

ECE-GY 7123 Deep Learning

Assignment -2

Abirami Sivakumar(as16288)

1) Designing convolution filters by hand. Consider an input 2D image and a 3×3 filter (say w) applied to this image. The goal is to guess good filters which implement each of the following elementary image processing operations.

- a. Write down the weights of w which acts as a blurring filter, i.e., the output is a blurry form in the input.*
- b. Write down the weights of w which acts as a sharpening filter in the horizontal direction.*
- c. Write down the weights of w which acts as a sharpening filter in the vertical direction.*
- d. Write down the weights of w which act as a sharpening filter in a diagonal (bottom-left to top-right) direction.*
- e. Give an example of an image operation which cannot be implemented using a 3×3 convolutional filter and briefly explain why.*

Solution:

- a) To create a blurring filter, we want the output to be a smoothed version of the input image. One way to do this is by using a filter that averages the pixel values in the neighbourhood of each pixel.

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

- b) To create a sharpening filter in the horizontal direction, we want to highlight the edges and details that are oriented horizontally in the image. One way to do this is by using a filter that subtracts the average of the neighbouring pixels from the central pixel, and then multiplies the result by -1 for the pixels on the left side of the central pixel and by 1 for the pixels on the right side of the central pixel.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- c) One possible filter for sharpening in the vertical direction is :

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

When this filter is applied to the input image, it will enhance the edges and details that are oriented vertically, while suppressing the details that are oriented horizontally or diagonally. The negative weights above the central pixel ensure that the edges above the central pixel are emphasized by darkening the background, while the positive weights below the central pixel ensure that the edges below the central pixel are emphasized by brightening the foreground.

d) One possible filter is:

$$\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

When this filter is applied to the input image, it will enhance the edges and details that are oriented diagonally, while suppressing the details that are oriented orthogonally. The negative weights on one diagonal ensure that the edges on that diagonal are emphasized by darkening the background, while the positive weights on the other diagonal ensure that the edges on that diagonal are emphasized by brightening the foreground.

- e) An example of an image operation that cannot be implemented using a 3x3 convolutional filter is edge detection using the Canny edge detection algorithm. The Canny edge detection algorithm involves multiple steps, including smoothing the image with a Gaussian filter, finding the gradient magnitude and direction of the image, applying non-maximum suppression to thin the edges, and finally applying hysteresis thresholding to remove false edges. While some of these steps involve convolutions with a 3x3 filter, the overall process cannot be fully implemented with just a 3x3 filter. This is because the algorithm involves operations that require a larger context, such as smoothing with a larger Gaussian filter to remove noise, and non-maximum suppression that involves comparing pixels in a local neighbourhood. Therefore, more complex algorithms such as the Canny edge detection algorithm require multiple convolutions with different filter sizes and shapes, as well as other image processing techniques, to achieve the desired result.

2) Weight decay. The use of l_2 regularization for training multilayer neural networks has a special name: weight decay. Assume an arbitrary dataset $\{(x_i, y_i)\}$ n $i=1$ and a loss function $L(w)$ where w is a vector that represents all the trainable weights (and biases).

a. Write down the l_2 regularized loss, using a weighting parameter λ for the regularizer.

Solution:

The L_2 regularized loss function is:

$$L(w) + \lambda \|w\|^2$$

where $L(w)$ is the original loss function, $\|w\|^2$ is the squared L_2 norm of the weight vector w , and λ is a hyperparameter that controls the strength of the regularization.

The regularizer penalizes large weights by adding their squared magnitude to the loss function. This encourages the model to use smaller weights, which can help prevent overfitting and improve generalization performance. The hyperparameter λ controls the trade-off between fitting the training data well and keeping the weights small. A larger value of λ leads to more regularization and smaller weights.

b. Derive the gradient descent update rules for this loss.

Solution:

To derive the gradient descent update rules for the regularized loss function, we need to compute its gradient with respect to the weight vector w :

$$\nabla L(w) + 2\lambda w$$

where $\nabla L(w)$ is the gradient of the original loss function with respect to w . The update rule for gradient descent with weight decay is then:

$$w \leftarrow w - \eta(\nabla L(w) + 2\lambda w)$$

where η is the learning rate.

This update rule is equivalent to performing regular gradient descent on the regularized loss function with an additional term that penalizes large weights. The λ parameter controls the strength of this penalty, and larger values of λ result in stronger weight decay.

Note that the bias terms are typically not regularized, as they do not contribute to overfitting in the same way as the weights. Therefore, the weight decay penalty is usually applied only to the weight parameters, not the bias terms.

c. Conclude that in each update, the weights are “shrunk” or “decayed” by a multiplicative factor before applying the descent update.

Solution:

We can rewrite the gradient descent update rule for the regularized loss function as follows:

$$w \leftarrow (1 - 2\eta\lambda)w - \eta\nabla L(w)$$

This can be interpreted as a two-step process: first, the weight vector w is “shrunk” or “decayed” by a factor of $(1 - 2\eta\lambda)$, and then the gradient descent update is applied as usual.

The factor $(1 - 2\eta\lambda)$ is less than 1, since λ and η are both positive. Therefore, in each update, the weight vector is multiplied by a factor that is less than 1 before the descent update is applied. This has the effect of shrinking the weights towards zero, which is why this regularization technique is often referred to as weight decay.

The amount of shrinkage depends on the values of λ and η . Larger values of λ or smaller values of η will result in greater shrinkage, while smaller values of λ or larger values of η will result in less shrinkage.

d. What does increasing λ achieve algorithmically, and how should the learning rate be chosen to make the updates stable?

Solution:

Increasing λ achieves algorithmically stronger regularization, which can help prevent overfitting by reducing the model's sensitivity to noise in the training data. However, increasing λ can also make the optimization problem more difficult, so the learning rate should be chosen carefully to make sure that the updates remain stable. In general, a smaller learning rate should be used when λ is larger to prevent the updates from “overshooting” the minimum of the regularized loss function.

3) The IoU metric. In class we defined the IoU metric (or the Jaccard similarity index) for comparing bounding boxes.

a. Using elementary properties of sets, argue that the IoU metric between any two pair of bounding boxes is always a non-negative real number in $[0, 1]$.

Solution:

The IoU metric (Intersection over Union) between any two pair of bounding boxes is defined as the ratio between the area of their intersection and the area of their union.

Let A and B be two sets representing the regions inside the bounding boxes. Then, the area of their intersection is $|A \cap B|$ and the area of their union is $|A \cup B|$.

The intersection of any two sets is a subset of their union,

$$\text{i.e., } A \cap B \subseteq A \cup B.$$

$$\text{Therefore, } |A \cap B| \leq |A \cup B|.$$

Since both the numerator and the denominator of the IoU metric are non-negative real numbers,

it follows that

$$\text{IoU} = |A \cap B| / |A \cup B| \text{ is also a non-negative real number.}$$

Moreover, since the intersection is always a subset of the union,

$$\text{it follows that } 0 \leq |A \cap B| \leq |A \cup B|.$$

Therefore, $0 \leq \text{IoU} \leq 1$ for any two pair of bounding boxes, since the intersection is at most as large as the union.

Therefore, we have shown that the IoU metric between any two pair of bounding boxes is always a non-negative real number in $[0, 1]$.

b. If we represent each bounding box as a function of the top-left and bottom-right coordinates (assume all coordinates are real numbers) then argue that the IoU metric is non-differentiable and hence cannot be directly optimized by gradient descent.

Solution:

The IoU metric is defined as the ratio of the area of the intersection of two bounding boxes to the area of their union. If we represent each bounding box as a function of the top-left and bottom-right coordinates, then the IoU metric can be written as a function of these coordinates.

Let's assume that we have two bounding boxes B1 and B2, with top-left and bottom-right coordinates $(x1, y1, x2, y2)$ and $(x3, y3, x4, y4)$, respectively.

Then the IoU metric can be written as:

$$\text{IoU} = \frac{A1}{A2} = \frac{\max(0, \min(x2, x4) - \max(x1, x3)) * \max(0, \min(y2, y4) - \max(y1, y3))}{(x2 - x1) * (y2 - y1) + (x4 - x3) * (y4 - y3) - A1}$$

where A1 is the area of the intersection of B1 and B2.

This function is not differentiable for all possible inputs. Specifically, the function is not differentiable when the numerator or denominator is equal to zero. This occurs when the two bounding boxes do not intersect (i.e., $A1 = 0$), or when one or both of the bounding boxes have zero area

$$\text{(i.e., } (x2 - x1) = 0 \text{ or } (y2 - y1) = 0 \text{ or } (x4 - x3) = 0 \text{ or } (y4 - y3) = 0).$$

In such cases, the gradient of the IoU metric with respect to the coordinates of the bounding boxes is not well-defined, which makes it impossible to optimize the IoU metric using gradient descent or other gradient-based optimization methods.

▼ Training AlexNet

▼ Abirami Sivakumar

Net ID: as16288

Collaborator : Nishal Sundaraman(ns5429)

```
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/
100% 170498071/170498071 [00:03<00:00,
```



```
# 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.

```
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')
```

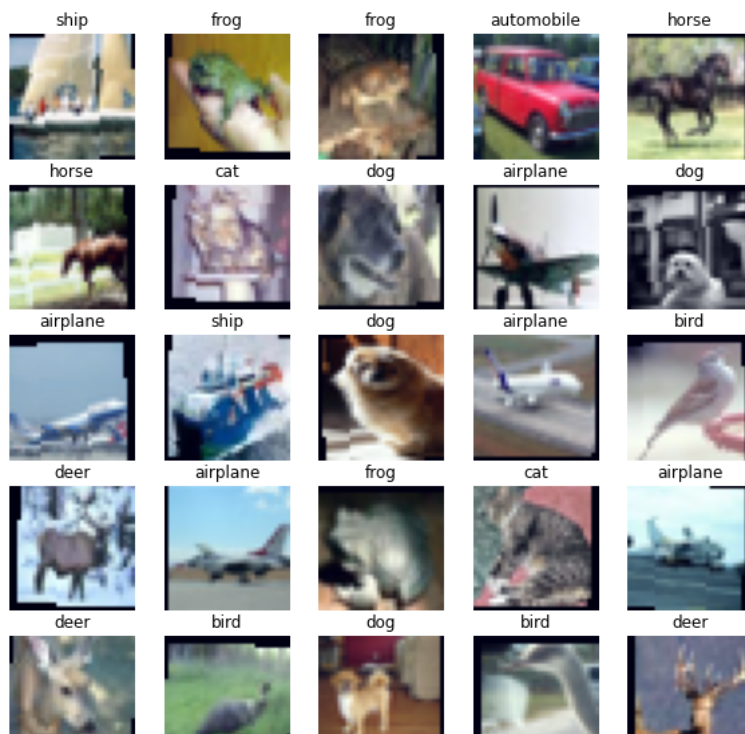
One point here: matplotlib is expecting the values of every pixel to be between [0, 1], however our normalization will cause them to be outside this range. By default matplotlib will then clip these values into the [0, 1] range. This clipping causes all of the images to look a bit weird - all of the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

```
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.

Q1.Create data loaders for train_data, valid_data, test_data. Use batch size 256

```
# Q1: Create data loaders for train_data, valid_data, test_data
# Use batch size 256
from torch.utils.data import DataLoader

BATCH_SIZE = 256

train_iterator = DataLoader(train_data, batch_size = BATCH_SIZE, shuffle = True)
valid_iterator = DataLoader(valid_data, batch_size = BATCH_SIZE, shuffle = True)
test_iterator = DataLoader(test_data, batch_size = BATCH_SIZE, shuffle = True)
```

Q2.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:

◦ $3 \rightarrow 64 \rightarrow 192 \rightarrow 384 \rightarrow 256 \rightarrow 256$

In the end, if everything is correct you should get a feature map of size $2 \times 2 \times 256 = 1024$.

- For the linear layers, the feature sizes are as follows:

◦ $1024 \rightarrow 4096 \rightarrow 4096 \rightarrow 10$.

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

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

        self.features = nn.Sequential(
            # Define according to the steps described above
            nn.Conv2d(3, 64, kernel_size = (3,3), stride = (2,2), padding = (1,1)),
            nn.MaxPool2d(kernel_size = 2, stride = 2),
            nn.ReLU(inplace = True),

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

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

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

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

        self.classifier = nn.Sequential(
            # define according to the steps described above
            nn.Dropout(p=0.5),
            nn.Linear(1024, 4096),
            nn.ReLU(inplace = True),

            nn.Dropout(p=0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace = True),

            nn.Linear(4096, 10)
        )

    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.

```

from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

!ls

gdrive  sample_data

```

Q3: train your model here for 25 epochs

```

# 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
min_valid_loss = 100000
file_path = "/content/sample_tensor.pt"
# Fill training code here
for epoch in range(EPOCHS):
    train_loss, train_acc = train(model, train_iterator, optimizer, criterion, device)
    valid_loss, valid_acc = train(model, valid_iterator, optimizer, criterion, device)
    print("Epoch : ", epoch, "Val_Loss :", valid_loss, "Val_Acc :", valid_acc, "Train_Loss :", train_loss, "Train_Acc", train_acc)
    if(valid_loss < min_valid_loss):
        min_valid_loss = valid_loss
        torch.save(model, file_path)

Epoch : 0 Val_Loss : 1.7978313982486724 Val_Acc : 0.33733915388584135 Train_Loss : 2.5859925442121248 Train_Acc 0.1662011718885465
Epoch : 1 Val_Loss : 1.428245085477829 Val_Acc : 0.4756778493523598 Train_Loss : 1.5788395587693562 Train_Acc 0.41642933237281715

```

```

Epoch : 2 Val_Loss : 1.2177160561084748 Val_Acc : 0.5575367659330368 Train_Loss : 1.3791222003373234 Train_Acc 0.4991974431005391
Epoch : 3 Val_Loss : 1.1004949808120728 Val_Acc : 0.6037798702716828 Train_Loss : 1.2672364312139424 Train_Acc 0.547654474323446
Epoch : 4 Val_Loss : 1.0214611619710923 Val_Acc : 0.6400620400905609 Train_Loss : 1.1772892116145655 Train_Acc 0.5787828479978171
Epoch : 5 Val_Loss : 0.9293188840150833 Val_Acc : 0.6637293189764023 Train_Loss : 1.1123832067982717 Train_Acc 0.6075461645695296
Epoch : 6 Val_Loss : 0.8379776239395141 Val_Acc : 0.703125 Train_Loss : 1.055105608972636 Train_Acc 0.6282093392854388
Epoch : 7 Val_Loss : 0.7695855736732483 Val_Acc : 0.7321691185235977 Train_Loss : 0.9957759370180693 Train_Acc 0.6519442471590909
Epoch : 8 Val_Loss : 0.6811581701040268 Val_Acc : 0.7614545047283172 Train_Loss : 0.9607100544328039 Train_Acc 0.6642045453190804
Epoch : 9 Val_Loss : 0.6314471036195755 Val_Acc : 0.7776769310235977 Train_Loss : 0.9259636517275464 Train_Acc 0.677377485754815
Epoch : 10 Val_Loss : 0.5582055687904358 Val_Acc : 0.8069738060235977 Train_Loss : 0.8823428313163194 Train_Acc 0.6932874643667177
Epoch : 11 Val_Loss : 0.5291714861989021 Val_Acc : 0.8187844663858413 Train_Loss : 0.8616086082025007 Train_Acc 0.7023242187093605
Epoch : 12 Val_Loss : 0.48354094922542573 Val_Acc : 0.8399356633424759 Train_Loss : 0.8311218345029787 Train_Acc 0.712771661918271
Epoch : 13 Val_Loss : 0.43112419098615645 Val_Acc : 0.8584673702716827 Train_Loss : 0.8130627091635357 Train_Acc 0.719644886187531
Epoch : 14 Val_Loss : 0.41389784812927244 Val_Acc : 0.8620519310235977 Train_Loss : 0.797236980362372 Train_Acc 0.7254279117015275
Epoch : 15 Val_Loss : 0.3631882056593895 Val_Acc : 0.8775160849094391 Train_Loss : 0.7881223017519171 Train_Acc 0.7295783023265275
Epoch : 16 Val_Loss : 0.3200647488236427 Val_Acc : 0.8926011025905609 Train_Loss : 0.7551284439184449 Train_Acc 0.7425239699130709
Epoch : 17 Val_Loss : 0.34056638181209564 Val_Acc : 0.8869715064764023 Train_Loss : 0.7410268864848397 Train_Acc 0.745000000027093
Epoch : 18 Val_Loss : 0.2882151618599892 Val_Acc : 0.9028147965669632 Train_Loss : 0.7259788580916144 Train_Acc 0.7519273791800846
Epoch : 19 Val_Loss : 0.2635608486831188 Val_Acc : 0.9208409935235977 Train_Loss : 0.7168971919877962 Train_Acc 0.7550372867421671
Epoch : 20 Val_Loss : 0.27288753241300584 Val_Acc : 0.9132123172283173 Train_Loss : 0.6977655809711326 Train_Acc 0.762798295440999
Epoch : 21 Val_Loss : 0.22725165486335755 Val_Acc : 0.9299517452716828 Train_Loss : 0.6814603693783283 Train_Acc 0.765912641855803
Epoch : 22 Val_Loss : 0.25066274777054787 Val_Acc : 0.9245978862047195 Train_Loss : 0.6682656766338781 Train_Acc 0.772437854923985
Epoch : 23 Val_Loss : 0.21884565316140653 Val_Acc : 0.9324678301811218 Train_Loss : 0.668934525075284 Train_Acc 0.7721058238636364
Epoch : 24 Val_Loss : 0.19763750433921815 Val_Acc : 0.9419117659330368 Train_Loss : 0.647126641632481 Train_Acc 0.7795179330489852

```

✶ 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.

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

```

model = torch.load(file_path)
test_loss, test_acc = evaluate(model, test_iterator, criterion, device)
print("Test Accuracy of the best model is : ", test_acc*100, "%")

```

Also, print out the confusion matrix.

```

Test Accuracy of the best model is : 76.69921875 %

```

```

def get_predictions(model, iterator, device):

```

```

    model.eval()

```

```

    labels = []
    probs = []

```

```

    # Q4: Fill code here.

```

```

    for i, data in enumerate(iterator,0):
        images, label = data
        images = images.to(device)
        label = label.to(device)
        labels.append(label)
        prob, _ = model(images)
        probs.append(prob)

```

```

    labels = torch.cat(labels, dim = 0)
    probs = torch.cat(probs, dim = 0)

```

```

    return labels, probs

```

```

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

```

```

pred_labels = torch.argmax(probs, 1)

```

```

pred_labels = pred_labels.cpu().numpy()
labels = labels.cpu().numpy()

```

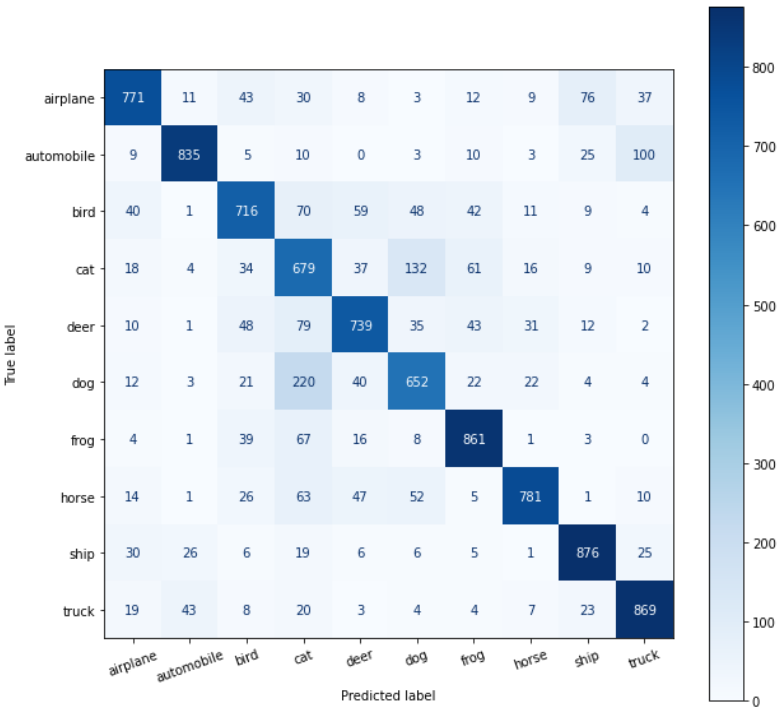
```

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)

plot_confusion_matrix(labels, pred_labels, classes)
```



#Q5. Object Detection

#Abirami Sivakumar

###Net ID: as16288

####Collaborator: Nagharjun Mathi Mariappan(nm4074)

####Based on the tutorial :

https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html

###We will be finetuning a pretrained torchvision object detection model in this problem.

###a. The tutorial shows how to finetune a pretrained model (what they call option 1). Now, follow their approach to add a different backbone (option 2). You can use the same dataset (and code) as the tutorial.

%%shell

```
wget https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
unzip PennFudanPed.zip
```

```
--2023-03-10 22:00:25--
https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip
Resolving www.cis.upenn.edu (www.cis.upenn.edu)... 158.130.69.163,
2607:f470:8:64:5ea5::d
Connecting to www.cis.upenn.edu (www.cis.upenn.edu)|
158.130.69.163|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 53723336 (51M) [application/zip]
Saving to: 'PennFudanPed.zip'
```

```
PennFudanPed.zip 100%[=====>] 51.23M 11.3MB/s in
5.8s
```

```
2023-03-10 22:00:32 (8.88 MB/s) - 'PennFudanPed.zip' saved
[53723336/53723336]
```

```
Archive: PennFudanPed.zip
  creating: PennFudanPed/
  inflating: PennFudanPed/added-object-list.txt
  creating: PennFudanPed/Annotation/
  inflating: PennFudanPed/Annotation/FudanPed00001.txt
  inflating: PennFudanPed/Annotation/FudanPed00002.txt
  inflating: PennFudanPed/Annotation/FudanPed00003.txt
  inflating: PennFudanPed/Annotation/FudanPed00004.txt
  inflating: PennFudanPed/Annotation/FudanPed00005.txt
  inflating: PennFudanPed/Annotation/FudanPed00006.txt
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

extracting: PennFudanPed/PedMasks/PennPed00062_mask.png
inflating: PennFudanPed/PedMasks/PennPed00063_mask.png
inflating: PennFudanPed/PedMasks/PennPed00064_mask.png
inflating: PennFudanPed/PedMasks/PennPed00065_mask.png
extracting: PennFudanPed/PedMasks/PennPed00066_mask.png
inflating: PennFudanPed/PedMasks/PennPed00067_mask.png
inflating: PennFudanPed/PedMasks/PennPed00068_mask.png
extracting: PennFudanPed/PedMasks/PennPed00069_mask.png
extracting: PennFudanPed/PedMasks/PennPed00070_mask.png
inflating: PennFudanPed/PedMasks/PennPed00071_mask.png
extracting: PennFudanPed/PedMasks/PennPed00072_mask.png
inflating: PennFudanPed/PedMasks/PennPed00073_mask.png
extracting: PennFudanPed/PedMasks/PennPed00074_mask.png
inflating: PennFudanPed/PedMasks/PennPed00075_mask.png
inflating: PennFudanPed/PedMasks/PennPed00076_mask.png
inflating: PennFudanPed/PedMasks/PennPed00077_mask.png
inflating: PennFudanPed/PedMasks/PennPed00078_mask.png
inflating: PennFudanPed/PedMasks/PennPed00079_mask.png
inflating: PennFudanPed/PedMasks/PennPed00080_mask.png
inflating: PennFudanPed/PedMasks/PennPed00081_mask.png
inflating: PennFudanPed/PedMasks/PennPed00082_mask.png
extracting: PennFudanPed/PedMasks/PennPed00083_mask.png
extracting: PennFudanPed/PedMasks/PennPed00084_mask.png
inflating: PennFudanPed/PedMasks/PennPed00085_mask.png
extracting: PennFudanPed/PedMasks/PennPed00086_mask.png
inflating: PennFudanPed/PedMasks/PennPed00087_mask.png
inflating: PennFudanPed/PedMasks/PennPed00088_mask.png
extracting: PennFudanPed/PedMasks/PennPed00089_mask.png
extracting: PennFudanPed/PedMasks/PennPed00090_mask.png
inflating: PennFudanPed/PedMasks/PennPed00091_mask.png
inflating: PennFudanPed/PedMasks/PennPed00092_mask.png
inflating: PennFudanPed/PedMasks/PennPed00093_mask.png
inflating: PennFudanPed/PedMasks/PennPed00094_mask.png
inflating: PennFudanPed/PedMasks/PennPed00095_mask.png
extracting: PennFudanPed/PedMasks/PennPed00096_mask.png
creating: PennFudanPed/PNGImages/
inflating: PennFudanPed/PNGImages/FudanPed00001.png
inflating: PennFudanPed/PNGImages/FudanPed00002.png
inflating: PennFudanPed/PNGImages/FudanPed00003.png
inflating: PennFudanPed/PNGImages/FudanPed00004.png
inflating: PennFudanPed/PNGImages/FudanPed00005.png
inflating: PennFudanPed/PNGImages/FudanPed00006.png
inflating: PennFudanPed/PNGImages/FudanPed00007.png
inflating: PennFudanPed/PNGImages/FudanPed00008.png
inflating: PennFudanPed/PNGImages/FudanPed00009.png
inflating: PennFudanPed/PNGImages/FudanPed00010.png
inflating: PennFudanPed/PNGImages/FudanPed00011.png
inflating: PennFudanPed/PNGImages/FudanPed00012.png
inflating: PennFudanPed/PNGImages/FudanPed00013.png
inflating: PennFudanPed/PNGImages/FudanPed00014.png

[illegible]

[illegible]

[illegible]


```
inflating: PennFudanPed/PNGImages/PennPed00091.png
inflating: PennFudanPed/PNGImages/PennPed00092.png
inflating: PennFudanPed/PNGImages/PennPed00093.png
inflating: PennFudanPed/PNGImages/PennPed00094.png
inflating: PennFudanPed/PNGImages/PennPed00095.png
inflating: PennFudanPed/PNGImages/PennPed00096.png
inflating: PennFudanPed/readme.txt
```

%%shell

```
# Download TorchVision repo to use some files from
# references/detection
```

```
git clone https://github.com/pytorch/vision.git
cd vision
git checkout v0.8.2
```

```
cp references/detection/utils.py ../
cp references/detection/transforms.py ../
cp references/detection/coco_eval.py ../
cp references/detection/engine.py ../
cp references/detection/coco_utils.py ../
```

Cloning into 'vision'...

```
remote: Enumerating objects: 316973, done.
remote: Total 316973 (delta 0),
remote: reused 0 (delta 0), pack-reused 316973
make experimental
changes and commit them, and you can discard any commits you make in
this
state without impacting any branches by switching back to a branch.
```

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-c` with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to `false`

HEAD is now at 2f40a483d7 [v0.8.X] .circleci: Add Python 3.9 to CI (#3063)

```

import os
import numpy as np
import torch
from PIL import Image

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 = Image.open(img_path).convert("RGB")
        # note that we haven't converted the mask to RGB,
        # because each color corresponds to a different instance
        # with 0 being background
        mask = Image.open(mask_path)
        # convert the PIL Image into a numpy array
        mask = np.array(mask)
        # instances are encoded as different colors
        obj_ids = np.unique(mask)
        # first id is the background, so remove it
        obj_ids = obj_ids[1:]

        # split the color-encoded mask into a set
        # of binary masks
        masks = mask == obj_ids[:, None, None]

        # get bounding box coordinates for each mask
        num_objs = len(obj_ids)
        boxes = []
        for i in range(num_objs):
            pos = np.where(masks[i])
            xmin = np.min(pos[1])
            xmax = np.max(pos[1])
            ymin = np.min(pos[0])
            ymax = np.max(pos[0])
            boxes.append([xmin, ymin, xmax, ymax])

        # convert everything into a torch.Tensor

```

```

boxes = torch.as_tensor(boxes, dtype=torch.float32)
# there is only one class
labels = torch.ones((num_objs,), dtype=torch.int64)
masks = torch.as_tensor(masks, dtype=torch.uint8)

image_id = torch.tensor([idx])
area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:,
0]))

# suppose all instances are not crowd
iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

target = {}
target["boxes"] = boxes
target["labels"] = labels
target["masks"] = masks
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)

```

####Option 1 - Finetuned model

```

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

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

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

    # now get the number of input features for the mask classifier
    in_features_mask =
    model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256
    # and replace the mask predictor with a new one

```

```

        model.roi_heads.mask_predictor =
MaskRCNNPredictor(in_features_mask,
                                hidden_layer,
                                num_classes)

```

```

    return model

```

```

####Option 2 - Model with a different backbone

```

```

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

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

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

    # let's define what are the feature maps that we will
    # use to perform the region of interest cropping, as well as
    # the size of the crop after rescaling.
    # if your backbone returns a Tensor, featmap_names is expected to
    # be [0]. More generally, the backbone should return an
    # OrderedDict[Tensor], and in featmap_names you can choose which
    # feature maps to use.
    roi_pooler = torchvision.ops.MultiScaleRoIAlign(featmap_names=['0'],
                                                    output_size=7,
                                                    sampling_ratio=2)

    # put the pieces together inside a FasterRCNN model
    model = FasterRCNN(backbone,
                        num_classes=num_classes,
                        rpn_anchor_generator=anchor_generator,
                        box_roi_pool=roi_pooler)

    return model

```

```

from engine import train_one_epoch, evaluate
import utils
import transforms as T

def get_transform(train):
    transforms = []
    # converts the image, a PIL image, into a PyTorch Tensor
    transforms.append(T.ToTensor())
    if train:
        # during training, randomly flip the training images
        # and ground-truth for data augmentation
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)

from engine import train_one_epoch, evaluate
import utils

# 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('PennFudanPed', get_transform(train=True))
dataset_test = PennFudanDataset('PennFudanPed',
get_transform(train=False))

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

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

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

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

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

```

```

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

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

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

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

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

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

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

print("Model without backbone \n")

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("Model with backbone \n")

for epoch in range(num_epochs):
    # train for one epoch, printing every 10 iterations
    train_one_epoch(model_backbone, optimizer_backbone, data_loader,
device, epoch, print_freq=10)
    # update the learning rate

```

```

lr_scheduler_backbone.step()
# evaluate on the test dataset
evaluate(model_backbone, data_loader_test, device=device)

print("That's it!")

/usr/local/lib/python3.9/dist-packages/torch/utils/data/
dataloader.py:554: 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

{"model_id": "c87734474e364c19b89cd2cd0437420c", "version_major": 2, "vers
ion_minor": 0}

Downloading: "https://download.pytorch.org/models/mobilenet_v2-
7ebf99e0.pth" to /root/.cache/torch/hub/checkpoints/mobilenet_v2-
7ebf99e0.pth

{"model_id": "5bce16bf252c42a5be83f4f64dc6110a", "version_major": 2, "vers
ion_minor": 0}

```

Model without backbone

```

Epoch: [0] [ 0/60] eta: 0:07:56 lr: 0.000090 loss: 3.7059 (3.7059)
loss_classifier: 0.6866 (0.6866) loss_box_reg: 0.4806 (0.4806)
loss_mask: 2.5189 (2.5189) loss_objectness: 0.0076 (0.0076)
loss_rpn_box_reg: 0.0122 (0.0122) time: 7.9443 data: 0.3383 max
mem: 2407
Epoch: [0] [10/60] eta: 0:01:00 lr: 0.000936 loss: 1.9065 (2.3671)
loss_classifier: 0.5062 (0.4523) loss_box_reg: 0.2639 (0.3083)
loss_mask: 0.9667 (1.5753) loss_objectness: 0.0250 (0.0261)
loss_rpn_box_reg: 0.0040 (0.0050) time: 1.2173 data: 0.0377 max
mem: 3634
Epoch: [0] [20/60] eta: 0:00:35 lr: 0.001783 loss: 0.9146 (1.5746)
loss_classifier: 0.2370 (0.3246) loss_box_reg: 0.2589 (0.2845)
loss_mask: 0.3416 (0.9400) loss_objectness: 0.0143 (0.0198)
loss_rpn_box_reg: 0.0040 (0.0057) time: 0.5305 data: 0.0082 max
mem: 3634
Epoch: [0] [30/60] eta: 0:00:22 lr: 0.002629 loss: 0.5426 (1.2461)
loss_classifier: 0.1141 (0.2488) loss_box_reg: 0.2108 (0.2611)
loss_mask: 0.2163 (0.7117) loss_objectness: 0.0115 (0.0183)
loss_rpn_box_reg: 0.0055 (0.0063) time: 0.5152 data: 0.0099 max

```

```
mem: 3634
Epoch: [0] [40/60] eta: 0:00:14 lr: 0.003476 loss: 0.4340 (1.0450)
loss_classifier: 0.0653 (0.2012) loss_box_reg: 0.1930 (0.2451)
loss_mask: 0.1704 (0.5771) loss_objectness: 0.0091 (0.0155)
loss_rpn_box_reg: 0.0060 (0.0060) time: 0.5094 data: 0.0104 max
mem: 3634
Epoch: [0] [50/60] eta: 0:00:06 lr: 0.004323 loss: 0.4472 (0.9346)
loss_classifier: 0.0527 (0.1728) loss_box_reg: 0.1930 (0.2380)
loss_mask: 0.1755 (0.5041) loss_objectness: 0.0030 (0.0132)
loss_rpn_box_reg: 0.0065 (0.0066) time: 0.5368 data: 0.0100 max
mem: 3634
Epoch: [0] [59/60] eta: 0:00:00 lr: 0.005000 loss: 0.3810 (0.8430)
loss_classifier: 0.0460 (0.1528) loss_box_reg: 0.1422 (0.2187)
loss_mask: 0.1851 (0.4535) loss_objectness: 0.0028 (0.0117)
loss_rpn_box_reg: 0.0054 (0.0063) time: 0.5431 data: 0.0092 max
mem: 3634
Epoch: [0] Total time: 0:00:39 (0.6536 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:17 model_time: 0.1594 (0.1594)
evaluator_time: 0.0033 (0.0033) time: 0.3460 data: 0.1822 max mem:
3634
Test: [49/50] eta: 0:00:00 model_time: 0.0960 (0.1055)
evaluator_time: 0.0050 (0.0081) time: 0.1156 data: 0.0041 max mem:
3634
Test: Total time: 0:00:06 (0.1258 s / it)
Averaged stats: model_time: 0.0960 (0.1055) evaluator_time: 0.0050
(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 ] = 0.690
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.983
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.897
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 ] = 0.563
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.693
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.333
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
10 ] = 0.747
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
```



```

maxDets=100 ] = 0.747
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 ] = 0.800
Average Recall (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.744
IoU metric: segm
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.700
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.983
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.884
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 ] = 0.395
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.707
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.340
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
10 ] = 0.737
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.737
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 ] = 0.683
Average Recall (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.740
Epoch: [1] [ 0/60] eta: 0:01:05 lr: 0.005000 loss: 0.2435 (0.2435)
loss_classifier: 0.0288 (0.0288) loss_box_reg: 0.0890 (0.0890)
loss_mask: 0.1186 (0.1186) loss_objectness: 0.0011 (0.0011)
loss_rpn_box_reg: 0.0060 (0.0060) time: 1.0936 data: 0.4667 max
mem: 3634
Epoch: [1] [10/60] eta: 0:00:29 lr: 0.005000 loss: 0.2905 (0.3132)
loss_classifier: 0.0409 (0.0450) loss_box_reg: 0.0996 (0.1118)
loss_mask: 0.1335 (0.1477) loss_objectness: 0.0014 (0.0021)
loss_rpn_box_reg: 0.0068 (0.0065) time: 0.5939 data: 0.0511 max
mem: 3634
Epoch: [1] [20/60] eta: 0:00:23 lr: 0.005000 loss: 0.2915 (0.3062)
loss_classifier: 0.0436 (0.0464) loss_box_reg: 0.0979 (0.1013)
loss_mask: 0.1335 (0.1509) loss_objectness: 0.0012 (0.0018)
loss_rpn_box_reg: 0.0058 (0.0058) time: 0.5626 data: 0.0098 max
mem: 3817
Epoch: [1] [30/60] eta: 0:00:16 lr: 0.005000 loss: 0.2752 (0.2841)
loss_classifier: 0.0403 (0.0420) loss_box_reg: 0.0731 (0.0904)
loss_mask: 0.1284 (0.1449) loss_objectness: 0.0007 (0.0017)

```

```
loss_rpn_box_reg: 0.0027 (0.0050)  time: 0.5459  data: 0.0100  max
mem: 3817
Epoch: [1]  [40/60]  eta: 0:00:11  lr: 0.005000  loss: 0.2292 (0.2817)
loss_classifier: 0.0296 (0.0406)  loss_box_reg: 0.0584 (0.0880)
loss_mask: 0.1284 (0.1465)  loss_objectness: 0.0007 (0.0018)
loss_rpn_box_reg: 0.0026 (0.0049)  time: 0.5236  data: 0.0095  max
mem: 3817
Epoch: [1]  [50/60]  eta: 0:00:05  lr: 0.005000  loss: 0.2551 (0.2768)
loss_classifier: 0.0329 (0.0396)  loss_box_reg: 0.0657 (0.0849)
loss_mask: 0.1355 (0.1459)  loss_objectness: 0.0007 (0.0016)
loss_rpn_box_reg: 0.0035 (0.0048)  time: 0.5502  data: 0.0097  max
mem: 3817
Epoch: [1]  [59/60]  eta: 0:00:00  lr: 0.005000  loss: 0.2827 (0.2807)
loss_classifier: 0.0355 (0.0402)  loss_box_reg: 0.0817 (0.0859)
loss_mask: 0.1471 (0.1481)  loss_objectness: 0.0008 (0.0016)
loss_rpn_box_reg: 0.0043 (0.0049)  time: 0.5514  data: 0.0094  max
mem: 3817
Epoch: [1] Total time: 0:00:33 (0.5564 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:17  model_time: 0.1652 (0.1652)
evaluator_time: 0.0039 (0.0039)  time: 0.3491  data: 0.1789  max mem:
3817
Test:  [49/50]  eta: 0:00:00  model_time: 0.0969 (0.1063)
evaluator_time: 0.0040 (0.0074)  time: 0.1110  data: 0.0041  max mem:
3817
Test: Total time: 0:00:06 (0.1267 s / it)
Averaged stats: model_time: 0.0969 (0.1063)  evaluator_time: 0.0040
(0.0074)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision  (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.782
Average Precision  (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.990
Average Precision  (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.943
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 ] = 0.691
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.787
Average Recall     (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.370
Average Recall     (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.829
```

```

Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.829
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 ] = 0.783
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.832
IoU metric: segm
Average Precision   (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.749
Average Precision   (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.990
Average Precision   (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.938
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 ] = 0.474
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 ] = 0.347
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 ] = 0.783
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 ] = 0.683
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.788
Epoch: [2] [ 0/60] eta: 0:00:50  lr: 0.005000  loss: 0.1923 (0.1923)
loss_classifier: 0.0257 (0.0257)  loss_box_reg: 0.0597 (0.0597)
loss_mask: 0.1025 (0.1025)  loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0043 (0.0043)  time: 0.8436  data: 0.2889  max
mem: 3817
Epoch: [2] [10/60] eta: 0:00:28  lr: 0.005000  loss: 0.2278 (0.2301)
loss_classifier: 0.0257 (0.0331)  loss_box_reg: 0.0549 (0.0570)
loss_mask: 0.1408 (0.1355)  loss_objectness: 0.0003 (0.0011)
loss_rpn_box_reg: 0.0033 (0.0035)  time: 0.5744  data: 0.0337  max
mem: 3817
Epoch: [2] [20/60] eta: 0:00:22  lr: 0.005000  loss: 0.2102 (0.2165)
loss_classifier: 0.0298 (0.0305)  loss_box_reg: 0.0478 (0.0544)
loss_mask: 0.1237 (0.1269)  loss_objectness: 0.0011 (0.0015)
loss_rpn_box_reg: 0.0026 (0.0032)  time: 0.5508  data: 0.0091  max
mem: 3817
Epoch: [2] [30/60] eta: 0:00:16  lr: 0.005000  loss: 0.1945 (0.2272)
loss_classifier: 0.0289 (0.0309)  loss_box_reg: 0.0518 (0.0596)

```

```
loss_mask: 0.1192 (0.1313) loss_objectness: 0.0009 (0.0014)
loss_rpn_box_reg: 0.0028 (0.0039) time: 0.5494 data: 0.0100 max
mem: 3817
Epoch: [2] [40/60] eta: 0:00:11 lr: 0.005000 loss: 0.2066 (0.2276)
loss_classifier: 0.0279 (0.0313) loss_box_reg: 0.0582 (0.0602)
loss_mask: 0.1212 (0.1311) loss_objectness: 0.0006 (0.0012)
loss_rpn_box_reg: 0.0032 (0.0039) time: 0.5414 data: 0.0111 max
mem: 3817
Epoch: [2] [50/60] eta: 0:00:05 lr: 0.005000 loss: 0.2140 (0.2251)
loss_classifier: 0.0276 (0.0304) loss_box_reg: 0.0599 (0.0613)
loss_mask: 0.1153 (0.1284) loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0032 (0.0039) time: 0.5528 data: 0.0107 max
mem: 3817
Epoch: [2] [59/60] eta: 0:00:00 lr: 0.005000 loss: 0.2146 (0.2259)
loss_classifier: 0.0276 (0.0306) loss_box_reg: 0.0576 (0.0625)
loss_mask: 0.1189 (0.1278) loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0032 (0.0038) time: 0.5763 data: 0.0090 max
mem: 3817
Epoch: [2] Total time: 0:00:33 (0.5627 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:16 model_time: 0.1396 (0.1396)
evaluator_time: 0.0032 (0.0032) time: 0.3367 data: 0.1929 max mem:
3817
Test: [49/50] eta: 0:00:00 model_time: 0.0948 (0.1043)
evaluator_time: 0.0032 (0.0053) time: 0.1115 data: 0.0049 max mem:
3817
Test: Total time: 0:00:06 (0.1208 s / it)
Averaged stats: model_time: 0.0948 (0.1043) evaluator_time: 0.0032
(0.0053)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.778
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.993
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.929
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 ] = 0.734
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.781
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.359
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
```

```

10 ] = 0.821
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.821
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 ] = 0.817
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.821
IoU metric: segm
Average Precision   (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.763
Average Precision   (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.993
Average Precision   (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.942
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 ] = 0.550
Average Precision   (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.771
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.353
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.795
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.796
Average Recall      (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
Average Recall      (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.717
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.800
Epoch: [3] [ 0/60] eta: 0:00:46 lr: 0.000500 loss: 0.1401 (0.1401)
loss_classifier: 0.0166 (0.0166) loss_box_reg: 0.0245 (0.0245)
loss_mask: 0.0985 (0.0985) loss_objectness: 0.0001 (0.0001)
loss_rpn_box_reg: 0.0004 (0.0004) time: 0.7822 data: 0.2851 max
mem: 3817
Epoch: [3] [10/60] eta: 0:00:29 lr: 0.000500 loss: 0.2171 (0.2141)
loss_classifier: 0.0321 (0.0295) loss_box_reg: 0.0667 (0.0560)
loss_mask: 0.1246 (0.1245) loss_objectness: 0.0005 (0.0008)
loss_rpn_box_reg: 0.0021 (0.0033) time: 0.5920 data: 0.0359 max
mem: 3817
Epoch: [3] [20/60] eta: 0:00:22 lr: 0.000500 loss: 0.1883 (0.2011)
loss_classifier: 0.0313 (0.0296) loss_box_reg: 0.0494 (0.0481)
loss_mask: 0.1072 (0.1195) loss_objectness: 0.0005 (0.0009)
loss_rpn_box_reg: 0.0021 (0.0030) time: 0.5553 data: 0.0101 max
mem: 3817
Epoch: [3] [30/60] eta: 0:00:17 lr: 0.000500 loss: 0.1728 (0.2027)

```

```

loss_classifier: 0.0272 (0.0278) loss_box_reg: 0.0421 (0.0489)
loss_mask: 0.1044 (0.1224) loss_objectness: 0.0004 (0.0007)
loss_rpn_box_reg: 0.0026 (0.0030) time: 0.5588 data: 0.0107 max
mem: 3817
Epoch: [3] [40/60] eta: 0:00:11 lr: 0.000500 loss: 0.1920 (0.2012)
loss_classifier: 0.0232 (0.0274) loss_box_reg: 0.0440 (0.0471)
loss_mask: 0.1139 (0.1230) loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0026 (0.0030) time: 0.5586 data: 0.0105 max
mem: 3817
Epoch: [3] [50/60] eta: 0:00:05 lr: 0.000500 loss: 0.1831 (0.1980)
loss_classifier: 0.0247 (0.0271) loss_box_reg: 0.0358 (0.0453)
loss_mask: 0.1137 (0.1219) loss_objectness: 0.0007 (0.0008)
loss_rpn_box_reg: 0.0027 (0.0029) time: 0.5378 data: 0.0087 max
mem: 3817
Epoch: [3] [59/60] eta: 0:00:00 lr: 0.000500 loss: 0.1916 (0.1987)
loss_classifier: 0.0280 (0.0280) loss_box_reg: 0.0424 (0.0458)
loss_mask: 0.1139 (0.1210) loss_objectness: 0.0005 (0.0008)
loss_rpn_box_reg: 0.0035 (0.0031) time: 0.5707 data: 0.0089 max
mem: 3817
Epoch: [3] Total time: 0:00:34 (0.5670 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:16 model_time: 0.1618 (0.1618)
evaluator_time: 0.0053 (0.0053) time: 0.3387 data: 0.1689 max mem:
3817
Test: [49/50] eta: 0:00:00 model_time: 0.0984 (0.1064)
evaluator_time: 0.0047 (0.0063) time: 0.1195 data: 0.0071 max mem:
3817
Test: Total time: 0:00:06 (0.1255 s / it)
Averaged stats: model_time: 0.0984 (0.1064) evaluator_time: 0.0047
(0.0063)
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 ] = 0.832
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.967
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 ] = 0.776
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.838
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.384

```

```

Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.872
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.872
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 ] = 0.833
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.874
IoU metric: segm
Average Precision   (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.764
Average Precision   (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.992
Average Precision   (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.933
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 ] = 0.613
Average Precision   (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.771
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.353
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.803
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.804
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 ] = 0.733
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.808
Epoch: [4] [ 0/60] eta: 0:00:49 lr: 0.000500 loss: 0.1643 (0.1643)
loss_classifier: 0.0290 (0.0290) loss_box_reg: 0.0375 (0.0375)
loss_mask: 0.0942 (0.0942) loss_objectness: 0.0004 (0.0004)
loss_rpn_box_reg: 0.0032 (0.0032) time: 0.8301 data: 0.3273 max
mem: 3817
Epoch: [4] [10/60] eta: 0:00:28 lr: 0.000500 loss: 0.1715 (0.1828)
loss_classifier: 0.0256 (0.0265) loss_box_reg: 0.0375 (0.0410)
loss_mask: 0.0979 (0.1114) loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0022 (0.0030) time: 0.5752 data: 0.0359 max
mem: 3817
Epoch: [4] [20/60] eta: 0:00:22 lr: 0.000500 loss: 0.1591 (0.1708)
loss_classifier: 0.0201 (0.0225) loss_box_reg: 0.0297 (0.0357)
loss_mask: 0.1036 (0.1092) loss_objectness: 0.0004 (0.0010)
loss_rpn_box_reg: 0.0018 (0.0024) time: 0.5513 data: 0.0083 max
mem: 3817

```

```
Epoch: [4] [30/60] eta: 0:00:17 lr: 0.000500 loss: 0.1653 (0.1727)
loss_classifier: 0.0198 (0.0222) loss_box_reg: 0.0335 (0.0362)
loss_mask: 0.1074 (0.1111) loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0020 (0.0025) time: 0.5700 data: 0.0097 max
mem: 3817
Epoch: [4] [40/60] eta: 0:00:11 lr: 0.000500 loss: 0.1790 (0.1773)
loss_classifier: 0.0232 (0.0235) loss_box_reg: 0.0369 (0.0371)
loss_mask: 0.1086 (0.1131) loss_objectness: 0.0004 (0.0009)
loss_rpn_box_reg: 0.0028 (0.0027) time: 0.5883 data: 0.0100 max
mem: 4189
Epoch: [4] [50/60] eta: 0:00:05 lr: 0.000500 loss: 0.2030 (0.1863)
loss_classifier: 0.0283 (0.0249) loss_box_reg: 0.0425 (0.0405)
loss_mask: 0.1282 (0.1173) loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0028 (0.0028) time: 0.6001 data: 0.0110 max
mem: 4189
Epoch: [4] [59/60] eta: 0:00:00 lr: 0.000500 loss: 0.2016 (0.1884)
loss_classifier: 0.0274 (0.0258) loss_box_reg: 0.0386 (0.0407)
loss_mask: 0.1180 (0.1183) loss_objectness: 0.0004 (0.0008)
loss_rpn_box_reg: 0.0027 (0.0028) time: 0.5735 data: 0.0099 max
mem: 4189
Epoch: [4] Total time: 0:00:34 (0.5781 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:18 model_time: 0.1463 (0.1463)
evaluator_time: 0.0025 (0.0025) time: 0.3641 data: 0.2140 max mem:
4189
Test: [49/50] eta: 0:00:00 model_time: 0.0993 (0.1053)
evaluator_time: 0.0032 (0.0050) time: 0.1118 data: 0.0043 max mem:
4189
Test: Total time: 0:00:06 (0.1215 s / it)
Averaged stats: model_time: 0.0993 (0.1053) evaluator_time: 0.0032
(0.0050)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.830
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.967
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 ] = 0.788
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.835
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
```



```

1 ] = 0.386
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.873
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.873
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 ] = 0.850
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.875
IoU metric: segm
Average Precision    (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.761
Average Precision    (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.992
Average Precision    (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.928
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 ] = 0.588
Average Precision    (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.768
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.356
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.799
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.799
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 ] = 0.717
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.804
Epoch: [5] [ 0/60] eta: 0:00:47 lr: 0.000500 loss: 0.1290 (0.1290)
loss_classifier: 0.0166 (0.0166) loss_box_reg: 0.0187 (0.0187)
loss_mask: 0.0929 (0.0929) loss_objectness: 0.0002 (0.0002)
loss_rpn_box_reg: 0.0006 (0.0006) time: 0.7918 data: 0.2888 max
mem: 4189
Epoch: [5] [10/60] eta: 0:00:30 lr: 0.000500 loss: 0.1743 (0.1676)
loss_classifier: 0.0216 (0.0216) loss_box_reg: 0.0403 (0.0359)
loss_mask: 0.0970 (0.1058) loss_objectness: 0.0004 (0.0017)
loss_rpn_box_reg: 0.0023 (0.0025) time: 0.6021 data: 0.0358 max
mem: 4189
Epoch: [5] [20/60] eta: 0:00:23 lr: 0.000500 loss: 0.1746 (0.1856)
loss_classifier: 0.0222 (0.0249) loss_box_reg: 0.0410 (0.0398)
loss_mask: 0.1129 (0.1167) loss_objectness: 0.0003 (0.0013)
loss_rpn_box_reg: 0.0024 (0.0030) time: 0.5865 data: 0.0103 max

```

```
mem: 4189
Epoch: [5] [30/60] eta: 0:00:17 lr: 0.000500 loss: 0.1674 (0.1775)
loss_classifier: 0.0214 (0.0229) loss_box_reg: 0.0311 (0.0357)
loss_mask: 0.1129 (0.1151) loss_objectness: 0.0003 (0.0011)
loss_rpn_box_reg: 0.0024 (0.0027) time: 0.5695 data: 0.0113 max
mem: 4189
Epoch: [5] [40/60] eta: 0:00:11 lr: 0.000500 loss: 0.1670 (0.1769)
loss_classifier: 0.0216 (0.0246) loss_box_reg: 0.0311 (0.0355)
loss_mask: 0.1078 (0.1130) loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0022 (0.0026) time: 0.5676 data: 0.0113 max
mem: 4189
Epoch: [5] [50/60] eta: 0:00:05 lr: 0.000500 loss: 0.1746 (0.1827)
loss_classifier: 0.0290 (0.0259) loss_box_reg: 0.0278 (0.0385)
loss_mask: 0.1116 (0.1146) loss_objectness: 0.0003 (0.0011)
loss_rpn_box_reg: 0.0020 (0.0027) time: 0.5851 data: 0.0103 max
mem: 4189
Epoch: [5] [59/60] eta: 0:00:00 lr: 0.000500 loss: 0.1716 (0.1807)
loss_classifier: 0.0229 (0.0254) loss_box_reg: 0.0259 (0.0377)
loss_mask: 0.1067 (0.1139) loss_objectness: 0.0002 (0.0010)
loss_rpn_box_reg: 0.0018 (0.0027) time: 0.5633 data: 0.0098 max
mem: 4189
Epoch: [5] Total time: 0:00:34 (0.5772 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:18 model_time: 0.1503 (0.1503)
evaluator_time: 0.0027 (0.0027) time: 0.3668 data: 0.2127 max mem:
4189
Test: [49/50] eta: 0:00:00 model_time: 0.1000 (0.1074)
evaluator_time: 0.0047 (0.0059) time: 0.1216 data: 0.0083 max mem:
4189
Test: Total time: 0:00:06 (0.1270 s / it)
Averaged stats: model_time: 0.1000 (0.1074) evaluator_time: 0.0047
(0.0059)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.839
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.968
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 ] = 0.751
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.844
```

```

Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.391
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.878
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.878
Average Recall      (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
Average Recall      (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.833
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.880
IoU metric: segm
Average Precision    (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.769
Average Precision    (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.992
Average Precision    (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.952
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 ] = 0.563
Average Precision    (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.776
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.355
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.804
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.804
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 ] = 0.733
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.808
Epoch: [6] [ 0/60] eta: 0:00:57 lr: 0.000050 loss: 0.1892 (0.1892)
loss_classifier: 0.0293 (0.0293) loss_box_reg: 0.0372 (0.0372)
loss_mask: 0.1200 (0.1200) loss_objectness: 0.0004 (0.0004)
loss_rpn_box_reg: 0.0023 (0.0023) time: 0.9578 data: 0.3570 max
mem: 4189
Epoch: [6] [10/60] eta: 0:00:29 lr: 0.000050 loss: 0.1533 (0.1572)
loss_classifier: 0.0159 (0.0223) loss_box_reg: 0.0235 (0.0263)
loss_mask: 0.1065 (0.1052) loss_objectness: 0.0003 (0.0012)
loss_rpn_box_reg: 0.0023 (0.0022) time: 0.5914 data: 0.0387 max
mem: 4189
Epoch: [6] [20/60] eta: 0:00:23 lr: 0.000050 loss: 0.1533 (0.1686)
loss_classifier: 0.0171 (0.0245) loss_box_reg: 0.0245 (0.0317)
loss_mask: 0.1038 (0.1088) loss_objectness: 0.0003 (0.0008)

```

```
loss_rpn_box_reg: 0.0025 (0.0027)  time: 0.5668  data: 0.0085  max
mem: 4189
Epoch: [6]  [30/60]  eta: 0:00:17  lr: 0.000050  loss: 0.1702 (0.1775)
loss_classifier: 0.0275 (0.0257)  loss_box_reg: 0.0361 (0.0358)
loss_mask: 0.1114 (0.1123)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0027 (0.0027)  time: 0.5817  data: 0.0098  max
mem: 4189
Epoch: [6]  [40/60]  eta: 0:00:11  lr: 0.000050  loss: 0.1846 (0.1753)
loss_classifier: 0.0254 (0.0252)  loss_box_reg: 0.0329 (0.0343)
loss_mask: 0.1166 (0.1123)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0022 (0.0026)  time: 0.5737  data: 0.0090  max
mem: 4189
Epoch: [6]  [50/60]  eta: 0:00:05  lr: 0.000050  loss: 0.1803 (0.1812)
loss_classifier: 0.0253 (0.0257)  loss_box_reg: 0.0329 (0.0372)
loss_mask: 0.1128 (0.1146)  loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0026 (0.0028)  time: 0.5878  data: 0.0112  max
mem: 4189
Epoch: [6]  [59/60]  eta: 0:00:00  lr: 0.000050  loss: 0.1769 (0.1809)
loss_classifier: 0.0249 (0.0253)  loss_box_reg: 0.0423 (0.0378)
loss_mask: 0.1037 (0.1142)  loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0025 (0.0028)  time: 0.5846  data: 0.0109  max
mem: 4189
Epoch: [6] Total time: 0:00:35 (0.5847 s / it)
creating index...
index created!
Test:  [ 0/50]  eta: 0:00:17  model_time: 0.1402 (0.1402)
evaluator_time: 0.0026 (0.0026)  time: 0.3514  data: 0.2074  max mem:
4189
Test:  [49/50]  eta: 0:00:00  model_time: 0.0970 (0.1061)
evaluator_time: 0.0029 (0.0049)  time: 0.1122  data: 0.0042  max mem:
4189
Test: Total time: 0:00:06 (0.1223 s / it)
Averaged stats: model_time: 0.0970 (0.1061)  evaluator_time: 0.0029
(0.0049)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision  (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.843
Average Precision  (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.992
Average Precision  (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.967
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 ] = 0.768
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
```

```

maxDets=100 ] = 0.848
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.391
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.881
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.881
Average Recall      (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
Average Recall      (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.850
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.883
IoU metric: segm
Average Precision    (AP) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.765
Average Precision    (AP) @[ IoU=0.50      | area=  all |
maxDets=100 ] = 0.992
Average Precision    (AP) @[ IoU=0.75      | area=  all |
maxDets=100 ] = 0.952
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 ] = 0.555
Average Precision    (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.772
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
1 ] = 0.355
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.802
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.802
Average Recall      (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
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: [7] [ 0/60] eta: 0:00:53  lr: 0.000050  loss: 0.2675 (0.2675)
loss_classifier: 0.0526 (0.0526)  loss_box_reg: 0.0755 (0.0755)
loss_mask: 0.1337 (0.1337)  loss_objectness: 0.0004 (0.0004)
loss_rpn_box_reg: 0.0052 (0.0052)  time: 0.8951  data: 0.2547  max
mem: 4189
Epoch: [7] [10/60] eta: 0:00:30  lr: 0.000050  loss: 0.1628 (0.1760)
loss_classifier: 0.0234 (0.0264)  loss_box_reg: 0.0311 (0.0360)
loss_mask: 0.1013 (0.1104)  loss_objectness: 0.0003 (0.0004)
loss_rpn_box_reg: 0.0024 (0.0029)  time: 0.6089  data: 0.0323  max
mem: 4189
Epoch: [7] [20/60] eta: 0:00:23  lr: 0.000050  loss: 0.1605 (0.1701)
loss_classifier: 0.0234 (0.0255)  loss_box_reg: 0.0298 (0.0341)

```

```
loss_mask: 0.0982 (0.1075) loss_objectness: 0.0003 (0.0004)
loss_rpn_box_reg: 0.0022 (0.0025) time: 0.5688 data: 0.0097 max
mem: 4189
Epoch: [7] [30/60] eta: 0:00:17 lr: 0.000050 loss: 0.1693 (0.1828)
loss_classifier: 0.0242 (0.0268) loss_box_reg: 0.0372 (0.0380)
loss_mask: 0.1032 (0.1146) loss_objectness: 0.0006 (0.0006)
loss_rpn_box_reg: 0.0025 (0.0028) time: 0.5594 data: 0.0105 max
mem: 4189
Epoch: [7] [40/60] eta: 0:00:11 lr: 0.000050 loss: 0.1806 (0.1787)
loss_classifier: 0.0218 (0.0250) loss_box_reg: 0.0342 (0.0361)
loss_mask: 0.1155 (0.1143) loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0027 (0.0027) time: 0.5828 data: 0.0112 max
mem: 4189
Epoch: [7] [50/60] eta: 0:00:05 lr: 0.000050 loss: 0.1614 (0.1762)
loss_classifier: 0.0199 (0.0242) loss_box_reg: 0.0274 (0.0350)
loss_mask: 0.1135 (0.1137) loss_objectness: 0.0003 (0.0006)
loss_rpn_box_reg: 0.0023 (0.0027) time: 0.5975 data: 0.0105 max
mem: 4189
Epoch: [7] [59/60] eta: 0:00:00 lr: 0.000050 loss: 0.1641 (0.1756)
loss_classifier: 0.0204 (0.0244) loss_box_reg: 0.0272 (0.0345)
loss_mask: 0.1135 (0.1134) loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0023 (0.0026) time: 0.5710 data: 0.0100 max
mem: 4189
Epoch: [7] Total time: 0:00:34 (0.5796 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:16 model_time: 0.1579 (0.1579)
evaluator_time: 0.0038 (0.0038) time: 0.3338 data: 0.1710 max mem:
4189
Test: [49/50] eta: 0:00:00 model_time: 0.1000 (0.1078)
evaluator_time: 0.0047 (0.0061) time: 0.1218 data: 0.0083 max mem:
4189
Test: Total time: 0:00:06 (0.1272 s / it)
Averaged stats: model_time: 0.1000 (0.1078) evaluator_time: 0.0047
(0.0061)
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 ] = 0.845
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.967
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 ] = 0.751
```

```

Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.850
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.391
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
10 ] = 0.881
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.881
Average Recall (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium |
maxDets=100 ] = 0.833
Average Recall (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.884
IoU metric: segm
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.768
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.952
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 ] = 0.563
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.775
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.356
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
10 ] = 0.804
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.804
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 ] = 0.733
Average Recall (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.808
Epoch: [8] [ 0/60] eta: 0:00:50 lr: 0.000050 loss: 0.1265 (0.1265)
loss_classifier: 0.0134 (0.0134) loss_box_reg: 0.0223 (0.0223)
loss_mask: 0.0887 (0.0887) loss_objectness: 0.0003 (0.0003)
loss_rpn_box_reg: 0.0016 (0.0016) time: 0.8394 data: 0.3291 max
mem: 4189
Epoch: [8] [10/60] eta: 0:00:30 lr: 0.000050 loss: 0.1558 (0.1746)
loss_classifier: 0.0214 (0.0237) loss_box_reg: 0.0244 (0.0349)
loss_mask: 0.1126 (0.1126) loss_objectness: 0.0004 (0.0007)
loss_rpn_box_reg: 0.0025 (0.0027) time: 0.6015 data: 0.0379 max
mem: 4189
Epoch: [8] [20/60] eta: 0:00:23 lr: 0.000050 loss: 0.1644 (0.1798)

```

```

loss_classifier: 0.0225 (0.0264) loss_box_reg: 0.0295 (0.0361)
loss_mask: 0.1095 (0.1135) loss_objectness: 0.0004 (0.0010)
loss_rpn_box_reg: 0.0023 (0.0028) time: 0.5781 data: 0.0097 max
mem: 4189
Epoch: [8] [30/60] eta: 0:00:17 lr: 0.000050 loss: 0.1708 (0.1772)
loss_classifier: 0.0225 (0.0249) loss_box_reg: 0.0388 (0.0360)
loss_mask: 0.1085 (0.1128) loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0024 (0.0027) time: 0.5861 data: 0.0105 max
mem: 4189
Epoch: [8] [40/60] eta: 0:00:11 lr: 0.000050 loss: 0.1688 (0.1797)
loss_classifier: 0.0211 (0.0249) loss_box_reg: 0.0368 (0.0374)
loss_mask: 0.1038 (0.1138) loss_objectness: 0.0002 (0.0009)
loss_rpn_box_reg: 0.0025 (0.0027) time: 0.5833 data: 0.0106 max
mem: 4189
Epoch: [8] [50/60] eta: 0:00:05 lr: 0.000050 loss: 0.1709 (0.1783)
loss_classifier: 0.0211 (0.0242) loss_box_reg: 0.0367 (0.0368)
loss_mask: 0.1107 (0.1138) loss_objectness: 0.0003 (0.0009)
loss_rpn_box_reg: 0.0021 (0.0026) time: 0.5706 data: 0.0110 max
mem: 4189
Epoch: [8] [59/60] eta: 0:00:00 lr: 0.000050 loss: 0.1709 (0.1784)
loss_classifier: 0.0182 (0.0238) loss_box_reg: 0.0332 (0.0364)
loss_mask: 0.1107 (0.1146) loss_objectness: 0.0003 (0.0008)
loss_rpn_box_reg: 0.0025 (0.0027) time: 0.5714 data: 0.0105 max
mem: 4189
Epoch: [8] Total time: 0:00:35 (0.5835 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:17 model_time: 0.1565 (0.1565)
evaluator_time: 0.0027 (0.0027) time: 0.3557 data: 0.1953 max mem:
4189
Test: [49/50] eta: 0:00:00 model_time: 0.0996 (0.1065)
evaluator_time: 0.0029 (0.0050) time: 0.1126 data: 0.0041 max mem:
4189
Test: Total time: 0:00:06 (0.1228 s / it)
Averaged stats: model_time: 0.0996 (0.1065) evaluator_time: 0.0029
(0.0050)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.848
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.967
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 ] = 0.751
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.854
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.392
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
10 ] = 0.884
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.884
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 ] = 0.833
Average Recall (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.887
IoU metric: segm
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.767
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.952
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 ] = 0.555
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.774
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.355
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
10 ] = 0.803
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.803
Average Recall (AR) @[ IoU=0.50:0.95 | area= small |
maxDets=100 ] = -1.000
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 ] = 0.806
Epoch: [9] [ 0/60] eta: 0:01:03 lr: 0.000005 loss: 0.3226 (0.3226)
loss_classifier: 0.0578 (0.0578) loss_box_reg: 0.0890 (0.0890)
loss_mask: 0.1635 (0.1635) loss_objectness: 0.0050 (0.0050)
loss_rpn_box_reg: 0.0073 (0.0073) time: 1.0575 data: 0.3808 max
mem: 4189
Epoch: [9] [10/60] eta: 0:00:30 lr: 0.000005 loss: 0.1775 (0.1962)
loss_classifier: 0.0238 (0.0284) loss_box_reg: 0.0393 (0.0460)
loss_mask: 0.1140 (0.1177) loss_objectness: 0.0004 (0.0012)
loss_rpn_box_reg: 0.0023 (0.0029) time: 0.6033 data: 0.0421 max
mem: 4189

```

```

Epoch: [9] [20/60] eta: 0:00:23 lr: 0.000005 loss: 0.1691 (0.1889)
loss_classifier: 0.0232 (0.0269) loss_box_reg: 0.0254 (0.0418)
loss_mask: 0.1051 (0.1168) loss_objectness: 0.0003 (0.0007)
loss_rpn_box_reg: 0.0017 (0.0026) time: 0.5669 data: 0.0086 max
mem: 4189
Epoch: [9] [30/60] eta: 0:00:17 lr: 0.000005 loss: 0.1673 (0.1878)
loss_classifier: 0.0249 (0.0264) loss_box_reg: 0.0216 (0.0405)
loss_mask: 0.1116 (0.1176) loss_objectness: 0.0002 (0.0007)
loss_rpn_box_reg: 0.0013 (0.0026) time: 0.5772 data: 0.0096 max
mem: 4189
Epoch: [9] [40/60] eta: 0:00:11 lr: 0.000005 loss: 0.1625 (0.1812)
loss_classifier: 0.0203 (0.0253) loss_box_reg: 0.0323 (0.0383)
loss_mask: 0.1036 (0.1141) loss_objectness: 0.0002 (0.0008)
loss_rpn_box_reg: 0.0021 (0.0026) time: 0.5697 data: 0.0100 max
mem: 4189
Epoch: [9] [50/60] eta: 0:00:05 lr: 0.000005 loss: 0.1530 (0.1775)
loss_classifier: 0.0199 (0.0248) loss_box_reg: 0.0257 (0.0364)
loss_mask: 0.1093 (0.1130) loss_objectness: 0.0002 (0.0007)
loss_rpn_box_reg: 0.0020 (0.0026) time: 0.5707 data: 0.0106 max
mem: 4189
Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.1584 (0.1779)
loss_classifier: 0.0247 (0.0254) loss_box_reg: 0.0257 (0.0363)
loss_mask: 0.1093 (0.1129) loss_objectness: 0.0005 (0.0008)
loss_rpn_box_reg: 0.0018 (0.0026) time: 0.5787 data: 0.0110 max
mem: 4189
Epoch: [9] Total time: 0:00:34 (0.5807 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:18 model_time: 0.1379 (0.1379)
evaluator_time: 0.0026 (0.0026) time: 0.3625 data: 0.2208 max mem:
4189
Test: [49/50] eta: 0:00:00 model_time: 0.0974 (0.1073)
evaluator_time: 0.0041 (0.0060) time: 0.1210 data: 0.0088 max mem:
4189
Test: Total time: 0:00:06 (0.1268 s / it)
Averaged stats: model_time: 0.0974 (0.1073) evaluator_time: 0.0041
(0.0060)
Accumulating evaluation results...
DONE (t=0.01s).
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.847
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.992
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.967
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] = 0.751
 Average Precision (AP) @[IoU=0.50:0.95 | area= large |
 maxDets=100] = 0.853
 Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
 1] = 0.392
 Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
 10] = 0.884
 Average Recall (AR) @[IoU=0.50:0.95 | area= all |
 maxDets=100] = 0.884
 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] = 0.833
 Average Recall (AR) @[IoU=0.50:0.95 | area= large |
 maxDets=100] = 0.887
 IoU metric: segm
 Average Precision (AP) @[IoU=0.50:0.95 | area= all |
 maxDets=100] = 0.767
 Average Precision (AP) @[IoU=0.50 | area= all |
 maxDets=100] = 0.992
 Average Precision (AP) @[IoU=0.75 | area= all |
 maxDets=100] = 0.952
 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] = 0.555
 Average Precision (AP) @[IoU=0.50:0.95 | area= large |
 maxDets=100] = 0.774
 Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
 1] = 0.356
 Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
 10] = 0.803
 Average Recall (AR) @[IoU=0.50:0.95 | area= all |
 maxDets=100] = 0.803
 Average Recall (AR) @[IoU=0.50:0.95 | area= small |
 maxDets=100] = -1.000
 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] = 0.807
 Model with backbone

Epoch: [0] [0/60] eta: 0:01:09 lr: 0.000090 loss: 1.4289 (1.4289)
 loss_classifier: 0.6940 (0.6940) loss_box_reg: 0.0158 (0.0158)
 loss_objectness: 0.6910 (0.6910) loss_rpn_box_reg: 0.0280 (0.0280)
 time: 1.1523 data: 0.3935 max mem: 4811
 Epoch: [0] [10/60] eta: 0:00:21 lr: 0.000936 loss: 1.4270 (1.3862)
 loss_classifier: 0.6473 (0.6116) loss_box_reg: 0.0341 (0.0431)
 loss_objectness: 0.6940 (0.6897) loss_rpn_box_reg: 0.0292 (0.0418)

```

time: 0.4274 data: 0.0423 max mem: 5695
Epoch: [0] [20/60] eta: 0:00:15 lr: 0.001783 loss: 1.2220 (1.2383)
loss_classifier: 0.4263 (0.4617) loss_box_reg: 0.0647 (0.0852)
loss_objectness: 0.6603 (0.6483) loss_rpn_box_reg: 0.0331 (0.0431)
time: 0.3510 data: 0.0082 max mem: 5695
Epoch: [0] [30/60] eta: 0:00:11 lr: 0.002629 loss: 1.0461 (1.1518)
loss_classifier: 0.2871 (0.4089) loss_box_reg: 0.1518 (0.1183)
loss_objectness: 0.5158 (0.5826) loss_rpn_box_reg: 0.0397 (0.0420)
time: 0.3499 data: 0.0104 max mem: 5718
Epoch: [0] [40/60] eta: 0:00:07 lr: 0.003476 loss: 0.8218 (1.0517)
loss_classifier: 0.2428 (0.3694) loss_box_reg: 0.1593 (0.1281)
loss_objectness: 0.3672 (0.5160) loss_rpn_box_reg: 0.0333 (0.0382)
time: 0.3490 data: 0.0125 max mem: 5718
Epoch: [0] [50/60] eta: 0:00:03 lr: 0.004323 loss: 0.7528 (1.0023)
loss_classifier: 0.2424 (0.3505) loss_box_reg: 0.1843 (0.1526)
loss_objectness: 0.2750 (0.4616) loss_rpn_box_reg: 0.0305 (0.0376)
time: 0.3474 data: 0.0133 max mem: 5718
Epoch: [0] [59/60] eta: 0:00:00 lr: 0.005000 loss: 0.6749 (0.9474)
loss_classifier: 0.2388 (0.3325) loss_box_reg: 0.1914 (0.1580)
loss_objectness: 0.2059 (0.4205) loss_rpn_box_reg: 0.0305 (0.0364)
time: 0.3463 data: 0.0105 max mem: 5718
Epoch: [0] Total time: 0:00:21 (0.3643 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:14 model_time: 0.0996 (0.0996)
evaluator_time: 0.0034 (0.0034) time: 0.2805 data: 0.1762 max mem:
5718
Test: [49/50] eta: 0:00:00 model_time: 0.0368 (0.0404)
evaluator_time: 0.0024 (0.0037) time: 0.0453 data: 0.0041 max mem:
5718
Test: Total time: 0:00:02 (0.0553 s / it)
Averaged stats: model_time: 0.0368 (0.0404) evaluator_time: 0.0024
(0.0037)
Accumulating evaluation results...
DONE (t=0.02s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.065
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.223
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.010
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 ] = 0.117
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.086

```

```

Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.294
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.356
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 ] = 0.375
Epoch: [1] [ 0/60] eta: 0:00:42 lr: 0.005000 loss: 0.3238 (0.3238)
loss_classifier: 0.1252 (0.1252) loss_box_reg: 0.0666 (0.0666)
loss_objectness: 0.1272 (0.1272) loss_rpn_box_reg: 0.0048 (0.0048)
time: 0.7152 data: 0.3508 max mem: 5718
Epoch: [1] [10/60] eta: 0:00:18 lr: 0.005000 loss: 0.5266 (0.5354)
loss_classifier: 0.1748 (0.1870) loss_box_reg: 0.1566 (0.1705)
loss_objectness: 0.1567 (0.1516) loss_rpn_box_reg: 0.0274 (0.0263)
time: 0.3774 data: 0.0385 max mem: 5718
Epoch: [1] [20/60] eta: 0:00:14 lr: 0.005000 loss: 0.5266 (0.5540)
loss_classifier: 0.1748 (0.1885) loss_box_reg: 0.1713 (0.1875)
loss_objectness: 0.1440 (0.1496) loss_rpn_box_reg: 0.0274 (0.0284)
time: 0.3507 data: 0.0089 max mem: 5725
Epoch: [1] [30/60] eta: 0:00:10 lr: 0.005000 loss: 0.5318 (0.5528)
loss_classifier: 0.1865 (0.1873) loss_box_reg: 0.1779 (0.1939)
loss_objectness: 0.1283 (0.1432) loss_rpn_box_reg: 0.0270 (0.0284)
time: 0.3530 data: 0.0116 max mem: 5725
Epoch: [1] [40/60] eta: 0:00:07 lr: 0.005000 loss: 0.4959 (0.5579)
loss_classifier: 0.1555 (0.1854) loss_box_reg: 0.1926 (0.2029)
loss_objectness: 0.1194 (0.1396) loss_rpn_box_reg: 0.0299 (0.0300)
time: 0.3496 data: 0.0109 max mem: 5725
Epoch: [1] [50/60] eta: 0:00:03 lr: 0.005000 loss: 0.4852 (0.5412)
loss_classifier: 0.1475 (0.1762) loss_box_reg: 0.1807 (0.1994)
loss_objectness: 0.1170 (0.1348) loss_rpn_box_reg: 0.0314 (0.0308)
time: 0.3470 data: 0.0093 max mem: 5725
Epoch: [1] [59/60] eta: 0:00:00 lr: 0.005000 loss: 0.4379 (0.5186)
loss_classifier: 0.1227 (0.1674) loss_box_reg: 0.1722 (0.1922)
loss_objectness: 0.1021 (0.1297) loss_rpn_box_reg: 0.0273 (0.0293)
time: 0.3484 data: 0.0091 max mem: 5945
Epoch: [1] Total time: 0:00:21 (0.3579 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:14 model_time: 0.0865 (0.0865)
evaluator_time: 0.0025 (0.0025) time: 0.2801 data: 0.1899 max mem:
5945
Test: [49/50] eta: 0:00:00 model_time: 0.0367 (0.0391)
evaluator_time: 0.0017 (0.0023) time: 0.0447 data: 0.0041 max mem:
5945
Test: Total time: 0:00:02 (0.0530 s / it)
Averaged stats: model_time: 0.0367 (0.0391) evaluator_time: 0.0017
(0.0023)

```

Accumulating evaluation results...

DONE (t=0.01s).

IoU metric: bbox

Average Precision (AP) @[IoU=0.50:0.95 | area= all |
maxDets=100] = 0.173

Average Precision (AP) @[IoU=0.50 | area= all |
maxDets=100] = 0.523

Average Precision (AP) @[IoU=0.75 | area= all |
maxDets=100] = 0.032

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] = 0.185

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
1] = 0.102

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
10] = 0.365

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] = 0.000

Average Recall (AR) @[IoU=0.50:0.95 | area= large |
maxDets=100] = 0.417

Epoch: [2] [0/60] eta: 0:00:42 lr: 0.005000 loss: 0.3912 (0.3912)

loss_classifier: 0.1268 (0.1268) loss_box_reg: 0.1767 (0.1767)

loss_objectness: 0.0756 (0.0756) loss_rpn_box_reg: 0.0121 (0.0121)

time: 0.7099 data: 0.3776 max mem: 5945

Epoch: [2] [10/60] eta: 0:00:18 lr: 0.005000 loss: 0.4669 (0.4401)

loss_classifier: 0.1360 (0.1348) loss_box_reg: 0.1767 (0.1833)

loss_objectness: 0.0851 (0.0992) loss_rpn_box_reg: 0.0227 (0.0228)

time: 0.3689 data: 0.0385 max mem: 5945

Epoch: [2] [20/60] eta: 0:00:14 lr: 0.005000 loss: 0.4049 (0.4241)

loss_classifier: 0.1241 (0.1292) loss_box_reg: 0.1659 (0.1797)

loss_objectness: 0.0848 (0.0925) loss_rpn_box_reg: 0.0227 (0.0227)

time: 0.3457 data: 0.0089 max mem: 5945

Epoch: [2] [30/60] eta: 0:00:10 lr: 0.005000 loss: 0.4001 (0.4267)

loss_classifier: 0.1184 (0.1287) loss_box_reg: 0.1719 (0.1864)

loss_objectness: 0.0809 (0.0879) loss_rpn_box_reg: 0.0215 (0.0238)

time: 0.3505 data: 0.0112 max mem: 5945

Epoch: [2] [40/60] eta: 0:00:07 lr: 0.005000 loss: 0.3807 (0.4310)

loss_classifier: 0.1181 (0.1311) loss_box_reg: 0.1800 (0.1890)

loss_objectness: 0.0727 (0.0858) loss_rpn_box_reg: 0.0255 (0.0252)

time: 0.3493 data: 0.0092 max mem: 5945

Epoch: [2] [50/60] eta: 0:00:03 lr: 0.005000 loss: 0.3357 (0.4102)

loss_classifier: 0.1033 (0.1250) loss_box_reg: 0.1450 (0.1789)

loss_objectness: 0.0653 (0.0814) loss_rpn_box_reg: 0.0257 (0.0249)

```

time: 0.3553 data: 0.0093 max mem: 5945
Epoch: [2] [59/60] eta: 0:00:00 lr: 0.005000 loss: 0.3343 (0.4031)
loss_classifier: 0.1029 (0.1220) loss_box_reg: 0.1190 (0.1766)
loss_objectness: 0.0638 (0.0795) loss_rpn_box_reg: 0.0269 (0.0250)
time: 0.3541 data: 0.0096 max mem: 5945
Epoch: [2] Total time: 0:00:21 (0.3575 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:15 model_time: 0.0758 (0.0758)
evaluator_time: 0.0019 (0.0019) time: 0.3080 data: 0.2292 max mem:
5945
Test: [49/50] eta: 0:00:00 model_time: 0.0370 (0.0392)
evaluator_time: 0.0015 (0.0021) time: 0.0444 data: 0.0041 max mem:
5945
Test: Total time: 0:00:02 (0.0534 s / it)
Averaged stats: model_time: 0.0370 (0.0392) evaluator_time: 0.0015
(0.0021)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.211
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.595
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.073
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 ] = 0.025
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.225
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 ] = 0.427
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.435
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 ] = 0.050
Average Recall (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.456
Epoch: [3] [ 0/60] eta: 0:00:44 lr: 0.000500 loss: 0.4632 (0.4632)
loss_classifier: 0.1222 (0.1222) loss_box_reg: 0.2606 (0.2606)
loss_objectness: 0.0556 (0.0556) loss_rpn_box_reg: 0.0249 (0.0249)
time: 0.7361 data: 0.3671 max mem: 5945
Epoch: [3] [10/60] eta: 0:00:19 lr: 0.000500 loss: 0.3073 (0.3001)
loss_classifier: 0.0895 (0.0928) loss_box_reg: 0.1380 (0.1346)

```

```
loss_objectness: 0.0513 (0.0547) loss_rpn_box_reg: 0.0166 (0.0180)
time: 0.3842 data: 0.0382 max mem: 5945
Epoch: [3] [20/60] eta: 0:00:14 lr: 0.000500 loss: 0.3073 (0.3390)
loss_classifier: 0.0895 (0.1043) loss_box_reg: 0.1430 (0.1558)
loss_objectness: 0.0523 (0.0580) loss_rpn_box_reg: 0.0177 (0.0209)
time: 0.3537 data: 0.0093 max mem: 5945
Epoch: [3] [30/60] eta: 0:00:10 lr: 0.000500 loss: 0.3575 (0.3433)
loss_classifier: 0.1099 (0.1025) loss_box_reg: 0.1586 (0.1594)
loss_objectness: 0.0540 (0.0612) loss_rpn_box_reg: 0.0195 (0.0203)
time: 0.3509 data: 0.0110 max mem: 5945
Epoch: [3] [40/60] eta: 0:00:07 lr: 0.000500 loss: 0.3572 (0.3569)
loss_classifier: 0.1099 (0.1067) loss_box_reg: 0.1750 (0.1673)
loss_objectness: 0.0531 (0.0622) loss_rpn_box_reg: 0.0176 (0.0207)
time: 0.3475 data: 0.0096 max mem: 5945
Epoch: [3] [50/60] eta: 0:00:03 lr: 0.000500 loss: 0.3006 (0.3386)
loss_classifier: 0.0950 (0.1020) loss_box_reg: 0.1581 (0.1583)
loss_objectness: 0.0506 (0.0586) loss_rpn_box_reg: 0.0175 (0.0197)
time: 0.3485 data: 0.0101 max mem: 5945
Epoch: [3] [59/60] eta: 0:00:00 lr: 0.000500 loss: 0.2849 (0.3417)
loss_classifier: 0.0916 (0.1025) loss_box_reg: 0.1227 (0.1617)
loss_objectness: 0.0486 (0.0569) loss_rpn_box_reg: 0.0173 (0.0206)
time: 0.3456 data: 0.0090 max mem: 5945
Epoch: [3] Total time: 0:00:21 (0.3577 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:14 model_time: 0.0718 (0.0718)
evaluator_time: 0.0024 (0.0024) time: 0.2911 data: 0.2156 max mem:
5945
Test: [49/50] eta: 0:00:00 model_time: 0.0371 (0.0394)
evaluator_time: 0.0016 (0.0022) time: 0.0443 data: 0.0041 max mem:
5945
Test: Total time: 0:00:02 (0.0531 s / it)
Averaged stats: model_time: 0.0371 (0.0394) evaluator_time: 0.0016
(0.0022)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.261
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.672
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.091
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 ] = 0.288
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
```



```

1 ] = 0.185
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all | maxDets=
10 ] = 0.434
Average Recall      (AR) @[ IoU=0.50:0.95 | area=  all |
maxDets=100 ] = 0.444
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 ] = 0.017
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.467
Epoch: [4] [ 0/60] eta: 0:00:41 lr: 0.000500 loss: 0.2540 (0.2540)
loss_classifier: 0.0863 (0.0863) loss_box_reg: 0.1155 (0.1155)
loss_objectness: 0.0412 (0.0412) loss_rpn_box_reg: 0.0110 (0.0110)
time: 0.6994 data: 0.2781 max mem: 5945
Epoch: [4] [10/60] eta: 0:00:19 lr: 0.000500 loss: 0.3179 (0.3418)
loss_classifier: 0.0953 (0.1045) loss_box_reg: 0.1583 (0.1664)
loss_objectness: 0.0492 (0.0509) loss_rpn_box_reg: 0.0191 (0.0201)
time: 0.3931 data: 0.0356 max mem: 6278
Epoch: [4] [20/60] eta: 0:00:14 lr: 0.000500 loss: 0.3044 (0.3306)
loss_classifier: 0.0953 (0.1007) loss_box_reg: 0.1365 (0.1557)
loss_objectness: 0.0492 (0.0528) loss_rpn_box_reg: 0.0199 (0.0214)
time: 0.3553 data: 0.0111 max mem: 6278
Epoch: [4] [30/60] eta: 0:00:10 lr: 0.000500 loss: 0.3066 (0.3306)
loss_classifier: 0.0959 (0.0999) loss_box_reg: 0.1460 (0.1546)
loss_objectness: 0.0479 (0.0541) loss_rpn_box_reg: 0.0219 (0.0220)
time: 0.3479 data: 0.0098 max mem: 6278
Epoch: [4] [40/60] eta: 0:00:07 lr: 0.000500 loss: 0.3054 (0.3292)
loss_classifier: 0.0852 (0.0987) loss_box_reg: 0.1302 (0.1540)
loss_objectness: 0.0461 (0.0552) loss_rpn_box_reg: 0.0181 (0.0214)
time: 0.3466 data: 0.0089 max mem: 6278
Epoch: [4] [50/60] eta: 0:00:03 lr: 0.000500 loss: 0.2624 (0.3248)
loss_classifier: 0.0751 (0.0968) loss_box_reg: 0.1288 (0.1531)
loss_objectness: 0.0447 (0.0544) loss_rpn_box_reg: 0.0150 (0.0205)
time: 0.3500 data: 0.0117 max mem: 6278
Epoch: [4] [59/60] eta: 0:00:00 lr: 0.000500 loss: 0.3168 (0.3274)
loss_classifier: 0.0871 (0.0976) loss_box_reg: 0.1426 (0.1539)
loss_objectness: 0.0582 (0.0554) loss_rpn_box_reg: 0.0189 (0.0205)
time: 0.3502 data: 0.0111 max mem: 6278
Epoch: [4] Total time: 0:00:21 (0.3585 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:15 model_time: 0.0908 (0.0908)
evaluator_time: 0.0020 (0.0020) time: 0.3031 data: 0.2091 max mem:
6278
Test: [49/50] eta: 0:00:00 model_time: 0.0370 (0.0440)
evaluator_time: 0.0016 (0.0041) time: 0.0443 data: 0.0040 max mem:
6278
Test: Total time: 0:00:03 (0.0626 s / it)
Averaged stats: model_time: 0.0370 (0.0440) evaluator_time: 0.0016

```

(0.0041)

Accumulating evaluation results...

DONE (t=0.01s).

IoU metric: bbox

Average Precision (AP) @[IoU=0.50:0.95 | area= all |
maxDets=100] = 0.230

Average Precision (AP) @[IoU=0.50 | area= all |
maxDets=100] = 0.659

Average Precision (AP) @[IoU=0.75 | area= all |
maxDets=100] = 0.106

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] = 0.252

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
1] = 0.155

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=
10] = 0.410

Average Recall (AR) @[IoU=0.50:0.95 | area= all |
maxDets=100] = 0.419

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] = 0.442

Epoch: [5] [0/60] eta: 0:00:43 lr: 0.000500 loss: 0.2222 (0.2222)

loss_classifier: 0.0820 (0.0820) loss_box_reg: 0.0825 (0.0825)

loss_objectness: 0.0495 (0.0495) loss_rpn_box_reg: 0.0081 (0.0081)

time: 0.7297 data: 0.3409 max mem: 6278

Epoch: [5] [10/60] eta: 0:00:19 lr: 0.000500 loss: 0.2525 (0.3356)

loss_classifier: 0.0820 (0.1006) loss_box_reg: 0.1314 (0.1646)

loss_objectness: 0.0451 (0.0515) loss_rpn_box_reg: 0.0177 (0.0189)

time: 0.3998 data: 0.0418 max mem: 6278

Epoch: [5] [20/60] eta: 0:00:15 lr: 0.000500 loss: 0.3672 (0.3521)

loss_classifier: 0.1023 (0.1063) loss_box_reg: 0.1852 (0.1737)

loss_objectness: 0.0457 (0.0522) loss_rpn_box_reg: 0.0177 (0.0199)

time: 0.3612 data: 0.0103 max mem: 6278

Epoch: [5] [30/60] eta: 0:00:11 lr: 0.000500 loss: 0.3183 (0.3448)

loss_classifier: 0.0958 (0.1033) loss_box_reg: 0.1510 (0.1649)

loss_objectness: 0.0511 (0.0561) loss_rpn_box_reg: 0.0176 (0.0206)

time: 0.3538 data: 0.0090 max mem: 6278

Epoch: [5] [40/60] eta: 0:00:07 lr: 0.000500 loss: 0.2791 (0.3321)

loss_classifier: 0.0781 (0.0996) loss_box_reg: 0.1396 (0.1598)

loss_objectness: 0.0394 (0.0532) loss_rpn_box_reg: 0.0163 (0.0196)

time: 0.3539 data: 0.0110 max mem: 6278

Epoch: [5] [50/60] eta: 0:00:03 lr: 0.000500 loss: 0.2801 (0.3304)

loss_classifier: 0.0806 (0.0997) loss_box_reg: 0.1362 (0.1591)

```
loss_objectness: 0.0373 (0.0525)  loss_rpn_box_reg: 0.0166 (0.0191)
time: 0.3523  data: 0.0120  max mem: 6278
Epoch: [5] [59/60]  eta: 0:00:00  lr: 0.000500  loss: 0.3023 (0.3290)
loss_classifier: 0.0895 (0.0994)  loss_box_reg: 0.1600 (0.1585)
loss_objectness: 0.0455 (0.0518)  loss_rpn_box_reg: 0.0182 (0.0193)
time: 0.3467  data: 0.0100  max mem: 6278
Epoch: [5] Total time: 0:00:21 (0.3614 s / it)
creating index...
index created!
Test: [ 0/50]  eta: 0:00:12  model_time: 0.0845 (0.0845)
evaluator_time: 0.0043 (0.0043)  time: 0.2558  data: 0.1659  max mem:
6278
Test: [49/50]  eta: 0:00:00  model_time: 0.0369 (0.0403)
evaluator_time: 0.0028 (0.0028)  time: 0.0491  data: 0.0067  max mem:
6278
Test: Total time: 0:00:02 (0.0560 s / it)
Averaged stats: model_time: 0.0369 (0.0403)  evaluator_time: 0.0028
(0.0028)
Accumulating evaluation results...
DONE (t=0.03s).
IoU metric: bbox
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.270
Average Precision  (AP) @[ IoU=0.50      | area=   all |
maxDets=100 ] = 0.728
Average Precision  (AP) @[ IoU=0.75      | area=   all |
maxDets=100 ] = 0.147
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 ] = 0.017
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.287
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.177
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.429
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.434
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 ] = 0.050
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.455
Epoch: [6] [ 0/60]  eta: 0:00:46  lr: 0.000050  loss: 0.2474 (0.2474)
loss_classifier: 0.0654 (0.0654)  loss_box_reg: 0.0965 (0.0965)
loss_objectness: 0.0725 (0.0725)  loss_rpn_box_reg: 0.0130 (0.0130)
time: 0.7773  data: 0.3169  max mem: 6278
Epoch: [6] [10/60]  eta: 0:00:19  lr: 0.000050  loss: 0.3108 (0.3436)
```

```
loss_classifier: 0.0956 (0.0967) loss_box_reg: 0.1464 (0.1666)
loss_objectness: 0.0545 (0.0570) loss_rpn_box_reg: 0.0219 (0.0233)
time: 0.3974 data: 0.0373 max mem: 6278
Epoch: [6] [20/60] eta: 0:00:14 lr: 0.000050 loss: 0.3093 (0.3258)
loss_classifier: 0.0921 (0.0933) loss_box_reg: 0.1338 (0.1539)
loss_objectness: 0.0438 (0.0587) loss_rpn_box_reg: 0.0201 (0.0200)
time: 0.3527 data: 0.0090 max mem: 6278
Epoch: [6] [30/60] eta: 0:00:10 lr: 0.000050 loss: 0.2626 (0.3180)
loss_classifier: 0.0798 (0.0925) loss_box_reg: 0.1196 (0.1511)
loss_objectness: 0.0405 (0.0548) loss_rpn_box_reg: 0.0155 (0.0196)
time: 0.3438 data: 0.0085 max mem: 6278
Epoch: [6] [40/60] eta: 0:00:07 lr: 0.000050 loss: 0.2778 (0.3163)
loss_classifier: 0.0903 (0.0931) loss_box_reg: 0.1457 (0.1528)
loss_objectness: 0.0387 (0.0516) loss_rpn_box_reg: 0.0180 (0.0188)
time: 0.3478 data: 0.0106 max mem: 6278
Epoch: [6] [50/60] eta: 0:00:03 lr: 0.000050 loss: 0.2581 (0.3138)
loss_classifier: 0.0822 (0.0927) loss_box_reg: 0.1319 (0.1517)
loss_objectness: 0.0387 (0.0507) loss_rpn_box_reg: 0.0159 (0.0186)
time: 0.3532 data: 0.0123 max mem: 6278
Epoch: [6] [59/60] eta: 0:00:00 lr: 0.000050 loss: 0.2581 (0.3161)
loss_classifier: 0.0870 (0.0941) loss_box_reg: 0.1319 (0.1523)
loss_objectness: 0.0403 (0.0506) loss_rpn_box_reg: 0.0180 (0.0192)
time: 0.3589 data: 0.0108 max mem: 6278
Epoch: [6] Total time: 0:00:21 (0.3615 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:20 model_time: 0.1321 (0.1321)
evaluator_time: 0.0280 (0.0280) time: 0.4028 data: 0.2416 max mem:
6278
Test: [49/50] eta: 0:00:00 model_time: 0.0347 (0.0452)
evaluator_time: 0.0017 (0.0039) time: 0.0447 data: 0.0049 max mem:
6278
Test: Total time: 0:00:03 (0.0673 s / it)
Averaged stats: model_time: 0.0347 (0.0452) evaluator_time: 0.0017
(0.0039)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.267
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.695
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.102
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 ] = 0.027
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.288
```

Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=1] = 0.172
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=10] = 0.423
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.430
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] = 0.050
Average Recall (AR) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.451
Epoch: [7] [0/60] eta: 0:00:39 lr: 0.000050 loss: 0.2034 (0.2034)
loss_classifier: 0.0653 (0.0653) loss_box_reg: 0.0958 (0.0958)
loss_objectness: 0.0362 (0.0362) loss_rpn_box_reg: 0.0062 (0.0062)
time: 0.6563 data: 0.2897 max mem: 6278
Epoch: [7] [10/60] eta: 0:00:18 lr: 0.000050 loss: 0.3023 (0.2956)
loss_classifier: 0.0895 (0.0870) loss_box_reg: 0.1510 (0.1408)
loss_objectness: 0.0408 (0.0491) loss_rpn_box_reg: 0.0195 (0.0187)
time: 0.3682 data: 0.0332 max mem: 6278
Epoch: [7] [20/60] eta: 0:00:14 lr: 0.000050 loss: 0.3114 (0.3151)
loss_classifier: 0.0895 (0.0922) loss_box_reg: 0.1672 (0.1530)
loss_objectness: 0.0459 (0.0510) loss_rpn_box_reg: 0.0200 (0.0189)
time: 0.3433 data: 0.0087 max mem: 6278
Epoch: [7] [30/60] eta: 0:00:10 lr: 0.000050 loss: 0.3334 (0.3216)
loss_classifier: 0.0915 (0.0950) loss_box_reg: 0.1672 (0.1560)
loss_objectness: 0.0463 (0.0513) loss_rpn_box_reg: 0.0203 (0.0193)
time: 0.3530 data: 0.0110 max mem: 6278
Epoch: [7] [40/60] eta: 0:00:07 lr: 0.000050 loss: 0.3095 (0.3246)
loss_classifier: 0.0978 (0.0951) loss_box_reg: 0.1574 (0.1568)
loss_objectness: 0.0424 (0.0529) loss_rpn_box_reg: 0.0173 (0.0199)
time: 0.3550 data: 0.0113 max mem: 6278
Epoch: [7] [50/60] eta: 0:00:03 lr: 0.000050 loss: 0.2939 (0.3183)
loss_classifier: 0.0868 (0.0941) loss_box_reg: 0.1395 (0.1523)
loss_objectness: 0.0406 (0.0531) loss_rpn_box_reg: 0.0138 (0.0188)
time: 0.3474 data: 0.0101 max mem: 6278
Epoch: [7] [59/60] eta: 0:00:00 lr: 0.000050 loss: 0.2939 (0.3210)
loss_classifier: 0.0868 (0.0949) loss_box_reg: 0.1421 (0.1543)
loss_objectness: 0.0489 (0.0531) loss_rpn_box_reg: 0.0138 (0.0187)
time: 0.3469 data: 0.0093 max mem: 6278
Epoch: [7] Total time: 0:00:21 (0.3551 s / it)
creating index...
index created!
Test: [0/50] eta: 0:00:14 model_time: 0.0970 (0.0970)
evaluator_time: 0.0068 (0.0068) time: 0.2935 data: 0.1886 max mem: 6278
Test: [49/50] eta: 0:00:00 model_time: 0.0366 (0.0398)
evaluator_time: 0.0017 (0.0023) time: 0.0436 data: 0.0040 max mem: 6278
Test: Total time: 0:00:02 (0.0531 s / it)

Averaged stats: model_time: 0.0366 (0.0398) evaluator_time: 0.0017 (0.0023)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.275
Average Precision (AP) @[IoU=0.50 | area= all | maxDets=100] = 0.702
Average Precision (AP) @[IoU=0.75 | area= all | maxDets=100] = 0.128
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] = 0.025
Average Precision (AP) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.290
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=1] = 0.186
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=10] = 0.414
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] = 0.421
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] = 0.033
Average Recall (AR) @[IoU=0.50:0.95 | area= large | maxDets=100] = 0.442
Epoch: [8] [0/60] eta: 0:00:39 lr: 0.000050 loss: 0.3311 (0.3311)
loss_classifier: 0.0962 (0.0962) loss_box_reg: 0.1863 (0.1863)
loss_objectness: 0.0330 (0.0330) loss_rpn_box_reg: 0.0155 (0.0155)
time: 0.6578 data: 0.2737 max mem: 6278
Epoch: [8] [10/60] eta: 0:00:18 lr: 0.000050 loss: 0.3344 (0.3211)
loss_classifier: 0.1082 (0.0967) loss_box_reg: 0.1673 (0.1581)
loss_objectness: 0.0471 (0.0465) loss_rpn_box_reg: 0.0186 (0.0198)
time: 0.3780 data: 0.0327 max mem: 6278
Epoch: [8] [20/60] eta: 0:00:14 lr: 0.000050 loss: 0.2744 (0.2999)
loss_classifier: 0.0813 (0.0886) loss_box_reg: 0.1388 (0.1466)
loss_objectness: 0.0367 (0.0458) loss_rpn_box_reg: 0.0186 (0.0189)
time: 0.3502 data: 0.0097 max mem: 6278
Epoch: [8] [30/60] eta: 0:00:10 lr: 0.000050 loss: 0.2744 (0.3081)
loss_classifier: 0.0802 (0.0889) loss_box_reg: 0.1421 (0.1502)
loss_objectness: 0.0386 (0.0498) loss_rpn_box_reg: 0.0158 (0.0191)
time: 0.3506 data: 0.0111 max mem: 6278
Epoch: [8] [40/60] eta: 0:00:07 lr: 0.000050 loss: 0.3263 (0.3196)
loss_classifier: 0.0879 (0.0937) loss_box_reg: 0.1543 (0.1574)
loss_objectness: 0.0450 (0.0487) loss_rpn_box_reg: 0.0201 (0.0198)
time: 0.3504 data: 0.0106 max mem: 6278
Epoch: [8] [50/60] eta: 0:00:03 lr: 0.000050 loss: 0.2811 (0.3132)

```

loss_classifier: 0.0879 (0.0927) loss_box_reg: 0.1472 (0.1537)
loss_objectness: 0.0410 (0.0480) loss_rpn_box_reg: 0.0181 (0.0188)
time: 0.3482 data: 0.0096 max mem: 6278
Epoch: [8] [59/60] eta: 0:00:00 lr: 0.000050 loss: 0.2718 (0.3158)
loss_classifier: 0.0832 (0.0937) loss_box_reg: 0.1239 (0.1541)
loss_objectness: 0.0410 (0.0489) loss_rpn_box_reg: 0.0171 (0.0190)
time: 0.3461 data: 0.0090 max mem: 6278
Epoch: [8] Total time: 0:00:21 (0.3572 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:14 model_time: 0.0940 (0.0940)
evaluator_time: 0.0028 (0.0028) time: 0.2921 data: 0.1940 max mem:
6278
Test: [49/50] eta: 0:00:00 model_time: 0.0369 (0.0399)
evaluator_time: 0.0017 (0.0023) time: 0.0450 data: 0.0043 max mem:
6278
Test: Total time: 0:00:02 (0.0535 s / it)
Averaged stats: model_time: 0.0369 (0.0399) evaluator_time: 0.0017
(0.0023)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.279
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.711
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.164
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 ] = 0.019
Average Precision (AP) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.295
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
1 ] = 0.196
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=
10 ] = 0.442
Average Recall (AR) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.450
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 ] = 0.067
Average Recall (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.471
Epoch: [9] [ 0/60] eta: 0:00:37 lr: 0.000005 loss: 0.1967 (0.1967)
loss_classifier: 0.0660 (0.0660) loss_box_reg: 0.0956 (0.0956)
loss_objectness: 0.0223 (0.0223) loss_rpn_box_reg: 0.0128 (0.0128)
time: 0.6292 data: 0.2335 max mem: 6278

```

```

Epoch: [9] [10/60] eta: 0:00:18 lr: 0.000005 loss: 0.2469 (0.3048)
loss_classifier: 0.0784 (0.0951) loss_box_reg: 0.1239 (0.1456)
loss_objectness: 0.0334 (0.0465) loss_rpn_box_reg: 0.0142 (0.0176)
time: 0.3633 data: 0.0285 max mem: 6278
Epoch: [9] [20/60] eta: 0:00:14 lr: 0.000005 loss: 0.2772 (0.3017)
loss_classifier: 0.0824 (0.0937) loss_box_reg: 0.1249 (0.1422)
loss_objectness: 0.0445 (0.0472) loss_rpn_box_reg: 0.0156 (0.0185)
time: 0.3453 data: 0.0103 max mem: 6278
Epoch: [9] [30/60] eta: 0:00:10 lr: 0.000005 loss: 0.3159 (0.3203)
loss_classifier: 0.0984 (0.0988) loss_box_reg: 0.1470 (0.1537)
loss_objectness: 0.0487 (0.0484) loss_rpn_box_reg: 0.0203 (0.0195)
time: 0.3614 data: 0.0115 max mem: 6278
Epoch: [9] [40/60] eta: 0:00:07 lr: 0.000005 loss: 0.2838 (0.3066)
loss_classifier: 0.0934 (0.0943) loss_box_reg: 0.1253 (0.1467)
loss_objectness: 0.0431 (0.0470) loss_rpn_box_reg: 0.0190 (0.0187)
time: 0.3572 data: 0.0100 max mem: 6278
Epoch: [9] [50/60] eta: 0:00:03 lr: 0.000005 loss: 0.2528 (0.3136)
loss_classifier: 0.0791 (0.0946) loss_box_reg: 0.1199 (0.1483)
loss_objectness: 0.0407 (0.0516) loss_rpn_box_reg: 0.0174 (0.0191)
time: 0.3480 data: 0.0094 max mem: 6278
Epoch: [9] [59/60] eta: 0:00:00 lr: 0.000005 loss: 0.2897 (0.3173)
loss_classifier: 0.0832 (0.0951) loss_box_reg: 0.1463 (0.1523)
loss_objectness: 0.0408 (0.0507) loss_rpn_box_reg: 0.0167 (0.0191)
time: 0.3527 data: 0.0089 max mem: 6278
Epoch: [9] Total time: 0:00:21 (0.3585 s / it)
creating index...
index created!
Test: [ 0/50] eta: 0:00:13 model_time: 0.0848 (0.0848)
evaluator_time: 0.0026 (0.0026) time: 0.2723 data: 0.1838 max mem:
6278
Test: [49/50] eta: 0:00:00 model_time: 0.0375 (0.0394)
evaluator_time: 0.0016 (0.0024) time: 0.0441 data: 0.0041 max mem:
6278
Test: Total time: 0:00:02 (0.0532 s / it)
Averaged stats: model_time: 0.0375 (0.0394) evaluator_time: 0.0016
(0.0024)
Accumulating evaluation results...
DONE (t=0.01s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all |
maxDets=100 ] = 0.278
Average Precision (AP) @[ IoU=0.50 | area= all |
maxDets=100 ] = 0.709
Average Precision (AP) @[ IoU=0.75 | area= all |
maxDets=100 ] = 0.139
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 ] = 0.295
Average Recall      (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
1 ] = 0.179
Average Recall      (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=
10 ] = 0.420
Average Recall      (AR) @[ IoU=0.50:0.95 | area=   all |
maxDets=100 ] = 0.434
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 ] = 0.067
Average Recall      (AR) @[ IoU=0.50:0.95 | area= large |
maxDets=100 ] = 0.454
That's it!

```

###b. How do the performance of the two models compare on the training data after 10 epochs?

```

from PIL import Image, ImageFont, ImageDraw
def drawBoxes(img,boxes):
    im = img.copy().convert("RGBA")
    draw = ImageDraw.Draw(im)
    for i in range(0,len(boxes)):
        draw.rectangle(boxes[i], fill=None, width=1, outline ="red")
    return im

# pick one image from the test set
img, _ = dataset_test[4]
# put the model in evaluation mode
model.eval()
with torch.no_grad():
    prediction = model([img.to(device)])

model_backbone.eval()
with torch.no_grad():
    prediction_backbone = model_backbone([img.to(device)])

test_img = Image.fromarray(img.mul(255).permute(1, 2,
0).byte().numpy())

Prediction without backbone network
prediction

[{'boxes': tensor([[330.1726,  64.0635, 437.8224, 323.6576],
                  [193.2116,  52.1087, 293.4012, 333.4836],
                  [138.6928,  76.6964, 202.0652, 253.8754],
                  [282.4706,  77.2592, 329.7892, 254.6026],
                  [430.3847,  71.1977, 451.0906, 153.4053],
                  [141.2240,  57.8479, 236.8255, 317.7221]]), device='cuda:0'),
  'labels': tensor([1, 1, 1, 1, 1, 1], device='cuda:0'),
  'scores': tensor([0.9981, 0.9979, 0.9946, 0.9794, 0.8290, 0.0671],

```

```
device='cuda:0'),
'masks': tensor([[[[0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    ...,
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.]]],
```

```
[[[0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   ...,
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.]])],
```

```
[[[0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   ...,
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.]])],
```

```
[[[0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   ...,
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.]])],
```

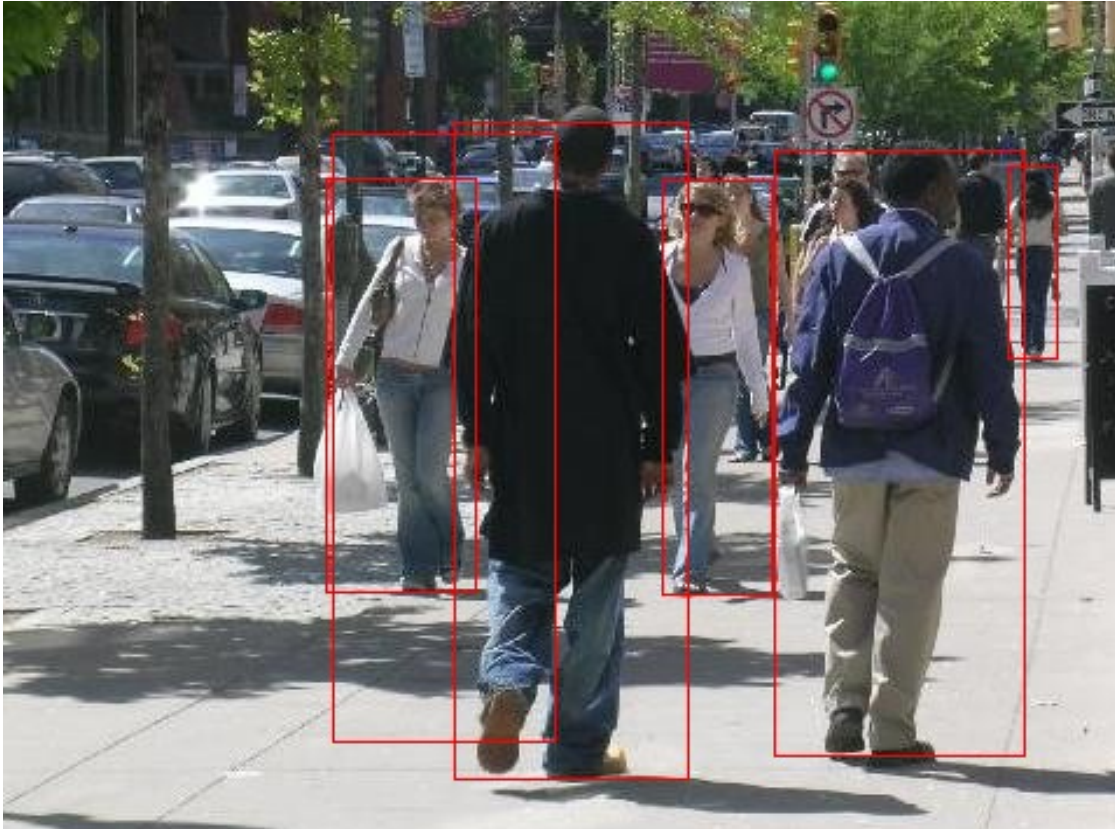
```
[[[0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   ...,
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.]])],
```

```
[[[0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   [0., 0., 0., ..., 0., 0., 0.],
   ...,
```

```
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.]]]], device='cuda:0'))]
```

#without backbone

```
drawBoxes(test_img,prediction[0]['boxes'].cpu().detach().numpy())
```



Prediction with backbone network

```
prediction_backbone
```

```
[{'boxes': tensor([[180.8176, 45.6307, 304.0180, 336.9954],
[158.7479, 67.9997, 224.2478, 297.6407],
[345.8654, 55.3600, 427.2272, 326.0550],
[284.5453, 64.4678, 342.1197, 271.4695],
[209.5271, 108.8732, 263.2868, 333.5840],
[124.9316, 73.2286, 210.1519, 273.9389],
[246.5954, 53.0289, 328.1284, 291.4227],
[191.8702, 170.4757, 290.2192, 336.8623],
[225.1544, 96.2929, 295.8960, 331.6685],
[378.3731, 20.9584, 445.1135, 331.4207],
[177.6187, 59.2855, 249.0703, 318.5334],
[111.0978, 41.5289, 269.6901, 336.9180],
[261.9694, 39.1867, 372.1928, 326.0304],
[324.8417, 28.8826, 400.1929, 329.7933],
[ 59.3918, 43.7860, 164.4137, 282.8600],
[335.9362, 147.4207, 397.3743, 335.3465],
```

```

        [ 44.0527,  67.6139, 116.5389, 274.3302],
        [ 14.2720,   8.8403,  93.8099, 289.7268]], device='cuda:0'),
    'labels': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1]), device='cuda:0'),
    'scores': tensor([0.7221, 0.6568, 0.6222, 0.5927, 0.5405, 0.5158,
0.4740, 0.3587, 0.3214,
                    0.2581, 0.2381, 0.2309, 0.2184, 0.1489, 0.1459, 0.0826,
0.0699, 0.0569]),
                    device='cuda:0')}}]

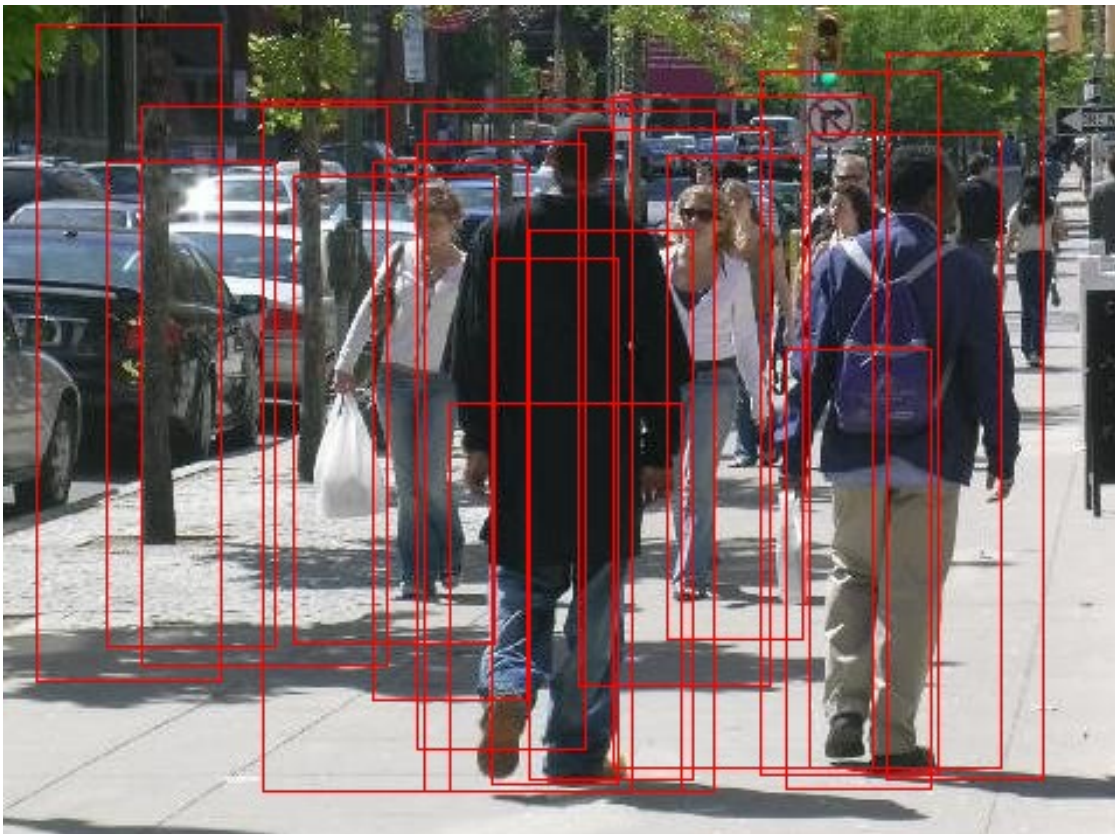
```

#with backbone

```

drawBoxes(test_img, prediction_backbone[0]
['boxes'].cpu().detach().numpy())

```



As is clear in the visualizations, the performance of the finetuned pretrained model (Option 1) is superior to the network with a different backbone (Option 2), which can be attributed to the fact that the former has a higher number of trainable parameters than the latter. This enables the finetuned model to accurately detect people with greater confidence, while the other model is less certain in its predictions.

###c. Test both models on this image of the Beatles. Be careful to make sure the dimensions align and the scaling is similar. How do the two models compare in terms of performance on this image?

```

import torchvision
from torchvision import transforms
im = Image.open('/content/Beatles_-_Abbey_Road.jpeg')
convert_tensor = transforms.ToTensor()
img = convert_tensor(im)

model.eval()
with torch.no_grad():
    beatles_prediction = model([img.to(device)])

model_backbone.eval()
with torch.no_grad():
    beatles_prediction_backbone = model_backbone([img.to(device)])

Prediction without backbone network
beatles_prediction

[{'boxes': tensor([[ 93.7469, 162.8455, 144.3531, 270.8776],
                  [151.9780, 168.5377, 211.8375, 278.5693],
                  [ 19.6899, 158.0344,  82.2313, 270.6464],
                  [223.2870, 163.0532, 283.4902, 288.0374],
                  [134.6620, 136.9727, 214.6796, 303.5094],
                  [ 26.2600, 154.1760, 171.0742, 273.6194],
                  [112.0801, 156.8865, 285.2253, 283.6689]]), device='cuda:0'),
 'labels': tensor([1, 1, 1, 1, 1, 1, 1], device='cuda:0'),
 'scores': tensor([0.9932, 0.9819, 0.9762, 0.9523, 0.1054, 0.1022,
0.0973],
                  device='cuda:0'),
 'masks': tensor([[[[0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    ...,
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.]]],

                  [[0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    ...,
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.]]],

                  [[0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    ...,
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.],
                    [0., 0., 0., ..., 0., 0., 0.]]],

```

```

        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]],

    ...,

    [[ [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]],

    [[ [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]],

    [[ [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]], device='cuda:0')}]

```

#Without backbone

```

boxes = beatles_prediction[0]['boxes'].cpu().detach().numpy()
drawBoxes(im,boxes)

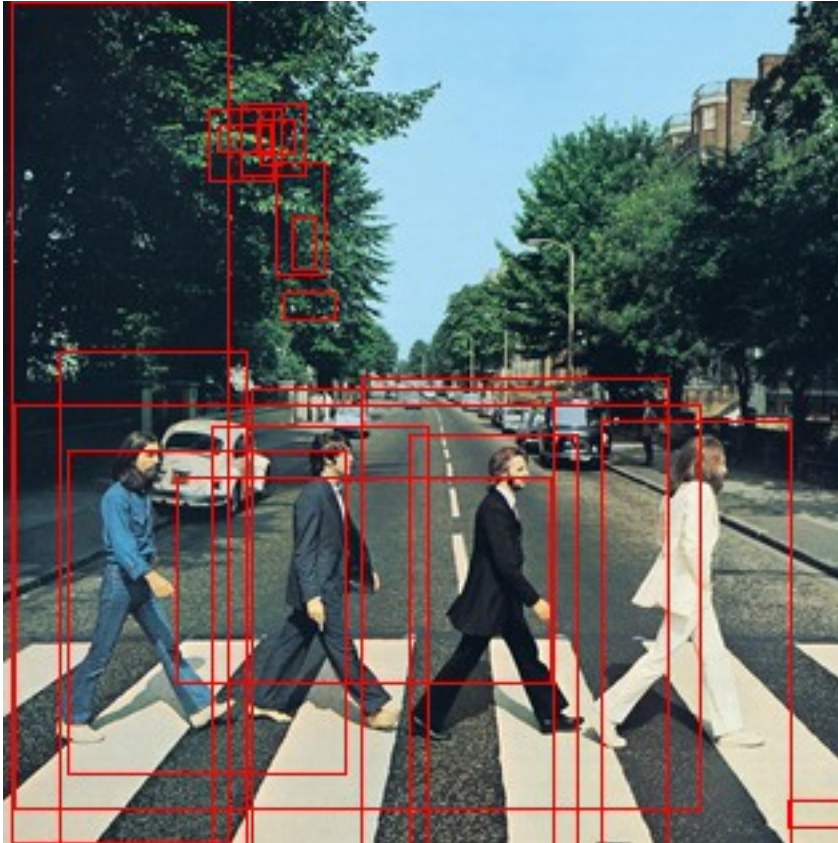
```



```
'scores': tensor([0.6225, 0.5441, 0.4024, 0.3633, 0.2854, 0.2472,
0.2228, 0.2116, 0.0991,
0.0933, 0.0706, 0.0661, 0.0603, 0.0598, 0.0533, 0.0528,
0.0518, 0.0504,
0.0503], device='cuda:0'))]
```

#With backbone

```
boxes_backbone = beatles_prediction_backbone[0]
['boxes'].cpu().detach().numpy()
drawBoxes(im, boxes_backbone)
```



Again, the finetuned model (option 1) performs better than model with a different backbone (option 2). The confidence scores are better for the fine tuned model. There are a lot of mispredictions in the latter model with low confidence scores. Training both these models for a higher number of epochs can improve the accuracy of detection.