hw5_ChengjunGuo

Chengjun Guo

March 2023

1 Data selection and preprocessing

The images are pre-downloaded. The dataloader would first load the category ID based on the category names and then load all the image IDs that include the category IDs. The largest segmentation with expected category would be the ground truth. The image would be resized to 256×256, the bounding box will be normalized to (0,1). The label is the corresponding category.

2 input with annotation

label0: pizza, label1: bus, label2: cat

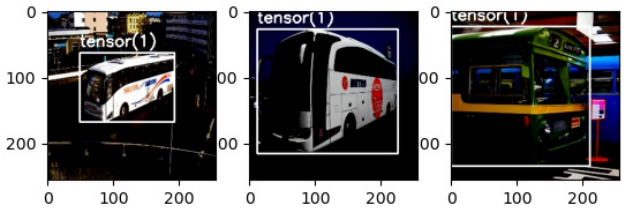


Figure 1: bus

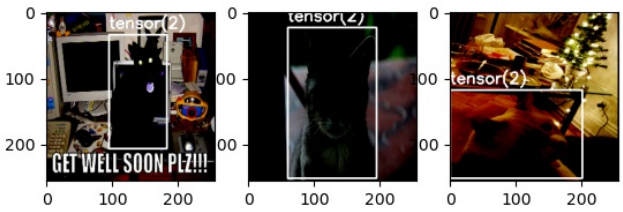


Figure 2: cat

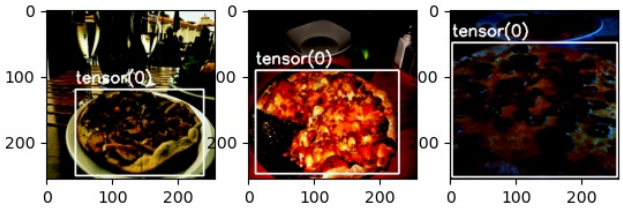


Figure 3: pizza

3 Network structure

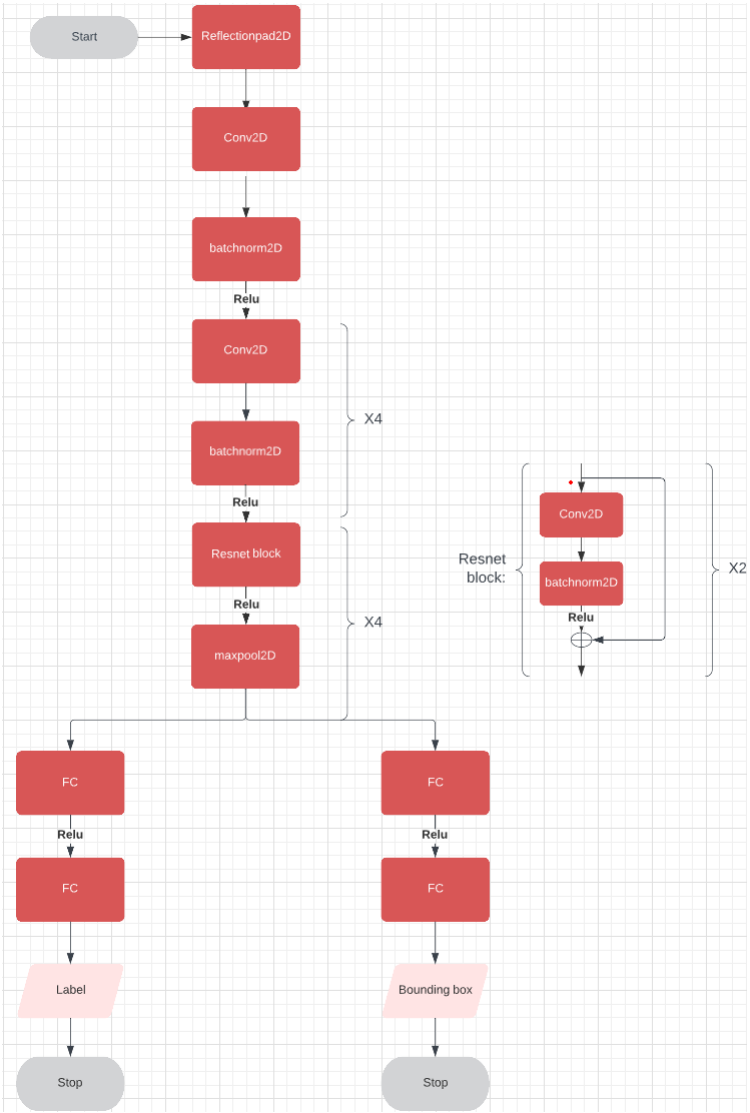


Figure 4: 60 layer network

4 Confusion Matrix

Confusion Matrix MSE:

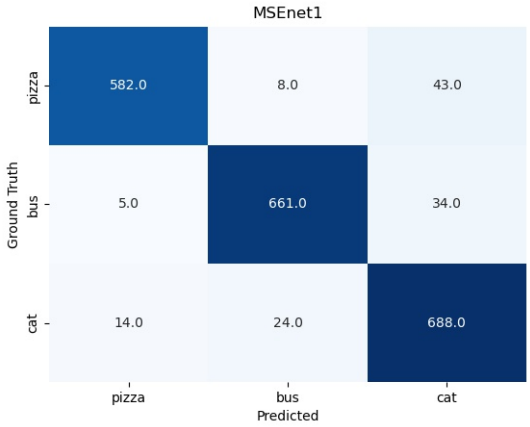


Figure 5: MSE confusion matrix

Confusion Matrix CIoU:

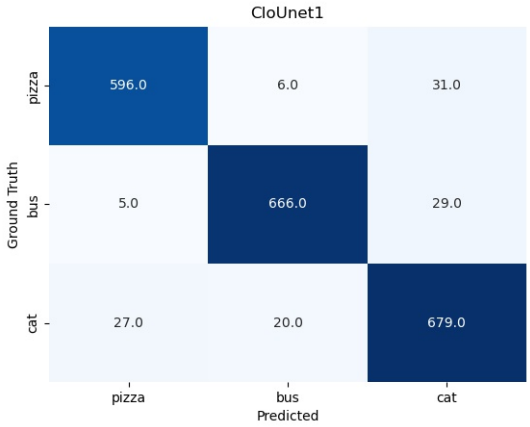
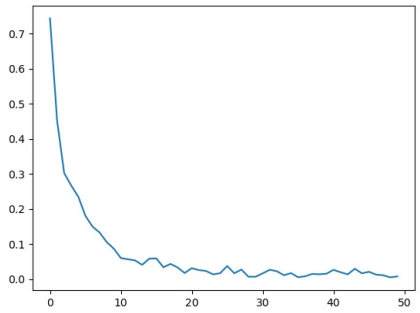
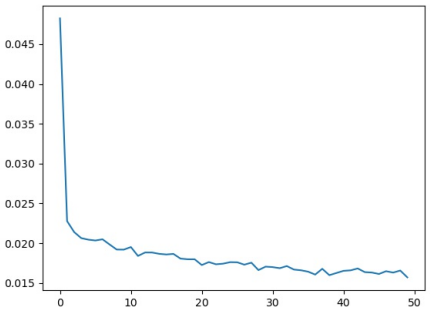


Figure 6: CIoU confusion matrix

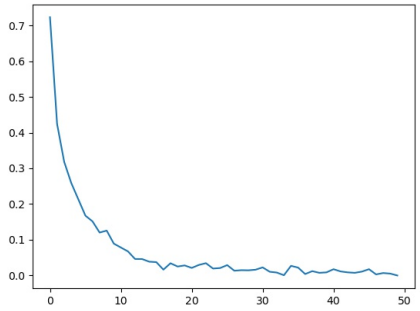
5 Training loss



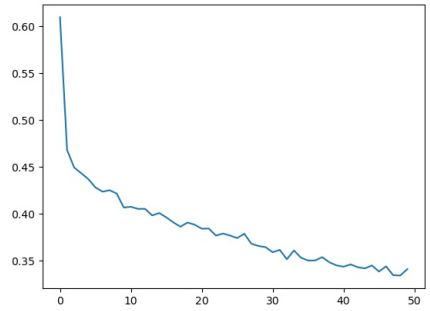
(a) MSE labeling



(b) MSE regression



(c) CIoU labeling



(d) CIoU regression

Figure 7: training loss

6 Accuracy

Classification accuracy:
MSE:0.937
CIoU:0.943
Localization accuracy:
MSE:0.563
CIoU:0.591

7 prediction with annotation

label0: pizza, label1: bus, label2: cat

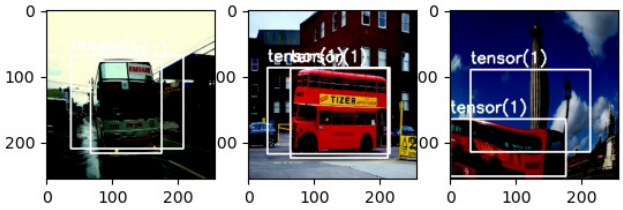


Figure 8: bus

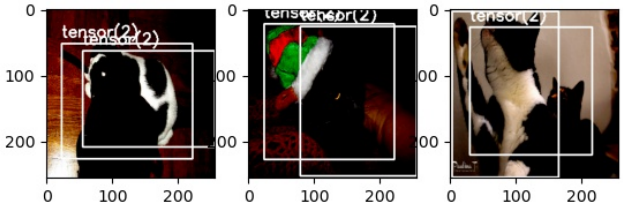


Figure 9: cat

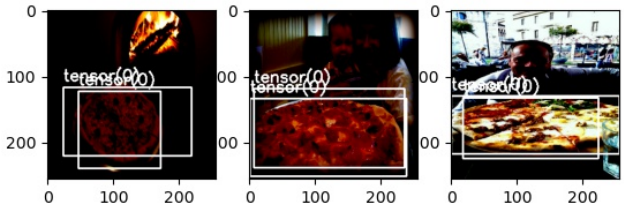


Figure 10: pizza

8 performance

The localization is relatively hard to train compared to the classification. After 10 epochs, the regression loss stopped to converge. For the localization in deep learning, my first guess is that the deep neural network would need more epochs. The second guess is that the localization information is not efficiently passed through the network. There would be several traditional computer vision approaches to achieve that. Dilation and edge detection information can be passed into the network to help fast converging. Just like the yolo algorithm, anchor box is one way to pass the information to the network.

9 Code

COCO dataset:

```
import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
import torch
from PIL import Image
import torchvision
import torchvision.transforms as tvt
```

```
from torchvision.io import read_image
from torch.utils.data import DataLoader, Dataset
import copy
from pycocotools.coco import COCO
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np
from skimage import color, io
from skimage.transform import resize
import cv2
from skimage import img_as_ubyte


class MyDataset(torch.utils.data.Dataset):
    def __init__(self,istrain):
        super().__init__()
        self.istrain = istrain
        self.root_path = 'E:\ECE60146DL\hw5_new/'
        self.train_path = 'E:\ECE60146DL\hw5_new/
            train2014/'
        self.val_path = 'E:\ECE60146DL\hw5_new/val2014/'
        self.train_json_path = 'annotations_trainval2014/
            annotations/instances_train2014.json'
        self.val_json_path = 'annotations_trainval2014/
            annotations/instances_val2014.json'
        self.class_list = ['pizza','bus','cat']
        self.info = {}
        self.data = []
        if self.istrain:
            self.coco = COCO(self.train_json_path)
        else:
            self.coco = COCO(self.val_json_path)
        self.transform = tvt.Compose([tvt.ToTensor(),tvt.
            Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])])
        # for label,cat in enumerate(self.class_list):
        catIds = self.coco.getCatIds(catNms=self.
            class_list)
        imgIds = []
        for i in catIds:
            imgIds += self.coco.getImgIds(catIds = i)
        # annIds = self.coco.getAnnIds(imgIds=imgIds,
            catIds=catIds, iscrowd=None)
        imgs = self.coco.loadImgs(imgIds)
        # print(catIds,len(imgIds))
        # print(len(imgs))
```

```
        # anns = self.coco.loadAnns(annIds)
        # # img [{'license': 3, 'file_name': '
            COCO_train2014_000000106497.jpg', 'coco_url':
            'http://images.cocodataset.org/train2014/
            COCO_train2014_000000106497.jpg', 'height':
            429, 'width': 640, 'date_captured':
            '2013-11-21 04:25:40', 'flickr_url': 'http://
            farm6.staticflickr.com/5214/5538330782
            _ea1580ea4f_z.jpg', 'id': 106497}]
        for img in imgs:
            annIds = self.coco.getAnnIds(imgIds=img['id
                '])
            anns = self.coco.loadAnns(annIds)
            area = []
            ann_list = []
            for ann in anns:
                if ann['category_id'] in catIds and ann['
                    area'] >= 40000:
                    # print('detected something')
                    area.append(ann['area'])
                    ann_list.append(ann)
            # for ann in anns:
            #     # if catIds[0] == ann['category_id']:
            #     #     print('?')
            #     if catIds[0] == ann['category_id'] and
                ann['area']==max(area):
            #         self.info[img['id']] = [img['
                file_name'], label, ann['bbox'],(img['
                height'],img['width'])]
            if ann_list:
                idx = area.index(max(area))
                self.info[img['id']] = [img['file_name'],
                    ann_list[idx]['category_id'],
                    ann_list[idx]['bbox'], (img['height'],
                    img['width'])]
        self.img_ids = list(self.info.keys())


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

    def __getitem__(self, index):
        data = self.info[self.img_ids[index]]
        # print(data)
        if self.istrain:
            pth = self.train_path
```

```
        else:
            pth = self.val_path
        img = io.imread(pth+data[0])
        if img.ndim == 2 or img.shape[-1]<3:
            # print('is this working?')
            img = color.gray2rgb(img)
        img = resize(img,(256,256),anti_aliasing=True)
        label = torch.tensor(encoder_for_pbc(data[1]))
        image = self.transform(img)
        x,y,w,h = data[2][0]/data[3][1],data[2][1]/data
            [3][0],data[2][2]/data[3][1],data[2][3]/data
            [3][0]
        bbox = torch.tensor([x,y,x+w,y+h],dtype=torch.
            float)
        return image.type(torch.float), label, bbox

def encoder_for_pbc(category_id):
    dict = {59:0,6:1,17:2}
    return dict[category_id]



if __name__ == '__main__':
    my_dataset = MyDataset(True)
    print(len(my_dataset))
    # print(my_dataset.info)
    print(my_dataset[10][0].type(),my_dataset[10][1].type
        (),my_dataset[10][2].type())
    print(my_dataset[10][0].shape)
    # for i in range(3000):
    #     if my_dataset[i][1].int() != 1:
    #         print(i,my_dataset[i][1].int())
    # plt.figure()
    # for j,i in enumerate([10,13,15]):
    #     ax = plt.subplot(1,3,j+1)
    #     image = my_dataset[i][0].numpy().transpose(1,
        2, 0).copy()
    #     image = cv2.rectangle(image, (int(my_dataset[i
        ][2][0] * 256),int(my_dataset[i][2][1] * 256)),(
        int(my_dataset[i][2][2] * 256),int(my_dataset[i
        ][2][3] * 256)), (36,255,12),2)
    #     image = cv2.putText(image, str(my_dataset[i
        ][1]), (int(my_dataset[i][2][0] * 256),int(
        my_dataset[i][2][1] * 256 - 10)), cv2.
        FONT_HERSHEY_SIMPLEX, 0.8, (36,255,12),2)
    #     ax.imshow(image)
```

```
    # plt.savefig('input.jpg')
    plt.figure()
    for j,i in enumerate([1437,1438,1439]):
        ax = plt.subplot(1,3,j+1)
        image = my_dataset[i][0].numpy().transpose(1, 2,
            0).copy()
        image = cv2.rectangle(image, (int(my_dataset[i
            ][2][0] * 256),int(my_dataset[i][2][1] * 256))
            ,(int(my_dataset[i][2][2] * 256),int(
            my_dataset[i][2][3] * 256)), (36,255,12),2)
        image = cv2.putText(image, str(my_dataset[i][1]),
             (int(my_dataset[i][2][0] * 256),int(
            my_dataset[i][2][1] * 256 - 10)), cv2.
            FONT_HERSHEY_SIMPLEX, 0.8, (36,255,12),2)
        ax.imshow(image)
    plt.savefig('input2.jpg')
    plt.figure()
    for j, i in enumerate([2731, 2732, 2733]):
        ax = plt.subplot(1, 3, j + 1)
        image = my_dataset[i][0].numpy().transpose(1, 2,
            0).copy()
        image = cv2.rectangle(image, (int(my_dataset[i
            ][2][0] * 256), int(my_dataset[i][2][1] * 256)
            ),(int(my_dataset[i][2][2] * 256), int(
            my_dataset[i][2][3] * 256)), (36, 255, 12), 2)
        image = cv2.putText(image, str(my_dataset[i][1])
            ,(int(my_dataset[i][2][0] * 256), int(
            my_dataset[i][2][1] * 256 - 10)),cv2.
            FONT_HERSHEY_SIMPLEX, 0.8, (36, 255, 12), 2)
        ax.imshow(image)
    plt.savefig('input3.jpg')
    # my_dataset = MyDataset(False)
    # print(len(my_dataset))
```

net:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class HW5Net(nn.Module):
    def __init__(self,input_nc, output_nc, ngf=8,
```

```
        n_blocks=4):
        super(HW5Net, self).__init__()
        model = [nn.ReflectionPad2d(3),
                 nn.Conv2d(input_nc, ngf, kernel_size=7,
                     padding=0),
                 nn.BatchNorm2d(ngf),
                 nn.ReLU(True)]
        n_downsampling = 4
        for i in range(n_downsampling):
            mult = 2 ** i
            model += [nn.Conv2d(ngf * mult, ngf * mult *
                2, kernel_size=3, stride=2, padding=1),
                      nn.BatchNorm2d(ngf * mult * 2),
                      nn.ReLU(True)]
        mult = 2 ** n_downsampling
        for i in range(n_blocks):
            model += [ResnetBlock(ngf * mult,ngf * mult *
                 2,ngf * mult),
                      nn.ReLU(True),
                      nn.MaxPool2d(2, 2)
                      ]
        self.model = nn.Sequential(*model)
        class_head = [
            nn.Linear(ngf * mult, 64),
            nn.ReLU(True),
            nn.Linear(64, 3)
        ]
        self.class_head = nn.Sequential(*class_head)
        bbox_head = [
            nn.Linear(ngf * mult, 64),
            nn.ReLU(True),
            nn.Linear(64, 4)
        ]
        self.bbox_head = nn.Sequential(*bbox_head)


    def forward(self, input):
        ft = self.model(input)
        ft = ft.view(ft.shape[0], -1)
        cls = self.class_head(ft)
        bbox = self.bbox_head(ft)
        return cls, bbox

class ResnetBlock(nn.Module):

    def __init__(self, in_size, mid_size, out_size):
```

```
        super(ResnetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_size, mid_size, 3,
            padding=1)
        self.conv2 = nn.Conv2d(mid_size, out_size, 3,
            padding=1)
        self.batchnorm1 = nn.BatchNorm2d(mid_size)
        self.batchnorm2 = nn.BatchNorm2d(out_size)


    def forward(self, x):
        y = x
        x = F.relu(self.batchnorm1(self.conv1(x)))
        x = F.relu(self.batchnorm2(self.conv2(x)))
        return y+x
```

train:

```
from hw5_COCO_dataset import MyDataset
import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
import torch
from PIL import Image
import torchvision
import torchvision.transforms as tvt
from torchvision.io import read_image
from torch.utils.data import DataLoader, Dataset
import copy
from pycocotools.coco import COCO
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from hw5_net import HW5Net
import time
import pickle
from statistics import mean
import cv2


def run_training_MSE(net,train_data_loader,net_save_path)
    :
    net = copy.deepcopy(net)
    net = net.to(device)
```

```
criterion1 = torch.nn.CrossEntropyLoss()
criterion2 = torch.nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=1e
    -3, betas=(0.9, 0.99))
epochs = 50
Loss_labeling_runtime = []
Loss_regression_runtime = []
for epoch in range(epochs):
    start_time = time.time()
    running_loss_labeling = 0.0
    running_loss_regression = 0.0
    for i,data in enumerate(train_data_loader):
        inputs, labels, bbox = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        bbox = bbox.to(device)
        optimizer.zero_grad()
        outputs = net(inputs)
        pred_label = outputs[0]
        # pred_label = torch.max(outputs[0],1)[1]
        pred_bbox = outputs[1]
        # print(pred_label.type(),pred_bbox.type(),
            labels.type(),bbox.type())
        # print(labels.dtype, pred_label.dtype)
        loss_labeling = criterion1(pred_label, labels
            )
        loss_labeling.backward(retain_graph=True)
        loss_regression = criterion2(pred_bbox, bbox)
        loss_regression.backward()
        optimizer.step()
        running_loss_labeling += loss_labeling.item()
        running_loss_regression += loss_regression.
            item()
        if (i+1) % 500 == 0:
            print("[epoch: %d, batch: %5d] regression
                 loss: %.3f labeling loss: %.3f" % (
                epoch+1, i+1, running_loss_regression
                /500, running_loss_labeling/500))
            Loss_regression_runtime.append(
                running_loss_regression/500)
            Loss_labeling_runtime.append(
                running_loss_labeling/500)
            running_loss_regression = 0.0
            running_loss_labeling = 0.0
            print("----------time executing for epoch
                %s : %s seconds----------" % (epoch +
```

```
                1, (time.time() - start_time)))
    torch.save(net, net_save_path)
    return Loss_labeling_runtime, Loss_regression_runtime


def run_training_CIoU(net,train_data_loader,net_save_path
    ):
    net = copy.deepcopy(net)
    net = net.to(device)
    criterion1 = torch.nn.CrossEntropyLoss()
    criterion2 = torchvision.ops.complete_box_iou_loss
    optimizer = torch.optim.Adam(net.parameters(), lr=1e
        -3, betas=(0.9, 0.99))
    epochs = 50
    Loss_labeling_runtime = []
    Loss_regression_runtime = []
    for epoch in range(epochs):
        running_loss_labeling = 0.0
        running_loss_regression = 0.0
        for i,data in enumerate(train_data_loader):
            inputs, labels, bbox = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            bbox = bbox.to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            pred_label = outputs[0]
            pred_bbox = outputs[1]
            loss_labeling = criterion1(pred_label, labels
                )
            loss_labeling.backward(retain_graph=True)
            loss_regression = criterion2(pred_bbox, bbox,
                reduction='mean')
            loss_regression.backward()
            optimizer.step()
            running_loss_labeling += loss_labeling.item()
            running_loss_regression += loss_regression.
                item()
            if (i+1) % 500 == 0:
                print("[epoch: %d, batch: %5d] regression
                    loss: %.3f labeling loss: %.3f" % (
                    epoch+1, i+1, running_loss_regression
                    /500, running_loss_labeling/500))
                Loss_regression_runtime.append(
                    running_loss_regression/500)
                Loss_labeling_runtime.append(
```

```
                    running_loss_labeling/500)
                running_loss_regression = 0.0
                running_loss_labeling = 0.0
    torch.save(net, net_save_path)
    return Loss_labeling_runtime, Loss_regression_runtime


def run_testing(net, validation_data_loader):
    net = copy.deepcopy(net)
    net = net.to(device)
    Confusion_Matrix = torch.zeros(3, 3)
    iou = []
    for i, data in enumerate(validation_data_loader):
        inputs, labels, bbox = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        bbox = bbox.to(device)
        outputs = net(inputs)
        pred_label = outputs[0]
        pred_bbox = outputs[1]
        batchiou = torch.diagonal(torchvision.ops.
            complete_box_iou(bbox,pred_bbox),0).tolist()
        iou += batchiou
        _, predicted = torch.max(pred_label.data, 1)
        predicted = predicted.tolist()
        for label, prediction in zip(labels, predicted):
            Confusion_Matrix[label][prediction] += 1
    return Confusion_Matrix, mean(iou)


if __name__ == '__main__':
    if torch.cuda.is_available() == True:
        device = torch.device("cuda:0")
    else:
        device = torch.device("cpu")
    train_dataset = MyDataset(True)

    train_data_loader = torch.utils.data.DataLoader(
        dataset=train_dataset, batch_size=4, shuffle=False
        , num_workers=4)
    # train_data_loader = torch.utils.data.DataLoader(
        dataset=train_dataset, batch_size=4, shuffle=True,
         num_workers=4)
    # net = HW5Net(3,0)
    # loss_labeling, loss_regression = run_training_MSE(
        net,train_data_loader,"netMSE1.pth")
```

```
# num_layers = len(list(net.parameters()))
# print(num_layers)
# with open("test", "wb") as fp:
#     pickle.dump(loss_labeling, fp)
#     pickle.dump(loss_regression, fp)
# with open("test", "rb") as fp:
#     loss_labeling = pickle.load(fp)
#     loss_regression = pickle.load(fp)
# plt.figure()
# plt.plot(loss_labeling, label = "Labeling Training
    Loss")
# plt.savefig('loss_MSE_labeling.jpg')
# plt.figure()
# plt.plot(loss_regression, label = "Regression
    Training Loss")
# plt.savefig('loss_MSE_regression.jpg')


# net = HW5Net(3,0)
# loss_labeling, loss_regression = run_training_CIoU(
    net,train_data_loader,"netCIoU1.pth")
# with open("CIoU", "wb") as fp:
#     pickle.dump(loss_labeling, fp)
#     pickle.dump(loss_regression, fp)
# with open("CIoU", "rb") as fp:
#     loss_labeling = pickle.load(fp)
#     loss_regression = pickle.load(fp)
# plt.figure()
# plt.plot(loss_labeling, label="Labeling Training
    Loss")
# plt.savefig('loss_CIoU_labeling.jpg')
# plt.figure()
# plt.plot(loss_regression, label="Regression
    Training Loss")
# plt.savefig('loss_CIoU_regression.jpg')

# val_dataset = MyDataset(False)
# val_data_loader = torch.utils.data.DataLoader(
    dataset=val_dataset, batch_size=4, shuffle=True,
    num_workers=4)

netmse1 = torch.load('netMSE1.pth')
# cm1, mioumse1 = run_testing(netmse1,val_data_loader
    )
# plt.figure()
# sns.heatmap(cm1, annot=True, fmt="", cmap="Blues",
```

```
    cbar=False, xticklabels=['pizza','bus','cat'],
    yticklabels=['pizza','bus','cat'])
# plt.title("MSEnet1")
# plt.xlabel("Predicted")
# plt.ylabel("Ground Truth")
# plt.savefig("cm1.jpg")
# print(mioumse1)
# print(cm1.diag() / cm1.sum(1))
netciou1 = torch.load('netCIoU1.pth')
# cm2, mioumse2 = run_testing(netciou1,
    val_data_loader)
# plt.figure()
# sns.heatmap(cm2, annot=True, fmt="", cmap="Blues",
    cbar=False, xticklabels=['pizza','bus','cat'],
    yticklabels=['pizza','bus','cat'])
# plt.title("CIoUnet1")
# plt.xlabel("Predicted")
# plt.ylabel("Ground Truth")
# plt.savefig("cm2.jpg")
# print(mioumse2)
# print(cm2.diag() / cm2.sum(1))

for i, data in enumerate(train_data_loader):
    if i == 4:
        inputs, labels, bbox = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        bbox = bbox.to(device)
        outputs = netmse1(inputs)
        # pred_label = outputs[0]
        _, pred_label = torch.max(outputs[0], 1)
        pred_bbox = outputs[1]
        plt.figure()
        for j in range(3):
            ax = plt.subplot(1, 3, j + 1)
            image = inputs[j].numpy().transpose(1, 2,
                0).copy()
            image = cv2.rectangle(image, (int(bbox[j
                ][0] * 256), int(bbox[j][1] * 256)),(
                int(bbox[j][2] * 256), int(bbox[j][3]
                * 256)), (36, 255, 12),2)
            image = cv2.putText(image, str(labels[j])
                ,(int(bbox[j][0] * 256), int(bbox[j
                ][1] * 256 - 10)),cv2.
                FONT_HERSHEY_SIMPLEX, 0.8, (36, 255,
                12), 2)
```

```
            image = cv2.rectangle(image, (int(
                pred_bbox[j][0] * 256), int(pred_bbox[
                j][1] * 256)),(int(pred_bbox[j][2] *
                256), int(pred_bbox[j][3] * 256)),
                (36, 255, 12), 2)
            image = cv2.putText(image, str(pred_label
                [j]), (int(pred_bbox[j][0] * 256), int
                (pred_bbox[j][1] * 256 - 10)),cv2.
                FONT_HERSHEY_SIMPLEX, 0.8, (36, 255,
                12), 2)
            ax.imshow(image)
        plt.savefig('output.jpg')
    if i == 360:
        inputs, labels, bbox = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        bbox = bbox.to(device)
        outputs = netmse1(inputs)
        # pred_label = outputs[0]
        _, pred_label = torch.max(outputs[0], 1)
        pred_bbox = outputs[1]
        plt.figure()
        for j in range(3):
            ax = plt.subplot(1, 3, j + 1)
            image = inputs[j].numpy().transpose(1, 2,
                 0).copy()
            image = cv2.rectangle(image, (int(bbox[j
                ][0] * 256), int(bbox[j][1] * 256)),(
                int(bbox[j][2] * 256), int(bbox[j][3]
                * 256)), (36, 255, 12),2)
            image = cv2.putText(image, str(labels[j])
                ,(int(bbox[j][0] * 256), int(bbox[j
                ][1] * 256 - 10)),cv2.
                FONT_HERSHEY_SIMPLEX, 0.8, (36, 255,
                12), 2)
            image = cv2.rectangle(image, (int(
                pred_bbox[j][0] * 256), int(pred_bbox[
                j][1] * 256)),(int(pred_bbox[j][2] *
                256), int(pred_bbox[j][3] * 256)),
                (36, 255, 12), 2)
            image = cv2.putText(image, str(pred_label
                [j]), (int(pred_bbox[j][0] * 256), int
                (pred_bbox[j][1] * 256 - 10)),cv2.
                FONT_HERSHEY_SIMPLEX, 0.8, (36, 255,
                12), 2)
            ax.imshow(image)
```

```
        plt.savefig('output2.jpg')
    if i == 684:
        inputs, labels, bbox = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        bbox = bbox.to(device)
        outputs = netmse1(inputs)
        # pred_label = outputs[0]
        _, pred_label = torch.max(outputs[0], 1)
        pred_bbox = outputs[1]
        plt.figure()
        for j in range(3):
            ax = plt.subplot(1, 3, j + 1)
            image = inputs[j].numpy().transpose(1, 2,
                0).copy()
            image = cv2.rectangle(image, (int(bbox[j
                ][0] * 256), int(bbox[j][1] * 256)),(
                int(bbox[j][2] * 256), int(bbox[j][3]
                * 256)), (36, 255, 12),2)
            image = cv2.putText(image, str(labels[j])
                ,(int(bbox[j][0] * 256), int(bbox[j
                ][1] * 256 - 10)),cv2.
                FONT_HERSHEY_SIMPLEX, 0.8, (36, 255,
                12), 2)
            image = cv2.rectangle(image, (int(
                pred_bbox[j][0] * 256), int(pred_bbox[
                j][1] * 256)),(int(pred_bbox[j][2] *
                256), int(pred_bbox[j][3] * 256)),
                (36, 255, 12), 2)
            image = cv2.putText(image, str(pred_label
                [j]), (int(pred_bbox[j][0] * 256), int
                (pred_bbox[j][1] * 256 - 10)),cv2.
                FONT_HERSHEY_SIMPLEX, 0.8, (36, 255,
                12), 2)
            ax.imshow(image)
        plt.savefig('output3.jpg')
```