**YOLO Reproduction-5**

Advisor: Dr. Chih-Yu Wang

Presenter: Shao-Heng Chen

Date: August 3, 2022

1. YOLOv3 progress status

(1) it’s trainable now! (<https://github.com/paulchen2713/YOLO_project>)

(pt3.7) D:\BeginnerPythonProjects>C:/Users/paulc/.conda/envs/pt3.7/python.exe d:/BeginnerPythonProjects/YOLOv3-

PyTorch/YOLOv3-0801/train.py

100%|████████████████████████████████████| 94/94 [00:43<00:00, 2.14it/s, loss=27]

Currently epoch 1

On Train loader:

100%|█████████████████████████████████████████| 94/94 [00:20<00:00, 4.54it/s]

Class accuracy is: 100.000000%

No obj accuracy is: 73.035225%

Obj accuracy is: 99.755394%

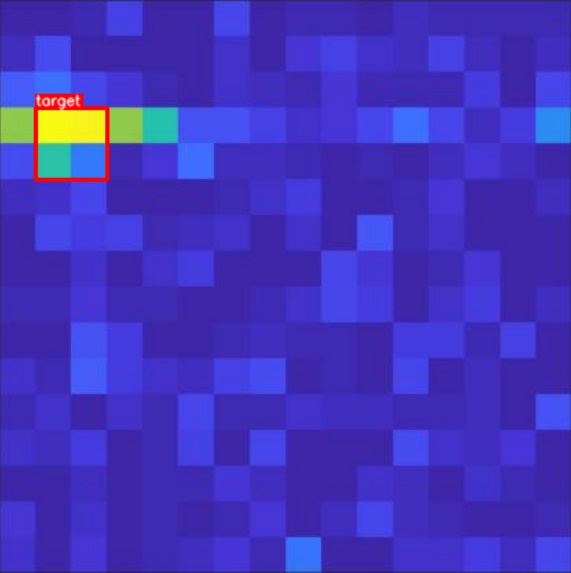
On Test loader:

100%|██████████████████████████████████████████| 7/7 [00:03<00:00, 1.81it/s]

Class accuracy is: 100.000000%

No obj accuracy is: 73.054192%

Obj accuracy is: 100.000000%



(2) training data

-image format: 416-by-416 .jpg files

D:\Datasets\RD\_maps>tree

D:.

├─checks

├─images

├─labels

├─mats

├─mesh\_figures

└─scaled\_colors

(pt3.7) D:\BeginnerPythonProjects>C:/Users/paulc/.conda/envs/pt3.7/python.exe d:/BeginnerPythonProjects/YOLOv3-

PyTorch/YOLOv3-0801/config.py

image: 229.txt

bbox: [[52.0, 104.0, 52.0, 52.0]]

(3) some unknown and unfixed bugs

RuntimeError: Given groups=1, weight of size [32, 3, 3, 3], expected input[16, 416, 416, 3] to have 3 channels,

but got 416 channels instead

RuntimeError: result type Float can't be cast to the desired output type Byte

RuntimeError: Input type (torch.cuda.ByteTensor) and weight type (torch.cuda.HalfTensor) should be the same

ValueError: Expected y\_max for bbox (0.375, 0.9375, 0.5, 1.0625, 0.0) to be in the range [0.0, 1.0]

Appendix.

(a) image resizing

*def* main(*max\_iter*=1, *file\_type*='jpg'):

    # 1600

    for i in range(1, max\_iter + 1):

        # read the input image

        img = Image.open(*f*'D:/Datasets/RD\_maps/scaled\_colors/{i}\_sc.{file\_type}')

        # define the transform function to resize the image with given size, say 416-by-416

        transform = T.Resize(*size*=(416,416))

        # apply the transform on the input image

        img = transform(img)

        # overwrite the original image with the resized one

        img = img.save(*f*'D:/Datasets/RD\_maps/scaled\_colors/{i}\_sc.{file\_type}')

        print(*f*"{i}")

if \_\_name\_\_ == '\_\_main\_\_':

    # testing(1, 'jpg')

    main(1, 'jpg')

(b) augmentation testing with albumentations and cv2 libraries

*def* test():

    # Import the required libraries, besides albumentations and cv2

    import random

    # from PIL import Image

    import numpy as np

    from matplotlib import pyplot as plt

    # Define functions to visualize bounding boxes and class labels on an image

    BOX\_COLOR = (255, 0, 0)      # Red

    TEXT\_COLOR = (255, 255, 255) # White

*def* visualize\_bbox(*img*, *bbox*, *class\_name*, *color*=BOX\_COLOR, *thickness*=2):

        """Visualizes a single bounding box on the image"""

        # YOLO format

        x, y, w, h = bbox

        x\_min, x\_max = *int*((2\*x - h) / 2), *int*((2\*x + h) / 2)

        y\_min, y\_max = *int*((2\*y - w) / 2), *int*((2\*y + w) / 2)

        # COCO format

        # x\_min, y\_min, w, h = bbox

        # x\_min, x\_max, y\_min, y\_max = int(x\_min), int(x\_min + w), int(y\_min), int(y\_min + h)

        cv2.rectangle(img, (x\_min, y\_min), (x\_max, y\_max), *color*=color, *thickness*=thickness)

        ((text\_width, text\_height), \_) = cv2.getTextSize(class\_name, cv2.FONT\_HERSHEY\_SIMPLEX, 0.35, 1)

        cv2.rectangle(img, (x\_min, y\_min - *int*(1.3 \* text\_height)), (x\_min + text\_width, y\_min), BOX\_COLOR, -1)

        cv2.putText(

            img,

*text*=class\_name,

*org*=(x\_min, y\_min - *int*(0.3 \* text\_height)),

*fontFace*=cv2.FONT\_HERSHEY\_SIMPLEX,

*fontScale*=0.35,

*color*=TEXT\_COLOR,

*lineType*=cv2.LINE\_AA,

        )

        return img

*def* visualize(*image*, *bboxes*, *category\_ids*, *category\_id\_to\_name*):

        img = image.copy()

        for bbox, category\_id in zip(bboxes, category\_ids):

            class\_name = category\_id\_to\_name[category\_id]

            img = visualize\_bbox(img, bbox, class\_name)

        plt.figure(*figsize*=(12, 12))

        plt.axis('off')

        plt.imshow(img)

        plt.show()

    # Load the image and the annotations for it

    img\_idx = random.randint(1, 1000) # get a random image index

    print(*f*"image: {img\_idx}.txt")

    # we can read the image through cv2.imread() in BGR or PIL.Image.open() in RGB, but the visualiz()

    # and visualize\_bbox() functions are implemented with cv2, so we should stick to it to avoid errors

    img\_path = IMG\_DIR + *f*'{img\_idx}\_sc.jpg'

    image = cv2.imread(img\_path) # NOTE cv2.imread() read the image in BGR, 0~255, (W, H, C)

    image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # must first convert BGR into RGB

    label\_path = LABEL\_DIR + *f*'{img\_idx}.txt'

    label = np.loadtxt(*fname*=label\_path, *delimiter*=" ", *ndmin*=2).tolist()

    # print(label[0][1:])

    true\_scale = [label[0][i]\*IMAGE\_SIZE for i in range(1, 5)]

    # print(true\_scale)

    bboxes = *list*()

    bboxes.append(true\_scale)

    print(*f*"bbox: {bboxes}")

    # bboxes, category\_ids and category\_id\_to\_name all has to be iterable object

    category\_ids = [0]

    category\_id\_to\_name = {0: 'target'}

    # Visuaize the original image with bounding boxes

    # visualize(image, bboxes, category\_ids, category\_id\_to\_name)

    # Define an augmentation pipeline

    tscale = 1.5

    transform = A.Compose(

        [

            # A.LongestMaxSize(max\_size=int(IMAGE\_SIZE \* tscale), p=1.0),

            # A.PadIfNeeded(min\_height=int(IMAGE\_SIZE \* tscale), min\_width=int(IMAGE\_SIZE \* tscale), border\_mode=cv2.BORDER\_CONSTANT, ),

            # A.RandomCrop(width=IMAGE\_SIZE, height=IMAGE\_SIZE, p=1.0),

            # A.ColorJitter(brightness=0.6, contrast=0.6, saturation=0.6, hue=0.6, p=1.0),

            # A.ShiftScaleRotate(rotate\_limit=20, p=1.0, border\_mode=cv2.BORDER\_CONSTANT),

            # A.HorizontalFlip(p=1.0),

            # A.Blur(blur\_limit=7, p=1.0),

            # A.CLAHE(clip\_limit=4.0, tile\_grid\_size=(8, 8), p=1.0),

            # A.Posterize(p=1.0),

            # A.ToGray(p=1.0),

            # A.ChannelShuffle(p=1.0),

            # A.Normalize(mean=[0, 0, 0], std=[1, 1, 1], max\_pixel\_value=255.0, p=1.0),

        ],

*bbox\_params*=A.BboxParams(

*format*='coco',

*label\_fields*=['category\_ids']

        ),

    )

    random.seed(33)

    transformed = transform(*image*=image, *bboxes*=bboxes, *category\_ids*=category\_ids)

    visualize(transformed['image'], transformed['bboxes'], transformed['category\_ids'], category\_id\_to\_name)

    # Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers)

if \_\_name\_\_ == "\_\_main\_\_":

    test()

(c) dataset testing

*def* test():

    anchors = config.ANCHORS

    transform = config.test\_transforms

    # S = [2, 4, 8]

    # S = [13, 26, 52]

    S = config.S

    # PASCAL VOC "D:/Datasets/PASCAL\_VOC/train.csv", "D:/Datasets/PASCAL\_VOC/images", "D:/Datasets/PASCAL\_VOC/labels",

    # MS COCO    "COCO/train.csv", "COCO/images/images/", "COCO/labels/labels\_new/"

    dataset = YOLODataset(

        "D:/Datasets/RD\_maps/train.csv", # csv\_file

        "D:/Datasets/RD\_maps/scaled\_colors",    # img\_dir

        "D:/Datasets/RD\_maps/labels",    # label\_dir

*S*=S, # S=[13, 26, 52],

*anchors*=anchors,

*transform*=transform,

    )

    scaled\_anchors = torch.tensor(anchors) / (1 / torch.tensor(S).unsqueeze(1).unsqueeze(1).repeat(1, 3, 2))

    loader = DataLoader(*dataset*=dataset, *batch\_size*=1, *shuffle*=True)

    counter = 0 # count number of tests

    for x, y in loader:

        # print(f"x[0] shape: {x[0].shape}") # NOTE torch.Size([416, 416, 3])

        boxes = []

        for i in range(y[0].shape[1]):

            anchor = scaled\_anchors[i]

            print(*f*"anchor.shape: {anchor.shape}") # torch.Size([3, 2])

            print(*f*"y[{i}].shape: {y[i].shape}")

            # y[0].shape: torch.Size([1, 3, 13, 13, 6])

            # y[1].shape: torch.Size([1, 3, 26, 26, 6])

            # y[2].shape: torch.Size([1, 3, 52, 52, 6])

            boxes += cells\_to\_bboxes(y[i], *is\_preds*=False, *S*=y[i].shape[2], *anchors*=anchor)[0]

        boxes = nms(boxes, *iou\_threshold*=1, *threshold*=0.7, *box\_format*="midpoint")

        print(*f*"boxes: {boxes}")

        # plot\_image(x[0].permute(1, 2, 0).to("cpu"), boxes) # torch.Size([416, 416, 3])

        #

        # TypeError: Invalid shape (3, 416, 416) for image data

        # x[0].permute(0, 1, 2).shape is the original shape with torch.Size([3, 416, 416]) and it's an Invalid shape for image data

        # so x[0].permute(1, 2, 0).shape is the valid shape with torch.Size([416, 416, 3])

        print("original shape: ", x[0].permute(0, 1, 2).shape) # torch.Size([3, 416, 416])

        plot\_image(x[0].permute(1, 2, 0).to("cpu"), boxes) #

        print("-----------------------------------------")

        counter += 1

        if counter == 1: break # run the test for some times then we stop

        # sometimes would run into out of bound ValueError, NOTE probabily caused by transforms, scale, and bbox\_params settings!

        # File "C:\Users\paulc\.conda\envs\pt3.7\lib\site-packages\albumentations\augmentations\bbox\_utils.py", line 330, in check\_bbox

        #     "to be in the range [0.0, 1.0], got {value}.".format(bbox=bbox, name=name, value=value)

        # ValueError: Expected x\_max for bbox (0.9375, 0.875, 1.0625, 1.0, 0.0) to be in the range [0.0, 1.0], got 1.0625.

if \_\_name\_\_ == "\_\_main\_\_":

    test()