Name: Advait Pai

REPORT

Image downscaling:

For the purpose of performance, I did some preprocessing on the image:-

1. Mean normalised the image with mean 0.5 and std dev 0.2.
2. Resizing from 200x200 into 50x50

Things that did not work:

When testing I made these initial parameters –

```
# Default Model Parameters
batch_size = 100
test_batch_size = 10000
epochs = 10
lr = 0.001
gamma = 0.09
seed = 2702
log_interval = 100
save_model = "store_true"
resize_size = 100
```

Initially I had created with the following basic structure:-

```
self.conv1 = nn.Conv2d(in_channels = 1,out_channels = 4,kernel_size = 4, stride = 1)
        self.conv2 = nn.Conv2d(in_channels = 4,out_channels = 9,kernel_size = 4, stride = 1)
        self.conv3 = nn.Conv2d(in_channels = 9,out_channels = 32,kernel_size = 2, stride = 1) # New added
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(3200, 256)
        self.fc2 = nn.Linear(256, 9)
```

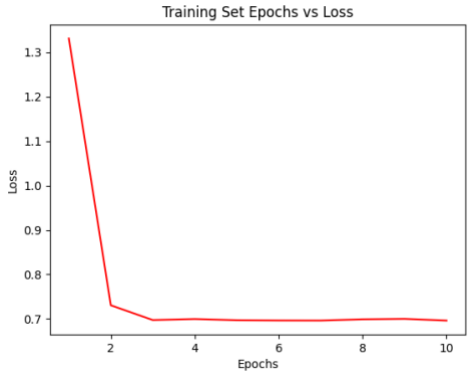
And while I tried different permutations and combinations to creating my network, like changing the in_channels and out_channels or the kernel size, I was never able to cross training accuract 37%.
I also tried to change the Linear Layer outputs to 1600 but that only increased the accuracy to 42%. I think this maybe due to oversimplistic assumptions I have made about the network. I then increased the in_channels and out_channels as well as added one more convuluted layer as well. I also made a change to the kernel size.

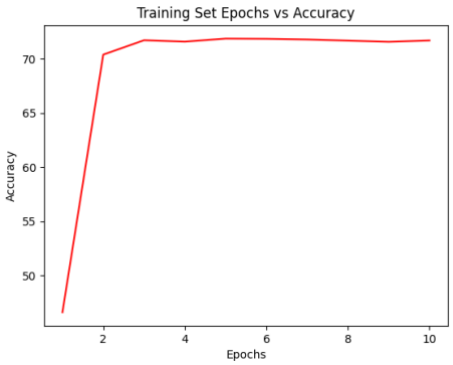
During testing for various hyperparameters, I found out that my gamma value which was initially 0.09 was impeding my learning by not updating weights after a certain number of training samples in a particular epoch, thus I also thought of changing that to 0.01, 0.02 and 0.03 and then I found out 0.03 giving me a better accuracy so I chose that. For a random choice, I even took 0.07

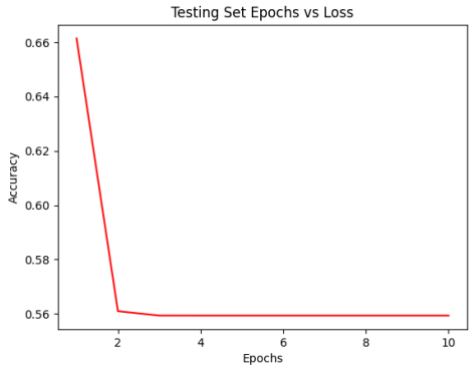
My initial resize of 100x100 was slowing down the learning process significantly, but I decided to finally go with a model with the following default parameters:

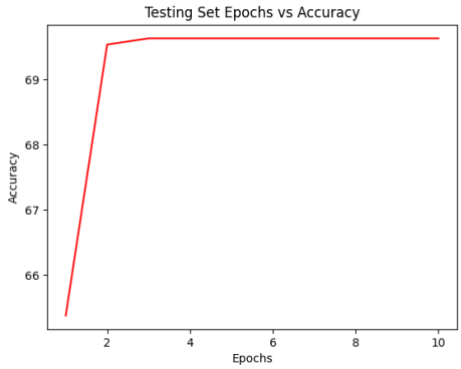
```
# Default Model Parameters
batch_size = 10
test_batch_size = 10000
epochs = 10
lr = 0.001
gamma = 0.02
seed = 2702
log_interval = 100
save_model = "store_true"
resize_size = 50
save_name = "0602-677368201-Pai.pt"
```

With this model that I have built, I am unable to see significant learning beyond the 3rd epoch itself, yet I have made it run for 10 epochs. The resulting graphs are given below:









(The Y-axis is Loss instead of accuracy)
After running for epochs, we see that training accuracy is at 71% and testing accuracy is at 69.25%

Design of the network:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3,out_channels = 9,kernel_size = 3, stride = 1)
        self.conv2 = nn.Conv2d(in_channels = 9,out_channels = 18,kernel_size = 3, stride = 1)
        self.conv3 = nn.Conv2d(in_channels = 18,out_channels = 27,kernel_size = 3, stride = 1)
        self.conv4 = nn.Conv2d(in_channels = 27,out_channels = 36,kernel_size = 3, stride = 1)


        self.dropout1 = nn.Dropout(0.2)
        self.dropout2 = nn.Dropout(0.2)
        self.fc1 = nn.Linear(576, 144)
        self.fc2 = nn.Linear(144, 9)
    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.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv4(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)
        return x
```

Code for Part 1:

```
# BEGIN CODE TO EXTRACT DATASET#
# import os
# import shutil
# files_list = os.listdir("output") # Get all files
# sorted_list = sorted(files_list)
```

```
# classes = set([x[:x.index("_")] for x in sorted_list]) # Code to get all classes

# training_filelist = dict()
# testing_filelist = dict()
# for c in classes:
#     # Creating a dictionary of sorted files with the labels
#     training_filelist[c] = [x for x in sorted_list if c in x][:8000]
#     testing_filelist[c] = [x for x in sorted_list if c in x][8000:]

# # Code to create the dataset -> We output this data to the Dataset folder
# train_path = "dataset/train/"
# test_path = "dataset/test/"
# for k,v in training_filelist.items(): # Loop over the classes
#     src = "output/"
#     dest = train_path+k+"/"
#     os.makedirs(dest)
#     for f in v: # Loop over individual files in the class
#         shutil.copy(src+f,dest+f)

# for k,v in testing_filelist.items(): # Loop over the classes
#     src = "output/"
#     dest = test_path+k+"/"
#     os.makedirs(dest)
#     for f in v: # Loop over individual files in the class
#         shutil.copy(src+f,dest+f)

# # Code to verify all dataset created correctly
# print ("Files Created! Verified Below")
# for c in classes:
#     print ("Images for",c)
#     print("Train Images:",len(os.listdir(train_path+c)))
#     print("Test Images:",len(os.listdir(test_path+c)))
#     print("")

# END CODE TO EXTRACT FILES #


# BEGIN CODE TO CREATE MODEL AND BUILD MODEL #

import torch
```

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

# Default Model Parameters
batch_size = 10
test_batch_size = 10000
epochs = 10
lr = 0.001
gamma = 0.02
seed = 2702
log_interval = 100
save_model = "store_true"
resize_size = 50
save_name = "0602-677368201-Pai.pt"

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3,out_channels = 9,kernel_size = 3, stride = 1)
        self.conv2 = nn.Conv2d(in_channels = 9,out_channels = 18,kernel_size = 3, stride = 1)
        self.conv3 = nn.Conv2d(in_channels = 18,out_channels = 27,kernel_size = 3, stride = 1)
        self.conv4 = nn.Conv2d(in_channels = 27,out_channels = 36,kernel_size = 3, stride = 1)

        self.dropout1 = nn.Dropout(0.2)
        self.dropout2 = nn.Dropout(0.2)
        self.fc1 = nn.Linear(576, 144)
        self.fc2 = nn.Linear(144, 9)
    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.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
```

```
        x = self.conv4(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)
        return x


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)
        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 % log_interval == 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)*batch_size)))

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

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

def main():

    # Training settings
    training_loss = []
    testing_loss = []
    training_accuracy = []
    testing_accuracy = []
    print(batch_size)
    torch.manual_seed(seed)
    #device = "cpu"
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Resize(resize_size),
        #transforms.Grayscale(),
        # transforms.Normalize((0,), (1,))
        transforms.Normalize((0.5), (0.2))
        ])

    dataset1 = datasets.ImageFolder('./dataset/train/',transform=transform)
    dataset2 = datasets.ImageFolder('./dataset/test/', transform=transform)
```

```
train_loader = torch.utils.data.DataLoader(dataset1, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset2, batch_size=test_batch_size)

model = Net().to(device)
optimizer = optim.Adam(model.parameters(), lr=lr)

scheduler = StepLR(optimizer, step_size=1, gamma=gamma)
for epoch in range(1, epochs + 1):
    loss1,accuracy1 = train(model, device, train_loader, optimizer, epoch)
    loss2, accuracy2 = test(model, device, test_loader)
    training_loss.append(loss1)
    testing_loss.append(loss2)
    training_accuracy.append(accuracy1)
    testing_accuracy.append(accuracy2)
    scheduler.step()

# Training Set - Epoch vs Loss
plt.title("Training Set Epochs vs Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot([i for i in range(1,epoch+1)],training_loss,color="red")
plt.show()
# Training Set - Epoch vs Accuracy
plt.title("Training Set Epochs vs Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.plot([i for i in range(1,epoch+1)],training_accuracy,color="red")
plt.show()

# Testing Set - Epoch vs Loss
plt.title("Testing Set Epochs vs Loss")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.plot([i for i in range(1,epoch+1)],testing_loss,color="red")
plt.show()

# Testing Set - Epoch vs Accuracy
plt.title("Testing Set Epochs vs Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
```

```
    plt.plot([i for i in range(1,epoch+1)],testing_accuracy,color="red")
    plt.show()


    if save_model:
        torch.save(model.state_dict(), save_name)


if __name__ == '__main__':
    main()

# END CODE TO BUILD MODEL #
```

Part 2 Code:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR



seed = 2702
save_name = "0602-677368201-Pai.pt"
resize_size = 50

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels = 3,out_channels = 9,kernel_size = 3, stride = 1)
        self.conv2 = nn.Conv2d(in_channels = 9,out_channels = 18,kernel_size = 3, stride = 1)
        self.conv3 = nn.Conv2d(in_channels = 18,out_channels = 27,kernel_size = 3, stride = 1)
        self.conv4 = nn.Conv2d(in_channels = 27,out_channels = 36,kernel_size = 3, stride = 1)

        self.dropout1 = nn.Dropout(0.2)
        self.dropout2 = nn.Dropout(0.2)
        self.fc1 = nn.Linear(576, 144)
```

```
        self.fc2 = nn.Linear(144, 9)
    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.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv4(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)
        return x

def main():

    # Training settings

    torch.manual_seed(seed)

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    load_model = torch.load(save_name,map_location=device)
    model = Net().to(device)
    model.load_state_dict(load_model)
    model.eval()
    # optimizer = optim.Adam(model.parameters(), lr=lr)

    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Resize(resize_size),
        transforms.Normalize((0.5), (0.2))
        ])
```

```
    dataset2 = datasets.ImageFolder('./test_images/', transform=transform)

    test_loader = torch.utils.data.DataLoader(dataset2,batch_size=1)

    tot_loss = 0
    correct = 0
    with torch.no_grad():
        classes = ['Circle','Heptagon','Hexagon','Nonagon','Octagon','Pentagon','Square','Star','Triangle']
        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(dataset2.imgs,classes[pred.item()])
    total_loss = tot_loss/(len(test_loader))
    total_accuracy = 100.0*correct/(len(test_loader)*1)
    print('Test Loss: {:.6f}, Test Accuracy: {:.2f}%'.format(
        total_loss,total_accuracy ))

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

Sample IO: (10 circle images were passed to the code)

Circle
Circle
Circle
Circle
Circle
Circle
Octagon
Circle
Circle
Circle
Octagon
Circle
Octagon
Circle
Circle
Test Loss: 0.609818, Test Accuracy: 80.00%