HW4_test

November 3, 2022

[ ]:
```
from __future__ import print_function, division
# !pip install pytorch_lightning &> /dev/null


from torchvision.utils import draw_bounding_boxes
import os
import copy
import torch
import pandas as pd
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader, random_split, TensorDataset
from torchvision import transforms, utils
from  matplotlib.patches import Rectangle as rec
import numpy as np
import h5py
import cv2
import matplotlib.pyplot as plt
import torch
import torchvision
import torchvision.transforms as transforms
from PIL import Image
from scipy import ndimage
import torch.nn.functional as F

import torch.nn as nn
import pytorch_lightning as pl
import pytorch_lightning.loggers as pl_loggers
import pytorch_lightning.callbacks as pl_callbacks

# CUDA for PyTorch
use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")
seed = 17
torch.manual_seed(seed);
```

[ ]:
```
from dataset import *
from utils import *
from rpn_3 import *

from multiprocessing.connection import wait
import torch
from torch.nn import functional as F
from torchvision import transforms
from torch import nn, Tensor
import matplotlib.pyplot as plt
import torchvision
from scipy import stats as st


from torch.utils.data import Dataset, DataLoader
import h5py
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

[ ]:
```
# file path and make a list
imgs_path   = '/home/josh/Desktop/CIS680/HW4/FasterRCNN/data/
 ↪hw3_mycocodata_img_comp_zlib.h5'
masks_path  = '/home/josh/Desktop/CIS680/HW4/FasterRCNN/data/
 ↪hw3_mycocodata_mask_comp_zlib.h5'
labels_path = '/home/josh/Desktop/CIS680/HW4/FasterRCNN/data/
 ↪hw3_mycocodata_labels_comp_zlib.npy'
bboxes_path = '/home/josh/Desktop/CIS680/HW4/FasterRCNN/data/
 ↪hw3_mycocodata_bboxes_comp_zlib.npy'
paths = [imgs_path, masks_path, labels_path, bboxes_path]
# load the data into data.Dataset
dataset = BuildDataset(paths)

# build the dataloader
# set 20% of the dataset as the training data
full_size = len(dataset)
train_size = int(full_size * 0.8)
test_size = full_size - train_size
# random split the dataset into training and testset

train_dataset, test_dataset = torch.utils.data.random_split(dataset,␣
 ↪[train_size, test_size])
rpn_net = RPNHead()
# push the randomized training data into the dataloader

# train_loader = DataLoader(train_dataset, batch_size=2, shuffle=True,␣
 ↪num_workers=0)
```

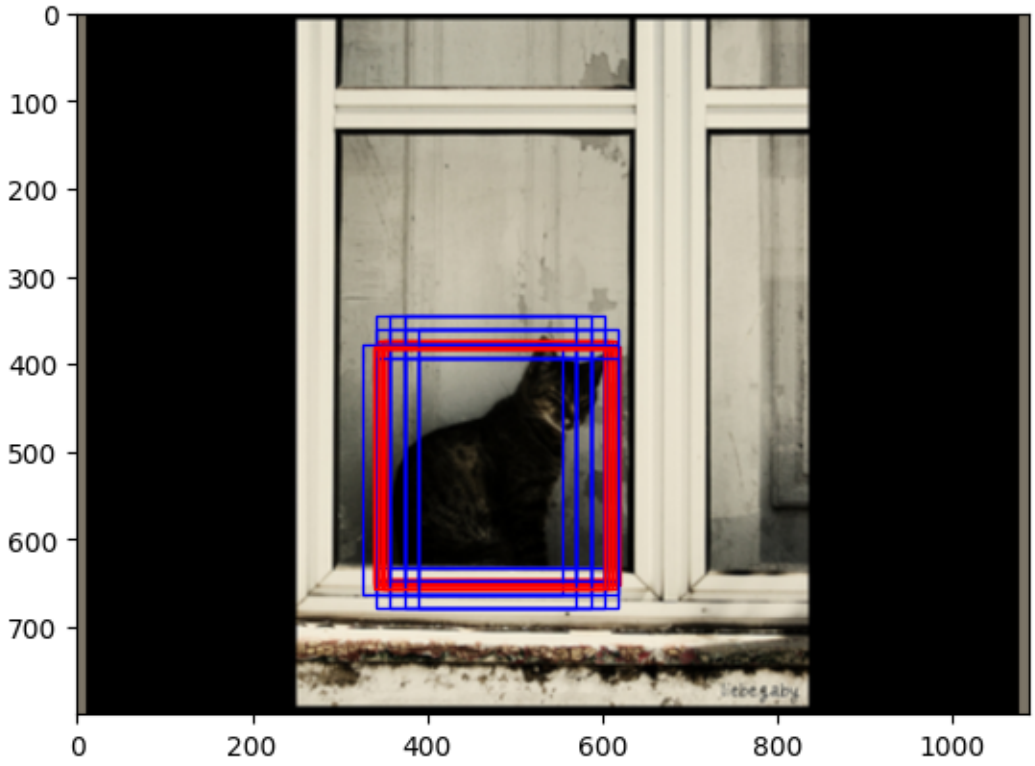
```
# test_loader  = DataLoader(test_dataset, batch_size=2, shuffle=False,␣
 ↪num_workers=0)
batch_size = 2
train_build_loader = BuildDataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=0)
train_loader = train_build_loader.loader()
test_build_loader = BuildDataLoader(test_dataset, batch_size=batch_size,␣
 ↪shuffle=False, num_workers=0)
test_loader = test_build_loader.loader()
```

```
/home/josh/.local/lib/python3.8/site-packages/torch/functional.py:568:
UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass
the indexing argument. (Triggered internally at
../aten/src/ATen/native/TensorShape.cpp:2228.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
```

[ ]:
```
for i,batch in enumerate(train_loader,0):
    images = batch['images'][0]
    # images = torch.stack(images[:])
    indexes= batch['index']
    boxes  = batch['bbox']

    anchors = rpn_net.create_anchors(rpn_net.anchors_param["ratio"], rpn_net.
 ↪anchors_param["scale"], rpn_net.anchors_param["grid_size"], rpn_net.
 ↪anchors_param["stride"])
    # gt,ground_coord=rpn_net.create_batch_truth(boxes, 1, (800,1088))
    gt,ground_coord = rpn_net.create_ground_truth(torch.from_numpy(boxes[0]),␣
 ↪indexes[0], rpn_net.anchors_param["grid_size"], anchors, (800,1088))


    # Flatten the ground truth and the anchors
    flatten_coord,flatten_gt,flatten_anchors=output_flattening(ground_coord.
 ↪unsqueeze(0),gt.unsqueeze(0),rpn_net.anchors)

    # Decode the ground truth box to get the upper left and lower right corners␣
 ↪of the ground truth boxes
    decoded_coord=output_decoding(flatten_coord,flatten_anchors)

    # Plot the image and the anchor boxes with the positive labels and their␣
 ↪corresponding ground truth box
    images = transforms.functional.normalize(images,
                                             [-0.485/0.229, -0.456/0.
 ↪224, -0.406/0.225],
                                             [1/0.229, 1/0.224, 1/0.
 ↪225], inplace=False)
    fig,ax=plt.subplots(1,1)
```

```
    ax.imshow(images.permute(1,2,0))

    find_cor=(flatten_gt==1).nonzero()
    find_neg=(flatten_gt==-1).nonzero()

    for elem in find_cor:
        coord=decoded_coord[elem,:].view(-1)
        anchor=flatten_anchors[elem,:].view(-1)

        col='r'
        rect=patches.
↪Rectangle((coord[0],coord[1]),coord[2]-coord[0],coord[3]-coord[1],fill=False,color=col)
        ax.add_patch(rect)
        rect=patches.Rectangle((anchor[0]-anchor[2]/2,anchor[1]-anchor[3]/
↪2),anchor[2],anchor[3],fill=False,color='b')
        ax.add_patch(rect)

    plt.show()

    if(i < 20):
        break
```

```
/home/josh/Desktop/CIS680/HW4/FasterRCNN/Code_template_HW4_PartA/rpn_3.py:249:
UserWarning: __floordiv__ is deprecated, and its behavior will change in a
future version of pytorch. It currently rounds toward 0 (like the 'trunc'
function NOT 'floor'). This results in incorrect rounding for negative values.
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for
actual floor division, use torch.div(a, b, rounding_mode='floor').
  row          = invalid // 68
/home/josh/Desktop/CIS680/HW4/FasterRCNN/Code_template_HW4_PartA/rpn_3.py:256:
UserWarning: __floordiv__ is deprecated, and its behavior will change in a
future version of pytorch. It currently rounds toward 0 (like the 'trunc'
function NOT 'floor'). This results in incorrect rounding for negative values.
To keep the current behavior, use torch.div(a, b, rounding_mode='trunc'), or for
actual floor division, use torch.div(a, b, rounding_mode='floor').
  row_anc          = valid_anchor_idx // 68
```



[ ]:
```
from pytorch_lightning.callbacks import ModelCheckpoint
val_checkpoint_callback = ModelCheckpoint(
    monitor="val_loss",
    dirpath="./training_data_new_model_10",
    filename="val_loss{epoch:02d}-{val_loss:.2f}",
    save_top_k=3,
    mode="min",
)
train_checkpoint_callback = ModelCheckpoint(
    monitor="train_loss",
    dirpath="./training_data_new_model_10",
    filename="train_loss{epoch:02d}-{train_loss:.2f}",
    save_top_k=3,
    mode="min",
)
model = RPNHead()
tb_logger = pl_loggers.TensorBoardLogger("logs9/")
trainer = pl.Trainer(gpus=1,logger=tb_logger,␣
 ↪max_epochs=36,callbacks=[val_checkpoint_callback,train_checkpoint_callback])
trainer.fit(model,train_loader)
```

```
----------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call last)
Cell In [6], line 5
      2 model = RPNHead()
      4 for i,batch in enumerate(train_loader,0):
----> 5     model.forward()

TypeError: forward() missing 1 required positional argument: 'X'
```

[ ]:
```
model2 = RPNHead()
trainer = pl.Trainer()
chk_path = "/home/josh/Desktop/CIS680/HW4/FasterRCNN/Code_template_HW4_PartA/
 ↪train_lossepoch=33-train_loss=1.29.ckpt"
model2 = RPNHead.load_from_checkpoint(chk_path)
# results = trainer.test(model=model2, datamodule=my_datamodule, verbose=True)
```

```
GPU available: True (cuda), used: False
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
/home/josh/.local/lib/python3.8/site-
packages/pytorch_lightning/trainer/trainer.py:1764: PossibleUserWarning: GPU
available but not used. Set `accelerator` and `devices` using
`Trainer(accelerator='gpu', devices=1)`.
  rank_zero_warn(
```

[ ]:
```
for i,batch in enumerate(train_loader,0):
    images = batch['images']
    images = torch.stack(images[:])
    print(images.shape)

    logits, bbox_regs = model2.forward(images)


    # print(bbox_regs.shape)
    if i == 0:
        break
```

```
torch.Size([2, 3, 800, 1088])
torch.Size([2, 4, 50, 68])
```

[ ]: