```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
```

```
#!unzip /content/aqua.zip
```

```
!pip install -U torchvision # We need a new versino of torchvision for this project
```

```
    Attempting uninstall: nvidia-cublas-cu12
      Found existing installation: nvidia-cublas-cu12 12.6.3.3
      Uninstalling nvidia-cublas-cu12-12.6.3.3:
        Successfully uninstalled nvidia-cublas-cu12-12.6.3.3
    Attempting uninstall: nvidia-cusparse-cu12
      Found existing installation: nvidia-cusparse-cu12 12.5.4.2
      Uninstalling nvidia-cusparse-cu12-12.5.4.2:
        Successfully uninstalled nvidia-cusparse-cu12-12.5.4.2
    Attempting uninstall: nvidia-cudnn-cu12
      Found existing installation: nvidia-cudnn-cu12 9.5.0.50
      Uninstalling nvidia-cudnn-cu12-9.5.0.50:
        Successfully uninstalled nvidia-cudnn-cu12-9.5.0.50
    Attempting uninstall: nvidia-cusolver-cu12
      Found existing installation: nvidia-cusolver-cu12 11.7.1.2
      Uninstalling nvidia-cusolver-cu12-11.7.1.2:
        Successfully uninstalled nvidia-cusolver-cu12-11.7.1.2
    Attempting uninstall: torch
      Found existing installation: torch 2.5.0+cu121
      Uninstalling torch-2.5.0+cu121:
        Successfully uninstalled torch-2.5.0+cu121
    Attempting uninstall: torchvision
```

```
import torch
import torchvision
from torchvision import datasets, models
from torchvision.transforms import functional as FT
from torchvision import transforms as T
from torch import nn, optim
from torch.nn import functional as F
from torch.utils.data import DataLoader, sampler, random_split, Dataset
import copy
import math
from PIL import Image
import cv2
import albumentations as A  # our data augmentation library

import matplotlib.pyplot as plt
%matplotlib inline
```

```
/usr/local/lib/python3.10/dist-packages/albumentations/__init__.py:13: UserWarning: A new version of Albumentations is available: 1.4.
  check_for_updates()
```

```
# remove arnings (optional)
import warnings
warnings.filterwarnings("ignore")
from collections import defaultdict, deque
import datetime
import time
from tqdm import tqdm # progress bar
from torchvision.utils import draw_bounding_boxes


print(torch.__version__)
print(torchvision.__version__)
```

```
2.5.1+cu124
0.20.1+cu124
```

```
# our dataset is in cocoformat, we will need pypcoco tools
!pip install pycocotools
from pycocotools.coco import COCO
```

```
Requirement already satisfied: pycocotools in /usr/local/lib/python3.10/dist-packages (2.0.8)
Requirement already satisfied: matplotlib>=2.1.0 in /usr/local/lib/python3.10/dist-packages (from pycocotools) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pycocotools) (1.26.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (1.3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (4.5
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (1.4
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (3.2.
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=2.1.0->pycocotools) (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=2.1.0->pyco
```

```
# Now, we will define our transforms
from albumentations.pytorch import ToTensorV2


def get_transforms(train=False):
    if train:
        transform = A.Compose([
            A.Resize(600, 600), # our input size can be 600px
            A.HorizontalFlip(p=0.3),
            A.VerticalFlip(p=0.3),
            A.RandomBrightnessContrast(p=0.1),
            A.ColorJitter(p=0.1),
            ToTensorV2()
        ], bbox_params=A.BboxParams(format='coco'))
    else:
        transform = A.Compose([
            A.Resize(600, 600), # our input size can be 600px
            ToTensorV2()
        ], bbox_params=A.BboxParams(format='coco'))
    return transform
```

Start coding or generate with AI.

```
class AquariumDetection(datasets.VisionDataset):
    def __init__(self, root, split='train', transform=None, target_transform=None, transforms=None):
        # the 3 transform parameters are reuqired for datasets.VisionDataset
        super().__init__(root, transforms, transform, target_transform)
        self.split = split #train, valid, test
        self.coco = COCO(os.path.join(root, split, "_annotations.coco.json")) # annotatiosn stored here
        self.ids = list(sorted(self.coco.imgs.keys()))
        self.ids = [id for id in self.ids if (len(self._load_target(id)) > 0)]

    def _load_image(self, id: int):
        path = self.coco.loadImgs(id)[0]['file_name']
        image = cv2.imread(os.path.join(self.root, self.split, path))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        return image
    def _load_target(self, id):
        return self.coco.loadAnns(self.coco.getAnnIds(id))

    def __getitem__(self, index):
        id = self.ids[index]
        image = self._load_image(id)
        target = self._load_target(id)
        target = copy.deepcopy(self._load_target(id))

        boxes = [t['bbox'] + [t['category_id']] for t in target] # required annotation format for albumentations
        if self.transforms is not None:
            transformed = self.transforms(image=image, bboxes=boxes)

        image = transformed['image']
        boxes = transformed['bboxes']
        new_boxes = [] # convert from xywh to xyxy
        for box in boxes:
            xmin = box[0]
            xmax = xmin + box[2]
            ymin = box[1]
            ymax = ymin + box[3]
            new_boxes.append([xmin, ymin, xmax, ymax])

        boxes = torch.tensor(new_boxes, dtype=torch.float32)

        targ = {} # here is our transformed target
        targ['boxes'] = boxes
        targ['labels'] = torch.tensor([t['category_id'] for t in target], dtype=torch.int64)
        targ['image_id'] = torch.tensor([t['image_id'] for t in target])
        targ['area'] = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0]) # we have a different area
        targ['iscrowd'] = torch.tensor([t['iscrowd'] for t in target], dtype=torch.int64)
        return image.div(255), targ # scale images
    def __len__(self):
        return len(self.ids)
```

Start coding or generate with AI.

```
dataset_path = "/content/Aquarium Combined/"
```

```
#load classes
coco = COCO(os.path.join(dataset_path, "train", "_annotations.coco.json"))
categories = coco.cats
n_classes = len(categories.keys())
categories
```

```
loading annotations into memory...
Done (t=0.03s)
creating index...
index created!
{0: {'id': 0, 'name': 'creatures', 'supercategory': 'none'},
 1: {'id': 1, 'name': 'fish', 'supercategory': 'creatures'},
 2: {'id': 2, 'name': 'jellyfish', 'supercategory': 'creatures'},
 3: {'id': 3, 'name': 'penguin', 'supercategory': 'creatures'},
 4: {'id': 4, 'name': 'puffin', 'supercategory': 'creatures'},
 5: {'id': 5, 'name': 'shark', 'supercategory': 'creatures'},
 6: {'id': 6, 'name': 'starfish', 'supercategory': 'creatures'},
 7: {'id': 7, 'name': 'stingray', 'supercategory': 'creatures'}}
```

```
classes = [i[1]['name'] for i in categories.items()]
classes
```

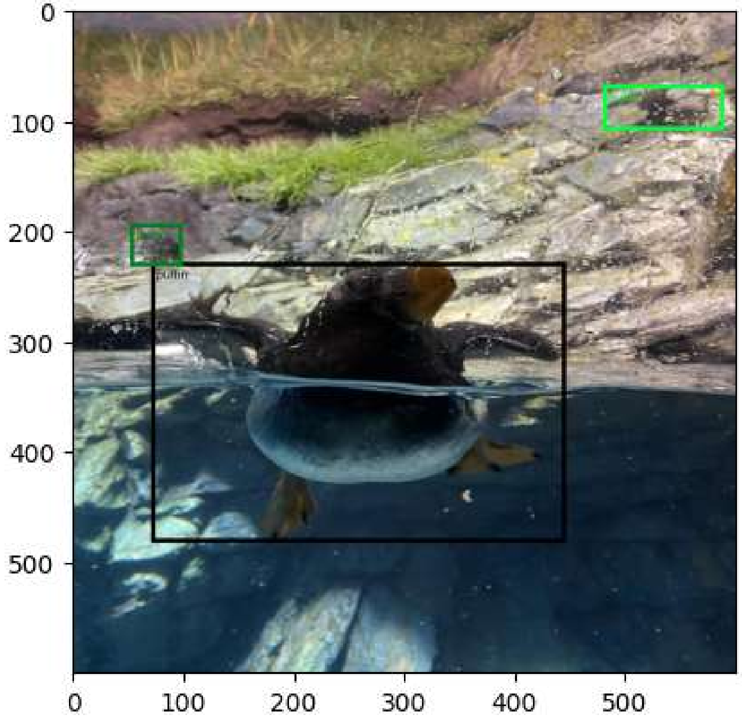
```
['creatures',
 'fish',
 'jellyfish',
 'penguin',
 'puffin',
 'shark',
 'starfish',
 'stingray']
```

```
train_dataset = AquariumDetection(root=dataset_path, transforms=get_transforms(True))
```

```
loading annotations into memory...
Done (t=0.03s)
creating index...
index created!
```

```
# Lets view a sample
sample = train_dataset[2]
img_int = torch.tensor(sample[0] * 255, dtype=torch.uint8)
plt.imshow(draw_bounding_boxes(img_int, sample[1]['boxes'], [classes[i] for i in sample[1]['labels']], width=4).permute(1, 2, 0))
```

```
<matplotlib.image.AxesImage at 0x7fd1c2230b50>
```



```
len(train_dataset)
```

```
447
```

```
# lets load the faster rcnn model
model = models.detection.fasterrcnn_mobilenet_v3_large_fpn(pretrained=True)
in_features = model.roi_heads.box_predictor.cls_score.in_features # we need to change the head
model.roi_heads.box_predictor = models.detection.faster_rcnn.FastRCNNPredictor(in_features, n_classes)
```

```
Downloading: "https://download.pytorch.org/models/fasterrcnn_mobilenet_v3_large_fpn-fb6a3cc7.pth" to /root/.cache/torch/hub/checkpoint
100%|██████████| 74.2M/74.2M [00:00<00:00, 123MB/s]
```

```
def collate_fn(batch):
    return tuple(zip(*batch))
```

```
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True, num_workers=4, collate_fn=collate_fn)


images,targets = next(iter(train_loader))
images = list(image for image in images)
targets = [{k:v for k, v in t.items()} for t in targets]
output = model(images, targets) # just make sure this runs without error


device = torch.device("cuda") # use GPU to train


model = model.to(device)


# Now, and optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.01, momentum=0.9, nesterov=True, weight_decay=1e-4)
# lr_scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones=[16, 22], gamma=0.1) # lr scheduler


import sys


def train_one_epoch(model, optimizer, loader, device, epoch):
    model.to(device)
    model.train()

#     lr_scheduler = None
#     if epoch == 0:
#         warmup_factor = 1.0 / 1000 # do lr warmup
#         warmup_iters = min(1000, len(loader) - 1)

#         lr_scheduler = optim.lr_scheduler.LinearLR(optimizer, start_factor = warmup_factor, total_iters=warmup_iters)

    all_losses = []
    all_losses_dict = []

    for images, targets in tqdm(loader):
        images = list(image.to(device) for image in images)
        targets = [{k: torch.tensor(v).to(device) for k, v in t.items()} for t in targets]

        loss_dict = model(images, targets) # the model computes the loss automatically if we pass in targets
        losses = sum(loss for loss in loss_dict.values())
        loss_dict_append = {k: v.item() for k, v in loss_dict.items()}
```

```
            loss_value = losses.item()

            all_losses.append(loss_value)
            all_losses_dict.append(loss_dict_append)

            if not math.isfinite(loss_value):
                print(f"Loss is {loss_value}, stopping trainig") # train if loss becomes infinity
                print(loss_dict)
                sys.exit(1)

            optimizer.zero_grad()
            losses.backward()
            optimizer.step()

#          if lr_scheduler is not None:
#              lr_scheduler.step() #

        all_losses_dict = pd.DataFrame(all_losses_dict) # for printing
        print("Epoch {}, lr: {:.6f}, loss: {:.6f}, loss_classifier: {:.6f}, loss_box: {:.6f}, loss_rpn_box: {:.6f}, loss_object: {:.6f}".forma
            epoch, optimizer.param_groups[0]['lr'], np.mean(all_losses),
            all_losses_dict['loss_classifier'].mean(),
            all_losses_dict['loss_box_reg'].mean(),
            all_losses_dict['loss_rpn_box_reg'].mean(),
            all_losses_dict['loss_objectness'].mean()
        ))


num_epochs=10

for epoch in range(num_epochs):
    train_one_epoch(model, optimizer, train_loader, device, epoch)
#     lr_scheduler.step()
```

```
100%|██████████| 112/112 [00:28<00:00,  3.94it/s]
Epoch 0, lr: 0.010000, loss: 0.971449, loss_classifier: 0.459347, loss_box: 0.385318, loss_rpn_box: 0.032460, loss_object: 0.094324
100%|██████████| 112/112 [00:20<00:00,  5.56it/s]
Epoch 1, lr: 0.010000, loss: 0.785062, loss_classifier: 0.356668, loss_box: 0.340599, loss_rpn_box: 0.028550, loss_object: 0.059244
100%|██████████| 112/112 [00:26<00:00,  4.28it/s]
Epoch 2, lr: 0.010000, loss: 0.730265, loss_classifier: 0.308541, loss_box: 0.349349, loss_rpn_box: 0.025680, loss_object: 0.046695
100%|██████████| 112/112 [00:21<00:00,  5.27it/s]
Epoch 3, lr: 0.010000, loss: 0.679279, loss_classifier: 0.267776, loss_box: 0.345082, loss_rpn_box: 0.025103, loss_object: 0.041318
100%|██████████| 112/112 [00:19<00:00,  5.74it/s]
Epoch 4, lr: 0.010000, loss: 0.675116, loss_classifier: 0.268450, loss_box: 0.346733, loss_rpn_box: 0.024465, loss_object: 0.035469
100%|██████████| 112/112 [00:22<00:00,  5.01it/s]
Epoch 5, lr: 0.010000, loss: 0.658785, loss_classifier: 0.251549, loss_box: 0.351364, loss_rpn_box: 0.023568, loss_object: 0.032304
```

```
100%|██████████| 112/112 [00:19<00:00,  5.86it/s]
Epoch 6, lr: 0.010000, loss: 0.643597, loss_classifier: 0.247392, loss_box: 0.342181, loss_rpn_box: 0.022566, loss_object: 0.031460
100%|██████████| 112/112 [00:19<00:00,  5.82it/s]
Epoch 7, lr: 0.010000, loss: 0.654817, loss_classifier: 0.254710, loss_box: 0.344596, loss_rpn_box: 0.022629, loss_object: 0.032882
100%|██████████| 112/112 [00:19<00:00,  5.69it/s]
Epoch 8, lr: 0.010000, loss: 0.616348, loss_classifier: 0.228900, loss_box: 0.334681, loss_rpn_box: 0.021569, loss_object: 0.031198
100%|██████████| 112/112 [00:18<00:00,  5.95it/s]Epoch 9, lr: 0.010000, loss: 0.612210, loss_classifier: 0.222375, loss_box: 0.341918,
```

```python
# we will watch first epoich to ensure no errrors
# while it is training, lets write code to see the models predictions. lets try again
model.eval()
torch.cuda.empty_cache()
```

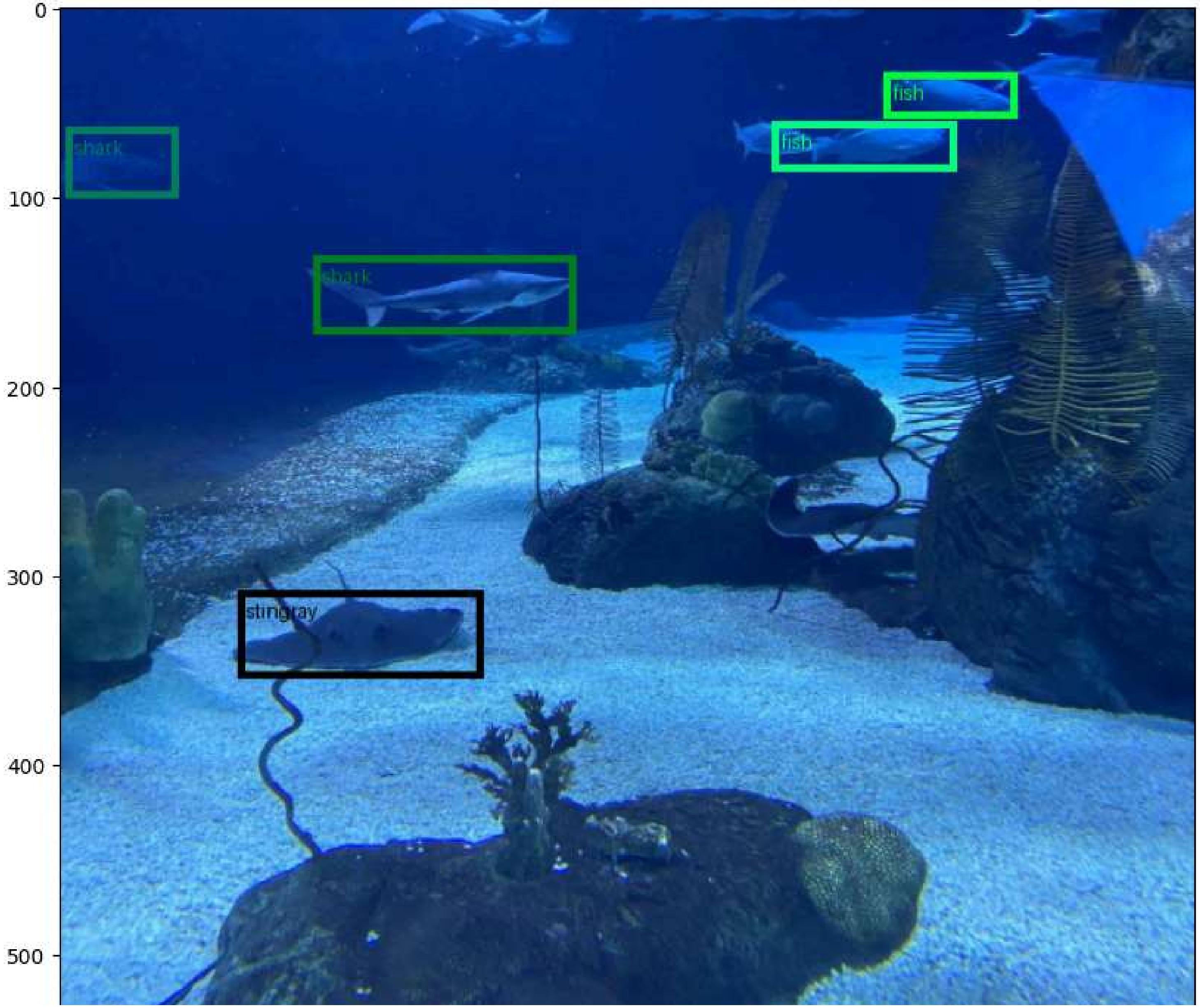
```python
test_dataset = AquariumDetection(root=dataset_path, split="test", transforms=get_transforms(False))
```

```
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
```

```python
img, _ = test_dataset[5]
img_int = torch.tensor(img*255, dtype=torch.uint8)
with torch.no_grad():
    prediction = model([img.to(device)])
    pred = prediction[0]
```

```python
fig = plt.figure(figsize=(14, 10))
plt.imshow(draw_bounding_boxes(img_int,
    pred['boxes'][pred['scores'] > 0.8],
    [classes[i] for i in pred['labels'][pred['scores'] > 0.8].tolist()], width=4
).permute(1, 2, 0))
```

```
<matplotlib.image.AxesImage at 0x7fd1c127f0a0>
```


0
100
200
300
400
500
fish
fish
shark
shark
stingray