CSCI946 Assignment

Yao Xiao
SID 2019180015

December 9, 2020

1 Describe this MNIST data set and its training and test subsets.

The MNIST database of handwritten digits consists of a training set of 60,000 examples, and a test set of 10,000 examples. The training set and test set are collected from 250 different people's handwritten numbers. It is a subset of a larger set available from NIST. Additionally, the black and white images from NIST were size-normalized and centered to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

2 Describe how you reshape each image into a long vector and how you train the LRC or SVM.

First use numpy vectors to represent image data. The image data is grayed out and then stored in a numpy array. At this time, the value of each element is between 0 and 255. Next, we convert it to grayscale, map it to the range of 0 to 1, round each element to make its value 0 or 1, and then convert it to a binary matrix. Finally, convert the 20 x 20 binary matrix into a 28 x 28 vector.

```
    def convert2vec(image):
        img = Image.open(image).convert('L')
        arr_img = np.arry(img, 'i')
        img_normlization = np.round(arr_img / 255)
        img_arr = np.reshape(img_normlization, (1, -1))
```

However, in our project, we can directly load the data using the set image input size. Using LRC model training, we can obtain better classification accuracy by adjusting the parameters multiple times.

```
input_size = 784
num_classes = 10
num_epochs = 50
batch_size = 64
learning_rate = 0.01

train_dataset = dsets.MNIST(root='mnist',
                            train=True,
                            transform=transforms.ToTensor(),
                            download=True)
```

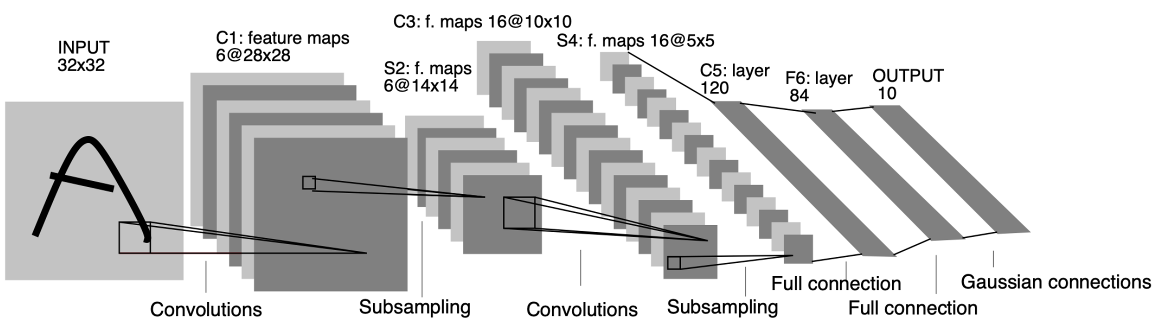
```
test_dataset = dsets.MNIST(root='mnist',
                           train=False,
                           transform=transforms.ToTensor(),
                           download=True)

# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=True)

# LRC model
model = nn.Linear(input_size, num_classes).cuda()
```

3 Describe how you designed your CNN, justify your approach, and describe how you trained it. Explain which other settings of the network parameters and/or training parameters that you tried, and describe the changes on classification accuracy and training time.

Figure 1: Structure of LeNet



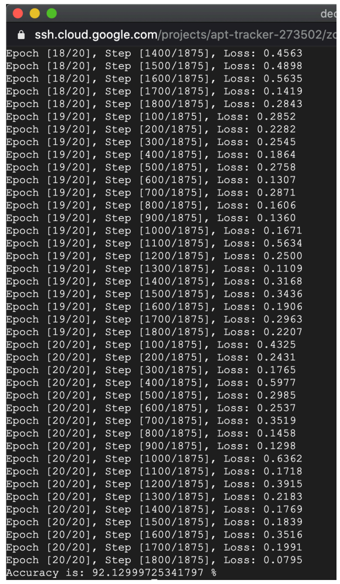
LeNet is the most classic convolutional neural network designed for handwritten digit recognition, and is known as one of the most representative experimental systems in the early convolutional neural networks. LeNet uses convolution, parameter sharing, pooling and other operations to extract features through clever design, avoiding a large amount of computational cost, and finally uses a fully connected neural network for classification and recognition. This network is also the starting point of a large number of neural network architectures recently. The LeNet model has 8 layers, Input 32*32 handwritten font pictures, these handwritten fonts contain 0-9 numbers, which is equivalent to 10 categories of pictures, and output the classification results, a number between 0-9.

For the LRC model:

In this model, there are there main attributes adjusted: learning rate, batch size and epoch times.

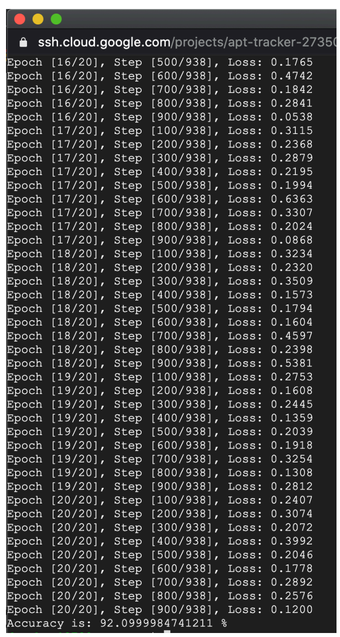
The experiment 1 settings: $lr = 0.1$, $batchsize = 32$, $epoch = 20$:

Figure 2: EXP1 for LRC



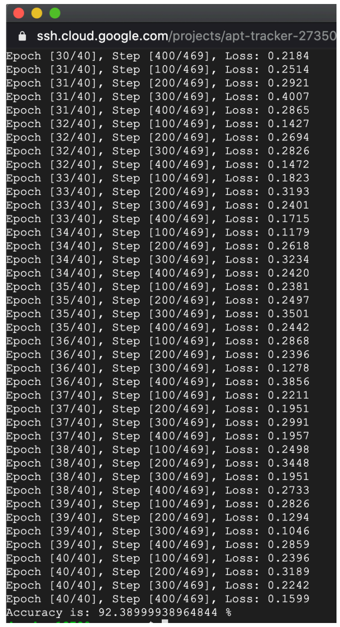
The experiment 2 settings: $lr = 0.3$, $batchsize = 64$, $epoch = 20$:

Figure 3: EXP2 for LRC



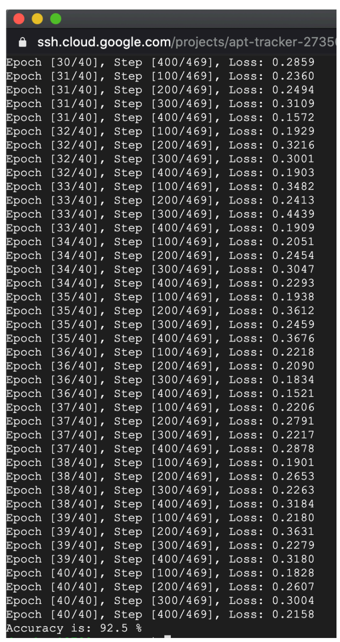
The experiment 3 settings: $lr = 0.1$, $batchsize = 128$, $epoch = 40$:

Figure 4: EXP3 for LRC



The experiment 4 settings: $lr = 0.5$, $batchsize = 128$, $epoch = 40$:

Figure 5: EXP4 for LRC

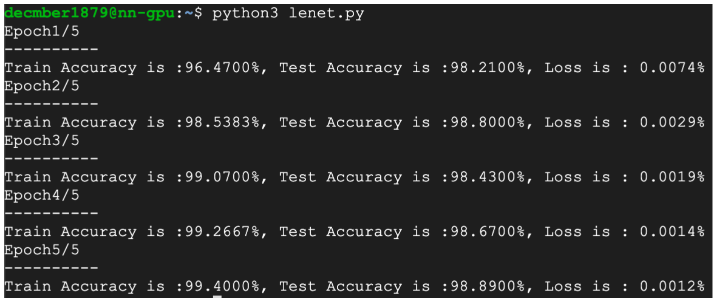


For the base LeNet model:

In this model, there are there main attributes adjusted: batch size and epoch times.

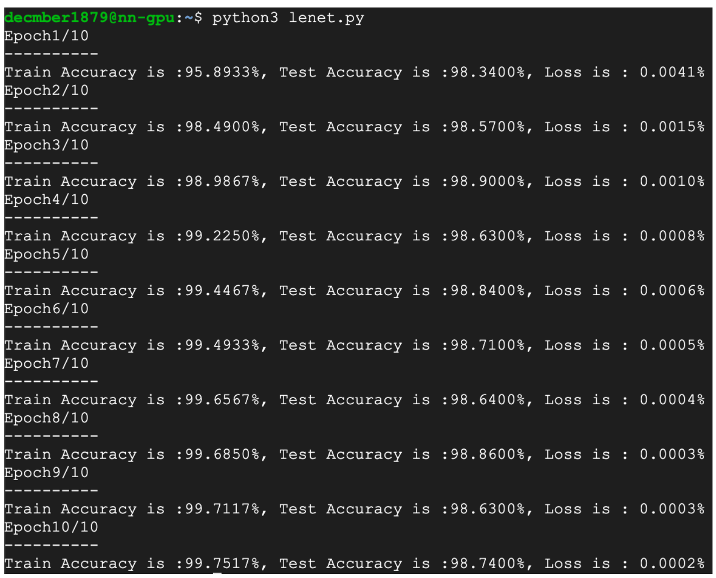
The experiment 1 settings: $batchsize = 16$, $epoch = 5$:

Figure 6: EXP1 for LeNet



The experiment 2 settings: $batchsize = 32$, $epoch = 10$:

Figure 7: EXP2 for LeNet



The experiment 3 settings: $batchsize = 64$, $epoch = 20$:

Figure 8: EXP3 for LeNet

```
----------
Train Accuracy is :99.8650%, Test Accuracy is :98.8700%, Loss is : 0.0001%
Epoch13/20
----------
Train Accuracy is :99.8133%, Test Accuracy is :98.9400%, Loss is : 0.0001%
Epoch14/20
----------
Train Accuracy is :99.8217%, Test Accuracy is :98.7600%, Loss is : 0.0001%
Epoch15/20
----------
Train Accuracy is :99.8317%, Test Accuracy is :98.9100%, Loss is : 0.0001%
Epoch16/20
----------
Train Accuracy is :99.8650%, Test Accuracy is :99.0200%, Loss is : 0.0001%
Epoch17/20
----------
Train Accuracy is :99.8517%, Test Accuracy is :99.1200%, Loss is : 0.0001%
Epoch18/20
----------
Train Accuracy is :99.8850%, Test Accuracy is :99.1000%, Loss is : 0.0001%
Epoch19/20
----------
Train Accuracy is :99.8550%, Test Accuracy is :99.0600%, Loss is : 0.0001%
Epoch20/20
----------
Train Accuracy is :99.8867%, Test Accuracy is :98.9500%, Loss is : 0.0001%
```

4 Report the best classification accuracy and the corresponding confusion matrices obtained by the classification methods (LRC, SVM, and the CNNs). Evaluate and compare the classification performances. Analyse and explain the results.

We can see the above, the best accuracy result of LRC model is: 92.5%, in addition, the adjustment results of each round are improved, and EXP1 can obtain the lowest loss value.

The best accuracy result of LeNet model is: 98.95%, although there are fluctuations, with the increase of batch sizh and epoch time, the result has also increased correspondingly, and the loss value has also achieved the lowest.

In summary, the larger the batch size and epoch time, the better the performance of CNN.

5 Source Code

```python
# Code for LRC
import torch
import torch.nn as nn
import torchvision.datasets as dsets
import torchvision.transforms as transforms
from torch.autograd import Variable

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

input_size = 784
num_classes = 10
num_epochs = 50
batch_size = 64
learning_rate = 0.01

```

```python
train_dataset = dsets.MNIST(root='mnist',
                            train=True,
                            transform=transforms.ToTensor(),
                            download=True)
test_dataset = dsets.MNIST(root='mnist',
                           train=False,
                           transform=transforms.ToTensor(),
                           download=True)

# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=True)

# LRC model
model = nn.Linear(input_size, num_classes).cuda()
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

# Train
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Reshape images
        images = Variable(images.view(-1, 28 * 28).cuda())
        labels = Variable(labels.cuda())
        outputs = model(images.cuda())
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if (i + 1) % 100 == 0:
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(
                epoch + 1, num_epochs, i + 1, total_step, loss.item()))

# Test
with torch.no_grad():
    correct = 0
    total = 0
for images, labels in test_loader:
    images = Variable(images.view(-1, 28 * 28).cuda())
    labels = Variable(labels.cuda())
    outputs = model(images.cuda())
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
```

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

print('Accuracy is: {} %'.format(100 * correct / total))
```

```
# Code for LeNet
import torch
from torchvision import transforms
import torchvision.datasets as dsets
from torch.autograd import Variable

batch_size = 16
n_epochs = 5
train_dataset = dsets.MNIST(root='mnist',
                            train=True,
                            transform=transforms.ToTensor(),
                            download=True)
test_dataset = dsets.MNIST(root='mnist',
                           train=False,
                           transform=transforms.ToTensor(),
                           download=True)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=True)


class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(), torch.nn.MaxPool2d(stride=2, kernel_size=2))
        self.dense = torch.nn.Sequential(torch.nn.Linear(14 * 14 * 128, 1024),
                                         torch.nn.ReLU(),
                                         torch.nn.Dropout(p=0.5),
                                         torch.nn.Linear(1024, 10))

    def forward(self, x):
        x = self.conv1(x)
        x = x.view(-1, 14 * 14 * 128)
        x = self.dense(x)
```

```
        return x


model = Model()
if torch.cuda.is_available():
    model.cuda()

cost = torch.nn.CrossEntropyLoss()
optimzer = torch.optim.Adam(model.parameters())

for epoch in range(n_epochs):
    running_loss = 0.0
    running_correct = 0
    print('Epoch{}/{}'.format(epoch + 1, n_epochs))
    print('-' * 10)

    for data in train_loader:
        X_train, y_train = data
        X_train, y_train = X_train.cuda(), y_train.cuda()
        X_train, y_train = Variable(X_train), Variable(y_train)
        outputs = model(X_train)
        _, pred = torch.max(outputs.data, 1)
        optimzer.zero_grad()
        loss = cost(outputs, y_train)
        loss.backward()
        optimzer.step()
        running_loss += loss.item()
        running_correct += torch.sum(pred == y_train.data)

    testing_correct = 0

    for data in test_loader:
        X_test, y_test = data
        X_test, y_test = X_test.cuda(), y_test.cuda()
        X_test, y_test = Variable(X_test), Variable(y_test)
        outputs = torch.mode(X_test)
        _, pred = torch.max(outputs, 1)
        testing_correct += torch.sum(pred == y_test.data)

    print(
        "Train Accuracy is :{:.4f}%, Test Accuracy is :{:.4f}%, Loss is : {:.4f}%"
        .format(100 * running_correct / len(train_dataset),
                100 * testing_correct / len(test_dataset),
                running_loss / len(train_dataset)))
```