BME 646/ ECE695DL: Homework 4

Varun Aggarwal

February 21, 2022

1 Introduction

This project was aimed at implementing dataloader and CNN for training a classifier on MSCOCO dataset.

2 Methodology

The three tasks in this assignment were tackled as follows:

Task 1

Four files were implemented for training and validating a CNN classifier. *dataloader.py* was used to load train and validation data. *model.py* for three CNN networks. *hw04_training* for training the model. *hw04_testing* for validating the trained network. Training took the longest for CNN with single convolution layer (Figure 1)..

Overall, CNN network with two convolution layer and no padding lead to the highest accuracy. Three confusion matrix were also generated for each of the three networks (Figure 2).

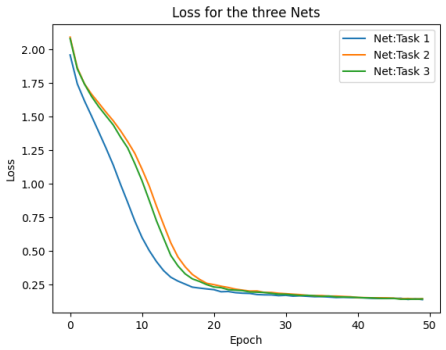

Figure 1: Training loss for three CNN

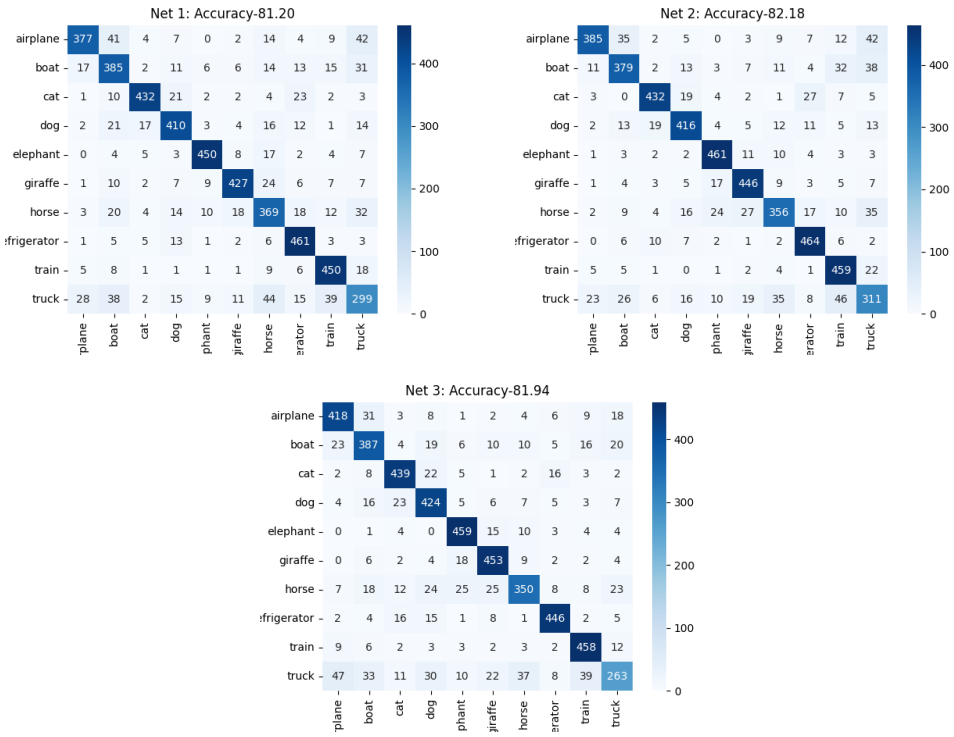

Figure 2: Left Top: Confusion matrix for single layer CNN network, Top Right: Confusion matrix for two layer CNN network, Bottom: Confusion matrix for two layer CNN network with padding

3 Implementation

CODE-hw04_training.py

```
import matplotlib.pyplot as plt

import torch
from torch.utils.data import DataLoader
import torchvision.transforms as tvt

import numpy as np
import time
import pickle
# import os, glob

# User imports
from dataLoader import dataLoader
from model import mynet

# training loop - modified from hw4_s22.pdf
def run_code_for_training(net, train_data_loader):
    net = net.to(device)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(net.parameters(), lr=1e-3, momentum=0.9)
    history = []
    for epoch in range(epochs):
        start = time.time()
        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("\n[epoch:%d, batch:%5d] loss: %.3f"% (epoch + 1, i +
                    ↪ 1, running_loss / float(500)))
                history.append(running_loss/float(500))
```

```
            running_loss = 0.0
        print("Estimated time left (hours): %0.2f"% ((time.time()-start)*(
            ↪ epochs-epoch)/3600))
    return history

# Start training
if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

# define paramters
batch=64
epochs = 50
dataPath = "../hw04_coco_data/Train"
transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5,0.5, 0.5), (0.5,
    ↪  0.5, 0.5))])

# data load
dt = dataLoader(dataPath,transform)
TrainDataLoader = DataLoader(dataset = dt, batch_size = batch, shuffle =
    ↪ True, num_workers = 16)

# train network
net = mynet(1)
history_net1 = run_code_for_training(net,TrainDataLoader)
torch.save(net, "../saves/net1.pth")
net = mynet(2)
history_net2 = run_code_for_training(net,TrainDataLoader)
torch.save(net, "../saves/net2.pth")
net = mynet(3)
history_net3 = run_code_for_training(net,TrainDataLoader)
torch.save(net, "../saves/net3.pth")

# plot graph
plt.plot(history_net1, label="Net:Task 1")
plt.plot(history_net2, label="Net:Task 2")
plt.plot(history_net3, label="Net:Task 3")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss for the three Nets")
plt.legend()
```

```
plt.savefig("../saves/train_loss.png")

# save loss history
losses_hist = [history_net1,history_net2,history_net3]
with open("../saves/losses_hist.pickle", 'wb') as f:
    pickle.dump(losses_hist, f)
```

CODE-hw04_validation.py

```
import matplotlib.pyplot as plt

import torch
from torch.utils.data import DataLoader
import torchvision.transforms as tvt

import numpy as np
import seaborn as sns
import pickle

# User imports
from dataLoader import dataLoader
from model import mynet

def run_code_for_validation(net, valDataLoader, classes):
    net = net.to(device)
    confusion = np.zeros((len(classes),len(classes)))

    for i,data in enumerate(valDataLoader):
        (inputs, labels) = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net(inputs)
        prediction = torch.argmax(outputs, dim=1)

        for j, gt in enumerate(labels):
            confusion[gt][prediction[j]] +=1
    return confusion

def cf_plot(no,confusion):
    sns.heatmap(confusion, fmt="0.0f", cmap='Blues', annot=True, xticklabels
    ↪ =dt.classes, yticklabels=dt.classes)
    plt.title("Net␣%d:␣Accuracy-%0.2f"%(no,(100*np.trace(confusion)/np.sum(
```

```
        ↪ confusion))))
    plt.savefig("../saves/net%d_confusion_matrix.png"%(no))
    plt.close()

if torch.cuda.is_available():
    device = torch.device("cuda:0")
else:
    device = torch.device("cpu")

batch=256
dataPath = "../hw04_coco_data/Val"
transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5,0.5, 0.5), (0.5,
    ↪  0.5, 0.5))])

dt = dataLoader(dataPath,transform)
valDataLoader = DataLoader(dataset = dt, batch_size = batch, shuffle = False
    ↪ , num_workers = 16)

# Net 1 - Validation
net = torch.load("../saves/net1.pth")
net.eval()
confusion1 = run_code_for_validation(net, valDataLoader, dt.classes)
cf_plot(1,confusion1)

# Net 2 - Validation
net = torch.load("../saves/net2.pth")
net.eval()
confusion2 = run_code_for_validation(net, valDataLoader, dt.classes)
cf_plot(2,confusion2)

# Net 3 - Validation
net = torch.load("../saves/net3.pth")
net.eval()
confusion3 = run_code_for_validation(net, valDataLoader, dt.classes)
cf_plot(3,confusion3)

# save loss history
confusion_hist = [confusion1,confusion2,confusion3]
with open("../saves/confusion_hist.pickle", 'wb') as f:
    pickle.dump(confusion_hist, f)
```

CODE-dataloader.py

```
import torch
from torch.utils.data import Dataset
from PIL import Image
import os, glob
import numpy as np

class dataLoader(Dataset):
    def __init__(self,dataPath,transform):
        self.dataPath = dataPath
        self.classes = []
        self.transform = transform
        # image list and corresponding label
        self.imgList = []
        self.labelForImg()

    def __len__(self):
        length = 0
        for cls in self.classes:
            length += len(glob.glob(os.path.join(self.dataPath,cls,"*")))
        return length

    def __getitem__(self,idx):
        img_path = self.imgList[idx,1]
        label = torch.tensor(int(self.imgList[idx,0]), dtype=torch.uint8)
        label = int(self.imgList[idx,0])

        image = Image.open(img_path)
        if self.transform:
            image = self.transform(image)

        return image, label

    def labelForImg(self):
        # get list of classes form datapath
        self.classes = np.sort(os.listdir(self.dataPath))
        print(self.classes)
        # connect label with classes
        for i, cls in enumerate(self.classes):
            imgs = glob.glob(os.path.join(self.dataPath,cls,'*'))
            if i==0:
```

```
                self.imgList = np.vstack((i*np.ones(len(imgs),dtype=int),imgs
                    ↪ ))
            else:
                self.imgList = np.hstack((self.imgList,np.vstack((i*np.ones(
                    ↪ len(imgs),dtype=int),imgs))))
        self.imgList = self.imgList.T
```

CODE-model.py

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

class mynet(nn.Module):
    def __init__(self, task):
        super(mynet, self).__init__()
        self.task = task
        self.conv1 = nn.Conv2d(3, 128, 3) # (A)
        self.conv2 = nn.Conv2d(128, 128, 3) # (B)
        self.pool = nn.MaxPool2d(2, 2)
        # Changing line (C)
        if self.task==1:
            self.fc1 = nn.Linear(128*31*31, 1000) # (C)
        elif self.task==2:
            self.fc1 = nn.Linear(128*14*14, 1000)
        else:
            self.conv1 = nn.Conv2d(3, 128, 3, padding=1)
            self.fc1 = nn.Linear(128*15*15, 1000)
        self.fc2 = nn.Linear(1000, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        ## Uncomment the next statement and see what happens to the
        ## performance of your classifier with and without padding.
        ## Note that you will have to change the first arg in the
        ## call to Linear in line (C) above and in the line (E)
        ## shown below. After you have done this experiment, see
        ## if the statement show n below can be invoked twice with
        ## and without padding. How about three times?

        # Changing line (E)
        if self.task==1:
            x = x.view(-1, 128*31*31)
        elif self.task==2:
```

```
            x = self.pool(F.relu(self.conv2(x))) ## (D)
            x = x.view(-1, 128*14*14)
        else:
            x = self.pool(F.relu(self.conv2(x)))
            x = x.view(-1, 128*15*15)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

4 Lessons Learned

Once again, the programming homework was straightforward and covered the basics of CNN implementation. I did not find any major issues with implementation of CNN. I was running short on time for this assignment week due to attendance in a conference.

5 Suggested Enhancements

The assignment was well written. I do not have any suggestions.