ECE/CS-559

Neural Networks

Homework – 6

Report

Submitted by: Sahil Lamba

UIN: 668186281

Used Architecture of CNN:

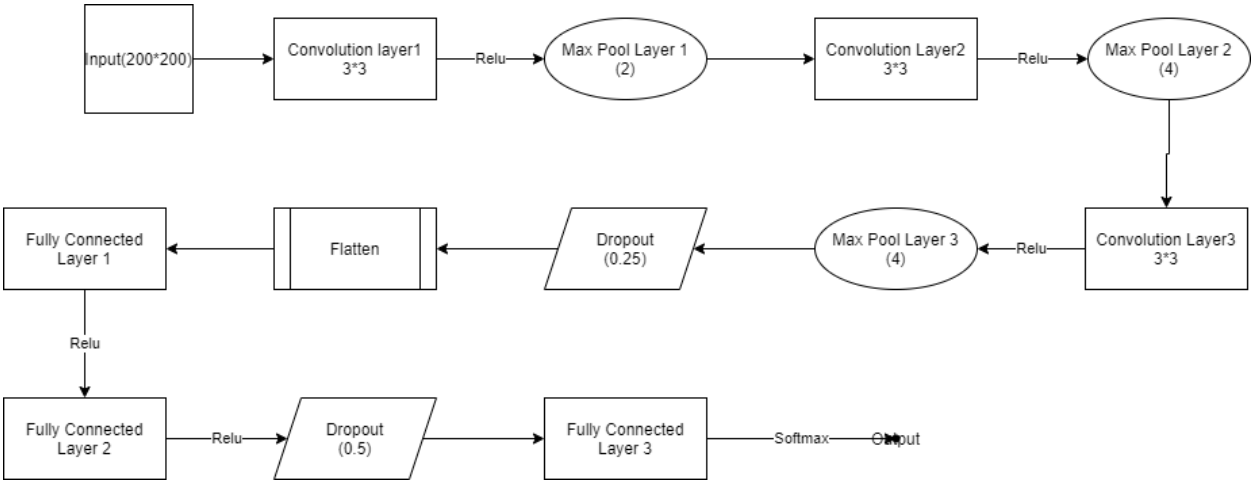

Figure 1: CNN architecture of the model

Hyperparameters Finalized:

1. **Epochs Run: 25**
2. **Gamma: 0.95**
3. **Optimizer: Adam**
4. **Loss Function: Cross Entropy**
5. **Activation Functions: ReLU, SoftMax**
6. **Learning Rate: 0.001**

Result:

1. **Training Loss: 1.417527**
2. **Training Accuracy: 95.55%**
3. **Test Loss: 1.404349**
4. **Test Accuracy: 96.84%**

Hyperparameters Tried and didn't work:

1. **Epochs: 10, 14, 15, 20, 30**

The experiments with epoch showed visible results like still increasing accuracy values when the epoch values are lesser than final epoch value and nearly same accuracy but more training time when epochs are more than the final epoch value of 25.

2. **Gamma: 0.7, 0.8, 0.9, 0.97, 0.99**

The experiments with gamma showed visible results too like model accuracy jumps got slower and slower each epoch and got settled around 50-60 percent with gamma values 0.7, 0.8, 0.9. Gamma values of 0.97 and 0.99 didn't help as much as decided.

3. **Optimizer: RMS prop**

Adam optimizer gave best accuracy i.e., 96% + while RMS prop couldn't take it to 90%+.

4. **Activation Functions: Sigmoid, Tanh**

Didn't give accuracy percentages as good as SoftMax on the last layer and ReLU on others.

5. **Learning Rate: 0.1, 0.01, 0.0001, 0.00001**

All the above learning rates froze both training and test accuracy percentages below 20%.

Graphs:

1. Train Accuracy, Test Accuracy vs Epochs:

Figure 2:Training Accuracy vs Test Accuracy vs Epochs

2. Train Loss, Test Loss vs Epochs:

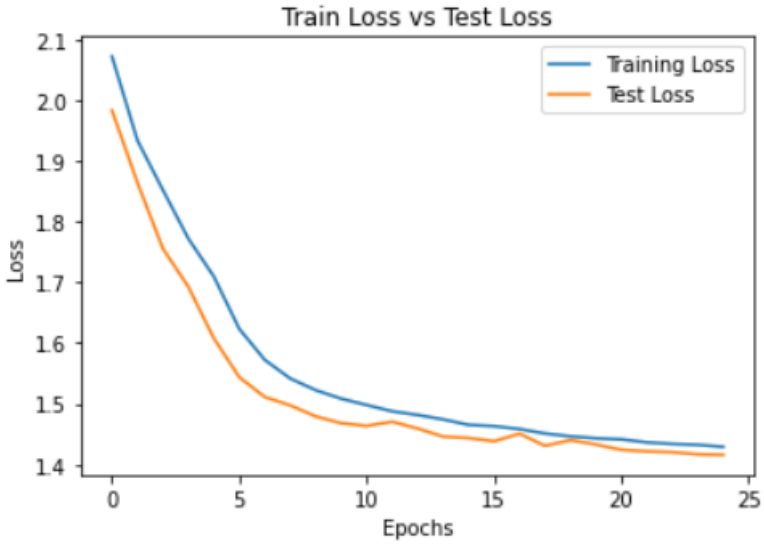

Figure 3: Training Loss vs Test Loss vs Epochs

Code

In [ ]:

```
import argparse
import torchvision
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torchvision.utils import save_image
from torch.optim.lr_scheduler import StepLR
from sklearn.model_selection import train_test_split
import numpy as np
import cv2
import glob
import os
import matplotlib.pyplot as plt
```

Below code creates different folders for Test and Train and within them folders for different classes of shapes

In [ ]:

```
def get_data(dataset, labels):
    train_data = []
    test_data = []
    train_label = []
    test_label = []
    subset={}
    train_test_data={}
    index=0
    classes = np.unique(np.array(labels))
    classes_split = np.array_split(labels, 9)

    start=0
    stop=10000

    for i in classes:
        train_path=os.path.join("/kaggle/Train",i)      #Change path to desired directory for new folders
        test_path=os.path.join("/kaggle/Test",i)
        if not os.path.exists(train_path) and not os.path.exists(test_path):
            os.mkdir(train_path)
            os.mkdir(test_path)
```

```
        subset[i] = torch.utils.data.Subset(dataset, np.arange(start,stop))
        start+=10000
        stop+=10000
        train_test_data[i] = torch.utils.data.random_split(subset[i], [8000, 2000])
        for j in range(len(train_test_data[i][0])):
            torchvision.utils.save_image(transforms.ToTensor()(train_test_data[i][0][j][0]),
                                os.path.join(train_path,i+"_"+str(j)+".png"))
        for k in range(len(train_test_data[i][1])):
            torchvision.utils.save_image(transforms.ToTensor()(train_test_data[i][1][k][0]),
                                os.path.join(test_path,i+"_test"+str(k)+".png"))
```

In [ ]:

```
def get_label():
    y=[]
    for file_name in os.listdir('/kaggle/input/dataset-geometry/output'):  #image folder path in dataset
        y.append(file_name.split("_")[0])
    print(len(y))
    return y
```

In [ ]:

```
def script():
    dataset = datasets.ImageFolder('/kaggle/input/dataset-geometry/')      #dataset path in direcotry
    labels = get_label()
    get_data(dataset, labels)
```

In [ ]:

```
train = "Train"
test = "Test"
dataset = "/kaggle"
train_path = os.path.join(dataset, train)
test_path = os.path.join(dataset, test)
if not os.path.exists(train_path) and not os.path.exists(test_path):
    os.mkdir(train_path)
    os.mkdir(test_path)
script()
```

Bewlo is the architecture of Neural Network used

In [ ]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
```

```
        self.conv3 = nn.Conv2d(64, 128, 3, 1)
#         self.conv4 = nn.Conv2d(128, 256,3,1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(3200, 256)
        self.fc2 = nn.Linear(256,128)
        self.fc3 = nn.Linear(128, 9)
#         self.fc4 = nn.Linear(128,9)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 4)
        x = self.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 4)
        x = self.dropout1(x)
#         x = self.conv4(x)
#         x = F.relu(x)
#         x = F.max_pool2d(x, 4)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
#         x = self.fc3(x)
#         x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        x = F.softmax(x)
        return x
```

In [ ]:

```
def train(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)
#         transform = transforms.Compose([transforms.Grayscale(num_output_channels=1)])
#         tensor_img = transform(data)
        optimizer.zero_grad()
```

```
        output = model(data)
        loss = torch.nn.CrossEntropyLoss()(output, target)
        loss.backward()
        optimizer.step()

        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

        tot_loss = tot_loss + loss.item()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}, Accuracy: {:.2f}%'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), tot_loss/(batch_idx+1), 100.0*correct/((batch_idx+1)*50)))

    print('End of Epoch: {}'.format(epoch))
    print('Training Loss: {:.6f}, Training Accuracy: {:.2f}%'.format(
        tot_loss/(len(train_loader)), 100.0*correct/(len(train_loader)*50)))
    return (tot_loss/(len(train_loader))),(100.0*correct/(len(train_loader)*50))
```

In [ ]:

```
def test(model, device, test_loader):
    model.eval()
    tot_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            tot_loss += torch.nn.CrossEntropyLoss()(output, target).item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    print('Test Loss: {:.6f}, Test Accuracy: {:.2f}%'.format(
        tot_loss/(len(test_loader)), 100.0*correct/(len(test_loader)*100)))
    return (tot_loss/(len(test_loader))), (100.0*correct/(len(test_loader)*100))
```

In [ ]:

```
torch.manual_seed(1)
train_accuracy_list=[]
test_accuracy_list=[]
train_loss_list=[]
test_lost_list=[]

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
transform = transforms.Compose([
```

```
transforms.Grayscale(num_output_channels=1),
transforms.ToTensor(),
transforms.Normalize((0.1307,), (0.3081,))])

train_set = datasets.ImageFolder('/kaggle/Train', transform=transform)      #path of newly created train folder
test_set = datasets.ImageFolder('/kaggle/Test', transform=transform)        #path of newly created test folder

train_loader = torch.utils.data.DataLoader(train_set, batch_size=50, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=100, shuffle = True)
model = Net().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

scheduler = StepLR(optimizer, step_size=1, gamma=0.95)
for epoch in range(1, 35 + 1):
    a,b=train(model, device, train_loader, optimizer, epoch)
    train_loss_list.append(a)
    train_accuracy_list.append(b)
#         torch.cuda.empty_cache()
    c,d=test(model, device, test_loader)
    test_lost_list.append(c)
    test_lost_list.append(d)
#         torch.cuda.empty_cache()
    scheduler.step()
    torch.save(model.state_dict(), "geometry_cnn.pt")
```

In [ ]:

```
epochs = np.arange(0,25)
```

In [ ]:

```
test_accuracy = []
test_loss = []
for i in range(50):
    if i%2==0:
        test_loss.append(test_lost_list[i])
    else:
        test_accuracy.append(test_lost_list[i])
```

In [ ]:

```
plt.plot(epochs,train_loss_list, label = "Training Loss")
plt.plot(epochs,test_loss, label = "Test Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```

```
plt.title("Train Loss vs Test Loss")
plt.show()
```

In [ ]:

```
plt.plot(epochs,train_accuracy_list, label = "Training Accuracy")
plt.plot(epochs,test_accuracy, label = "Test Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.title("Train Accuracy vs Test Accuracy")
plt.show()
```

Inference Module

In [ ]:
```
import argparse
import torchvision
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torchvision.utils import save_image
from torch.optim.lr_scheduler import StepLR
from sklearn.model_selection import train_test_split
import numpy as np
import cv2
import glob
import os
import matplotlib.pyplot as plt
from PIL import Image
```

In [ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [ ]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.conv3 = nn.Conv2d(64, 128, 3, 1)
#         self.conv4 = nn.Conv2d(128, 256,3,1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(3200, 256)
        self.fc2 = nn.Linear(256,128)
        self.fc3 = nn.Linear(128, 9)
#         self.fc4 = nn.Linear(128,9)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 4)
        x = self.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 4)
        x = self.dropout1(x)
#         x = self.conv4(x)
#         x = F.relu(x)
#         x = F.max_pool2d(x, 4)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
#         x = self.fc3(x)
#         x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        x = F.softmax(x)
        return x
```

In [ ]:
```
pretrained_model = Net()
pretrained_model.load_state_dict(torch.load('/kaggle/input/geometry-cnn/geometry_cnn.pt')) #Location of pret
pretrained_model.eval()
```

In [ ]:
```
classes = ['Circle', 'Heptagon', 'Hexagon', 'Nonagon', 'Octagon', 'Pentagon', 'Square', 'Star', 'Triangle']

transform = transforms.Compose([
transforms.Grayscale(num_output_channels=1),
transforms.ToTensor(),
transforms.Normalize((0.1307,), (0.3081,))])

path_val = '/kaggle/input/validation/val_set'
for i in os.listdir(path_val):
    img = Image.open(os.path.join(path_val,i))
    image = transform(img)
    image = image.unsqueeze(0)
    output = pretrained_model(image)
    print(i,":",classes[output.argmax()])
```

In [ ]:
```

```