华南理工大学

South China University of Technology

# The Experiment Report of Deep Learning

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

**SUBJECT:** SOFTWARE ENGINEERING

Author:
褚涛 陈奕男 朱昶熹

Supervisor:
Mingkui Tan

Student ID：
201710106550
201710106543
201720144894

Grade:
Graduate

December 22, 2017

# Handwritten digit recognition based on shallow neural network

**Abstract—The experiment *Handwritten digit recognition based on shallow neural network* is implemented using a LeNet5, which is a small CNN network using for simple image recognition such as handwritten digit recognition. The main step of the experiment includes the obtainment and load of the dataset, the construction of LeNet5 model, apply the loss backward method to optimize the parameters, and finally test the model and get the results.**

## I. INTRODUCTION

The motivation of the experiment includes:

i. Learn deep neural network construction process and use
ii. Understand deep learning framework pytorch and preliminary use
iii. Experience neural network training and testing process
iv. Deepen understanding of convolution, pooling, ReLu, fully connected and other network layers

The experiment is implemented using the LeNet5, which is a small CNN network using for simple image recognition such as handwritten digit recognition. LeNet5 is small but it contains the basic modules of deep learning including convolution layers, pooling layers and full connection layers, and can help us better understand the main structure of neural network, and the construction process of a deep neural network.

In Section II, the main principle of the LeNet5 is clarified, with the main components of it and the function of each layer. In Section III, the step of the experiment is stated with the sequence of the experiment requirements, following by the results of the experiment. And finally Section IV is the conclusion of this experiment.

## II. METHODS AND THEORY

The experiment is implemented using the LeNet5, which is a small CNN network using for simple image recognition such as handwritten digit recognition. Although it is small but it contains the basic modules of deep learning including convolution layers, pooling layers and full connection layers.

The convolution layer C1 uses 6 5x5 kernels to get 6 C1 image features. For C1, every digit in C1 is connected with 5x5 digits and 1 bias.

Then pooling layer S2 use a 2x2 kernel in pooling, with 5880 connections to C1 in total.

Then C3, S4 and C5 finished nearly the same jobs as C1 and S2, following these the layers, a full connection layer F6 is added, which has 84 bit nodes corresponding to a 7x12 bit-image. And finally, a 10 nodes output layer is added, which represent the 10 numeric digit from 0 to 9.

## III. EXPERIMENT

### 3.1 Dataset

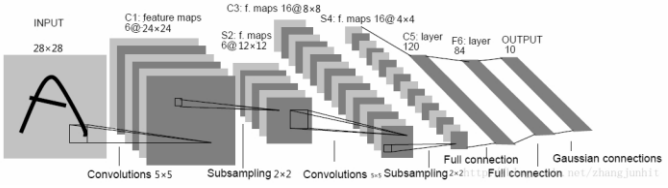
This experiment use handwritten digital data set - MNIST, which contains 60,000 hand-written digital images foe training and 10,000 hand-written digital images for validation. Each image is 28 × 28 pixels in size. Torchvision is used to download dataset and reading images.

### 3.2 Environment for Experiment

python3, including following python package: numpy, flask, pytorch，torchvision.

### 3.3 Experiment Step

Build shallow neural network LeNet5 (A class inheriting from nn.Module. You are supposed to at least implement init and forward function) in model.py. Network structure is as follows:




Use pytorch to build this network, detailed steps refer to PPT and pytorch documentation

```
#model.py
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()  # 图片输出尺寸(1, 28, 28)
        self.C1 = nn.Sequential(
            nn.Conv2d(1, 6, 5, 1),  # C1 卷积后尺寸 (6, 24, 24)
            nn.ReLU(),
        )
        self.S2 = nn.MaxPool2d(2) # S2 池化后尺寸 (6, 12, 12)
        self.C3 = nn.Sequential(
            nn.Conv2d(6, 16, 5, 1),  # C3 卷积后尺寸 (16,
```

```
8, 8)
            nn.ReLU(),
        )

        self.S4 = nn.MaxPool2d(2)  # S4 池化后尺寸 (16, 4, 4)
        self.C5 = nn.Linear(16 * 4 * 4, 120)  # 全连接层输出 (120,)
        self.F6 = nn.Linear(120, 84)  # 全连接层输出 (84,)
        self.out = nn.Linear(84, 10)  # 输出层为 10

    def forward(self, x):
        x = self.C1(x)
        x = self.S2(x)
        x = self.C3(x)
        x = self.S4(x)
        x = x.view(x.size(0), -1)  # 将三维 tensor 转为一维
        x = self.C5(x)
        x = self.F6(x)
        return self.out(x)
```

Complete the network training and validation process in train.py:

2.1 Use torch.utils.data.DataLoader, torchvision.datasets.MNIST and torchvision.transforms to load dataset.

```
#train.py
train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True,                                    # this is training data
    transform=torchvision.transforms.ToTensor(),    # Converts a PIL.Image or numpy.ndarray to
)

train_loader = Data.DataLoader(dataset=train_data, batch_size=BATCH_SIZE, shuffle=True)

test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)
test_x = Variable(torch.unsqueeze(test_data.test_data, dim=1), volatile=True).type(torch.FloatTensor)[:2000]/255.   # 将验证样本集尺寸从(2000, 28, 28)转为(2000, 1, 28, 28), 值域为(0,1)
test_y = test_data.test_labels[:2000]
```

2.2 Instantiate torch.optim.SGD to get the optimizer

2.3 Instantiate LeNet5 to get net.

```
#train.py
net = model.Model()
optimizer = torch.optim.SGD(net.parameters(), lr=LR)   # 实例化 SGD 优化器
```

2.4 Input data, which size is batch size, into the network for forward propagation to get predict target.

```
#train.py

for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):   # 取每个样本块训练
        b_x = Variable(x)
        b_y = Variable(y)
```

2.5 Use torch.nn.CrossEntropyLoss to calculate the loss between predict target and ground truth target.

```
#train.py

loss_func = nn.CrossEntropyLoss()                       # 定义 loss 函数
```

2.6 Calculate gradient using loss.backward and optimize the net using optimizer.step.

```
#train.py

for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):
        ...
        output = net(b_x)                # 模型输出
        loss = loss_func(output, b_y)    # 计算损失
        optimizer.zero_grad()            # 清除当前优化器梯度
        loss.backward()                  # 损失回传
        optimizer.step()                 # 应用梯度
        ...
```

2.7 Calculate accuracy of recognition.

```
#train.py

for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):
        ...
        if step % 50 == 0:
            test_output = net(test_x)   # 每 50 次计算分类精度
            pred_y = torch.max(test_output, 1)[1].data.squeeze()     # 验证样本预测值最大的标签
            accuracy = sum(pred_y == test_y) / float(test_y.size(0))     # 计算分类精度
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.data[0], '| test accuracy: %.2f' % accuracy)  # 输出分类精度
torch.save(net, 'net.pkl')  # 存储模型
        ...
```

2.8 Repeat step 2.4 to 2.7 until all data are inputed to the net.

2.9 Define the number of selected training epoch, repeat the training process 2.5--2.9

2.10 Save the best model as a .pkl file

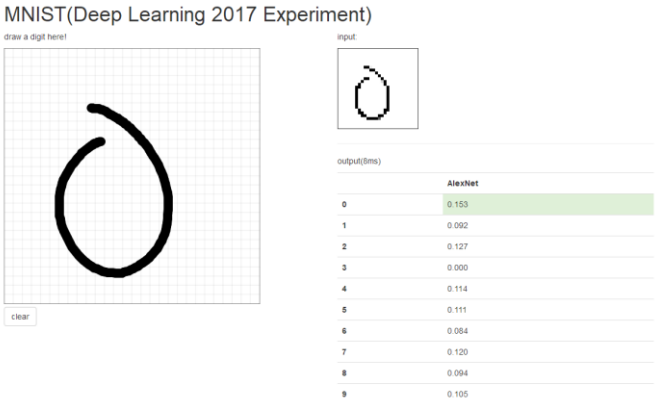
Compared with the training , there is no optimization process in validation. The other processes between training and validation are basically the same.

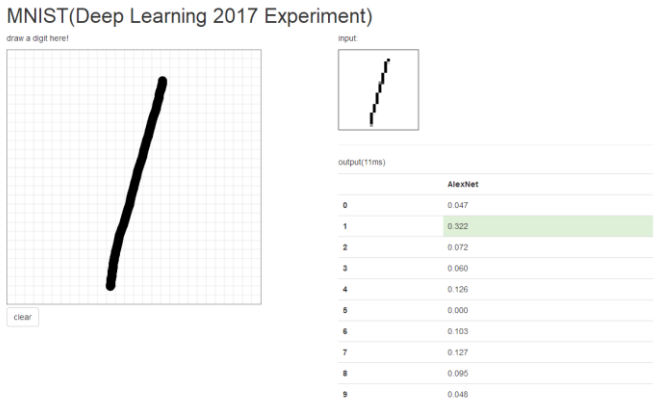
Load the trained model (.pkl file) into the main.py application. The implementation of main.py can test the

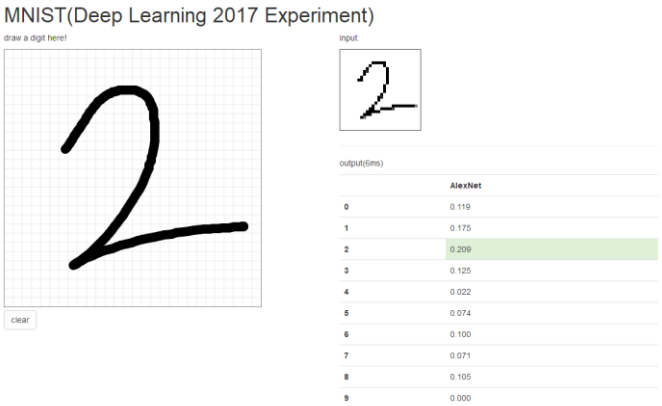
model's performance on the web page.

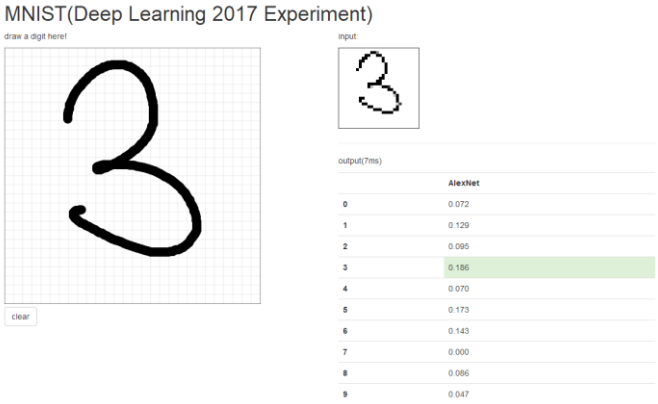
```
#main.py

# webapp
app = Flask(__name__)
net = torch.load('net.pkl')      # 载入训练好的模型

def predict_with_pretrain_model(sample):
    s = 1-Variable(
        torch.unsqueeze(
            torch.unsqueeze(torch.from_numpy(sample),
dim=0),
            dim=0),
        volatile=True
    ).type(torch.FloatTensor) / 255      # 将 28x28 的矩
阵扩展为 1x1x28x28 的形式，且为黑底白字
    r = net(s).data.numpy()[0]  # 对图像进行预测
    r = r-numpy.min(r)  # 将结果置为正
    r = r/numpy.sum(r)  # 归一化
    return r.tolist()

@app.route('/api/mnist', methods=['POST'])
def mnist():
    input = ((numpy.array(request.json,
dtype=numpy.uint8))).reshape(28, 28)
    output = predict_with_pretrain_model(input)
    return jsonify(results=output)


@app.route('/')
def main():
    return render_template('index.html')


if __name__ == '__main__':
    app.run()
```

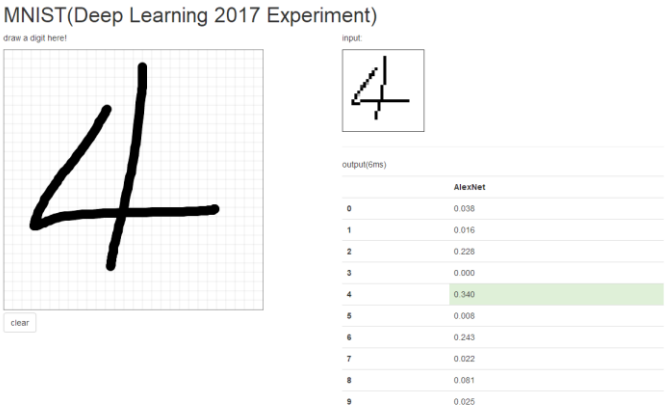
### 3.4 Result

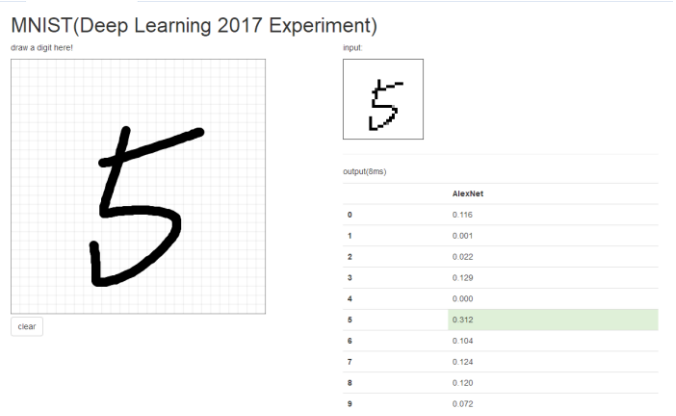


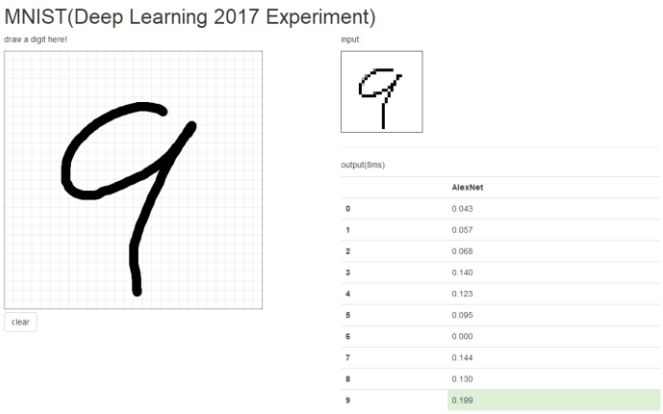



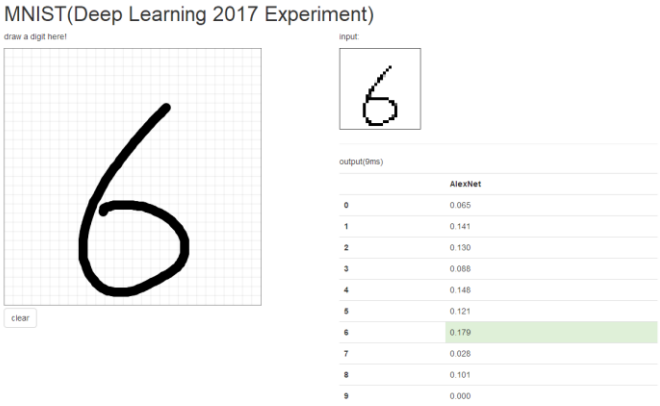



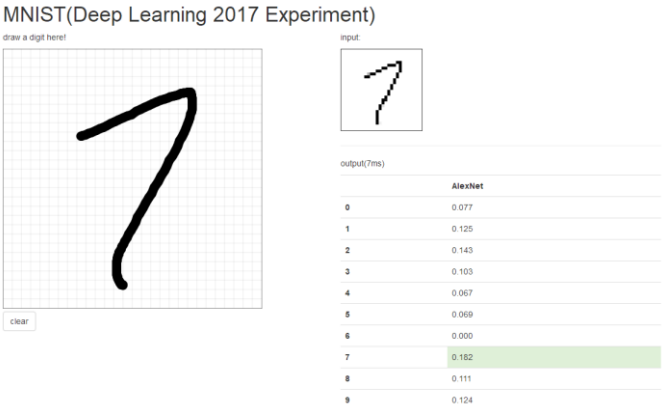



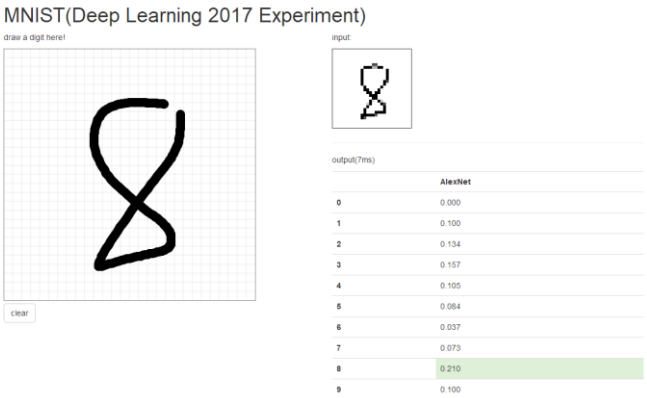






## IV. CONCLUSION

The experiment is implemented using a LeNet5, which is a small CNN network using for simple image recognition such as handwritten digit recognition.

Through this experiment, we mastered the basic steps of deep neural network construction process and use, and also have a deeper understand of neural network training and testing process.