Lab 4: Convolutional Neural Networks (CNNs)

CSC 592: Machine Learning Security and Privacy

**Background**

Convolutional Neural Networks (CNNs) are crucial in modern artificial intelligence, particularly in image classification, due to their ability to automatically extract hierarchical features from visual data. Unlike classical machine learning methods that require manual feature engineering, CNNs use convolutional layers to detect patterns such as edges, textures, and complex structures, making them highly effective in tasks like object detection, facial recognition, and medical image analysis. CNN’s ability to learn spatial patterns enables them to generalize well across different image datasets.

In this lab assignment, you will create and train a convolutional neural network (CNN) using PyTorch. You will train the CNN on the same custom dataset used in the previous lab (the Airplanes-Cars-Ships dataset). We will then compare the performance of the model in the previous lab to the CNN model used in this lab.

**Step by Step Guide**

Step 1: Downloading the Dataset

First, create a Python application called SimpleCNNTrainer. In this assignment we will be working again with the Airplanes-Cars-Ships dataset using PyTorch. This dataset contains color images of airplanes, cars and ships.

We will create a CNN to classify an image into one of these three categories. The dataset is available as a zip file on BrightSpace (the same dataset as the previous lab). After downloading the zip file, extract it to a folder. It will contain a train and a test folder with three sub 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 2: Setting up the Dataset

Create a Python application and add in the “MyDataSet.py” class as written below:

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

Step 3: Setting up the Dataloader

Add in the “Utils.py” class as written below:

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 4: Creating the CNN model

Add a class called “NetworkCNN.py” with the following code in it.

import torch

import torch.nn as nn

import torch.nn.functional as F

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

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

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

self.conv1 = nn.Conv2d(3, 32, 5) # input is color image with 3 channels

# 32 is the number of feature maps and the kernel size is 5x5

self.pool = nn.MaxPool2d(2,2)

# maxpool will be used multiple times, but defined once

# in\_channels = 32 because self.conv1 output is 32 channels

self.conv2 = nn.Conv2d(32,6,5)

# 53\*53 comes from the dimension of the last conv layer

self.fc1 = nn.Linear(6\*53\*53, 100)

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

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

def forward(self, x):

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = x.view(-1, 6\*53\*53)

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

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

return x

Step 5: Creating the Training Algorithm

In the main file (“SimpleCNNTrainer.py”) copy the following code as shown below. 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).

import sys

from torchvision import transforms

import torch

import Utils

import torch.optim as optim

from NetworkCNN import NetworkCNN

def main():

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

data\_dir\_train = "C:/Users/Kaleel/Desktop/Machine Learning Security and Privacy/Week 4/Dataset\_PlanesCarsShips/train" #TODO: change to your filepath

data\_dir\_test = "C:/Users/Kaleel/Desktop/Machine Learning Security and Privacy/Week 4/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 = NetworkCNN() # create the simple linear 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))

Step 6: Training the CNN model

Run “SimpleCNNTrainer.py” and record the output. Your output should appear similar to the output shown below:

A screenshot of a computer

AI-generated content may be incorrect.

Note due to the randomized weights you may need to run the training 2-3 times to get an accuracy close to 80%.

**Lab Assignment**

**Exercise 1:** In your code for the lab after you create the model, measure the number of trainable parameters for the CNN model using the following code snippet:

model\_parameters = filter(lambda p: p.requires\_grad, net.parameters())

params = sum([numpy.prod(p.size()) for p in model\_parameters])

print("Number of trainable parameters:", params)

Note: to use the code snippet you can insert it before the training occurs. You will also need to import numpy before using the code snippet. You can use the code snippet BEFORE the model is trained.

**Exercise 2:** Go back to your code for lab 3 (the simple feedforward neural network model). Use the code snippet from Exercise 1 to measure the number of trainable parameters for the simple model from lab 3?

**Exercise 3:** Which model is more efficient to train in terms of the number of trainable parameters?

**Exercise 4:** What is the size (dimension) of the input for the CNN model?

**Exercise 5:** What is the size (dimension) of the input for the simple feedforward model from lab 3?

**Exercise 6:** Which model performed better on the Airplanes-Cars-Ships dataset, the feedforward neural network model from lab 3 or the CNN from this lab? How large was the difference in performance? Can you hypothesize on why one model performed better than the other model?

**Deliverables**

Submit the following two documents on Brightspace:

Deliverable #1: A screenshot of the output of your code and answers to the exercises.

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