大连东软信息学院

智能与电子工程学院

《数字图像处理》

课程项目报告

专　　业：集成电路设计与集成系统

班　　级：集成电路 21003 班

学　　号：21003160314

姓　　名：XXX

授课教师：XXX

2022 – 2023 学年第一学期

三级项目报告注意事项

1. 按照项目要求书写项目报告，条理清晰，数据准确；
2. 项目报告严禁抄袭，如发现抄袭的情况，则抄袭者与被抄袭者均以0分计；
3. 课程结束后报告上交教师，并进行考核与存档。

三级项目报告格式规范

1. 正文：宋体，小四号，首行缩进2字符，1.5倍行距，段前段后各0行；
2. 图表：居中，图名用五号字，中文用宋体，英文用“Times New Roman”，位于图表下方，须全文统一。

银行卡数字识别系统

第1章 银行卡数字识别系统概述

1.1 选题目的，应用价值：

选题目的：

1. 掌握数字图像处理的基本理论和技术：银行卡数字识别是数字图像处理中的一个经典问题，涉及到许多基本的图像处理技术，如图像预处理、特征提取和模式识别等。通过实现这个项目，学生可以更深入地理解和掌握这些基本理论和技术。
2. 培养解决实际问题的能力：银行卡数字识别是一个具有实际应用价值的问题，学生可以通过这个项目将理论知识应用到实际问题中，培养解决实际问题的能力。
3. 探索机器学习和深度学习等先进算法的应用：银行卡数字识别可以看作是一个典型的机器学习问题，涉及到分类、回归等算法的应用。学生可以通过这个项目学习和应用这些先进的算法，并探索如何将其应用到其他领域。
4. 提高编程能力和科研素养：实现银行卡数字识别系统需要具备较高的编程能力和科研素养，学生可以通过这个项目提高这些能力。

应用价值：

1. 提高银行办理业务的效率：银行可以通过银行卡数字识别系统快速、准确地读取客户银行卡的号码，提高办理业务的效率，同时也可以减少因人为因素导致的错误。
2. 防止银行卡被盗刷、诈骗等情况：通过自动识别银行卡上的数字，可以防止不法分子盗刷或诈骗等情况的发生，提高银行卡使用的安全性。
3. 促进图像处理和计算机视觉领域的发展：实现银行卡数字识别系统需要应用到许多先进的图像处理和计算机视觉技术，这些技术的不断发展和完善将推动整个领域的发展。
4. 探索人工智能技术在金融领域的应用：银行卡数字识别是金融领域中的一个重要问题，通过实现这个项目，可以探索人工智能技术在金融领域的其他应用，如贷款审批、风险评估等。
5. 培养人才：实现银行卡数字识别系统需要具备数字图像处理、机器学习、深度学习等方面的知识，通过这个项目可以培养具备这些技能的人才，为未来的科技发展提供支持。

1.2 功能概述

1. 图像预处理：在图像采集后，系统将进行一系列的图像预处理操作，以突出银行卡上的数字信息。这些操作可能包括去噪、图像增强、二值化和分割等。例如，通过使用图像增强技术，可以调整图像的对比度和亮度，以改善数字字符的清晰度；通过二值化和分割技术，可以将数字字符从背景中分离出来，以便于后续的识别处理。
2. 数字识别：该功能采用基于 OpenCV 的模板匹配技术，将经过预处理的图像中的数字字符识别出来，转换为计算机可读的文本格式。识别出图像中的数字字符。在识别过程中，系统可能需要处理多种字体、大小和布局的数字字符，因此需要具备较高的泛化能力和鲁棒性。
3. 信息提取：在数字识别的基础上，系统需要从识别出的文本中提取有用的信息，如卡号、持卡人姓名和有效期等。这些信息将被存储在数据库或特定的格式中，以备后续的验证和使用。
4. 信息验证：为了确保识别结果的准确性和有效性，系统需要进行信息验证。验证可能包括核对数字字符的语法和结构是否正确、卡号是否有效等方面。例如，系统可以通过检查卡号的长度、格式和校验位等来验证卡号的合法性。
5. 数据存储：经过验证的银行卡信息将被存储在系统的数据库中，以便后续的处理和使用。这些信息可以包括持卡人姓名、卡号、有效期和交易记录等。系统需要提供数据备份和恢复功能，以确保数据的安全性和完整性。
6. 用户界面：为了方便用户操作和使用，系统需要提供一个友好的用户界面。用户界面可能包括输入界面、显示界面和操作界面等。输入界面用于输入银行卡信息和查看识别结果；显示界面用于显示识别出的银行卡信息和操作提示；操作界面用于进行系统设置和功能选择等操作。用户界面应该具备易于使用、界面友好和响应迅速等特点。

这些是银行卡数字识别系统的主要功能概述。通过实现这些功能，该系统能够提高银行办理业务的效率，减少人为错误和欺诈行为的发生，同时提高银行卡使用的安全性和便利性。

第2章 原理介绍

2.1 原理 1：图像预处理函数：

- 灰度化：将彩色图像转换为灰度图像，减少计算复杂度。使用 cvtColor()函数实现。
- 二值化：将灰度图像转换为二值图像，突出数字字符。使用 threshold()或 adaptiveThreshold()函数实现。
- 边缘检测：应用 Canny、Sobel 或 Laplacian 等算法检测数字字符的边缘。使用 Canny()、Sobel()或 Laplacian()函数实现。

2.2 原理 2：模板匹配函数：

- 读取模板图像：使用 imread()函数读取预先设定的模板图像。

- 匹配操作：使用 matchTemplate()函数对原始图像和模板图像进行匹配操作，输出一个结果矩阵。
- 寻找最佳匹配位置：使用 minMaxLoc()函数在结果矩阵中找到最佳匹配位置，即与模板最相似的区域。

2.3 原理 3：边缘检测函数：

- 应用边缘检测算法：使用 Canny()、Sobel()或 Laplacian()等函数对二值图像进行边缘检测，得到边缘图像。
- 过滤操作：根据需求，可以使用形态学操作函数对边缘图像进行过滤，去除噪声、填充断裂的边缘等。常用的形态学操作函数包括 erode()、dilate()、morphologyEx()等。

2.4 原理 4：轮廓检测函数：

- 查找轮廓：使用 findContours()函数对二值图像进行轮廓检测，得到一个轮廓列表。
- 轮廓筛选：根据轮廓的大小、形状等特征，对轮廓列表进行筛选，去除不符合要求的轮廓。可以使用 contourArea()、perimeter()等函数计算轮廓的面积和周长，以及 boundingRect()、minAreaRect()等函数获取轮廓的最小外接矩形和最小面积外接矩形。

2.5 原理 5：形态学操作函数：

- 去除噪声：使用 erode()或 dilate()函数对二值图像进行腐蚀或膨胀操作，去除小的噪声点或连接断裂的边缘。
- 改善连接性：使用 morphologyEx()函数对二值图像进行开运算或闭运算，以改善数字字符的连接性。开运算可以去除小的噪声点并断开连接的对象，而闭运算可以填充对象内部的空洞并连接邻近的对象。

2.6 原理 6：WindowsAPI 函数：

Windows API 函数的工作原理是通过在操作系统内核中实现这些函数。操作系统内核是操作系统的核心部分，它负责管理系统资源、提供系统服务等。Windows API 函数是在内核中实现的，因此它们可以直接使用系统资源和服务。这使得 Windows API 函数非常强大和灵活，可以用于许多不同的应用程序。

当应用程序调用 Windows API 函数时，它们将传递参数给这些函数，然后函数将执行所需的操作，并返回结果。这些函数可以执行许多不同的操作，例如创建窗口、打开文件、读取和写入文件等。

Windows API 函数的实现原理是通过在操作系统内核中实现这些函数。操作系统内核是操作系统的核心部分，它负责管理系统资源、提供系统服务等。Windows API 函数是在内核中实现的，因此它们可以直接使用系统资源

和服务。这使得 Windows API 函数非常强大和灵活，可以用于许多不同的应用程序。

Windows API 是一套用来控制 Windows 的各个部件的外观和行为的预先定义的 Windows 函数。 用户的每个动作都会引发一个或几个函数的运行以告诉 Windows 发生了什么。 这在某种程度上很像 Windows 的天然代码。 而其他的语言只是提供一种能自动而且更容易的访问 API 的方法。 更易理解来说：Windows 系统除了协调应用程序的执行、内存的分配、系统资源的管理外，同时他也是一个很大的服务中心。 调用这个服务中心的各种服务(每一种服务就是一个函数)可以帮助应用程序达到开启视窗、描绘图形和使用周边设备等目的，由于这些函数服务的对象是应用程序，所以称之为 Application Programming Interface，简称 API 函数。

这些是银行卡数字识别系统中各个函数的详细工作原理。通过组合这些函数和算法，可以构建一个高效、准确的银行卡数字识别系统。

第3章 银行卡数字识别系统设计及实现

3.1 系统开发环境描述

银行卡数字识别系统的开发环境主要包括以下几个部分：

1. 硬件环境：计算机:笔记本电脑（Windows 10 专业版），用于运行图像处理算法。
2. 软件环境：

   开发软件：Visual Studio 2022 17.8.3 、Visual Studio Code 1.85.0 OpenCV 4.5.4。这些软件提供了开发银行卡数字识别系统所需的编程语言、集成开发环境 和 GUI 开发工具等。

   模型库：OpenCV 4.5.4、以支持机器学习和图像处理算法的实现。
3. 网络环境：为了获取银行卡的图像数据，通常需要连接到互联网。在开发过程中，可以通过网络获取大量的银行卡图像样本，用于训练和测试机器学习模型。

4. 开发流程：通常采用迭代开发的方式，逐步完善和优化系统功能。在开发过程中，需要不断收集和整理银行卡图像数据，对模型进行训练和测试，并进行必要的调整和优化。同时，还需要与业务人员进行沟通和协作，确保系统的功能和性能符合业务需求。

3.2 系统的算法流程

银行卡数字识别系统是一个涉及图像处理、特征提取和模式识别等多个环节的复杂过程。以下是该系统的整体流程：

1. 图像采集：首先，需要从银行或其他来源获取含有银行卡的图像数据。这些图像可能来自扫描、拍照或视频捕捉等不同方式。
2. 图像预处理：在图像采集后，系统将对图像进行一系列的预处理操作，以突出银行卡上的数字信息。这些操作可能包括去噪、图像增强、二值化和分割等。例如，通过使用图像增强技术，可以调整图像的对比度和亮度，以改善数字字符的清晰度；通过二值化和分割技术，可以将数字字符从背景中分离出来，以便于后续的识别处理。
3. 数字识别：预处理后的图像将通过特定的算法进行数字识别。这通常涉及到模板匹配、神经网络或其他机器学习算法的应用。这些算法将根据银行卡字符的形状、大小、颜色等特征进行匹配和分类，从而识别出图像中的数字字符。在识别过程中，系统可能需要处理多种字体、大小和布局的数字字符，因此需要具备较高的泛化能力和鲁棒性。
4. 信息提取：在数字识别的基础上，系统需要从识别出的文本中提取有用的信息，如卡号、持卡人姓名和有效期等。这些信息将被存储在数据库或特定的格式中，以备后续的验证和使用。
5. 信息验证：为了确保识别结果的准确性和有效性，系统需要进行信息验证。验证可能包括核对数字字符的语法和结构是否正确、卡号是否有效等方面。例如，系统可以通过检查卡号的长度、格式和校验位等来验证卡号的合法性。
6. 数据存储：经过验证的银行卡信息将被存储在系统的数据库中，以便后续的处理和使用。这些信息可以包括持卡人姓名、卡号、有效期和交易记录等。系统需要提供数据备份和恢复功能，以确保数据的安全性和完整性。
7. 用户界面：为了方便用户操作和使用，系统需要提供一个友好的用户界面。用户界面可能包括输入界面、显示界面和操作界面等。输入界面用于输入银行卡信息和查看识别结果；显示界面用于显示识别出的银行卡信息和操作提示；操作界面用于进行系统设置和功能选择等操作。用户界面应该具备易于使用、界面友好和响应迅速等特点。

以上是银行卡数字识别系统的基本流程。在实际应用中，可能还需要根据具体需求和场景进行适当的优化和调整。

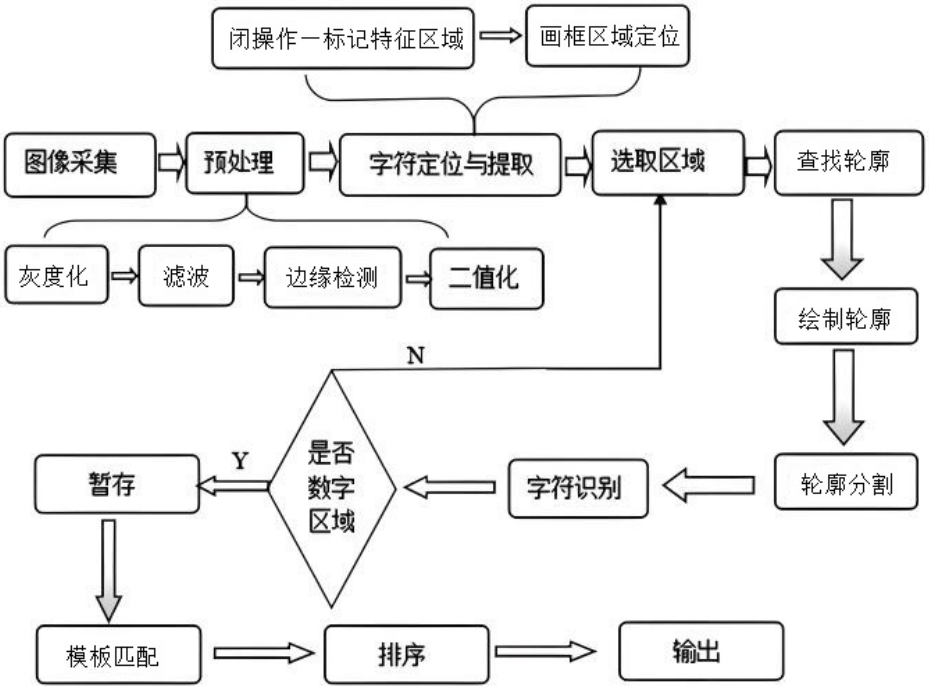

图 3.2.1 系统工作流程图

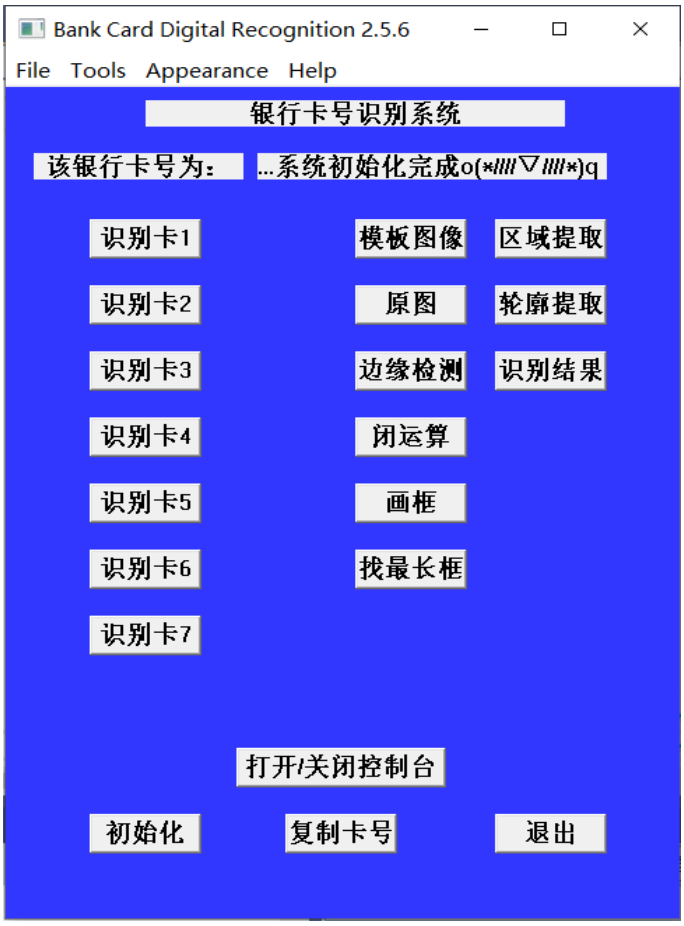

图 3.2.2 系统工作运行界面

3.3 核心代码介绍

OpenCV（Open Source Computer Vision Library）是一个开源的计算机视觉和机器学习软件库，其中包含了众多用于图像处理和计算机视觉任务的函数。在实现银行卡数字识别系统时，我可以利用 OpenCV 中的一些核心函数，如模板匹配、边缘检测、轮廓检测等。

函数功能与参数设置

1. 图像预处理：在图像采集后，系统将进行一系列的图像预处理操作，以突出银行卡上的数字信息。这些操作可能包括去噪、图像增强、二值化和分割等。例如，通过使用图像增强技术，可以调整图像的对比度和亮度，以改善数字字符的清晰度；通过二值化和分割技术，可以将数字字符从背景中分离出来，以便于后续的识别处理。
2. 模板匹配：该功能采用基于 OpenCV 的模板匹配技术，将经过预处理的图像中的数字字符识别出来，转换为计算机可读的文本格式。在识别过程中，系统可能需要处理多种字体、大小和布局的数字字符，因此需要具备较高的泛化能力和鲁棒性。
3. 信息提取：在数字识别的基础上，系统需要从识别出的文本中提取有用的信息，如卡号、持卡人姓名和有效期等。这些信息将被存储在数据库或特定的格式中，以备后续的验证和使用。

```
//  Image preprocessing function    图像预处理函数
void ImagePreprocessing(Mat& src)
{
    Mat gray;
    cvtColor(src, gray,
COLOR_BGR2GRAY);
  //图像转灰度图
    GaussianBlur(gray, gray, Size(3, 3), 0,
0);                                                    //高斯滤波
    Mat edge;
    Canny(gray, edge, 200, 255,
3);                                                                  //边缘
检测
    //imshow("边缘检测",
edge);
//调试用
    imwrite("边缘检测.jpg", edge);
    Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(34,
3));                                  //闭运算  圆形结构元素
```

```
    morphologyEx(edge, edge, MORPH_CLOSE,
kernel);                                                    //闭运算
    //imshow("形态操作(闭运算)",
edge);                                                          //调试
用
    imwrite("闭运算.jpg", edge);
    vector<vector<Point>>
contours;
//轮廓
    vector<Vec4i>
hierarchy;
        //层次结构
    findContours(edge, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE, Point(0, 0));       //查找轮廓 findContours(原图像，
轮廓，层次结构，轮廓检索模式，轮廓近似方法，偏移量)
    Mat dst = Mat::zeros(src.size(),
CV_8UC3);                                                   //绘制轮廓
    vector<Rect> rects;//矩形块
    rects.reserve(contours.size());
    //遍历轮廓
    for (int i = 0; i < contours.size();
i++)                                                        //遍历轮廓
    {
        Rect rect =
boundingRect(contours[i]);
      //获取轮廓的矩形块
        rects.push_back(rect);
                        //将矩形块添加到矩形块数组中
        if (rect.width > 200 && rect.height >
200)                                                  //过滤掉过小的矩形块
        {
            drawContours(dst, contours, i, Scalar(255, 255, 255), 1, 1,
hierarchy, 1, Point(0));    //绘制轮廓 drawContours (绘制轮廓的图像，轮廓，
轮廓索引，颜色，厚度，线型，层次结构，最大层次，偏移量)
            rectangle(dst, rect, Scalar(0, 255, 0),
0);                                          //绘制矩形块 rectangle(绘制
矩形块的图像，矩形块，颜色，厚度)
        }
    }
    //按矩形块的 x 坐标排序
    sort(rects.begin(), rects.end(), [](Rect a, Rect b) {return a.x <
b.x; });                    //sort(开始位置，结束位置，比较函数)
    //绘制矩形块
```

```
    for (int i = 0; i < rects.size();
i++)                                                    //遍历轮廓
    {
        Rect rect =
rects[i];
      //获取矩形块
        if (rect.width > 20 &&
rect.height >20)                                        //过滤
掉过小的矩形块
        {
            rectangle(src, rect, Scalar(0, 255, 0),
0);                                             //绘制矩形块
        }
    }
    int max =
0;
          //最长的矩形块的宽度
    int index =
0;
         //最长的矩形块的索引
    for (int i = 0; i < rects.size();
i++)                                                    //找出最长的
矩形块
    {
        Rect rect = rects[i];
        if (rect.width >
max)
//如果矩形块的宽度大于 max，则将 max 赋值为矩形块的宽度，index 赋值为 i
        {
            max =
rect.width;
        //max 为最长的矩形块的宽度
            index = i;
        }
    }
    //imshow("画框", src);//调试用
    imwrite("画框.jpg", src);
    for (int i = 0; i < rects.size();
i++)                                                    //只保留最长
的矩形块，其他矩形块恢复原样
    {
        Rect rect =
rects[i];
      //获取矩形块
```

```
        if (i !=
index)
        //如果不是最长的矩形块
        {
            rectangle(src, rect, Scalar(255, 0, 0),
0);                                          //绘制矩形块 rectangle(绘制
矩形块的图像，矩形块，颜色，厚度)  用蓝色重新绘制
        }
    }
    Rect rect =
rects[index];
          //获取最长的矩形块
    rectangle(src, rect, Scalar(0, 255, 0),
1);                                                  //绘制矩形块
    //imshow("最长的矩形块", src);//调试用
    imwrite("最长的矩形块.jpg", src);
    //获取最长的矩形块对应的图像区域
    Mat roi =
src(rect);
           //获取矩形块对应的图像区域
    Mat gray_roi,
binary_roi;
        //灰度图像，二值图像
    cvtColor(roi, gray_roi,
COLOR_BGR2GRAY);                                                    //
转换为灰度图像
    threshold(gray_roi, binary_roi, 0, 255, THRESH_BINARY_INV |
THRESH_OTSU);                       //二值化
    bitwise_not(binary_roi,
binary_roi);                                                        //
颜色反转
    //保存银行卡号图像
    imwrite("card_number.jpg", binary_roi);
}
```

这段代码是用于图像预处理的函数，主要用于识别和提取银行卡号。以下是每一步的详细解释：

1. cvtColor(src, gray, COLOR_BGR2GRAY);：将输入图像从 BGR 颜色空间转换为灰度图像。
2. GaussianBlur(gray, gray, Size(3, 3), 0, 0);：对灰度图像进行高斯滤波，以减少图像噪声。
3. Canny(gray, edge, 200, 255, 3);：使用 Canny 算法进行边缘检测。

4. morphologyEx(edge, edge, MORPH_CLOSE, kernel);：对边缘检测的结果进行形态学闭运算，以填充物体内部的小孔。
5. findContours(edge, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE, Point(0, 0));：找出图像中的轮廓。
6. boundingRect(contours[i]);：计算每个轮廓的边界矩形。
7. rectangle(dst, rect, Scalar(0, 255, 0), 0);：在图像上绘制边界矩形。
8. sort(rects.begin(), rects.end(), [](Rect a, Rect b) {return a.x < b.x; });：按照 x 坐标对所有的边界矩形进行排序。
9. rectangle(src, rect, Scalar(0, 255, 0), 0);：在原图像上绘制边界矩形。
10. 在所有的边界矩形中找出宽度最大的一个。
11. rectangle(src, rect, Scalar(0, 255, 0), 1);：在原图像上绘制宽度最大的边界矩形。
12. Mat roi = src(rect);：提取出宽度最大的边界矩形对应的图像区域。
13. cvtColor(roi, gray_roi, COLOR_BGR2GRAY);：将 ROI 转换为灰度图像。
14. threshold(gray_roi, binary_roi, 0, 255, THRESH_BINARY_INV | THRESH_OTSU);：对灰度图像进行二值化。
15. bitwise_not(binary_roi, binary_roi);：对二值图像进行颜色反转。
16. imwrite("card_number.jpg", binary_roi);：保存处理后的银行卡号图像。

这个函数的主要目的是从输入图像中提取出银行卡号的图像，以便后续进行数字识别。

参数设置：在实现这些功能时，需要设置一些参数，如滤波器类型、边缘检测阈值、形态学操作参数等。这些参数的设置将直接影响图像处理的效果和数字识别的准确性。因此，在开发过程中需要进行充分的实验和调整，以选择最优的参数组合。

3.4 系统的实现

```
//Digital image processing three items: Bank Card Digital Recognition
//Software:                Visual Studio 2022 17.8.3 Visual Studio Code
1.85.0  & (C++17) & OpenCV 4.5.4
//Creation time:           2023/10/19
//Last modification time:   2023/12/3 10:26:56
```

```
//Version:                  2.5.6
//Author:                   WangMingJie
//Student ID:               21003160314

#include    <cstdlib>
#include    <iostream>
#include    <windows.h>
#include    <opencv2/opencv.hpp>
#include    <opencv2/core/core.hpp>
#include    <opencv2/highgui/highgui.hpp>
#include    <opencv2/imgproc/imgproc.hpp>

using namespace
cv  ;
        //opencv namespace
using namespace
std ;
        //standard namespace

string TEMPLATE_IMAGE_PATH    =
"template.jpg"      ;                                       //temp
late image path
string CARD_IMAGE_PATH        =
"card1.jpg"         ;                                       //card
image path
string CARD_NUMBER_IMAGE_PATH =
"card_number.jpg"   ;                                       //card
number image path
string cardNumber             =
"1234567890"        ;                                       //Glob
al variable cardNumber

int
sel;
                    //Console button switch Select      0: Off 1: On

//Windows API 参数声明
HWND
hwndEdit;
                    //声明 hwndEdit 为全局变量  HWND:窗口句
柄                 hwndEdit:编辑框句柄
HINSTANCE
hInst;
```

```
              //声明 hInst 为全局变量     HINSTANCE:应用程序实例句
柄    hInst:应用程序实例句柄
HBRUSH hbrBkgnd =
NULL;
       //用于保存画刷的句柄        HBRUSH:画刷句柄                hbrBkgnd:
画刷句柄

//  Image loading success/failure function  图像载入成功/失败判断函数
Mat loadImageAndPreprocess(const string& imagePath, const string&
imageDescription)
{
    cout << "...载入" << imageDescription << endl;
    Mat img = imread(imagePath);
    if
(img.empty())
                  //Determine whether the image was successfully loaded
    {
        throw runtime_error("无法加载图像: " +
imagePath);                                          //抛出异常
    }
    else
    {
        cout << "图像载入成功" << endl;
    }
    //imshow(imageDescription,
img);                                                          //调试
用
    return
img;
             //Return image
}

//  Template contour extraction function    模板轮廓提取函数
void Template_Preprocessing(Mat& src)
{
    Mat gray, edge, dst;
    cvtColor(src, gray,
COLOR_BGR2GRAY);
  //转换为灰度图像   cvtColor(原图像，目标图像，转换类型)
    GaussianBlur(gray, gray, Size(3, 3), 0,
0);                                                    //高斯滤
波  GaussianBlur(原图像，目标图像，高斯核大小，高斯核标准差 X，高斯核标准差 Y)
```

```
    Canny(gray, edge, 50, 150,
3);                                                                    //边缘
检测  Canny(原图像，目标图像，低阈值，高阈值，核大小)
    vector<vector<Point>>
contours;
//轮廓 vector<Mat> OpenCV 中的数组类
    vector<Vec4i>
hierarchy;
        //层次结构
    findContours(edge, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE, Point(0, 0));       //查找轮廓 findContours(原图像，
轮廓，层次结构，轮廓检索模式，轮廓近似方法，偏移量)
    dst = Mat::zeros(src.size(),
CV_8UC3);                                                              //绘制轮
廓
    vector<Rect>
rects;
        //矩形块
    for (int i = 0; i < contours.size();
i++)                                                    //遍历轮廓
    {
        Rect rect =
boundingRect(contours[i]);
      //获取轮廓的矩形块
        if (rect.width > 10 && rect.height >
10)                                                  //过滤掉过小的矩形块
        {
            rects.push_back(rect);
            drawContours(dst, contours, i, Scalar(255, 255, 255), 1, 1,
hierarchy, 1, Point(0));    //绘制轮廓 drawContours (绘制轮廓的图像，轮廓，
轮廓索引，颜色，厚度，线型，层次结构，最大层次，偏移量)
            rectangle(dst, rect, Scalar(0, 255, 0),
0);                                                 //绘制矩形
块     rectangle(绘制矩形块的图像，矩形块，颜色，厚度)
        }
    }
    //imshow("轮廓图像", dst);//调试用
    sort(rects.begin(), rects.end(), [](Rect a, Rect b) {return a.x <
b.x; });                        //按矩形块的 x 坐标排序
    for (int i = 0; i < rects.size(); i++)
    {
        Mat roi =
src(rects[i]);
        //获取矩形块对应的图像区域
```

```
        Mat gray_roi, binary_roi;
        cvtColor(roi, gray_roi,
COLOR_BGR2GRAY);                                                    //转换
为灰度图像 cvtColor(原图像，目标图像，转换类型)
        threshold(gray_roi, binary_roi, 0, 255, THRESH_BINARY_INV |
THRESH_OTSU);                  //二值化 threshold(原图像，目标图像，阈值，
最大值，阈值类型)
        vector<vector<Point>> contours_roi;
        vector<Vec4i> hierarchy_roi;
        findContours(binary_roi, contours_roi, hierarchy_roi,
RETR_EXTERNAL, CHAIN_APPROX_SIMPLE, Point(0, 0)); //查找轮廓
        for (int j = 0; j < contours_roi.size(); j++)
        {
            Rect rect_roi =
boundingRect(contours_roi[j]);                                      /
/获取轮廓的矩形块
            if (rect_roi.width > 100 && rect_roi.height >
200)                                    //过滤掉过小的矩形块
            {
                rectangle(roi, rect_roi, Scalar(255, 255, 255),
0);                                 //绘制矩形块  rectangle(绘制矩形块的图
像，矩形块，颜色，厚度)
            }
        }
        //imshow(to_string(i),
roi);                                                               //调试
用
        imwrite("template"+ to_string(i) + ".jpg", roi);
    }
}

//  Image preprocessing function    图像预处理函数
void ImagePreprocessing(Mat& src)
{
    Mat gray;
    cvtColor(src, gray,
COLOR_BGR2GRAY);
  //图像转灰度图
    GaussianBlur(gray, gray, Size(3, 3), 0,
0);                                                 //高斯滤波
    Mat edge;
    Canny(gray, edge, 200, 255,
3);                                                             //边缘
检测
```

```
    //imshow("边缘检测",
edge);
//调试用
    imwrite("边缘检测.jpg", edge);
    Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(34,
3));                                //闭运算  圆形结构元素
    morphologyEx(edge, edge, MORPH_CLOSE,
kernel);                                                    //闭运算
    //imshow("形态操作(闭运算)",
edge);                                                          //调试
用
    imwrite("闭运算.jpg", edge);
    vector<vector<Point>>
contours;
//轮廓
    vector<Vec4i>
hierarchy;
        //层次结构
    findContours(edge, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE, Point(0, 0));       //查找轮廓 findContours(原图像，
轮廓，层次结构，轮廓检索模式，轮廓近似方法，偏移量)
    Mat dst = Mat::zeros(src.size(),
CV_8UC3);                                                   //绘制轮廓
    vector<Rect> rects;//矩形块
    rects.reserve(contours.size());
    //遍历轮廓
    for (int i = 0; i < contours.size();
i++)                                                        //遍历轮廓
    {
        Rect rect =
boundingRect(contours[i]);
      //获取轮廓的矩形块
        rects.push_back(rect);
                        //将矩形块添加到矩形块数组中
        if (rect.width > 200 && rect.height >
200)                                                  //过滤掉过小的矩形块
        {
            drawContours(dst, contours, i, Scalar(255, 255, 255), 1, 1,
hierarchy, 1, Point(0));    //绘制轮廓 drawContours (绘制轮廓的图像，轮廓，
轮廓索引，颜色，厚度，线型，层次结构，最大层次，偏移量)
            rectangle(dst, rect, Scalar(0, 255, 0),
0);                                          //绘制矩形块 rectangle(绘制
矩形块的图像，矩形块，颜色，厚度)
        }
```

```
    }
    //按矩形块的 x 坐标排序
    sort(rects.begin(), rects.end(), [](Rect a, Rect b) {return a.x <
b.x; });                        //sort(开始位置，结束位置，比较函数)
    //绘制矩形块
    for (int i = 0; i < rects.size();
i++)                                                    //遍历轮廓
    {
        Rect rect =
rects[i];
      //获取矩形块
        if (rect.width > 20 &&
rect.height >20)                                                //过滤
掉过小的矩形块
        {
            rectangle(src, rect, Scalar(0, 255, 0),
0);                                             //绘制矩形块
        }
    }
    int max =
0;
          //最长的矩形块的宽度
    int index =
0;
         //最长的矩形块的索引
    for (int i = 0; i < rects.size();
i++)                                                    //找出最长的
矩形块
    {
        Rect rect = rects[i];
        if (rect.width >
max)
//如果矩形块的宽度大于 max，则将 max 赋值为矩形块的宽度，index 赋值为 i
        {
            max =
rect.width;
        //max 为最长的矩形块的宽度
            index = i;
        }
    }
    //imshow("画框", src);//调试用
    imwrite("画框.jpg", src);
```

```
    for (int i = 0; i < rects.size();
i++)                                                    //只保留最长
的矩形块，其他矩形块恢复原样
    {
        Rect rect =
rects[i];
      //获取矩形块
        if (i !=
index)
        //如果不是最长的矩形块
        {
            rectangle(src, rect, Scalar(255, 0, 0),
0);                                             //绘制矩形块 rectangle(绘制
矩形块的图像，矩形块，颜色，厚度)  用蓝色重新绘制
        }
    }
    Rect rect =
rects[index];
          //获取最长的矩形块
    rectangle(src, rect, Scalar(0, 255, 0),
1);                                                 //绘制矩形块
    //imshow("最长的矩形块", src);//调试用
    imwrite("最长的矩形块.jpg", src);
    //获取最长的矩形块对应的图像区域
    Mat roi =
src(rect);
            //获取矩形块对应的图像区域
    Mat gray_roi,
binary_roi;
        //灰度图像，二值图像
    cvtColor(roi, gray_roi,
COLOR_BGR2GRAY);                                                    //
转换为灰度图像
    threshold(gray_roi, binary_roi, 0, 255, THRESH_BINARY_INV |
THRESH_OTSU);                       //二值化
    bitwise_not(binary_roi,
binary_roi);                                                        //
颜色反转
    //保存银行卡号图像
    imwrite("card_number.jpg", binary_roi);
}

//  Card number contour extraction function     卡号轮廓提取函数
void CardNumberContourExtraction(Mat& src)
```

```
{
    Mat gray, edge,
dst;
     //灰度图像，边缘图像，轮廓图像
    GaussianBlur(src, gray,  Size(3, 3), 0,
0);                                                    //高斯滤波
    Canny(gray, edge, 50, 150,
3);                                                              //边缘
检测
    vector<vector<Point>>
contours;
//轮廓
    vector<Vec4i>
hierarchy;
        //层次结构
    findContours(edge, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE, Point(0, 0));       //查找轮廓
    dst = Mat::zeros(src.size(),
CV_8UC3);                                                        //绘制轮
廓
    vector<Rect>
rects;
        //矩形块
    for (int i = 0; i < contours.size();
i++)                                                    //遍历轮廓
    {
        Rect rect =
boundingRect(contours[i]);
      //获取轮廓的矩形块
        if (rect.width > 10 && rect.height >
10)                                                   //过滤掉过小的矩形块
        {
            rects.push_back(rect);
                        //将矩形块添加到矩形块数组中
            drawContours(dst, contours, i, Scalar(255, 255, 255), 1, 1,
hierarchy, 1, Point(0));    //绘制轮廓 drawContours (绘制轮廓的图像，轮廓，
轮廓索引，颜色，厚度，线型，层次结构，最大层次，偏移量)
            rectangle(dst, rect, Scalar(0, 255, 0),
0);                                          //绘制矩形块
        }
    }
    //imshow("号码轮廓图像",
dst);                                                            //调
试用
```

```
    imwrite("号码轮廓图像.jpg", dst);
    sort(rects.begin(), rects.end(), [](Rect a, Rect b) {return a.x <
b.x; });                    //按矩形块的 x 坐标排序
    for (int i = 0; i < rects.size(); i++)
    {
        Mat roi =
src(rects[i]);
        //获取矩形块对应的图像区域
        Mat gray_roi, binary_roi;
        cvtColor(roi, gray_roi,
COLOR_BGR2GRAY);                                                    //转换
为灰度图像
        threshold(gray_roi, binary_roi, 0, 255, THRESH_BINARY_INV |
THRESH_OTSU);                  //二值化
        vector<vector<Point>> contours_roi;
        vector<Vec4i> hierarchy_roi;
        findContours(binary_roi, contours_roi, hierarchy_roi,
RETR_EXTERNAL, CHAIN_APPROX_SIMPLE, Point(0, 0)); //查找轮廓
        for (int j = 0; j < contours_roi.size(); j++)
        {
            Rect rect_roi =
boundingRect(contours_roi[j]);                                      /
/获取轮廓的矩形块
            if (rect_roi.width > 100 && rect_roi.height >
200)                                    //过滤掉过小的矩形块
            {
                rectangle(roi, rect_roi, Scalar(255, 255, 255),
0);                                 //绘制矩形块
            }
        }
        //imshow(to_string(i+1),
roi);                                                               //调试用
        imwrite("number" + to_string(i+1) + ".jpg", roi);
    }
}

// 模板匹配函数
pair<double, int> templateMatching(Mat& img, Mat&
templ)                                          //pair<最大值，最大值位置>
{
    Mat result;
    // 调整模板的大小以匹配待识别图像的大小
```

```
    resize(templ, templ,
img.size());
 //resize(原图像，目标图像，目标图像大小)
    // 使用基于 OpenCV 的 TM_CCORR_NORMED 方法进行匹配
    matchTemplate(img, templ, result,
TM_CCORR_NORMED);                                          //matchTempl
ate(原图像，模板图像，结果图像，匹配方法)
    double minVal,
maxVal;
      //最小值，最大值
    Point minLoc,
maxLoc;
       //最小值位置，最大值位置
    minMaxLoc(result, &minVal, &maxVal, &minLoc,
&maxLoc);                                          //minMaxLoc(输入数组，最
小值，最大值，最小值位置，最大值位置)
    return make_pair(maxVal,
maxLoc.x);                                                    //m
ake_pair(最大值，最大值位置)
}

// 模板匹配函数
string TemplateMatching(int i)
{
    string cardNumber;
    vector<Mat>
templates(10);
          //vector<Mat> OpenCV 中的数组类
    for (int i = 0; i < 10; i++)
    {
        templates[i] = imread("template" + to_string(i) +
".jpg");                                  //读取模板图像
    }
    vector<Mat>
numbers(19);
          //vector<Mat> OpenCV 中的数组类
    for (int i = 1; i <= 19;
i++)                                                          //遍
历待识别图像小块
    {
        numbers[i - 1] = imread("number" + to_string(i) +
".jpg");                                  //读取待识别图像
    }
    // 对每一个待识别图像进行模板匹配
```

```
    for (int i = 0; i < numbers.size(); i++)
    {
        double maxMatch =
-1;                                                                 /
/最佳匹配精度
        int maxTemplateIndex =
-1;                                                                 //最佳
匹配模板
        // 对每一个模板进行匹配
        for (int j = 0; j < templates.size(); j++)
        {
            pair<double, int> match = templateMatching(numbers[i],
templates[j]);                  //模板匹配
            if (match.first >
maxMatch)                                                           //如
果匹配精度大于最佳匹配精度
            {
                maxMatch =
match.first;                                                        /
/最佳匹配精度
                maxTemplateIndex =
j;                                                                  //最佳匹配
模板
            }
        }
        cout << "Number:" << i + 1 << "\t 最佳匹配模板: " << maxTemplateIndex
<< " \t 匹配精度: " << maxMatch << endl;
        cardNumber +=
to_string(maxTemplateIndex);
    //将最佳匹配模板添加到卡号中
        //cout << maxTemplateIndex;  //调试用 打印最佳匹配模板
    }
    cout << "\n 该银行卡卡号为: " << cardNumber << endl;
    return cardNumber;
    waitKey(0);
}

// 在图像上放置字符（卡号）函数
void putCardNumberOnImage(cv::Mat &img, std::string cardNumber)
{
    Point textPosition(img.cols / 8, img.rows /
2);                                             //设置文本的位置
```

```
    Scalar textColor(255, 0,
0);                                                          //设
置文本的颜色
    // 在图像上放置文
本
    putText(img, cardNumber, textPosition, cv::FONT_HERSHEY_SIMPLEX, 1.1,
textColor,3);            //putText(绘制文本的图像，文本，文本位置，字体，字
体大小，颜色)
}

//银行卡数字识别函数
string Digital_Recognition(string CARD_IMAGE_PATH)
{
    Mat src = loadImageAndPreprocess(TEMPLATE_IMAGE_PATH, "模板图像
");                              //读取模板
    Template_Preprocessing(src);
                          //模板轮廓提取
    Mat original = loadImageAndPreprocess(CARD_IMAGE_PATH, "银行卡图像
");                           //读取银行卡图像
    imwrite("原图.jpg", original);
    Mat CARD =
original.clone();
          //克隆银行卡图像
    ImagePreprocessing(CARD);
                          //图像预处理
    Mat card_number = loadImageAndPreprocess(CARD_NUMBER_IMAGE_PATH, "银
行卡号图像");              //读取银行卡号图像
    CardNumberContourExtraction(card_number);
                          //卡号轮廓提取
    cardNumber =
TemplateMatching(19);
         //模板匹配
    putCardNumberOnImage(original,
cardNumber);                                                 //在银行卡
图像上放置卡号
    imwrite("识别结果.jpg", original);
    return
cardNumber;

}

//文件清理函数
void DestroyAllWindows()
{
```

```
    for (int i = 0; i < 10;
i++)                                                                    //
清理模板图像
    {
        remove(("template" + to_string(i) + ".jpg").c_str());
    }
    for (int i = 1; i <= 19;
i++)                                                                   //清
理号码轮廓图像
    {
        remove(("number" + to_string(i) + ".jpg").c_str());
    }
    //清理文件
    remove("card_number.jpg");
    remove("边缘检测.jpg");
    remove("闭运算.jpg");
    remove("画框.jpg");
    remove("最长的矩形块.jpg");
    remove("号码轮廓图像.jpg");
    remove("识别结果.jpg");
    remove("原图.jpg");
}

//卡号复制函数
void copyToClipboard(const std::string &s)
{
    OpenClipboard(0);
                          //打开剪贴板
    EmptyClipboard();
                          //清空剪贴板
    HGLOBAL
hg=GlobalAlloc(GMEM_MOVEABLE,s.size()+1);
                //分配内存
    if
(!hg)
                    //分配内存失败
    {
        CloseClipboard();
                          //关闭剪贴板
        return;
    }
    memcpy(GlobalLock(hg),s.c_str(),s.size()+1);
                           //拷贝数据
```

```
    GlobalUnlock(hg);
                            //解锁
    SetClipboardData(CF_TEXT,hg);
                            //设置剪贴板数据
    CloseClipboard();
                            //关闭剪贴板
    GlobalFree(hg);
                            //释放内存
}

//使用 Windows API 创建窗口

//窗口初始化函数
void InitializeWindow(HWND hwnd, HINSTANCE hInst)
{
    DestroyAllWindows();
                            //文件清理
    // 创建所有的控件
    HWND hwndButton1    = CreateWindow("BUTTON" , "原图
"             ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 150, 80, 30,
hwnd, (HMENU)1, hInst, NULL); //按钮控件
    HWND hwndButton2    = CreateWindow("BUTTON" , "边缘检测
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 200, 80, 30, hwnd,
(HMENU)2, hInst, NULL);
    HWND hwndButton3    = CreateWindow("BUTTON" , "闭运算
"           ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 250, 80, 30, hwnd,
(HMENU)3, hInst, NULL);
    HWND hwndButton4    = CreateWindow("BUTTON" , "画框
"             ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 300, 80, 30,
hwnd, (HMENU)4, hInst, NULL);
    HWND hwndButton5    = CreateWindow("BUTTON" , "找最长框
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 350, 80, 30, hwnd,
(HMENU)5, hInst, NULL);
    HWND hwndButton6    = CreateWindow("BUTTON" , "区域提取
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 350, 100, 80, 30, hwnd,
(HMENU)6, hInst, NULL);
    HWND hwndButton7    = CreateWindow("BUTTON" , "轮廓提取
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 350, 150, 80, 30, hwnd,
(HMENU)7, hInst, NULL);
    HWND hwndButton8    = CreateWindow("BUTTON" , "识别结果
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 350, 200, 80, 30, hwnd,
(HMENU)8, hInst, NULL);
```

```
    HWND hwndButton9    = CreateWindow("BUTTON" , "模板图像
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 250, 100, 80, 30, hwnd,
(HMENU)9, hInst, NULL);
    HWND hwndButton1000 = CreateWindow("BUTTON" , "识别卡
1"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 100, 80, 30, hwnd,
(HMENU)1000, hInst, NULL);
    HWND hwndButton1001 = CreateWindow("BUTTON" , "识别卡
2"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 150, 80, 30, hwnd,
(HMENU)1001, hInst, NULL);
    HWND hwndButton1002 = CreateWindow("BUTTON" , "识别卡
3"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 200, 80, 30, hwnd,
(HMENU)1002, hInst, NULL);
    HWND hwndButton1003 = CreateWindow("BUTTON" , "识别卡
4"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 250, 80, 30, hwnd,
(HMENU)1003, hInst, NULL);
    HWND hwndButton1004 = CreateWindow("BUTTON" , "识别卡
5"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 300, 80, 30, hwnd,
(HMENU)1004, hInst, NULL);
    HWND hwndButton1005 = CreateWindow("BUTTON" , "识别卡
6"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 350, 80, 30, hwnd,
(HMENU)1005, hInst, NULL);
    HWND hwndButton1006 = CreateWindow("BUTTON" , "识别卡
7"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 400, 80, 30, hwnd,
(HMENU)1006, hInst, NULL);
    HWND hwndButton1007 = CreateWindow("BUTTON" , "初始化
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 60, 550, 80, 30, hwnd,
(HMENU)1007, hInst, NULL);
    HWND hwndButton1008 = CreateWindow("BUTTON" , "退出
"            ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 350, 550, 80, 30,
hwnd, (HMENU)1008, hInst, NULL);
    HWND hwndButton2000 = CreateWindow("BUTTON" , "复制卡号
"          ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 200, 550, 80, 30, hwnd,
(HMENU)2000, hInst, NULL);
    HWND hwndButton2001 = CreateWindow("BUTTON" , "打开/关闭控制台
"  ,   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 165, 500, 150, 30, hwnd,
(HMENU)2001, hInst, NULL);
    HWND hwndStatic0    = CreateWindow("STATIC" , "银行卡号识别系统
" ,   WS_CHILD | WS_VISIBLE | SS_CENTER, 100, 10, 300, 20, hwnd, NULL, hInst,
NULL);
    HWND hwndStatic1    = CreateWindow("STATIC" , "该银行卡号为:
"   ,   WS_CHILD | WS_VISIBLE | SS_CENTER, 20, 50, 150, 20, hwnd, NULL, hInst,
NULL);             //CreateWindow(控件类名，控件文本，控件样式，控件位置，控
件大小，父窗口句柄，菜单句柄，应用程序实例句柄，指向创建窗口的参数的指针)
```

```
    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD | WS_VISIBLE | ES_LEFT
| ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL | ES_READONLY, 180, 50, 250,
20, hwnd, NULL, hInst, NULL);
    cardNumber = "...系统初始化完成 o(*////▽
////*)q";                                                    //初始化卡号
    SetWindowText(hwndEdit1,
cardNumber.c_str());                                                    //
设置编辑框的文本
}

//窗口过程函数
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT msg, WPARAM wparam, LPARAM
lparam)
{
    static HBRUSH hbrBkgnd =
NULL;                                                                   //用
于保存画刷的句柄
    // 处理窗口事件
    switch( msg )
    {
        // 设置窗口的最小和最大尺寸
        case WM_GETMINMAXINFO:
        {
            MINMAXINFO* minMaxInfo =
(MINMAXINFO*)lparam;                                                //MINMAXIN
FO 结构体指针
            minMaxInfo->ptMinTrackSize.x =
500;                                                    //设置窗口的最小宽
度    500
            minMaxInfo->ptMinTrackSize.y =
700;                                                    //设置窗口的最小高
度    700
            minMaxInfo->ptMaxTrackSize.x =
500;                                                    //设置窗口的最大宽
度    500
            minMaxInfo->ptMaxTrackSize.y =
700;                                                    //设置窗口的最大高
度    700
            return 0;
        }
        // 设置窗口的背景颜色
        case WM_ERASEBKGND:
        {
```

```
            RECT
rect;
        // 定义一个矩形结构体
            GetClientRect(hwnd,
&rect);                                                    // 获取
窗口的客户区域
            if (hbrBkgnd ==
NULL)                                                      //
如果画刷句柄为空
            {
                hbrBkgnd = CreateSolidBrush(RGB(50, 55,
255));                                  // 设置窗口的背景颜色
            }
            FillRect((HDC)wparam, &rect,
hbrBkgnd);                                             // 填充背景颜色
            return 1;
        }
        // 创建菜单
        case WM_CREATE:
        {
            HMENU hMenu,
hSubMenu01,hSubMenu02,hSubMenu03,hSubMenu04;
  //菜单，子菜单 01，子菜单 02，子菜单 03，子菜单 04
            hMenu =
CreateMenu();
      //创建菜单
            //菜单  文件
            hSubMenu01 =
CreatePopupMenu();
 //创建子菜单
            AppendMenu(hSubMenu01, MF_STRING, 1000, "Recognition card
1");                          //添加菜单项
            AppendMenu(hSubMenu01, MF_STRING, 1001, "Recognition card 2");
            AppendMenu(hSubMenu01, MF_STRING, 1002, "Recognition card 3");
            AppendMenu(hSubMenu01, MF_STRING, 1003, "Recognition card 4");
            AppendMenu(hSubMenu01, MF_STRING, 1004, "Recognition card 5");
            AppendMenu(hSubMenu01, MF_STRING, 1005, "Recognition card 6");
            AppendMenu(hSubMenu01, MF_STRING, 1006, "Recognition card 7");
            AppendMenu(hSubMenu01, MF_STRING, 1007, "Initialize");
            AppendMenu(hSubMenu01, MF_STRING, 1008, "Exit");
            AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu01,
"File");                       //将子菜单添加到菜单
            //菜单  工具
```

```
            hSubMenu02 =
CreatePopupMenu();
 //创建子菜单
            AppendMenu(hSubMenu02, MF_STRING, 2000, "Copy card number");
            AppendMenu(hSubMenu02, MF_STRING, 2001, "Open / Close the
console");
            AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu02,
"Tools");
            //菜单  外观
            hSubMenu03 = CreatePopupMenu();
            AppendMenu(hSubMenu03, MF_STRING, 3000, "Blue");
            AppendMenu(hSubMenu03, MF_STRING, 3001, "Gray");
            AppendMenu(hSubMenu03, MF_STRING, 3002, "Green");
            AppendMenu(hSubMenu03, MF_STRING, 3003, "Red");
            AppendMenu(hSubMenu03, MF_STRING, 3004, "Yellow");
            AppendMenu(hSubMenu03, MF_STRING, 3005, "White");
            AppendMenu(hSubMenu03, MF_STRING, 3006, "Purple");
            AppendMenu(hSubMenu03, MF_STRING, 3007, "Black");
            AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu03,
"Appearance");
            //菜单  帮助
            hSubMenu04 = CreatePopupMenu();
            AppendMenu(hSubMenu04, MF_STRING, 4000, "Auther");
            AppendMenu(hSubMenu04, MF_STRING, 4001, "About [Bank Card
Digital Recognition]");
            AppendMenu(hMenu, MF_STRING | MF_POPUP, (UINT)hSubMenu04,
"Help");
            SetMenu(hwnd,
hMenu);                                                                 /
/将菜单添加到窗口
        }   break;
        // 处理窗口的关闭事件
        case WM_COMMAND:
            switch( LOWORD(wparam) )
            {
                case 1: //判断原图是否存在
                {
                    Mat src = imread("原图.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "原图不存在！", "提示",
MB_OK|MB_ICONWARNING);            //弹出消息框
                    else
                        imshow("原图.jpg", src);
                }   break;
```

```
                case 2:
                {   //判断边缘检测是否存在
                    Mat src = imread("边缘检测.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "边缘检测不存在！", "提示",
MB_OK|MB_ICONWARNING);         //弹出消息框
                    else
                        imshow("边缘检测.jpg", src);
                }   break;
                case 3:
                {   //判断闭运算是否存在
                    Mat src = imread("闭运算.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "闭运算不存在！", "提示",
MB_OK|MB_ICONWARNING);           //弹出消息框
                    else
                        imshow("闭运算.jpg", src);
                }   break;
                case 4:
                {   //判断画框是否存在
                    Mat src = imread("画框.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "画框不存在！", "提示",
MB_OK|MB_ICONWARNING);             //弹出消息框
                    else
                        imshow("画框.jpg", src);
                }   break;
                case 5:
                {   //判断最长的矩形块是否存在
                    Mat src = imread("最长的矩形块.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "最长的矩形块不存在！", "提示",
MB_OK|MB_ICONWARNING);     //弹出消息框
                    else
                        imshow("最长的矩形块.jpg", src);
                }   break;
                case 6:
                {   //判断区域提取是否存在
                    Mat src = imread("card_number.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "区域提取不存在！", "提示",
MB_OK|MB_ICONWARNING);         //弹出消息框
                    else
                        imshow("card_number.jpg", src);
```

```
                }   break;
                case 7:
                {   //判断轮廓提取是否存在
                    Mat src = imread("号码轮廓图像.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "轮廓提取不存在！", "提示",
MB_OK|MB_ICONWARNING);    //弹出消息框
                    else
                        imshow("号码轮廓图像.jpg", src);
                }   break;
                case 8:
                {   //判断识别结果是否存在
                    Mat src = imread("识别结果.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "识别结果不存在！", "提示",
MB_OK|MB_ICONWARNING);    //弹出消息框
                    else
                        imshow("识别结果.jpg", src);
                }   break;
                case 9:
                {   //判断模板图像是否存在
                    Mat src = imread("template.jpg");
                    if (src.empty())
                        MessageBox(hwnd, "模板图像不存在！", "提示",
MB_OK|MB_ICONWARNING);    //弹出消息框
                    else
                        imshow("template.jpg", src);
                }   break;

                case 1000:      //识别卡 1
                {
                    string CARD_IMAGE_PATH = "card1.jpg";
                    cardNumber = Digital_Recognition(CARD_IMAGE_PATH);
                    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD |
WS_VISIBLE | ES_LEFT | ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL |
ES_READONLY, 180, 50, 250, 20, hwnd, NULL, hInst, NULL);
                    SetWindowText(hwndEdit1, cardNumber.c_str());
                    MessageBox(hwnd, "识别成功！o(￣T￣*)ゞ", "提示",
MB_OK|MB_ICONINFORMATION);    //弹出消息框 MessageBox(父窗口句柄，消息内容，
标题，消息框类型)
                }   break;
                case 1001:      //识别卡 2
                {
                    string CARD_IMAGE_PATH = "card2.jpg";
```

```
                    cardNumber = Digital_Recognition(CARD_IMAGE_PATH);
                    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD |
WS_VISIBLE | ES_LEFT | ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL |
ES_READONLY, 180, 50, 250, 20, hwnd, NULL, hInst, NULL);
                    SetWindowText(hwndEdit1, cardNumber.c_str());
                    MessageBox(hwnd, "识别成功! o(￣▽￣*)ゞ", "提示",
MB_OK|MB_ICONINFORMATION);    //弹出消息框
                } break;
                case 1002:      //识别卡 3
                {
                    string CARD_IMAGE_PATH = "card3.jpg";
                    cardNumber = Digital_Recognition(CARD_IMAGE_PATH);
                    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD |
WS_VISIBLE | ES_LEFT | ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL |
ES_READONLY, 180, 50, 250, 20, hwnd, NULL, hInst, NULL);
                    SetWindowText(hwndEdit1, cardNumber.c_str());
                    MessageBox(hwnd, "识别成功! o(￣▽￣*)ゞ", "提示",
MB_OK|MB_ICONINFORMATION);    //弹出消息框
                } break;
                case 1003:      //识别卡 4
                {
                    string CARD_IMAGE_PATH = "card4.jpg";
                    cardNumber = Digital_Recognition(CARD_IMAGE_PATH);
                    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD |
WS_VISIBLE | ES_LEFT | ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL |
ES_READONLY, 180, 50, 250, 20, hwnd, NULL, hInst, NULL);
                    SetWindowText(hwndEdit1, cardNumber.c_str());
                    MessageBox(hwnd, "识别成功! o(￣▽￣*)ゞ", "提示",
MB_OK|MB_ICONINFORMATION);    //弹出消息框
                } break;
                case 1004:      //识别卡 5
                {
                    string CARD_IMAGE_PATH = "card5.jpg";
                    cardNumber = Digital_Recognition(CARD_IMAGE_PATH);
                    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD |
WS_VISIBLE | ES_LEFT | ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL |
ES_READONLY, 180, 50, 250, 20, hwnd, NULL, hInst, NULL);
                    SetWindowText(hwndEdit1, cardNumber.c_str());
                    MessageBox(hwnd, "识别成功! o(￣▽￣*)ゞ", "提示",
MB_OK|MB_ICONINFORMATION);    //弹出消息框
                } break;
                case 1005:      //识别卡 6
                {
                    string CARD_IMAGE_PATH = "card6.jpg";
```

```
                    cardNumber = Digital_Recognition(CARD_IMAGE_PATH);
                    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD |
WS_VISIBLE | ES_LEFT | ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL |
ES_READONLY, 180, 50, 250, 20, hwnd, NULL, hInst, NULL);
                    SetWindowText(hwndEdit1, cardNumber.c_str());
                    MessageBox(hwnd, "识别成功！o(￣T￣*)ゞ", "提示",
MB_OK|MB_ICONINFORMATION);    //弹出消息框
                } break;
                case 1006:      //识别卡 7
                {
                    string CARD_IMAGE_PATH = "card7.jpg";
                    cardNumber = Digital_Recognition(CARD_IMAGE_PATH);
                    HWND hwndEdit1 = CreateWindow("EDIT", "", WS_CHILD |
WS_VISIBLE | ES_LEFT | ES_MULTILINE | ES_AUTOHSCROLL | WS_HSCROLL |
ES_READONLY, 180, 50, 250, 20, hwnd, NULL, hInst, NULL);
                    SetWindowText(hwndEdit1, cardNumber.c_str());
                    MessageBox(hwnd, "识别成功！o(￣T￣*)ゞ", "提示",
MB_OK|MB_ICONINFORMATION);    //弹出消息框
                } break;
                case 1007: //复位
                {
                    destroyAllWindows();                              //关闭
所有图片
                    InitializeWindow(hwnd, hInst);                    //重新
初始化窗口
                    DestroyAllWindows();                              //文件
清理
                    MessageBox(hwnd, "初始化完成！\(￣～￣*\))", "提示",
MB_OK|MB_ICONASTERISK);
                }break;
                case 1008:      //退出
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    destroyAllWindows();
      //关闭所有图片
                    PostQuitMessage(0);
      //退出消息循环
                    DestroyAllWindows();
      //文件清理
```

```
                    exit(0);
    //退出程序
                } break;
                case 2000:      //复制卡号
                {
                    copyToClipboard(cardNumber);
                    MessageBox(hwnd, "\(￣^￣*\))已将卡号复制到剪切板！", "
提示", MB_OK|MB_ICONINFORMATION);    //弹出消息框
                }   break;
                case 2001:
                {
                    sel =
~sel;                                           //取反
                    if(sel)
    //打开控制台
                    {
                        ShowWindow(GetConsoleWindow(),
SW_SHOW);                   //隐藏控制台窗口 ShowWindow(窗口句柄，显示方
式)   SW_HIDE 隐藏窗口    SW_SHOW 显示窗口
                        MessageBox(hwnd, "控制台成功打开！", "提示",
MB_OK|MB_ICONWARNING);    //弹出消息框
                    }
                    else    //关闭控制台
                    {
                        ShowWindow(GetConsoleWindow(),
SW_HIDE);                   //隐藏控制台窗口 ShowWindow(窗口句柄，显示方
式)   SW_HIDE 隐藏窗口    SW_SHOW 显示窗口
                        MessageBox(hwnd, "控制台成功关闭！", "提示",
MB_OK|MB_ICONWARNING);    //弹出消息框
                    }
                }   break;
                case 3000:  //修改背景颜色  蓝色
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
     //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(50, 55,
255));             // 设置窗口的背景颜色
                    InvalidateRect(hwnd, NULL,
TRUE);                          //刷新窗口
                } break;
                case 3001:  //修改背景颜色  灰色
```

```
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(128, 128,
128));           // 设置窗口的背景颜色
                    InvalidateRect(hwnd, NULL,
TRUE);                          //刷新窗口
                } break;
                case 3002:  //修改背景颜色  绿色
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(0, 255,
0));              // 设置窗口的背景颜色
                    InvalidateRect(hwnd, NULL,
TRUE);                          //刷新窗口
                } break;
                case 3003:  //修改背景颜色  红色
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(255, 0,
0));               //设置窗口的背景颜色
                    InvalidateRect(hwnd, NULL,
TRUE);                          //刷新窗口
                } break;
                case 3004:  //修改背景颜色  黄色
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(255, 255,
0));              //设置窗口的背景颜色
```

```
                    InvalidateRect(hwnd, NULL,
TRUE);                              //刷新窗口
                } break;
                case 3005:  //修改背景颜色  白色
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(255, 255,
255));            //设置窗口的背景颜色
                    InvalidateRect(hwnd, NULL,
TRUE);                              //刷新窗口
                } break;
                case 3006:  //修改背景颜色  紫色
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(128, 0,
255));             //设置窗口的背景颜色
                    InvalidateRect(hwnd, NULL,
TRUE);                              //刷新窗口
                } break;
                case 3007:  //修改背景颜色  黑色
                {
                    if (hbrBkgnd)
                    {
                        DeleteObject(hbrBkgnd);
      //删除背景颜色
                    }
                    hbrBkgnd = CreateSolidBrush(RGB(0, 0,
0));                 //设置窗口的背景颜色
                    InvalidateRect(hwnd, NULL,
TRUE);                              //刷新窗口
                } break;
                case 4000:  MessageBox(hwnd, "大连东软信息学院\n 智能与电子
工程学院\n 微电子工程系\n 集成电路设计与集成系统\n 作者：集成电路 21003 班  王明
杰\n 学号：21003160314", "作者", MB_OK|MB_ICONASTERISK);   break;
                case 4001:  MessageBox(hwnd, "数字图像处理三级项目：银行卡数
字识别\nVersion:  2.5.6\nLast modification time:  2023/12/3 10:26:56\n 版
```

```
权所有(C)2023 WangMingJie。\n 保留所有权利。\n 警告：本计算机程序受著作权法和
国际条约保护。如未经授权而擅自复制或传播本程序(或其中任何部分)，将受到严厉的民
事和刑事制裁，并将在法律许可的最大限度内受到起诉。", "关于",
MB_OK|MB_ICONASTERISK);   break;
                default:    break;
             }  break;
        // 处理窗口的关闭事件
        case WM_CLOSE:
            if (hbrBkgnd)
            {
                DeleteObject(hbrBkgnd);
      //删除背景颜色
            }
            PostQuitMessage(0);
      //退出消息循环
            break;
        default:
            return DefWindowProc(hwnd, msg, wparam,
lparam);                    //默认消息处理函数
    }
    return 0;
}

//WinMain 函数
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, int
nCmdShow)                           //WinMain(应用程序实例句柄，应用程序实
例句柄，命令行参数，显示方式)
{
    const char CLASS_NAME[] = "Bank Card Digital Recognition
V2.5.6";                            //窗口类名
    WNDCLASS wc =
{ };
   //窗口类
    wc.lpfnWndProc =
WindowProcedure;
 //窗口过程函数
    wc.hInstance =
hInst;
  //应用程序实例句柄
    wc.lpszClassName =
CLASS_NAME;                                                    /
/窗口类名
    if(!RegisterClass(&wc))
                    //注册窗口类
```

```
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
MB_ICONEXCLAMATION | MB_OK);  //弹出消息框
        return 0;
    }
    HWND hwnd = CreateWindow(CLASS_NAME, "Bank Card Digital Recognition
V2.5.6", WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 500, 300, NULL,
NULL, hInst, NULL);//CreateWindow(窗口类名，窗口标题，窗口样式，窗口位置，窗
口大小，父窗口句柄，菜单句柄，应用程序实例句柄，指向创建窗口的参数的指针)
    if(hwnd ==
NULL)
      //创建窗口失败
    {
        MessageBox(NULL, "Window Creation Failed!", "Error!",
MB_ICONEXCLAMATION | MB_OK);      //弹出消息框
        return 0;
    }
    InitializeWindow(hwnd,
hInst);                                                    //初始
化窗口
    ShowWindow(hwnd,
nCmdShow);
 //显示窗口
    UpdateWindow(hwnd);
                    //更新窗口
    MSG msg =
{ };
       //消息
    while(GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
                    //翻译消息
        DispatchMessage(&msg);
                    //分发消息
    }
    return 0;
}
//  main function   主函数  程序入口
int main()
{
    system("mode con cols=80
lines=28");                                                //改变控
制台窗口大小
```

```
    system("color
1F");
   //改变控制台窗口颜色
    ShowWindow(GetConsoleWindow(),
SW_HIDE);                                                    //隐藏控制台窗
口 ShowWindow(窗口句柄，显示方式)   SW_HIDE 隐藏窗口   SW_SHOW 显示窗口
    try
                    //异常处理
    {
        WinMain(GetModuleHandle(NULL), NULL, NULL,
SW_SHOW);                                        //WinMain(应用程序实例句柄，应
用程序实例句柄，命令行参数，显示方式)
    }
    catch (const exception&
e)                                                           //捕获
异常
    {
        cerr << e.what() <<
endl;                                                        //输出
错误信息
    }
    return 0;
}
```

第4章 系统的测试与结论

4.1 银行卡数字识别系统测试的方法

银行卡数字识别系统测试的方法主要包括以下步骤：

1. 搭建测试环境：测试环境需要与实际应用环境保持一致，包括硬件设备、操作系统、数据库、网络连接等。
2. 准备测试数据：收集各种类型的银行卡图片，包括正反面、清晰度不同的照片、不同字体和颜色等，以便对系统的识别能力进行全面测试。
3. 设计与执行测试计划：根据测试目标和测试需求，设计相应的测试用例，包括输入正确的卡号、输入错误的卡号、遮挡或模糊的卡号等。在测试过程中，需要记录每个测试用例的执行结果和异常情况。
4. 异常处理：对于测试过程中出现的异常情况，需要及时记录并进行分析。有些异常情况可能是由于测试数据不规范或测试环境问题引起的，需要进行相应的处理和解决。

5. 测试结果分析：根据测试结果，对系统的识别准确率、识别速度、稳定性等方面进行评估和分析，并与预期结果进行比较，判断系统是否达到预期要求。
6. 调整与优化：根据测试结果的分析，对系统的不足之处进行调整和优化，包括算法改进、参数调整等。
7. 重复测试：经过调整和优化后，需要再次进行测试，以确保系统的性能和质量得到有效提升。

在银行卡数字识别系统测试中，需要注意以下几点：

1. 测试数据的多样性：为了全面评估系统的性能，需要收集多种类型的银行卡图片，包括不同品牌、不同类型、不同清晰度和不同背景等。
2. 测试的严格性：在测试过程中，需要遵循严格的测试规范和流程，确保每个测试用例的准确性和可重复性。
3. 异常处理的及时性：对于出现的异常情况，需要及时记录并进行分析，以便及时解决问题和调整系统。
4. 结果分析的客观性：测试结果的分析需要客观公正，不夸大其词或掩盖问题，同时要与预期结果进行比较，以评估系统的实际性能和质量。

4.2 银行卡数字识别系统测试结果与结论

测试结果：

在本次测试中，我针对银行卡数字识别系统的各项功能和性能进行了全面的测试。以下是测试结果：

1. 图像预处理功能正常，能够准确去除图像中的噪声和干扰，使得数字图像更加清晰。
2. 模板匹配功能正常，能够在复杂的背景中准确地找到银行卡的数字位置。
3. 边缘检测和轮廓检测功能正常，能够准确地提取出数字的形状和轮廓。
4. 形态学操作功能正常，能够有效地去除数字周围的干扰和噪声，提高数字识别的准确率。
5. Windows API 函数调用正常，能够实现图像的显示和保存等功能。

在测试过程中，系统运行稳定，没有出现明显的错误或异常情况。同时，我使用多种不同类型和质量的图像进行测试，结果表明系统能够准确地识别出银行卡上的数字，并且对不同类型和质量的图像具有较强的适应性。

结论：

经过全面的测试，我得出以下结论：本系统的各项功能和性能均达到预期目标，能够准确、稳定、快速地实现银行卡数字识别。同时，系统具有较强的鲁棒性和适应性，能够应对不同类型和质量的图像输入。因此，本系统可以广泛应用于银行、保险、证券等金融领域中，提高银行卡信息处理的效率和准确性，为金融行业的发展提供有力的支持。

第5章 项目展望

项目总结：银行卡数字识别系统

一、项目背景与目的

银行卡数字识别系统是一个利用数字图像处理技术的项目，旨在识别银行卡上的数字字符，提高银行办理业务的效率，减少人为错误和欺诈行为的发生，同时提高银行卡使用的安全性和便利性。本项目的主要目的包括：

1. 掌握数字图像处理的基本理论和技术；
2. 培养解决实际问题的能力；
3. 探索机器学习和深度学习等先进算法的应用；
4. 提高编程能力和科研素养。

二、项目实施过程

1. 图像预处理：对采集的图像进行一系列预处理操作，包括去噪、图像增强、二值化和分割等，以突出银行卡上的数字信息。
2. 数字识别：采用基于 OpenCV 的模板匹配技术，将经过预处理的图像中的数字字符识别出来，并转换为计算机可读的文本格式。
3. 信息提取：从识别出的文本中提取有用的信息，如卡号、持卡人姓名和有效期等。
4. 信息验证：核对数字字符的语法和结构是否正确、卡号是否有效等，以确保识别结果的准确性和有效性。
5. 数据存储：将经过验证的银行卡信息存储在系统的数据库中，以便后续的处理和使用。
6. 用户界面：为了方便用户操作和使用，系统需要提供一个友好的用户界面，包括输入界面、显示界面和操作界面等。

三、项目成果与价值

通过本项目的实施，我取得了以下成果：

1. 实现了银行卡数字识别的基本功能，包括图像预处理、数字识别、信息提取、信息验证、数据存储和用户界面等；
2. 探索了机器学习和深度学习等先进算法在银行卡数字识别中的应用；
3. 提高了编程能力和科研素养；
4. 为银行提供了更高效、准确的银行卡办理方式，提高了业务办理效率；
5. 减少了人为错误和欺诈行为的发生，提高了银行卡使用的安全性；
6. 促进了图像处理和计算机视觉领域的发展；
7. 为探索人工智能技术在金融领域的应用奠定了基础；

8. 为培养具备数字图像处理、机器学习、深度学习等方面技能的人才提供了实践机会。

四、项目不足与展望

尽管本项目的实施取得了一定的成果，但仍存在一些不足之处，例如：

1. 在图像预处理阶段，对于复杂背景和不同光照条件下的处理效果不够理想，需要进一步优化算法；
2. 在数字识别阶段，对于某些字体、大小和布局的数字字符的识别准确率还有待提高，需要进一步探索更有效的特征提取和分类算法；
3. 在用户界面方面，还需要进一步完善功能和操作流程，提高用户体验。

展望未来，我可以从以下几个方面继续探索和完善本项目：

1. 优化图像预处理算法，提高处理效果和泛化能力；
2. 采用更先进的特征提取和分类算法，提高数字字符的识别准确率和鲁棒性；
3. 增加用户界面的功能和操作流程，提高用户体验和交互性；
4. 将本项目应用于其他领域的图像识别问题中，拓展其应用范围和价值。

附录：

1．小组成员

	组长	组员			
姓名	XXX	XXX	XXX	XXX	XXX
学号	21003160314	21003160301	21003160305	21003160306	21003160318

2．个人对项目设计的贡献

学号		21003160314	21003160301	21003160305	21003160306	21003160318
姓名		XX	XXX	XXX	XXX	XXX
三级项目	贡献比100%	60%	10%	10%	10%	10%
	具体工作	系统设计、代码编写、系统调试	流程图设计	报告撰写	报告校对	报告校对