ECE695DL: Homework 4

Leann Demorest

15 March 2021

model.py

```
#######################################################
# BME/ECE 695 Deep Learning
# Homework 4 - Leann Demorest
# model.py
# March 15, 2021
#######################################################
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self, Convo_layers, Pad = False):
        super(Net, self).__init__()
        self.Convo_layers = Convo_layers #Set number of
        convolutional layers
        self.Pad = Pad #Padding True or False

        #Set up convolutions and max pool
        self.conv1 = nn.Conv2d(3, 128, 3)
        self.conv2 = nn.Conv2d(128, 128, 3)
        self.pool = nn.MaxPool2d(2, 2)

        #Net 1: One layer
        if self.Convo_layers == 1 and self.Pad == False:
            self.fc1 = nn.Linear(128*31*31, 1000)
        #Net 2: Two layers
        elif self.Convo_layers == 2 and self.Pad == False:
            self.fc1 = nn.Linear(128*14*14, 1000)
        #Net 3: Two layers and padding
        elif self.Pad == True:
            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):
        #First convoltional layer
        x = self.pool(F.relu(self.conv1(x)))
        #Send through 2nd convolution (if applicable) and reshape
        tensor
        if self.Convo_layers == 1 and self.Pad == False:
            x = x.view(-1, 128*31*31)
        elif self.Convo_layers == 2 and self.Pad == False:
            x = self.pool(F.relu(self.conv2(x)))
            x = x.view(-1, 128*14*14)
        elif self.Pad == True:
            x = self.pool(F.relu(self.conv2(x)))
            x = x.view(-1, 128*15*15)
        #send through fully connected first layer
        x = F.relu(self.fc1(x))
        #send through second fully conneced later to output
        x = self.fc2(x)
        return x
```

dataloader.py

```
#######################################################
# BME/ECE 695 Deep Learning
# Homework 4 - Leann Demorest
# dataloader.py
# March 15, 2021
#######################################################
import torch
import torchvision
import torchvision.transforms as tvt
from torch.utils.data import DataLoader, Dataset

import glob
import os
from PIL import Image

class my_dataset_class(Dataset):
    def __init__(self, args, transform):
        self.FolderPath = []
```

```
        self.RootFolder = args.root_path
        self.class_list = args.class_list
        labels = []
        for items in self.class_list:
            self.FolderPath.append(os.path.join(self.RootFolder,
            items))
            labels.append([items, self.class_list.index(items)])
        self.ImageList = []
        for path in self.FolderPath:
            ClassType = path.split("/")[-1]
            #Get the class type from the folder name
            EncodedClass = self.class_list.index(ClassType)
            #Encode it based on class_list
            for image in glob.glob1(path, '*'):
                self.ImageList.append([EncodedClass,
                os.path.join(path, image)])
        self.transform = transform
        self.Count = {}
        for label in labels:
            self.Count[label[1]] = [label[0],
            [0]*len(self.class_list)]

    def __len__(self): #get length by looking at total contents of
    each folder.
        length = 0
        for path in self.FolderPath:
            length += len(glob.glob1(path, '*'))
        return length

    def __getitem__(self, index): #return one transformed item and
    label
        image = self.ImageList[index][1]
        image = Image.open(image)
        H, W = image.size
        image = self.transform(image).to(dtype=torch.float64)
        Label = self.ImageList[index][0]
        return image, Label
```

hw04_coco_downloader.py

```
#######################################################
# BME/ECE 695 Deep Learning
# Homework 4 - Leann Demorest
# hw04_coco_downloader.py
```

```
# March 15, 2021
##################################################
import argparse
import json
import ast
import requests
import os
from PIL import Image
from requests.exceptions import ConnectionError, ReadTimeout,
TooManyRedirects, MissingSchema, InvalidURL
import logging
from pycocotools.coco import COCO
import numpy as np
import matplotlib.pyplot as plt

import argparse
import json
import ast
import requests
import os
from PIL import Image
from requests.exceptions import ConnectionError, ReadTimeout,
TooManyRedirects, MissingSchema, InvalidURL
import logging

class CocoDownloader():
    def __init__(self, args):
        #set up coco and class variables
        self.json = args.coco_json_path
        self.classes = args.class_list
        self.images_per_class = args.images_per_class
        self.coco = COCO(self.json)
        self.root_path = args.root_path
        self.class_folder = dict.fromkeys(self.classes)
        #make folders if they don't exist
        for category in self.classes:
            self.class_folder[category]=self.root_path+category
            if not os.path.exists(self.class_folder[category]):
                os.makedirs(self.class_folder[category])
    #get image urls for each class
    def get_image_urls(self):
        self.class_urls = dict.fromkeys(self.classes)
        for key in self.class_urls:
            ID = self.coco.getCatIds(key)
            ImgId = self.coco.getImgIds(catIds = ID)
            img_url = self.coco.loadImgs(ImgId)
```

```
            self.class_urls[key] = [i['coco_url'] for i in img_url]
    #download and resize images while i<self.images_per_class
    def save_resize_images(self):
        for key in self.class_urls:
            i = 0; j = 0 #i = images downloaded, j = list index
            folder = self.class_folder[key]
            while i < self.images_per_class:
                self.current_url = self.class_urls[key][j]
                Result = self.get_image(folder)
                if Result == True: i += 1
                j+=1
    #get image (ignore duplicates)
    def get_image(self, folder):
        if len(self.current_url) <= 1:
            return False
        try: img_response = requests.get(self.current_url, timeout
        = 1)
        except Exception as e:
            return False
        img_name = self.current_url.split('/')[-1]
        img_name = img_name.split("?")[0]
        img_file_path = os.path.join(folder, img_name)
        if os.path.exists(img_file_path):
            print('duplicate')
            return False
        with open(img_file_path, 'wb') as img_f:
            img_f.write(img_response.content)
        im = Image.open(img_file_path)
        if im.mode != "RGB":
            im = im.convert(mode = "RGB")
        im_resized = im.resize((64, 64), Image.BOX)
        im_resized.save(img_file_path)
        return True

#parse arguments from commandline
parser = argparse.ArgumentParser(description = 'HW04 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()

#set up instance and call functions
```

```
CocoDownload = CocoDownloader(args)
CocoDownload.get_image_urls()
CocoDownload.save_resize_images()
```

hw04_training.py

```
#######################################################
# BME/ECE 695 Deep Learning
# Homework 4 - Leann Demorest
# hw04_training.py
# March 15, 2021
#######################################################
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tvt
import torch.optim as optim
from torchsummary import summary
from torch.utils.data import DataLoader, Dataset

import copy
import glob
import os
import random
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

from dataloader import my_dataset_class
from model import Net

def run_training(net, path):
    net = copy.deepcopy(net)
    net = net.to(device) #set up CPU/GPU
    CrossEntropy = torch.nn.CrossEntropyLoss() #Set loss function
    optimizer = torch.optim.SGD(net.parameters(), lr = 1e-3,
    momentum = 0.95) #set optimization params
    epochs = 10 #ran 10 epochs on google colab. Only have CPU on PC
    AverageLossPlot = []; IterationLossPlot = [] #Initialize plot
    lists
    for epoch in range(epochs):
        summed_loss = 0.0
```

```
        for batch, data in enumerate(TrainDataLoader):
        #run through dataset with a batch at a time
            images, ground_truth = data
            images = images.to(device).float()
            ground_truth = ground_truth.to(device)
            #forward propogate
            optimizer.zero_grad()
            outputs = net(images)
            #back-propogate
            loss = CrossEntropy(outputs, ground_truth)
            loss.backward()
            optimizer.step()
            summed_loss += loss.item() #convert to float
            if (batch+1)%500 == 0: #append data at batch 250.
                print("\n[epoch:%d, batch:%d] loss:%.2f"%(epoch+1,
                batch+1, summed_loss/float(250)))
                AverageLossPlot.append(summed_loss/float(500))
                IterationLossPlot.append(len(AverageLossPlot))
                summed_loss = 0.0
    torch.save(net.state_dict(), path) #save output to be used for
    validation
    return AverageLossPlot, IterationLossPlot

if __name__ == '__main__':
    #runs CPU on my personal computer. Ran GPU on Google Colab
    if torch.cuda.is_available()== True: device =
    torch.device("cuda:0")
    else: device = torch.device("cpu")

    # initialization reference:
    https://engineering.purdue.edu/DeepLearn/pdf-kak/week3.pdf
    seed = 0
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    np.random.seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmarks = False
    os.environ['PYTHONHASHSEED'] =  str(seed)

    #parse arguments from commandline
    parser = argparse.ArgumentParser(description =
    'HW04_Training/Validation')
    parser.add_argument('--root_path', type=str, required=True)
    parser.add_argument('--class_list', nargs='*', type=str,
    required = True)
```

```
    args, args_other = parser.parse_known_args()

    #Initialize transformation
    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5,
    0.5, 0.5), (0.5, 0.5, 0.5))])
    batch = 10

    #Call dataloader and initialize instance
    TrainingClass = my_dataset_class(args, transform)
    TrainDataLoader = torch.utils.data.DataLoader(dataset =
    TrainingClass, batch_size = batch, shuffle = True, num_workers
    = 0)

    #run net 1 and plot
    model1 = Net(1, False)
    AverageLossPlot, IterationLossPlot = run_training(model1,
    "net1.pth")
    plt.plot(IterationLossPlot, AverageLossPlot, label = 'Net 1')
    #run net 2 and plot
    model2 = Net(2, False)
    AverageLossPlot, IterationLossPlot = run_training(model2,
    "net2.pth")
    plt.plot(IterationLossPlot, AverageLossPlot, label ='Net 2')
    #run net 3 and plot
    model3 = Net(2, True)
    AverageLossPlot, IterationLossPlot = run_training(model3,
    "net3.pth")
    plt.plot(IterationLossPlot, AverageLossPlot, label='Net 3')
    #format plot and save
    plt.legend()
    plt.title('Comparison of CNN Architectures')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.savefig('train_loss.jpg')
```

hw04_validation.py

```
#######################################################
# BME/ECE 695 Deep Learning
# Homework 4 - Leann Demorest
# hw04_validation.py
```

```
# March 15, 2021
#######################################################
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tvt
import torch.optim as optim
from torchsummary import summary
from torch.utils.data import DataLoader, Dataset
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
import copy
import glob
import os
import random
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

from dataloader import my_dataset_class
from model import Net

def run_validation(net, path, Validation = False):
    net = copy.deepcopy(net)
    NetValidationClassCount = copy.deepcopy(ValidationClass.Count)
    net = net.to(device)
    CrossEntropy = torch.nn.CrossEntropyLoss()
    TotalRight = 0
    Total = 0
    for batch, data in enumerate(ValidationDataLoader): #run
    through dataset with a batch at a time
        images, ground_truth = data
        images = images.to(device).float()
        ground_truth = ground_truth.to(device)
        outputs = net(images)
        #Determine which label was selected for each image
        prediction = [torch.argmax(output) for output in outputs]
        #Compare with ground truth to get number right
        NumberRight = [prediction[k] == ground_truth[k] for k in
        prediction]
        #Increment total right and total
        TotalRight += sum(bool(l) for l in NumberRight)
        Total+= len(ground_truth)
```

```
            #Create dictionary to store predictions for confusion
            matrix
            for j in range(len(ground_truth)):
                NetValidationClassCount[int(ground_truth[j])][1][int(pr
                ediction[j])] += 1
        #Calculate accuracy
        Accuracy = TotalRight/Total

        #Create Confusion Matrix Array
        List =[]
        for key in NetValidationClassCount:
            List.append(NetValidationClassCount[key][1])
        Array = pd.DataFrame(List, columns =
        ValidationClass.class_list, index = ValidationClass.class_list)
        plt.figure(figsize = (10,7))
        sn.heatmap(Array, annot=True, fmt = 'd', linewidths=.5)
        NetNumber = path.strip(".pth").strip("net")
        #Plot confusion matrix and save figure
        plt.title("Net "+str(NetNumber)+"\n"+r'Accuracy:
        '+str(Accuracy))
        plt.xlabel('Prediction Label')
        plt.ylabel('Ground Truth Label')
        plt.tight_layout()
        plt.savefig("net"+str(NetNumber)+"_confusion_matrix.jpg")

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

        # initialization reference:
        https://engineering.purdue.edu/DeepLearn/pdf-kak/week3.pdf
        seed = 0
        random.seed(seed)
        torch.manual_seed(seed)
        torch.cuda.manual_seed(seed)
        np.random.seed(seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmarks = False
        os.environ['PYTHONHASHSEED'] =  str(seed)

        #parse arguments from commandline
        parser = argparse.ArgumentParser(description =
        'HW04_Training/Validation')
        parser.add_argument('--root_path', type=str, required=True)
        parser.add_argument('--class_list', nargs='*', type=str,
```

```
required = True)
args, args_other = parser.parse_known_args()

#Initialize transformation
transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5,
0.5, 0.5), (0.5, 0.5, 0.5))])
batch = 10

#Call dataloader and initialize instance
ValidationClass = my_dataset_class(args, transform)
ValidationDataLoader = torch.utils.data.DataLoader(dataset =
ValidationClass, batch_size = batch, shuffle = True,
num_workers = 0)

#run nets for validation
model1 = Net(1, False)
model1.load_state_dict(torch.load("net1.pth",
map_location=device))
model1.eval()
run_validation(model1, "net1.pth", True)

model2 = Net(2, False)
model2.load_state_dict(torch.load("net2.pth",
map_location=device))
model2.eval()
run_validation(model2, "net2.pth", True)

model3 = Net(2, True)
model3.load_state_dict(torch.load("net3.pth",
map_location=device))
model3.eval()
run_validation(model3, "net3.pth", True)
```

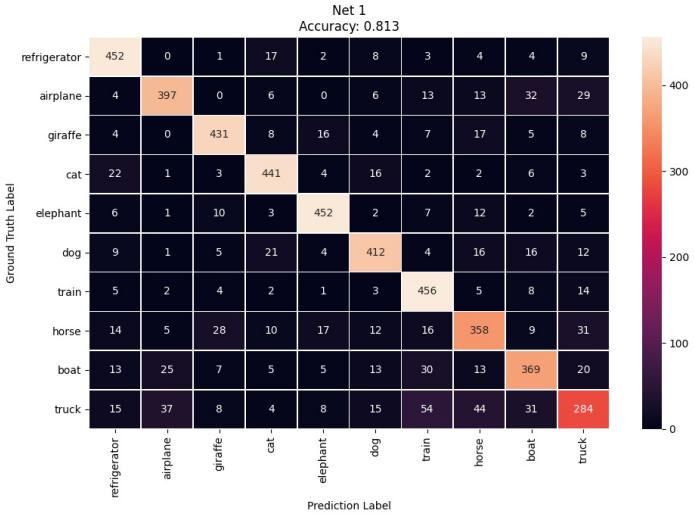

Figure 1: net1 confusion matrix

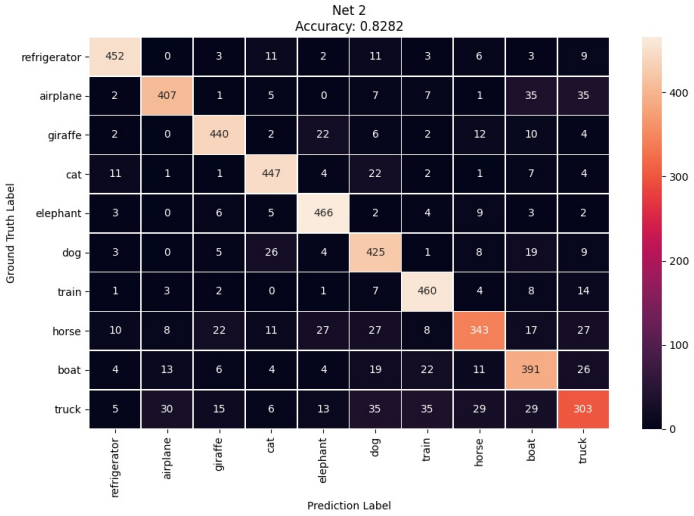

Figure 2: net2 confusion matrix

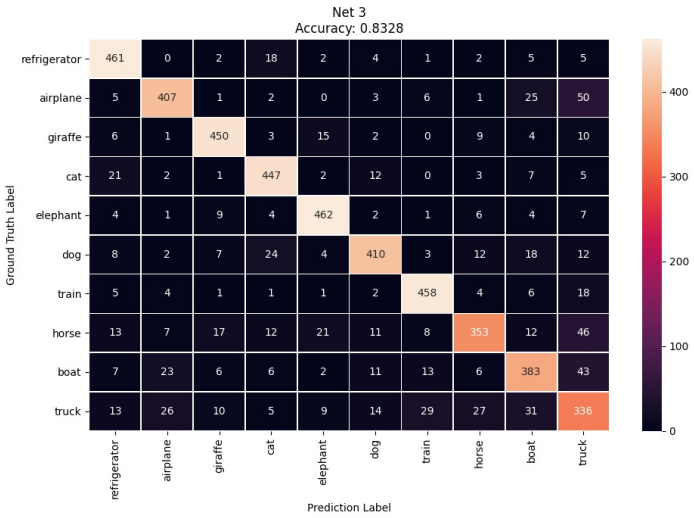

Figure 3: net3 confusion matrix

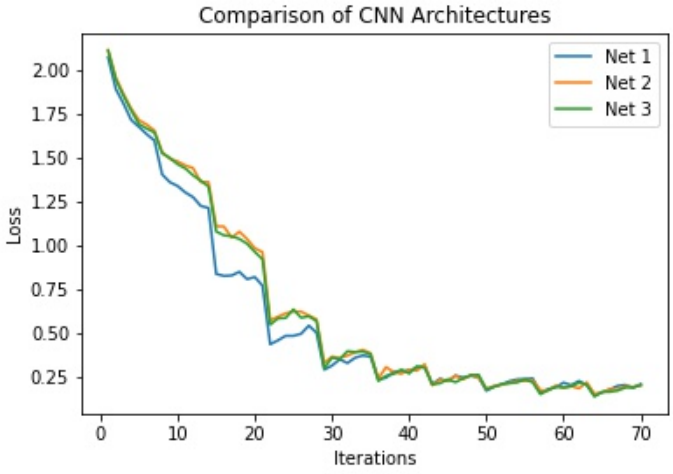

Figure 4: train loss