《人机交互技术》

项目报告

学生姓名	王殿仪，王正宝，胡南，焦元鸿，袁煜恒
学号	2020303349，2020302781，2020302607，2020302876，2020303356
专业	人工智能
学院名称	计算机学院
报告日期	**2023.1.5**
论文题目	基于 QT 和深度学习实现手写数字识别的人机交互智能系统
课程教师	赵歆波

摘　要

手写数字识别作为人工智能的一个重要组成部分，有着极其广泛地应用前景和发展。首先就国外来讲，阿拉伯数字作为全球通用的数字语言，与各个国家、民族和地区的文化背景无关，是世界上统一使用的符号。数字的识别类别较小，可以方便评估研究方法的有效性和可行性，同时也为字符的识别提供借鉴。最后，数字识别的智能系统，可以应用于财税、金融、邮件分拣等领域，比如现在银行开始投入使用的电子签名交易，以及电子支票慢慢进入市场，在教育领域，线上考试与试卷批阅也有慢慢成为主流的趋势，因此需要这样的数字识别智能系统去减少人工操作带来的不便性和出错率，方便人们生活，为人类向智能领域的发展提供更大的实用价值。

目前，手写识别作为常见的图像识别任务，让计算机识别手写的数字。与印刷字体不同的是，不同人的手写体风格迥异，大小不一，造成了一些困难，为此提高识别精度具有重大的现实意义，并且基于数字识别设计相应交互界面的智能系统具有很强的应用价值如上文阐述的金融交易，教育等领域

深度神经网络作为人工智能领域的强大算法模型基于人脑的神经元互联作为启发，一方面能够根据不同的数据自动提取数据特征，不需要人工设计特征提取方式，可以快速建立模型，另一方面其特有的多层网络结构，能够捕捉到图像中多种隐藏特征，使得研究中可以根据不同的需要获取不同的数据特征。因此具有强大的分类能力、容错能力。通过大量的数据对模型进行训练驱动学习，使其可以对数字进行较准确的识别，从而表现出其智能的特性。

为此本小组使用深度神经网络作为核心模型，并加入卷积等操作构建卷积神经网络改进，进一步提升模型精度，并基于PyQt进行图形用户界面设计，实现手写数字识别的智能交互系统。

关键词: 手写数字识别，深度神经网络，人机交互

一. 算法原理介绍

1.1 深度学习中图像识别任务

让计算机识别数字实际上是某种模式的识别，是机器学习中常见的图像分类任务(监督学习方式)，进而使得模型能够识别出图片中的数字是几，按照数字对图片进行分类 ,传统的机器学习做法是通常首先是以某种方式，提取这个模式中的特征。这个特征的提取方式通常是人工设计或指定的，并让最后的分类器学习抽取出的特征，利用传统的 BP 学习算法，最优化损失函数(减小训练时与标签的误差)，使得分类器得到正确的参数，最后进行测试模型是否能正确识别出手写的数字属于哪一类，用准确率作为指标去衡量模型的好坏(即模型是否学到正确的知识)。

准确率（accuracy）是分类任务最常见的评价指标，是被正确分类的样本数在所有样本数中的占比，通常情况下，准确率越高，分类器效果越好：

accuracy = (TP+TN)/(P+N)

	预测类别			
实际类别		Yes	No	总计
	Yes	TP	FN	P（实际为 Yes）
	No	FP	TN	N（实际为 No）
	总计	P’（被分为 Yes）	N’（被分为 No）	P+N

正例即正确的识别，负例是错误的识别

True positives(TP): 被正确地划分为正例的个数，即实际为正例且被分类器划分为正例的实例数；

False positives(FP): 被错误地划分为正例的个数，即实际为负例但被分类器划分为正例的实例数；

False negatives(FN): 被错误地划分为负例的个数，即实际为正例但被分类器划分为负例的实例数；

True negatives(TN): 被正确地划分为负例的个数，即实际为负例且被分类器划分为负例的实例数。

本文利用了深度学习，即深度神经网络模型去构建数字识别模型，深度学习是一种特殊的机器学习，即机器学习中的一个分支，利用深度神经网络模型去解决问题，与机器学习不同的是在解决问题时，传统机器学习算法通常先把问题分成几块，一个个地解决好之后，再重新组合起来(即人工特征抽取+分类器设计)。但是深度学习则是一次性地、端到端地解决，包括抽取特征，这也是深度学习的大优势，与人工提取特征相比较，深度神经网络一方面能够根据不同的数据自动提取数据特征，不需要人工设计特征提取方式，可以快速建立模型，另一方面其特有的多层网络结构，能够捕捉到图像中多种隐藏特征，将特征学习融入到了建立模型的过程中，从而减少了人为设计特征造成的不完备性，在研究中可以根据不同的需要获取不同的数据特征，进而大大提升模型的分类精度，深度学习的优势还有深度神经网络的自学习能力很强，只用把图像和对应识别的结果(label)输入深度神经网络中，网络就会通过自学习功能，学到知识，简便，端到端，效果好是深度学习的优势。

1.2 卷积神经网络的使用

1.2.1 卷积神经网络概述

卷积神经网络（Convolutional Neural Networks, CNN）是计算机视觉技术最经典的模型结构。在早期的视觉任务--图像分类中，卷积神经网络被提出之前，通常是先人工提取图像特征，提取图像特征使用的都是传统的图像处理方法，例如提取图像的纹理、边界、线条等特征，根据提取到的特征做下一步，再用机器学习算法，对这些特征进行分类，分类的结果强依赖于特征提取方法，往往只有经验丰富的研究者才能完成，在这种背景下，基于神经网络的特征提取方法应运而生。Yann LeCun 是最早将卷积神经网络应用到图像识别领域的，其主要逻辑是使用卷积神经网络提取图像特征，并对图像所属类别进行预测，通过训练数据不断调整网络参数，最终形成一套能自动提取图像特征并对这些特征进行分类的网络，并且卷积神经网络的特征抽取能力更强，模型效果更好，卷积神经网络被提出以后，通过卷积神经网络提取到的特征用于图像的识别在某些数据集上识别正确率甚至超过了人类。

图片中的某些空间结构是由不同数字构成，而这样的空间结构就是由像素点与

像素点之间的关系形成。普通的前馈神经网络是对输入图片每个像素点同等对待的，也就是说它此时并没有考虑像素点与像素点之间的关系。而卷积神经网络却能有效考虑像素点之间的联系，学到图像空间的丰富知识。

卷积神经网络之所以在图像识别上有好的表现得益于图像识别的三个特点：局部性、相同性和不变性。

1. **局部感受野(局部性):**卷积神经网络借鉴了人眼局部感受野的特点，即我们分辨图像信息，是通过一块分辨的，而不是一个像素，我们结合一个像素及其周围像素信息去分辨才能有效避免噪声，卷积神经网络就是如此，通过卷积滑动窗口，将一个区域的像素信息进行整合形成一个像素，这就是卷积有效特征提取的过程，综合信息，减小噪声，从区域中拿出更抽象的有用的语义信息(像素点与像素点之间的关系)，这比人为特征提取效果更好，比前馈神经网络效果具有大幅提升，有效保证数据空间信息不被丢失，并且引入卷积神经网络进行特征提取，既能提取到相邻像素点之间的特征模式，又能保证参数的个数不随图片尺寸变化，最终的大小，参数维度，由每一层的卷积核，池化核大小，步长，填充决定，但经历一次卷积操作后，图像尺寸会进行一个缩小(感受野卷积整合作用)，减少网络参数，和最后特征维度。

例子：识别一张图像的类别需要得到该图像类别特征，一般情况下这些类别特征不是由整张图像决定的，往往存在于图像的局部。如图 1.2.1 中鸟类的鸟喙，该特征只位于图像的局部。

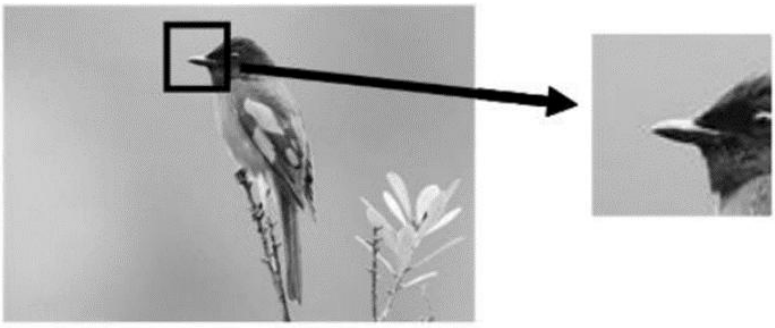


图 1.2.1.1 局部性例子

2. **权值共享(相同性):可以理解卷积是一个多维矩阵，卷积操作其实就是每个神经元和卷积矩阵上每个像素的加权求和，再通过滑动生程下一个输出点，**如果图像具有相同的类别特征，这些类别特征可能出现在图像的不同位置，即可以使用同样的提取特征模式去提取不同图像的相同特征。这些特征可能在图像的不同位置，但是对特征的提取操作却是基本相同共享一样的参数**因为权重共享，使得网络参数大大下降，并且卷积不仅可以特征抽取，在抽取过程中可以压缩图像大小，最后进行分类器是经过一个前馈神经网络，压缩图像大小展平成特征向量后，向量维度不会很高(卷积降维操作)，因此卷积操作可以大大减少计算量。**

如图 1.2.2 中鸟类的鸟喙位于图像的不同位置，但是鸟喙特征是一样的，可以使用同样的检测方法去检测。

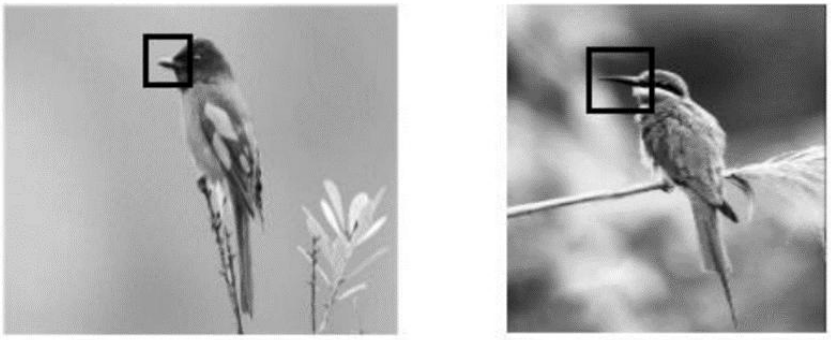


图 1.2.1.2 相同性例子

3. **池化(不变性): 卷积神经网络还有一个重要思想就是池化，**池化层通常接在卷积层后面，相当于在卷积层上架了一个窗口，但这个窗口比卷积层的窗口简单许多，不需要学习参数，它只是对窗口范围内的神经元做简单的操作，如求和，求最大值，把求得的值作为池化层神经元的输入值。

池化的作用:

1.对卷积之后的图像特征进行进一步压缩，大大减少了模型学到的特征值，也减少了后面网络层的参数(对图像下采样实现，降维操作，并扩大感受野)

2.使卷积神经网络抽取特征是保证特征局部不变性(旋转平移不变性)

3. 进一步实现非线性

4.一定程度上抑制过拟合

例子：对于一张比较大的图像，可以进行下采样（Subsampling）操作对图像做缩小操作，缩小之后的图像性质几乎保持不变。如图 1.2.1.3 所示，图像经过下采样之后仍旧可以识别出是一张鸟类图像。

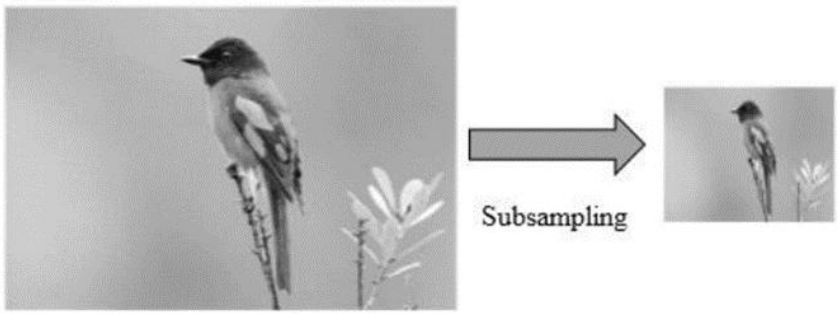


图 1.2.1.3 不变性

1.2.2 卷积神经网络计算原理

卷积（Convolution）

关于卷积算法的原理和实现方案，以及如何使用卷积对图片进行操作，有如下内容：

- 卷积计算
- 填充（padding）
- 步幅（stride）
- 感受野（Receptive Field）
- 多输入通道、多输出通道和批量操作

卷积计算

卷积是数学分析中的一种积分变换的方法，在图像处理中采用的是卷积的离散形式，在卷积神经网络中，卷积层的实现方式实际上是数学中定义的互相关（cross-correlation）运算，如图 1.2.2.1 所示 ：

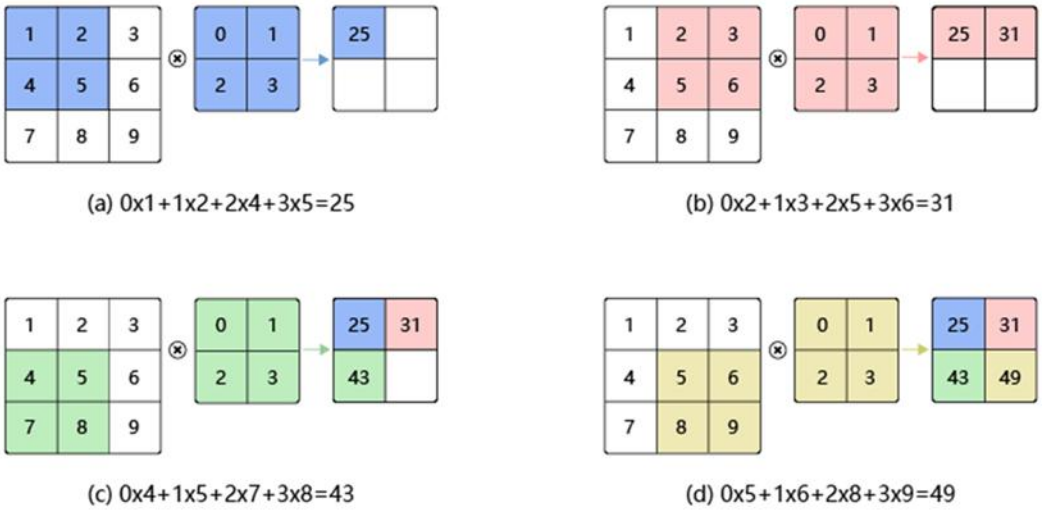


图 1.2.2.1：卷积计算过程

说明：

卷积核（kernel）也被叫做滤波器（filter），假设卷积核的高和宽分别为 *kh* 和 *kw*，则将称为 *kh*×*kw* 卷积，比如 3×5 卷积，就是指卷积核的高为 3，宽为 5。

- 如图 1.2.2.1（a）所示：左边的图大小是 3×3，表示输入数据是一个维度为 3×3 的二维数组；中间的图大小是 2×2，表示一个维度为 2×2 的二维数组，我们将这个二维数组称为卷积核。先将卷积核的左上角与输入数据的左上角（即：输入数据的(0, 0)位置）对齐，把卷积核的每个元素跟其位置对应的输入数据中的元素相乘，再把所有乘积相加，得到卷积输出的第一个结果：

  $$0\times1+1\times2+2\times4+3\times5=25 \qquad (a)$$

- 如图 1.2.2.1（b）所示：将卷积核向右滑动，让卷积核左上角与输入数据中的(0,1)位置对齐，同样将卷积核的每个元素跟其位置对应的输入数据中的元素相乘，再把这 4 个乘积相加，得到卷积输出的第二个结果：

  $$0\times2+1\times3+2\times5+3\times6=31 \qquad (b)$$

- 如图 1.2.2.1（c）所示：将卷积核向下滑动，让卷积核左上角与输入数据中的(1, 0)位置对齐，可以计算得到卷积输出的第三个结果：

  $$0\times4+1\times5+2\times7+3\times8=43 \qquad (c)$$

以此类推。

可以总结卷积的步骤为，根据卷积核大小、对应原图的大小、每个像素对应相乘再求和+偏置，成为特征图的一个像素值，然后再根据步长 stride 进行滑动，从左至右，从上到下，直到整张图卷积完成。

填充（padding）

在上面的例子中，输入图片尺寸为 3×3，输出图片尺寸为 2×2，经过一次卷积之后，图片尺寸变小。卷积输出特征图的尺寸计算方法如下（卷积核的高和宽分别为 kh 和 kw ）：

$$Hout = H - kh + 1$$

$$Wout = W - kw + 1$$

如果输入尺寸为 4，卷积核大小为 3 时，输出尺寸为 4−3+1=24-3+1=24−3+1=2。当卷积核尺寸大于 1 时，输出特征图的尺寸会小于输入图片尺寸。如果经过多次卷积，输出图片尺寸会不断减小。为了避免卷积之后图片尺寸变小，通常会在图片的外围进行填充(padding)，如图 1.2.2.2 所示。

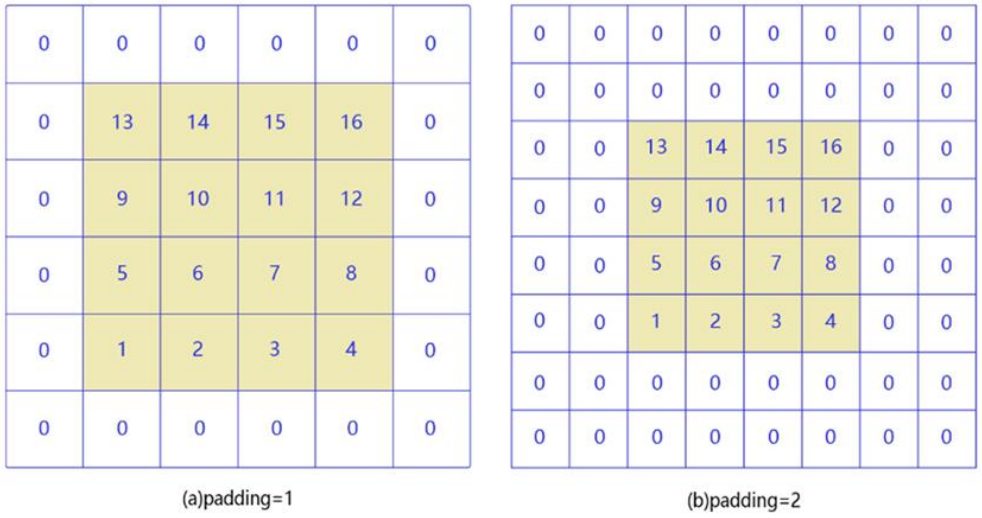


图 1.2.2.2：图形填充

- 如图 1.2.2.2（a）所示：填充的大小为 1，填充值为 0。填充之后，输入图片尺寸 4×4 变成了 6×6，使用 3×3 的卷积核，输出图片尺寸为 4×4。
- 如图 1.2.2.2（b）所示：填充的大小为 2，填充值为 0。填充之后，输入图片尺寸从 4×4 变成了 8×8，使用 3×3 的卷积核，输出图片尺寸为 6×6。

如果在图片高度方向，在第一行之前填充 $ph1$ 行，在最后一行之后填充 $ph2$ 行；在图片的宽度方向，在第 1 列之前填充 $pw1$ 列，在最后 1 列之后填充 $pw2$ 列；则卷积核操作之后(填充加卷积)，输出图片的尺寸为：

$$Hout = H+ph1 + ph2 -kh +1$$

$$Wout = W+pw1 + pw2 -kw +1$$

在卷积计算过程中，通常会在高度或者宽度的两侧采取等量填充，即 $ph1=ph2=ph$，$pw1=pw2=pw$，上面计算公式也就变为：

$$Hout = H + 2ph -kh +1$$

$$Wout = W + 2pw -kw +1$$

卷积核大小通常使用 1，3，5，7 这样的奇数，如果使用的填充大小为 $ph=(kh-1)/2$，$pw=(kw-1)/2$，则卷积之后图像尺寸不变。例如当卷积核大小为 3 时，padding 大小为 1，卷积之后图像尺寸不变；同理，如果卷积核大小为 5，padding 大小为 2，也能保持图像尺寸不变。(卷积操作后图像尺寸会变小，即经过卷积操作，一般用 padding 来保证图像大小不变，填充的像素为 0，填充有四个参数，即第一行最后一行，第一列，最后一列，填充宽度，即填充是四周包围的操作，若设置参数为 1 这样的一个数，默认四个参数都相等于那个数)

步幅（stride）

图 1.2.2.2 中卷积核每次滑动一个像素点，步幅为 1。图 1.2.2.3 是步幅为 2 的卷积过程，卷积核在图片上移动时，每次移动大小为 2 个像素点。

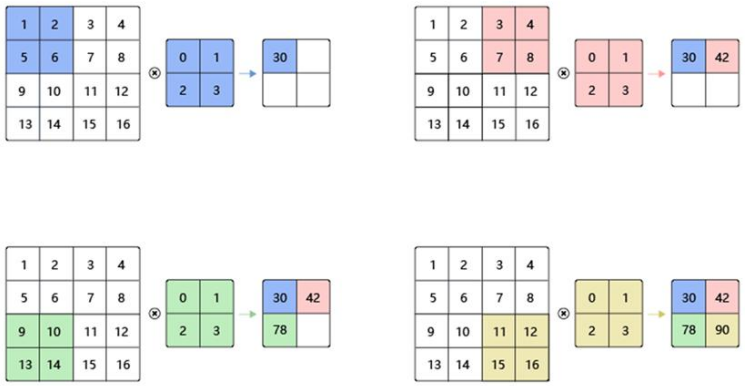


图 1.2.2.3：步幅为 2 的卷积过程

当宽和高方向的步幅分别 sh 和 sw 时，输出特征图尺寸的计算公式是：

$$Hout = H+2ph-kh+1/\text{sh}$$

$$Wout = W+2pw-kw+1/\text{sw}$$

感受野（Receptive Field）

输出特征图上每个点的数值，是由输入图片上大小为 $kh \times kw$ 的区域的元素与卷积核每个元素相乘再相加得到的，所以输入图像上 $kh \times kw$ 区域内每个元素数值的改变，都会影响输出点的像素值。我们将这个区域叫做输出特征图上对应点的感受野。感受野内每个元素数值的变动，都会影响输出点的数值变化。3×3 卷积对应的感受野大小就是 3×3，如图 1.2.2.4 所示。

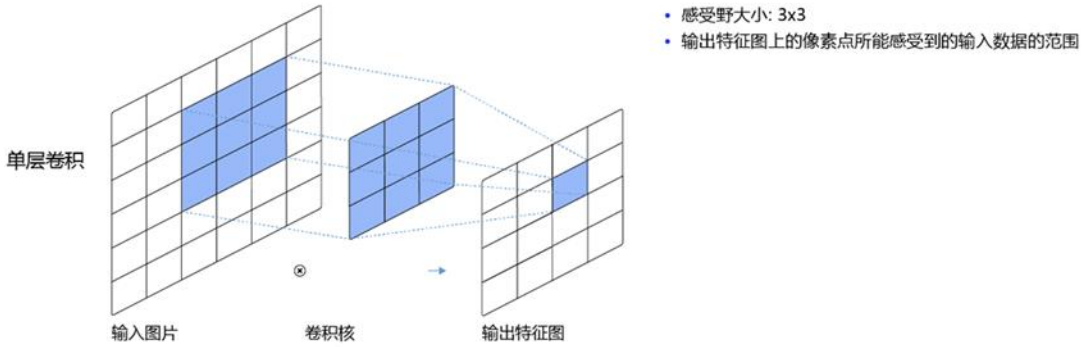


图 1.2.2.4：感受野为 3×3 的卷积

而当通过两层 3×3 的卷积之后，感受野的大小将会增加到 5×5，如图 1.2.2.5 所示。

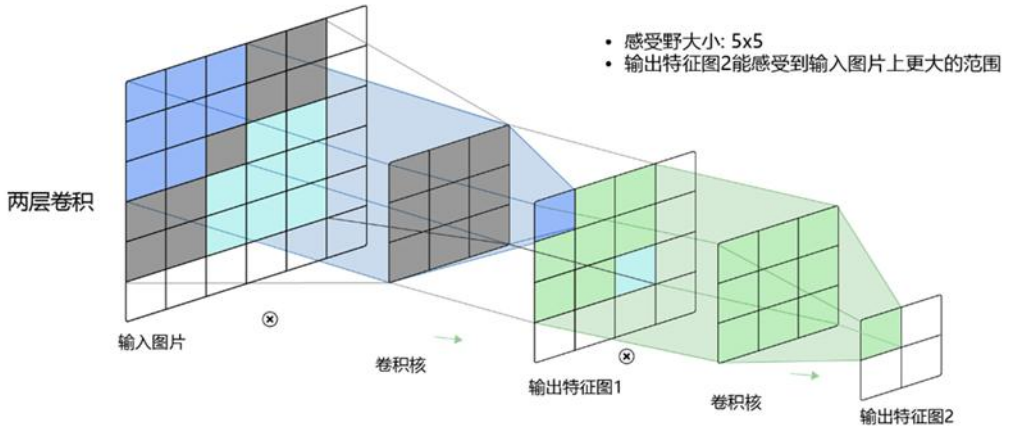


图 1.2.2.5：感受野为 5×5 的卷积

因此，当增加卷积网络深度的同时，感受野将会增大，输出特征图中的一个像素点将会包含更多的图像语义信息。

卷积越多，感受野越大，即后续特征图一个像素就对应于之前原图的一个区域，也与卷积神经网络网络，卷积抽取特征的作用，得出最后的特征向量以便分类

多输入通道、多输出通道和批量操作

前面介绍的卷积计算过程比较简单，实际应用时，处理的问题要复杂的多。例如：对于彩色图片有 RGB 三个通道，需要处理多输入通道的场景。输出特征图往往也会具有多个通道，而且在神经网络的计算中常常是把一个批次的样本放在一起计算，所以卷积算子需要具有批量处理多输入和多输出通道数据的功能，下面将分别介绍这几种场景的操作方式。

多输入通道场景

上面的例子中，卷积层的数据是一个 2 维数组，但实际上一张图片往往含有 RGB 三个通道，要计算卷积的输出结果，卷积核的形式也会发生变化。假设输入图片的通道数为 *Cin*，输入数据的形状是 *Cin*×*Hin*×*Win*，计算过程如图 1.2.2.6 ：

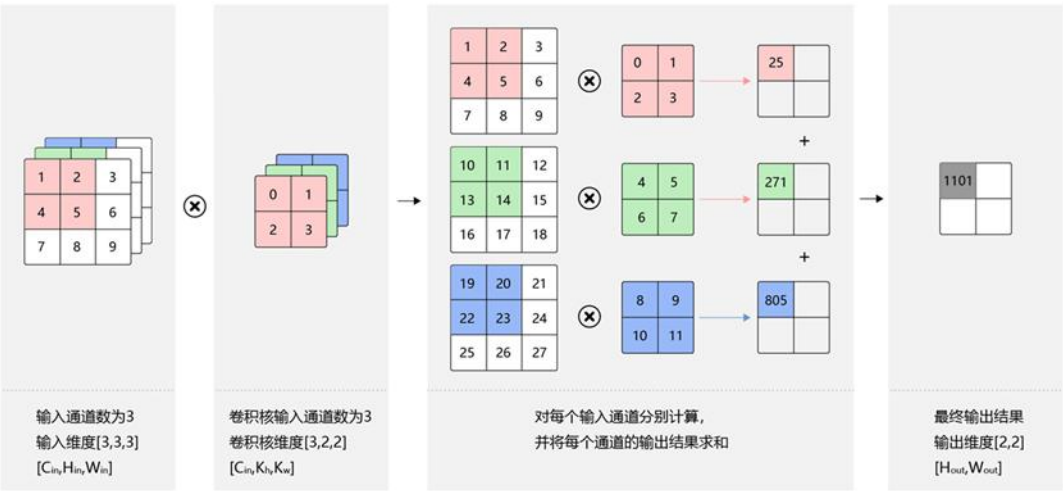


图 1.2.2.6：多输入通道计算过程

1. 对每个通道分别设计一个 2 维数组作为卷积核，卷积核数组的形状是 *Cin*×*kh*×*kw*。

2. 对任一通道 $Cin \in [0, Cin)$，分别用大小为 $kh \times kw$ 的卷积核在大小为 $Hin \times Win$ 的二维数组上做卷积。
3. 将这 Cin 个通道的计算结果相加，得到一个形为 $Hout \times Wout$ 的二维数组。

多输出通道场景

一般来说，卷积操作的输出特征图也会具有多个通道 $Cout$ ，这时我们需要设计 $Cout$ 个维度为 $Cin \times kh \times kw$ 的卷积核，卷积核数组的维度是 $Cout \times Cin \times kh \times kw$，如图 1.2.2.7 所示。

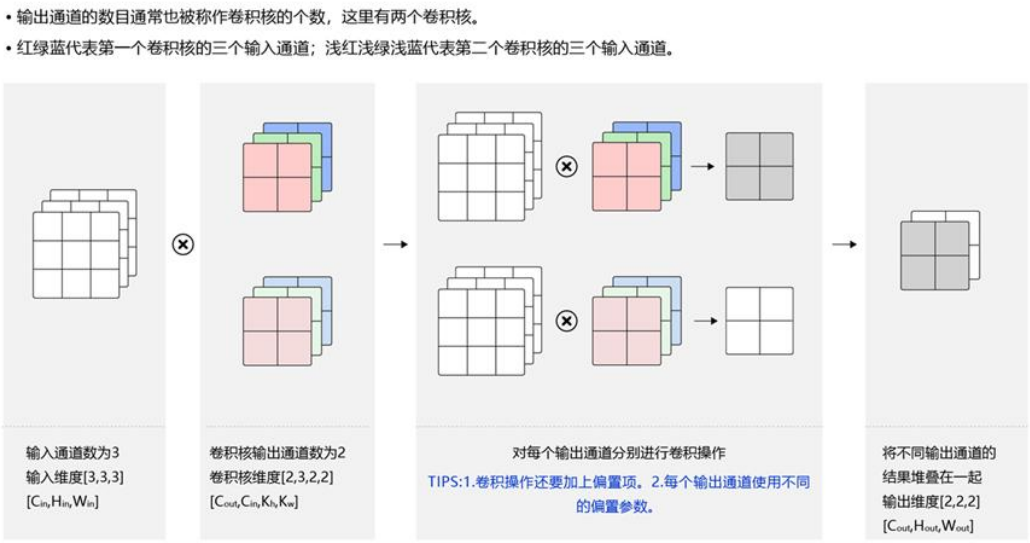


图 1.2.2.7：多输出通道计算过程

1. 对任一输出通道 $cout \in [0,Cout)$，分别使用上面描述的形状为 $Cin \times kh \times kw$ 的卷积核对输入图片做卷积。
2. 将这 $Cout$ 个形状为 $Hout \times Wout$ 的二维数组拼接在一起，形成维度为 $Cout \times Hout \times Wout$ 的三维数组。

说明：(数据结构 tensor 表示的信息)

通常将卷积核的输出通道数叫做卷积核的个数。

例如，输入 x = [3,28,28]，表示输入通道数为 3，图像尺寸为 28x28 (输入通道为 3 是 RGB，1 是灰度图)。

进入卷积层后，输入通道数 in_channel 与卷积核个数对应。如有三个通道，那么每个通道上的图片就要进行一次卷积运算；如卷积核对应[3,5,5]，就是对应三个输入通道，5x5 的卷积操作，设置 padding 和 stride

输出通道 out_channel 可以设置，表征输出图像通道数。如 Xout = [16, wout,hout]，表示输出 16 通道图像，wout，hout 可以利用上述公式计算。

卷积核各个像素的数值参数由梯度下降反向传播进行学习

批量操作

在卷积神经网络的计算中，通常将多个样本放在一起形成一个 mini-batch 进行批量操作，即输入数据的维度是 $N \times Cin \times Hin \times Win$。由于会对每张图片使用同样的卷积核进行卷积操作，卷积核的维度与上面多输出通道的情况一样，仍然是 $Cout \times Cin \times kh \times kw$，输出特征图的维度是 $N \times Cout \times Hout \times Wout$，如图 1.2.2.8 所示。

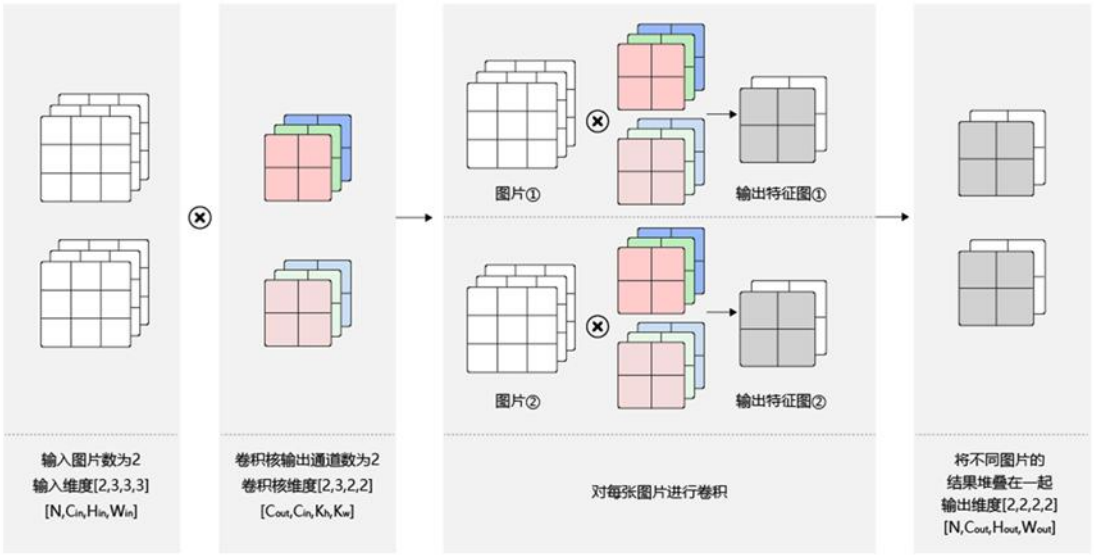


图 1.2.2.8：批量操作

考虑批量操作，则数据结构是 4 维向量，第一位表征训练的 batch 后三维上述操作一致

池化（Pooling）

池化是使用某一位置的相邻输出的总体统计特征代替网络在该位置的输出，其

好处是当输入数据做出少量平移时，经过池化函数后的大多数输出还能保持不变。比如：当识别一张图像是否是人脸时，我们需要知道人脸左边有一只眼睛，右边也有一只眼睛，而不需要知道眼睛的精确位置，这时候通过池化某一片区域的像素点来得到总体统计特征会显得很有用。由于池化之后特征图会变得更小，如果后面连接的是全连接层，能有效的减小神经元的个数，节省存储空间并提高计算效率。如 图 1.2.2.9 所示，将一个 2×2 的区域池化成一个像素点。通常有两种方法，平均池化和最大池化。

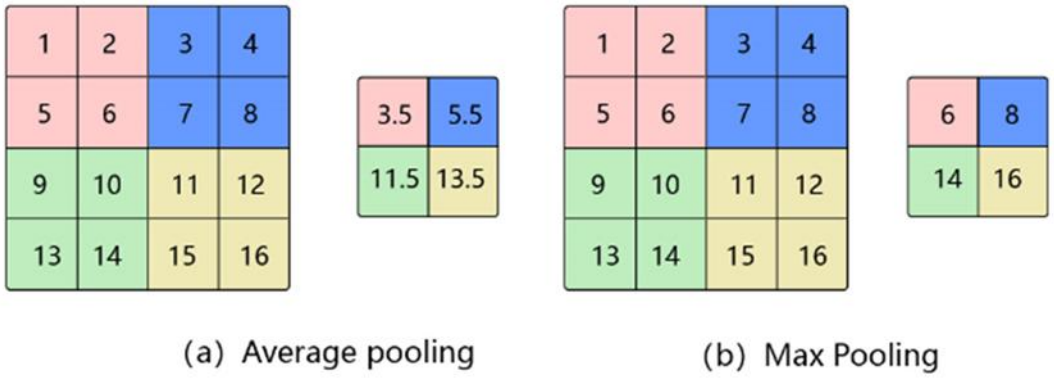


图 1.2.2.9：池化

- 如图 1.2.2.9（a）：平均池化。这里使用大小为、2×2 的池化窗口，每次移动的步幅为 2，对池化窗口覆盖区域内的像素取平均值，得到相应的输出特征图的像素值。
- 如图 1.2.2.9（b）：最大池化。对池化窗口覆盖区域内的像素取最大值，得到输出特征图的像素值。当池化窗口在图片上滑动时，会得到整张输出特征图。池化窗口的大小称为池化大小，用 $kh \times kw$ 表示。在卷积神经网络中用的比较多的是窗口大小为 2×2，步幅为 2 的池化。

与卷积核类似，池化窗口在图片上滑动时，每次移动的步长称为步幅，当宽和高方向的移动大小不一样时，分别用 sw 和 sh 表示。也可以对需要进行池化的图片进行填充，填充方式与卷积类似，假设在第一行之前填充 $ph1$ 行，在最后一行后面填充 $ph2$ 行。在第一列之前填充 $pw1$ 列，在最后一列之后填充 $pw2$ 列，则池化层的输出特征图大小为：

$$Hout = H+ph1 +ph2 -kh /\text{sh} + 1$$

$$Wout = W+pw1 +pw2 -kw /\text{sw} + 1$$

在卷积神经网络中，通常使用 2×2 大小的池化窗口，步幅也使用 2，填充为 0，则输出特征图的尺寸为：$Hout = 2H$； $Wout = 2W$

通过这种方式的池化，输出特征图的高和宽都减半，但通道数不会改变。

全连接

全连接层（Fully Connected Layer）与一般的神经网络相同，每一个神经元 结点都会与前一层的所有神经元结点相连接。一般情况下，全连接层都放在神经 络模型的最后两层，用于输出分类结果。全连接层的操作类似于将一个三维的矩阵空间平铺为一个一维的向量。例如经过卷积、池化等操作后输出的特征图大小为 3 × 3 × 5，经过全连接层后可以得到一个 1 × N 的输出，N 可以由实际效果设定，具体的操作如图 1.2.2.10 所示。

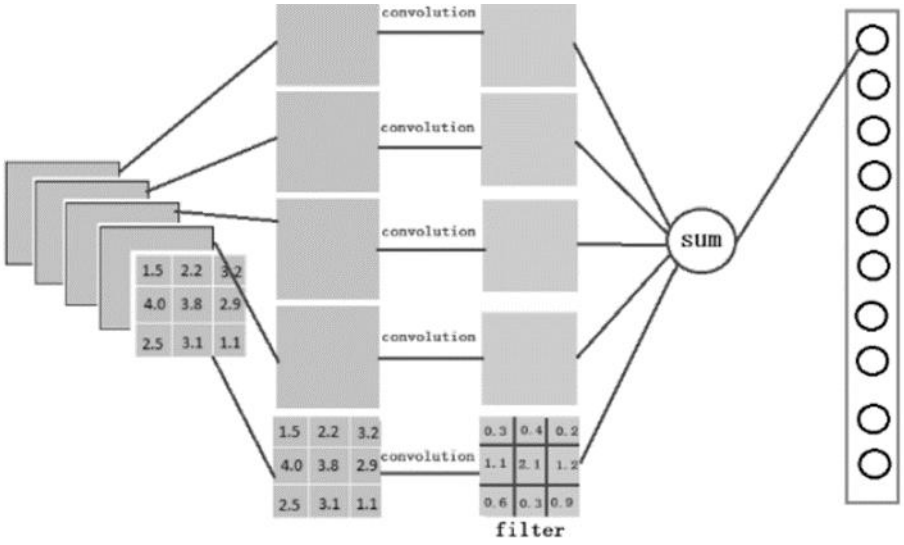


图 1.2.2.10 全连接计算

使用一个和特征图大小一样的滤波器去和特征图做卷积操作，得到的结果 相加即为全连接层的一个神经元结点的输出，这样的计算做 N 次即得到 1 × N 的全连接层输出。由此可见，全连接操作可以看作是用大小为 3 × 3 × 5 × N 的滤波 器去做卷积操作。由于全连接层每个结点都彼此相连，所以全连接层可以聚合之前卷积层或者池化层中得到局部区域性的特征信息，把前面卷积层、池化层学习到的局

部特征信息映射到整个样本空间中，用以分类。因为全连接层把局部特征整合在一起输出为一个值，所以极大的减少了由于特征信息位置不同而带来对分类结果的影响，即前文提到的图像识别具有相同性,网络模型最后一层全连接层输出维度和数据中分类数一致。

综上，卷积神经网络有局部感受野，权值共享，池化三大功能，更能有效抽取到图像特征，从图像空间区域中学习更抽象丰富有用的语义信息(像素点与像素点之间的关系)，即图像的空间联系是局部的像素联系较为紧密，而距离较远的像素相关性则较弱。因而，每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知，然后在更高层将局部的信息综合起来就得到了全局的信息。这样做既能保证有效保证数据空间信息不被丢失，并且因为有权值共享的性质，可以大大减少网络参数量与复杂度(特征降维操作)，节约了算力资源，缓解过拟合问题，并且池化保证特征的局部不变性，有利于数据增广操作，保证提取特征是有效的，因此本文对于手写数字识别模型的构建，特征抽取部分使用卷积神经网络结构。

1.3 GUI 与 Qt 概述

图形用户界面（GUI）是指计算机程序的界面，其中使用图形元素（如按钮、菜单、文本框等）来向用户提供信息和接受用户输入。 GUI 可以让用户使用图形化界面更方便地操作计算机，而无需熟悉命令行界面或编写代码。

Qt 是一种跨平台的 C++ 图形用户界面（GUI）库，可以用于开发 GUI 应用程序。因此，基于 Qt 和人工神经网络的手写数字识别板可能是一个应用程序，其中包含一个人工神经网络模型，并使用 Qt 的 GUI 元素来显示输入和输出。

在本项目中，为了方便项目的打包，我们使用了 Qt 在 Python 环境中的库 PyQt5 来设计用户界面，并按照人机交互设计模式的不同将界面设计分为两大板块，画笔交互板块和鼠标选择交互板块，详细如下图：

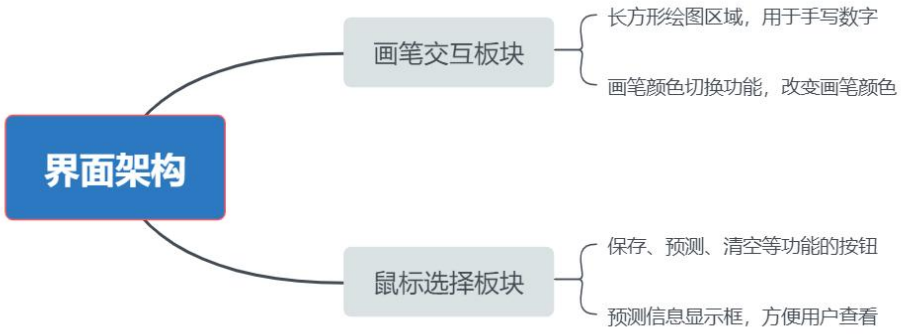


图 1.3.1 用户界面架构设计

满足人机交互图形用户界面三大思想:

图 1.3.2 用户界面三大思想

二. 实验过程

2.1 数据集介绍

本项目使用 MINST 数据集进行模型训练

发布方：National Institute of Standards and Technology(美国国家标准技术研究所，简称 NIST)

发布时间：1998

简介：MNIST 数据集是从 NIST 的两个手写数字数据集：Special Database 3 和 Special Database 1 中分别取出部分图像，并经过一些图像处理后得到的。MNIST 数据集共有 70000 张图像，其中训练集 60000 张，测试集 10000 张。所有图像都是 28×28 的灰度图像，每张图像包含一个手写数字。

背景：MNIST 是一个手写体数字的图片数据集，该数据集来由美国国家标准与技术研究所（National Institute of Standards and Technology (NIST)）发起整理，一共统计了来自 250 个不同的人手写数字图片，其中 50%是高中生，50%来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的精确识别。

数据集详细信息：

数据量训练集 60000 张图像，其中 30000 张来自 NIST 的 Special Database 3，30000 张来自 NIST 的 Special Database 1。测试集 10000 张图像，其中 5000 张来自 NIST 的 Special Database 3，5000 张来自 NIST 的 Special Database 1。标注量每张图像都有标注。标注类别共 10 个类别，每个类别代表 0~9 之间的一个数字，每张图像只有一个类别。

MNIST 样例图 NIST 原始的 Special Database 3 数据集和 Special Database 1 数据集均是二值图像，MNIST 从这两个数据集中取出图像后，通过图像处理方法使得每张图像都变成 28×28 大小的灰度图像，且手写数字在图像中居中显示。数据集和对应 label 在使用的深度学习框架中已经封装好，直接进行 load 调用即可。具体样式如下图所示,

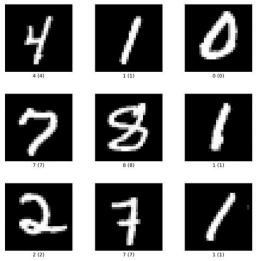


图 2.1.1 MNIST 数据集图像

2.2 实验环境:

- ✧ 编程语言：python
- ✧ IDE：vscode，PyCharm
- ✧ 深度学习框架：tensorflow.keras
- ✧ 工具包：Windows，python==3.x，tensorflow-cpu==2.7.0，PyQt5==5.x，numpy，pandas，opencv-python，（Windows or Linux）

2.3 数据预处理

采用交叉验证的方式对模型进行更好的训练效果监视和评估，我们将数据集划分成训练集(train)，验证集(val)，测试集(test)比例为 8:1:1

此外，对数据集中每一张图像进行归一化。具体实现如图 2.3.1 所示

```
mnist = keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()
#x_train, x_test = x_train/255.0, x_test/255.0  # 除以 255 是为了归一化。

X_train4D = x_train.reshape(x_train.shape[0], 28, 28, 1).astype('float32')
X_test4D = x_test.reshape(x_test.shape[0], 28, 28, 1).astype('float32')

X_train4D_Normalize = X_train4D / 255 # 归一化
X_test4D_Normalize = X_test4D / 255

y_trainOnehot = to_categorical(y_train)
y_testOnehot = to_categorical(y_test)
X_train, X_val, Y_train, Y_val = train_test_split(X_train4D_Normalize, y_trainOnehot, test_size = 0.1, rando
```

图 2.3.1 数据归一化

数据增广

高质量的训练数据对模型训练大有裨益。在已有的训练数据集基础上，我们引入了数据增广，对图像进行预处理并生成用于训练深度学习模型的增强图像数据，进而得到更好的训练效果。下面是应用于图像的预处理和增强选项。

- featurewise_center 和 samplewise_center：指定是否通过从每个特征 (featurewise_center) 或每个样本 (samplewise_center) 中减去数据的平均值来使数据居中。
- featurewise_std_normalization 和 samplewise_std_normalization：指定是否通过将每个特征 (featurewise_std_normalization) 或每个样本 (samplewise_std_normalization) 除以数据的标准差来对数据进行归一化。
- zca_whitening：指定是否对数据应用 ZCA 白化，这是一种去相关数据的技术，有助于提高模型的性能。
- rotation_range、zoom_range、width_shift_range 和 height_shift_range：指定将在训练期间应用于图像的随机变换范围，包括旋转，缩放，水平或垂直移动图像。
- horizontal_flip 和 vertical_flip：指定在训练期间是否随机水平或垂直翻转图像。这些几何转换有助于提高模型的鲁棒性和泛化性能。

```
# 增加数据以防止过拟合
datagen = ImageDataGenerator(
        featurewise_center=False,              # 在数据集上将输入平均值设置为0
        samplewise_center=False,               # 将每个样本的平均值设置为0
        featurewise_std_normalization=False,   # 将输入除以数据集的std
        samplewise_std_normalization=False,    # 将每个输入除以它的std
        zca_whitening=False,                   # 使用ZCA白化
        rotation_range=10,                     # 在范围内随机旋转图像（0到180度）
        zoom_range = 0.1,                      # 随机缩放图像
        width_shift_range=0.1,                 # 水平随机移动图像（总宽度的一部分）
        height_shift_range=0.1,                # 垂直随机移动图像（总高度的一部分）
        horizontal_flip=False,                 # 随机翻转图像
        vertical_flip=False)                   # 随机翻转图像
```

图 2.3.2 数据增广

2.4 网络架构设计

2.4.1 深度全连接神经网络

结构设计时有以下几个因素：

1. 输入层：输入层用于接收数据，在手写数字识别中，输入层的节点数可以根据图像大小来确定，MINST 数据集图像大小为标准的 28*28*1。

2. 隐藏层：隐藏层用于处理输入数据并提取图像特征，转化数据维度信息，在手写数字识别中，可以设置一个或多个隐藏层，节点数可以通过实验来确定，但网络过深容易出现梯度消失，过拟合现象，考虑数据量和图像的特征我们设计三层全连接神经网络。
3. 输出层：输出层用于输出识别结果，在手写数字识别中，输出层的节点数应为 10（对应 0~9 的 10 个数字）并加上分类问题常用的 softmax 激活函数。
4. 激活函数：激活函数用于将节点的输入信息转化为输出信息，在手写数字识别中，可以使用常用的激活函数，如 sigmoid 函数或 ReLU 函数，但我们使用 ReLu 函数，以便获得更好的梯度信息。
5. 加入 Dropout 以 0.5 概率使神经元失活，环境过拟合

总的来说，全连接神经网络结构的常见结构如下：

输入层 -> 隐藏层 1 -> ... -> 隐藏层 n -> 输出层

在训练过程中，目的是最优化损失函数使用后向传播算法来进行权重更新，使得网络的输出结果与实际结果更加接近，进而达到高精度的预测。

在实际应用中，神经网络的结构需要根据问题的具体情况，数据的特征分布，大小来进行网络结构的调整，比如不同的层数、每层节点数和激活函数等。

```
class Net(nn.Module):
    def __init__(self,in_dim,n_hidden_1,n_hidden_2,out_dim):
        super(Net,self).__init__()
        self.layer1=nn.Sequential(nn.Linear(in_dim,n_hidden_1),nn.BatchNorm1d(n_hidden_1),)
        self.layer2=nn.Sequential(nn.Linear(n_hidden_1,n_hidden_2),nn.BatchNorm1d(n_hidden_2))
        self.layer3=nn.Sequential(nn.Linear(n_hidden_2,out_dim))
        #self.dropout = nn.Dropout(p=0.5)
    def forward(self,x):
        x=F.relu(self.layer1(x))
        x=F.relu(self.layer2(x))
        x=self.layer3(x)
        #x=F.softmax(self.layer3(x))
        return x
```

图 2.4.1.1 基础模型架构

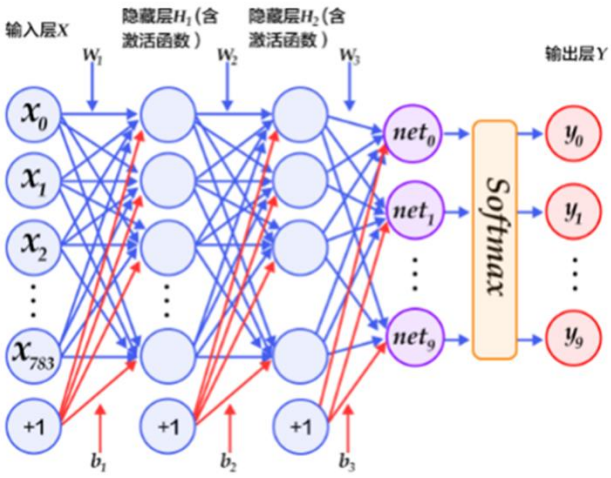


图 2.4.1.2 全连接结构

2.4.2 卷积神经网络改进

```
# 设置CNN模型
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size = (5,5), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.4))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size = (5,5), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(128, kernel_size = 4, activation='relu'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
```

图 2.4.2.1 模型设计代码

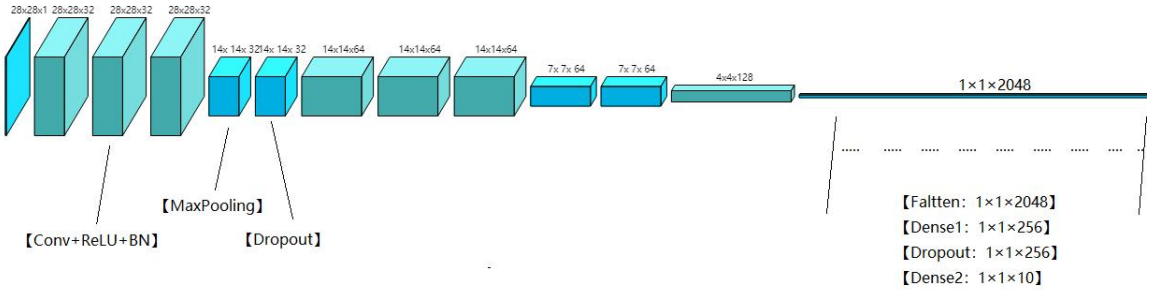


图 2.4.2.2 模型结构

1. 使用 Keras 中的 Sequential，定义神经网络中的线性层堆栈，创建顺序模型。它是构建由线性堆叠层组成的模型的一种简单方法，由上至下每一层的输出下一层的输入，通过 add 来进行网络结构的自定义叠加。

2. 我们这次设计的卷积神经网络结构参考了 vgg 的网络结构设计，采用三个连续的卷积块进行有效的图像特征抽取，其中前两个卷积块采用 3x3，3x3，5x5 的卷积核保证了前后感受野的一致性，采用这样的小型卷积是为了降低网络复习度，节约训练成本和缓解过拟合现象，最后再采用一个 4x4 的卷积快进行特征通道变换

3. 激活函数采用 ReLu

4. 特征提取采用 0.4 概率 dropout，分类器采用 0.5 概率 dropout 缓解过拟合

   在训练期间，dropout 层随机将一小部分激活设置为 0，这有助于防止模型过度拟合训练数据，因为它会强制模型更通用并依赖多个特征而不是仅仅几个特征。 在预测过程中，dropout 层通常被关闭，并激活所有的神经元，这将使得模型利用所有可用信息从而提高性能，是一种用于常用的减少神经网络过度拟合的正则化技术。

5. 每层卷积之后采用 BatchNormalization()

   添加 BN 层是为了保证数据真实分布与模型非线性表达能力，批量归一化是一种用于提高神经网络性能和稳定性的技术。 它通过标准化网络中各层的激活来工作，这意味着它将激活缩放为均值为 0 和标准差为 1。这有助于防止在训练深度神经网络时可能发生的梯度消失和爆炸问题 网络，通过将激活保持在合理的范围内。在实际训练中，向模型添加批量归一化层涉及在每个具有可训练参数的层之后插入一个批量归一化层。 该层将前一层的激活作为输入，对其进行归一化，然后将归一化的激活传递给下一层。 归一化是使用每个小批量激活的均值和标准差完成的。

通过以这种方式对激活进行归一化，批量归一化可以显着提高训练的速度和准确性，特别是对于深度神经网络。 它还可以使模型对输入数据分布的变化更加稳健，并有助于减少过度拟合。

6. 采用 MAX 池化对卷积之后的图像特征进行进一步的压缩，大大减少了后面网络层的参数(对图像下采样实现，降维操作，并扩大感受野)，使抽取特征是保证特征局部不变性(旋转平移不变性)是有效的特征， 进一步实现非线性，一定程度上抑制过拟合

7. 分类器：两层全连接 + softmax，输出维度对应数据集中 0-9 的数字类别。

2.5 训练超参数设置

指定了损失函数、优化器类型。在训练期间以识别准确率为跟踪目标。

1、loss 参数指定将用于衡量模型执行情况的损失函数。分类交叉熵是根据预测的类概率与真实类概率之间的差异计算的，对于具有大量类且类别互斥的分类任务是一个很好的选择。

2、optimizer 参数指定将用于在训练期间更新模型权重的优化算法。 在这种情况下，将使用 Adam 优化器。 Adam 是神经网络训练的热门选择，因为它可以根据损失函数的梯度自适应地调整学习率。

3、metrics 参数指定您要在训练期间跟踪的指标列表。 在这种情况下，正在训练模型以优化分类交叉熵损失，并且还会跟踪准确性指标。 准确性是衡量模型能够正确分类训练数据中的示例的程度。

学习率衰减

为了在验证准确性停止提高时降低优化器的学习率，我们增加了对学习率的调度，这可以提高模型的泛化性能并防止过度拟合。

1. monitor：指定用以确定是否应降低学习率的指标。我们使用了“val_accuracy”。
2. patience：如果指定的指标没有改善，则在降低学习率之前要等待的时期数。
3. verbose：决定当学习率改变时是否打印消息。
4. factor：指定当指定指标没有改善时学习率将降低的因子。

5. min_lr：指定将使用的最小学习率。防止学习率变得太小，从而减慢训练速度。

在实际训练时，我们把学习率调度函数与 fit() 函数结合使用，在每个 epoch 之后监控验证准确性，如果验证准确性在指定的 epoch 数内没有提高，则降低学习率。经过对比实验，可以认为这有助于提高模型的泛化性能并防止过度拟合。

```
model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
# 设置一个学习率衰减
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
# 增加数据以防止过拟合
```

图 2.5.1 学习率设计

2.6 PyQt 设计

本项目使用 PyQt 设计 GUI 时，总体需要以下几个组件：

- Main window：这是顶层窗口，将包含 GUI 的所有其他元素。
- Widgets：这些是 GUI 的基本构建块，包括按钮、标签和文本框等元素。
- Layout：这些用于在 GUI 中排列小部件的布局，可以基于绝对位置或相对位置的约束。
- Central Widget：这是放置在主窗口中的小部件，用于包含其他小部件或布局。
- Signals and slots：它们用于将小部件和 GUI 的其他元素连接到事件处理程序，这些事件处理程序在某些事件发生时触发，例如单击按钮。
- Event loop：这是一个侦听事件并在事件发生时对其进行处理的循环，允许 GUI 响应用户输入和其他事件。
- Application object：这是一个代表应用程序的对象，负责管理主事件循环、处理命令行参数以及与操作系统交互。
- QPixmap：一种 off-screen 图像表示形式，可以用作绘画设备。本项目设置背景色为白色，用作画板区域。
- Qpainter：用于执行绘图操作的类，本项目将其与 QPen 和 Qcolor 结合实现画笔功能。

- QColorDialog：用于创建颜色选择窗口，在每次点击修改颜色按钮时，创建颜色选择窗口，用户选择颜色后范围颜色信息用来修改画笔颜色。

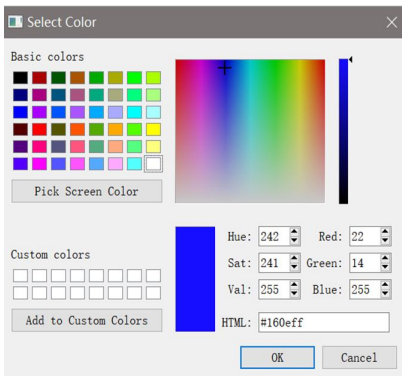


图 2.6.1 QColorDialog 颜色选择界面

2.7 PyQt 具体实现

PaintBoard 类

画笔交互板块，定义绘画区域 board，绘图工具 painter，以及画笔颜色 penColor 等与画笔交互相关的控件及变量，并定义清空画板方法 Clear()，以及用于渲染画笔轨迹的事件：mousePressEvent 和 mouseMoveEvent，即通过获取画笔的移动位置进行实时渲染。

每一次绘制手写数字时，都先鼠标左键再绘制区域按住首先触发 mousePressEvent 事件，获得当前画笔（即鼠标光标）位置，相当于开启本次绘制，之后根据 mouseMoveEvent 画笔发生移动时，就会再过去位置到现在位置的直线路径上渲染当前画笔颜色，也就是进行实时渲染。因此，实现了画笔交互功能。

值得一提的是，PaintBoard 类不仅定义了绘制相关的方法和变量，也封装了包括以 PNG 格式保存绘制区域、清空绘制区域、修改画笔颜色的方法接口，便于主界面类 MainWiddget 调用，以实现保存作品、清空画板、改变画笔颜色的按钮功能。

```python
class PaintBoard(QWidget):

    def __init__(self, Parent=None):
        '''  Constructor  '''
        super().__init__(Parent)

        self.__InitData()  # 先初始化数据，再初始化界面
        self.__InitView()
        self.setWindowTitle("画笔")

    def __InitData(self):
        self.__size = QSize(280, 280)

        # 新建 QPixmap 作为画板，尺寸为__size
        self.__board = QPixmap(self.__size)
        self.__board.fill(Qt.white)  # 用黑色填充画板
        self.__IsEmpty = True  # 默认为空画板
        self.__lastPos = QPoint(0, 0)  # 上一次鼠标位置
        self.__currentPos = QPoint(0, 0)  # 当前的鼠标位置
        self.__painter = QPainter()  # 新建绘图工具
        self.__thickness = 10  # 默认画笔粗细为10px
        self.__penColor = QColor("black")  # 设置默认画笔颜色为白色
        self.__pen = QPen(self.__penColor, self.__thickness) # 设置画笔

    def __InitView(self):
        # 设置界面的尺寸为__size
        self.setFixedSize(self.__size)

    def Clear(self):
        # 清空画板
        self.__board.fill(Qt.white)
        self.update()
        self.__IsEmpty = True

    def IsEmpty(self):
        # 返回画板是否为空
        return self.__IsEmpty

    def GetContentAsQImage(self):
        # 获取画板内容（返回QImage）
        image = self.__board.toImage()
        return image

    def paintEvent(self, paintEvent):
        # 绘图事件
        # 绘图时必须使用QPainter的实例，此处为__painter
        # 绘图在begin()函数与end()函数间进行
        # begin(param)的参数要指定绘图设备，即把图画在哪里
        # drawPixmap用于绘制QPixmap类型的对象
```

```
        self.__painter.begin(self)
        # 0,0 为绘图的左上角起点的坐标，__board 即要绘制的图
        self.__painter.drawPixmap(0, 0, self.__board)
        self.__painter.end()

    def mousePressEvent(self, mouseEvent):
        # 鼠标按下时，获取鼠标的当前位置保存为上一次位置
        self.__currentPos = mouseEvent.pos()
        self.__lastPos = self.__currentPos

    def mouseMoveEvent(self, mouseEvent):
        # 鼠标移动时，更新当前位置，并在上一个位置和当前位置间画线
        self.__currentPos = mouseEvent.pos()
        self.__painter.begin(self.__board)
        self.__painter.setPen(self.__pen)  # 设置画笔颜色，粗细
        # 画线
        self.__painter.drawLine(self.__lastPos, self.__currentPos)
        self.__painter.end()
        self.__lastPos = self.__currentPos
        self.update()  # 更新显示

    def mouseReleaseEvent(self, mouseEvent):
        self.__IsEmpty = False  # 画板不再为空

    def ChangePenColor(self, color):
        self.__pen.setColor(color) # 修改画笔颜色
```

MainWidget 类

整体界面类以及鼠标选择交互板块，定义整体界面的控件，布局以及各按钮控件绑定的方法，例如颜色选择按钮：ChangeColor()、预测按钮：Predict()、退出按钮：Quit()等。

布局按照总体架构设计，整体采用左右布局，左侧右侧各嵌套一个上下布局，用于按钮控件的放置；ChangeColor()方法定义 QColorDialog 类弹出颜色选择窗口，并中断主界面运行，直到颜色选择窗口关闭并返回所选择的颜色，并调用 PaintBoard 类封装的 ChangePenColor 方法改变画笔颜色；Predict()方法是整个用户界面设计的核心只之一，首先加载训练好的神经网络模型，其次获取绘制区域的绘制作品，并生成对应的灰度图。然后对图像进行二值化，归一化，送入模型预测，最后调用 QTextBrowser.append()方法显示返回的预测结果。

```python
class MainWidget(QWidget):
    def __init__(self, Parent=None):
        '''   Constructor   '''
        super().__init__(Parent)

        self.__InitData()  # 先初始化数据，再初始化界面
        self.__InitView()

    def __InitData(self):
        '''   初始化成员变量  '''
        self.__paintBoard = PaintBoard(self)

    def __InitView(self):
        '''    初始化界面     '''
        self.setFixedSize(550, 380)
        self.setWindowTitle("手写数字识别")

        # 新建一个水平布局作为本窗体的主布局
        main_layout = QHBoxLayout(self)
        # 设置主布局内边距以及控件间距为10px
        main_layout.setSpacing(10)

        sub_layout2 = QVBoxLayout()
        sub_layout2.setContentsMargins(10, 10, 0, 10)
        sub_layout2.addWidget(self.__paintBoard)

        # 调节画笔颜色的按钮
        sub_layout3 = QHBoxLayout()
        sub_layout3.setSpacing(10)

        # 颜色显示
        self.__colorwidget = QWidget(self)
        self.__colorwidget.setAutoFillBackground(True)
        palette = QPalette()
        palette.setColor(QPalette.Window,Qt.black)
        self.__colorwidget.setPalette(palette)
        sub_layout3.addWidget(self.__colorwidget)

        # 设置按钮
        self.__btn_ChangeColor = QPushButton("画笔颜色")
        self.__btn_ChangeColor.setParent(self)
        self.__btn_ChangeColor.clicked.connect(self.ChangePainterColor)
        sub_layout3.addWidget(self.__btn_ChangeColor)

        sub_layout2.addLayout(sub_layout3)

        # 在主界面左侧放置画板
        main_layout.addLayout(sub_layout2)

        # 新建垂直子布局用于放置按键
        sub_layout = QVBoxLayout()
```

```
        # 设置此子布局和内部控件的间距为10px
        sub_layout.setContentsMargins(10, 10, 10, 10)

        self.__btn_Clear = QPushButton("清空画板")
        self.__btn_Clear.setParent(self)  # 设置父对象为本界面

        # 将按键按下信号与画板清空函数相关联
        self.__btn_Clear.clicked.connect(self.__paintBoard.Clear)
        sub_layout.addWidget(self.__btn_Clear)


        self.__btn_Save = QPushButton("保存作品")
        self.__btn_Save.setParent(self)
        self.__btn_Save.clicked.connect(self.on_btn_Save_Clicked)
        sub_layout.addWidget(self.__btn_Save)

        self.__btn_Predict = QPushButton("预测")
        self.__btn_Predict.setParent(self)  # 设置父对象为本界面
        self.__btn_Predict.clicked.connect(self.Predict)
        sub_layout.addWidget(self.__btn_Predict)

        self.__btn_Quit = QPushButton("退出")
        self.__btn_Quit.setParent(self)  # 设置父对象为本界面
        self.__btn_Quit.clicked.connect(self.Quit)
        sub_layout.addWidget(self.__btn_Quit)
        self.__text_browser = QTextBrowser(self)
        self.__text_browser.setParent(self)
        sub_layout.addWidget(self.__text_browser)

        splitter = QSplitter(self)  # 占位符
        sub_layout.addWidget(splitter)

        main_layout.addLayout(sub_layout)  # 将子布局加入主布局

    def __fillColorList(self, comboBox):
        index_black = 0
        index = 0
        for color in self.__colorList:
            if color == "black":
                index_black = index
            index += 1
            pix = QPixmap(70, 20)
            pix.fill(QColor(color))
            comboBox.addItem(QIcon(pix), None)
            comboBox.setIconSize(QSize(70, 20))
            comboBox.setSizeAdjustPolicy(QComboBox.AdjustToContents)
         comboBox.setCurrentIndex(index_black)

    def on_PenColorChange(self):
        color_index = self.__comboBox_penColor.currentIndex()
        color_str = self.__colorList[color_index]
        self.__paintBoard.ChangePenColor(color_str)
```

```
    def on_PenThicknessChange(self):
        penThickness = self.__spinBox_penThickness.value()
        self.__paintBoard.ChangePenThickness(penThickness)

    def on_btn_Save_Clicked(self):
        image = self.__paintBoard.GetContentAsQImage()
        image.save('1.png')

    def ChangePainterColor(self):
        #创建颜色窗口
        color_d = QColorDialog()
        #颜色变化相应
        color_d.currentColorChanged.connect(self.on_change_color)
        #锁住程序直到用户关闭该对话框为止
        color_d.exec_()

    def on_change_color(self, color):
        #创建 Qpalette 类
        palette = QPalette()
        #用来设置 ColorRole
        palette.setColor(QPalette.Window,color)
        #设置 widget 填充颜色
        self.__colorwidget.setAutoFillBackground(True)
        #设置修改好的 palette
        self.__colorwidget.setPalette(palette)
        # 修改画笔颜色
        self.__paintBoard.ChangePenColor(color)

     def Predict(self):
        # 清空文本框
        self.cursot = self.__text_browser.clear()
        # 调用模型
        newmodel = models.load_model('new_model.h5')
        # 读取图片
        image = self.__paintBoard.GetContentAsQImage()
        image.save('predict.png')
        img = cv2.imread('predict.png', 0)
        os.remove('predict.png')

        plt.imshow(img)
        # print(img.shape)
        img = cv2.resize(img, (28, 28))

        rows = img.shape[0]
        cols = img.shape[1]
        for i in range(rows):
            for j in range(cols):
                if (img[i, j] > 150):
                    img[i, j] = 255
                else:
                    img[i, j] = 0
```

```
        # cv2.imshow("img", img)

        img = img.reshape(1, 28, 28, 1)
        img = img / 255  # 归一化
        # print(img.shape)

        predict = newmodel.predict(img)
        print(predict)
        np.argmax(predict)
        # print("预测图像中的数字为：" + str(np.argmax(predict)))
        self.__text_browser.append("预测图像中的数字为：" +
str(np.argmax(predict)))
        self.cursot = self.__text_browser.textCursor()
        self.__text_browser.moveCursor(self.cursot.End)


  def Quit(self):
        self.close()
```

Main()

```
def main():
    app = QApplication(sys.argv)

    mainWidget = MainWidget() #新建一个主界面
    mainWidget.show()    #显示主界面

    exit(app.exec_()) #进入消息循环
```

三.实验结果展示和分析

3.1 网络模型的训练测试结果

采用 model.fit 进行封装训练，验证，测试过程内部是模型进行训练双重循环遍历 epoch 和 batch,按 batch 大小从对应的 dataloader 里面取数据进行训练，验证和测试，将一 batch 量的数据输入模型前馈计算得出相应样本的匹配结果 outputs 和对应 label 输入交叉熵损失函数中计算损失，并计算一轮的评估函数 acc 和 loss 计算，利用 BP 算法，方向传播->更新参数->清除梯度(更新模型参数，优化器参数，学习率衰减)每轮输出训练损失函数 loss 值和验证损失函数 val_loss，以及训练准确

率 acc，验证准确率 val_loss，并进行记录，最后结束时调用可视化函数画出 loss 曲线，acc 曲线，并输出测试集上预测的准确率。

```
datagen.fit(X_train)

# 拟合模型
history = model.fit(datagen.flow(X_train,Y_train, batch_size=128),
                              epochs = 30, validation_data = (X_val,Y_val),
                              verbose = 2, steps_per_epoch=X_train.shape[0] // 128,
                              callbacks=[learning_rate_reduction])
```

图 3.1.1 模型训练

128 个数据为一个 batch，一个 epoch 共训练 421 个 batch。共 30 个 epochs。

部分过程

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
Epoch 1/30
421/421 - 35s - loss: 0.3545 - accuracy: 0.8906 - val_loss: 11.9062 - val_accuracy: 0.1128 - lr: 0.0010 - 35s/epoch - 83ms/step
Epoch 2/30
421/421 - 17s - loss: 0.1024 - accuracy: 0.9705 - val_loss: 0.0396 - val_accuracy: 0.9902 - lr: 0.0010 - 17s/epoch - 41ms/step
Epoch 3/30
421/421 - 17s - loss: 0.0764 - accuracy: 0.9782 - val_loss: 0.0308 - val_accuracy: 0.9908 - lr: 0.0010 - 17s/epoch - 40ms/step
Epoch 4/30
421/421 - 23s - loss: 0.0667 - accuracy: 0.9816 - val_loss: 0.0395 - val_accuracy: 0.9877 - lr: 0.0010 - 23s/epoch - 54ms/step
Epoch 5/30
421/421 - 17s - loss: 0.0593 - accuracy: 0.9830 - val_loss: 0.0390 - val_accuracy: 0.9907 - lr: 0.0010 - 17s/epoch - 41ms/step
Epoch 6/30
421/421 - 17s - loss: 0.0540 - accuracy: 0.9851 - val_loss: 0.0266 - val_accuracy: 0.9922 - lr: 0.0010 - 17s/epoch - 41ms/step
Epoch 7/30
421/421 - 19s - loss: 0.0526 - accuracy: 0.9856 - val_loss: 0.0409 - val_accuracy: 0.9895 - lr: 0.0010 - 19s/epoch - 44ms/step
Epoch 8/30
421/421 - 18s - loss: 0.0447 - accuracy: 0.9875 - val_loss: 0.0369 - val_accuracy: 0.9912 - lr: 0.0010 - 18s/epoch - 42ms/step
Epoch 9/30

Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
421/421 - 18s - loss: 0.0457 - accuracy: 0.9870 - val_loss: 0.0312 - val_accuracy: 0.9913 - lr: 0.0010 - 18s/epoch - 43ms/step
Epoch 10/30
421/421 - 17s - loss: 0.0331 - accuracy: 0.9907 - val_loss: 0.0240 - val_accuracy: 0.9925 - lr: 5.0000e-04 - 17s/epoch - 40ms/step
```

图 3.1.2 训练过程

最终结果:

```
Epoch 24: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
421/421 - 17s - loss: 0.0131 - accuracy: 0.9965 - val_loss: 0.0179 - val_accuracy: 0.9958 - lr: 1.2500e-04 - 17s/epoch - 40ms/step
Epoch 25/30
421/421 - 17s - loss: 0.0113 - accuracy: 0.9966 - val_loss: 0.0187 - val_accuracy: 0.9955 - lr: 6.2500e-05 - 17s/epoch - 40ms/step
Epoch 26/30
421/421 - 17s - loss: 0.0107 - accuracy: 0.9969 - val_loss: 0.0176 - val_accuracy: 0.9955 - lr: 6.2500e-05 - 17s/epoch - 40ms/step
Epoch 27/30

Epoch 27: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
421/421 - 17s - loss: 0.0104 - accuracy: 0.9968 - val_loss: 0.0169 - val_accuracy: 0.9960 - lr: 6.2500e-05 - 17s/epoch - 40ms/step
Epoch 28/30
421/421 - 17s - loss: 0.0100 - accuracy: 0.9969 - val_loss: 0.0177 - val_accuracy: 0.9957 - lr: 3.1250e-05 - 17s/epoch - 41ms/step
Epoch 29/30
421/421 - 17s - loss: 0.0111 - accuracy: 0.9967 - val_loss: 0.0175 - val_accuracy: 0.9962 - lr: 3.1250e-05 - 17s/epoch - 40ms/step
Epoch 30/30
421/421 - 19s - loss: 0.0094 - accuracy: 0.9970 - val_loss: 0.0179 - val_accuracy: 0.9960 - lr: 3.1250e-05 - 19s/epoch - 44ms/step
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0109 - accuracy: 0.9970
[0.010933740995824337, 0.996999979019165]
```

图 3.1.3 训练结果

可以看到：

在训练集上，最终评估结果：[loss：0.0094, accuracy：0.9970]

在验证集上，最终评估结果：[val_loss：0.0179,val_ accuracy：0.9960]

在测试集上，最终评估结果：[test_loss：0.0109334, test_accuracy：0.9969999]

最终准确度约为：0.9970

记录训练中关键指标数值波动曲线，如下图所示

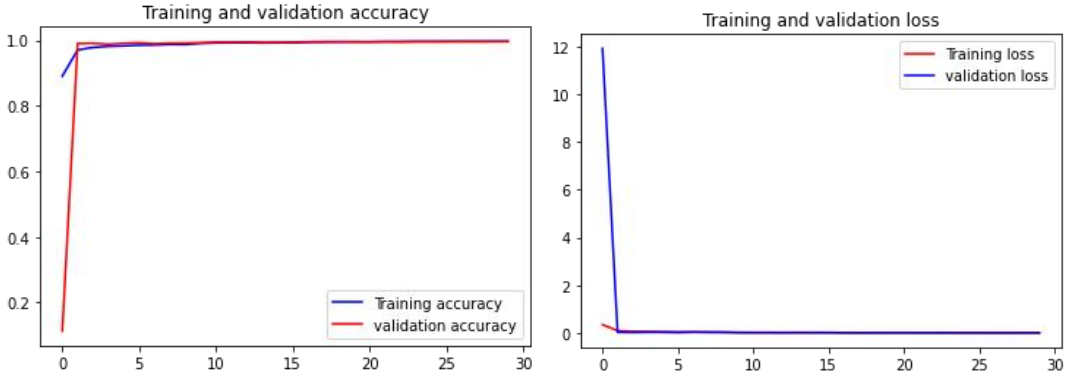


图 3.1.4 训练结果

对应的全连接神经网络结构可视化图像如下:

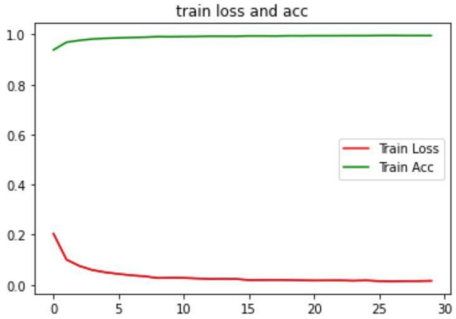


图 3.1.5 损失图像

可以看到曲线趋势与之前训练结果类似，无论是在训练集，验证集，还是测试集准确率都更高，收敛过程更平滑。

可以认为模型训练效果很好，评估测试达到 99.7%以上的高精度，识别正确率甚至超过了人类，而普通的全连接神经网络只能达到 98.5%，说明卷积神经网络相比于普通的全连接网络结构更能有效抽取到图像特征，从图像空间区域中学习更抽象丰富有用的语义信息，能有效考虑像素点之间的联系，学到图像空间的丰富知识，进而获得更高的精度结果，并且基于 dropout，BN 等 trick 解决过拟合等不良现象的产生，保证了模型的高精度，高泛化能力。

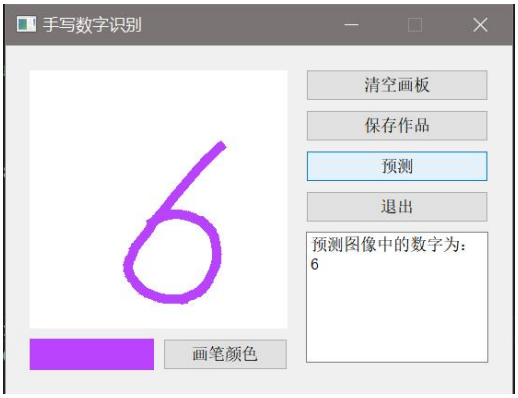
3.2 QT 交互界面效果展示

基于上文中的界面设计架构，本项目采用左右分布的总体布局，左侧放置绘图区域、右侧放置各类按钮接口。对于这些按钮接口，本项目采用上下布局的方式放置。其中的“画笔颜色修改”按钮较为特殊，由于其与画笔交互板块强相关，本项目将其置于绘图区域的下方，保证交互界面的美观性，所见即所得。拥有手写数字，清空画板，保存作品，预测，退出界面等交互功能，实现画笔颜色调整的个性化定制功能。最终用户界面设计如下图所示：

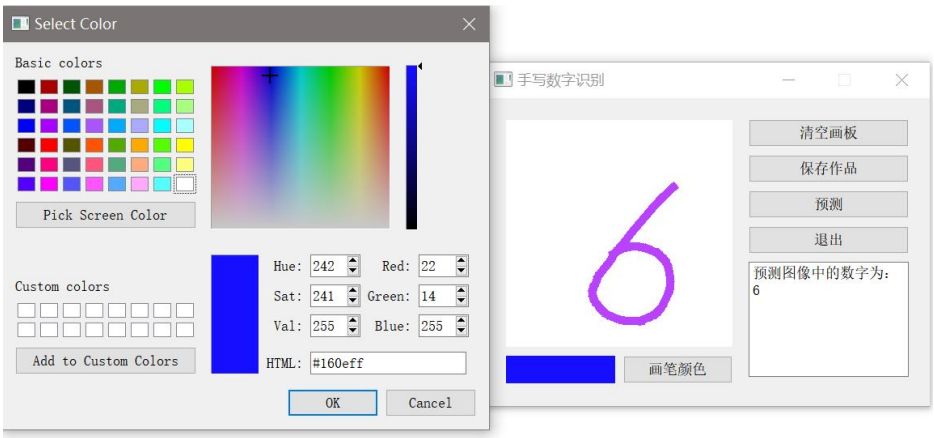
图 3.2.1：用户界面最终设计结果

交互效果:

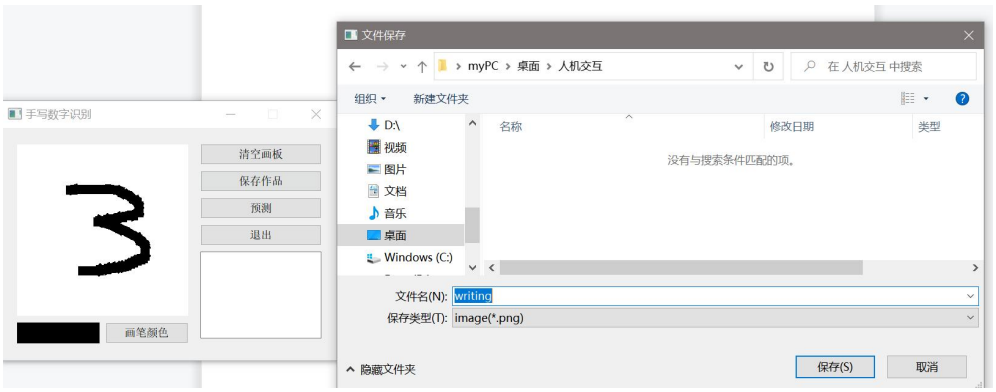
1. 手写数字识别，实现高精度预测



2. 修改画笔颜色个性化定制



3. 保存作品，收集用户数据特色



四.总结展望

本项目通过基于 TensorFlow 构建的卷积神经网络模型，通过 Python 编程实现完善的 QT 界面，从而构建出手写数字识别系统。

卷积神经网络部分

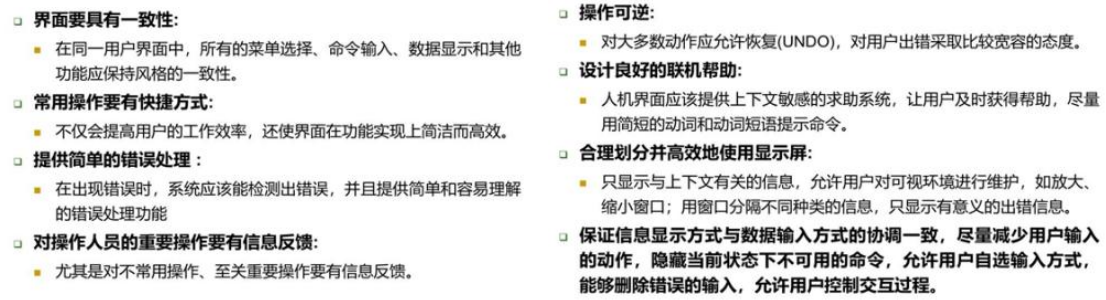
我们在经典训练集图像基础上进行数据增广，将其作为输入，省去了复杂的特征提取与加注标签的过程，还可以使模型更全面地从大量样本中学习到相应的特征，增强了训练效果。神经网络主体结构包括卷积层、池化层、激活函数等。经过多个神经网络结构的比对，权衡模型结构复杂度、训练时间以及预测准度，我组最终选择此网络进行训练。在我们的神经网络模型中，每个卷积层后面都会添加一个批量归一化层，通过批量归一化将特征值分布重新拉回标准正态分布，特征值将落在激活函数对于输入较为敏感的区间，输入的小变化可导致损失函数较大的变化，使得梯度变大，避免梯度消失，同时批量操作也可加快收敛，从而加快了训练速度，并且提高了精度。在模型中加入了 dropout 操作，有效地防止了模型对于部分特征的过度依赖，使得模型利用到了所有的信息，从而提高了精度，最终可以达到 99.7% 以上，通过实验进行验证卷积神经网络的效果要优于全连接神经网络。

人机交互部分

基于桌面隐喻、所见即所得、直接操纵的思想，我们通过 QT 界面，将每个按钮的具体操作功能表明，使得用户可以更加方便的进行操纵，同时我们留足了手写空间，方便用户的使用，界面足够简洁，但功能十分完善，并为用户提供个性化操作，不同颜色的书写可以应用于不同的场合。最终构建出完整的手写数字识别系统。综上所述，本系统具备良好的界面设计，同时功能实现优秀，能够达到较好的识别效果，实现手写数字识别的人机交互智能系统。

在未来将在加入多字符识别，扩大绘图区域，增加多个字符一起识别的功能，使得程序弄够一次性识别多个绘制出的数字。加入文字与字母识别，可以增加输入框选项，点击即可在绘图区域创建输入框输入文字或者字母，并支持输入框的拖动，最终进行文字和字母识别，同样，也可以增加手写文字或字母的识别，同方向一结

合识别手写的中文句子，英文句子提升实用的广泛性。并提高交互系统预测精度，对预测图片进一步预处理加入平滑去噪、 字符切分、笔画细化等操作，再次进一步优化界面设计。未来我们将进行上述的改进方向做下去，并有志应用于金融，电子交易与存储记录，线上办公，教育领域在不同场合为用户提供个性化操作，把握人机交互设计原则，为用户提供功能强大，便捷，优质的，准确的高质量交互体验。



五.文献引用

[1] HE K, ZHANG X, REN S, 等. Deep Residual Learning for Image Recognition[M/OL]. arXiv, 2015[2023-01-06]. http://arxiv.org/abs/1512.03385.

[2] GLOROT X, BORDES A, BENGIO Y. Deep Sparse Rectifier Neural Networks[C]//Journal of Machine Learning Research: 卷 15. 2010.

[3] SZEGEDY C, LIU W, JIA Y, 等. Going Deeper with Convolutions[M/OL]. arXiv, 2014[2023-01-06]. http://arxiv.org/abs/1409.4842.

[4] Gradient-based learning applied to document recognition | IEEE Journals & Magazine | IEEE Xplore[EB/OL]. [2023-01-06]. https://ieeexplore.ieee.org/abstract/document/726791.

[5] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet classification with deep convolutional neural networks[J/OL]. Communications of the ACM, 2017, 60(6): 84-90. https://doi.org/10.1145/3065386.

[6] SZEGEDY C, IOFFE S, VANHOUCKE V, 等. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning[M/OL]. arXiv, 2016[2023-01-06]. http://arxiv.org/abs/1602.07261.

[7] SZEGEDY C, VANHOUCKE V, IOFFE S, 等. Rethinking the Inception Architecture for Computer Vision[M/OL]. arXiv, 2015[2023-01-06]. http://arxiv.org/abs/1512.00567.

[8] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J/OL]. 2014[2023-01-06]. https://arxiv.org/abs/1409.1556v6.

附 录

train.py：网络结构搭建+数据预处理+模型训练+模型验证测试

shibie.py: 用户界面布局+画笔、鼠标交互板块+按钮接口处理逻辑

用户在配置完环境后，直接运行 shibie.py 即可，实现全部功能