Lab 3: Neural Networks and Dataloaders in PyTorch

CSC 592: Machine Learning Security and Privacy

**Background**

In this lab assignment, you will create a linear network using PyTorch and experiment with using Dataloaders on a custom dataset.

**Step by Step Guide**

Step 1: Installing PyTorch

If you had previously installed PyTorch for another class or for research you can skip this step.

If you are not very familiar with installing software packages and creating environments in Python it is recommended that you go through the “Software Installation Instructions for PyTorch (not required).doc” document on BrightSpace before completing this step.

The installation steps for PyTorch vary slightly depending on the operating system of your computer. For Linux and Mac users the following link gives the instructions for install PyTorch: <https://pytorch.org/get-started/locally/>

Step 2: Downloading the Dataset

In this assignment we will be working with the Airplanes-Cars-Ships dataset using PyTorch. This dataset contains color images of airplanes, cars and ships. We will create a classifier to classify an image into one of these three categories. The dataset is available as a zip file on BrightSpace. After downloading the zip file, extract it to a folder. It will contain a train and a test folder with further three folders for images belonging to each category. The number of images in each category for the train and test parts is shown below.

airplanes ships cars

**train** 1000 1000 1000

**test** 189 200 193

Step 3: Setting up the Dataset Object

Create a Python application called SimpleClassification as shown below.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Make sure to add the proper Python environment e.g., pytorch2x for Pytorch for the project.

By right clicking on the project name, add a class to the project called MyDataSet.py with the following code in it:

import torch

import torchvision

import numpy as np

import os

from PIL import Image

**class MyDataset(torch.utils.data.Dataset):**

# for Airplanes, Cars and Ships dataset

# train or test folders further have subfolders containing images

# for each category

def \_\_init\_\_(self, data\_dir, transform=None):

self.data\_dir = data\_dir

self.transform = transform

self.all\_image\_paths = []

self.all\_labels = []

for planecarship\_dir in os.listdir(data\_dir): # each category folder

category\_path = os.path.join(data\_dir, planecarship\_dir)

if not os.path.isdir(category\_path):

continue

if planecarship\_dir == "airplanes":

label = 0

elif planecarship\_dir == "cars":

label = 1

elif planecarship\_dir == "ships":

label = 2

image\_paths = [os.path.join(category\_path, f) for f in os.listdir(category\_path) if f.endswith('.jpg')]

# list of image filenames for a category, e.g., airplane, car, or ship

labels = [label for i in range(len(image\_paths))]

self.all\_image\_paths += image\_paths

self.all\_labels += labels

self.num\_classes = len(set(self.all\_labels))

**def \_\_len\_\_(self):**

return len(self.all\_image\_paths)

**def \_\_getitem\_\_(self, index):**

img\_path = self.all\_image\_paths[index]

label = self.all\_labels[index]

img = Image.open(img\_path).convert('RGB')

if self.transform is not None:

img = self.transform(img)

return img, label

The purpose of the DataSet object is to access the training and test data. If the total training data is small, we can read all the image pixels and store all images in multi-dimensional tensors in the dataset in the computer memory (RAM) directly. However, if the number of images is large and images happen to be relatively high resolution, then one option is to store the image filenames (with full path info) in the dataset. The following code stores all images in the train or test folders as list of filenames and their labels. It will also apply a transform to the images. Transform usually resizes the image and changes the 0-255 pixel scale to either 0 to 1, or -1 to 1 range.

The \_\_get\_item\_\_ function in the DataSet class returns one data item and its label.

Step 4: Setting up the Dataloader

The DataSet is tied to a DataLoader which obtains a batch of data from the dataset and feeds to the training or test loop in the code. Add a file called Utils.py to the project with the following code in it:

import torch

from MyDataSet import MyDataset

import torchvision

import matplotlib.pyplot as plt

**def get\_train\_loader(data\_dir, batch\_size, transform=None):**

dataset = MyDataset(data\_dir, transform=transform)

data\_loader = torch.utils.data.DataLoader(

dataset,

batch\_size=batch\_size,

shuffle=True,

num\_workers=4,

pin\_memory=True,

drop\_last=True

)

return data\_loader

**def get\_test\_loader(data\_dir, batch\_size, transform=None):**

dataset = MyDataset(data\_dir, transform=transform)

data\_loader = torch.utils.data.DataLoader(

dataset,

batch\_size=batch\_size,

shuffle=False,

num\_workers=4,

pin\_memory=True

)

return data\_loader

**def plot\_images(images, labels):**

# normalise=True below shifts [-1,1] to [0,1]

img\_grid = torchvision.utils.make\_grid(images, nrow=4, normalize=True)

np\_img = img\_grid.numpy().transpose(1,2,0) # pytorch has the order, c,w,h

# to be able to view an image, we need to change the order and

# put it in width, height, color order

plt.imshow(np\_img)

plt.show()

Step 5: Making the Model

Add a class called NetworkLinear with the following code in it.

import torch

import torch.nn as nn

import torch.nn.functional as F

**class NetworkLinear(nn.Module):**

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.fc1 = nn.Linear(3\*224\*224, 100)

self.fc2 = nn.Linear(100, 3)

self.sm = nn.Softmax(dim=1)

**def forward(self, x):**

x = x.view(-1, 3\*224\*224)

x = F.relu(self.fc1(x))

x = self.sm(self.fc2(x)) # softmax activation on final layer

return x

As you can see, it uses two linear layers. The first one matches its input to the input image size i.e., 3x224x224 and has 100 neurons in it. It uses the Relu activation function.

The second layer has 3 neurons in it as we have three categories of images. The input specification of the second layer has to match the output of the first layer e.g., 100 in our example. We will test the difference between using a linear network versus a CNN based network in the next lab assignment. For this assignment we will only work with standard fully connected feed forward neural networks.

Step 6: The Training Algorithm

Type the following code in the main file, SimpleClassification.py:

import sys

from torchvision import transforms

import torch

import Utils

import torch.optim as optim

from NetworkLinear import NetworkLinear

def main():

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

data\_dir\_train = "D:/DeepLearning2/data/Dataset\_PlanesCarsShips/train" #TODO: change to your filepath

data\_dir\_test = "D:/DeepLearning2/data/Dataset\_PlanesCarsShips/test" #TODO: change to your filepath

image\_transforms = {

"train": transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

transforms.Normalize([0.5, 0.5, 0.5],

[0.5, 0.5, 0.5])

]),

"test": transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

transforms.Normalize([0.5, 0.5, 0.5],

[0.5, 0.5, 0.5])

])

}

# ToTensor converts a PIL Image or numpy.ndarray (HxWxC) in the range [0, 255]

# to a torch.FloatTensor of shape (CxHxW) in the range [0.0, 1.0]

batch\_size = 16

num\_epochs = 25

train\_loader = Utils.get\_train\_loader(data\_dir\_train, batch\_size, image\_transforms["train"])

test\_loader = Utils.get\_test\_loader(data\_dir\_test, batch\_size, image\_transforms["test"])

train\_iter = iter(train\_loader)

images, labels = next(train\_iter) # get a batch of data e.g., 16x3x224x224

print(images[0].shape)

Utils.plot\_images(images,labels) # plot images

net = NetworkLinear() # create the simple linear model

#net = NetworkCNN() # create the CNN model

loss\_criterion = torch.nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

running\_loss = 0

print\_freq = 100

for epoch in range(num\_epochs):

for i, data in enumerate(train\_loader):

inputs, labels = data

optimizer.zero\_grad()

outputs = net(inputs) # forward pass

loss = loss\_criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

#if i % print\_freq == print\_freq-1:

print('epoch:',epoch, i+1, running\_loss/print\_freq)

running\_loss = 0

#-----------compute accuracy on trained model-------------

total = 0 # keeps track of how many images we have processed

correct = 0 # keeps track of how many correct images our net predicts

with torch.no\_grad():

for i, data in enumerate(test\_loader):

images, labels = data

outputs = net(images)

\_, predicted = torch.max(outputs.data, 1)

total += labels.size()[0]

correct += (predicted == labels).sum().item()

print("Accuracy: ", correct/total)

if \_\_name\_\_ == "\_\_main\_\_":

sys.exit(int(main() or 0))

Study the above code to see how the training and test loops are set up. Where it says “#TODO: change to your filepath” change the file path to where you saved the dataset (you must make this change in two places in the code).

Step 7: Training the Model

Run the program, you should get a classification accuracy of about 75% when using the linear network as the classifier:

A screenshot of a computer

Description automatically generated

**Deliverables**

Submit the following two documents on Brightspace:

Deliverable #1: A screenshot of the output of your code.

Deliverable #2: A copy of your code (the .py files).