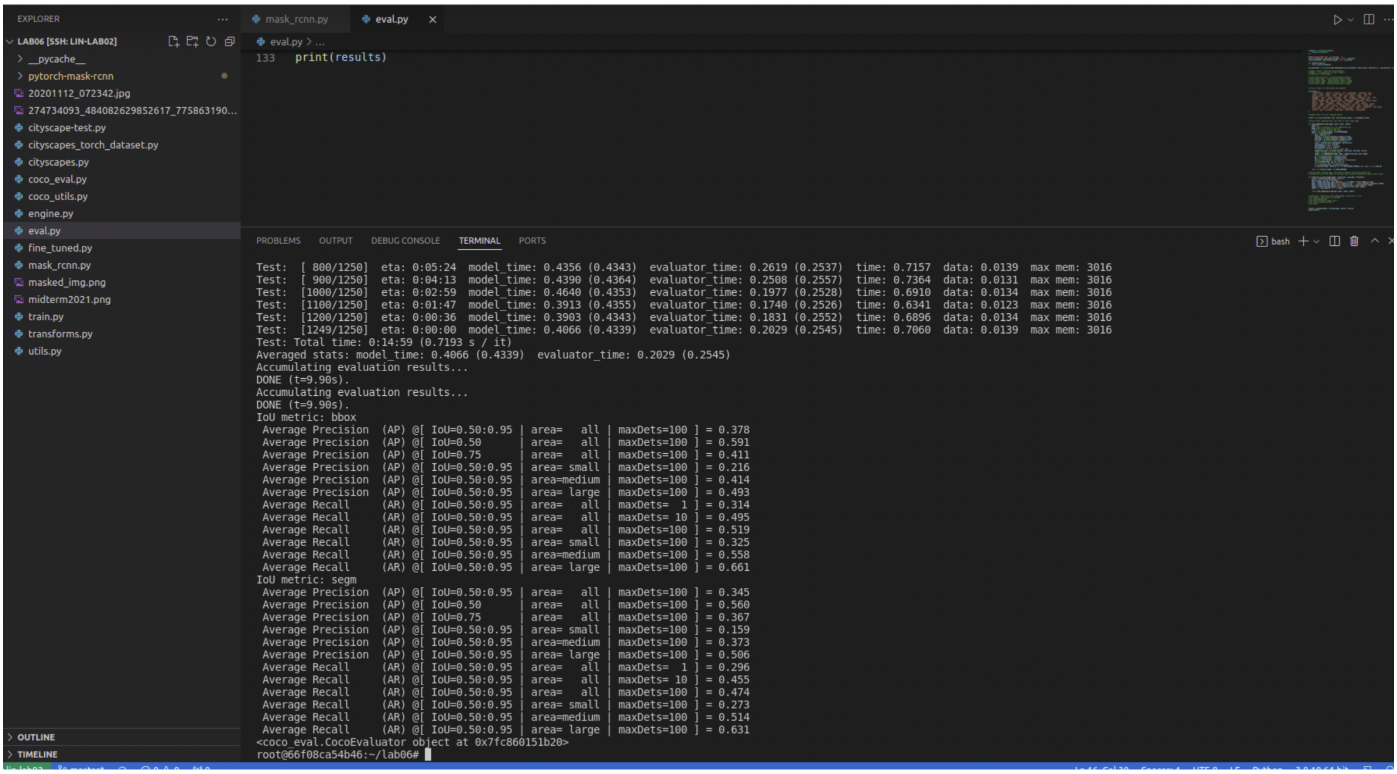
Lab06 Report

st122314

In this report I learnt about Masked-RCNN using COCO dataset. Moreover, I also tested with Cityscapses data with Pre-train mdoel and then leant how to make fine-tuning.

The follwing figure is the report after running COCO eval.py.



```
print(results)
```

```
Test:  [ 800/1250]  eta: 0:05:24  model_time: 0.4356 (0.4343)  evaluator_time: 0.2619 (0.2537)  time: 0.7157  data: 0.0139  max mem: 3016
Test:  [ 900/1250]  eta: 0:04:13  model_time: 0.4390 (0.4364)  evaluator_time: 0.2508 (0.2557)  time: 0.7364  data: 0.0131  max mem: 3016
Test:  [1000/1250]  eta: 0:02:59  model_time: 0.4640 (0.4353)  evaluator_time: 0.1977 (0.2528)  time: 0.6910  data: 0.0134  max mem: 3016
Test:  [1100/1250]  eta: 0:01:47  model_time: 0.3913 (0.4355)  evaluator_time: 0.1740 (0.2526)  time: 0.6341  data: 0.0123  max mem: 3016
Test:  [1200/1250]  eta: 0:00:36  model_time: 0.3903 (0.4343)  evaluator_time: 0.1831 (0.2552)  time: 0.6896  data: 0.0134  max mem: 3016
Test:  [1249/1250]  eta: 0:00:00  model_time: 0.4066 (0.4339)  evaluator_time: 0.2029 (0.2545)  time: 0.7060  data: 0.0139  max mem: 3016
Test: Total time: 0:14:59 (0.7193 s / it)
Averaged stats: model_time: 0.4066 (0.4339)  evaluator_time: 0.2029 (0.2545)
Accumulating evaluation results...
DONE (t=9.90s).
Accumulating evaluation results...
DONE (t=9.90s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.378
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.591
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.411
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.216
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.414
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.493
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.314
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.495
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.519
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.325
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.558
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.661
IoU metric: segm
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.345
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.560
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.367
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.159
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.373
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.506
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.296
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.455
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.474
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.273
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.514
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.631
<coco_eval.CocoEvaluator object at 0x7fc860151b20>
root@66f08ca54b46:~/lab06#
```

In [ ]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [ ]:

```
cd /content/drive/MyDrive/RTML/Midsem_prep/data
```

```
/content/drive/MyDrive/RTML/Midsem_prep/data
```

CityScape data download

In [ ]:

```
!wget --keep-session-cookies --save-cookies=cookies.txt --post-data 'username=winwir
```

```
--2022-03-03 10:37:54--  https://www.cityscapes-dataset.com/login/ (ht
tps://www.cityscapes-dataset.com/login/)
Resolving www.cityscapes-dataset.com (www.cityscapes-dataset.com)... 1
39.19.217.8
Connecting to www.cityscapes-dataset.com (www.cityscapes-dataset.com)|
139.19.217.8|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://www.cityscapes-dataset.com/downloads/ (https://www.c
ityscapes-dataset.com/downloads/) [following]
--2022-03-03 10:37:55--  https://www.cityscapes-dataset.com/downloads/
(https://www.cityscapes-dataset.com/downloads/)
Reusing existing connection to www.cityscapes-dataset.com:443.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘index.html.1’

index.html.1            [ <=>                ]  47.44K  --.-KB/s    in
0.1s

2022-03-03 10:37:55 (393 KB/s) - ‘index.html.1’ saved [48580]
```

In [ ]:

```
!wget --load-cookies cookies.txt --content-disposition https://www.cityscapes-datase
```

```
--2022-03-03 10:37:59--  https://www.cityscapes-dataset.com/file-handl
ing/?packageID=1 (https://www.cityscapes-dataset.com/file-handling/?pa
ckageID=1)
Resolving www.cityscapes-dataset.com (www.cityscapes-dataset.com)... 1
39.19.217.8
Connecting to www.cityscapes-dataset.com (www.cityscapes-dataset.com)|
139.19.217.8|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 252567705 (241M) [application/octet-stream]
Saving to: ‘gtFine_trainvaltest.zip’

gtFine_trainvaltest 100%[===================>] 240.87M  24.9MB/s    in
11s

2022-03-03 10:38:11 (21.2 MB/s) - ‘gtFine_trainvaltest.zip’ saved [252
567705/252567705]
```

In [ ]:

```
!wget --load-cookies cookies.txt --content-disposition https://www.cityscapes-datase
```

```
--2022-03-03 10:38:31--  https://www.cityscapes-dataset.com/file-handl
ing/?packageID=3 (https://www.cityscapes-dataset.com/file-handling/?pa
ckageID=3)
Resolving www.cityscapes-dataset.com (www.cityscapes-dataset.com)... 1
39.19.217.8
Connecting to www.cityscapes-dataset.com (www.cityscapes-dataset.com)|
139.19.217.8|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11592327197 (11G) [application/octet-stream]
Saving to: ‘leftImg8bit_trainvaltest.zip’

leftImg8bit_trainva 100%[===================>]  10.80G  22.2MB/s    in
7m 24s

2022-03-03 10:45:55 (24.9 MB/s) - ‘leftImg8bit_trainvaltest.zip’ saved
[11592327197/11592327197]
```

In [ ]:

```
ls
```

```
 AlexNet.ipynb                GAN.ipynb                      List.gdoc
 cityscape_mask_rcnn.ipynb    GoogLeNet.ipynb                Masked-R-C
NN.ipynb
 Collection.ipynb             index.html                    'Resnet&SEn
et.ipynb'
 cookies.txt                  index.html.1                   YOLO.ipynb
 data/                        leftImg8bit_trainvaltest.zip
```

In [ ]:

```
cd data/
```

```
/content/drive/MyDrive/RTML/Midsem_prep/data
```

In [ ]:

```
!unzip gtFine_trainvaltest.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: gtFine/test/berlin/berlin_000117_000019_gtFine_color.png
  inflating: gtFine/test/berlin/berlin_000114_000019_gtFine_color.png
  inflating: gtFine/test/berlin/berlin_000434_000019_gtFine_labelIds.p
ng
  inflating: gtFine/test/berlin/berlin_000420_000019_gtFine_color.png
  inflating: gtFine/test/berlin/berlin_000483_000019_gtFine_instanceId
s.png
  inflating: gtFine/test/berlin/berlin_000420_000019_gtFine_instanceId
s.png
  inflating: gtFine/test/berlin/berlin_000254_000019_gtFine_color.png
  inflating: gtFine/test/berlin/berlin_000490_000019_gtFine_color.png
  inflating: gtFine/test/berlin/berlin_000448_000019_gtFine_polygons.j
son
  inflating: gtFine/test/berlin/berlin_000099_000019_gtFine_labelIds.p
ng
  inflating: gtFine/test/berlin/berlin_000068_000019_gtFine_instanceId
s.png
  inflating: gtFine/test/berlin/berlin_000288_000019_gtFine_instanceId
```

In [ ]:

```
!unzip leftImg8bit_trainvaltest.zip
```

```
Streaming output truncated to the last 5000 lines.
 extracting: leftImg8bit/train/jena/jena_000074_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000040_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000020_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000030_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000005_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000059_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000100_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000034_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000089_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000104_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000107_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000080_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000006_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000082_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000044_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000026_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000115_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000023_000019_leftImg8bit.png
 extracting: leftImg8bit/train/jena/jena_000085_000019_leftImg8bit.png
```

In [ ]:

```

```

Cityscape run on mask rcnn

In [ ]:

```
import torch
import torchvision
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
from torchvision.datasets import Cityscapes


print('Loading pretrained model...')
model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True).cuda()
model.eval()
```

```
Loading pretrained model...
```

In [ ]:

```
dom
py as np
mport Image
plotlib.pyplot as plt
ch
.utils.data import Dataset, DataLoader
vision import transforms, utils
chvision.transforms.functional as TF
ctions import namedtuple


scapesDataset(Dataset):

init__(self, root, split='train', mode='fine', augment=False):

lf.root = os.path.expanduser(root)
lf.mode = 'gtFine' if mode == 'fine' else 'gtCoarse'
lf.images_dir = os.path.join(self.root, 'leftImg8bit', split)
lf.targets_dir = os.path.join(self.root, self.mode, split)
int(self.images_dir)
int(self.targets_dir)
lf.split = split
lf.augment = augment
lf.images = []
lf.targets = []
lf.mapping = {
  0: 0,  # unlabeled
  1: 0,  # ego vehicle
  2: 0,  # rect border
  3: 0,  # out of roi
  4: 0,  # static
  5: 0,  # dynamic
  6: 0,  # ground
  7: 1,  # road
  8: 0,  # sidewalk
  9: 0,  # parking
  10: 0,  # rail track
  11: 0,  # building
  12: 0,  # wall
  13: 0,  # fence
  14: 0,  # guard rail
  15: 0,  # bridge
  16: 0,  # tunnel
  17: 0,  # pole
  18: 0,  # polegroup
  19: 0,  # traffic light
  20: 0,  # traffic sign
  21: 0,  # vegetation
  22: 0,  # terrain
  23: 2,  # sky
  24: 0,  # person
  25: 0,  # rider
  26: 3,  # car
  27: 0,  # truck
  28: 0,  # bus
  29: 0,  # caravan
  30: 0,  # trailer
  31: 0,  # train
  32: 0,  # motorcycle
  33: 0,  # bicycle
  -1: 0  # licenseplate

lf.mappingrgb = {
  0: (255, 0, 0),  # unlabeled
  1: (255, 0, 0),  # ego vehicle
  2: (255, 0, 0),  # rect border
  3: (255, 0, 0),  # out of roi
  4: (255, 0, 0),  # static
  5: (255, 0, 0),  # dynamic
  6: (255, 0, 0),  # ground
  7: (0, 255, 0),  # road
  8: (255, 0, 0),  # sidewalk
  9: (255, 0, 0),  # parking
  10: (255, 0, 0),  # rail track
  11: (255, 0, 0),  # building
  12: (255, 0, 0),  # wall
  13: (255, 0, 0),  # fence
  14: (255, 0, 0),  # guard rail
  15: (255, 0, 0),  # bridge
  16: (255, 0, 0),  # tunnel
  17: (255, 0, 0),  # pole
  18: (255, 0, 0),  # polegroup
  19: (255, 0, 0),  # traffic light
  20: (255, 0, 0),  # traffic sign
  21: (255, 0, 0),  # vegetation
  22: (255, 0, 0),  # terrain
  23: (0, 0, 255),  # sky
  24: (255, 0, 0),  # person
  25: (255, 0, 0),  # rider
  26: (255, 255, 0),  # car
  27: (255, 0, 0),  # truck
  28: (255, 0, 0),  # bus
  29: (255, 0, 0),  # caravan
  30: (255, 0, 0),  # trailer
  31: (255, 0, 0),  # train
  32: (255, 0, 0),  # motorcycle
  33: (255, 0, 0),  # bicycle
  -1: (255, 0, 0)  # licenseplate


Ensure that this matches the above mapping!#!@#!@#
For example 4 classes, means we should map to the ids=(0,1,2,3)
This is used to specify how many outputs the network should product...
lf.num_classes = 4

=============================================
Check that inputs are valid
=============================================
 mode not in ['fine', 'coarse']:
  raise ValueError('Invalid mode! Please use mode="fine" or mode="coarse"')
 mode == 'fine' and split not in ['train', 'test', 'val']:
  raise ValueError('Invalid split for mode "fine"! Please use split="train", split="
if mode == 'coarse' and split not in ['train', 'train_extra', 'val']:
  raise ValueError('Invalid split for mode "coarse"! Please use split="train", split
 not os.path.isdir(self.images_dir) or not os.path.isdir(self.targets_dir):
  raise RuntimeError('Dataset not found or incomplete. Please make sure all required
                     ' specified "split" and "mode" are inside the "root" directory'

=============================================
Read in the paths to all images
=============================================
r city in os.listdir(self.images_dir):
  img_dir = os.path.join(self.images_dir, city)
  target_dir = os.path.join(self.targets_dir, city)
  for file_name in os.listdir(img_dir):
      self.images.append(os.path.join(img_dir, file_name))
      target_name = '{}_{}'.format(file_name.split('_leftImg8bit')[0], '{}_labelIds.
      # target_name = '{}_{}'.format(file_name.split('_leftImg8bit')[0], '{}_color.p
      self.targets.append(os.path.join(target_dir, target_name))

repr__(self):
t_str = 'Dataset ' + self.__class__.__name__ + '\n'
t_str += '    Number of images: {}\n'.format(self.__len__())
t_str += '    Split: {}\n'.format(self.split)
t_str += '    Mode: {}\n'.format(self.mode)
t_str += '    Augment: {}\n'.format(self.augment)
t_str += '    Root Location: {}\n'.format(self.root)
turn fmt_str

len__(self):
turn len(self.images)

sk_to_class(self, mask):
'
ven the cityscapes dataset, this maps to a 0..classes numbers.
is is because we are using a subset of all masks, so we have this "mapping" function
is mapping function is used to map all the standard ids into the smaller subset.
'
skimg = torch.zeros((mask.size()[0], mask.size()[1]), dtype=torch.uint8)
r k in self.mapping:
  maskimg[mask == k] = self.mapping[k]
turn maskimg

sk_to_rgb(self, mask):
'
ven the Cityscapes mask file, this converts the ids into rgb colors.
is is needed as we are interested in a sub-set of labels, thus can't just use the
andard color output provided by the dataset.
'
bimg = torch.zeros((3, mask.size()[0], mask.size()[1]), dtype=torch.uint8)
r k in self.mappingrgb:
  rgbimg[0][mask == k] = self.mappingrgb[k][0]
  rgbimg[1][mask == k] = self.mappingrgb[k][1]
  rgbimg[2][mask == k] = self.mappingrgb[k][2]
turn rgbimg

ass_to_rgb(self, mask):
'
is function maps the classification index ids into the rgb.
r example after the argmax from the network, you want to find what class
given pixel belongs too. This does that but just changes the color
 that we can compare it directly to the rgb groundtruth label.
'
sk2class = dict((v, k) for k, v in self.mapping.items())
bimg = torch.zeros((3, mask.size()[0], mask.size()[1]), dtype=torch.uint8)
r k in mask2class:
  rgbimg[0][mask == k] = self.mappingrgb[mask2class[k]][0]
  rgbimg[1][mask == k] = self.mappingrgb[mask2class[k]][1]
  rgbimg[2][mask == k] = self.mappingrgb[mask2class[k]][2]
turn rgbimg
```

```
getitem__(self, index):

first load the RGB image
age = Image.open(self.images[index]).convert('RGB')

next load the target
rget = Image.open(self.targets[index]).convert('L')



convert to pytorch tensors
target = TF.to_tensor(target)
rget = torch.from_numpy(np.array(target, dtype=np.uint8))
age = TF.to_tensor(image)

convert the labels into a mask
rgetrgb = self.mask_to_rgb(target)
rgetmask = self.mask_to_class(target)
rgetmask = targetmask.long()
rgetrgb = targetrgb.long()

finally return the image pair
return image, targetmask, targetrgb
turn image, targetmask
```

In [ ]:

```
cityscapes_path = "data/"

print('Loading Cityscapes val datasets...')
dataset = CityscapesDataset(cityscapes_path, split='val', mode='fine', augment=False

print(dataset)

val_dataloader = torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=True, nu
```

```
Loading Cityscapes val datasets...
data/leftImg8bit/val
data/gtFine/val
Dataset CityscapesDataset
    Number of images: 500
    Split: val
    Mode: gtFine
    Augment: False
    Root Location: data/
```

In [ ]:

```
images, targetmask = next(iter(val_dataloader))
print(images.shape)
print(targetmask.shape)

images = [ img.cuda() for img in images ]
print(type(images))
print(images)
predictions = model(images)


print('Prediction keys:', predictions[0].keys())
print('Boxes shape:', predictions[0]['boxes'].shape)
print('Labels shape:', predictions[0]['labels'].shape)
print('Scores shape:', predictions[0]['scores'].shape)
print('Masks shape:', predictions[0]['masks'].shape)
```

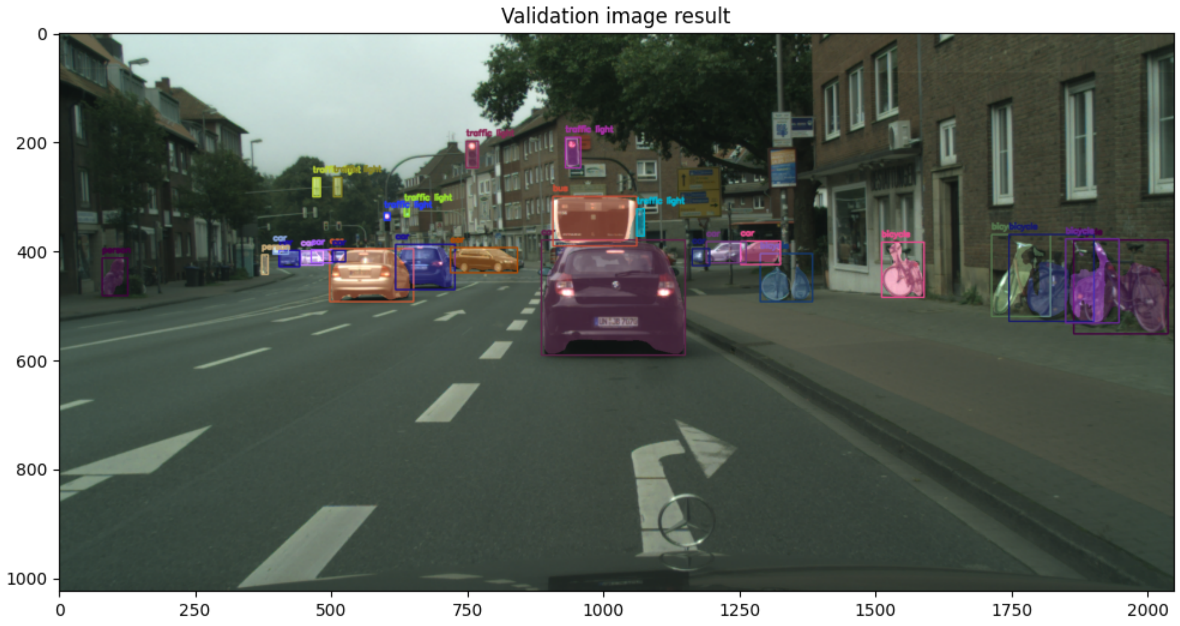
```
torch.Size([1, 3, 1024, 2048])
torch.Size([1, 1024, 2048])
<class 'list'>
[tensor([[[0.3804, 0.3686, 0.3686,  ..., 0.2745, 0.2784, 0.2745],
         [0.3608, 0.3608, 0.3608,  ..., 0.2784, 0.2745, 0.2706],
         [0.3412, 0.3490, 0.3490,  ..., 0.2784, 0.2706, 0.2627],
         ...,
         [0.2039, 0.1961, 0.1922,  ..., 0.1608, 0.1608, 0.1569],
         [0.2039, 0.1961, 0.1922,  ..., 0.1725, 0.1686, 0.1608],
         [0.2039, 0.1961, 0.1922,  ..., 0.1843, 0.1843, 0.1804]],

        [[0.3843, 0.3843, 0.3922,  ..., 0.2902, 0.2863, 0.2824],
         [0.3922, 0.4039, 0.4039,  ..., 0.2902, 0.2863, 0.2824],
         [0.4000, 0.4118, 0.4157,  ..., 0.2863, 0.2784, 0.2745],
         ...,
         [0.2588, 0.2588, 0.2588,  ..., 0.2078, 0.2078, 0.2078],
         [0.2588, 0.2588, 0.2588,  ..., 0.2196, 0.2157, 0.2078],
         [0.2588, 0.2588, 0.2588,  ..., 0.2314, 0.2314, 0.2275]],

        [[0.3255, 0.3255, 0.3137,  ..., 0.2353, 0.2314, 0.2235],
         [0.3412, 0.3608, 0.3490,  ..., 0.2314, 0.2275, 0.2196],
         [0.3569, 0.3725, 0.3765,  ..., 0.2275, 0.2235, 0.2157],
         ...,
         [0.2275, 0.2235, 0.2275,  ..., 0.1765, 0.1765, 0.1765],
         [0.2275, 0.2235, 0.2275,  ..., 0.1882, 0.1843, 0.1725],
         [0.2275, 0.2235, 0.2275,  ..., 0.2039, 0.2000, 0.1961]]],
       device='cuda:0')]

/usr/local/lib/python3.7/dist-packages/torch/functional.py:445: UserWa
rning: torch.meshgrid: in an upcoming release, it will be required to
pass the indexing argument. (Triggered internally at  ../aten/src/ATe
n/native/TensorShape.cpp:2157.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]

Prediction keys: dict_keys(['boxes', 'labels', 'scores', 'masks'])
Boxes shape: torch.Size([56, 4])
Labels shape: torch.Size([56])
Scores shape: torch.Size([56])
Masks shape: torch.Size([56, 1, 1024, 2048])
```

In [ ]:

```
import numpy as np
import cv2
import random

# Array of labels for COCO dataset (91 elements)

coco_names = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis rac
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'c
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]

# Random colors to use for labeling objects

COLORS = np.random.uniform(0, 255, size=(len(coco_names), 3)).astype(np.uint8)

# Overlay masks, bounding boxes, and labels on input numpy image

def draw_segmentation_map(image, masks, boxes, labels):
    alpha = 1
    beta = 0.5 # transparency for the segmentation map
    gamma = 0 # scalar added to each sum
    # convert from RGB to OpenCV BGR format
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    for i in range(len(masks)):
        mask = masks[i,:,:]
        red_map = np.zeros_like(mask).astype(np.uint8)
        green_map = np.zeros_like(mask).astype(np.uint8)
        blue_map = np.zeros_like(mask).astype(np.uint8)
        # apply a randon color mask to each object
        color = COLORS[random.randrange(0, len(COLORS))]
        red_map[mask > 0.5] = color[0]
        green_map[mask > 0.5] = color[1]
        blue_map[mask > 0.5] = color[2]
        # combine all the masks into a single image
        segmentation_map = np.stack([red_map, green_map, blue_map], axis=2)
        # apply colored mask to the image
        image = cv2.addWeighted(image, alpha, segmentation_map, beta, gamma)
        # draw the bounding box around each object
        p1 = (int(boxes[i][0]), int(boxes[i][1]))
        p2 = (int(boxes[i][2]), int(boxes[i][3]))
        color = (int(color[0]), int(color[1]), int(color[2]))
        cv2.rectangle(image, p1, p2, color, 2)
        # put the label text above the objects
        p = (int(boxes[i][0]), int(boxes[i][1]-10))
        cv2.putText(image, labels[i], p, cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2, cv

    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Overlay masks, bounding boxes, and labels of objects with scores greater than
# threshold on one of the images in the input tensor using the predictions output by

def prediction_to_mask_image(images, predictions, img_index, threshold):
    scores = predictions[img_index]['scores']
    boxes_to_use = scores >= threshold
    img = (images[img_index].cpu().permute(1, 2, 0).numpy() * 255).astype(np.uint8)
    masks = predictions[img_index]['masks'][boxes_to_use, :, :].cpu().detach().squee
    boxes = predictions[img_index]['boxes'][boxes_to_use, :].cpu().detach().numpy()
    labels = predictions[img_index]['labels'][boxes_to_use].cpu().numpy()
    labels = [ coco_names[l] for l in labels ]

    return draw_segmentation_map(img, masks, boxes, labels)
```

In [ ]:

```
from matplotlib import pyplot as plt

masked_img = prediction_to_mask_image(images, predictions, 0, 0.5)
plt.figure(1, figsize=(12, 9), dpi=100)
plt.imshow(masked_img)
plt.title('Validation image result')
plt.show()
```



In [ ]:

```

```

coco-finetune

In [ ]:

```
es_path = "data/"

oading Cityscapes val datasets...')
taset = CityscapesDataset(cityscapes_path, split='train', mode='fine', augment=False
set = CityscapesDataset(cityscapes_path, split='val', mode='fine', augment=False)

rain dataset')
ain_dataset)
al dataset')
l_dataset)

dataset = get_cityscapes()

llate_fn(batch):
turn tuple(zip(*batch))

taloader = torch.utils.data.DataLoader(train_dataset, batch_size=1, shuffle=True, nu
loader = torch.utils.data.DataLoader(val_dataset, batch_size=1, shuffle=True, num_wo
```

```
Loading Cityscapes val datasets...
data/leftImg8bit/train
data/gtFine/train
data/leftImg8bit/val
data/gtFine/val
train dataset
Dataset CityscapesDataset
    Number of images: 2975
    Split: train
    Mode: gtFine
    Augment: False
    Root Location: data/

val dataset
Dataset CityscapesDataset
    Number of images: 500
    Split: val
    Mode: gtFine
    Augment: False
    Root Location: data/
```

In [ ]:

```
import torch
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
from PIL import Image

num_classes = 91

print('loading pretrained model')
model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)

# Modify model for the given number of classes

in_features = model.roi_heads.box_predictor.cls_score.in_features
num_classes = 8

model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
hidden_layer = 256
model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask,
                                                   hidden_layer,
                                                   num_classes)

model.cuda()
```

```
loading pretrained model
```

In [ ]:

```
import numpy as np
import cv2
import random
import matplotlib.pyplot as plt

# Array of labels for COCO dataset (91 elements)

coco_names = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis rac
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'c
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]

# Random colors to use for labeling objects

COLORS = np.random.uniform(0, 255, size=(len(coco_names), 3)).astype(np.uint8)

# Overlay masks, bounding boxes, and labels on input numpy image

def draw_segmentation_map(image, masks, boxes, labels):
    alpha = 1
    beta = 0.5 # transparency for the segmentation map
    gamma = 0 # scalar added to each sum
    # convert from RGB to OpenCV BGR format
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    for i in range(len(masks)):
        mask = masks[i,:,:]
        red_map = np.zeros_like(mask).astype(np.uint8)
        green_map = np.zeros_like(mask).astype(np.uint8)
        blue_map = np.zeros_like(mask).astype(np.uint8)
        # apply a randon color mask to each object
        color = COLORS[random.randrange(0, len(COLORS))]
        red_map[mask > 0.5] = color[0]
        green_map[mask > 0.5] = color[1]
        blue_map[mask > 0.5] = color[2]
        # combine all the masks into a single image
        segmentation_map = np.stack([red_map, green_map, blue_map], axis=2)
        # apply colored mask to the image
        image = cv2.addWeighted(image, alpha, segmentation_map, beta, gamma)
        # draw the bounding box around each object
        p1 = (int(boxes[i][0]), int(boxes[i][1]))
        p2 = (int(boxes[i][2]), int(boxes[i][3]))
        color = (int(color[0]), int(color[1]), int(color[2]))
        cv2.rectangle(image, p1, p2, color, 2)
        # put the label text above the objects
        p = (int(boxes[i][0]), int(boxes[i][1]-10))
        cv2.putText(image, labels[i], p, cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2, cv

    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
# Overlay masks, bounding boxes, and labels of objects with scores greater than
# threshold on one of the images in the input tensor using the predictions output by

def prediction_to_mask_image(images, predictions, img_index, threshold):
    scores = predictions[img_index]['scores']
    boxes_to_use = scores >= threshold
    img = (images[img_index].cpu().permute(1, 2, 0).numpy() * 255).astype(np.uint8)
    masks = predictions[img_index]['masks'][boxes_to_use, :, :].cpu().detach().squee
    boxes = predictions[img_index]['boxes'][boxes_to_use, :].cpu().detach().numpy()
    labels = predictions[img_index]['labels'][boxes_to_use].cpu().numpy()
    labels = [ coco_names[l] for l in labels ]

    return draw_segmentation_map(img, masks, boxes, labels)
```

Inference

In [ ]:

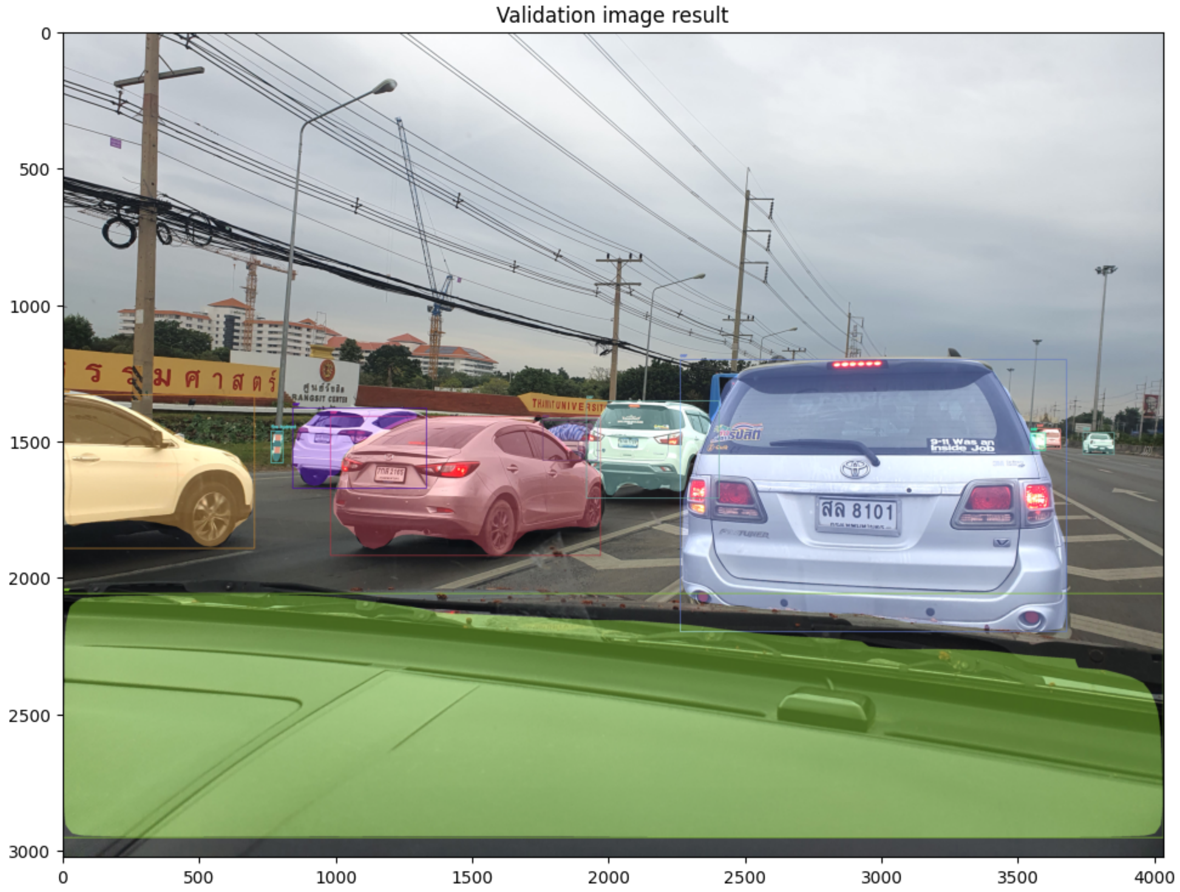
```
!wget https://www.cs.ait.ac.th/~mdailey/20201112_072342.jpg
```

```
--2022-03-03 14:24:40--  https://www.cs.ait.ac.th/~mdailey/20201112_07
2342.jpg (https://www.cs.ait.ac.th/~mdailey/20201112_072342.jpg)
Resolving www.cs.ait.ac.th (www.cs.ait.ac.th)... 192.41.170.42
Connecting to www.cs.ait.ac.th (www.cs.ait.ac.th)|192.41.170.42|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3320786 (3.2M) [image/jpeg]
Saving to: ‘20201112_072342.jpg.1’

20201112_072342.jpg 100%[===================>]   3.17M   867KB/s    in
3.7s

2022-03-03 14:24:45 (867 KB/s) - ‘20201112_072342.jpg.1’ saved [332078
6/3320786]
```

In [ ]:

```
#model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained = True)
model_test = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)
model_test.eval()

im = Image.open('20201112_072342.jpg').rotate(180)

print(type(images))
#images = [ img.cuda() for img in images ]
transform = T.ToTensor()
image_batch = transform(im).unsqueeze(0)
# #image_batch.shap
print(image_batch)
predictions = model_test(image_batch)


print('Prediction keys:', predictions[0].keys())
print('Boxes shape:', predictions[0]['boxes'].shape)
print('Labels shape:', predictions[0]['labels'].shape)
print('Scores shape:', predictions[0]['scores'].shape)
print('Masks shape:', predictions[0]['masks'].shape)

import numpy as np
import cv2
import random

# Array of labels for COCO dataset (91 elements)

coco_names = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A', 'N
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis rac
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'c
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]

# Random colors to use for labeling objects

COLORS = np.random.uniform(0, 255, size=(len(coco_names), 3)).astype(np.uint8)

# Overlay masks, bounding boxes, and labels on input numpy image

def draw_segmentation_map(image, masks, boxes, labels):
    alpha = 1
    beta = 0.5 # transparency for the segmentation map
    gamma = 0 # scalar added to each sum
    # convert from RGB to OpenCV BGR format
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    for i in range(len(masks)):
        mask = masks[i,:,:]
        red_map = np.zeros_like(mask).astype(np.uint8)
        green_map = np.zeros_like(mask).astype(np.uint8)
        blue_map = np.zeros_like(mask).astype(np.uint8)
        # apply a randon color mask to each object
        color = COLORS[random.randrange(0, len(COLORS))]
        red_map[mask > 0.5] = color[0]
        green_map[mask > 0.5] = color[1]
        blue_map[mask > 0.5] = color[2]
        # combine all the masks into a single image
        segmentation_map = np.stack([red_map, green_map, blue_map], axis=2)
        # apply colored mask to the image
        image = cv2.addWeighted(image, alpha, segmentation_map, beta, gamma)
        # draw the bounding box around each object
        p1 = (int(boxes[i][0]), int(boxes[i][1]))
        p2 = (int(boxes[i][2]), int(boxes[i][3]))
        color = (int(color[0]), int(color[1]), int(color[2]))
        cv2.rectangle(image, p1, p2, color, 2)
        # put the label text above the objects
        p = (int(boxes[i][0]), int(boxes[i][1]-10))
        cv2.putText(image, labels[i], p, cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2, cv

    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Overlay masks, bounding boxes, and labels of objects with scores greater than
# threshold on one of the images in the input tensor using the predictions output by

def prediction_to_mask_image(images, predictions, img_index, threshold):
    scores = predictions[img_index]['scores']
    boxes_to_use = scores >= threshold
    img = (images[img_index].cpu().permute(1, 2, 0).numpy() * 255).astype(np.uint8)
    masks = predictions[img_index]['masks'][boxes_to_use, :, :].cpu().detach().squee
    boxes = predictions[img_index]['boxes'][boxes_to_use, :].cpu().detach().numpy()
    labels = predictions[img_index]['labels'][boxes_to_use].cpu().numpy()
    labels = [ coco_names[l] for l in labels ]

    return draw_segmentation_map(img, masks, boxes, labels)


from matplotlib import pyplot as plt

masked_img = prediction_to_mask_image(image_batch, predictions, 0, 0.5)
plt.figure(1, figsize=(12, 9), dpi=100)
plt.imshow(masked_img)
plt.title('Validation image result')
plt.show()
```

```
<class 'list'>
tensor([[[[0.8314, 0.8314, 0.8353,  ..., 0.8275, 0.8275, 0.8314],
          [0.8275, 0.8314, 0.8314,  ..., 0.8196, 0.8196, 0.8235],
          [0.8314, 0.8353, 0.8353,  ..., 0.8314, 0.8235, 0.8235],
          ...,
          [0.3216, 0.3137, 0.2902,  ..., 0.1961, 0.1843, 0.1451],
          [0.2902, 0.3098, 0.2745,  ..., 0.1255, 0.1098, 0.1255],
          [0.2667, 0.2784, 0.2784,  ..., 0.1176, 0.1176, 0.2314]],

         [[0.8510, 0.8510, 0.8549,  ..., 0.8471, 0.8471, 0.8510],
          [0.8471, 0.8510, 0.8510,  ..., 0.8392, 0.8392, 0.8431],
          [0.8510, 0.8549, 0.8549,  ..., 0.8510, 0.8431, 0.8431],
          ...,
          [0.3608, 0.3529, 0.3294,  ..., 0.2118, 0.2000, 0.1608],
          [0.3294, 0.3490, 0.3137,  ..., 0.1412, 0.1255, 0.1412],
          [0.3059, 0.3176, 0.3176,  ..., 0.1333, 0.1333, 0.2471]],

         [[0.8745, 0.8745, 0.8784,  ..., 0.8706, 0.8706, 0.8745],
          [0.8706, 0.8745, 0.8745,  ..., 0.8627, 0.8627, 0.8667],
          [0.8745, 0.8784, 0.8784,  ..., 0.8745, 0.8667, 0.8667],
          ...,
          [0.3255, 0.3176, 0.2941,  ..., 0.2235, 0.2118, 0.1725],
          [0.2941, 0.3137, 0.2784,  ..., 0.1529, 0.1373, 0.1529],
          [0.2706, 0.2824, 0.2824,  ..., 0.1451, 0.1451, 0.2588]]]])
Prediction keys: dict_keys(['boxes', 'labels', 'scores', 'masks'])
Boxes shape: torch.Size([34, 4])
Labels shape: torch.Size([34])
Scores shape: torch.Size([34])
Masks shape: torch.Size([34, 1, 3024, 4032])
```



Validation image result

In [ ]: