pytorch-project

July 13, 2024

[1]:
```
!pip install pycocotools
```

```
Collecting pycocotools
  Downloading pycocotools-2.0.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl.metadata (1.1 kB)
Requirement already satisfied: matplotlib>=2.1.0 in
/opt/conda/lib/python3.10/site-packages (from pycocotools) (3.7.5)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages
(from pycocotools) (1.26.4)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.1.0->pycocotools)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib>=2.1.0->pycocotools) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.1.0->pycocotools)
(4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.1.0->pycocotools)
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.1.0->pycocotools)
(21.3)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-
packages (from matplotlib>=2.1.0->pycocotools) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.1.0->pycocotools)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=2.1.0->pycocotools)
(2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-
packages (from python-dateutil>=2.7->matplotlib>=2.1.0->pycocotools) (1.16.0)
Downloading
pycocotools-2.0.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(426 kB)
                         426.2/426.2 kB
10.4 MB/s eta 0:00:0000:01
```

```
Installing collected packages: pycocotools
Successfully installed pycocotools-2.0.7
```

[2]:
```
import PIL.Image
import random
import torch
import torch.utils.data
import numpy as np
from tqdm import tqdm
from collections import defaultdict
import torchvision.datasets as dset
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
import torchvision
torchvision.disable_beta_transforms_warning()
from torchvision import models
import torchvision.transforms as original_transforms
import torchvision.transforms.v2 as transforms
from torchvision.transforms.v2 import functional as F
from torchvision.utils import draw_bounding_boxes
import multiprocessing as mp
from torch import nn
import torch.optim as optim
from tqdm import tqdm
```

[3]:
```
n_gpus = torch.cuda.device_count()
USING_CPU = not torch.cuda.is_available()

DEVICE = torch.device("cuda:0" if (torch.cuda.is_available()  and n_gpus > 0)␣
 ↪else "cpu")
kwargs = {'num_workers': mp.cpu_count() , 'pin_memory': True} if DEVICE.
 ↪type=='cuda' else {'num_workers': mp.cpu_count()//2, 'prefetch_factor': 4}


print(f'Num of CPUs: {mp.cpu_count()}')
print(f'Device in use: {DEVICE}')
print(f'Found {n_gpus} GPU Device/s.')
```

```
Num of CPUs: 4
Device in use: cpu
Found 0 GPU Device/s.
```

[26]:
```
Train_Data = '/kaggle/input/coco-2017-dataset/coco2017/train2017'
Annotation = '/kaggle/input/coco-2017-dataset/coco2017/annotations/
 ↪instances_train2017.json'
USE_PRETRAINED = True
SAVED_MODEL_PATH = '/kaggle/working/checkpoint'
```

```
def load_dataset(transform):
    return dset.CocoDetection(root = Train_Data,
                              annFile = Annotation)
```

[5]:
```
class RandomHorizontalFlip(object):
    def __init__(self, p=0.5):
        self.p = p
        self.hf = transforms.RandomHorizontalFlip(1)

    def __call__(self, img, bboxes):

        if torch.rand(1)[0] < self.p:
            img = self.hf.forward(img)
            bboxes = self.hf.forward(bboxes)

        return img, bboxes


class RandomVerticalFlip(object):
    def __init__(self, p=0.5):
        self.p = p
        self.vf = transforms.RandomVerticalFlip(1)

    def __call__(self, img, bboxes):
        if torch.rand(1)[0] < self.p:
            img = self.vf.forward(img)
            bboxes = self.vf.forward(bboxes)

        return img, bboxes

class Resize(object):
    def __init__(self, size):
        self.size = size
        self.resize = transforms.Resize(self.size, antialias=True)

    def __call__(self, img, bboxes):
        img = self.resize.forward(img)
        bboxes = self.resize.forward(bboxes)
        return img, bboxes
```

[6]:
```
def show(sample):
    import matplotlib.pyplot as plt
    from torchvision.transforms.v2 import functional as F
    from torchvision.utils import draw_bounding_boxes

    resize = Resize((300, 300))
```

```
    rhf = RandomHorizontalFlip()
    rvf = RandomVerticalFlip()
    image, target = sample

    image, bboxes = image,target["boxes"]

    image, bboxes = resize(image, bboxes)
    image, bboxes = rhf(image, bboxes)
    image, bboxes = rvf(image, bboxes)

    if isinstance(image, PIL.Image.Image):
        image = F.to_tensor(image)

    image = F.convert_dtype(image, torch.uint8)
    annotated_image = draw_bounding_boxes(image, bboxes, colors="yellow",␣
 ↪width=3)

    fig, ax = plt.subplots()
    ax.imshow(annotated_image.permute(1, 2, 0).numpy())
    ax.set(xticklabels=[], yticklabels=[], xticks=[], yticks=[])
    fig.tight_layout()

    fig.show()
```

[7]:
```
transform = transforms.Compose(
    [
        transforms.RandomPhotometricDistort(),
        transforms.RandomAutocontrast(),
        transforms.RandomEqualize(),
        transforms.GaussianBlur(kernel_size=3),
        transforms.ToTensor(),
        transforms.ConvertImageDtype(torch.float32),
    ]
)
```

```
/opt/conda/lib/python3.10/site-
packages/torchvision/transforms/v2/_deprecated.py:43: UserWarning: The transform
`ToTensor()` is deprecated and will be removed in a future release. Instead,
please use `v2.Compose([v2.ToImage(), v2.ToDtype(torch.float32, scale=True)])`.
  warnings.warn(
```

[8]:
```
coco_train = load_dataset(transform=transform)
coco_train = dset.wrap_dataset_for_transforms_v2(coco_train)
```

```
loading annotations into memory…
Done (t=36.78s)
```

```
creating index…
index created!
```

[9]:
```python
class NewCocoDataset(Dataset):
    def __init__(self, coco_dataset, image_size=(312, 312)):
        """
        Arguments:
            coco_dataset (dataset): The coco dataset containing all the␣
 ↪expected transforms.
            image_size (tuple): Target image size. Default is (512, 512)
        """

        self.coco_dataset = coco_dataset
        self.resize = Resize(image_size)
        self.rhf = RandomHorizontalFlip()
        self.rvf = RandomVerticalFlip()
        self.transformer = transforms.Compose([
            transforms.ToTensor(),
            transforms.ConvertImageDtype(torch.float32),
        ])


    def __len__(self):
        return len(self.coco_dataset)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        new_target = {}

        image, target = self.coco_dataset[idx]

        if 'boxes' not in target:
            new_idx = idx-1
            _img, _t = self.coco_dataset[new_idx]
            while 'boxes' not in _t :
                new_idx -= 1
                _img, _t = self.coco_dataset[new_idx]

            image, target = self.coco_dataset[new_idx]


        image, bboxes = image, target["boxes"]

        image, bboxes = self.resize(image, bboxes)
        image, bboxes = self.rhf(image, bboxes)
```

```
        image, bboxes = self.rvf(image, bboxes)

        image = self.transformer(image)

        new_boxes = []
        for box in bboxes:
            if box[0] < box[2] and box[1] < box[3]:
                new_boxes.append(box)

        new_target["boxes"] = torch.stack(new_boxes)
        new_target["labels"] = target["labels"]

        return (image, new_target)
```

[10]:
```
class CustomBatchs:
    def __init__(self, data):
        transposed_data = list(zip(*data))
        self.inp = torch.stack(transposed_data[0], 0)
        self.tgt = transposed_data[1]

    # custom memory pinning method on custom type
    def pin_memory(self):
        self.inp = self.inp.pin_memory()
        return (self.inp, self.tgt)

def collate_wrapper(batch):
    if torch.cuda.is_available():
        return CustomBatchs(batch)
    else:
        return tuple(zip(*batch))
```

[11]:
```
new_coco_train = NewCocoDataset(coco_train)

data_loader = torch.utils.data.DataLoader(
    new_coco_train,
    batch_size=50 if not USING_CPU else 8,
    shuffle=True,
#     collate_fn=lambda batch: tuple(zip(*batch)),
    collate_fn=collate_wrapper,
     **kwargs
)
```

[12]:
```
import pycocotools.coco

coco_anns = pycocotools.coco.COCO(Annotation)
catIDs = coco_anns.getCatIds()
cats = coco_anns.loadCats(catIDs)
```

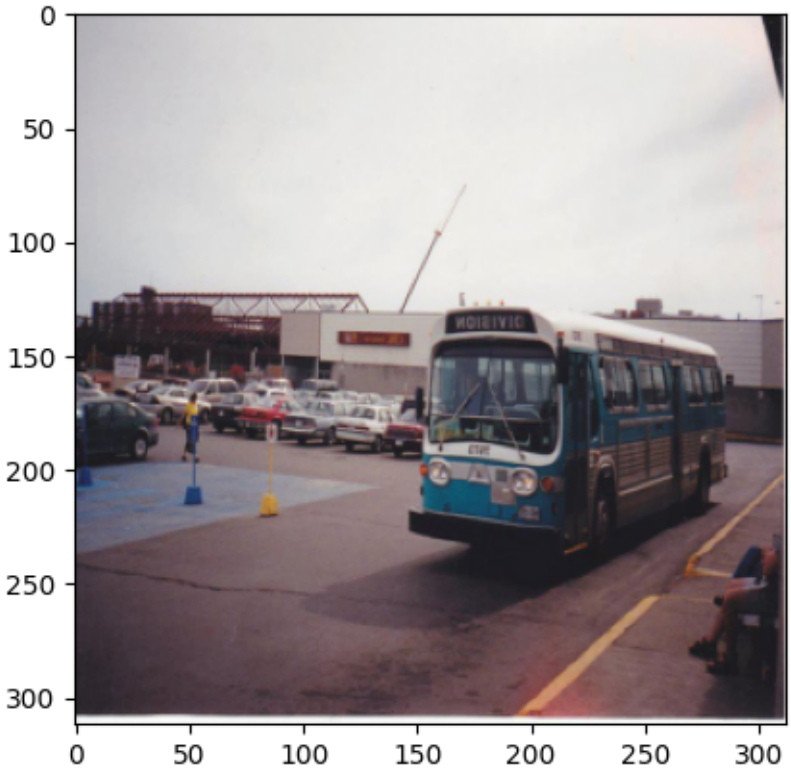
```
name_idx = {}

for sub_dict in cats:
    name_idx[sub_dict["id"]] = sub_dict["name"]

del coco_anns, catIDs, cats
```

```
loading annotations into memory…
Done (t=39.94s)
creating index…
index created!
```

[13]:
```
data = next(iter(data_loader))
if USING_CPU:
    x = torch.stack(data[0])
else:
    x = data[0]
print(x.shape)
# _labels = [name_idx[i] for i in data[1][0]['labels'].tolist()]
# print(_labels)

plt.imshow(data[0][0].permute(1, 2, 0).numpy())
```

```
/opt/conda/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX
is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
/opt/conda/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX
is multithreaded, so this will likely lead to a deadlock.
  self.pid = os.fork()
torch.Size([8, 3, 312, 312])
```

[13]:
```
<matplotlib.image.AxesImage at 0x7e4183040b20>
```



[21]:
```
base_model = models.get_model("ssd300_vgg16", weights=None,␣
 ↪weights_backbone=None).train()
```

[22]:
```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
```

[23]:
```
base_model.apply(weights_init)

if (DEVICE.type == 'cuda') and (n_gpus > 1):
    base_model = nn.DataParallel(base_model, list(range(n_gpus)))
```

[24]:
```
base_model.to(DEVICE)
```

[24]:
```
SSD(
  (backbone): SSDFeatureExtractorVGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU(inplace=True)
```

```
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (6): ReLU(inplace=True)
      (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): ReLU(inplace=True)
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (11): ReLU(inplace=True)
      (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (13): ReLU(inplace=True)
      (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (15): ReLU(inplace=True)
      (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=True)
      (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (18): ReLU(inplace=True)
      (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (20): ReLU(inplace=True)
      (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (22): ReLU(inplace=True)
    )
    (extra): ModuleList(
      (0): Sequential(
        (0): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
        (1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (2): ReLU(inplace=True)
        (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4): ReLU(inplace=True)
        (5): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (6): ReLU(inplace=True)
        (7): Sequential(
          (0): MaxPool2d(kernel_size=3, stride=1, padding=1, dilation=1,
ceil_mode=False)
          (1): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(6,
6), dilation=(6, 6))
          (2): ReLU(inplace=True)
          (3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1))
          (4): ReLU(inplace=True)
        )
      )
      (1): Sequential(
        (0): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
```

```
        (3): ReLU(inplace=True)
      )
      (2): Sequential(
        (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        (3): ReLU(inplace=True)
      )
      (3-4): 2 x Sequential(
        (0): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
        (1): ReLU(inplace=True)
        (2): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
        (3): ReLU(inplace=True)
      )
    )
  )
  (anchor_generator): DefaultBoxGenerator(aspect_ratios=[[2], [2, 3], [2, 3],
[2, 3], [2], [2]], clip=True, scales=[0.07, 0.15, 0.33, 0.51, 0.69, 0.87, 1.05],
steps=[8, 16, 32, 64, 100, 300])
  (head): SSDHead(
    (classification_head): SSDClassificationHead(
      (module_list): ModuleList(
        (0): Conv2d(512, 364, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): Conv2d(1024, 546, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        (2): Conv2d(512, 546, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): Conv2d(256, 546, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4-5): 2 x Conv2d(256, 364, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      )
    )
    (regression_head): SSDRegressionHead(
      (module_list): ModuleList(
        (0): Conv2d(512, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): Conv2d(1024, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (2): Conv2d(512, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): Conv2d(256, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (4-5): 2 x Conv2d(256, 16, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      )
    )
  )
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.48235, 0.45882, 0.40784], std=[0.00392156862745098,
0.00392156862745098, 0.00392156862745098])
      Resize(min_size=(300,), max_size=300, mode='bilinear')
  )
```

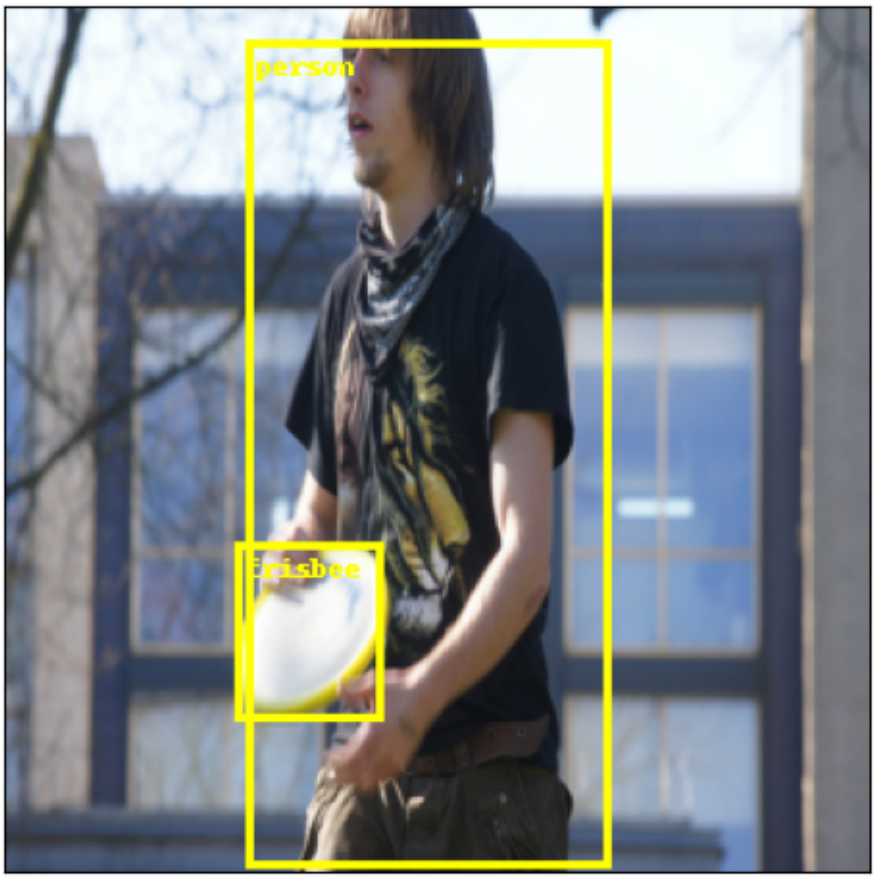
```
)
```

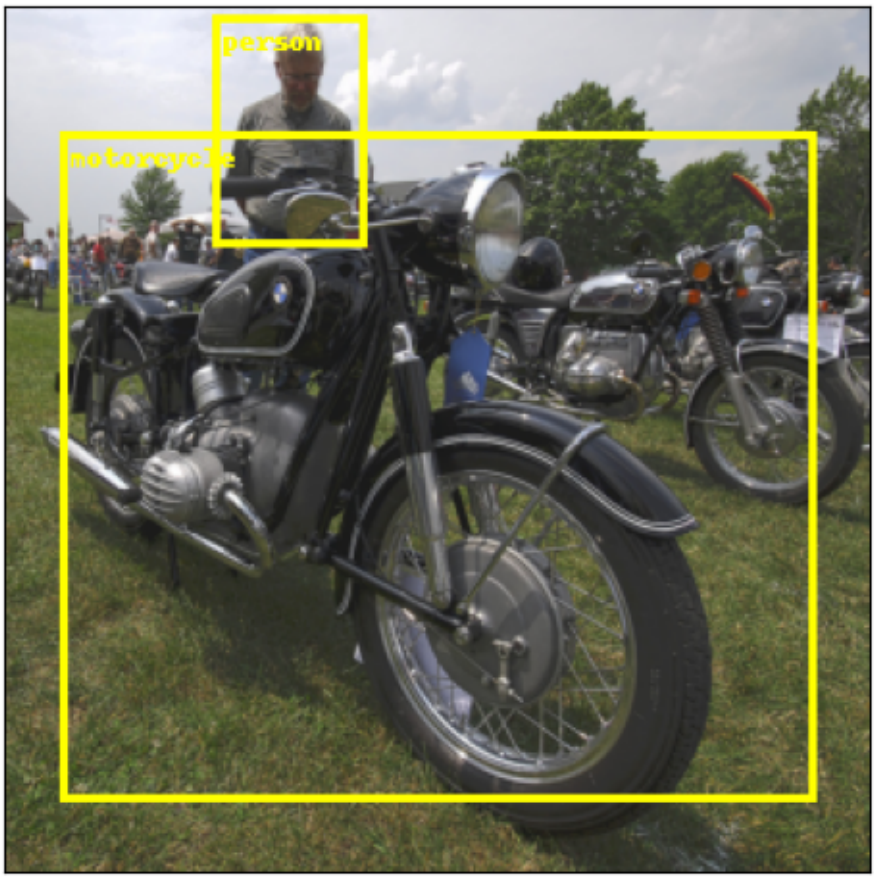
[18]:
```
learning_rate = 1e-4

optimizer = optim.Adam(base_model.parameters(), lr=learning_rate)
```

[27]:
```
if USE_PRETRAINED:
    new_LR = 1e-5 # change this value to set a new Learning Rate for the␣
 ↪version of notebook

    if USING_CPU:
        checkpoint = torch.load(SAVED_MODEL_PATH, map_location=torch.
 ↪device('cpu'))
    else:
        checkpoint = torch.load(SAVED_MODEL_PATH)

    base_model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    for g in optimizer.param_groups:
        g['lr'] = new_LR
```

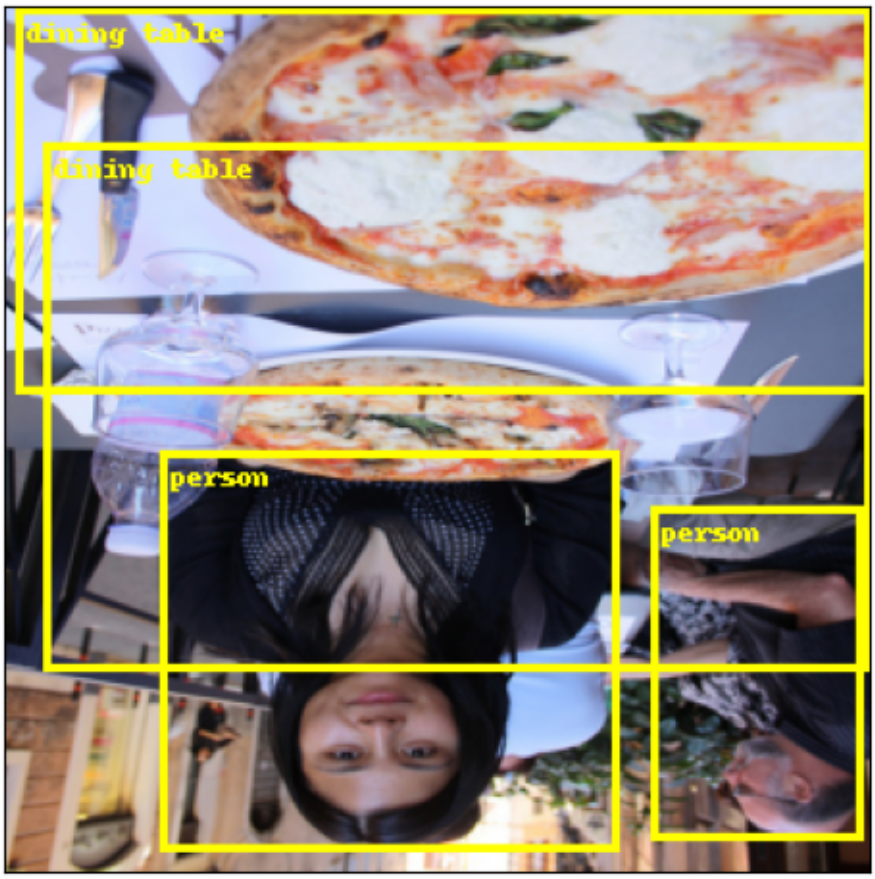
[31]:
```
img_dtype_converter = transforms.ConvertImageDtype(torch.uint8)
data = next(iter(val_data_loader))

_i = data[0]

threshold = 0.3

if USING_CPU:
    _i = torch.stack(_i)

_i = _i.to(DEVICE)
base_model.eval()
p_t = base_model(_i)

indices = range(len(p_t))  # Choose all indices

for idx in indices:
    confidence_length = len(np.argwhere(p_t[idx]['scores'] > threshold)[0])

    p_boxes = p_t[idx]['boxes'][:confidence_length]
    p_labels = [name_idx[i] for i in p_t[idx]['labels'][:confidence_length].
 ↪tolist()]
    i_img = img_dtype_converter(_i[idx])

    annotated_image = draw_bounding_boxes(i_img, p_boxes, p_labels,␣
 ↪colors="yellow", width=3)
```

```
    fig, ax = plt.subplots()
    ax.imshow(annotated_image.permute(1, 2, 0).numpy())
    ax.set(xticklabels=[], yticklabels=[], xticks=[], yticks=[])
    fig.tight_layout()

    fig.show()
```


person
frisbee


person
motorcycle


person
person


dining table
dining table
person
person


laptop
tv


person