hw4_ChengjunGuo

Chengjun Guo

February 2023

COCO dataset

For this homework, we only need 5 classes of the dataset. I didn't directly downloaded the full dataset of the images. I used an API from coco: coco.download. This function can download the images with image IDs to specific path. The coco API getcatIds and getImgIds are used to extract the category IDs and Images IDs.

Outputs

The images of dataset:

Figure 1: airplane

Figure 2: bus

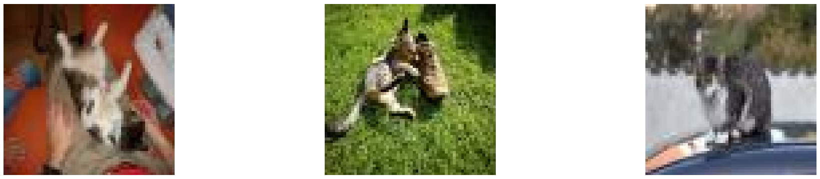


Figure 3: cat

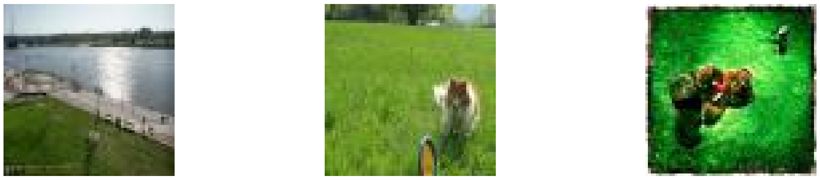


Figure 4: dog

Figure 5: pizza

Training loss plot: The green line is the Net3, the orange line is Net2 and

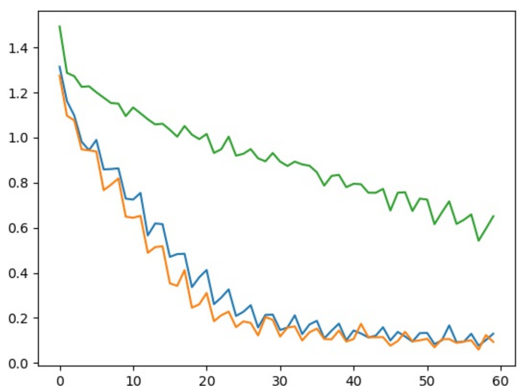


Figure 6: pizza

the blue line is Net1.

The confusion matrix:

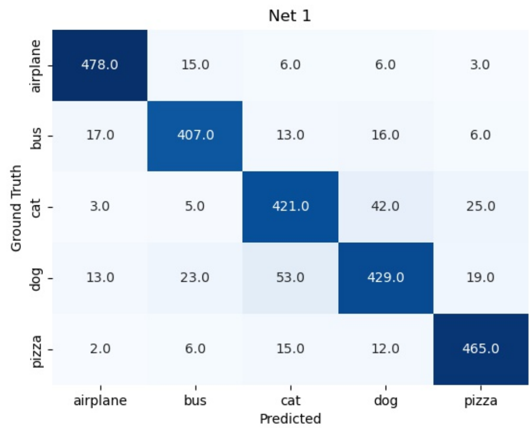


Figure 7: Net1

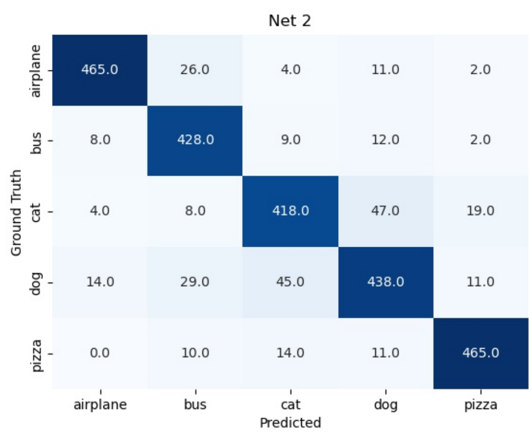
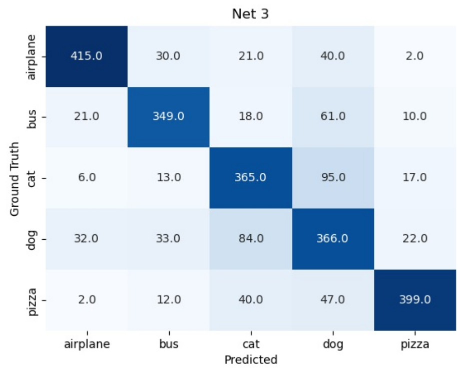
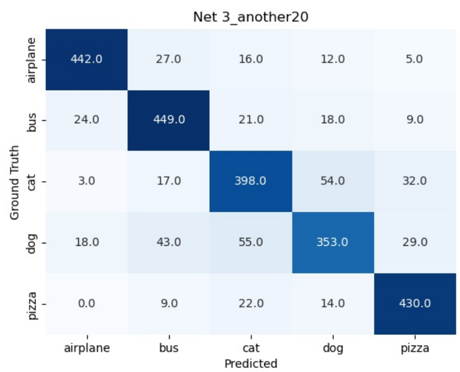


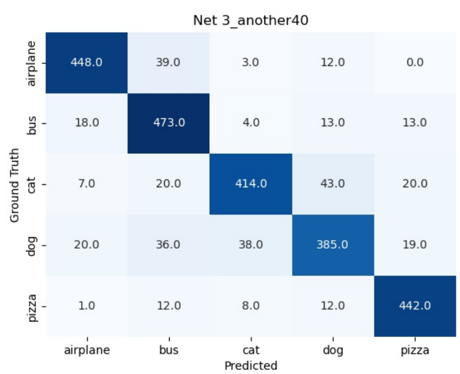
Figure 8: Net2



(a) 20 epochs



(b) 40 epochs



(c) 60 epochs

Figure 9: Net3

Questions

Does adding padding to the convolutional layers make a difference in classification performance?

Yes, the Net2 is the padding version of Net1. It converges faster. It also have better performance than Net1.

As you may have known, naively chaining a large number of layers can result in difficulties in training. This phenomenon is often referred to as vanishing gradient. Do you observe something like that in Net3 ?

The Net3 converges much slower than the previous two networks. The train loss decreases slower than the other two networks.

Compare the classification results by all three networks, which CNN do you think is the best performer?

I think Net2 have the best performance based on the current performance. The average accuracy of Net1 is 0.8780, Net2 accuracy is 0.8826, Net3 accuracy after 60 epoch is 0.8651.

By observing your confusion matrices, which class or classes do you think are more difficult to correctly differentiate and why?

I think dog is the most difficult to correctly differentiate. I think the shape of the dog is mostly confused with cat. It is reasonable because the size and shape of the edges looks alike among all the other items.

What is one thing that you propose to make the classification performance better?

I normalized the images to make the performance better. Preprocess of the dataset is also important to the training.

Code

coco downloader:

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

class Coco_Downloader():
    def __init__(self):
        self.root_path = 'E:\ECE60146DL\hw4_new\data/'
        self.coco_json_path = 'annotations_trainval2014/
            annotations/instances_train2014.json'
        self.class_list = ['airplane','bus','cat','dog','
            pizza']
        self.images_per_class = 2000    # 1500 for train,
             500 for validation
        self.coco = COCO(self.coco_json_path)

    def save_images(self):
        for i in self.class_list:
            if not os.path.exists(self.root_path + i):
                os.makedirs(self.root_path + i)
    #download images
            catIds = self.coco.getCatIds(catNms=[i])
            img_ids = self.coco.getImgIds(catIds = catIds
                )
            #download   - Download COCO images from
                mscoco.org server.
            self.coco.download(tarDir = self.root_path +
                i, imgIds = img_ids[:self.images_per_class
                ])
            #check if enough image
            x = 0
            while True:
                path, dirs, files = next(os.walk(self.
                    root_path + i))
                if len(files) == self.images_per_class:
                    break
                self.coco.download(tarDir = self.
                    root_path + i, imgIds = img_ids[len(
                    files):(2*self.images_per_class-len(
                    files))])
                if x > 100:
                    raise Exception("Too many iterations
                        ")
            #resize
```

```
            path, dirs, files = next(os.walk(self.
                root_path + i))
            #print(path,dirs,files)
            for file in files:
                im = Image.open(os.path.join(path,file))
                im_resized = im.resize((64, 64), Image.
                    Resampling.BOX)
                im_resized.save(os.path.join(path,file))
                print(file+" resized!")

if __name__ == '__main__':
    CocoDownloader = Coco_Downloader()
    CocoDownloader.save_images()
```

net:

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


class HW4Net1(nn.Module):
    def __init__(self):
        super(HW4Net1, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)  # 62x62
        self.pool = nn.MaxPool2d(2,2)   # 31x31
        self.conv2 = nn.Conv2d(16, 32, 3)   # 29x29
        self.fc1 = nn.Linear(32*14*14,64)
        self.fc2 = nn.Linear(64, 5)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

class HW4Net2(nn.Module):
    def __init__(self):
        super(HW4Net2, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding="same")
```

```
                # 64x64
        self.pool = nn.MaxPool2d(2,2)         # 32x32
        self.conv2 = nn.Conv2d(16, 32, 3, padding="same")
                # 32x32
        self.fc1 = nn.Linear(32*16*16,64)
        self.fc2 = nn.Linear(64, 5)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

class HW4Net3(nn.Module):
    def __init__(self):
        super(HW4Net3, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.pool = nn.MaxPool2d(2,2)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.convd1 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd2 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd3 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd4 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd5 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd6 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd7 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd8 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd9 = nn.Conv2d(32, 32, 3, padding=1)
        self.convd10 = nn.Conv2d(32, 32, 3, padding=1)
        self.fc1 = nn.Linear(32*8*8,64)
        self.fc2 = nn.Linear(64, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(F.relu(self.conv2(x)))
        x = F.relu(self.convd1(x))
        x = F.relu(self.convd2(x))
        x = F.relu(self.convd3(x))
        x = F.relu(self.convd4(x))
        x = self.pool(F.relu(self.convd5(x)))
        x = F.relu(self.convd6(x))
        x = F.relu(self.convd7(x))
        x = F.relu(self.convd8(x))
        x = F.relu(self.convd9(x))
```

```
        x = self.pool(F.relu(self.convd10(x)))
        x = x.view(x.shape[0],-1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

train:

```
import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
from hw4_net import HW4Net1
from hw4_net import HW4Net2
from hw4_net import HW4Net3
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



class MyDataset(torch.utils.data.Dataset):
    def __init__(self):
        super().__init__()
        self.root_path = 'E:\ECE60146DL\hw4_new\data/'
        self.coco_json_path = 'annotations_trainval2014/
            annotations/instances_train2014.json'
        self.class_list = ['airplane','bus','cat','dog','
            pizza']
        self.images_per_class = 2000    # 1500 for train,
             500 for validation
        self.coco = COCO(self.coco_json_path)
        self.img_labels = []
        self.transform = tvt.Compose([tvt.ToTensor(),tvt.
            Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])])
        for cat in self.class_list:
```

```
            path, dirs, files = next(os.walk(self.
                root_path + cat))
            for file in files:
                # first image path, second label
                self.img_labels.append([cat + '/' + file,
                     self.class_list.index(cat)])


    def __len__(self):
        return int(self.images_per_class * len(self.
            class_list))

    def __getitem__(self, index):
        img_path = os.path.join(self.root_path, self.
            img_labels[index][0])
        image = Image.open(img_path).convert("RGB")
        label = torch.tensor(self.img_labels[index][1])

        image = self.transform(image)
        return image, label


def run_training(net,train_data_loader,net_save_path):
    net = copy.deepcopy(net)
    net = net.to(device)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(net.parameters(), lr=1e
        -3, betas=(0.9, 0.99))
    epochs = 20
    Loss_runtime = []
    for epoch in range(epochs):
        running_loss = 0.0
        for i,data in enumerate(train_data_loader):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if (i+1) % 500 == 0:
                print("[epoch: %d, batch: %5d] loss: %.3f
                    " % (epoch+1, i+1, running_loss/500))
                Loss_runtime.append(running_loss/500)
```

```
            running_loss = 0.0
    torch.save(net, net_save_path)
    return Loss_runtime


def run_testing(net, validation_data_loader):
    net = copy.deepcopy(net)
    net = net.to(device)
    Confusion_Matrix = torch.zeros(5, 5)
    for i, data in enumerate(validation_data_loader):
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = net(inputs)
        _, predicted = torch.max(outputs.data, 1)
        predicted = predicted.tolist()
        for label, prediction in zip(labels, predicted):
            Confusion_Matrix[label][prediction] += 1
    return Confusion_Matrix


if torch.cuda.is_available() == True:
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")
my_dataset = MyDataset()
train_dataset, test_dataset = torch.utils.data.
   random_split(my_dataset, [7500, 2500])

## train
train_data_loader = torch.utils.data.DataLoader(dataset=
   train_dataset, batch_size=4, shuffle=True, num_workers
   =0)
# net = HW4Net1()
# loss1 = run_training(net,train_data_loader,"net1.pth")
# net = HW4Net2()
# loss2 = run_training(net,train_data_loader,"net2.pth")
# net = HW4Net3()
# loss3 = run_training(net,train_data_loader,"net3.pth")
# plt.figure()
# plt.plot(loss1, label = "Net1 Training Loss")
# plt.plot(loss2, label = "Net2 Training Loss")
# plt.plot(loss3, label = "Net3 Training Loss")
# plt.savefig('123.jpg')
```

```
# validate
test_data_loader = torch.utils.data.DataLoader(dataset=
    test_dataset, batch_size=4, shuffle=True, num_workers
    =0)
model1 = torch.load('net1.pth').eval()
cm1 = run_testing(model1, test_data_loader)
plt.figure()
sns.heatmap(cm1,annot = True, fmt = "", cmap = "Blues",
    cbar = False, xticklabels = ['airplane','bus','cat','
    dog','pizza'], yticklabels = ['airplane','bus','cat','
    dog','pizza'])
plt.title("Net 1")
plt.xlabel("Predicted")
plt.ylabel("Ground Truth")
plt.savefig("cm1.jpg")
model2 = torch.load('net2.pth').eval()
cm2 = run_testing(model2, test_data_loader)
plt.figure()
sns.heatmap(cm2,annot = True, fmt = "", cmap = "Blues",
    cbar = False, xticklabels = ['airplane','bus','cat','
    dog','pizza'], yticklabels = ['airplane','bus','cat','
    dog','pizza'])
plt.title("Net 2")
plt.xlabel("Predicted")
plt.ylabel("Ground Truth")
plt.savefig("cm2.jpg")
model3 = torch.load('net3.pth').eval()
cm3 = run_testing(model3, test_data_loader)
plt.figure()
sns.heatmap(cm3,annot = True, fmt = "", cmap = "Blues",
    cbar = False, xticklabels = ['airplane','bus','cat','
    dog','pizza'], yticklabels = ['airplane','bus','cat','
    dog','pizza'])
plt.title("Net 3")
plt.xlabel("Predicted")
plt.ylabel("Ground Truth")
plt.savefig("cm3.jpg")

# another 20 epochs for net3 since it's too deep and
    converge slow
# model4 = torch.load('net3.pth')
# loss4 = run_training(model4,train_data_loader,"net4.pth
    ")
model4 = torch.load('net4.pth')
cm4 = run_testing(model4, test_data_loader)
plt.figure()
```

```
sns.heatmap(cm4,annot = True, fmt = "", cmap = "Blues",
    cbar = False, xticklabels = ['airplane','bus','cat','
    dog','pizza'], yticklabels = ['airplane','bus','cat','
    dog','pizza'])
plt.title("Net 3_another20")
plt.xlabel("Predicted")
plt.ylabel("Ground Truth")
plt.savefig("cm4.jpg")


model5 = torch.load('net4.pth')
loss5 = run_training(model5,train_data_loader,"net5.pth")
model5 = torch.load('net5.pth')
cm5 = run_testing(model5, test_data_loader)
plt.figure()
sns.heatmap(cm5,annot = True, fmt = "", cmap = "Blues",
    cbar = False, xticklabels = ['airplane','bus','cat','
    dog','pizza'], yticklabels = ['airplane','bus','cat','
    dog','pizza'])
plt.title("Net 3_another40")
plt.xlabel("Predicted")
plt.ylabel("Ground Truth")
plt.savefig("cm5.jpg")

print(cm1.diag()/cm1.sum(1))
print(cm2.diag()/cm2.sum(1))
print(cm3.diag()/cm3.sum(1))
print(cm4.diag()/cm4.sum(1))
print(cm5.diag()/cm5.sum(1))
x1 = cm1.diag()/cm1.sum(1)
x2 = cm2.diag()/cm2.sum(1)
x3 = cm3.diag()/cm3.sum(1)
x4 = cm4.diag()/cm4.sum(1)
x5 = cm5.diag()/cm5.sum(1)
print(x1.sum(1)/5)
print(x2.sum(1)/5)
print(x3.sum(1)/5)
print(x4.sum(1)/5)
print(x5.sum(1)/5)
```