手写数字识别算法实现

郭峻杞 16337069

本次实验用了 K-近邻（KNN），支持向量机（SVM）和卷积神经网络（CNN）三种方法，根据 MNIST 数据集进行训练和识别。包括了三种方法的介绍，实验步骤及实验结果。

最后，简单实现了一个 UI 界面。

零．实验环境，数据的获取及处理

实验环境：python3

可能用到的库：numpy，sklearn，tensorflow，matplotlib，cv2

从官网下载 MNIST 数据集。数据文件是二进制文件，需要自己读取。

在程序文件 process.py 中，写了多个函数对数据进行读取和处理。

运行 process.py，将得到：以 npy 格式存储的图像和标签，二值化后的图像，归一化后的图像，以及 one_hot 形式的标签。

一．K-近邻（KNN）

1.算法介绍

KNN 应该是三类中原理最简单的算法了。其基本思想是，对于测试集中的一个数据，找到训练集中与该数据“距离”最近的 K 个点，然后看这 K 个点中属于哪一类的点最多，便将该数据归为哪一类。

从以上叙述便可以看出 KNN 中两个关键的因素：“距离”的定义和 K 值的选取。

“距离”：定义距离前，我们一般要提取特征，假设我们提取的特征有 n 维，那么一个数据便为 n 维空间上的一个点。不同数据的“距离”便为对应的坐标系的两个点之间的距离。所以，定义“距离”，首先要定义特征空间，然后在特征空间上，选择一范数，二范数（欧氏距离）等作为距离的衡量。一个好的“距离”，应该满足同类之间距离很小，不同类之间距离

很大。对于手写数字识别算法，好的“距离”应该有平移不变性，旋转不变性和线条粗细不变性。本次 KNN 分别使用像素的灰度值（在 0 到 255 内），二值化（大于 128 的灰度值为 1，否则为 0）后的灰度值，归一化后的每一个像素点的灰度值作为特征，欧式距离为距离。也就是说，特征空间有 28*28=784 维，距离为特征空间里的欧式距离。这么定义的距离，性质并不算好，其对平移，旋转，线条粗细的鲁棒性都很差。不过，在数据足够多，K 值选取恰当的情况下，也能取得不错的效果。

K 值：K 值的选取不宜过小，也不宜太大。一般从 3 开始到 10 结束，选择最优的 K 值即可。

2.实验步骤

使用 numpy 库，用数据处理后得到的三种类型的图像进行预测。具体步骤和实现代码见 KNN.py。

3.实验结果

Parameter	Description	Value
0 到 255 的灰度值，K=3	先选取了前一百个作为测试，效果很差。便没在全部测试集上测试。	Correct rate: 0.24 Time used: 10.268012762069702 s
二值化后的图像，K=3	对全部测试集进行了测试。	Correct rate: 0.9573 Time used: 3601.07088470459 s
归一化后的图像，K=3	对全部测试集进行了测试。	Correct rate: 0.9705 Time used: 1638.0315289497375 s

可以看到，使用归一化的图像效果最好。

二．支持向量机（SVM）

1.算法介绍

对 MNIST 数据集使用 SVM，特征空间还是归一化后的图像的每个像素点的灰度值。

对于 MNIST 数据集，并不能保证其线性可分。因此我们考虑要考虑升维，升维后的计算可以使用”核函数“这一技巧。我们分别采用了多项式核以及高斯核来构造模型。

对于 SVM 算法，升维后利用核函数解决二分类问题，课程已讲的很清楚了。基本方法是，要构造一条特征空间内的直线，使该直线到两类的最近的点的距离尽可能大。这可以转化为一个具有约束条件的极小值问题，然后利用拉格朗日乘数法，结合 KKT 条件，即可解了。

对于多分类问题，有两种策略，一种是 one vs one，一种是 one vs all。此处我们用 one vs all 的策略。对每一个类 y_i，都有一个对应的权向量集 a_i。当 x 属于类 y_i 时，意为着对所有的 j≠i，有 a_i·x>a_j·x。这样定义的分类器，可通过 Kesler 构造法等方法将多类问题转化为两类问题来解决。

2.实验步骤

使用 sklearn 库中的 SVM 函数来实现。具体步骤及代码见 SVM.py

3.实验结果

Parameter	Description	Value
高斯核 错误的惩罚系数：5.0 核系数=0.05	使用高斯核建模。	Time used: 1694.0657200813293s Correct rate: 0.983700
多项式核	使用多项式核建模。	Time used: 823.9832899570465 Model is done. score: 0.944600

可以看到，如果高斯核的系数选择恰当，虽然速度较慢，但效果是比多项式核的效果好的。

三．卷积神经网络（CNN）

1.算法介绍

算法原理从字面理解就可以。即对图像先做卷积，再利用神经网络分类。卷积的过程可

以看做提取特征的过程，提取完特征后，再利用多个全连接层进行分类。

本次实验采用了 LeNet-5 模型，LeNet-5 模型总共有 7 层，简介如下：

第一层，卷积层

这一层输入为原始的图像像素（已经归一化处理），因此输入层大小为 28*28*1。第一个卷积层过滤器的尺寸为 5*5，深度为 6，不使用全 0 填充，步长为 1。这样得到的输出为 24*24*6。

第二层，池化层

这一层输入为上一层输出，大小为 24*24*6。本层采用过滤器大小为 2*2，同一个图像上长和宽的步长为 2，因此本层的输出为 12*12*6。

第三层，卷积层

这一层的输入为上一层输出，大小为 12*12*6，使用的过滤器大小为 5*5*6，深度为 16。同样不使用全零填充。这样输出为 8*8*16 大小。

第四层，池化层

这一层输入为上一层输出，大小为 8*8*16，采用的过滤器和步长与第二层相同，得到输出的大小为 4*4*16。

第五层，全连接层

这一层输入为上一层输出，大小为 4*4*16，输出节点为 120 个，该层参数个数为 4*4*16*120+120 个。

第六层，全连接层

这一层输入节点为 120 个，输出节点为 84 个。

第七层，全连接层

输入节点为 84 个，输出节点为 10 个。

注意以上除了池化层，所有层最后都要经过 ReLU 处理。

其他：

损失函数用交叉熵来表示。

优化用梯度下降法。

为了防止过拟合，采用了 dropout 和正则化方法。

学习率初始化为 0.1，每次变为上一次的 0.99。

权值的迭代用指数移动平均的策略，衰减速率为 0.99。

2.实验步骤

使用 tensorflow 和 numpy 库来实现。具体步骤及代码见 CNN.py。

3.实验结果

Parameter	Description	Value
参数设置可见算法介绍部分或代码部分	使用 CNN 建模。	由于是分次训练的,时间未统计，迭代次数约为 9000 次迭代，此时正确率为 0.9932。

四．UI 界面

用到的库：matplotlib，cv2，numpy。

使用的算法：训练好的 CNN 模型。

运行 UI.py，即可在生成的窗口手写数字，按空格识别，识别完后按 a 键清空，便可以重新写下一个数字了。（尽量写大一些，亲测成功率会很高）