**物联网创想秀软件组**

**技术报告**

**学年学期 2017-2018学年第二学期**

**队 伍 名 XXX**

**队 长**

**队 员**

**选用语言 java、Python**

**应用方向 桌面应用**

**目 录**

[引言 1](#_Toc514099957)

[1 业务与功能分析 1](#_Toc514099958)

[1.1 系统的业务流程 2](#_Toc514099959)

[1.2 数据需求 2](#_Toc514099960)

[1.3 功能需求 4](#_Toc514099961)

[2 系统设计 5](#_Toc514099962)

[3 系统实现与测试 7](#_Toc514099963)

[3.1 系统操作界面 7](#_Toc514099964)

[3.2 系统功能实现 13](#_Toc514099965)

[3.3 系统测试 15](#_Toc514099966)

[4 总结与展望 15](#_Toc514099967)

[参考文献 16](#_Toc514099968)

# 引言

自上世纪六十年代以来，计算机视觉与图像的处理越来越受到人们的关注，并逐渐成为一门重要的学科领域。而作为它们的研究对象的数字图像，也因为它含有研究目标的丰富信息而成为越来越重要的研究对象。图像识别的目标是用计算机自动完成某些信息的处理，用来代替人工去处理图像分类及识别的任务。

手写数字识别是图像识别学科的下一个分支，是图像处理和模式识别领域研究的课题之一，由于其具有很强的实用性一直是多年来的研究热点。由于手写体的随意性很大，例如，笔画的粗细，字体的大小，倾斜等等都直接影响到数字的正确识别，所以手写体数字识别是一个很有挑战性的课题。手写体数字识别实用性很强，在大规模数据统计（如例行车检，人口普查），财务，税务，邮件分拣等应用领域中都有广阔的应用前景。

# 业务与功能分析

本应用设计实现了一个基于swing界面、应用java、python语言的手写数字识别系统，采用了模块化设计方法，主要分为手写板输入和直接读取图片两个模块，实现对单个手写数字的识别。Java主要负责界面平台的搭建以及Python脚本的调用，Python则负责模型的构建和算法的实现。

## 系统的业务流程

调用

结果文件

## 1.2 数据需求

本功能的主要数据需求是要求图像的格式以及具体呈现形式与minst数据集的严格匹配。对于通过手写板输出的图片以及需要检测的图片，都需要进行图像处理已达到所需的效果。通过对普通二值图像进行预处理、分析、切割、压缩、扩展，使数字在28\*28的png格式的图片中居中显示，同时对于RGB图像要先进行灰度化、归一化和二值化，再通过与二值图像相同的操作步骤达到所需要求。经过处理原始图像变化过程为：

## 1.3 功能需求

根据对系统设计过程的具体分析，除了命题本身的需求以外，还应包含以下功能：

1. 数字图片的预处理

因为特征库用的是minst数据库，为了提高模型的泛化能力，我们对minst数据集进行了仔细的研究，总结了如下特点：

1. 28\*28像素，黑底白字
2. 图像位于正中间，**且上下左右都至少留4个像素点**

基于这些，我们需将数字图片处理以达到minst数据集的要求。

1. 数字识别

在手写数字识别中，我们使用的方法是bp神经网络法，主要通过正向传播过程得出预测值，再比较预测值与真实值，利用反向传播和梯度下降法不断更新各层的权重，并利用学习完得到的权重预测。

实际过程中隐藏层层数，每一层的节点数目，学习率……都需要多次调试,来提高正确率。

# 系统设计

2.1系统应用结构

2.2系统应用功能



* 画板：java用swing组件做了一个画板，可以用鼠标点击拖动实现画图，橡皮按钮可以进行对已画数字的修改，重置按钮则可以清除画板。
* 颜色选择：画板和按钮的下面可以实现对颜色的选择，画出不同颜色的数字。
* 图片按钮：点击图片按钮，出现选择图片张数的面板，选择张数后选择图片，显示图片后点击识别，进行多张数字的识别。
* 确定按钮：对画板上的数字进行截屏，传到后台进行手写数字的识别。

# 系统实现与测试

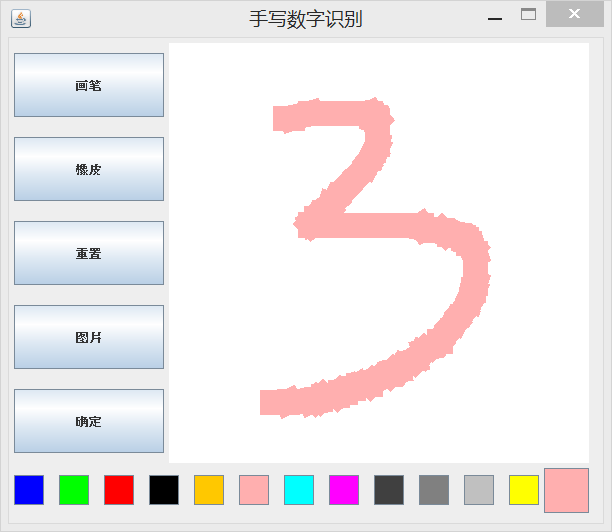
## 系统操作界面



左侧为按钮组，可以选择画图，橡皮，重置画板，图片显示图片，确定识别数字；右侧为画板，选择画笔后可拖动鼠标在画板上画图；下侧为选择颜色的按钮组，可以根据需求选择喜欢的颜色。



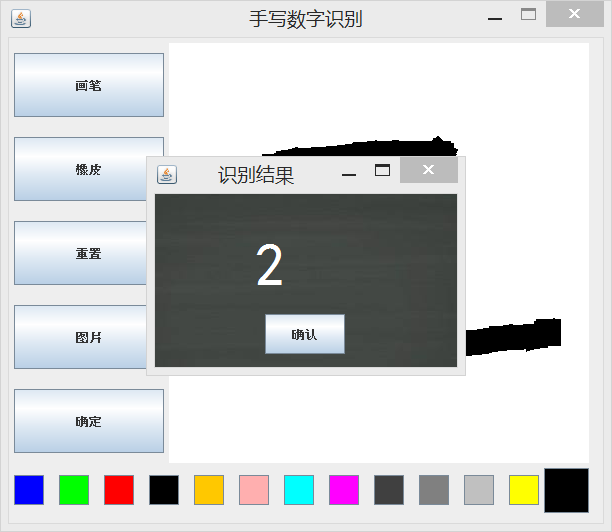
画笔画图。



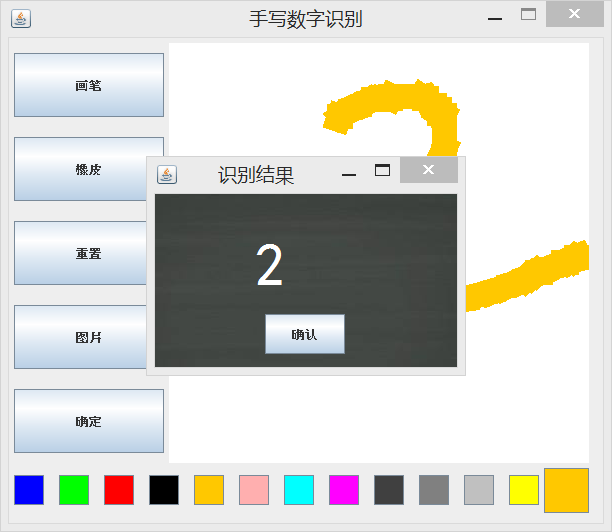
更换颜色。



橡皮擦除。



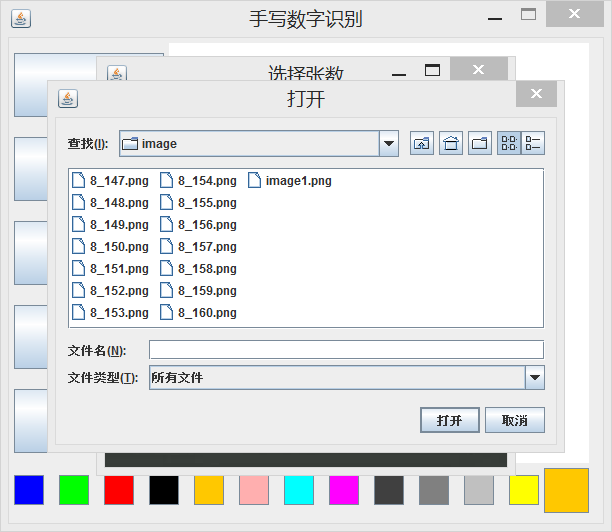
画板上黑色数字识别。



画板上彩色数字识别。



选择图片—选择要识别的图片张数。



弹出选择文件的对话框，选择所要识别数字的图片。



将选择的图片显示在面板上。



识别多张图片。



也可以只识别一张。



显示识别结果。

## 系统功能实现

**Python部分:**

* **神经网络算法**

**class** **NeuralNetwork:**#创建神经网络类

**def** \_\_init\_\_**(**self**,** layers**,** activation**=**'tanh'**):**

# 1.layers代表神经网络的结构

# 是一个列表

# 如【784,300,10】代表输入层中有784个节点，一个隐藏层中有300个节点，输出层有10个节点

# 2.activation代表激励函数，分logistics和tanh两种函数

# activation\_derivative代表激励函数的导数

**if** activation **==** 'logistic'**:**

self**.**activation **=** logistic

self**.**activation\_derivative **=** logistic\_derivative

**elif** activation **==** 'tanh'**:**

self**.**activation **=** tanh

self**.**activation\_derivative **=** tanh\_derivative

self**.**weights **=** **[]**

# 随机初始化weights，范围为-0.25~0.25

**for** i **in** range**(**1**,** len**(**layers**)** **-** 1**):**

self**.**weights**.**append**((**2**\***np**.**random**.**random**((**layers**[**i **-** 1**]** **+** 1**,** layers**[**i**]** **+** 1**))-**1**)\***0.25**)**

self**.**weights**.**append**((**2**\***np**.**random**.**random**((**layers**[**i**]** **+** 1**,** layers**[**i **+** 1**]))-**1**)\***0.25**)**

#定义激励函数以及导数

**def** tanh**(**x**):**

**return** np**.**tanh**(**x**)**

**def** tanh\_derivative**(**x**):**

**return** 1.0 **-** np**.**tanh**(**x**)** **\*** np**.**tanh**(**x**)**

**def** logistic**(**x**):**

**return** 1 **/** **(**1 **+** np**.**exp**(-**x**))**

**def** logistic\_derivative**(**x**):**

**return** logistic**(**x**)** **\*** **(**1 **-** logistic**(**x**))**

# fit方法就是训练模型的过程，通过正向传播和反向传播不断更新weights

**def** fit**(**self**,** X**,** y**,** learning\_rate**=**0.15**,** epochs**=**60000**):**

#learning\_rate:学习率，epochs：迭代次数（为了提高速度，每一个输入向量都只进行一次正向传播和一次负向传播）

X **=** np**.**atleast\_2d**(**X**)**

#给输入的矩阵加上一行，便于计算bias

temp **=** np**.**ones**([**X**.**shape**[**0**],** X**.**shape**[**1**]+**1**])**

temp**[:,** 0**:-**1**]** **=** X

X **=** temp

y **=** np**.**array**(**y**)**

**for** k **in** range**(**epochs**):**

i **=** np**.**random**.**randint**(**X**.**shape**[**0**])**#随机学习

a **=** **[**X**[**i**]]**

# 对每一层进行正向传播

**for** l **in** range**(**len**(**self**.**weights**)):**

a**.**append**(**self**.**activation**(**np**.**dot**(**a**[**l**],** self**.**weights**[**l**])))**

error **=** y**[**i**]** **-** a**[-**1**]** #计算正向传播得到的输出值与真实值的差距

deltas **=** **[**error **\*** self**.**activation\_derivative**(**a**[-**1**])]** #最后一层的导数

#对每一层进行反向传播

**for** l **in** range**(**len**(**a**)** **-** 2**,** 0**,** **-**1**):** # 从倒数第二层开始

deltas**.**append**(**deltas**[-**1**].**dot**(**self**.**weights**[**l**].**T**)\***self**.**activation\_derivative**(**a**[**l**]))**

deltas**.**reverse**()**

**for** i **in** range**(**len**(**self**.**weights**)):**

layer **=** np**.**atleast\_2d**(**a**[**i**])**

delta **=** np**.**atleast\_2d**(**deltas**[**i**])**

self**.**weights**[**i**]** **+=** learning\_rate **\*** layer**.**T**.**dot**(**delta**)**#利用梯度下降的方法更新weights

np**.**save**(**"weights"**,**self**.**weights**)**#保存学习完得到的weights

**def** predict**(**self**,** x**):**

self**.**weights**=**np**.**load**(**"weights.npy"**)**#导入生成的weights

x **=** np**.**array**(**x**)**

temp **=** np**.**ones**(**x**.**shape**[**0**]+**1**)**

temp**[**0**:-**1**]** **=** x #同样给输入的矩阵加上一行

a **=** temp

**for** l **in** range**(**0**,** len**(**self**.**weights**)):**#类似于正向传播的过程得到预测值

a **=** self**.**activation**(**np**.**dot**(**a**,** self**.**weights**[**l**]))**

**return** a

* **图像处理**

#图像处理各个函数

#数据归一化

**def** Normalization**(**dataset**):**

temp**=**dataset**-**np**.**tile**(**dataset**.**min**(),**dataset**.**shape**)**

maxmatrix**=**np**.**tile**(**temp**.**max**(),**dataset**.**shape**)**

**return** temp**/**maxmatrix

#RGB图像灰值化

**def** rgb2gray**(**rgb**):**

**return** np**.**dot**(**rgb**[...,:**3**],[**0.299**,**0.587**,**0.114**])**

#判断边界

**def** JudgeEdge**(**img\_array**):**

height **=** len**(**img\_array**)**

width **=** len**(**img\_array**[**0**])**

size **=** **[-**1**,** **-**1**,** **-**1**,** **-**1**]**

**for** i **in** range**(**height**):**

high **=** img\_array**[**i**]**

low **=** img\_array**[**height **-** 1 **-** i**]**

**if** len**(**high**[**high **>** 0**])** **>** 0 **and** size**[**0**]==-**1**:**

size**[**0**]** **=** i

**if** len**(**low**[**low **>** 0**])** **>** 0 **and** size**[**1**]==-**1**:**

size**[**1**]** **=** height **-** 1 **-** i

**if** size**[**1**]** **!=** **-**1 **and** size**[**0**]** **!=** **-**1**:**

**break**

**for** i **in** range**(**width**):**

left **=** img\_array**[:,** i**]**

right **=** img\_array**[:,** width **-** 1 **-** i**]**

**if** len**(**left**[**left **>** 0**])** **>** 0 **and** size**[**2**]==-**1**:**

size**[**2**]** **=** i

**if** len**(**right**[**right **>** 0**])** **>** 0 **and** size**[**3**]==-**1**:**

size**[**3**]** **=** width **-** i **-** 1

**if** size**[**2**]** **!=** **-**1 **and** size**[**3**]** **!=** **-**1**:**

**break**

**return** size

#图片处理主函数：包括灰值化、归一化、背景转换、消噪、切割、压缩扩展

**def** GetCutZip**(**imagename**):**

img **=** Image**.**open**(**imagename**)**

img\_array **=** np**.**array**(**img**)**

#判断图片类型，若为rgb图像则灰值化

**if** img\_array**.**ndim **==** 3**:**

img\_array **=** rgb2gray**(**img\_array**)**

#归一化

img\_array**=**Normalization**(**img\_array**)**

#背景转化

arr1**=(**img\_array**>=**0.9**)**

arr0**=(**img\_array**<=**0.1**)**

**if** arr1**.**sum**()>** arr0**.**sum**():**

img\_array **=** 1 **-** img\_array

#消噪

img\_array**[**img\_array**>**0.7**]=**1

img\_array**[**img\_array**<**0.4**]=**0

edge **=** JudgeEdge**(**img\_array**)**

cut\_array **=** img\_array**[**edge**[**0**]:**edge**[**1**]** **+** 1**,** edge**[**2**]:**edge**[**3**]** **+** 1**]**

cut\_img **=** Image**.**fromarray**(**np**.**uint8**(**cut\_array **\*** 255**))**

#等比例压缩成20\*20的图像

**if** cut\_img**.**size**[**0**]<=**cut\_img**.**size**[**1**]:**

zip\_img **=** cut\_img**.**resize**((**20 **\*** cut\_img**.**size**[**0**]** **//** cut\_img**.**size**[**1**],** 20**),** Image**.**ANTIALIAS**)**

**else:**

zip\_img **=**cut\_img**.**resize**((**20**,**20**\***cut\_img**.**size**[**1**]//**cut\_img**.**size**[**0**]),**Image**.**ANTIALIAS**)**

zip\_img\_array **=** np**.**array**(**zip\_img**)**

#扩展成28\*28,图像位于正中心

final\_array **=** np**.**zeros**((**28**,** 28**))**

height **=** len**(**zip\_img\_array**)**

width **=** len**(**zip\_img\_array**[**0**])**

high **=** **(**28 **-** height**)** **//** 2

left **=** **(**28 **-** width**)** **//** 2

final\_array**[**high**:**high **+** height**,** left**:**left **+** width**]** **=** zip\_img\_array

final\_array**=**Normalization**(**final\_array**)**

**return** final\_array

* **其他算法**
* **导入minst训练集**

**def** load\_minist**(**labels\_path**,**images\_path**):**

**with** open**(**labels\_path**,**'rb'**)** **as** lbpath**:**

magic**,**n **=**struct**.**unpack**(**'>II'**,**lbpath**.**read**(**8**))**

labels**=**np**.**fromfile**(**lbpath**,**dtype**=**np**.**uint8**)**

**with** open**(**images\_path**,**"rb"**)as** imgpath**:**

magic**,**num**,**rows**,**cols**=**struct**.**unpack**(**'>IIII'**,**imgpath**.**read**(**16**))**

images**=**np**.**fromfile**(**imgpath**,**dtype**=**np**.**uint8**).**reshape**(**len**(**labels**),**784**)**

**return** images**,**labels

images**,**labels**=**load\_minist**(**'train-labels.idx1-ubyte'**,**'train-images.idx3-ubyte'**)**

test\_images**,**test\_labels**=**load\_minist**(**'t10k-labels.idx1-ubyte'**,**'t10k-images.idx3-ubyte'**)**

* **标签二值化**

#例如：6转化为[0,0,0,0,0,0,1,0,0,0]

**def** LabelBinarizer**(**label**):**

relabel**=**np**.**zeros**([**len**(**label**),**10**],**dtype**=**np**.**int32**)**

**for** i **in** range**(**len**(**label**)):**

relabel**[**i**][**label**[**i**]]=**1

**return** relabel

**Java部分：**

* 手写数字识别主界面的实现

String buttonName**[]** **=** **{**"画笔"**,**"橡皮"**,** "重置"**,**"图片"**,**"确定"**};** //使用数组保存按钮名

JPanel jp1**=new** JPanel**(new** GridLayout**(**5**,** 2**,**20**,**20**));** //用于保存图形按钮的

jp1**.**setPreferredSize**(new** Dimension**(**150**,** 400**));** //面板，使用网格布局

dl**=new** drawlistener**();**

**for** **(**int i **=** 0**;** i **<** buttonName**.**length**;** i**++)** **{** //循环为左侧按钮面板

JButton jbutton **=** **new** JButton**(**buttonName**[**i**]);** //添加按钮

jbutton**.**addActionListener**(**dl**);**

jp1**.**add**(**jbutton**);**

**}**

JPanel jp2**=new** JPanel**();** //创建画布面板

jp2**.**setPreferredSize**(new** Dimension**(**420**,** 420**));**

jp2**.**setBackground**(**Color**.**WHITE**);**

Color**[]** colorArray **=** **{** Color**.**BLUE**,** Color**.**GREEN**,** Color**.**RED**,**

Color**.**BLACK**,**Color**.**ORANGE**,**Color**.**PINK**,**Color**.**CYAN**,**

Color**.**MAGENTA**,**Color**.**DARK\_GRAY**,**Color**.**GRAY**,**

Color**.**LIGHT\_GRAY**,**Color**.**YELLOW**};** //使用数组保存按钮上要显示的颜色信息

JPanel jp3**=new** JPanel**(new** GridLayout**(**1**,**colorArray**.**length**,**15**,**3**));**

**for** **(**int i **=** 0**;** i **<** colorArray**.**length**;** i**++)** **{** //循环为下侧按钮面板添加按钮

JButton button **=** **new** JButton**();**

button**.**setBackground**(**colorArray**[**i**]);**

button**.**setPreferredSize**(new** Dimension**(**30**,** 30**));**

button**.**addActionListener**(**dl**);**

jp3**.**add**(**button**);**

**}**

* **Java的调用python方法**

Process proc **=** **null;**

**try** **{**

proc **=** Runtime**.**getRuntime**().**exec**(**"python "**+**PY\_URL**);** //调用python脚本

proc**.**waitFor**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}** **catch** **(**InterruptedException e**)** **{**

e**.**printStackTrace**();**

**}**

* **Java和python通过结果文件连接**

BufferedReader br**;**

String answer **=** **null;**

**try** **{**

br **=** **new** BufferedReader**(new** FileReader**(new** File**(**DATA\_SWAP**)));**

answer **=** br**.**readLine**();** //从文件读取结果

**}** **catch** **(**FileNotFoundException e**)** **{**

e**.**printStackTrace**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**网页初步构想部分：**

* **网页主界面的实现**

**<**body onselectstart**=**"return false" **>**

**<**div id**=**"mnist-pad"**>**

**<**div class**=**"mnist-pad-body"**>**

**<**canvas**>**

**</**canvas**>**

**</**div**>**

**<**div class**=**"mnist-pad-footer"**>**

**<**div class**=**"mnist-pad-result"**>**

**<**h5**>**识别结果**:</**h5**>**

**<**h5 id**=**"mnist-pad-result"**></**h5**>**

**</**div**>**

**<**div class**=**"mnist-pad-actions"**>**

**<**button type**=**"button" id**=**"mnist-pad-clear"**>**清除**</**button**>** //添加清除按钮

**<**button type**=**"button" id**=**"mnist-pad-save"**>**识别**</**button**>** //添加识别按钮

**</**div**>**

**</**div**>**

**</**div**>**

**<**script src**=**"sxszsb\JS\signature\_pad.js"**></**script**>**

**<**script src**=**"sxszsb\JS\mnist.js"**></**script**>**

**<**script src**=**"sxszsb\JS\app.js"**></**script**>**

**<**body background**=**"Pictures\5.jpg"**>**//设置背景图片

**</**body**>**

## 系统测试

【设计合适的测试数据，根据系统需求实现对系统的各种测试。】

3.3.1功能测试

**①手写板识别功能已实现,且运行正常：**

**②上传图片识别已实现且运行正常：**

**③多个识别初步实现，但算法还需改进：**

3.3.2正确率测试：

①minst测试集10000个样本，正确率在93%左右。

②手写板的正确率如下：

# 总结与展望

* 1. 存在的问题：
* 画板尺寸稍微偏小，手写多个数字识别时，限制数字的数量。
* 显示图片的张数太少，测试数据比较麻烦。
* Swing界面不美观且不怎么实用。
* 在现有图片切割的算法下，一张图片识别多个数字要求比较高。
  1. 改进方向：
* 对手写画板进行放大，方便多个数字的识别。
* 对Swing界面继续进行美化，同时将网页构想实现，增加它的实用性。
* 继续改进图片切割的算法

# 参考文献

【列出设计过程中所参考的书籍、文献、以及网络资源】

1. 画板的实现：<https://www.cnblogs.com/cbs-writing/p/7192698.html>
2. Java调用python的方法：<https://www.linuxidc.com/Linux/2017-05/144371.htm>