BME 646/ ECE695DL: Homework 5

Bianjiang Yang

8 Mar 2022

1 Introduction

The main goal for homework5:
1. To create a CNN that carries out both classification and regression at the same time.
2. To use a my own skip block class for the CNN.
3. To use COCO images and annotations include the classification labels and the bounding boxes for various types of objects in the images.

2 Methodology

Packages: torch, torch.nn, torch.nn.functional, torchvision.transforms, matplotlib.pyplot, copy, sklearn.metrics, os, sys, glob, seaborn, argparse, requests, PIL, pycocotools.coco, tqdm
Language: Python3
Tools: Anaconda3 virtual environment
System: Ubuntu 20.04.2 LTS (GNU/Linux 5.8.0-43-generic x86_64)

Instructions for running the code: The code I submitted has been modified to be runable, so you can simply run it by excuting the following script:
(1)train.py: CUDA_VISIBLE_DEVICES=1 python train.py
(2)validation.py: CUDA_VISIBLE_DEVICES=1 python -u validation.py

Other information: The dataloader will automatically skip the invalid bounding boxes, resize and save the resized images. You should modify the savefig directory or just use the plt.show() function to show the plots. For faster speed, I resized the images to 128×128

3 Implementation and Results

network.py

```
import torch
import torch.nn as nn
```

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

class Single_LOAD_Net(nn.Module):
    '''
    Single object detection and localization network
    It contains a deep feature extraction stream with a multi-scale style,
    it is consisted of 3 CNNs, 6 Residual Blocks,
    and 3 skip connections in the same scale;
    a classification stream with Conv and FC layers;
    a regression stream with Conv and FC layers,
    a ReLU is utilized at the end of the regression stream
    for non-negative output.
    '''
    def __init__(self):
        super(Single_LOAD_Net, self).__init__()
        # main stream
        fc_num = 8 * 8 * 256
        self.conv1_1 = nn.Sequential(nn.Conv2d(in_channels=3,
                                               out_channels=32,
                                               kernel_size=3,
                                               stride=2, padding=1),
                                     nn.BatchNorm2d(32),
                                     nn.LeakyReLU(0.2, True))
        self.conv1_2 = nn.Sequential(ResBlock(num_feat=32),
                                     ResBlock(num_feat=32))
        self.conv2_1 = nn.Sequential(nn.Conv2d(in_channels=32,
                                               out_channels=64,
                                               kernel_size=3,
                                               stride=2, padding=1),
                                       nn.BatchNorm2d(64),
                                       nn.LeakyReLU(0.2, True))
        self.conv2_2 = nn.Sequential(ResBlock(num_feat=64),
                                       ResBlock(num_feat=64))
        self.conv3_1 = nn.Sequential(nn.Conv2d(in_channels=64,
                                               out_channels=128,
                                               kernel_size=3,
                                               stride=2, padding=1),
                                       nn.BatchNorm2d(128),
                                       nn.LeakyReLU(0.2, True))
        self.conv3_2 = nn.Sequential(ResBlock(num_feat=128),
                                       ResBlock(num_feat=128))

        # classification stream
        self.classification = nn.Sequential(nn.Conv2d(in_channels=128,
                                                      out_channels=256,
```

```
                                                    kernel_size=3,
                                                    stride=2, padding=1),
                                            nn.LeakyReLU(0.2, True),
                                            nn.Flatten(),
                                            nn.Linear(fc_num, 100),
                                            nn.LeakyReLU(0.2, True),
                                            nn.Linear(100,10))

        # regression stream
        self.regression = nn.Sequential(nn.Conv2d(in_channels=128,
                                                  out_channels=256,
                                                  kernel_size=3,
                                                  stride=2, padding=1),
                                        nn.LeakyReLU(0.2, True),
                                        nn.Flatten(),
                                        nn.Linear(fc_num, 100),
                                        nn.LeakyReLU(0.2, True),
                                        nn.Linear(100,4),
                                        nn.ReLU())

    def forward(self, x):
        x = self.conv1_1(x)
        x = x + self.conv1_2(x)
        x = self.conv2_1(x)
        x = x + self.conv2_2(x)
        x = self.conv3_1(x)
        x = x + self.conv3_2(x)
        classify = self.classification(x)
        localize = self.regression(x)

        return classify, localize

class ResBlock(nn.Module):
    """Residual block with BN.

    It has a style of:
        ---Conv-BN-ReLU-Conv-BN+-ReLU
         |______________________|

    """

    def __init__(self, num_feat=64, res_scale=1, dilation=1):
        super().__init__()
        self.res_scale = res_scale
        padding = get_valid_padding(3, dilation)
```

```
        self.conv1 = nn.Conv2d(num_feat, num_feat, 3, 1, bias=True, \
                               padding=padding, dilation=dilation)
        self.bn1 = nn.BatchNorm2d(num_feat)
        self.conv2 = nn.Conv2d(num_feat, num_feat, 3, 1, bias=True, \
                               padding=padding, dilation=dilation)
        self.bn2 = nn.BatchNorm2d(num_feat)
        self.act = nn.LeakyReLU(0.2, True)

        default_init_weights([self.conv1, self.conv2], 0.1)

    def forward(self, x):
        identity = x
        out = self.bn2(self.conv2(self.act(self.bn1(self.conv1(x)))))
        return identity + out * self.res_scale

@torch.no_grad()
def default_init_weights(module_list, scale=1, bias_fill=0, **kwargs):
    """Initialize network weights with kaiming_normal.
    """
    if not isinstance(module_list, list):
        module_list = [module_list]
    for module in module_list:
        for m in module.modules():
            if isinstance(m, nn.Conv2d):
                init.kaiming_normal_(m.weight, **kwargs)
                m.weight.data *= scale
                if m.bias is not None:
                    m.bias.data.fill_(bias_fill)
            elif isinstance(m, nn.Linear):
                init.kaiming_normal_(m.weight, **kwargs)
                m.weight.data *= scale
                if m.bias is not None:
                    m.bias.data.fill_(bias_fill)
            elif isinstance(m, _BatchNorm):
                init.constant_(m.weight, 1)
                if m.bias is not None:
                    m.bias.data.fill_(bias_fill)

def get_valid_padding(kernel_size, dilation):
    # get valid padding number
    kernel_size = kernel_size + (kernel_size - 1) * (dilation - 1)
    padding = (kernel_size - 1) // 2
    return padding
```

dataloader.py

```
import torch
import glob
import os
from PIL import Image
import numpy as np
from pycocotools.coco import COCO

class Hw05_Coco_Dataset(torch.utils.data.Dataset):
  'Characterizes a dataset for PyTorch'
  def __init__(self, datapath, transform, resize_w,
               resize_h, jsonpath, resized_data_path):
        'Initialization'
        self.skipnumber = 0
        self.transform = transform
        path_list = glob.glob(datapath + '/*')
        coco = COCO(jsonpath)
        class_list = [os.path.splitext(os.path.basename(p))[0] for p in path_list]
        self.list_IDs = []
        self.labels = []
        self.bbox = []
        n_class = len(path_list)

        # for each class
        for p in range(n_class):
            img_path_list = sorted(glob.glob(path_list[p] + '/*'))
            catIds = coco.getCatIds(catNms=class_list[p])
            imgIds = coco.getImgIds(catIds=catIds)

            # for each sample
            for k in range(len(img_path_list)):

                # get annotions for bbox
                annIds = coco.getAnnIds(imgIds=imgIds[k], catIds=catIds, iscrowd=False)
                annotions = coco.loadAnns(annIds)
                Max = 0
                Max_index = 0
                for j in range(len(annotions)):
                    ann_j = annotions[j]
                    bbox = ann_j['bbox'] # bbox = [x,y,w,h]
                    w, h = bbox[2], bbox[3]
                    Area = w * h
                    if Area > Max:
                        Max = Area
```

```
                Max_index = j
            annotion = annotions[Max_index]
            boundingbox = annotion['bbox']

            # get annotions for imgs
            img_id = annotion['image_id']
            img_id_str = str(img_id)
            img_id_str = img_id_str.zfill(12)
            img_path = os.path.join(datapath, class_list[p],
                                    'COCO_val2014_' + img_id_str + ".jpg")
            img_sample = Image.open(img_path)
            w, h = img_sample.size

            # skip small boundbox samples
            if boundingbox[2]< w/3 or boundingbox[3]< w/3:
                self.skipnumber += 1
                print("skip", self.skipnumber)
                continue

            # save resized images
            resized_save_path = os.path.join(resized_data_path,
                                             class_list[p], img_id_str + ".jpg")
            if not os.path.exists(resized_save_path):
                img_resize = img_sample.resize((resize_h, resize_w), Image.BOX)
                img_resize.save(resized_save_path)

            # rescale boundingbox
            boundingbox[0] = boundingbox[0] * resize_w / w
            boundingbox[1] = boundingbox[1] * resize_h / h
            boundingbox[2] = boundingbox[2] * resize_w / w
            boundingbox[3] = boundingbox[3] * resize_h / h

            # append list_ids, labels, and bbox for getitem function
            self.list_IDs.append(resized_save_path)
            self.labels.append(p)
            self.bbox.append(boundingbox)

def __len__(self):
      'Denotes the total number of samples'
      return len(self.list_IDs)

def __getitem__(self, index):
      'Generates one sample of data'
      # Select sample
      img_path = self.list_IDs[index]
      label = self.labels[index]
```

```
        bbox = self.bbox[index]

        # Load data
        img = Image.open(img_path)
        if img.mode!='RGB':
            img = img.convert('RGB')
        img = self.transform(img).to(dtype = torch.float32)

        return img, label, bbox, img_path
```

hw05_coco_downloader.py

```
# required argparse arguments
parser = argparse.ArgumentParser(description = 'HW05 COCO downloader')
parser.add_argument('--root_path', required = True, type = str)
parser.add_argument('--coco_json_path', required = True, type = str)
parser.add_argument('--class_list', required = True, nargs='*', type=str,)
parser.add_argument('--images_per_class', required=True, type=int)
args, args_other = parser.parse_known_args()

def Coco_Downloader(args):
    coco = COCO(args.coco_json_path)
    urls = dict.fromkeys(args.class_list)
    folder_dict = dict.fromkeys(args.class_list)

    for c in args.class_list:
        if not os.path.exists(args.root_path + c):
            os.makedirs(args.root_path + c)
        folder_dict[c] = args.root_path + c
        catIds = coco.getCatIds(c)
        imgIds = coco.getImgIds(catIds=catIds)
        imgs = coco.loadImgs(imgIds)
        urls[c] = [i['coco_url'] for i in imgs]

    for c in args.class_list:
        folder = folder_dict[c]
        url = urls[c]
        print("Downloading " + c + " class")
        for i in tqdm(range(args.images_per_class)):
            per_url = url[i]
            img_name = per_url.split('/')[-1]
```

```
            file_path = os.path.join(folder, img_name)

            if os.path.exists(file_path):
                # print("File already exists: " + per_url + "\n " +
                #       "Will skip it and continue to the next one.")
                continue

            try: response = requests.get(per_url, timeout=1)
            except Exception:
                try: response = requests.get(per_url, timeout=1) # try again
                except Exception:
                    print("Tried twice and still no response for: " + per_url + "\n " +
                          "Will skip it and continue to the next one.")
                    continue


            with open(file_path, 'wb') as im:
                im.write(response.content)

            img = Image.open(file_path)
            # img_resize = img.resize((64, 64), Image.BOX)
            # img_resize.save(file_path)
            img.save(file_path)
        print("Class "+ c +" Finished!")

Coco_Downloader(args)
```

train.py

```
import torch
import torch.nn as nn
import network
import torchvision.transforms as tvt
import matplotlib.pyplot as plt
import dataloader
import copy

def train(transform, device, lr = 1e-3, momentum = 0.9, epochs = 10,
          batch_size = 10, data_path = "/home/yangbj/695/YANG/hw5/COCO_Download/Train",
          save_path = "/home/yangbj/695/hw5/",
          cocodatapath = "/home/yangbj/695/YANG/hw4/annotations/instances_train2017.json"):
```

```
coco_data = dataloader.hw5_dataloader(datapath=data_path, cocodatapath=cocodatapath,
                                  transform=transform, resize_w=128, resize_h=128)
train_data = torch.utils.data.DataLoader(coco_data, batch_size=batch_size,
                                         shuffle=True, num_workers=2)
net = network.Single_LOAD_Net()
net = copy.deepcopy(net)
# net.load_state_dict(torch.load("/home/yangbj/695/hw5/net50.pth"))
net = net.to(device)
running_loss1 = []
running_loss2 = []
loss_item1 = 0
loss_item2 = 0
criterion1 = nn.CrossEntropyLoss()
criterion2 = nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=lr, betas=(0.9, 0.999),
                             eps=1e-08, weight_decay=0, amsgrad=False)
# optimizer = torch.optim.SGD(net.parameters(), lr=lr, momentum=momentum)
print("\n\nStarting training...")
for epoch in range(epochs):
    print("")
    running_loss = 0.0
    for i, data in enumerate(train_data):
        inputs, labels, bbox = data
        inputs = inputs.to(device).float()
        labels = labels.to(device)
        bbox = torch.transpose(torch.stack(bbox), 0, 1)
        bbox = bbox.to(torch.float32).to(device)
        optimizer.zero_grad()
        classify, boundingbox = net(inputs)
        loss1 = criterion1(classify, labels)
        # print("classify: ",classify)
        # print("labels: ", labels)
        # print("loss1: ",loss1)
        loss1.backward(retain_graph=True)
        loss2  = criterion2(boundingbox, bbox)
        # print("boundingbox: ", boundingbox)
        # print("bbox: ", bbox)
        # print("loss2: ", loss2)
        loss2.backward()
        optimizer.step()
        loss_item1 += loss1.item()
        loss_item2 += loss2.item()
        if (i+1) % 500 == 0:
            print("\n[epoch:%d, batch:%5d] loss: %.3f" %
                  (epoch + 1, i + 1, loss_item1 / float(500)))
            print("\n[epoch:%d, batch:%5d] loss: %.3f" %
```

```
                    (epoch + 1, i + 1, loss_item2 / float(500)))
                running_loss1.append(loss_item1/float(500))
                running_loss2.append(loss_item2 / float(500))
                loss_item1, loss_item2 = 0.0, 0.0
    torch.save(net.state_dict(), save_path+'net50lr1e-4_wo_softmax.pth')

    return running_loss1, running_loss2


if __name__ == '__main__':
    device  = torch.device('cuda:1')
    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    trainset_path = "/home/yangbj/695/YANG/hw5/COCO_Download/Train"
    coco_json = "/home/yangbj/695/YANG/hw4/annotations/instances_train2017.json"
    save_path = "/home/yangbj/695/hw5/"
    running_loss1, running_loss2 = train(transform = transform, device = device,
    lr = 1e-4, momentum = 0.9,epochs = 50, batch_size = 10,
    data_path = trainset_path, save_path = save_path,
    cocodatapath = "/home/yangbj/695/YANG/hw4/annotations/instances_train2017.json")

    plt.figure()
    plt.title('Train Loss1')
    plt.xlabel('Per 500 Iterations')
    plt.ylabel('Loss')
    plt.plot(running_loss1, label = 'Loss1')
    # plt.plot(running_loss2, label = 'Loss2')
    plt.legend(loc='upper right')
    plt.show()
    plt.savefig(save_path + "train_loss1_1lr1e-4_wo_softmax" + ".jpg")
    plt.figure()
    plt.title('Train Loss2')
    plt.xlabel('Per 500 Iterations')
    plt.ylabel('Loss')
    # plt.plot(running_loss1, label='Loss1')
    plt.plot(running_loss2, label='Loss2')
    plt.legend(loc='upper right')
    plt.show()
    plt.savefig(save_path + "train_loss2_1lr1e-4_wo_softmax" + ".jpg")
```

validation.py

```
import torch
```

```
import torch.nn as nn
import network
import torchvision.transforms as tvt
import matplotlib.pyplot as plt
import testdataloader, testdata_loader
import copy
import sklearn.metrics
import os
import sys
import glob
import seaborn
import cv2

def test(name, net, transform, device,
         batch_size, data_path, save_path,
         jsonpath, resized_data_path, resize):

    coco_data = testdata_loader.Hw05_Coco_Dataset\
        (datapath = data_path, transform = transform,
         resize_w = resize[0], resize_h = resize[1],
         jsonpath = jsonpath, resized_data_path = resized_data_path)

    test_data = torch.utils.data.DataLoader(coco_data,
                    batch_size=batch_size, shuffle=False, num_workers=2)
    print("test data len: ",len(test_data))

    net = copy.deepcopy(net)
    net.load_state_dict(torch.load('/home/yangbj/695/hw5/net50lr1e-4_wo_softmax.pth'))
    net = net.to(device)
    net.eval()

    print("\n\nStarting testing...")
    output_total = []
    label_total = []

    for i, data in enumerate(test_data):
        inputs, labels, bbox, img_path = data
        inputs = inputs.to(device).float()
        labels = labels.to(device)
        bbox = torch.transpose(torch.stack(bbox), 0, 1)
        bbox = bbox.int().numpy()#.to(torch.float32)#.to(device)
        classify, boundingbox = net(inputs)
        boundingbox = boundingbox.cpu().int().numpy()
        prediction = [torch.argmax(output).cpu() for output in classify]
        output_total = output_total + prediction
        labels = [label.cpu() for label in labels]
```

```
            label_total = label_total + labels

            for i in range(inputs.shape[0]):
                img = cv2.imread(img_path[i])
                cv2.rectangle(img, (bbox[i][0], bbox[i][1]),
                              (bbox[i][0]+bbox[i][2], bbox[i][1]+bbox[i][3]),
                              (0, 0, 255), 1)
                cv2.rectangle(img, (boundingbox[i][0], boundingbox[i][1]),
                              (boundingbox[i][0]+boundingbox[i][2],
                               boundingbox[i][1]+boundingbox[i][3]), (0, 255, 0), 1)
                if not os.path.exists(save_path+'tested_imgs/'+
                                      str(labels[i]) +'/'):
                    os.makedirs(save_path+'tested_imgs/'+
                                str(labels[i]) +'/')
                cv2.imwrite(save_path+'tested_imgs/'+
                            str(labels[i]) +'/'+img_path[i][-16:], img)
                print("save an img to:" + save_path+
                      'tested_imgs/'+str(labels[i]) +'/'+img_path[i][-16:])

        # calculate confusion matrix with sklearn module
        confus_matrix = sklearn.metrics.confusion_matrix(
            label_total, output_total, labels=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
        print(confus_matrix)

        # classification accuracy calculation
        acc = 0
        for i in range(confus_matrix.shape[0]):
            acc += confus_matrix[i][i]
        Accuracy = acc / confus_matrix.sum() * 100
        print('Accuracy:'+str(Accuracy)+'%')

        # plot confusion matrix
        plt_labels = []
        path_list = sorted(glob.glob(data_path + '/*'))
        for p in path_list:
            plt_labels.append(os.path.splitext(os.path.basename(p))[0])
        plt.figure(figsize = (10,7))
        seaborn.heatmap(confus_matrix, annot=True, fmt= 'd', linewidths = .5,
                        xticklabels= plt_labels, yticklabels= plt_labels)
        plt.title("Net" + name + " " + 'Accuracy:'+str(Accuracy)+'%')
        plt.savefig(save_path + "net_"+name+"confusion_matrix.jpg")



if __name__ == '__main__':
```

```
# settings
device  = torch.device('cuda:2')
transform = tvt.Compose([tvt.ToTensor(),
                         tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
dataset_path = "/home/yangbj/695/YANG/hw5/COCO_Download/Val"
save_path = "/home/yangbj/695/hw5/"
jsonpath = "/home/yangbj/695/YANG/hw4/annotations/instances_val2014.json"
resized_data_path = "/home/yangbj/695/YANG/hw5/hw5_coco_data/Val/"
batch_size = 1
name = '1'
resize = [128, 128]
net = network.Single_LOAD_Net()

# Run test
test(name = name,  net= net,
     transform = transform, device = device, batch_size = batch_size,
     data_path = dataset_path, save_path = save_path,
     jsonpath = jsonpath,
     resized_data_path = resized_data_path, resize = resize)
```

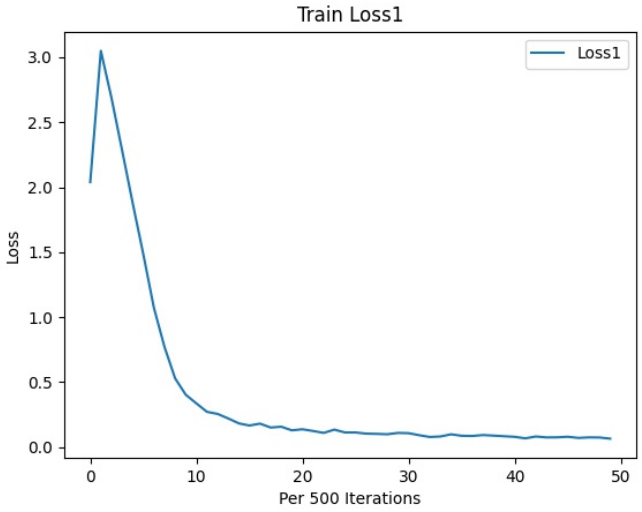

Figure 1: classification train loss–crossentropy loss

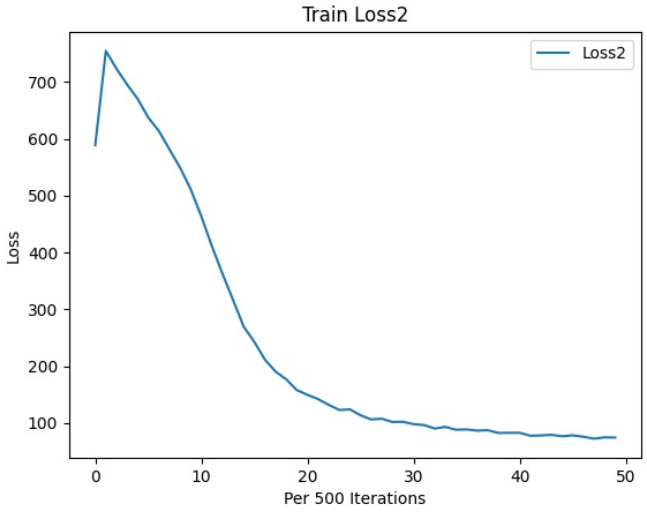

Figure 2: regression loss–MSE loss

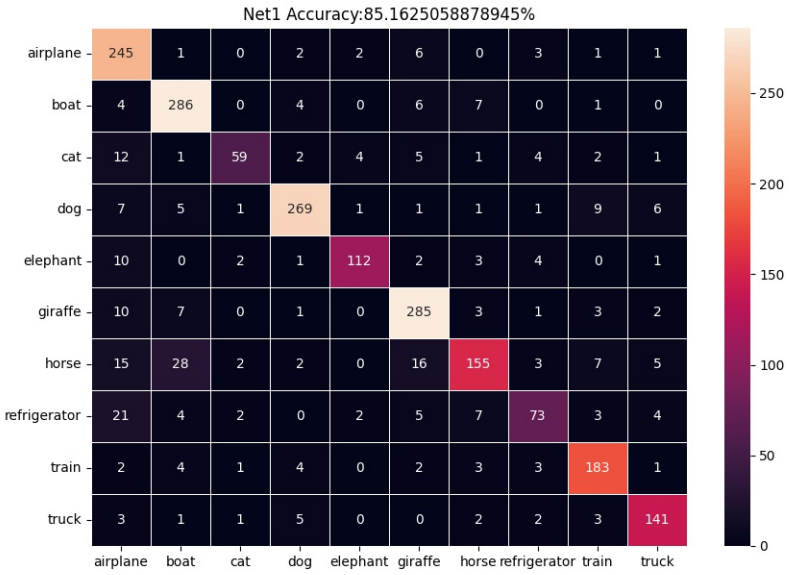

Figure 3: net: confusion matrix

Train Loss

The results in Fig. 1 and Fig. 2 above show that:
(1) Crossentropy loss typcially is smaller than MSE Loss
(2) After 40×500 iterations, the network converged.

Confusion Matrix

The results in Fig. 4 above shows that:
(1) The network has a classification accuracy of 85%, which indicates most of the prediction is correct.;
(2) The numbers of test samples for each categories are not the same because I choose to skip the image if the bounding box is less than 1/3 of the image;
(3) 'refrigerator' class may be misclassified to 'airplane'. 'horse' sometimes is classified to 'boat', which may be reasonable because refrigerator and airplane share some common features so they are hard to be distinguished to some extent.

Framework

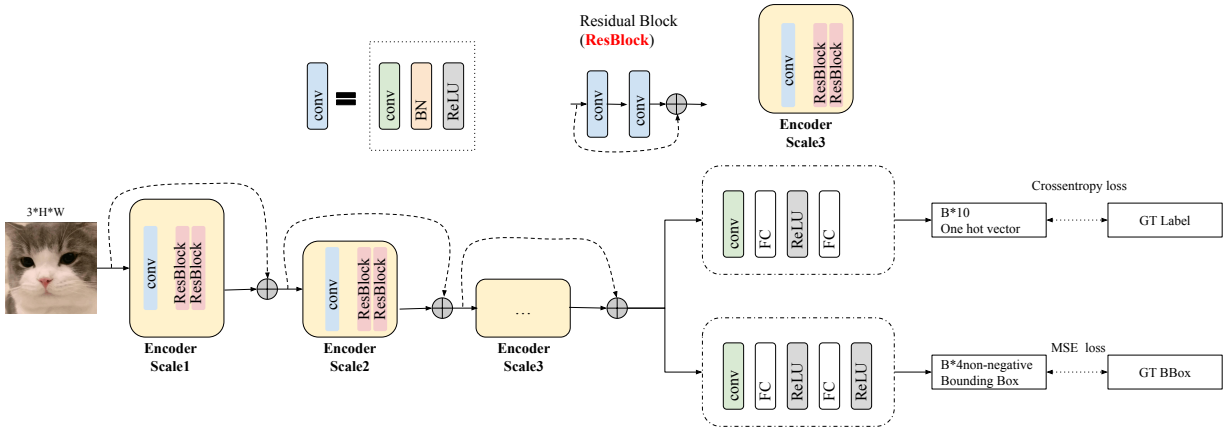

Figure 4: The framework: single object detection and localization network (Single_LOAD_Net). It contains a deep feature extraction stream with a multi-scale style, which is consisted of 3 CNNs, 6 Residual Blocks, and 3 skip connections in the same scale; a classification stream with Conv and FC layers; a regression stream with Conv and FC layers, a ReLU is utilized at the end of the regression stream for non-negative output.

Bounding Box Samples: As shown in Fig. 5, the green bounding box is the prediction; the red bounding box is the true label.

I selected different types of predictions including successful predictions and failed predictions. For the reasonable failure predictions, such as the third sample in 'train'; the fifth sample in 'elephant'; the forth sample in 'cat'; all of them are interfered by other similar or same objects.
Our network can handle both gray scale images and RGB inputs.
Our network can handle some difficult samples such as the fifth in 'boat', which is even hard to a human.

Implementation Details. Batch size=10; I choose Leaky ReLU with negative slop 0.2 except for the last activation function in the regression stream. I choose Adam optimizer with learning rate 10^{-4} for more stable training because SGD sometimes failed for such learning rate with loss nan issue. I initialize the convolution layers of my Residual Blocks with Kaiming Normal.

4 Lessons Learned

The hurdles faced and the techniques employed to overcome them:

(1) For dataloader,

We need to output the label with shape (1,) for one sample, which means the labels are int numbers ranging (0,9). However, the outputs of the network are one-hot-vectors with shape(10,). nn.CrossEntropyLoss() can accept such two inputs.

(2) For training,

Backward: $retain\_graph = True$ is needed the first loss backward;

(3) For validation,

We need to eval the model to keep the model parameters static.
seaborn is a better tool than sklearn to plot the heatmap.
Confusion Matrix should be computed on CPU because of sklearn
bounding box should be int value because of cv2.rectangle

5 Suggested Enhancements

To the best of my understanding, this task emphasize more on downloader because it takes more time to accomplish. Also, most of the time was consumed in the ploting and some unimportant part debugging. So I suggest you could give a colab file and let the students fill necessary blanks.

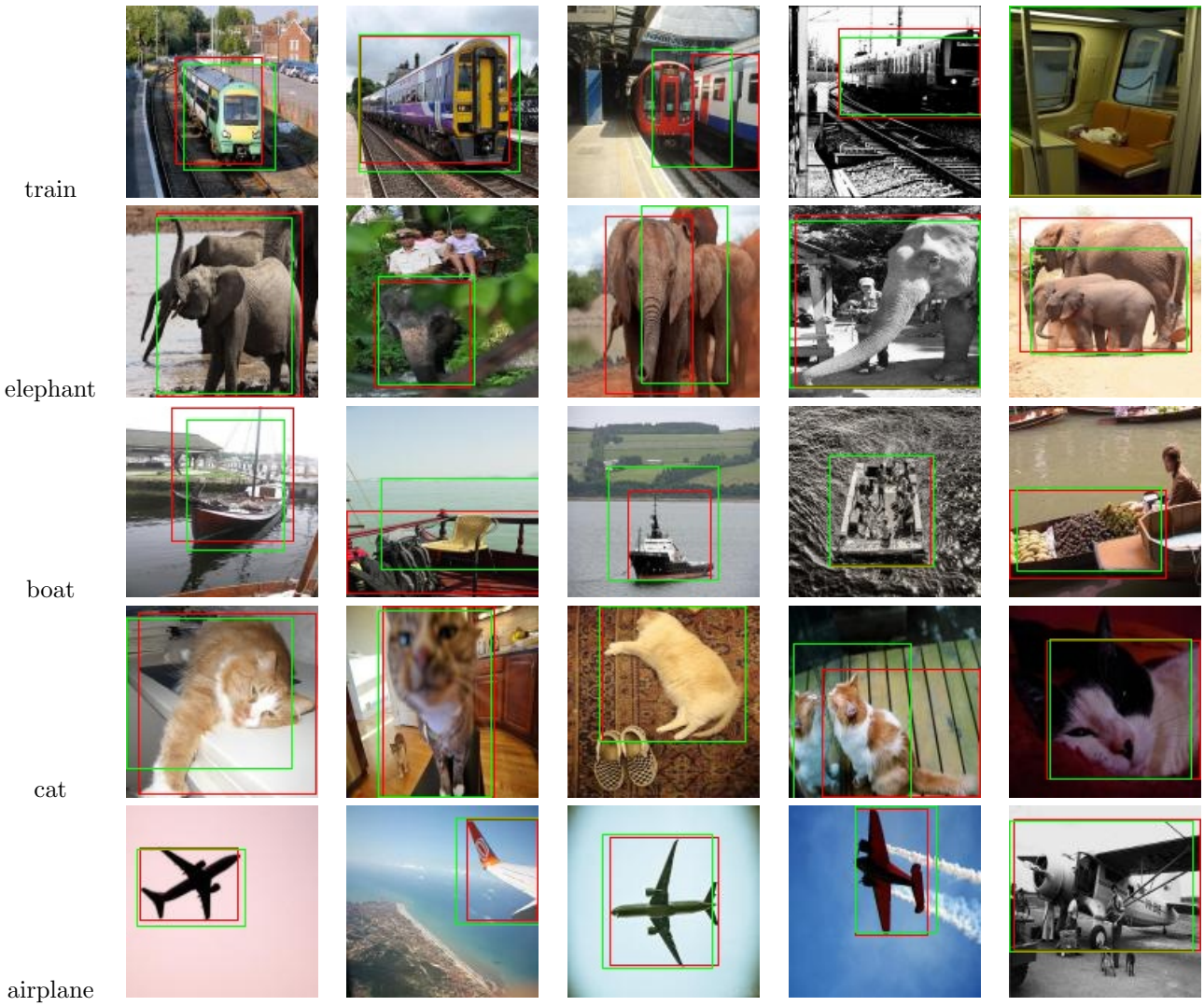

Figure 5: Testing samples