

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.

```
1 def convert2vec(image):
2     img = Image.open(image).convert('L')
3     arr_img = np.array(img, 'i')
4     img_normlization = np.round(arr_img / 255)
5     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.

```
1 input_size = 784
2 num_classes = 10
3 num_epochs = 50
4 batch_size = 64
5 learning_rate = 0.01
6
7 train_dataset = datasets.MNIST(root='mnist',
8                                train=True,
9                                transform=transforms.ToTensor(),
10                               download=True)
```

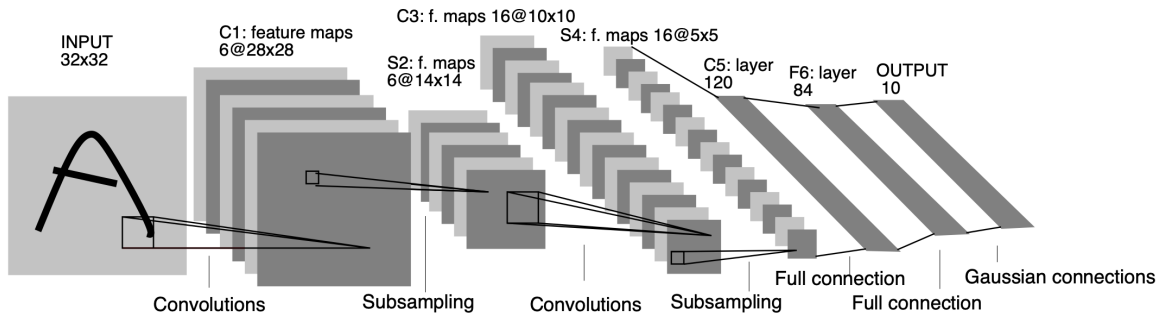
```

11 test_dataset = datasets.MNIST(root='mnist',
12                               train=False,
13                               transform=transforms.ToTensor(),
14                               download=True)
15
16 # Data loader
17 train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
18                                             batch_size=batch_size,
19                                             shuffle=True)
20 test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
21                                           batch_size=batch_size,
22                                           shuffle=True)
23
24 # LRC model
25 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



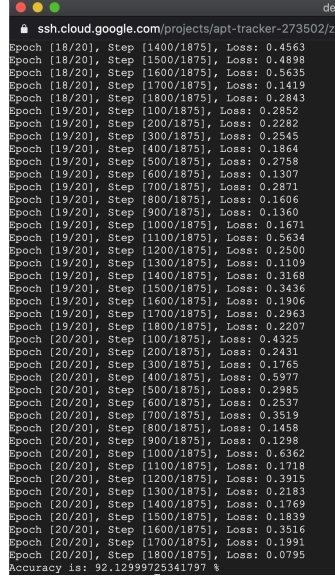
LeNet is the most classic convolutional neural network designed for handwritten digit recognition, and is known as one of the most representative experimental systems in the early convolutional neural networks. LeNet uses convolution, parameter sharing, pooling and other operations to extract features 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: $lr = 0.1$, $batchsize = 32$, $epoch = 20$:

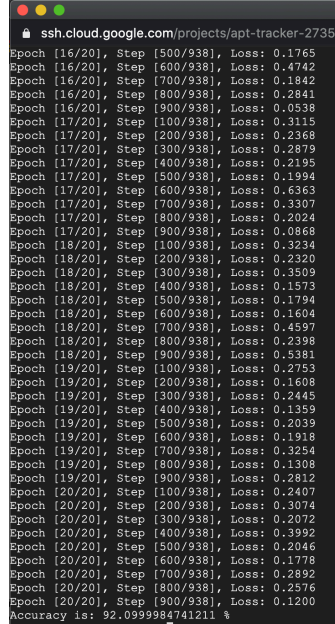
Figure 2: EXP1 for LRC



```
ssh.cloud.google.com/projects/apt-tracker-273502/zc
Epoch [18/20], Step [1400/1875], Loss: 0.4563
Epoch [18/20], Step [1500/1875], Loss: 0.4898
Epoch [18/20], Step [1600/1875], Loss: 0.5638
Epoch [18/20], Step [1700/1875], Loss: 0.1819
Epoch [18/20], Step [1800/1875], Loss: 0.2843
Epoch [19/20], Step [100/1875], Loss: 0.2852
Epoch [19/20], Step [200/1875], Loss: 0.2282
Epoch [19/20], Step [300/1875], Loss: 0.2545
Epoch [19/20], Step [400/1875], Loss: 0.1864
Epoch [19/20], Step [500/1875], Loss: 0.2758
Epoch [19/20], Step [600/1875], Loss: 0.1307
Epoch [19/20], Step [700/1875], Loss: 0.2871
Epoch [19/20], Step [800/1875], Loss: 0.1606
Epoch [19/20], Step [900/1875], Loss: 0.1360
Epoch [19/20], Step [1000/1875], Loss: 0.1671
Epoch [19/20], Step [1100/1875], Loss: 0.5634
Epoch [19/20], Step [1200/1875], Loss: 0.2500
Epoch [19/20], Step [1300/1875], Loss: 0.1109
Epoch [19/20], Step [1400/1875], Loss: 0.3168
Epoch [19/20], Step [1500/1875], Loss: 0.3436
Epoch [19/20], Step [1600/1875], Loss: 0.1906
Epoch [19/20], Step [1700/1875], Loss: 0.2963
Epoch [19/20], Step [1800/1875], Loss: 0.2207
Epoch [20/20], Step [100/1875], Loss: 0.4325
Epoch [20/20], Step [200/1875], Loss: 0.2431
Epoch [20/20], Step [300/1875], Loss: 0.1765
Epoch [20/20], Step [400/1875], Loss: 0.5977
Epoch [20/20], Step [500/1875], Loss: 0.2985
Epoch [20/20], Step [600/1875], Loss: 0.2537
Epoch [20/20], Step [700/1875], Loss: 0.3519
Epoch [20/20], Step [800/1875], Loss: 0.1458
Epoch [20/20], Step [900/1875], Loss: 0.1298
Epoch [20/20], Step [1000/1875], Loss: 0.6362
Epoch [20/20], Step [1100/1875], Loss: 0.1718
Epoch [20/20], Step [1200/1875], Loss: 0.3915
Epoch [20/20], Step [1300/1875], Loss: 0.2183
Epoch [20/20], Step [1400/1875], Loss: 0.1769
Epoch [20/20], Step [1500/1875], Loss: 0.1839
Epoch [20/20], Step [1600/1875], Loss: 0.3516
Epoch [20/20], Step [1700/1875], Loss: 0.1991
Epoch [20/20], Step [1800/1875], Loss: 0.0795
Accuracy is: 92.12999725341797 %
```

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

Figure 3: EXP2 for LRC



```
ssh.cloud.google.com/projects/apt-tracker-273502/zc
Epoch [16/20], Step [500/938], Loss: 0.1765
Epoch [16/20], Step [600/938], Loss: 0.4742
Epoch [16/20], Step [700/938], Loss: 0.1842
Epoch [16/20], Step [800/938], Loss: 0.2841
Epoch [16/20], Step [900/938], Loss: 0.0538
Epoch [17/20], Step [100/938], Loss: 0.3115
Epoch [17/20], Step [200/938], Loss: 0.2368
Epoch [17/20], Step [300/938], Loss: 0.2879
Epoch [17/20], Step [400/938], Loss: 0.2195
Epoch [17/20], Step [500/938], Loss: 0.1994
Epoch [17/20], Step [600/938], Loss: 0.6363
Epoch [17/20], Step [700/938], Loss: 0.3307
Epoch [17/20], Step [800/938], Loss: 0.2024
Epoch [17/20], Step [900/938], Loss: 0.0868
Epoch [18/20], Step [100/938], Loss: 0.3234
Epoch [18/20], Step [200/938], Loss: 0.2320
Epoch [18/20], Step [300/938], Loss: 0.3509
Epoch [18/20], Step [400/938], Loss: 0.1573
Epoch [18/20], Step [500/938], Loss: 0.1794
Epoch [18/20], Step [600/938], Loss: 0.1604
Epoch [18/20], Step [700/938], Loss: 0.4597
Epoch [18/20], Step [800/938], Loss: 0.2398
Epoch [18/20], Step [900/938], Loss: 0.5381
Epoch [19/20], Step [100/938], Loss: 0.2753
Epoch [19/20], Step [200/938], Loss: 0.1608
Epoch [19/20], Step [300/938], Loss: 0.2445
Epoch [19/20], Step [400/938], Loss: 0.1359
Epoch [19/20], Step [500/938], Loss: 0.2039
Epoch [19/20], Step [600/938], Loss: 0.1918
Epoch [19/20], Step [700/938], Loss: 0.3254
Epoch [19/20], Step [800/938], Loss: 0.1308
Epoch [19/20], Step [900/938], Loss: 0.2812
Epoch [20/20], Step [100/938], Loss: 0.2407
Epoch [20/20], Step [200/938], Loss: 0.3074
Epoch [20/20], Step [300/938], Loss: 0.2072
Epoch [20/20], Step [400/938], Loss: 0.3992
Epoch [20/20], Step [500/938], Loss: 0.2046
Epoch [20/20], Step [600/938], Loss: 0.1778
Epoch [20/20], Step [700/938], Loss: 0.2892
Epoch [20/20], Step [800/938], Loss: 0.2576
Epoch [20/20], Step [900/938], Loss: 0.1200
Accuracy is: 92.0999984741211 %
```

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

Figure 4: EXP3 for LRC

```
ssh.cloud.google.com/projects/apt-tracker-27350
Epoch [30/40], Step [400/469], Loss: 0.2184
Epoch [31/40], Step [100/469], Loss: 0.2514
Epoch [31/40], Step [200/469], Loss: 0.2921
Epoch [31/40], Step [300/469], Loss: 0.4007
Epoch [31/40], Step [400/469], Loss: 0.2865
Epoch [32/40], Step [100/469], Loss: 0.1427
Epoch [32/40], Step [200/469], Loss: 0.2694
Epoch [32/40], Step [300/469], Loss: 0.2826
Epoch [32/40], Step [400/469], Loss: 0.1472
Epoch [33/40], Step [100/469], Loss: 0.1823
Epoch [33/40], Step [200/469], Loss: 0.3193
Epoch [33/40], Step [300/469], Loss: 0.2401
Epoch [33/40], Step [400/469], Loss: 0.1715
Epoch [34/40], Step [100/469], Loss: 0.1179
Epoch [34/40], Step [200/469], Loss: 0.2618
Epoch [34/40], Step [300/469], Loss: 0.3234
Epoch [34/40], Step [400/469], Loss: 0.2420
Epoch [35/40], Step [100/469], Loss: 0.2381
Epoch [35/40], Step [200/469], Loss: 0.2497
Epoch [35/40], Step [300/469], Loss: 0.3501
Epoch [35/40], Step [400/469], Loss: 0.2442
Epoch [36/40], Step [100/469], Loss: 0.2868
Epoch [36/40], Step [200/469], Loss: 0.2396
Epoch [36/40], Step [300/469], Loss: 0.1278
Epoch [36/40], Step [400/469], Loss: 0.3856
Epoch [37/40], Step [100/469], Loss: 0.2211
Epoch [37/40], Step [200/469], Loss: 0.1951
Epoch [37/40], Step [300/469], Loss: 0.2991
Epoch [37/40], Step [400/469], Loss: 0.1957
Epoch [38/40], Step [100/469], Loss: 0.2498
Epoch [38/40], Step [200/469], Loss: 0.3448
Epoch [38/40], Step [300/469], Loss: 0.1951
Epoch [38/40], Step [400/469], Loss: 0.2733
Epoch [39/40], Step [100/469], Loss: 0.2826
Epoch [39/40], Step [200/469], Loss: 0.1294
Epoch [39/40], Step [300/469], Loss: 0.1046
Epoch [39/40], Step [400/469], Loss: 0.2859
Epoch [40/40], Step [100/469], Loss: 0.2396
Epoch [40/40], Step [200/469], Loss: 0.3189
Epoch [40/40], Step [300/469], Loss: 0.2242
Epoch [40/40], Step [400/469], Loss: 0.1599
Accuracy is: 92.38999938964844 %
```

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

Figure 5: EXP4 for LRC

```
ssh.cloud.google.com/projects/apt-tracker-27350
Epoch [30/40], Step [400/469], Loss: 0.2859
Epoch [31/40], Step [100/469], Loss: 0.2360
Epoch [31/40], Step [200/469], Loss: 0.2494
Epoch [31/40], Step [300/469], Loss: 0.3109
Epoch [31/40], Step [400/469], Loss: 0.1572
Epoch [32/40], Step [100/469], Loss: 0.1929
Epoch [32/40], Step [200/469], Loss: 0.3216
Epoch [32/40], Step [300/469], Loss: 0.3001
Epoch [32/40], Step [400/469], Loss: 0.1903
Epoch [33/40], Step [100/469], Loss: 0.3482
Epoch [33/40], Step [200/469], Loss: 0.2413
Epoch [33/40], Step [300/469], Loss: 0.4439
Epoch [33/40], Step [400/469], Loss: 0.1909
Epoch [34/40], Step [100/469], Loss: 0.2051
Epoch [34/40], Step [200/469], Loss: 0.2454
Epoch [34/40], Step [300/469], Loss: 0.3047
Epoch [34/40], Step [400/469], Loss: 0.2293
Epoch [35/40], Step [100/469], Loss: 0.1938
Epoch [35/40], Step [200/469], Loss: 0.3612
Epoch [35/40], Step [300/469], Loss: 0.2459
Epoch [35/40], Step [400/469], Loss: 0.3676
Epoch [36/40], Step [100/469], Loss: 0.2218
Epoch [36/40], Step [200/469], Loss: 0.2090
Epoch [36/40], Step [300/469], Loss: 0.1834
Epoch [36/40], Step [400/469], Loss: 0.1521
Epoch [37/40], Step [100/469], Loss: 0.2206
Epoch [37/40], Step [200/469], Loss: 0.2791
Epoch [37/40], Step [300/469], Loss: 0.2217
Epoch [37/40], Step [400/469], Loss: 0.2878
Epoch [38/40], Step [100/469], Loss: 0.1901
Epoch [38/40], Step [200/469], Loss: 0.2653
Epoch [38/40], Step [300/469], Loss: 0.2263
Epoch [38/40], Step [400/469], Loss: 0.3184
Epoch [39/40], Step [100/469], Loss: 0.2180
Epoch [39/40], Step [200/469], Loss: 0.3631
Epoch [39/40], Step [300/469], Loss: 0.2279
Epoch [39/40], Step [400/469], Loss: 0.3180
Epoch [40/40], Step [100/469], Loss: 0.1828
Epoch [40/40], Step [200/469], Loss: 0.2607
Epoch [40/40], Step [300/469], Loss: 0.3004
Epoch [40/40], Step [400/469], Loss: 0.2158
Accuracy is: 92.5 %
```

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

```
december1879@nn-gpu:~$ python3 lenet.py
Epoch1/5
-----
Train Accuracy is :96.4700%, Test Accuracy is :98.2100%, Loss is : 0.0074%
Epoch2/5
-----
Train Accuracy is :98.5383%, Test Accuracy is :98.8000%, Loss is : 0.0029%
Epoch3/5
-----
Train Accuracy is :99.0700%, Test Accuracy is :98.4300%, Loss is : 0.0019%
Epoch4/5
-----
Train Accuracy is :99.2667%, Test Accuracy is :98.6700%, Loss is : 0.0014%
Epoch5/5
-----
Train Accuracy is :99.4000%, Test Accuracy is :98.8900%, Loss is : 0.0012%
```

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

Figure 7: EXP2 for LeNet

```
december1879@nn-gpu:~$ python3 lenet.py
Epoch1/10
-----
Train Accuracy is :95.8933%, Test Accuracy is :98.3400%, Loss is : 0.0041%
Epoch2/10
-----
Train Accuracy is :98.4900%, Test Accuracy is :98.5700%, Loss is : 0.0015%
Epoch3/10
-----
Train Accuracy is :98.9867%, Test Accuracy is :98.9000%, Loss is : 0.0010%
Epoch4/10
-----
Train Accuracy is :99.2250%, Test Accuracy is :98.6300%, Loss is : 0.0008%
Epoch5/10
-----
Train Accuracy is :99.4467%, Test Accuracy is :98.8400%, Loss is : 0.0006%
Epoch6/10
-----
Train Accuracy is :99.4933%, Test Accuracy is :98.7100%, Loss is : 0.0005%
Epoch7/10
-----
Train Accuracy is :99.6567%, Test Accuracy is :98.6400%, Loss is : 0.0004%
Epoch8/10
-----
Train Accuracy is :99.6850%, Test Accuracy is :98.8600%, Loss is : 0.0003%
Epoch9/10
-----
Train Accuracy is :99.7117%, Test Accuracy is :98.6300%, Loss is : 0.0003%
Epoch10/10
-----
Train Accuracy is :99.7517%, Test Accuracy is :98.7400%, Loss is : 0.0002%
```

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 size 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

```
1 # Code for LRC
2 import torch
3 import torch.nn as nn
4 import torchvision.datasets as dsets
5 import torchvision.transforms as transforms
6 from torch.autograd import Variable
7
8 device = 0 if torch.cuda.is_available() else 'cpu'
9
10 input_size = 784
11 num_classes = 10
12 num_epochs = 50
13 batch_size = 64
14 learning_rate = 0.01
15
```

```

16 train_dataset = datasets.MNIST(root='mnist',
17                                train=True,
18                                transform=transforms.ToTensor(),
19                                download=True)
20 test_dataset = datasets.MNIST(root='mnist',
21                                train=False,
22                                transform=transforms.ToTensor(),
23                                download=True)
24
25 # Data loader
26 train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
27                                             batch_size=batch_size,
28                                             shuffle=True)
29 test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
30                                           batch_size=batch_size,
31                                           shuffle=True)
32
33 # LRC model
34 model = nn.Linear(input_size, num_classes).cuda()
35 # Loss and optimizer
36 criterion = nn.CrossEntropyLoss()
37 optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
38
39 # Train
40 total_step = len(train_loader)
41 for epoch in range(num_epochs):
42     for i, (images, labels) in enumerate(train_loader):
43         # Reshape images
44         images = Variable(images.view(-1, 28 * 28).cuda())
45         labels = Variable(labels.cuda())
46         outputs = model(images.cuda())
47         loss = criterion(outputs, labels)
48         optimizer.zero_grad()
49         loss.backward()
50         optimizer.step()
51         if (i + 1) % 100 == 0:
52             print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.
53                   format(
54                       epoch + 1, num_epochs, i + 1, total_step, loss.item()
55                   ))
56
57 # Test
58 with torch.no_grad():
59     correct = 0
60     total = 0
61     for images, labels in test_loader:
62         images = Variable(images.view(-1, 28 * 28).cuda())
63         labels = Variable(labels.cuda())
64         outputs = model(images.cuda())
65         _, predicted = torch.max(outputs.data, 1)
66         total += labels.size(0)

```

```

65         correct += (predicted == labels).sum()
66
67     print('Accuracy is: {}%'.format(100 * correct / total))

```

```

1  # Code for LeNet
2  import torch
3  from torchvision import transforms
4  import torchvision.datasets as dsets
5  from torch.autograd import Variable
6
7  batch_size = 16
8  n_epochs = 5
9  train_dataset = dsets.MNIST(root='mnist',
10                             train=True,
11                             transform=transforms.ToTensor(),
12                             download=True)
13  test_dataset = dsets.MNIST(root='mnist',
14                             train=False,
15                             transform=transforms.ToTensor(),
16                             download=True)
17
18  train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
19                                              batch_size=batch_size,
20                                              shuffle=True)
21  test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
22                                             batch_size=batch_size,
23                                             shuffle=True)
24
25
26  class Model(torch.nn.Module):
27      def __init__(self):
28          super(Model, self).__init__()
29          self.conv1 = torch.nn.Sequential(
30              torch.nn.Conv2d(1, 64, kernel_size=3, stride=1, padding
31                             =1),
32              torch.nn.ReLU(),
33              torch.nn.Conv2d(64, 128, kernel_size=3, stride=1,
34                             padding=1),
35              torch.nn.ReLU(), torch.nn.MaxPool2d(stride=2, kernel_
36                                                     size=2))
37          self.dense = torch.nn.Sequential(torch.nn.Linear(14 * 14 *
38                                                         128, 1024),
39                                          torch.nn.ReLU(),
40                                          torch.nn.Dropout(p=0.5),
41                                          torch.nn.Linear(1024, 10))
42
43      def forward(self, x):
44          x = self.conv1(x)
45          x = x.view(-1, 14 * 14 * 128)
46          x = self.dense(x)

```



```

43         return x
44
45
46 model = Model()
47 if torch.cuda.is_available():
48     model.cuda()
49
50 cost = torch.nn.CrossEntropyLoss()
51 optimizer = torch.optim.Adam(model.parameters())
52
53 for epoch in range(n_epochs):
54     running_loss = 0.0
55     running_correct = 0
56     print('Epoch{}/{}'.format(epoch + 1, n_epochs))
57     print('-' * 10)
58
59     for data in train_loader:
60         X_train, y_train = data
61         X_train, y_train = X_train.cuda(), y_train.cuda()
62         X_train, y_train = Variable(X_train), Variable(y_train)
63         outputs = model(X_train)
64         _, pred = torch.max(outputs.data, 1)
65         optimizer.zero_grad()
66         loss = cost(outputs, y_train)
67         loss.backward()
68         optimizer.step()
69         running_loss += loss.item()
70         running_correct += torch.sum(pred == y_train.data)
71
72     testing_correct = 0
73
74     for data in test_loader:
75         X_test, y_test = data
76         X_test, y_test = X_test.cuda(), y_test.cuda()
77         X_test, y_test = Variable(X_test), Variable(y_test)
78         outputs = torch.mode(X_test)
79         _, pred = torch.max(outputs, 1)
80         testing_correct += torch.sum(pred == y_test.data)
81
82     print(
83         "Train Accuracy is {:.4f}%, Test Accuracy is {:.4f}%,
84         Loss is {:.4f}%"
85         .format(100 * running_correct / len(train_dataset),
86               100 * testing_correct / len(test_dataset),
87               running_loss / len(train_dataset)))

```