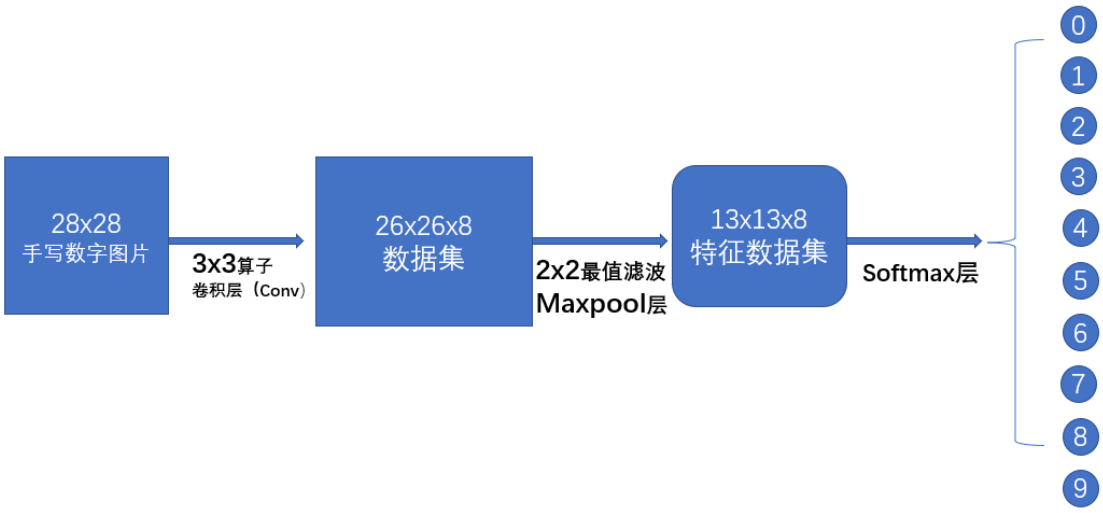
Key words：卷积神经网络，最值滤波，梯度下降法优化，Softmax 回归

1. 实验方案与步骤

2.1 方法原理

搭建三层神经网络，分别为卷积层，池化层，与带有 10 个结点的 Softmax 回归层。在 Softmax 变换后，概率最大的节点所代表的数字即此 CNN 模型的分类输出结果示意图如下：



示意图

卷积与池化层：直接原理均为提取有效特征值，减小程序计算量，卷积层初步提取特征值，池化层则放大部分特征值，并进一步压缩体积。

Softmax 层：Softmax 函数分类过程是 logistics 函数的多分类版本优化，主要通过将各个数值转换为概率大小进行分类。它将多个神经元的输出，映射到(0,1)区间内。假设我们有一个数组，V，Vi 表示 V 中的第 i 个元素，那么这个元素的 softmax 值表达式为：

$$S_i = \frac{e^i}{\sum_j e^j}$$

且各个元素概率和相加为 1。

在本实验中，Softmax 将神经网络输出的多维向量进行概率转换，以概率最大的那一类进行归类，量化对预测的确信程度。

同时，我们引入交叉熵损失函数的概念来考虑对每一个预测的正确性评估。（Pi 是第 i 类的预测概率）

$$L = -\ln\ (\mathrm{Pi})$$

同时 L 越小，正确度越高。

神经网络训练 分为两个阶段：

1. 前向阶段，输入完全通过神经网络；

```
out = conv.forward((image/255)-0.5)##像素值归一化【-0.5,0.5】
out = pool.forward(out)
out = softmax.forward(out)   ##概率计算结果
```

2. 后向阶段，其中梯度反向传播 (backprop)并更新权重。

```
grad = softmax.backprop(grad, study)
grad = pool.backprop(grad)
grad = conv.backprop(grad,study)
```

在前向阶段主要存入数据（输入图像等）。
在后向阶段，每一层将收到一个梯度，也返回一个梯度。即每一层梯度计算来自上一层返回值的相关梯度。它将接收关于其输出的损耗梯度并返回关于其输入的损耗梯度。（这里我们采用梯度下降法，主要思路公式如下）。

$$W' = W - learn_{rate} * \frac{\partial L}{\partial W}$$

整体优化流程如下：

（1） 从 Softmax 层入手，首先需要计算是输入到 Softmax 层的后向阶段，其中 out，是 Softmax 层的输出:10x1 的概率的向量，公式如下：

$$\frac{\partial L}{\partial out_s} = \begin{cases} 0, & \text{if } i \neq c \\ -\frac{1}{p_i}, & \text{if } i = c \end{cases}$$

（2） 由于输出层 Softmax 与池化层为全连接神经网络，我们需要的是相对于权重、偏差和输入的损失梯度。引入权重梯度$\frac{\partial L}{\partial W}$，偏差梯度$\frac{\partial L}{\partial b}$（代码中的 biases），通过 backprop（）函数的输入梯度$\frac{\partial L}{\partial input}$返回给下一层使用。计算这 3 个损失梯度，需要计算总的权重、偏差和输入的梯度。引入相关关系式$t = W * input + b.$

各梯度计算公式为（这部分内容参考 victorzhou 的博客）：

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial out}\frac{\partial out}{\partial t}\frac{\partial t}{\partial w} = input$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial out}\frac{\partial out}{\partial t}\frac{\partial t}{\partial b} = 1$$

$$\frac{\partial L}{\partial \text{ input}} = \frac{\partial L}{\partial out}\frac{\partial out}{\partial t}\frac{\partial t}{\partial \text{ input}} = w$$

接下来确定 out_(c)相对于 Softmax 函数传入的值的梯度，设t_i为 i 类的总和。公式可表达为：

$$out_s(c) = \frac{e^{t_c}}{\sum i\, e^{t_i}} = e^{t_c} S^{-1}$$

当 k≠c 时，利用链式法则$\frac{\partial out_s(c)}{\partial t_k} = \frac{\partial out_s(c)}{\partial S}\frac{\partial S}{\partial t_k} = -e^{t_c} S^{-2}\frac{\partial S}{\partial t_k} = \frac{-e^{t_c}e^{t_k}}{S^2}$

当 k=c 时，$\frac{\partial out_s(c)}{\partial t_c} = \frac{Se^{t_c} - e^{t_c}\frac{\partial S}{\partial t_c}}{S^2} = \frac{e^{t_c}(S - e^{t_c})}{S^2}$

其中，S = $\sum i\, e^{t_i}$

（3） 在 backprop（）操作中，池化层部分较为特殊，返回过程中将最值外回归的像素值大小均归零，便于求取相同矩阵规模的梯度。

（4） 在卷积层（Conv3x3）中我们需要求取$\frac{\partial L}{\partial filters}$，因为易知$\frac{\partial L}{\partial out}$与$\frac{\partial L}{\partial filters}$由数学公式推导我们可以通过$\frac{\partial L}{\partial filters} = \frac{\partial L}{\partial out} * \frac{\partial L}{\partial filters}$得到。

同时由资料及计算可以发现，一个特定的输出像素相对于一个特定的滤波器权重的导数只是相应的图像像素值，表达式如下：

$$\text{out}\ (i,j) = \text{conv}(\text{image},\ \text{filter}) = \sum_{x=0}^{2}\sum_{y=0}^{2} \text{image}\ (i+x, j+y) * \text{filter}\ (x,y)$$

$$\frac{\partial out(i,j)}{\partial\ \text{filter}\ (x,y)} = \ \text{image}\ (i+x, j+y)$$

所以，易得$\frac{\partial L}{\partial filters}$最终表达式为：

$$\frac{\partial L}{\partial\ \text{filter}\ (x,y)} = \sum_{i}\sum_{j} \frac{\partial L}{\partial \text{out}\,(i,j)} * \frac{\partial out(i,j)}{\partial\ \text{filter}\ (x,y)}$$

（5） 接着按照偏导数推导补齐各层中 backprop（）函数，并依照梯度下降法进行学习训练，提升模型鲁棒性。

（6） 解释说明：由于调用了较多成熟指令与库，部分原理解释可能存在一些谬误和欠缺。

2.2 实验方案

利用 mnist 数据库中的大量已经打好标签的手写数据图片进行训练，并进行正确性展示。

同时，再利用测试数据集进行测试与该 CNN 模型鲁棒性检验。

通过一层层模型搭建，逐步实现卷积神经网络分类模型。

以输出大量测试集的测试正确率来展现模型完成度与可视化。

2.3 实现步骤

第一步：完成环境配置与数据集下载；

第二步：对于第一层（Conv）卷积层提取图像特征值的代码编写与对应测试（代码见压缩包）；

第三步：虽然上一层卷积操作后筛出来了部分特征值，但依旧存在较多相似像素值的信息冗余，故采用 2x2 最值滤波进一步提取有效特征值，减小数据量，并缩短程序运行时间；

第四步：直接引入 Softmax 回归方法进行一次全神经网络检验，不采用训练直接检验；

第五步：由于全神经网络直接分类效果不佳，需要进行训练集学习优化。本实验中引入梯度下降法进行回馈性参数调整学习。

第六步：完全态代码编写并调试，完成实验。

2. 实验结果

3.1 实验数据详情

训练集：mnist 手写数字训练集。

数据大小：训练集一共包含 60,000 张图像和标签，而测试集一共包含了 10,000 张图像和标签。（单张图片像素规格为 28x28）。

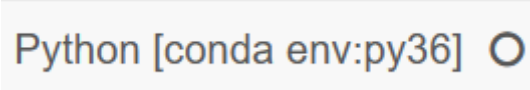
本实验依次各选用 1000、2000、3000 张训练集图片与标签进行训练。

3.2 实验环境设置

编程语言：python。

编译方式：不限，代码可以使用 vscode，pycharm 等运行

调用环境： 受限于 Python 的版本与库的匹配问题，利用 anaconda prompt 在 Jupyter Notebook 中配置 Python3.6 的虚拟环境并导入需要的函数库；



tensorflow	2.1.0
numpy	1.19.2
python	3.6.13
keras	2.3.1

使用 3x3 算子作为卷积核构建神经网络；

利用在训练过程中使用梯度下降法优化神经网络模型，提升分类准确度。

参数选择：训练数据依次为 1000 张，2000 张，3000 张。测试 800 张测试数据。（受限于 Jupyter Notebook 本实验采用手动修改）

```
(train_images, train_labels),(test_images, test_labels) = mnist.load_data()##导入mnist库
conv = conv3x3(8) ##卷积层对象
##使用需要张数的训练图片
train_images = train_images[:2000]
train_labels = train_labels[:2000]
##使用需要张数的测试图片
test_images = test_images[:800]
test_labels = test_labels[:800]
```

以每一百张图片为一代进行训练。

3.3 实验结果及分析

第一步：未经训练的神经网络模型进行预测分析运行效果图如下所示。介于神经网络特殊性，每一次执行结果虽然略有不同，但是可以看出不经过训练效果不佳，准确度均位于 15%以下。（具体代码见压缩包）

```
Mnist CNN init!
[Steps 100] 过去100张图中：平均损失值：2.303|正确率：6%
[Steps 200] 过去100张图中：平均损失值：2.303|正确率：7%
[Steps 300] 过去100张图中：平均损失值：2.303|正确率：10%
[Steps 400] 过去100张图中：平均损失值：2.304|正确率：12%
[Steps 500] 过去100张图中：平均损失值：2.303|正确率：8%
[Steps 600] 过去100张图中：平均损失值：2.303|正确率：9%
[Steps 700] 过去100张图中：平均损失值：2.303|正确率：9%
[Steps 800] 过去100张图中：平均损失值：2.303|正确率：9%
[Steps 900] 过去100张图中：平均损失值：2.303|正确率：9%
[Steps 1000] 过去100张图中：平均损失值：2.303|正确率：14%
```

第二步：加入 Softmax 层优化训练后的效果呈现，可以发现正确率得到了显著提高（具体代码见 code 文件）：

```
        loss = 0
        num_correct = 0
    l,ac = train(im,label)
    loss += l
    num_correct += ac
```

```
Mnist CNN Init!
[Steps 100] 过去100张图中：平均损失值：2.202|正确率：22%
[Steps 200] 过去100张图中：平均损失值：2.048|正确率：38%
[Steps 300] 过去100张图中：平均损失值：1.855|正确率：57%
[Steps 400] 过去100张图中：平均损失值：1.685|正确率：68%
[Steps 500] 过去100张图中：平均损失值：1.600|正确率：66%
[Steps 600] 过去100张图中：平均损失值：1.633|正确率：59%
[Steps 700] 过去100张图中：平均损失值：1.544|正确率：67%
[Steps 800] 过去100张图中：平均损失值：1.392|正确率：72%
[Steps 900] 过去100张图中：平均损失值：1.303|正确率：76%
[Steps 1000] 过去100张图中：平均损失值：1.290|正确率：74%
[Steps 1100] 过去100张图中：平均损失值：1.322|正确率：66%
[Steps 1200] 过去100张图中：平均损失值：1.242|正确率：74%
[Steps 1300] 过去100张图中：平均损失值：1.203|正确率：70%
[Steps 1400] 过去100张图中：平均损失值：1.096|正确率：79%
[Steps 1500] 过去100张图中：平均损失值：1.068|正确率：74%
[Steps 1600] 过去100张图中：平均损失值：1.075|正确率：84%
[Steps 1700] 过去100张图中：平均损失值：0.902|正确率：84%
[Steps 1800] 过去100张图中：平均损失值：0.788|正确率：88%
[Steps 1900] 过去100张图中：平均损失值：0.889|正确率：83%
[Steps 2000] 过去100张图中：平均损失值：0.900|正确率：78%
```

第三步：加入 Pooling 层与卷积层的优化训练，得到综合的优化后分类效果，并加入 800 张测试集。

```
Mnist CNN Init!
[Steps 100] 过去100张图中：平均损失值：2.252|正确率：19%
[Steps 200] 过去100张图中：平均损失值：2.123|正确率：33%
[Steps 300] 过去100张图中：平均损失值：1.665|正确率：59%
[Steps 400] 过去100张图中：平均损失值：1.068|正确率：67%
[Steps 500] 过去100张图中：平均损失值：0.927|正确率：71%
[Steps 600] 过去100张图中：平均损失值：0.972|正确率：67%
[Steps 700] 过去100张图中：平均损失值：0.840|正确率：75%
[Steps 800] 过去100张图中：平均损失值：0.556|正确率：81%
[Steps 900] 过去100张图中：平均损失值：0.737|正确率：76%
[Steps 1000] 过去100张图中：平均损失值：0.724|正确率：82%

开始测试
Test Loss: 0.8299098099688693
测试正确率：  0.70625
```

1000 张训练集结果之一

```
Mnist CNN Init!
[Steps 100] 过去100张图中：平均损失值：2.189|正确率：23%
[Steps 200] 过去100张图中：平均损失值：1.854|正确率：42%
[Steps 300] 过去100张图中：平均损失值：1.276|正确率：65%
[Steps 400] 过去100张图中：平均损失值：0.873|正确率：75%
[Steps 500] 过去100张图中：平均损失值：0.851|正确率：74%
[Steps 600] 过去100张图中：平均损失值：0.928|正确率：69%
[Steps 700] 过去100张图中：平均损失值：0.809|正确率：75%
[Steps 800] 过去100张图中：平均损失值：0.547|正确率：81%
[Steps 900] 过去100张图中：平均损失值：0.739|正确率：78%
[Steps 1000] 过去100张图中：平均损失值：0.716|正确率：82%
[Steps 1100] 过去100张图中：平均损失值：0.917|正确率：72%
[Steps 1200] 过去100张图中：平均损失值：0.658|正确率：79%
[Steps 1300] 过去100张图中：平均损失值：0.723|正确率：79%
[Steps 1400] 过去100张图中：平均损失值：0.605|正确率：81%
[Steps 1500] 过去100张图中：平均损失值：0.500|正确率：84%
[Steps 1600] 过去100张图中：平均损失值：0.527|正确率：81%
[Steps 1700] 过去100张图中：平均损失值：0.466|正确率：84%
[Steps 1800] 过去100张图中：平均损失值：0.280|正确率：88%
[Steps 1900] 过去100张图中：平均损失值：0.378|正确率：87%
[Steps 2000] 过去100张图中：平均损失值：0.426|正确率：85%

开始测试
Test Loss: 0.4979463480601525
测试正确率： 0.83875
```

2000 张训练集结果之一

```
Mnist CNN Init!
[Steps 100] 过去100张图中：平均损失值：2.238|正确率：20%
[Steps 200] 过去100张图中：平均损失值：2.031|正确率：38%
[Steps 300] 过去100张图中：平均损失值：1.470|正确率：62%
[Steps 400] 过去100张图中：平均损失值：0.966|正确率：69%
[Steps 500] 过去100张图中：平均损失值：0.901|正确率：72%
[Steps 600] 过去100张图中：平均损失值：0.958|正确率：68%
[Steps 700] 过去100张图中：平均损失值：0.832|正确率：73%
[Steps 800] 过去100张图中：平均损失值：0.553|正确率：80%
[Steps 900] 过去100张图中：平均损失值：0.749|正确率：76%
[Steps 1000] 过去100张图中：平均损失值：0.724|正确率：81%
[Steps 1100] 过去100张图中：平均损失值：0.925|正确率：72%
[Steps 1200] 过去100张图中：平均损失值：0.670|正确率：79%
[Steps 1300] 过去100张图中：平均损失值：0.725|正确率：77%
[Steps 1400] 过去100张图中：平均损失值：0.605|正确率：80%
[Steps 1500] 过去100张图中：平均损失值：0.507|正确率：84%
[Steps 1600] 过去100张图中：平均损失值：0.524|正确率：82%
[Steps 1700] 过去100张图中：平均损失值：0.463|正确率：85%
[Steps 1800] 过去100张图中：平均损失值：0.277|正确率：90%
[Steps 1900] 过去100张图中：平均损失值：0.394|正确率：87%
[Steps 2000] 过去100张图中：平均损失值：0.437|正确率：86%
[Steps 2100] 过去100张图中：平均损失值：0.675|正确率：82%
[Steps 2200] 过去100张图中：平均损失值：0.251|正确率：94%
[Steps 2300] 过去100张图中：平均损失值：0.352|正确率：91%
[Steps 2400] 过去100张图中：平均损失值：0.385|正确率：85%
[Steps 2500] 过去100张图中：平均损失值：0.494|正确率：84%
[Steps 2600] 过去100张图中：平均损失值：0.224|正确率：94%
[Steps 2700] 过去100张图中：平均损失值：0.437|正确率：84%
[Steps 2800] 过去100张图中：平均损失值：0.415|正确率：85%
[Steps 2900] 过去100张图中：平均损失值：0.399|正确率：86%
[Steps 3000] 过去100张图中：平均损失值：0.434|正确率：85%

开始测试
Test Loss: 0.4589509231631546
测试正确率： 0.86875
```

3000 张训练集结果之一

综上，我们不难发现，随着训练数据集的增加，利用 CNN 神经网络的手写数字识别实验中我们将获得更好的分类效果。在 5000 张训练集下，测试集正确预测概

率接近 90%

3. 总结

依旧存在的问题与思考：

1、此模型编译时速度较慢；

2、相对网上平均 90%+的成熟模型来讲优化效果依旧欠佳（3000 张训练集学习后依旧只有 86.8%且不稳定）；