CS 559: Neural Networks

Homework 5

Author:
Gaetano Coppoletta
Email:
gcoppo2@uic.edu

October 20, 2022

1 Introduction

In this homework we have to design and train a neural network to classify images containing various shape. The dataset contains 90,000 images, where each image has size 200x200 and belongs to one of the 9 classes: Circle, Square, Octagon, Heptagon, Nonagon, Star, Hexagon, Pentagon, Triangle. There are 10,000 images per class.

2 Create training and testing sets

First of all we want to write a program that reads the files and creates one variable for training and another for testing. The training set should contain 8,000 images per class, and the test set should contain the remaining 2,000 images per class, with the corresponding labels as indicated by the file names. We use the file names to split training and testing sets.

3 Design the neural network

Now we want to design and implement a neural network that will take a 200x200 image and decide which one of the 9 classes the input corresponds to.

3.1 First network

First of all we try the neural network implemented in torch3.py, adapting some parameters and training it for 9 epochs. This network uses Adam optimizer as optimization method, training batch size of 100, test batch size equal to 1000 and Cross Entropy as loss function. The code of the network is listed below.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(614656, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
```

```
23          return x
```

The results are listed below.

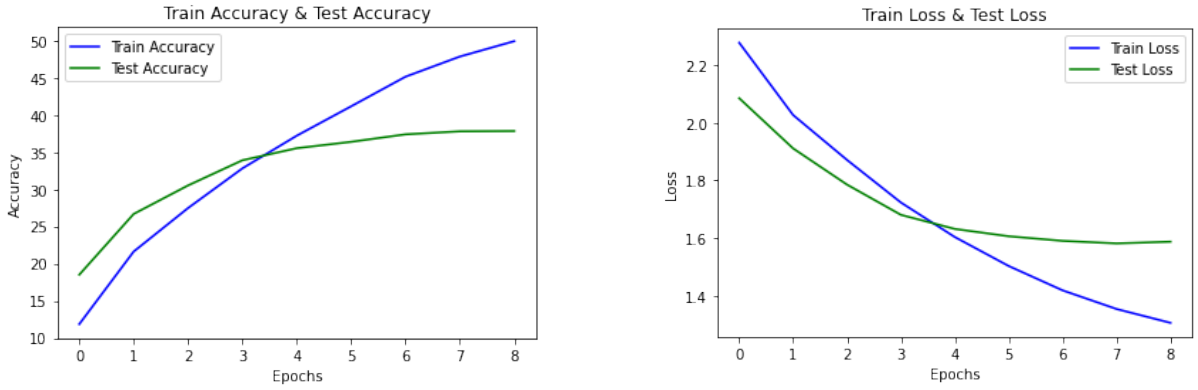

Figure 1: Accuracy and loss on train and test sets

As we can see from the chart, this network is not working as desired, in fact we have a test accuracy which is less than 40

3.2 Second network

After doing some tests, in particular adding, removing and modifying the layers of the network, a good solution was found using three convolutional layer, three max pooling layer and 3 linear layer, using relu as activation function. This network uses Adam optimizer as optimization method, training batch size of 100, test batch size equal to 100 and Cross Entropy as loss function. This network has been trained for 20 epochs and it uses a learning parameter equal to 10^{-3}. With this network we obtain better results, at the end of epoch 20 we have:

- Training Loss: 0.373081, Training Accuracy: 85.81
- Test Loss: 0.451795, Test Accuracy: 83.95

The code of the network is listed below:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 5)
        self.max_pool1 = nn.MaxPool2d(4, 4)
        self.conv2 = nn.Conv2d(8, 16, 5)
        self.max_pool2 = nn.MaxPool2d(5, 5)
        self.conv3 = nn.Conv2d(16, 32, 5)
        self.max_pool3 = nn.MaxPool2d(5, 5)
        self.lin1 = nn.Linear(32, 128)
        self.lin2 = nn.Linear(128, 84)
        self.lin3 = nn.Linear(84, 9)
    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.max_pool2(x)
        x = self.conv3(x)
        x = F.relu(x)
        x = self.max_pool3(x)
        x = torch.flatten(x, 1)
        x = self.lin1(x)
        x = F.relu(x)
        x = self.lin2(x)
        x = F.relu(x)
        x = self.lin3(x)
        return x
```

The results are listed below.

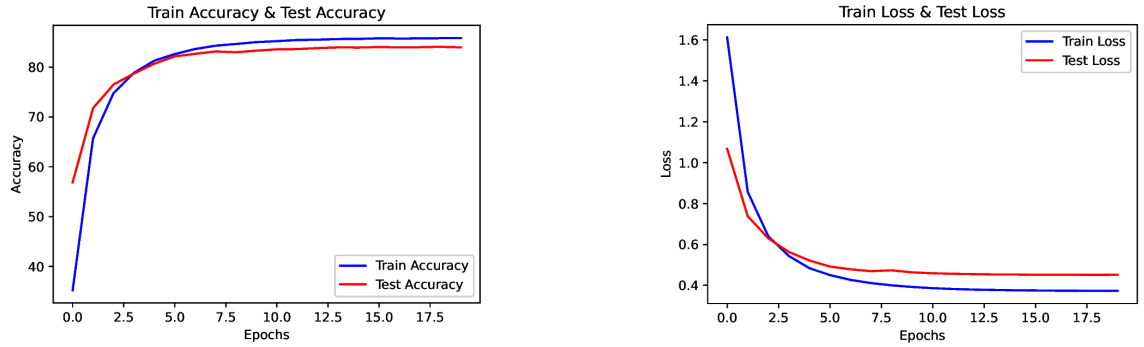

Figure 2: Accuracy and loss on train and test sets

4 Code 1

```
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR
import os
import shutil

from torchvision.datasets import ImageFolder

names = ["Circle", "Square", "Octagon", "Heptagon", "Nonagon", "Star", "Hexagon"
    , "Pentagon", "Tringle"]
train_loss=[]
train_acc=[]
test_loss=[]
test_acc=[]

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 5)
        self.max_pool1 = nn.MaxPool2d(4, 4)
        self.conv2 = nn.Conv2d(8, 16, 5)
        self.max_pool2 = nn.MaxPool2d(5, 5)
        self.conv3 = nn.Conv2d(16, 32, 5)
        self.max_pool3 = nn.MaxPool2d(5, 5)
        self.lin1 = nn.Linear(32, 128)
        self.lin2 = nn.Linear(128, 84)
        self.lin3 = nn.Linear(84, 9)
    def forward(self, x):
```

```
        x = self.conv1(x)
        x = F.relu(x)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.max_pool2(x)
        x = self.conv3(x)
        x = F.relu(x)
        x = self.max_pool3(x)
        x = torch.flatten(x, 1)
        x = self.lin1(x)
        x = F.relu(x)
        x = self.lin2(x)
        x = F.relu(x)
        x = self.lin3(x)
        return x
#create directories
os.mkdir("images")
os.mkdir("./images/test")
os.mkdir("./images/train")

for i in range (0,len(names)):
  os.mkdir("./images/test/"+names[i])
  os.mkdir("./images/train/"+names[i])

i=0
j=0
names.sort()
filenames = os.listdir("./output") #where the images are stores
filenames.sort()
for filename in filenames:
  if i < 8000:
    shutil.copy("./output/"+filename,"./images/train/"+names[j]+"/")
    i+=1
  elif i<10000:
    shutil.copy("./output/"+filename,"./images/test/"+names[j]+"/")
    i+=1
  else:
    j+=1
    i=0
    if j==9:
      break



def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    tot_loss = 0
    correct = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
```

```
87          loss = torch.nn.CrossEntropyLoss()(output, target)
88          loss.backward()
89          optimizer.step()
90
91          pred = output.argmax(dim=1, keepdim=True)
92          correct += pred.eq(target.view_as(pred)).sum().item()
93
94          tot_loss = tot_loss + loss.item()
95          if batch_idx % args.log_interval == 0:
96              print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}, Accuracy:
                    {:.2f}%'.format(
97                  epoch, batch_idx * len(data), len(train_loader.dataset),
98                         100. * batch_idx / len(train_loader), tot_loss / (
                            batch_idx + 1),
99                         100.0 * correct / ((batch_idx + 1) * args.batch_size)))
100
101     print('End of Epoch: {}'.format(epoch))
102     train_loss.append(tot_loss / (len(train_loader)))
103     train_acc.append(100.0 * correct / (len(train_loader) * args.batch_size))
104     print('Training Loss: {:.6f}, Training Accuracy: {:.2f}%'.format(
105         tot_loss / (len(train_loader)), 100.0 * correct / (len(train_loader) *
                args.batch_size)))
106
107
108 def test(args, model, device, test_loader):
109     model.eval()
110     tot_loss = 0
111     correct = 0
112     with torch.no_grad():
113         for data, target in test_loader:
114             data, target = data.to(device), target.to(device)
115             output = model(data)
116             tot_loss += torch.nn.CrossEntropyLoss()(output, target).item()  #
                    sum up batch loss
117             pred = output.argmax(dim=1, keepdim=True)  # get the index of the
                    max log-probability
118             correct += pred.eq(target.view_as(pred)).sum().item()
119     test_loss.append(tot_loss / (len(test_loader)))
120     test_acc.append(100.0 * correct / (len(test_loader) * args.test_batch_size))
121     print('Test Loss: {:.6f}, Test Accuracy: {:.2f}%'.format(
122         tot_loss / (len(test_loader)), 100.0 * correct / (len(test_loader) *
                args.test_batch_size)))
123
124
125 def main():
126     # Training settings
127
128     parser = argparse.ArgumentParser(description='HW5')
129     parser.add_argument('--batch-size', type=int, default=100, help='input batch
            size for training (default: 100)')
130     parser.add_argument('--test-batch-size', type=int, default=100,
131                         help='input batch size for testing (default: 100)')
132     parser.add_argument('--epochs', type=int, default=20, help='number of epochs
```

```
        to train')
133     parser.add_argument('--lr', type=float, default=1e-3, help='learning rate')
134     parser.add_argument('--gamma', type=float, default=0.7, help='Learning rate
            step gamma')
135     parser.add_argument('--seed', type=int, default=2022)
136     parser.add_argument('--log-interval', type=int, default=100,
137                         help='how many batches to wait before logging training
                                status')
138     parser.add_argument('--save-model', action='store_true', default=True, help=
            'For Saving the current Model')
139     parser.add_argument('-f')
140     args = parser.parse_args()
141
142     torch.manual_seed(args.seed)
143
144     device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
145
146     transform = transforms.Compose([transforms.ToTensor()])
147
148     dataset1 = datasets.ImageFolder('./images/train',transform)
149     dataset2 = datasets.ImageFolder('./images/test',transform)
150     train_loader = torch.utils.data.DataLoader(dataset1, batch_size=100, shuffle
            =True)
151     test_loader = torch.utils.data.DataLoader(dataset2, batch_size=100)
152
153     model = Net().to(device)
154     optimizer = optim.Adam(model.parameters(), lr=args.lr)
155
156     scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)
157     for epoch in range(1, args.epochs+1):
158         train(args, model, device, train_loader, optimizer, epoch)
159         test(args, model, device, test_loader)
160         scheduler.step()
161
162     if args.save_model:
163         torch.save(model.state_dict(), "/content/drive/MyDrive/Colab Notebooks
                /0602-657811153-Coppoletta.pt")
164
165     plt.plot(train_loss, c = 'b')
166     plt.plot(test_loss, c = 'r')
167     plt.legend(['Train Loss','Test Loss'])
168     plt.title("Train Loss & Test Loss")
169     plt.xlabel("Epochs")
170     plt.ylabel("Loss")
171     plt.savefig('/content/drive/MyDrive/Colab Notebooks/loss', format = 'eps')
172     plt.show()
173     plt.plot(train_acc, c = 'b')
174     plt.plot(test_acc, c = 'r')
175     plt.legend(['Train Accuracy','Test Accuracy'])
176     plt.title("Train Accuracy & Test Accuracy")
177     plt.xlabel("Epochs")
178     plt.ylabel("Accuracy")
179     plt.savefig('/content/drive/MyDrive/Colab Notebooks/accuracy', format = 'eps
```

```
        ')
    plt.show()


if __name__ == '__main__':
    main()
```

5 Code 2

```
import os
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
from PIL import Image

name = []
image = []
image_path = './images' #current directory, where the images are
for i in os.listdir(image_path):
    name.append(i)
    image.append(Image.open(os.path.join(image_path, i)))
image = list(map(lambda i: transforms.Compose([transforms.ToTensor()])(i).
    unsqueeze(0),image))

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 5)
        self.max_pool1 = nn.MaxPool2d(4, 4)
        self.conv2 = nn.Conv2d(8, 16, 5)
        self.max_pool2 = nn.MaxPool2d(5, 5)
        self.conv3 = nn.Conv2d(16, 32, 5)
        self.max_pool3 = nn.MaxPool2d(5, 5)
        self.lin1 = nn.Linear(32, 128)
        self.lin2 = nn.Linear(128, 84)
        self.lin3 = nn.Linear(84, 9)
    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.max_pool2(x)
        x = self.conv3(x)
        x = F.relu(x)
        x = self.max_pool3(x)
        x = torch.flatten(x, 1)
        x = self.lin1(x)
```

```
41          x = F.relu(x)
42          x = self.lin2(x)
43          x = F.relu(x)
44          x = self.lin3(x)
45          return x
46  def main():
47      net = Net()
48      net.load_state_dict(torch.load('0602-657811153-Coppoletta.pt'))
49      net.eval()
50      names = ['Circle','Heptagon','Hexagon','Nonagon','Octagon','Pentagon','
            Square','Star','Triangle']
51      solutions = list(zip(name, list(map(lambda k: names[k], list(map(lambda l:
            net(l).argmax().item(), image))))))
52      textfile = open("prediction.txt", "w")
53      for i in solutions:
54          print(f'{i[0]}: {i[1]}')
55          textfile.write(f'{i[0]}: {i[1]}')
56          textfile.write("\n")
57      textfile.close()
58  main()
```