 11
     Time: 0m 7s
     Train Loss: 0.903 | Train Acc: 68.43%
```

```
	Valid Loss: 0.869 | Valid Acc: 70.53%
Epoch: 12
	Time: 0m 7s
	Train Loss: 0.865 | Train Acc: 69.78%
	Valid Loss: 0.817 | Valid Acc: 72.11%
Epoch: 13
	Time: 0m 7s
	Train Loss: 0.847 | Train Acc: 70.67%
	Valid Loss: 0.860 | Valid Acc: 70.88%
Epoch: 14
	Time: 0m 7s
	Train Loss: 0.815 | Train Acc: 71.98%
	Valid Loss: 0.808 | Valid Acc: 72.67%
Epoch: 15
	Time: 0m 7s
	Train Loss: 0.793 | Train Acc: 72.47%
	Valid Loss: 0.780 | Valid Acc: 74.08%
Epoch: 16
	Time: 0m 8s
	Train Loss: 0.779 | Train Acc: 73.00%
	Valid Loss: 0.803 | Valid Acc: 73.38%
Epoch: 17
	Time: 0m 7s
	Train Loss: 0.750 | Train Acc: 74.50%
	Valid Loss: 0.775 | Valid Acc: 74.60%
Epoch: 18
	Time: 0m 7s
	Train Loss: 0.738 | Train Acc: 74.43%
	Valid Loss: 0.772 | Valid Acc: 74.15%
Epoch: 19
	Time: 0m 7s
	Train Loss: 0.716 | Train Acc: 75.32%
	Valid Loss: 0.758 | Valid Acc: 74.40%
Epoch: 20
	Time: 0m 7s
	Train Loss: 0.703 | Train Acc: 75.62%
	Valid Loss: 0.743 | Valid Acc: 75.58%
Epoch: 21
	Time: 0m 7s
	Train Loss: 0.682 | Train Acc: 76.59%
	Valid Loss: 0.737 | Valid Acc: 75.98%
Epoch: 22
	Time: 0m 8s
	Train Loss: 0.668 | Train Acc: 77.08%
	Valid Loss: 0.746 | Valid Acc: 75.04%
Epoch: 23
	Time: 0m 7s
	Train Loss: 0.670 | Train Acc: 76.83%
	Valid Loss: 0.744 | Valid Acc: 75.70%
```

```
Epoch: 24
	Time: 0m 7s
	Train Loss: 0.652 | Train Acc: 77.64%
	Valid Loss: 0.747 | Valid Acc: 75.94%
Epoch: 25
	Time: 0m 7s
	Train Loss: 0.631 | Train Acc: 78.37%
	Valid Loss: 0.711 | Valid Acc: 76.75%
```

Evaluating the model

We then load the parameters of our model that achieved the best validation loss. You should expect to see ~75% accuracy of this model on the test dataset.

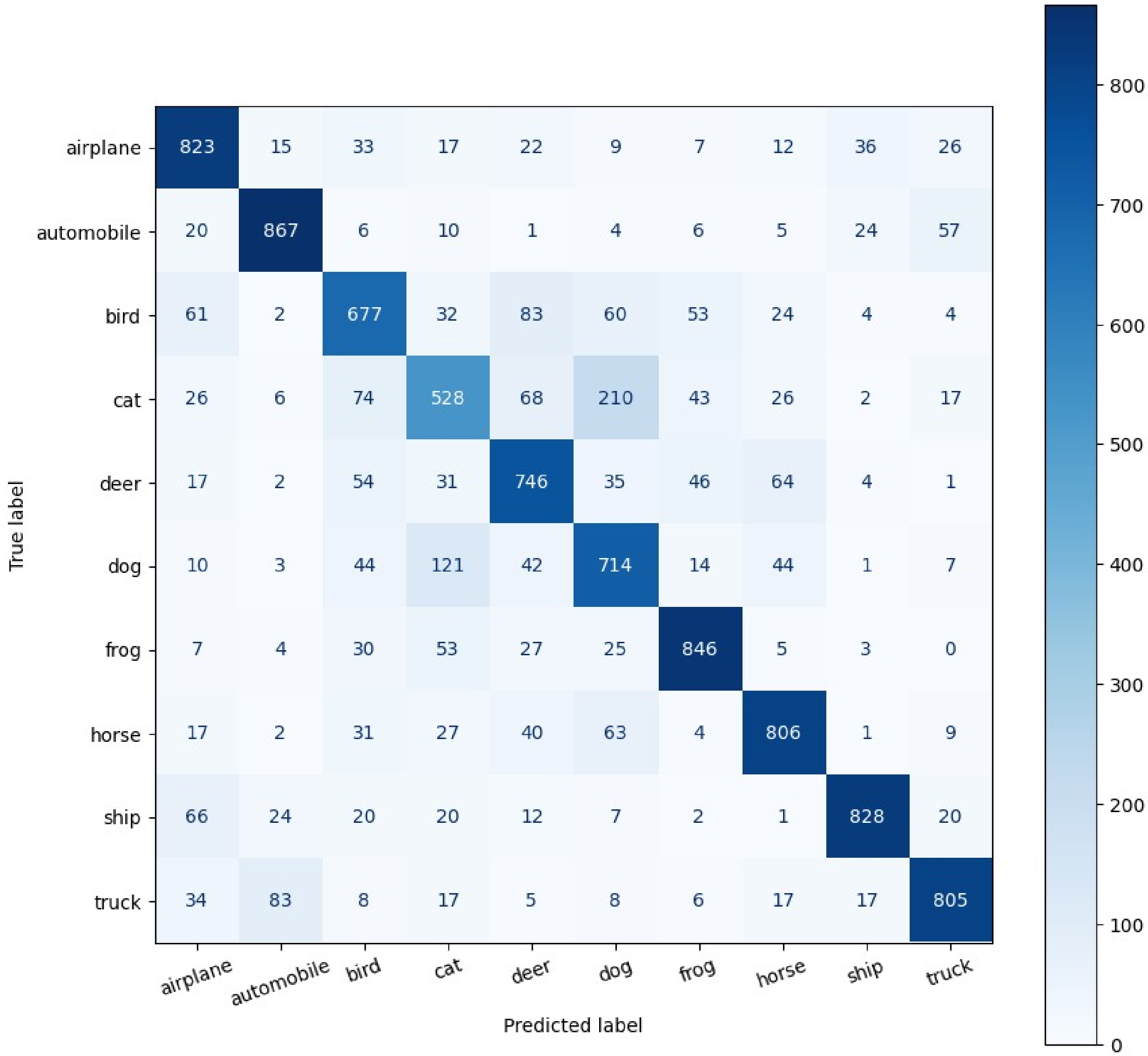
Finally, plot the confusion matrix of this model and comment on any interesting patterns you can observe there. For example, which two classes are confused the most?

```
# Q4: Load the best performing model, evaluate it on the test dataset, and print test accuracy.

# Also, print out the confusion matrox.


# Load the best performing model,
best_model_state_dict = best_model
model.load_state_dict(best_model_state_dict)
```

```
<All keys matched successfully>
```

```
def get_predictions(model, iterator, device):

    model.eval()

    labels = []
    probs = []

    epoch_loss = 0
    epoch_acc = 0

    # Q4: Fill code here.
    with torch.no_grad():
        for (x, y) in iterator:
            x = x.to(device)
            y = y.to(device)
            y_pred, _ = model(x)

            # Get predicted labels and their probabilities
            _, predicted_labels = torch.max(y_pred, 1)
            predicted_probs = torch.softmax(y_pred, dim=1)
```

```
            # Append true labels and predicted probabilities to their
respective lists
            labels.append(y)
            probs.append(predicted_probs)

            # Calculate loss and accuracy for the current batch
            loss = criterion(y_pred, y)
            acc = calculate_accuracy(y_pred, y)

            # Accumulate batch loss and accuracy to compute epoch-
level metrics
            epoch_loss += loss.item()
            epoch_acc += acc.item()

    # Calculate the average test loss and accuracy for the entire
dataset
    test_loss = epoch_loss / len(iterator)
    test_acc = epoch_acc / len(iterator)
    print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}
%')

    # Concatenate predicted probabilities and true labels for the
entire dataset
    probs = torch.cat(probs, dim=0)
    labels = torch.cat(labels, dim=0)

    return labels, probs

labels, probs = get_predictions(model, test_iterator, device)

Test Loss: 0.688 | Test Acc: 76.51%

pred_labels = torch.argmax(probs, 1)

def plot_confusion_matrix(labels, pred_labels, classes):

    fig = plt.figure(figsize = (10, 10));
    ax = fig.add_subplot(1, 1, 1);
    cm = confusion_matrix(labels, pred_labels);
    cm = ConfusionMatrixDisplay(cm, display_labels = classes);
    cm.plot(values_format = 'd', cmap = 'Blues', ax = ax)
    plt.xticks(rotation = 20)

labels = labels.to('cpu')
pred_labels = pred_labels.to('cpu')

plot_confusion_matrix(labels, pred_labels, classes)
```



Confusion Matrix comments:

See which diagnol element in the matrix (same true and predicted label) has the least value, that must be the most confused class.

Thus, the two most confused classes as per the confusion matrix is cat and bird.

The third and fourth most confused classes are dog and deer.

The least confused classes is automobile.

Conclusion

That's it! As a side project (this is not for credit and won't be graded), feel free to play around with different design choices that you made while building this network.

- Whether or not to normalize the color channels in the input.

- The learning rate parameter in Adam.
- The batch size.
- The number of training epochs.
- (and if you are feeling brave -- the AlexNet architecture itself.)

```
#Mounting google drive
from google.colab import drive
drive.mount('/content/drive')
```

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

Defining the dataset

```
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)
```

Finetuning from a pretrained model (Option 1)

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


def get_fine_tuned_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
```

Downloading some utility scripts and setting up data augmentation/transformations

```
# Download utility scripts for object detection from the PyTorch
Vision GitHub repository

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
import utils

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)
```

Training and validation for our Fine-Tuned Model

```
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('/content/drive/MyDrive/PennFudanPed',
get_transform(train=True))
dataset_test = PennFudanDataset('/content/drive/MyDrive/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_fine_tuned_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 10 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!")
```

```
/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(
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, 92.1MB/s]

Epoch: [0]  [ 0/60]  eta: 0:12:14  lr: 0.000090  loss: 9.9941 (9.9941)
loss_classifier: 0.9717 (0.9717)  loss_box_reg: 0.2146 (0.2146)
loss_mask: 8.7743 (8.7743)  loss_objectness: 0.0326 (0.0326)
loss_rpn_box_reg: 0.0009 (0.0009)  time: 12.2345  data: 3.3264  max
mem: 2151
Epoch: [0]  [10/60]  eta: 0:01:20  lr: 0.000936  loss: 2.5403 (4.3461)
loss_classifier: 0.5083 (0.5573)  loss_box_reg: 0.2605 (0.2714)
loss_mask: 1.6730 (3.4892)  loss_objectness: 0.0193 (0.0230)
loss_rpn_box_reg: 0.0039 (0.0052)  time: 1.6123  data: 0.3083  max
mem: 3407
Epoch: [0]  [20/60]  eta: 0:00:43  lr: 0.001783  loss: 1.0655 (2.6353)
```

```
loss_classifier: 0.1359 (0.3474)  loss_box_reg: 0.1742 (0.2197)
loss_mask: 0.5585 (2.0381)  loss_objectness: 0.0197 (0.0249)
loss_rpn_box_reg: 0.0038 (0.0051)  time: 0.5348  data: 0.0090  max
mem: 3407
Epoch: [0]  [30/60]  eta: 0:00:27  lr: 0.002629  loss: 0.6801 (1.9754)
loss_classifier: 0.0949 (0.2621)  loss_box_reg: 0.1624 (0.2047)
loss_mask: 0.3543 (1.4814)  loss_objectness: 0.0181 (0.0214)
loss_rpn_box_reg: 0.0042 (0.0059)  time: 0.5279  data: 0.0122  max
mem: 3407
Epoch: [0]  [40/60]  eta: 0:00:16  lr: 0.003476  loss: 0.5885 (1.6333)
loss_classifier: 0.0888 (0.2183)  loss_box_reg: 0.1846 (0.2099)
loss_mask: 0.2525 (1.1802)  loss_objectness: 0.0089 (0.0180)
loss_rpn_box_reg: 0.0080 (0.0069)  time: 0.5507  data: 0.0108  max
mem: 3407
Epoch: [0]  [50/60]  eta: 0:00:07  lr: 0.004323  loss: 0.5317 (1.4095)
loss_classifier: 0.0639 (0.1870)  loss_box_reg: 0.2042 (0.2095)
loss_mask: 0.2147 (0.9905)  loss_objectness: 0.0048 (0.0155)
loss_rpn_box_reg: 0.0081 (0.0071)  time: 0.5623  data: 0.0119  max
mem: 3407
Epoch: [0]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.4507 (1.2689)
loss_classifier: 0.0513 (0.1673)  loss_box_reg: 0.1874 (0.2083)
loss_mask: 0.1876 (0.8724)  loss_objectness: 0.0034 (0.0137)
loss_rpn_box_reg: 0.0068 (0.0072)  time: 0.5444  data: 0.0113  max
mem: 3407
Epoch: [0] Total time: 0:00:44 (0.7407 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:22  model_time: 0.2382 (0.2382)
evaluator_time: 0.0101 (0.0101)  time: 0.4453  data: 0.1947  max mem:
3407
Test:  [49/50]  eta: 0:00:00  model_time: 0.1134 (0.1350)
evaluator_time: 0.0145 (0.0211)  time: 0.1478  data: 0.0039  max mem:
3407
Test: Total time: 0:00:08 (0.1701 s / it)
Averaged stats: model_time: 0.1134 (0.1350)  evaluator_time: 0.0145
(0.0211)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.597
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.966
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.661
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.438
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.341
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.618
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.231
 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 ] = 0.672
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.520
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.600
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.683
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.665
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.966
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.841
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.374
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.276
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.689
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.250
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.706
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.710
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.480
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.678
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.722
Epoch: [1]  [ 0/60]  eta: 0:00:47  lr: 0.005000  loss: 0.2926 (0.2926)
loss_classifier: 0.0359 (0.0359)  loss_box_reg: 0.0899 (0.0899)
loss_mask: 0.1515 (0.1515)  loss_objectness: 0.0130 (0.0130)
loss_rpn_box_reg: 0.0023 (0.0023)  time: 0.7908  data: 0.2946  max
mem: 3407
Epoch: [1]  [10/60]  eta: 0:00:27  lr: 0.005000  loss: 0.2926 (0.3008)
loss_classifier: 0.0359 (0.0350)  loss_box_reg: 0.0916 (0.1098)
loss_mask: 0.1515 (0.1481)  loss_objectness: 0.0043 (0.0041)
```

```
loss_rpn_box_reg: 0.0038 (0.0037)  time: 0.5599  data: 0.0327  max
mem: 3407
Epoch: [1]  [20/60]  eta: 0:00:22  lr: 0.005000  loss: 0.2810 (0.2983)
loss_classifier: 0.0300 (0.0350)  loss_box_reg: 0.0915 (0.1056)
loss_mask: 0.1423 (0.1489)  loss_objectness: 0.0016 (0.0036)
loss_rpn_box_reg: 0.0039 (0.0052)  time: 0.5504  data: 0.0087  max
mem: 3407
Epoch: [1]  [30/60]  eta: 0:00:17  lr: 0.005000  loss: 0.2944 (0.3077)
loss_classifier: 0.0388 (0.0381)  loss_box_reg: 0.0845 (0.1094)
loss_mask: 0.1423 (0.1515)  loss_objectness: 0.0009 (0.0029)
loss_rpn_box_reg: 0.0049 (0.0058)  time: 0.5785  data: 0.0103  max
mem: 3407
Epoch: [1]  [40/60]  eta: 0:00:11  lr: 0.005000  loss: 0.2944 (0.3061)
loss_classifier: 0.0428 (0.0379)  loss_box_reg: 0.0987 (0.1085)
loss_mask: 0.1425 (0.1514)  loss_objectness: 0.0008 (0.0028)
loss_rpn_box_reg: 0.0046 (0.0054)  time: 0.5801  data: 0.0086  max
mem: 3407
Epoch: [1]  [50/60]  eta: 0:00:05  lr: 0.005000  loss: 0.2822 (0.3015)
loss_classifier: 0.0318 (0.0366)  loss_box_reg: 0.0833 (0.1049)
loss_mask: 0.1502 (0.1524)  loss_objectness: 0.0008 (0.0025)
loss_rpn_box_reg: 0.0036 (0.0051)  time: 0.5824  data: 0.0094  max
mem: 3407
Epoch: [1]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.3220 (0.3023)
loss_classifier: 0.0328 (0.0373)  loss_box_reg: 0.0833 (0.1041)
loss_mask: 0.1563 (0.1536)  loss_objectness: 0.0007 (0.0024)
loss_rpn_box_reg: 0.0036 (0.0050)  time: 0.5964  data: 0.0091  max
mem: 3407
Epoch: [1] Total time: 0:00:34 (0.5827 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:29  model_time: 0.2382 (0.2382)
evaluator_time: 0.0065 (0.0065)  time: 0.5801  data: 0.3342  max mem:
3407
Test:  [49/50]  eta: 0:00:00  model_time: 0.1005 (0.1145)
evaluator_time: 0.0038 (0.0081)  time: 0.1161  data: 0.0036  max mem:
3407
Test: Total time: 0:00:06 (0.1400 s / it)
Averaged stats: model_time: 0.1005 (0.1145)  evaluator_time: 0.0038
(0.0081)
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 ] = 0.673
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
```

```
maxDets=100 ] = 0.867
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.444
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.483
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.691
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.265
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.731
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.731
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.460
 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 ] = 0.741
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.695
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.970
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.879
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.404
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.326
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.718
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.272
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.735
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.735
 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 ] = 0.722
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.749
Epoch: [2]  [ 0/60]  eta: 0:00:54  lr: 0.005000  loss: 0.2531 (0.2531)
loss_classifier: 0.0354 (0.0354)  loss_box_reg: 0.0526 (0.0526)
loss_mask: 0.1615 (0.1615)  loss_objectness: 0.0009 (0.0009)
loss_rpn_box_reg: 0.0027 (0.0027)  time: 0.9139  data: 0.3393  max
mem: 3407
```

```
Epoch: [2]  [10/60]  eta: 0:00:31  lr: 0.005000  loss: 0.2206 (0.2512)
loss_classifier: 0.0274 (0.0311)  loss_box_reg: 0.0551 (0.0679)
loss_mask: 0.1279 (0.1456)  loss_objectness: 0.0010 (0.0027)
loss_rpn_box_reg: 0.0039 (0.0039)  time: 0.6235  data: 0.0387  max
mem: 3407
Epoch: [2]  [20/60]  eta: 0:00:23  lr: 0.005000  loss: 0.2161 (0.2409)
loss_classifier: 0.0267 (0.0304)  loss_box_reg: 0.0551 (0.0619)
loss_mask: 0.1256 (0.1421)  loss_objectness: 0.0012 (0.0026)
loss_rpn_box_reg: 0.0039 (0.0038)  time: 0.5759  data: 0.0092  max
mem: 3407
Epoch: [2]  [30/60]  eta: 0:00:17  lr: 0.005000  loss: 0.2111 (0.2378)
loss_classifier: 0.0245 (0.0302)  loss_box_reg: 0.0492 (0.0612)
loss_mask: 0.1246 (0.1403)  loss_objectness: 0.0009 (0.0022)
loss_rpn_box_reg: 0.0031 (0.0039)  time: 0.5549  data: 0.0084  max
mem: 3407
Epoch: [2]  [40/60]  eta: 0:00:11  lr: 0.005000  loss: 0.2152 (0.2376)
loss_classifier: 0.0258 (0.0303)  loss_box_reg: 0.0588 (0.0628)
loss_mask: 0.1315 (0.1385)  loss_objectness: 0.0008 (0.0019)
loss_rpn_box_reg: 0.0036 (0.0040)  time: 0.5718  data: 0.0087  max
mem: 3407
Epoch: [2]  [50/60]  eta: 0:00:05  lr: 0.005000  loss: 0.2153 (0.2311)
loss_classifier: 0.0287 (0.0294)  loss_box_reg: 0.0609 (0.0618)
loss_mask: 0.1174 (0.1343)  loss_objectness: 0.0007 (0.0018)
loss_rpn_box_reg: 0.0036 (0.0038)  time: 0.6102  data: 0.0094  max
mem: 3407
Epoch: [2]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.2048 (0.2284)
loss_classifier: 0.0254 (0.0288)  loss_box_reg: 0.0579 (0.0608)
loss_mask: 0.1157 (0.1336)  loss_objectness: 0.0005 (0.0016)
loss_rpn_box_reg: 0.0027 (0.0037)  time: 0.5979  data: 0.0079  max
mem: 3407
Epoch: [2] Total time: 0:00:35 (0.5909 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:20  model_time: 0.1705 (0.1705)
evaluator_time: 0.0044 (0.0044)  time: 0.4113  data: 0.2351  max mem:
3407
Test:  [49/50]  eta: 0:00:00  model_time: 0.0984 (0.1079)
evaluator_time: 0.0031 (0.0056)  time: 0.1135  data: 0.0037  max mem:
3407
Test: Total time: 0:00:06 (0.1261 s / it)
Averaged stats: model_time: 0.0984 (0.1079)  evaluator_time: 0.0031
(0.0056)
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 ] = 0.741
```

```
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.913
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.308
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.515
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.765
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.296
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.793
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.793
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.380
 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 ] = 0.813
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.722
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.967
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.889
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.295
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.342
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.747
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.285
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.772
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.774
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.380
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.744
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.791
Epoch: [3]  [ 0/60]  eta: 0:00:50  lr: 0.000500  loss: 0.1559 (0.1559)
loss_classifier: 0.0216 (0.0216)  loss_box_reg: 0.0263 (0.0263)
```

```
loss_mask: 0.1068 (0.1068)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0011 (0.0011)  time: 0.8373  data: 0.2845  max
mem: 3407
Epoch: [3]  [10/60]  eta: 0:00:31  lr: 0.000500  loss: 0.1706 (0.1884)
loss_classifier: 0.0238 (0.0223)  loss_box_reg: 0.0357 (0.0448)
loss_mask: 0.1068 (0.1167)  loss_objectness: 0.0005 (0.0016)
loss_rpn_box_reg: 0.0021 (0.0030)  time: 0.6253  data: 0.0351  max
mem: 3407
Epoch: [3]  [20/60]  eta: 0:00:23  lr: 0.000500  loss: 0.1706 (0.1887)
loss_classifier: 0.0238 (0.0246)  loss_box_reg: 0.0357 (0.0439)
loss_mask: 0.1093 (0.1158)  loss_objectness: 0.0004 (0.0013)
loss_rpn_box_reg: 0.0020 (0.0030)  time: 0.5789  data: 0.0088  max
mem: 3407
Epoch: [3]  [30/60]  eta: 0:00:17  lr: 0.000500  loss: 0.1689 (0.1857)
loss_classifier: 0.0207 (0.0242)  loss_box_reg: 0.0273 (0.0413)
loss_mask: 0.1099 (0.1165)  loss_objectness: 0.0004 (0.0011)
loss_rpn_box_reg: 0.0020 (0.0027)  time: 0.5743  data: 0.0090  max
mem: 3407
Epoch: [3]  [40/60]  eta: 0:00:11  lr: 0.000500  loss: 0.1799 (0.1888)
loss_classifier: 0.0198 (0.0242)  loss_box_reg: 0.0323 (0.0408)
loss_mask: 0.1154 (0.1199)  loss_objectness: 0.0004 (0.0012)
loss_rpn_box_reg: 0.0025 (0.0027)  time: 0.5977  data: 0.0102  max
mem: 3407
Epoch: [3]  [50/60]  eta: 0:00:05  lr: 0.000500  loss: 0.1842 (0.1881)
loss_classifier: 0.0198 (0.0232)  loss_box_reg: 0.0326 (0.0406)
loss_mask: 0.1230 (0.1204)  loss_objectness: 0.0005 (0.0011)
loss_rpn_box_reg: 0.0026 (0.0028)  time: 0.5926  data: 0.0091  max
mem: 3407
Epoch: [3]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1908 (0.1891)
loss_classifier: 0.0205 (0.0234)  loss_box_reg: 0.0351 (0.0413)
loss_mask: 0.1183 (0.1204)  loss_objectness: 0.0005 (0.0011)
loss_rpn_box_reg: 0.0024 (0.0028)  time: 0.5856  data: 0.0084  max
mem: 3407
Epoch: [3] Total time: 0:00:35 (0.5951 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:20  model_time: 0.1936 (0.1936)
evaluator_time: 0.0046 (0.0046)  time: 0.4008  data: 0.2013  max mem:
3407
Test:  [49/50]  eta: 0:00:00  model_time: 0.1010 (0.1097)
evaluator_time: 0.0041 (0.0061)  time: 0.1166  data: 0.0038  max mem:
3407
Test: Total time: 0:00:06 (0.1297 s / it)
Averaged stats: model_time: 0.1010 (0.1097)  evaluator_time: 0.0041
(0.0061)
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 ] = 0.973
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.943
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.334
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.569
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.817
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.313
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.837
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.837
 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 ] = 0.767
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.858
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.729
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.969
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.895
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.285
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.372
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.755
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.287
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.780
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.780
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.380
 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 ] = 0.798
Epoch: [4]  [ 0/60]  eta: 0:01:14  lr: 0.000500  loss: 0.2000 (0.2000)
loss_classifier: 0.0283 (0.0283)  loss_box_reg: 0.0574 (0.0574)
loss_mask: 0.1124 (0.1124)  loss_objectness: 0.0004 (0.0004)
loss_rpn_box_reg: 0.0015 (0.0015)  time: 1.2390  data: 0.4964  max
mem: 3407
Epoch: [4]  [10/60]  eta: 0:00:32  lr: 0.000500  loss: 0.1837 (0.1751)
loss_classifier: 0.0210 (0.0233)  loss_box_reg: 0.0352 (0.0367)
loss_mask: 0.1124 (0.1119)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0022 (0.0023)  time: 0.6400  data: 0.0528  max
mem: 3407
Epoch: [4]  [20/60]  eta: 0:00:24  lr: 0.000500  loss: 0.1549 (0.1658)
loss_classifier: 0.0165 (0.0201)  loss_box_reg: 0.0288 (0.0326)
loss_mask: 0.1073 (0.1103)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0022 (0.0022)  time: 0.5862  data: 0.0086  max
mem: 3410
Epoch: [4]  [30/60]  eta: 0:00:18  lr: 0.000500  loss: 0.1382 (0.1709)
loss_classifier: 0.0147 (0.0215)  loss_box_reg: 0.0230 (0.0342)
loss_mask: 0.1014 (0.1120)  loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0019 (0.0025)  time: 0.5929  data: 0.0090  max
mem: 3410
Epoch: [4]  [40/60]  eta: 0:00:12  lr: 0.000500  loss: 0.1879 (0.1780)
loss_classifier: 0.0206 (0.0223)  loss_box_reg: 0.0416 (0.0374)
loss_mask: 0.1151 (0.1150)  loss_objectness: 0.0006 (0.0007)
loss_rpn_box_reg: 0.0025 (0.0026)  time: 0.5954  data: 0.0085  max
mem: 3410
Epoch: [4]  [50/60]  eta: 0:00:06  lr: 0.000500  loss: 0.1976 (0.1836)
loss_classifier: 0.0208 (0.0232)  loss_box_reg: 0.0457 (0.0395)
loss_mask: 0.1255 (0.1173)  loss_objectness: 0.0006 (0.0008)
loss_rpn_box_reg: 0.0025 (0.0027)  time: 0.6122  data: 0.0094  max
mem: 3410
Epoch: [4]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1911 (0.1855)
loss_classifier: 0.0293 (0.0234)  loss_box_reg: 0.0457 (0.0398)
loss_mask: 0.1134 (0.1188)  loss_objectness: 0.0005 (0.0008)
loss_rpn_box_reg: 0.0022 (0.0027)  time: 0.5879  data: 0.0092  max
mem: 3410
Epoch: [4] Total time: 0:00:36 (0.6034 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:21  model_time: 0.1406 (0.1406)
evaluator_time: 0.0039 (0.0039)  time: 0.4317  data: 0.2859  max mem:
3410
Test:  [49/50]  eta: 0:00:00  model_time: 0.1012 (0.1153)
evaluator_time: 0.0039 (0.0086)  time: 0.1174  data: 0.0039  max mem:
3410
Test: Total time: 0:00:07 (0.1409 s / it)
Averaged stats: model_time: 0.1012 (0.1153)  evaluator_time: 0.0039
(0.0086)
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 ] = 0.790
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.921
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.369
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.575
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.816
 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 ] = 0.842
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.842
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.420
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.778
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.862
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.737
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.968
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.902
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.306
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.365
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.763
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.287
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.784
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.786
 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 ] = 0.805
Epoch: [5]  [ 0/60]  eta: 0:00:54  lr: 0.000500  loss: 0.1570 (0.1570)
loss_classifier: 0.0241 (0.0241)  loss_box_reg: 0.0265 (0.0265)
loss_mask: 0.1041 (0.1041)  loss_objectness: 0.0005 (0.0005)
loss_rpn_box_reg: 0.0019 (0.0019)  time: 0.9105  data: 0.2797  max
mem: 3410
Epoch: [5]  [10/60]  eta: 0:00:30  lr: 0.000500  loss: 0.1600 (0.1809)
loss_classifier: 0.0242 (0.0243)  loss_box_reg: 0.0299 (0.0363)
loss_mask: 0.1113 (0.1160)  loss_objectness: 0.0010 (0.0020)
loss_rpn_box_reg: 0.0019 (0.0023)  time: 0.6086  data: 0.0319  max
mem: 3410
Epoch: [5]  [20/60]  eta: 0:00:24  lr: 0.000500  loss: 0.1600 (0.1764)
loss_classifier: 0.0226 (0.0246)  loss_box_reg: 0.0269 (0.0348)
loss_mask: 0.1113 (0.1127)  loss_objectness: 0.0006 (0.0015)
loss_rpn_box_reg: 0.0019 (0.0028)  time: 0.6059  data: 0.0084  max
mem: 3410
Epoch: [5]  [30/60]  eta: 0:00:18  lr: 0.000500  loss: 0.1522 (0.1732)
loss_classifier: 0.0186 (0.0229)  loss_box_reg: 0.0269 (0.0349)
loss_mask: 0.1044 (0.1116)  loss_objectness: 0.0004 (0.0012)
loss_rpn_box_reg: 0.0016 (0.0025)  time: 0.6002  data: 0.0092  max
mem: 3410
Epoch: [5]  [40/60]  eta: 0:00:12  lr: 0.000500  loss: 0.1547 (0.1733)
loss_classifier: 0.0197 (0.0228)  loss_box_reg: 0.0320 (0.0343)
loss_mask: 0.1059 (0.1126)  loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0015 (0.0023)  time: 0.5807  data: 0.0090  max
mem: 3779
Epoch: [5]  [50/60]  eta: 0:00:05  lr: 0.000500  loss: 0.1674 (0.1724)
loss_classifier: 0.0222 (0.0229)  loss_box_reg: 0.0320 (0.0342)
loss_mask: 0.1059 (0.1118)  loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0015 (0.0023)  time: 0.5845  data: 0.0097  max
mem: 3779
Epoch: [5]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.1864 (0.1783)
loss_classifier: 0.0237 (0.0231)  loss_box_reg: 0.0342 (0.0361)
loss_mask: 0.1142 (0.1155)  loss_objectness: 0.0004 (0.0011)
loss_rpn_box_reg: 0.0021 (0.0025)  time: 0.5814  data: 0.0085  max
mem: 3779
Epoch: [5] Total time: 0:00:35 (0.5992 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:19  model_time: 0.1444 (0.1444)
evaluator_time: 0.0061 (0.0061)  time: 0.3975  data: 0.2457  max mem:
3779
Test:  [49/50]  eta: 0:00:00  model_time: 0.1005 (0.1083)
evaluator_time: 0.0030 (0.0054)  time: 0.1134  data: 0.0036  max mem:
3779
Test: Total time: 0:00:06 (0.1264 s / it)
Averaged stats: model_time: 0.1005 (0.1083)  evaluator_time: 0.0030
```

```
(0.0054)
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 ] = 0.793
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.930
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.339
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.589
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.819
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.315
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.844
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.844
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.420
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.789
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.864
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.737
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.916
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.308
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.347
 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 ] = 0.288
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.786
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.787
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.420
 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 ] = 0.802
Epoch: [6]  [ 0/60]  eta: 0:01:00  lr: 0.000050  loss: 0.2225 (0.2225)
loss_classifier: 0.0314 (0.0314)  loss_box_reg: 0.0443 (0.0443)
loss_mask: 0.1423 (0.1423)  loss_objectness: 0.0002 (0.0002)
loss_rpn_box_reg: 0.0043 (0.0043)  time: 1.0073  data: 0.3170  max
mem: 3779
Epoch: [6]  [10/60]  eta: 0:00:33  lr: 0.000050  loss: 0.1692 (0.1704)
loss_classifier: 0.0220 (0.0220)  loss_box_reg: 0.0307 (0.0314)
loss_mask: 0.1047 (0.1130)  loss_objectness: 0.0005 (0.0014)
loss_rpn_box_reg: 0.0022 (0.0026)  time: 0.6642  data: 0.0387  max
mem: 3779
Epoch: [6]  [20/60]  eta: 0:00:24  lr: 0.000050  loss: 0.1605 (0.1715)
loss_classifier: 0.0202 (0.0224)  loss_box_reg: 0.0281 (0.0313)
loss_mask: 0.1100 (0.1145)  loss_objectness: 0.0005 (0.0012)
loss_rpn_box_reg: 0.0017 (0.0021)  time: 0.5981  data: 0.0097  max
mem: 3779
Epoch: [6]  [30/60]  eta: 0:00:17  lr: 0.000050  loss: 0.1690 (0.1744)
loss_classifier: 0.0202 (0.0232)  loss_box_reg: 0.0281 (0.0323)
loss_mask: 0.1133 (0.1156)  loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0015 (0.0020)  time: 0.5603  data: 0.0089  max
mem: 3779
Epoch: [6]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1902 (0.1749)
loss_classifier: 0.0248 (0.0236)  loss_box_reg: 0.0345 (0.0336)
loss_mask: 0.1122 (0.1144)  loss_objectness: 0.0005 (0.0011)
loss_rpn_box_reg: 0.0019 (0.0022)  time: 0.5786  data: 0.0098  max
mem: 3779
Epoch: [6]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1706 (0.1746)
loss_classifier: 0.0248 (0.0231)  loss_box_reg: 0.0345 (0.0336)
loss_mask: 0.1102 (0.1146)  loss_objectness: 0.0005 (0.0010)
loss_rpn_box_reg: 0.0027 (0.0023)  time: 0.5921  data: 0.0095  max
mem: 3779
Epoch: [6]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1706 (0.1763)
loss_classifier: 0.0233 (0.0228)  loss_box_reg: 0.0322 (0.0339)
loss_mask: 0.1174 (0.1163)  loss_objectness: 0.0004 (0.0010)
loss_rpn_box_reg: 0.0029 (0.0024)  time: 0.6022  data: 0.0081  max
mem: 3779
Epoch: [6] Total time: 0:00:36 (0.6023 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:20  model_time: 0.1700 (0.1700)
evaluator_time: 0.0040 (0.0040)  time: 0.4024  data: 0.2271  max mem:
3779
Test:  [49/50]  eta: 0:00:00  model_time: 0.1016 (0.1096)
evaluator_time: 0.0041 (0.0057)  time: 0.1160  data: 0.0039  max mem:
```

```
3779
Test: Total time: 0:00:06 (0.1293 s / it)
Averaged stats: model_time: 0.1016 (0.1096)  evaluator_time: 0.0041
(0.0057)
Accumulating evaluation results...
DONE (t=0.03s).
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.792
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.930
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.334
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.589
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.818
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.315
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.842
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.842
 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 ] = 0.789
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.863
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.734
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.968
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.908
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.299
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.339
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.759
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.289
 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 ] = 0.784
 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 ] = 0.744
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.802
Epoch: [7]  [ 0/60]  eta: 0:01:10  lr: 0.000050  loss: 0.1581 (0.1581)
loss_classifier: 0.0195 (0.0195)  loss_box_reg: 0.0305 (0.0305)
loss_mask: 0.1040 (0.1040)  loss_objectness: 0.0011 (0.0011)
loss_rpn_box_reg: 0.0030 (0.0030)  time: 1.1739  data: 0.5194  max
mem: 3779
Epoch: [7]  [10/60]  eta: 0:00:32  lr: 0.000050  loss: 0.1719 (0.1826)
loss_classifier: 0.0195 (0.0235)  loss_box_reg: 0.0384 (0.0371)
loss_mask: 0.1124 (0.1185)  loss_objectness: 0.0004 (0.0006)
loss_rpn_box_reg: 0.0028 (0.0029)  time: 0.6466  data: 0.0533  max
mem: 3779
Epoch: [7]  [20/60]  eta: 0:00:24  lr: 0.000050  loss: 0.1719 (0.1936)
loss_classifier: 0.0225 (0.0260)  loss_box_reg: 0.0384 (0.0400)
loss_mask: 0.1154 (0.1236)  loss_objectness: 0.0004 (0.0011)
loss_rpn_box_reg: 0.0025 (0.0029)  time: 0.5911  data: 0.0080  max
mem: 3779
Epoch: [7]  [30/60]  eta: 0:00:18  lr: 0.000050  loss: 0.1718 (0.1838)
loss_classifier: 0.0200 (0.0241)  loss_box_reg: 0.0291 (0.0366)
loss_mask: 0.1131 (0.1198)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0022 (0.0026)  time: 0.5773  data: 0.0096  max
mem: 3779
Epoch: [7]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1669 (0.1785)
loss_classifier: 0.0179 (0.0226)  loss_box_reg: 0.0289 (0.0351)
loss_mask: 0.1117 (0.1175)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0017 (0.0025)  time: 0.5661  data: 0.0088  max
mem: 3779
Epoch: [7]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1603 (0.1758)
loss_classifier: 0.0179 (0.0221)  loss_box_reg: 0.0281 (0.0341)
loss_mask: 0.1034 (0.1162)  loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0021 (0.0024)  time: 0.5967  data: 0.0105  max
mem: 3779
Epoch: [7]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1628 (0.1761)
loss_classifier: 0.0193 (0.0229)  loss_box_reg: 0.0281 (0.0345)
loss_mask: 0.1034 (0.1154)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0018 (0.0024)  time: 0.5929  data: 0.0101  max
mem: 3779
Epoch: [7] Total time: 0:00:35 (0.5961 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:20  model_time: 0.1477 (0.1477)
evaluator_time: 0.0046 (0.0046)  time: 0.4061  data: 0.2524  max mem:
```

```
3779
Test:  [49/50]  eta: 0:00:00  model_time: 0.1069 (0.1163)
evaluator_time: 0.0037 (0.0077)  time: 0.1259  data: 0.0062  max mem:
3779
Test: Total time: 0:00:07 (0.1406 s / it)
Averaged stats: model_time: 0.1069 (0.1163)  evaluator_time: 0.0037
(0.0077)
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 ] = 0.792
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.922
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.334
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.589
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.817
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.315
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.841
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.841
 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 ] = 0.789
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.862
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.735
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.908
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.308
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.333
 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 ] = 0.289
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.783
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.785
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.420
 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 ] = 0.803
Epoch: [8]  [ 0/60]  eta: 0:01:01  lr: 0.000050  loss: 0.1972 (0.1972)
loss_classifier: 0.0473 (0.0473)  loss_box_reg: 0.0457 (0.0457)
loss_mask: 0.1003 (0.1003)  loss_objectness: 0.0012 (0.0012)
loss_rpn_box_reg: 0.0027 (0.0027)  time: 1.0282  data: 0.3046  max
mem: 3779
Epoch: [8]  [10/60]  eta: 0:00:31  lr: 0.000050  loss: 0.1721 (0.1809)
loss_classifier: 0.0237 (0.0249)  loss_box_reg: 0.0286 (0.0378)
loss_mask: 0.1126 (0.1152)  loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0020 (0.0022)  time: 0.6214  data: 0.0346  max
mem: 3779
Epoch: [8]  [20/60]  eta: 0:00:24  lr: 0.000050  loss: 0.1669 (0.1819)
loss_classifier: 0.0207 (0.0244)  loss_box_reg: 0.0286 (0.0392)
loss_mask: 0.1121 (0.1152)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0019 (0.0025)  time: 0.5878  data: 0.0098  max
mem: 3779
Epoch: [8]  [30/60]  eta: 0:00:17  lr: 0.000050  loss: 0.1732 (0.1791)
loss_classifier: 0.0221 (0.0240)  loss_box_reg: 0.0316 (0.0371)
loss_mask: 0.1121 (0.1147)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0019 (0.0024)  time: 0.5788  data: 0.0101  max
mem: 3779
Epoch: [8]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1644 (0.1758)
loss_classifier: 0.0187 (0.0230)  loss_box_reg: 0.0300 (0.0364)
loss_mask: 0.1056 (0.1133)  loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0013 (0.0023)  time: 0.5661  data: 0.0101  max
mem: 3779
Epoch: [8]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1460 (0.1709)
loss_classifier: 0.0148 (0.0217)  loss_box_reg: 0.0224 (0.0347)
loss_mask: 0.1005 (0.1115)  loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0020 (0.0023)  time: 0.5697  data: 0.0099  max
mem: 3779
Epoch: [8]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1596 (0.1747)
loss_classifier: 0.0173 (0.0223)  loss_box_reg: 0.0295 (0.0356)
loss_mask: 0.1069 (0.1137)  loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0024 (0.0024)  time: 0.5978  data: 0.0081  max
mem: 3779
Epoch: [8] Total time: 0:00:35 (0.5921 s / it)
creating index...
```

```
index created!
Test:  [ 0/50]  eta: 0:00:29  model_time: 0.2405 (0.2405)
evaluator_time: 0.0040 (0.0040)  time: 0.5880  data: 0.3419  max mem:
3779
Test:  [49/50]  eta: 0:00:00  model_time: 0.1009 (0.1111)
evaluator_time: 0.0030 (0.0055)  time: 0.1148  data: 0.0037  max mem:
3779
Test: Total time: 0:00:06 (0.1317 s / it)
Averaged stats: model_time: 0.1009 (0.1111)  evaluator_time: 0.0030
(0.0055)
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 ] = 0.792
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.922
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.334
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.589
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.817
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.316
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.841
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.841
 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 ] = 0.789
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.862
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.735
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.908
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.308
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
```

```
maxDets=100 ] = 0.336
 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 ] = 0.288
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.783
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.785
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.420
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.744
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.802
Epoch: [9]  [ 0/60]  eta: 0:00:56  lr: 0.000005  loss: 0.3186 (0.3186)
loss_classifier: 0.0453 (0.0453)  loss_box_reg: 0.1030 (0.1030)
loss_mask: 0.1553 (0.1553)  loss_objectness: 0.0088 (0.0088)
loss_rpn_box_reg: 0.0061 (0.0061)  time: 0.9412  data: 0.3493  max
mem: 3779
Epoch: [9]  [10/60]  eta: 0:00:29  lr: 0.000005  loss: 0.1587 (0.1720)
loss_classifier: 0.0184 (0.0197)  loss_box_reg: 0.0285 (0.0338)
loss_mask: 0.1106 (0.1150)  loss_objectness: 0.0003 (0.0013)
loss_rpn_box_reg: 0.0019 (0.0023)  time: 0.5939  data: 0.0393  max
mem: 3779
Epoch: [9]  [20/60]  eta: 0:00:23  lr: 0.000005  loss: 0.1529 (0.1661)
loss_classifier: 0.0172 (0.0202)  loss_box_reg: 0.0282 (0.0323)
loss_mask: 0.1032 (0.1105)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0019 (0.0021)  time: 0.5664  data: 0.0081  max
mem: 3779
Epoch: [9]  [30/60]  eta: 0:00:17  lr: 0.000005  loss: 0.1641 (0.1727)
loss_classifier: 0.0190 (0.0211)  loss_box_reg: 0.0297 (0.0341)
loss_mask: 0.1054 (0.1145)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0019 (0.0021)  time: 0.5646  data: 0.0082  max
mem: 3779
Epoch: [9]  [40/60]  eta: 0:00:11  lr: 0.000005  loss: 0.1668 (0.1693)
loss_classifier: 0.0190 (0.0207)  loss_box_reg: 0.0282 (0.0328)
loss_mask: 0.1064 (0.1127)  loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0019 (0.0022)  time: 0.5789  data: 0.0086  max
mem: 3779
Epoch: [9]  [50/60]  eta: 0:00:05  lr: 0.000005  loss: 0.1668 (0.1765)
loss_classifier: 0.0227 (0.0227)  loss_box_reg: 0.0283 (0.0349)
loss_mask: 0.1100 (0.1158)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0019 (0.0025)  time: 0.6191  data: 0.0088  max
mem: 3779
Epoch: [9]  [59/60]  eta: 0:00:00  lr: 0.000005  loss: 0.1646 (0.1758)
loss_classifier: 0.0197 (0.0221)  loss_box_reg: 0.0283 (0.0343)
loss_mask: 0.1131 (0.1162)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0018 (0.0024)  time: 0.6305  data: 0.0085  max
```

```
mem: 3779
Epoch: [9] Total time: 0:00:35 (0.5994 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:21  model_time: 0.1965 (0.1965)
evaluator_time: 0.0040 (0.0040)  time: 0.4318  data: 0.2301  max mem:
3779
Test:  [49/50]  eta: 0:00:00  model_time: 0.1138 (0.1178)
evaluator_time: 0.0051 (0.0068)  time: 0.1340  data: 0.0062  max mem:
3779
Test: Total time: 0:00:06 (0.1386 s / it)
Averaged stats: model_time: 0.1138 (0.1178)  evaluator_time: 0.0051
(0.0068)
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 ] = 0.791
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.922
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.334
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.589
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.817
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.316
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.840
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.840
 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 ] = 0.789
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.860
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.734
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.972
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.908
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = 0.299
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.336
 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 ] = 0.288
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.782
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.784
 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 ] = 0.744
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.802
That's it!
```

Comments on training log:

In the above training log, note the last batch of the 10th epoch result.

Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.1646 (0.1758) loss_classifier: 0.0197 (0.0221) loss_box_reg: 0.0283 (0.0343) loss_mask: 0.1131 (0.1162) loss_objectness: 0.0003 (0.0008) loss_rpn_box_reg: 0.0018 (0.0024) time: 0.6305 data: 0.0085 max mem: 3779

Here, the loss value in paranthesis represents the cumulative loss over the entire epoch upto that point(here its the last batch so its for the entire epoch) using the weights after the completion of the 10th epoch.

Similarly for loss_classifier, loss_mask,oss_objectness, loss_rpn_box_reg

These results can be used for comparing the model performance asked in Q5B)

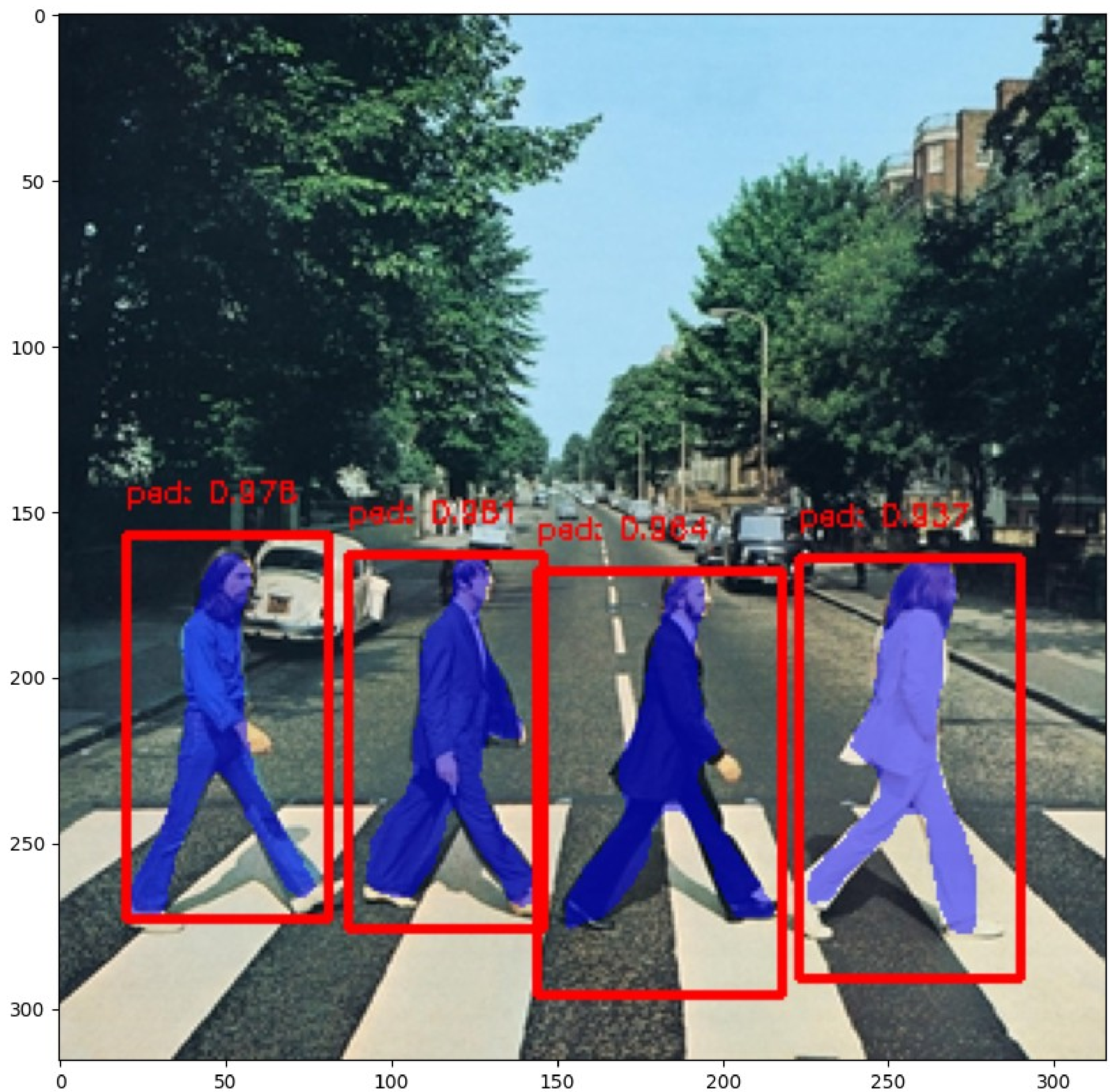
Testing of finetuned model on Beatles_Abbey_Road Test image: (Method 1)

```
import matplotlib.pyplot as plt
import cv2
from torchvision.utils import draw_bounding_boxes,
draw_segmentation_masks

# Read an image from a specified path
image = read_image("/content/sample_data/Beatles_-_Abbey_Road.jpeg")

# Create an output image to visualize the results
output_image = image

# Obtain an evaluation transformation with 'train=False'
eval_transform = get_transform(train=False)
```

```python
# Set the model in evaluation mode
model.eval()

with torch.no_grad():
    x = eval_transform(image)

    # convert RGBA -> RGB and move to device
    x = x[:3, ...].to(device)

    # Make predictions using the model
    predictions = model([x, ])
    pred = predictions[0]

# Normalize and convert the image to 8-bit integers (uint8)
image = (255.0 * (image - image.min()) / (image.max() -
image.min())).to(torch.uint8)
image = image[:3, ...]

# Filter predictions based on confidence scores (only keep scores >
0.65)
# mask refers to binary-mask ie true, false of predictions above
confidence( not meaning the mask displayed in the image)
mask = pred["scores"] > 0.65
filtered_pred = {key: value[mask] for key, value in pred.items()}


#Obtaining labels, boxes and masks for filtered predictions
filtered_labels = [f"ped: {score:.3f}" for score in
filtered_pred["scores"]]
filtered_boxes = filtered_pred["boxes"].long()
masks = (filtered_pred["masks"] > 0.7).squeeze(1)

#output image having the filtered prediction masks now
output_image = draw_segmentation_masks(output_image, masks, alpha=0.5,
colors="blue")

# Convert to NumPy array
output_image = output_image.permute(1, 2, 0).cpu().numpy().copy()

#Drawing the boxes, labels using cv2
for label, box in zip(filtered_labels, filtered_boxes):
    x_1, y_1, x_2, y_2 = [coord.item() for coord in box]
    output_image = cv2.rectangle(output_image, (x_1, y_1), (x_2, y_2),
(255, 0, 0), 2)
    output_image = cv2.putText(output_image, label, (x_1, y_1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.3, (255, 0, 0), 1)

#Plotting the final image
plt.figure(figsize=(10, 10))
```

```
plt.imshow(output_image)
```

```
<matplotlib.image.AxesImage at 0x7a4c54537cd0>
```




```
print(pred["scores"])
```

```
tensor([0.9806, 0.9776, 0.9639, 0.9368, 0.1338, 0.1002, 0.0907],
       device='cuda:0')
```

Comments on testing (method 1) result:

In the above method for testing, predictions were inferred from the model. Each prediction dictionary consisted of labels, boxes, masks, confidence scores as keys.

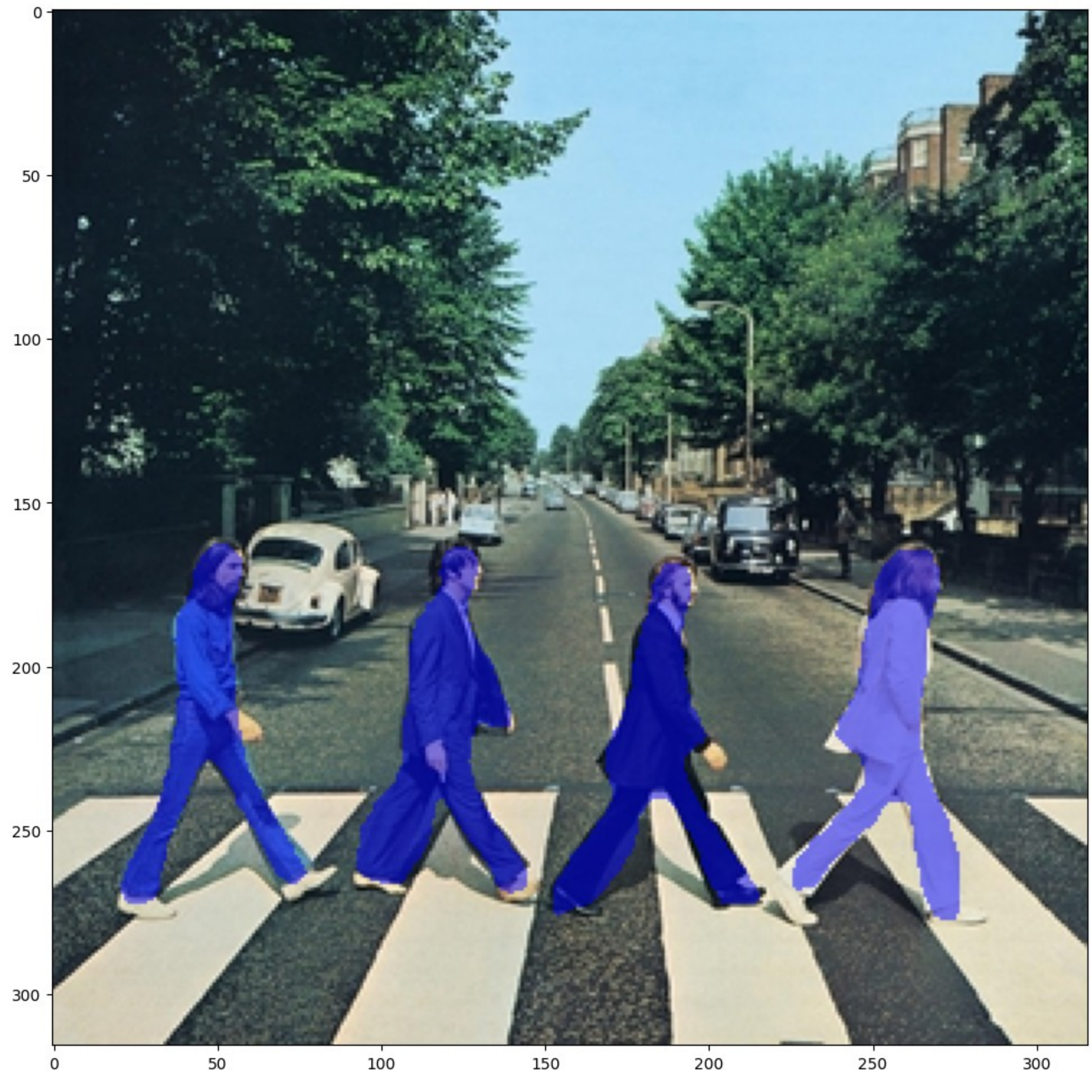
Using confidence score threshold of 0.7, predictions in the list were filtered and the corresponding boxes, labels, masks for the the filtered predictions were outputed. In this case, out of all prediction scores (as printed above) four of them crossed the threshold.

Testing of finetuned model on Beatles_Abbey_Road Test image: (Method 2)

```
import matplotlib.pyplot as plt

from torchvision.utils import draw_bounding_boxes,
draw_segmentation_masks

# Read an image from a specified path
image = read_image("/content/sample_data/Beatles_-_Abbey_Road.jpeg")

# Create an output image to visualize the results
output_image = image

# Obtain an evaluation transformation with 'train=False'
eval_transform = get_transform(train=False)

# Set the model in evaluation mode
model.eval()

with torch.no_grad():
    x = eval_transform(image)

    # convert RGBA -> RGB and move to device
    x = x[:3, ...].to(device)

    # Make predictions using the model
    predictions = model([x, ])
    pred = predictions[0]

# Normalize and convert the image to 8-bit integers (uint8)
image = (255.0 * (image - image.min()) / (image.max() -
image.min())).to(torch.uint8)
image = image[:3, ...]

#Filtering masks based on confidence
masks = (pred["masks"] > 0.7).squeeze(1)

#output image having the filtered prediction masks now
output_image = draw_segmentation_masks(output_image, masks, alpha=0.5,
colors="blue")

#Plotting the final output image
plt.figure(figsize=(12, 12))
```

```
plt.imshow(output_image.permute(1, 2, 0))
```

```
&lt;matplotlib.image.AxesImage at 0x7a4c54332e00&gt;
```



Comments on testing (method 2) result:

In the above method for testing, predictions were inferred from the model.

From each prediction, the corresponding masks whose values exceeded the set threshold were used for the output image.

Prediction dictionaries with low scores like 0.1 etc might have some values in their masks as 0.8 etc (above the threshod). These masks are included in this method but discared in the previous method.

2 - Modifying the model to add a different backbone - Mobilenet (Option 2)

```
import torchvision
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
from torchvision.models.detection import MaskRCNN



def get_backbone_model_instance_segmentation(num_classes):
    # Load a pre-trained model for classification and return only the
features
    backbone =
torchvision.models.mobilenet_v2(weights="DEFAULT").features
    # Set the number of output channels in the backbone to 1280
    backbone.out_channels = 1280

    # Define the anchor generator with desired anchor sizes and aspect
ratios
    anchor_generator = AnchorGenerator(
        sizes=((32, 64, 128, 256, 512),),
        aspect_ratios=((0.5, 1.0, 2.0),)
    )

    # Define the feature maps to use for region of interest cropping
and resizing
    roi_pooler = torchvision.ops.MultiScaleRoIAlign(
        featmap_names=['0'],
        output_size=7,
        sampling_ratio=2,
    )

    mask_roi_pooler =
torchvision.ops.MultiScaleRoIAlign(featmap_names=['0'],

output_size=14,

sampling_ratio=2)


    # Create a Mask R-CNN model with the custom backbone
    model = MaskRCNN(
        backbone,
        num_classes=num_classes,
        rpn_anchor_generator=anchor_generator,
        box_roi_pool=roi_pooler,
```

```
        mask_roi_pool=mask_roi_pooler
    )



    return model
```

Q5A) The above is the code for the modified backbone model.

Training and validation for our Modified backbone Model

```
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('/content/drive/MyDrive/PennFudanPed',
get_transform(train=True))
dataset_test = PennFudanDataset('/content/drive/MyDrive/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_backbone_model_instance_segmentation(num_classes)
```

```python
# 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!")
```

```
/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(
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, 27.0MB/s]

Epoch: [0]  [ 0/60]  eta: 0:01:18  lr: 0.000090  loss: 3.8727 (3.8727)
loss_classifier: 0.6975 (0.6975)  loss_box_reg: 0.0829 (0.0829)
loss_mask: 2.2960 (2.2960)  loss_objectness: 0.6972 (0.6972)
loss_rpn_box_reg: 0.0992 (0.0992)  time: 1.3155  data: 0.4969  max
```

```
mem: 4376
Epoch: [0]  [10/60]  eta: 0:00:23  lr: 0.000936  loss: 3.5443 (3.4568)
loss_classifier: 0.6548 (0.6223)  loss_box_reg: 0.0848 (0.0949)
loss_mask: 2.0635 (2.0079)  loss_objectness: 0.6926 (0.6870)
loss_rpn_box_reg: 0.0426 (0.0448)  time: 0.4704  data: 0.0509  max
mem: 5168
Epoch: [0]  [20/60]  eta: 0:00:17  lr: 0.001783  loss: 2.6578 (2.8665)
loss_classifier: 0.4284 (0.4715)  loss_box_reg: 0.1107 (0.1281)
loss_mask: 1.3966 (1.5790)  loss_objectness: 0.6510 (0.6464)
loss_rpn_box_reg: 0.0311 (0.0416)  time: 0.4059  data: 0.0087  max
mem: 5187
Epoch: [0]  [30/60]  eta: 0:00:13  lr: 0.002629  loss: 1.7916 (2.4441)
loss_classifier: 0.2523 (0.3883)  loss_box_reg: 0.1195 (0.1288)
loss_mask: 0.7981 (1.3042)  loss_objectness: 0.5453 (0.5848)
loss_rpn_box_reg: 0.0255 (0.0381)  time: 0.4202  data: 0.0102  max
mem: 5238
Epoch: [0]  [40/60]  eta: 0:00:08  lr: 0.003476  loss: 1.4503 (2.2111)
loss_classifier: 0.2523 (0.3698)  loss_box_reg: 0.1484 (0.1424)
loss_mask: 0.6806 (1.1420)  loss_objectness: 0.3846 (0.5214)
loss_rpn_box_reg: 0.0230 (0.0354)  time: 0.4082  data: 0.0088  max
mem: 5238
Epoch: [0]  [50/60]  eta: 0:00:04  lr: 0.004323  loss: 1.4167 (2.0313)
loss_classifier: 0.2569 (0.3498)  loss_box_reg: 0.1516 (0.1513)
loss_mask: 0.5925 (1.0291)  loss_objectness: 0.2670 (0.4668)
loss_rpn_box_reg: 0.0260 (0.0344)  time: 0.4050  data: 0.0094  max
mem: 5238
Epoch: [0]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 1.1461 (1.8956)
loss_classifier: 0.2199 (0.3301)  loss_box_reg: 0.1576 (0.1543)
loss_mask: 0.5550 (0.9523)  loss_objectness: 0.2200 (0.4261)
loss_rpn_box_reg: 0.0255 (0.0329)  time: 0.4047  data: 0.0092  max
mem: 5238
Epoch: [0] Total time: 0:00:25 (0.4244 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:29  model_time: 0.2214 (0.2214)
evaluator_time: 0.0480 (0.0480)  time: 0.5894  data: 0.3163  max mem:
5238
Test:  [49/50]  eta: 0:00:00  model_time: 0.1692 (0.1585)
evaluator_time: 0.0416 (0.0388)  time: 0.2242  data: 0.0056  max mem:
5238
Test: Total time: 0:00:10 (0.2145 s / it)
Averaged stats: model_time: 0.1692 (0.1585)  evaluator_time: 0.0416
(0.0388)
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 ] = 0.010
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.035
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.002
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.093
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.018
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.082
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.310
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.329
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.006
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.031
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.063
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.009
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.057
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.178
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.189
Epoch: [1]  [ 0/60]  eta: 0:00:50  lr: 0.005000  loss: 1.1749 (1.1749)
```

```
loss_classifier: 0.2219 (0.2219)  loss_box_reg: 0.2162 (0.2162)
loss_mask: 0.5386 (0.5386)  loss_objectness: 0.1663 (0.1663)
loss_rpn_box_reg: 0.0319 (0.0319)  time: 0.8437  data: 0.3308  max
mem: 5238
Epoch: [1]  [10/60]  eta: 0:00:22  lr: 0.005000  loss: 1.1403 (1.0908)
loss_classifier: 0.2185 (0.2094)  loss_box_reg: 0.2050 (0.1862)
loss_mask: 0.5008 (0.5024)  loss_objectness: 0.1663 (0.1667)
loss_rpn_box_reg: 0.0230 (0.0261)  time: 0.4479  data: 0.0367  max
mem: 5238
Epoch: [1]  [20/60]  eta: 0:00:17  lr: 0.005000  loss: 1.0351 (1.0603)
loss_classifier: 0.2005 (0.2010)  loss_box_reg: 0.1857 (0.1847)
loss_mask: 0.4656 (0.4918)  loss_objectness: 0.1493 (0.1560)
loss_rpn_box_reg: 0.0255 (0.0269)  time: 0.4136  data: 0.0088  max
mem: 5238
Epoch: [1]  [30/60]  eta: 0:00:12  lr: 0.005000  loss: 0.9634 (1.0375)
loss_classifier: 0.1728 (0.1897)  loss_box_reg: 0.1574 (0.1869)
loss_mask: 0.4656 (0.4854)  loss_objectness: 0.1364 (0.1483)
loss_rpn_box_reg: 0.0255 (0.0272)  time: 0.4240  data: 0.0105  max
mem: 5238
Epoch: [1]  [40/60]  eta: 0:00:08  lr: 0.005000  loss: 0.8829 (1.0028)
loss_classifier: 0.1422 (0.1786)  loss_box_reg: 0.1574 (0.1845)
loss_mask: 0.4474 (0.4745)  loss_objectness: 0.1141 (0.1388)
loss_rpn_box_reg: 0.0228 (0.0263)  time: 0.4159  data: 0.0096  max
mem: 5238
Epoch: [1]  [50/60]  eta: 0:00:04  lr: 0.005000  loss: 0.8409 (0.9674)
loss_classifier: 0.1318 (0.1692)  loss_box_reg: 0.1496 (0.1809)
loss_mask: 0.4180 (0.4609)  loss_objectness: 0.0969 (0.1306)
loss_rpn_box_reg: 0.0226 (0.0258)  time: 0.4066  data: 0.0087  max
mem: 6183
Epoch: [1]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.7235 (0.9345)
loss_classifier: 0.1133 (0.1602)  loss_box_reg: 0.1133 (0.1743)
loss_mask: 0.3708 (0.4501)  loss_objectness: 0.0819 (0.1242)
loss_rpn_box_reg: 0.0158 (0.0257)  time: 0.4098  data: 0.0083  max
mem: 6183
Epoch: [1] Total time: 0:00:25 (0.4228 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:24  model_time: 0.1586 (0.1586)
evaluator_time: 0.0226 (0.0226)  time: 0.4891  data: 0.3065  max mem:
6183
Test:  [49/50]  eta: 0:00:00  model_time: 0.0857 (0.0932)
evaluator_time: 0.0095 (0.0121)  time: 0.1026  data: 0.0037  max mem:
6183
Test: Total time: 0:00:06 (0.1200 s / it)
Averaged stats: model_time: 0.0857 (0.0932)  evaluator_time: 0.0095
(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 ] = 0.187
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.545
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.025
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.001
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.199
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.110
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.376
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.390
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.014
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.413
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.195
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.625
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.017
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.218
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.120
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.308
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.318
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.043
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.335
Epoch: [2]  [ 0/60]  eta: 0:01:00  lr: 0.005000  loss: 1.1471 (1.1471)
loss_classifier: 0.1634 (0.1634)  loss_box_reg: 0.1893 (0.1893)
loss_mask: 0.6610 (0.6610)  loss_objectness: 0.0976 (0.0976)
loss_rpn_box_reg: 0.0358 (0.0358)  time: 1.0152  data: 0.3117  max
mem: 6183
Epoch: [2]  [10/60]  eta: 0:00:24  lr: 0.005000  loss: 0.7911 (0.8399)
loss_classifier: 0.1262 (0.1248)  loss_box_reg: 0.1725 (0.1754)
loss_mask: 0.4143 (0.4348)  loss_objectness: 0.0736 (0.0780)
loss_rpn_box_reg: 0.0235 (0.0269)  time: 0.4916  data: 0.0387  max
mem: 6183
Epoch: [2]  [20/60]  eta: 0:00:19  lr: 0.005000  loss: 0.7463 (0.7800)
loss_classifier: 0.0967 (0.1107)  loss_box_reg: 0.1487 (0.1608)
loss_mask: 0.3866 (0.4070)  loss_objectness: 0.0597 (0.0760)
loss_rpn_box_reg: 0.0211 (0.0256)  time: 0.4530  data: 0.0129  max
mem: 6183
Epoch: [2]  [30/60]  eta: 0:00:13  lr: 0.005000  loss: 0.7201 (0.7765)
loss_classifier: 0.0961 (0.1138)  loss_box_reg: 0.1487 (0.1704)
loss_mask: 0.3622 (0.3954)  loss_objectness: 0.0534 (0.0705)
loss_rpn_box_reg: 0.0237 (0.0265)  time: 0.4463  data: 0.0119  max
mem: 6183
Epoch: [2]  [40/60]  eta: 0:00:09  lr: 0.005000  loss: 0.7201 (0.7566)
loss_classifier: 0.0987 (0.1097)  loss_box_reg: 0.1438 (0.1641)
loss_mask: 0.3633 (0.3919)  loss_objectness: 0.0515 (0.0656)
loss_rpn_box_reg: 0.0239 (0.0253)  time: 0.4342  data: 0.0099  max
mem: 6183
Epoch: [2]  [50/60]  eta: 0:00:04  lr: 0.005000  loss: 0.7464 (0.7527)
loss_classifier: 0.0987 (0.1097)  loss_box_reg: 0.1450 (0.1646)
loss_mask: 0.3633 (0.3901)  loss_objectness: 0.0465 (0.0625)
loss_rpn_box_reg: 0.0239 (0.0259)  time: 0.4306  data: 0.0099  max
mem: 6183
Epoch: [2]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.7483 (0.7566)
loss_classifier: 0.1072 (0.1100)  loss_box_reg: 0.1488 (0.1696)
loss_mask: 0.3710 (0.3897)  loss_objectness: 0.0465 (0.0600)
loss_rpn_box_reg: 0.0305 (0.0274)  time: 0.4229  data: 0.0085  max
mem: 6183
Epoch: [2] Total time: 0:00:26 (0.4492 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:36  model_time: 0.2338 (0.2338)
evaluator_time: 0.0356 (0.0356)  time: 0.7262  data: 0.4553  max mem:
6183
Test:  [49/50]  eta: 0:00:00  model_time: 0.1016 (0.1135)
evaluator_time: 0.0135 (0.0187)  time: 0.1327  data: 0.0037  max mem:
6183
Test: Total time: 0:00:07 (0.1503 s / it)
Averaged stats: model_time: 0.1016 (0.1135)  evaluator_time: 0.0135
(0.0187)
```

```
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.172
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.541
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.033
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.004
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.186
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.087
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.413
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.426
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.029
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.451
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.229
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.662
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.061
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.005
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.253
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.142
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.366
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.378
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
```

```
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.114
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.395
Epoch: [3]  [ 0/60]  eta: 0:00:52  lr: 0.000500  loss: 0.8332 (0.8332)
loss_classifier: 0.1469 (0.1469)  loss_box_reg: 0.2316 (0.2316)
loss_mask: 0.3584 (0.3584)  loss_objectness: 0.0625 (0.0625)
loss_rpn_box_reg: 0.0337 (0.0337)  time: 0.8698  data: 0.3433  max
mem: 6183
Epoch: [3]  [10/60]  eta: 0:00:23  lr: 0.000500  loss: 0.6265 (0.6456)
loss_classifier: 0.0793 (0.0893)  loss_box_reg: 0.1117 (0.1371)
loss_mask: 0.3481 (0.3566)  loss_objectness: 0.0410 (0.0410)
loss_rpn_box_reg: 0.0198 (0.0216)  time: 0.4714  data: 0.0408  max
mem: 6183
Epoch: [3]  [20/60]  eta: 0:00:18  lr: 0.000500  loss: 0.7121 (0.7147)
loss_classifier: 0.0970 (0.1055)  loss_box_reg: 0.1546 (0.1658)
loss_mask: 0.3503 (0.3704)  loss_objectness: 0.0410 (0.0477)
loss_rpn_box_reg: 0.0236 (0.0254)  time: 0.4460  data: 0.0113  max
mem: 6183
Epoch: [3]  [30/60]  eta: 0:00:13  lr: 0.000500  loss: 0.7439 (0.7137)
loss_classifier: 0.1094 (0.1074)  loss_box_reg: 0.1584 (0.1659)
loss_mask: 0.3725 (0.3671)  loss_objectness: 0.0466 (0.0475)
loss_rpn_box_reg: 0.0239 (0.0258)  time: 0.4461  data: 0.0108  max
mem: 6183
Epoch: [3]  [40/60]  eta: 0:00:08  lr: 0.000500  loss: 0.6234 (0.6886)
loss_classifier: 0.0945 (0.1037)  loss_box_reg: 0.1393 (0.1584)
loss_mask: 0.3224 (0.3561)  loss_objectness: 0.0407 (0.0465)
loss_rpn_box_reg: 0.0193 (0.0238)  time: 0.4300  data: 0.0094  max
mem: 6183
Epoch: [3]  [50/60]  eta: 0:00:04  lr: 0.000500  loss: 0.5540 (0.6643)
loss_classifier: 0.0719 (0.0975)  loss_box_reg: 0.0995 (0.1491)
loss_mask: 0.3218 (0.3502)  loss_objectness: 0.0392 (0.0457)
loss_rpn_box_reg: 0.0130 (0.0218)  time: 0.4317  data: 0.0108  max
mem: 6183
Epoch: [3]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.5885 (0.6620)
loss_classifier: 0.0780 (0.0969)  loss_box_reg: 0.1203 (0.1482)
loss_mask: 0.3260 (0.3488)  loss_objectness: 0.0382 (0.0466)
loss_rpn_box_reg: 0.0165 (0.0215)  time: 0.4255  data: 0.0102  max
mem: 6183
Epoch: [3] Total time: 0:00:26 (0.4436 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:24  model_time: 0.1375 (0.1375)
evaluator_time: 0.0135 (0.0135)  time: 0.4847  data: 0.3322  max mem:
6183
Test:  [49/50]  eta: 0:00:00  model_time: 0.1000 (0.1176)
evaluator_time: 0.0107 (0.0175)  time: 0.1229  data: 0.0038  max mem:
6183
```

```
Test: Total time: 0:00:07 (0.1531 s / it)
Averaged stats: model_time: 0.1000 (0.1176)  evaluator_time: 0.0107
(0.0175)
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 ] = 0.321
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.691
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.228
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.002
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.343
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.157
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.495
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.503
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.014
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.534
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.266
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.701
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.084
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.007
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.298
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.149
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.367
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.370
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.086
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.388
Epoch: [4]  [ 0/60]  eta: 0:01:00  lr: 0.000500  loss: 0.8458 (0.8458)
loss_classifier: 0.1525 (0.1525)  loss_box_reg: 0.2353 (0.2353)
loss_mask: 0.3796 (0.3796)  loss_objectness: 0.0449 (0.0449)
loss_rpn_box_reg: 0.0335 (0.0335)  time: 1.0156  data: 0.4515  max
mem: 6183
Epoch: [4]  [10/60]  eta: 0:00:23  lr: 0.000500  loss: 0.6267 (0.6354)
loss_classifier: 0.0832 (0.0913)  loss_box_reg: 0.1378 (0.1425)
loss_mask: 0.3424 (0.3462)  loss_objectness: 0.0352 (0.0376)
loss_rpn_box_reg: 0.0161 (0.0179)  time: 0.4774  data: 0.0477  max
mem: 6183
Epoch: [4]  [20/60]  eta: 0:00:18  lr: 0.000500  loss: 0.6267 (0.6643)
loss_classifier: 0.0832 (0.0973)  loss_box_reg: 0.1378 (0.1537)
loss_mask: 0.3354 (0.3528)  loss_objectness: 0.0357 (0.0426)
loss_rpn_box_reg: 0.0145 (0.0179)  time: 0.4387  data: 0.0098  max
mem: 6183
Epoch: [4]  [30/60]  eta: 0:00:13  lr: 0.000500  loss: 0.6212 (0.6537)
loss_classifier: 0.0890 (0.0946)  loss_box_reg: 0.1292 (0.1488)
loss_mask: 0.3219 (0.3465)  loss_objectness: 0.0441 (0.0441)
loss_rpn_box_reg: 0.0165 (0.0196)  time: 0.4395  data: 0.0110  max
mem: 6183
Epoch: [4]  [40/60]  eta: 0:00:08  lr: 0.000500  loss: 0.5359 (0.6376)
loss_classifier: 0.0712 (0.0911)  loss_box_reg: 0.1084 (0.1442)
loss_mask: 0.3119 (0.3369)  loss_objectness: 0.0391 (0.0457)
loss_rpn_box_reg: 0.0183 (0.0197)  time: 0.4222  data: 0.0092  max
mem: 6183
Epoch: [4]  [50/60]  eta: 0:00:04  lr: 0.000500  loss: 0.5708 (0.6422)
loss_classifier: 0.0751 (0.0916)  loss_box_reg: 0.1189 (0.1453)
loss_mask: 0.3252 (0.3399)  loss_objectness: 0.0331 (0.0447)
loss_rpn_box_reg: 0.0198 (0.0207)  time: 0.4346  data: 0.0093  max
mem: 6183
Epoch: [4]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.6419 (0.6426)
loss_classifier: 0.0798 (0.0911)  loss_box_reg: 0.1289 (0.1465)
loss_mask: 0.3501 (0.3412)  loss_objectness: 0.0314 (0.0432)
loss_rpn_box_reg: 0.0219 (0.0207)  time: 0.4379  data: 0.0092  max
mem: 6183
Epoch: [4] Total time: 0:00:26 (0.4462 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:24  model_time: 0.1605 (0.1605)
evaluator_time: 0.0165 (0.0165)  time: 0.4862  data: 0.3077  max mem:
6183
```

```
Test:  [49/50]  eta: 0:00:00  model_time: 0.1222 (0.1189)
evaluator_time: 0.0208 (0.0195)  time: 0.1653  data: 0.0063  max mem:
6183
Test: Total time: 0:00:07 (0.1563 s / it)
Averaged stats: model_time: 0.1222 (0.1189)  evaluator_time: 0.0208
(0.0195)
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 ] = 0.308
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.698
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.206
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.007
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.327
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.164
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.502
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.505
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.029
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.535
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.262
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.667
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.080
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.003
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.285
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.163
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.370
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.376
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.071
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.395
Epoch: [5]  [ 0/60]  eta: 0:01:03  lr: 0.000500  loss: 0.7245 (0.7245)
loss_classifier: 0.1085 (0.1085)  loss_box_reg: 0.1718 (0.1718)
loss_mask: 0.3913 (0.3913)  loss_objectness: 0.0277 (0.0277)
loss_rpn_box_reg: 0.0250 (0.0250)  time: 1.0635  data: 0.4115  max
mem: 6183
Epoch: [5]  [10/60]  eta: 0:00:24  lr: 0.000500  loss: 0.6815 (0.7052)
loss_classifier: 0.1052 (0.1035)  loss_box_reg: 0.1591 (0.1632)
loss_mask: 0.3807 (0.3729)  loss_objectness: 0.0422 (0.0459)
loss_rpn_box_reg: 0.0184 (0.0196)  time: 0.4903  data: 0.0447  max
mem: 6183
Epoch: [5]  [20/60]  eta: 0:00:18  lr: 0.000500  loss: 0.6325 (0.6620)
loss_classifier: 0.0885 (0.0937)  loss_box_reg: 0.1364 (0.1454)
loss_mask: 0.3398 (0.3554)  loss_objectness: 0.0422 (0.0444)
loss_rpn_box_reg: 0.0203 (0.0231)  time: 0.4343  data: 0.0090  max
mem: 6183
Epoch: [5]  [30/60]  eta: 0:00:13  lr: 0.000500  loss: 0.5582 (0.6269)
loss_classifier: 0.0639 (0.0864)  loss_box_reg: 0.1032 (0.1318)
loss_mask: 0.3040 (0.3465)  loss_objectness: 0.0346 (0.0419)
loss_rpn_box_reg: 0.0174 (0.0204)  time: 0.4385  data: 0.0100  max
mem: 6183
Epoch: [5]  [40/60]  eta: 0:00:08  lr: 0.000500  loss: 0.5365 (0.6248)
loss_classifier: 0.0671 (0.0859)  loss_box_reg: 0.1032 (0.1328)
loss_mask: 0.3096 (0.3436)  loss_objectness: 0.0346 (0.0422)
loss_rpn_box_reg: 0.0140 (0.0203)  time: 0.4347  data: 0.0098  max
mem: 6183
Epoch: [5]  [50/60]  eta: 0:00:04  lr: 0.000500  loss: 0.6168 (0.6271)
loss_classifier: 0.0794 (0.0864)  loss_box_reg: 0.1335 (0.1345)
loss_mask: 0.3255 (0.3438)  loss_objectness: 0.0386 (0.0422)
loss_rpn_box_reg: 0.0183 (0.0202)  time: 0.4273  data: 0.0098  max
mem: 6183
Epoch: [5]  [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.6168 (0.6215)
loss_classifier: 0.0864 (0.0869)  loss_box_reg: 0.1335 (0.1357)
loss_mask: 0.3076 (0.3368)  loss_objectness: 0.0361 (0.0421)
loss_rpn_box_reg: 0.0190 (0.0200)  time: 0.4290  data: 0.0089  max
mem: 6183
Epoch: [5] Total time: 0:00:26 (0.4488 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:24  model_time: 0.1630 (0.1630)
```

```
evaluator_time: 0.0140 (0.0140)  time: 0.4981  data: 0.3196  max mem:
6183
Test:  [49/50]  eta: 0:00:00  model_time: 0.1020 (0.1077)
evaluator_time: 0.0105 (0.0132)  time: 0.1234  data: 0.0040  max mem:
6183
Test: Total time: 0:00:06 (0.1360 s / it)
Averaged stats: model_time: 0.1020 (0.1077)  evaluator_time: 0.0105
(0.0132)
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 ] = 0.313
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.719
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.234
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.006
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.335
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.161
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.490
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.497
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.029
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.527
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.278
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.718
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.123
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.008
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
```

```
maxDets=100 ] = 0.309
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.161
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.392
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.396
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.114
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.413
Epoch: [6]  [ 0/60]  eta: 0:00:57  lr: 0.000050  loss: 0.7117 (0.7117)
loss_classifier: 0.1020 (0.1020)  loss_box_reg: 0.2017 (0.2017)
loss_mask: 0.3243 (0.3243)  loss_objectness: 0.0519 (0.0519)
loss_rpn_box_reg: 0.0317 (0.0317)  time: 0.9569  data: 0.3761  max
mem: 6183
Epoch: [6]  [10/60]  eta: 0:00:25  lr: 0.000050  loss: 0.6789 (0.6812)
loss_classifier: 0.0925 (0.0971)  loss_box_reg: 0.1653 (0.1522)
loss_mask: 0.3516 (0.3547)  loss_objectness: 0.0468 (0.0554)
loss_rpn_box_reg: 0.0201 (0.0217)  time: 0.5068  data: 0.0427  max
mem: 6183
Epoch: [6]  [20/60]  eta: 0:00:18  lr: 0.000050  loss: 0.6227 (0.6372)
loss_classifier: 0.0839 (0.0902)  loss_box_reg: 0.1162 (0.1417)
loss_mask: 0.3342 (0.3348)  loss_objectness: 0.0404 (0.0504)
loss_rpn_box_reg: 0.0188 (0.0201)  time: 0.4423  data: 0.0091  max
mem: 6183
Epoch: [6]  [30/60]  eta: 0:00:13  lr: 0.000050  loss: 0.5788 (0.6376)
loss_classifier: 0.0795 (0.0867)  loss_box_reg: 0.1121 (0.1373)
loss_mask: 0.3030 (0.3438)  loss_objectness: 0.0400 (0.0489)
loss_rpn_box_reg: 0.0196 (0.0210)  time: 0.4211  data: 0.0091  max
mem: 6183
Epoch: [6]  [40/60]  eta: 0:00:09  lr: 0.000050  loss: 0.5370 (0.6097)
loss_classifier: 0.0746 (0.0827)  loss_box_reg: 0.1006 (0.1308)
loss_mask: 0.3002 (0.3321)  loss_objectness: 0.0322 (0.0441)
loss_rpn_box_reg: 0.0200 (0.0200)  time: 0.4399  data: 0.0099  max
mem: 6183
Epoch: [6]  [50/60]  eta: 0:00:04  lr: 0.000050  loss: 0.5223 (0.6096)
loss_classifier: 0.0680 (0.0837)  loss_box_reg: 0.1117 (0.1326)
loss_mask: 0.2959 (0.3314)  loss_objectness: 0.0298 (0.0428)
loss_rpn_box_reg: 0.0132 (0.0191)  time: 0.4486  data: 0.0102  max
mem: 6183
Epoch: [6]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.5875 (0.6110)
loss_classifier: 0.0843 (0.0846)  loss_box_reg: 0.1176 (0.1345)
loss_mask: 0.3157 (0.3312)  loss_objectness: 0.0330 (0.0417)
loss_rpn_box_reg: 0.0158 (0.0189)  time: 0.4291  data: 0.0089  max
mem: 6183
Epoch: [6] Total time: 0:00:26 (0.4486 s / it)
```

```
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:37  model_time: 0.2324 (0.2324)
evaluator_time: 0.0248 (0.0248)  time: 0.7444  data: 0.4851  max mem:
6183
Test:  [49/50]  eta: 0:00:00  model_time: 0.1017 (0.1117)
evaluator_time: 0.0113 (0.0136)  time: 0.1266  data: 0.0040  max mem:
6183
Test: Total time: 0:00:07 (0.1438 s / it)
Averaged stats: model_time: 0.1017 (0.1117)  evaluator_time: 0.0113
(0.0136)
Accumulating evaluation results...
DONE (t=0.02s).
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.302
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.685
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.205
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.324
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.173
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.483
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.492
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.523
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.275
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.713
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.104
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.006
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.307
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.169
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.385
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.388
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.100
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.406
Epoch: [7]  [ 0/60]  eta: 0:00:57  lr: 0.000050  loss: 0.5761 (0.5761)
loss_classifier: 0.0837 (0.0837)  loss_box_reg: 0.1236 (0.1236)
loss_mask: 0.3147 (0.3147)  loss_objectness: 0.0353 (0.0353)
loss_rpn_box_reg: 0.0188 (0.0188)  time: 0.9569  data: 0.4152  max
mem: 6183
Epoch: [7]  [10/60]  eta: 0:00:24  lr: 0.000050  loss: 0.5761 (0.6286)
loss_classifier: 0.0820 (0.0869)  loss_box_reg: 0.1268 (0.1465)
loss_mask: 0.3147 (0.3304)  loss_objectness: 0.0359 (0.0445)
loss_rpn_box_reg: 0.0174 (0.0203)  time: 0.4945  data: 0.0456  max
mem: 6183
Epoch: [7]  [20/60]  eta: 0:00:18  lr: 0.000050  loss: 0.5666 (0.6401)
loss_classifier: 0.0784 (0.0890)  loss_box_reg: 0.1336 (0.1466)
loss_mask: 0.3140 (0.3366)  loss_objectness: 0.0397 (0.0465)
loss_rpn_box_reg: 0.0226 (0.0215)  time: 0.4492  data: 0.0097  max
mem: 6183
Epoch: [7]  [30/60]  eta: 0:00:13  lr: 0.000050  loss: 0.5603 (0.6153)
loss_classifier: 0.0743 (0.0847)  loss_box_reg: 0.1095 (0.1369)
loss_mask: 0.3106 (0.3308)  loss_objectness: 0.0374 (0.0432)
loss_rpn_box_reg: 0.0208 (0.0197)  time: 0.4340  data: 0.0095  max
mem: 6183
Epoch: [7]  [40/60]  eta: 0:00:09  lr: 0.000050  loss: 0.5922 (0.6248)
loss_classifier: 0.0779 (0.0881)  loss_box_reg: 0.1081 (0.1423)
loss_mask: 0.3125 (0.3308)  loss_objectness: 0.0364 (0.0438)
loss_rpn_box_reg: 0.0184 (0.0198)  time: 0.4351  data: 0.0083  max
mem: 6183
Epoch: [7]  [50/60]  eta: 0:00:04  lr: 0.000050  loss: 0.6158 (0.6204)
loss_classifier: 0.0936 (0.0866)  loss_box_reg: 0.1060 (0.1399)
loss_mask: 0.3191 (0.3313)  loss_objectness: 0.0408 (0.0430)
loss_rpn_box_reg: 0.0184 (0.0197)  time: 0.4539  data: 0.0096  max
mem: 6183
Epoch: [7]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.5649 (0.6236)
loss_classifier: 0.0800 (0.0864)  loss_box_reg: 0.1205 (0.1402)
loss_mask: 0.3216 (0.3355)  loss_objectness: 0.0391 (0.0422)
```

```
loss_rpn_box_reg: 0.0162 (0.0195)  time: 0.4381  data: 0.0091  max
mem: 6183
Epoch: [7] Total time: 0:00:27 (0.4523 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:24  model_time: 0.1626 (0.1626)
evaluator_time: 0.0159 (0.0159)  time: 0.4915  data: 0.3013  max mem:
6183
Test:  [49/50]  eta: 0:00:00  model_time: 0.1015 (0.1229)
evaluator_time: 0.0091 (0.0170)  time: 0.1260  data: 0.0040  max mem:
6183
Test: Total time: 0:00:07 (0.1579 s / it)
Averaged stats: model_time: 0.1015 (0.1229)  evaluator_time: 0.0091
(0.0170)
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 ] = 0.307
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.710
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.174
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.007
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.327
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.178
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.482
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.494
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.043
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.522
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.279
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.691
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
```

```
maxDets=100 ] = 0.111
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.007
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.312
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.171
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.388
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.391
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.129
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.407
Epoch: [8]  [ 0/60]  eta: 0:00:55  lr: 0.000050  loss: 0.6837 (0.6837)
loss_classifier: 0.1019 (0.1019)  loss_box_reg: 0.1832 (0.1832)
loss_mask: 0.3461 (0.3461)  loss_objectness: 0.0296 (0.0296)
loss_rpn_box_reg: 0.0229 (0.0229)  time: 0.9262  data: 0.3697  max
mem: 6183
Epoch: [8]  [10/60]  eta: 0:00:23  lr: 0.000050  loss: 0.5116 (0.5398)
loss_classifier: 0.0680 (0.0746)  loss_box_reg: 0.1039 (0.1144)
loss_mask: 0.2978 (0.2984)  loss_objectness: 0.0340 (0.0349)
loss_rpn_box_reg: 0.0143 (0.0175)  time: 0.4659  data: 0.0402  max
mem: 6183
Epoch: [8]  [20/60]  eta: 0:00:18  lr: 0.000050  loss: 0.4972 (0.5452)
loss_classifier: 0.0680 (0.0756)  loss_box_reg: 0.0901 (0.1090)
loss_mask: 0.2931 (0.3038)  loss_objectness: 0.0340 (0.0404)
loss_rpn_box_reg: 0.0143 (0.0164)  time: 0.4350  data: 0.0083  max
mem: 6183
Epoch: [8]  [30/60]  eta: 0:00:13  lr: 0.000050  loss: 0.5934 (0.5805)
loss_classifier: 0.0765 (0.0822)  loss_box_reg: 0.1062 (0.1195)
loss_mask: 0.3194 (0.3211)  loss_objectness: 0.0365 (0.0406)
loss_rpn_box_reg: 0.0148 (0.0171)  time: 0.4453  data: 0.0089  max
mem: 6183
Epoch: [8]  [40/60]  eta: 0:00:09  lr: 0.000050  loss: 0.6306 (0.6022)
loss_classifier: 0.0970 (0.0869)  loss_box_reg: 0.1469 (0.1339)
loss_mask: 0.3323 (0.3207)  loss_objectness: 0.0382 (0.0420)
loss_rpn_box_reg: 0.0218 (0.0188)  time: 0.4470  data: 0.0085  max
mem: 6244
Epoch: [8]  [50/60]  eta: 0:00:04  lr: 0.000050  loss: 0.6306 (0.6087)
loss_classifier: 0.0890 (0.0869)  loss_box_reg: 0.1527 (0.1359)
loss_mask: 0.3323 (0.3256)  loss_objectness: 0.0343 (0.0415)
loss_rpn_box_reg: 0.0217 (0.0189)  time: 0.4622  data: 0.0119  max
mem: 6244
```

```
Epoch: [8]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.6214 (0.6103)
loss_classifier: 0.0828 (0.0869)  loss_box_reg: 0.1358 (0.1360)
loss_mask: 0.3344 (0.3274)  loss_objectness: 0.0327 (0.0411)
loss_rpn_box_reg: 0.0187 (0.0189)  time: 0.4503  data: 0.0115  max
mem: 6244
Epoch: [8] Total time: 0:00:27 (0.4548 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:24  model_time: 0.2058 (0.2058)
evaluator_time: 0.0193 (0.0193)  time: 0.4965  data: 0.2696  max mem:
6244
Test:  [49/50]  eta: 0:00:00  model_time: 0.1229 (0.1261)
evaluator_time: 0.0158 (0.0189)  time: 0.1597  data: 0.0079  max mem:
6244
Test: Total time: 0:00:08 (0.1621 s / it)
Averaged stats: model_time: 0.1229 (0.1261)  evaluator_time: 0.0158
(0.0189)
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 ] = 0.337
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.702
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.241
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.002
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.361
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.187
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.518
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.528
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.014
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.560
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.274
```

```
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.691
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.111
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.007
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.303
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.163
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.382
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.383
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.143
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.398
Epoch: [9]  [ 0/60]  eta: 0:00:55  lr: 0.000005  loss: 0.5576 (0.5576)
loss_classifier: 0.0978 (0.0978)  loss_box_reg: 0.1063 (0.1063)
loss_mask: 0.2981 (0.2981)  loss_objectness: 0.0373 (0.0373)
loss_rpn_box_reg: 0.0181 (0.0181)  time: 0.9230  data: 0.3424  max
mem: 6244
Epoch: [9]  [10/60]  eta: 0:00:23  lr: 0.000005  loss: 0.5576 (0.6111)
loss_classifier: 0.0718 (0.0837)  loss_box_reg: 0.1093 (0.1259)
loss_mask: 0.3419 (0.3431)  loss_objectness: 0.0373 (0.0416)
loss_rpn_box_reg: 0.0177 (0.0169)  time: 0.4736  data: 0.0390  max
mem: 6244
Epoch: [9]  [20/60]  eta: 0:00:18  lr: 0.000005  loss: 0.5708 (0.6228)
loss_classifier: 0.0752 (0.0859)  loss_box_reg: 0.1188 (0.1423)
loss_mask: 0.3121 (0.3319)  loss_objectness: 0.0370 (0.0423)
loss_rpn_box_reg: 0.0177 (0.0203)  time: 0.4455  data: 0.0093  max
mem: 6244
Epoch: [9]  [30/60]  eta: 0:00:13  lr: 0.000005  loss: 0.5708 (0.6139)
loss_classifier: 0.0761 (0.0865)  loss_box_reg: 0.1278 (0.1397)
loss_mask: 0.2954 (0.3253)  loss_objectness: 0.0350 (0.0425)
loss_rpn_box_reg: 0.0193 (0.0198)  time: 0.4619  data: 0.0098  max
mem: 6244
Epoch: [9]  [40/60]  eta: 0:00:09  lr: 0.000005  loss: 0.5281 (0.6200)
loss_classifier: 0.0808 (0.0855)  loss_box_reg: 0.0889 (0.1376)
loss_mask: 0.3113 (0.3339)  loss_objectness: 0.0355 (0.0439)
loss_rpn_box_reg: 0.0201 (0.0192)  time: 0.4407  data: 0.0087  max
mem: 6244
Epoch: [9]  [50/60]  eta: 0:00:04  lr: 0.000005  loss: 0.6068 (0.6288)
loss_classifier: 0.0847 (0.0870)  loss_box_reg: 0.1374 (0.1424)
```

```
loss_mask: 0.3301 (0.3357)  loss_objectness: 0.0403 (0.0443)
loss_rpn_box_reg: 0.0197 (0.0193)  time: 0.4364  data: 0.0092  max
mem: 6244
Epoch: [9]  [59/60]  eta: 0:00:00  lr: 0.000005  loss: 0.5627 (0.6184)
loss_classifier: 0.0730 (0.0846)  loss_box_reg: 0.1200 (0.1382)
loss_mask: 0.3146 (0.3338)  loss_objectness: 0.0403 (0.0433)
loss_rpn_box_reg: 0.0137 (0.0186)  time: 0.4404  data: 0.0092  max
mem: 6244
Epoch: [9] Total time: 0:00:27 (0.4543 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:25  model_time: 0.1727 (0.1727)
evaluator_time: 0.0176 (0.0176)  time: 0.5005  data: 0.3087  max mem:
6244
Test:  [49/50]  eta: 0:00:00  model_time: 0.1152 (0.1169)
evaluator_time: 0.0156 (0.0170)  time: 0.1473  data: 0.0044  max mem:
6244
Test: Total time: 0:00:07 (0.1522 s / it)
Averaged stats: model_time: 0.1152 (0.1169)  evaluator_time: 0.0156
(0.0170)
Accumulating evaluation results...
DONE (t=0.04s).
Accumulating evaluation results...
DONE (t=0.04s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.324
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.718
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.242
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.009
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.345
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.165
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.507
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.519
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.043
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.549
```

```
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.275
 Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.730
 Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.092
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.009
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.305
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.161
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.380
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.382
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.143
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.397
That's it!
```

Comments on training log:

In the above training log, note the last batch of the 10th epoch result.

Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.5627 (0.6184) loss_classifier: 0.0730 (0.0846) loss_box_reg: 0.1200 (0.1382) loss_mask: 0.3146 (0.3338) loss_objectness: 0.0403 (0.0433) loss_rpn_box_reg: 0.0137 (0.0186) time: 0.4404 data: 0.0092 max mem: 6244

Here, the loss value in paranthesis represents the cumulative loss over the entire epoch upto that point(here its the last batch so its for the entire epoch) using the weights after the completion of the 10th epoch.

Similarly for loss_classifier, loss_mask,oss_objectness, loss_rpn_box_reg

These results can be used for comparing the model performance asked in Q5B)

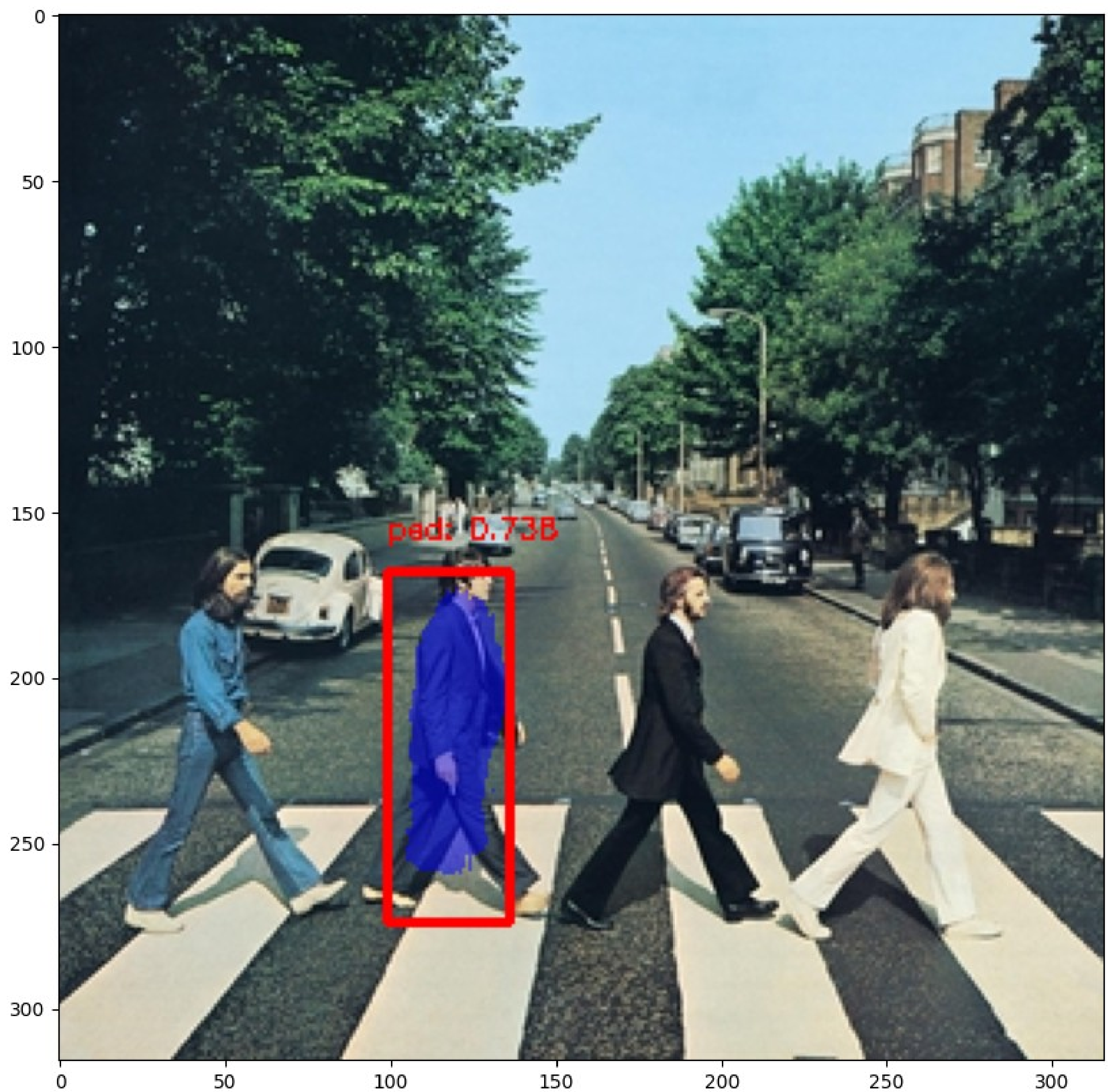
Tetsing of Backbone model on Beatles_Abbey_Road test image (Method 1)

```
import matplotlib.pyplot as plt
import cv2
from torchvision.utils import draw_bounding_boxes,
draw_segmentation_masks

# Read an image from a specified path
```

```
image = read_image("/content/sample_data/Beatles_-_Abbey_Road.jpeg")

# Create an output image to visualize the results
output_image = image

# Obtain an evaluation transformation with 'train=False'
eval_transform = get_transform(train=False)

# Set the model in evaluation mode
model.eval()

with torch.no_grad():
    x = eval_transform(image)

    # convert RGBA -> RGB and move to device
    x = x[:3, ...].to(device)

    # Make predictions using the model
    predictions = model([x, ])
    pred = predictions[0]

# Normalize and convert the image to 8-bit integers (uint8)
image = (255.0 * (image - image.min()) / (image.max() -
image.min())).to(torch.uint8)
image = image[:3, ...]

# Filter predictions based on confidence scores (only keep scores >
0.65)
# mask refers to binary-mask ie true, false of predictions above
confidence( not meaning the mask displayed in the image)
mask = pred["scores"] > 0.65
filtered_pred = {key: value[mask] for key, value in pred.items()}


#Obtaining labels, boxes and masks for filtered predictions
filtered_labels = [f"ped: {score:.3f}" for score in
filtered_pred["scores"]]
filtered_boxes = filtered_pred["boxes"].long()
masks = (filtered_pred["masks"] > 0.7).squeeze(1)

#output image having the filtered prediction masks now
output_image = draw_segmentation_masks(output_image, masks, alpha=0.5,
colors="blue")

# Convert to NumPy array
output_image = output_image.permute(1, 2, 0).cpu().numpy().copy()

#Drawing the boxes, labels using cv2
for label, box in zip(filtered_labels, filtered_boxes):
    x_1, y_1, x_2, y_2 = [coord.item() for coord in box]
```

```
    output_image = cv2.rectangle(output_image, (x_1, y_1), (x_2, y_2),
(255, 0, 0), 2)
    output_image = cv2.putText(output_image, label, (x_1, y_1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.3, (255, 0, 0), 1)

#Plotting the final image
plt.figure(figsize=(10, 10))
plt.imshow(output_image)
```

```
<matplotlib.image.AxesImage at 0x7a4c543b09a0>
```



```
print(pred["scores"])
```

```
tensor([0.7383, 0.5354, 0.5029, 0.4548, 0.3040, 0.2587, 0.0986,
0.0715, 0.0707],
       device='cuda:0')
```

Comments on testing (method 1) result:

In the above method for testing, predictions were inferred from the model. Each prediction dictionary consisted of labels, boxes, masks, confidence scores as keys.

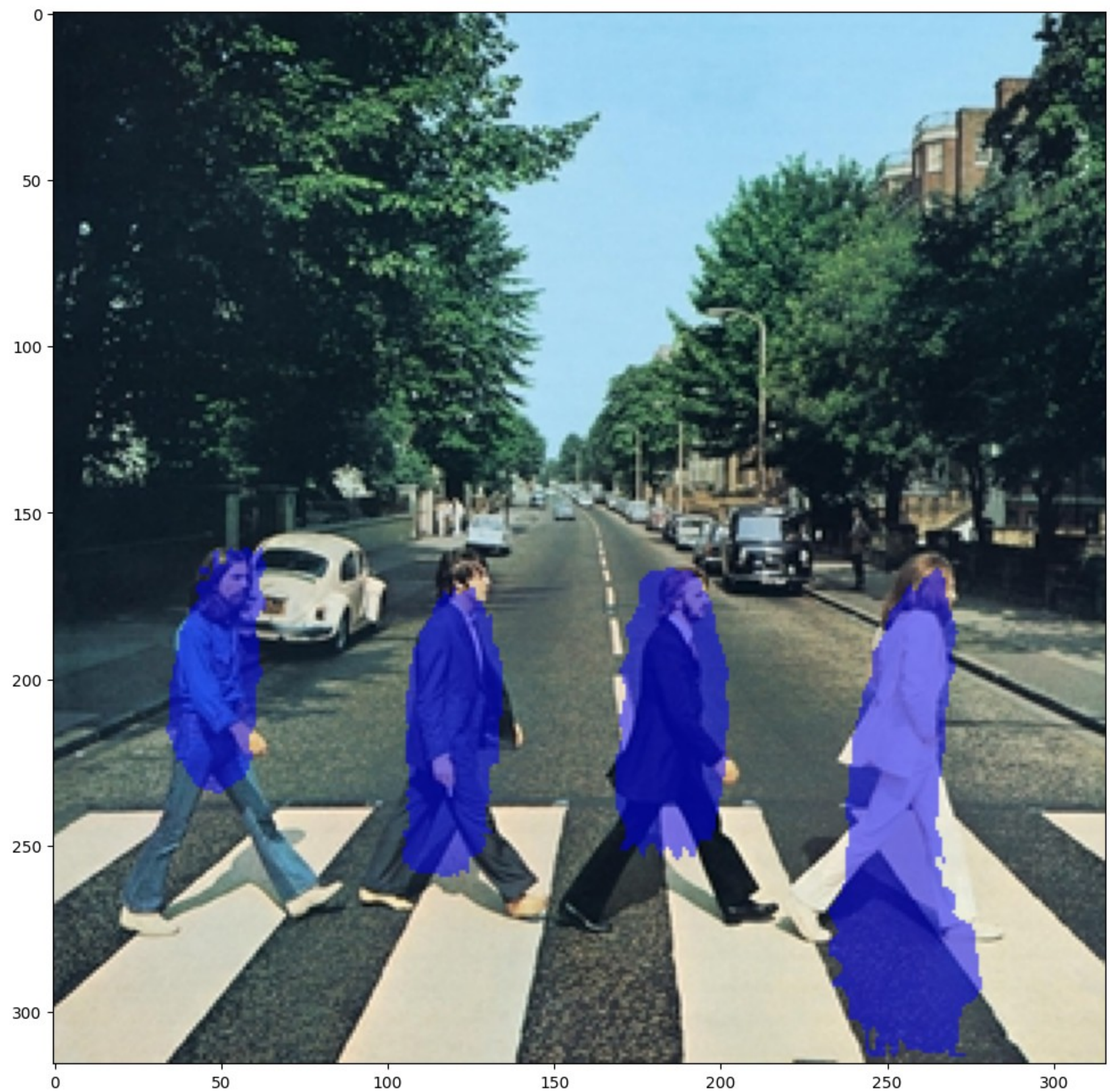
Using confidence score threshold of 0.7, predctions in the list were filtered and the corresponding boxes, labels, masks for the the filtered predictions were outputed. In this case, of all prediction scores (as printed above) only one of them crossed the threshold.

Tetsing of Backbone model on Beatles_Abbey_Road test image (Method 2)

```
import matplotlib.pyplot as plt

from torchvision.utils import draw_bounding_boxes,
draw_segmentation_masks

# Read an image from a specified path
image = read_image("/content/sample_data/Beatles_-_Abbey_Road.jpeg")

# Create an output image to visualize the results
output_image = image

# Obtain an evaluation transformation with 'train=False'
eval_transform = get_transform(train=False)

# Set the model in evaluation mode
model.eval()

with torch.no_grad():
    x = eval_transform(image)

    # convert RGBA -> RGB and move to device
    x = x[:3, ...].to(device)

    # Make predictions using the model
    predictions = model([x, ])
    pred = predictions[0]

# Normalize and convert the image to 8-bit integers (uint8)
image = (255.0 * (image - image.min()) / (image.max() -
image.min())).to(torch.uint8)
image = image[:3, ...]

#Filtering masks based on confidence
masks = (pred["masks"] > 0.7).squeeze(1)
```

```
#output image having the filtered prediction masks now
output_image = draw_segmentation_masks(output_image, masks, alpha=0.5,
colors="blue")

#Plotting the final output image
plt.figure(figsize=(12, 12))
plt.imshow(output_image.permute(1, 2, 0))
```

```
<matplotlib.image.AxesImage at 0x7a4c98c92050>
```



Comments on testing (method 2) result:

In the above method for testing, predictions were inferred from the model.

From each prediction, the corresponding masks whose values exceeded the set threshold were used for the output image.

Prediction dictionaries with low scores like 0.1 etc might have some values in their masks as 0.8 etc (above the threshod). These masks are included in this method but discared in the previous method.

Q5.b)

Performance of two models on the training data after 10 epochs:

Option 1 model backbone resnet performance:

Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.1646 (0.1758) loss_classifier: 0.0197 (0.0221) loss_box_reg: 0.0283 (0.0343) loss_mask: 0.1131 (0.1162) loss_objectness: 0.0003 (0.0008) loss_rpn_box_reg: 0.0018 (0.0024) time: 0.6305 data: 0.0085 max mem: 3779

Option 2 model backbone Mobilenet performance:

Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.5627 (0.6184) loss_classifier: 0.0730 (0.0846) loss_box_reg: 0.1200 (0.1382) loss_mask: 0.3146 (0.3338) loss_objectness: 0.0403 (0.0433) loss_rpn_box_reg: 0.0137 (0.0186) time: 0.4404 data: 0.0092 max mem: 6244

Comparing the loss in paranthesis which represents the training loss of the entire dataset using the weights at the end of the 10th epoch, we can see option 1 model has loss of 0.1758 but option 2 has a loss of 0.6184, similarly loss_classifier for option 1 is 0.0221 but for option 2 is 0.0846. Similarly see for loss_mask, loss_rpn_box etc

Thus option 1 model (resnet backbone) has lesser training loss than option 2 model (mobilenet backbone).

Another way of comparing the performance is the training time.

Option 1 model:

Epoch: [9] Total time: 0:00:35 (0.5994 s / it)

Option 2 model:

Epoch: [9] Total time: 0:00:27 (0.4543 s / it)

In general, in the two model's training logs we can see that option 1 model takes more training time per epoch compared to option 2 model.

Thus to summarize, Option 1 model (Resnet) has more accuracy than option 2 model (Mobilenet). But Option 1 Model (Resnet) takes more training time compared to option 2 model (MobileNet)

Q5C)

Testing method 1:

In testing method 1, we saw that for option 1 model, four predictions crossed the set threshold and correspondigly four pedestrians were detected.

However, for option 2 model, only one prediction crossed the set threshold and correspondingly only one pedestrain was detected, thus missing the remaining pedestrians.

The bounding box co-ordinates and labels scores prediction in option 1 model is also better as the co-ordinates of the box are more or less bounding the pedesetrian correctly that too with a high confidence score.

Testing method 2:

The masks for option 1 model is almost correctly segmenting the four pedestrains

Whereas in option 2 model, some parts of the pedestrain like their legs etc are not masked or some parts of the road is also incorrectly masked.

To summarize, in terms of accuracy, option 1 Model (Resnet) is performing better than option 2 Model (Mobilenet) on the test image.