1 实验内容及目的

本次实验旨在使用Matlab实现对图片的基本操作并且依据差值法实现对MNIST数据集的手写数字识别任务。

2 实验相关原理描述

2.1 Matlab涉及图像的基本操作

1. 读取和显示图像：使用 `imread` 函数读取图像，并使用 `imshow` 函数显示图像。图像在计算机中以矩阵形式存储，每个像素值表示图像上的一个位置的亮度或颜色值。
2. 调整图像大小：使用 `imresize` 函数调整图像大小。该函数可以缩小或放大图像，同时保持图像质量。
3. 裁剪图像：使用矩阵索引操作来裁剪图像。可以使用 `:` 操作符选择行或列，从而选择需要保留的图像区域。
4. 灰度转换：使用 `rgb2gray` 函数将 RGB 彩色图像转换为灰度图像。在灰度图像中，每个像素仅有一个亮度值，而在彩色图像中，每个像素具有三个颜色值（红色、绿色、蓝色）。
5. 二值图反转：使用 `imcomplement` 函数将二值图进行反转。该函数将输入图像中的每个像素值取反，即将亮度值为 0 的像素变为 1，将亮度值为 1 的像素变为 0，从而生成反转后的二值图像。

2.2 差值法

差值法是一种常用的图像处理技术，它的基本思想是将输入的数字图像与一组标准数字图像进行比较，找到最相似的标准数字图像，并将其与输入数字匹配。

具体实现中，差值法将每个数字图像表示为一个矩阵，该矩阵中的每个元素代表图像上一个像素的亮度值。然后，对于输入数字图像和每个标准数字图像，都计算它们之间的差异，并将它们的差异分数进行比较，以找到最相似的标准数字图像。该实验使用的差值法的具体公式如下：

$$Error = \sum_{i=1}^{\mathrm{h}} \sum_{j=1}^{w} |(img(i,j) - \mathrm{model}(i,j))| \tag{1}$$

上述公式中的 `img` 指待测图片，`model` 指模版图片，这里遍历二者的每一行每一列来统计差异像素的个数，最后求和，以此来表示二者之间的差距，差距最小的就是最为相似的数字图像。

3 实验过程

3.1 读入数据

我将每个数字的前900张作为模版图片，后100张作为待测图片。模版图片用10*900的元胞数组进行存放，待测图片用1*1000的元胞数组进行存放。

在读取数字图片后，依次进行以下操作：a) 将图片为切割为28像素*28像素； b) 灰度化； c) 二值化；d) 依据情况判断是否需要黑白反转。以下是完整代码。

```
data_dir = '/Users/LYU/Documents/MATLAB/图像处理实验/lesson1/number_recognize/train_dataset/';
% 每个数字都是前900张作为模版，后100张作为测试集
Image = cell(10, 900);
test_num = 1000;  % 待测图片数量
test_image = cell(1, test_num);  % 存储处理后的待测图片
```

```
% 读入数字0的模板图片
for num = 1:900
    img = imread([data_dir, '0/image' ,int2str(9000+num),'.png']);
    img = imresize(img, [28, 28]);  % 统一图片大小为28*28
    img = mat2gray(img);  % 灰度化
    img = imbinarize(img, 0.5);  % 二值化
    if mean(img(:)) > 0.5
        img = imcomplement(img);
    end
    img1 = deal(img);
    Image{1, num} = img1;
end

% 读入数字0的测试图片
for num = 901:1000
    img = imread([data_dir, '0/image' ,int2str(9000+num),'.png']);
    img = imresize(img, [28, 28]);
    img = mat2gray(img);
    img = imbinarize(img, 0.5);
    if mean(img(:)) > 0.5
        img = imcomplement(img);
    end
    img1 = deal(img);
    test_image{num - 900} = img1;
end

for index = 1:9
    % 读入数字1-9的模版图片
    for num = 1:900
        img = imread([data_dir, int2str(index), '/image' ,int2str(num +
(index-1)*1000),'.png']);
        img = imresize(img, [28, 28]);
        img = mat2gray(img);
        img = imbinarize(img, 0.5);
        if mean(img(:)) > 0.5
            img = imcomplement(img);
        end
        img1 = deal(img);
        Image{index+1, num} = img1;
    end
    % 读入数字1-9的测试图片
    for num = 901:1000
        img = imread([data_dir, int2str(index), '/image' ,int2str(num +(index-1)*1000),'.png']);
        img = imresize(img, [28, 28]);
        img = mat2gray(img);
        img = imbinarize(img, 0.5);
        if mean(img(:)) > 0.5
            img = imcomplement(img);
        end
        img1 = deal(img);
        test_image{index*100 + num - 900} = img1;
    end
```

```
end
```

3.2 差值法对比识别

该步骤的主要思想是，遍历待测图片，分别与所有模版图片进行匹配，选择最小的`error`对应的模版图片的索引作为识别出的标签，之后将该标签与待测图片的真实标签进行匹配，如果相同则证明分类正确，最后输出分类的准确率。

```
% 利用差值法对待测图片进行识别
acc = 0;  % 存储识别准确率
for num = 1:test_num
    % 计算待测图片与模板图片的差值，并找到最小差值对应的模板图片
    min_error = inf;  % 存储最小差值
    min_index = -1;
    for index = 1:10
        for i = 1:900
            error = sum(abs(test_image{num} - Image{index, i}), 'all');
            if error < min_error
                min_error = error;
                min_index = index-1;
            end
        end
    end
    % 比较最优匹配结果与待测图片真实标签，统计准确率
    if min_index == floor((num - 1) / 100)
        acc = acc + 1;
    end
end

acc = acc / test_num;  % 计算准确率
disp(['准确率为 ', num2str(acc)]);
```

3.3 实验结果

输出的结果为：**准确率为0.238。**

3.4 讨论

从实验结果来看，使用差值法实现手写数字识别的准确率比较低，可能的原因如下：1、使用传统方法完成一个十分类模型本身存在困难；2、待测图片可能在原先基础上进行了旋转，比如说一个1旋转了45度，人眼可以判断出这依然是1，但是计算机只能识别图片转换成的二值图矩阵与模版图片之间的差距，所以只有当有一张模版图片发生了相同的旋转时计算机才能识别出来。

4 拓展

在使用差值法完成实验后，我回想起之前复现过的LeNet神经网络[1]。LeNet是深度学习历史上的第一个卷积神经网络，由Yann LeCun等人在1998年提出，是用于手写数字识别的经典模型。它通过卷积层、池化层和全连接层的组合来提取图像特征并进行分类。下图是LeNet卷积神经网络的基本结构，可以看出，它仅包含了两个卷积层、两个池化层和两个全连接层，整个网络总共只有6层。

我基于PyTorch完成了LeNet对于MNIST数据集的手写数字识别的任务，经过100轮的训练，最后在测试集上的准确率为98.74%，损失为0.0074，远超过传统的差值法。这种优势来自以下两点原因：

1. 自适应特征学习：LeNet神经网络具备自适应特征学习的能力，它可以根据输入的手写数字图像，自动学习特征，并对特征进行提取和分类，从而提高了手写数字识别的准确率。

2. 鲁棒性：LeNet神经网络具有很强的鲁棒性，即对输入图像的小变化（例如旋转、平移、缩放等）有较好的容忍度，能够保持较好的识别准确率。

总之，相比于传统的差值法，LeNet神经网络在手写数字识别方面具有更好的性能和准确率，是一种更加优秀和有效的识别方法。

5 总结

通过本次实验，我了解了Matlab对于图片的基本操作，深刻体会到了Matlab对于矩阵运算的优势，并且使用差值法简单完成了手写数字识别任务，并对其有了更深刻的认识。

差值法是一种基于像素级别的图像匹配算法，它的优点包括：算法简单易懂，易于实现；计算速度快，对于小规模的图像匹配任务，速度较快；适用于对图像进行简单的处理（如二值化、灰度化）后的匹配，不需要对图像进行特征提取。

同时，差值法也存在一些缺点：对于大规模的图像匹配任务，计算时间较长；对于变形、旋转、光照等影响像素值的因素，匹配准确度较低；该算法对于像素变化较大的图像匹配效果较差。

最后，我将LeNet神经网络在MNIST数据集上的表现与差值法进行了对比，综合了二者各自的优势，这对我后续的学习有很大帮助

6 引用

[1] Handwritten Digit Recognition with a Back-Propagation Network, Yann LeCun, 1998

7 附录

7.1 Matlab差值法代码

```
clear
clc

data_dir = '/Users/LYU/Documents/MATLAB/图像处理实验/lesson1/number_recognize/train_dataset/';
% 每个数字都是前900张作为模版，后100张作为测试集
Image = cell(10, 900);
test_num = 1000;  % 待测图片数量
test_image = cell(1, test_num);  % 存储处理后的待测图片

for num = 1:900
    img = imread([data_dir, '0/image' ,int2str(9000+num),'.png']);
    img = imresize(img, [28, 28]);
    img = mat2gray(img);
    img = imbinarize(img, 0.5);
    if mean(img(:)) > 0.5
        img = imcomplement(img);
    end
    img1 = deal(img);
    Image{1, num} = img1;
end

for num = 901:1000
```

```
    img = imread([data_dir, '0/image' ,int2str(9000+num),'.png']);
    img = imresize(img, [28, 28]);
    img = mat2gray(img);
    img = imbinarize(img, 0.5);
    if mean(img(:)) > 0.5
        img = imcomplement(img);
    end
    img1 = deal(img);
    test_image{num - 900} = img1;
end

for index = 1:9
    for num = 1:900
        img = imread([data_dir, int2str(index), '/image' ,int2str(num +
(index-1)*1000),'.png']);
        img = imresize(img, [28, 28]);  % 统一图片大小为28*28
        img = mat2gray(img);  % 灰度化
        img = imbinarize(img, 0.5);  % 二值化
        if mean(img(:)) > 0.5
            img = imcomplement(img);
        end
        img1 = deal(img);
        Image{index+1, num} = img1;
    end
    for num = 901:1000
        img = imread([data_dir, int2str(index), '/image' ,int2str(num +(index-1)*1000),'.png']);
        img = imresize(img, [28, 28]);  % 统一图片大小为28*28
        img = mat2gray(img);  % 灰度化
        img = imbinarize(img, 0.5);  % 二值化
        if mean(img(:)) > 0.5
            img = imcomplement(img);
        end
        img1 = deal(img);
        test_image{index*100 + num - 900} = img1;
    end
end

% 利用差值法对待测图片进行识别
acc = 0;  % 存储识别准确率
for num = 1:test_num
    % 计算待测图片与模板图片的差值，并找到最小差值对应的模板图片
    min_error = inf;  % 存储最小差值
    min_index = -1;
    for index = 1:10
        for i = 1:900
            error = sum(abs(test_image{num} - Image{index, i}), 'all');
            if error < min_error
                min_error = error;
                min_index = index-1;
            end
        end
    end
```

```
    % 比较最优匹配结果与待测图片真实标签，统计准确率
    if min_index == floor((num - 1) / 100)
        acc = acc + 1;
    end
end

acc = acc / test_num;  % 计算准确率
disp(['准确率为 ', num2str(acc)]);
```

7.2 LeNet手写数字识别（基于PyTorch）

```
import torch
import torchvision
import matplotlib.pyplot as  plt
import torch.nn  as  nn
import torch.nn.functional as F
import torch.utils.data as Data
import torchvision.transforms as transforms

class LeNet(nn.Module):
    def __init__(self, num_class=10):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, 1, 2)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    # 定义前向传播过程，输入为x
    def forward(self, x):
        x = F.relu(self.conv1(x))  # conv 28x28x1 -> 5x5 -> 28x28x6
        x = self.pool1(x)  # 28x28x6 -> 2x2 -> 14x14x6
        x = F.relu(self.conv2(x))  # 14x14x6 -> 5x5 -> 10x10x16
        x = self.pool2(x)  # 10x10x16 -> 2x2 -> 5x5x16

        x = x.view(x.size()[0], -1)  # 将多维的tensor数据展平成一维

        x = F.relu(self.fc1(x))  # 5x5x16 -> 120
        x = F.relu(self.fc2(x))  # 120 -> 84
        x = self.fc3(x)  # 84 -> 10
        return x

# 设置超参数
NUM_CLASSES = 10  # 10分类
EPOCHS = 10  # 训练轮次
BATCH_SIZE = 100  # 一轮训练批量大小
LR = 0.05  # 学习率learning_rate

train_dataset = torchvision.datasets.MNIST(root='./', train=True,
transform=transforms.ToTensor(), download=True)
```

```
test_dataset = torchvision.datasets.MNIST(root='./', train=False,
transform=transforms.ToTensor(), download=False)

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=False)

# 训练模型
total_step = len(train_loader)  # 总训练次数
Acc = list() # 存放准确率
Loss = list() # 存放损失
for epoch in range(EPOCHS):
    for i, (images, labels) in enumerate(train_loader):
        # 前向传播
        outputs = model(images)
        loss = criterion(outputs, labels)
        # 反向传播
        optimizer.zero_grad()  # 梯度置零
        loss.backward()  # loss反向传播计算梯度
        optimizer.step()  # 更新网络参数
        # 将每轮的损失和准确率输出
        if (i + 1) % 100 == 0:
            # 测试模型
            model.eval()  # 切换到评估模式而非训练模式即固定参数
            with torch.no_grad():
                correct = 0
                total = 0
                for images, labels in test_loader:
                    outputs = model(images)
                    _, predicted = torch.max(outputs.data, 1)
                    total += labels.size(0)  # 数据总量
                    correct += (predicted == labels).sum().item()  # 准确总数量
                Loss.append(float(loss))
                Acc.append(correct / total)
                print(f'epoch[{epoch + 1}/{EPOCHS}], step[{i + 1}/{total_step}], acc:{(100 *
correct / total)}%, loss:{loss.item():.4f}')
```