身份证号识别报告

## 一、选题分析

项目需求：使用 OpenCV 库实现身份证号的识别，即导入一张身份证的图像，通过调用 OpenCV 的相关函数来将身份证的数字（号）识别出来。具体流程如下：



## 二、模块分解

### 1.预处理

#### 1.1比例缩放至预设大小

**1)初始化变量：**

1. **int** lr = 0, lc = 0;                             // 记录迭代时较小的 row col
2. **int** hr = idcard.rows, hc = idcard.cols;         // 记录迭代时较大的 row col
3. **int** mr = (lr + hr) >> 1, mc = (lc + hc) >> 1;   // 记录迭代时二分点 row col
4. **int** size = mr \* mc;                             // 当前二分点尺寸
5. **int** toSize = toRows \* toCols;                   // 目标尺寸
6. **int** sub = toSize - size;                        // 当前尺寸与目标尺寸差

 lr, lc：初始值为0，用于记录较小的行和列。

 hr, hc：初始值为图像的行数和列数，用于记录较大的行和列。

 mr, mc：初始值为行和列的一半，用于记录当前二分点的行和列。

 size：当前二分点的尺寸（行数乘以列数）。

 toSize：目标尺寸（目标行数乘以目标列数）。

 sub：当前尺寸与目标尺寸的差值。

**2)计算允许的尺寸误差：**

1. int delta = (toCols + toRows) << 1;               // 允许缩放至目标尺寸的误差

·delta：允许的尺寸误差，设置为目标行数和列数之和的两倍。通过左移运算（乘以2）来计算。

**3）迭代调整尺寸：**

1. **while** (abs(sub) > delta)                     // 差大于误差时继续循环
2. {
3. // 原图大于目标图
4. **if** (sub < 0) { hr = mr; hc = mc; }           // 降低较大的 row col
5. // 原图小于目标图
6. **else** { hr += mr; hc += mc; lr = mr; lc = mc; }// 提升较小的 row col
7. mr = (hr + lr) >> 1;                      // 更新二分点 row
8. mc = (hc + lc) >> 1;                      // 更新二分点 col
9. size = mr \* mc;                             // 更新当前尺寸
10. sub = toSize - size;                        // 更新当前差值
11. }

* 使用while循环，迭代调整尺寸，直到当前尺寸与目标尺寸的差值小于允许误差。
* 如果当前尺寸大于目标尺寸，则降低较大的行和列。
* 如果当前尺寸小于目标尺寸，则提升较小的行和列。
* 更新二分点的行和列，以及当前尺寸和差值。

**4)调整图像大小：**

1. cv::resize(idcard, idcard, Size(mc, mr));       // OpenCV 双线性插值缩放

使用OpenCV的cv::resize函数将图像调整为二分法确定的尺寸。Size(mc, mr)表示最终的目标尺寸，cv::resize函数默认使用双线性插值进行缩放。

#### 1.2 数字尺寸归一化

实现步骤：

1）分割出每个数字图像。

2）计算尺寸差异：计算图像的高宽差值。

3）填充边框：根据差值填充边框，使其变为正方形。

4）缩放至固定大小：将图像缩放到28x28大小。

**1）计算高宽差值：**

1. int k = \_img.rows - \_img.cols;                      // 高 - 宽

计算图像的高度与宽度之差k，用于确定需要填充的边界。

**2）判断高宽关系并填充边界：**

1. **if** (k > 0) {                                   // 如果宽 < 高 用黑色填充左右部分
2. cv::copyMakeBorder(\_img, \_img, 0, 0, k/2, k - k/2, cv::BORDER\_CONSTANT, 0);
3. }
4. **else** {                                              // 用黑色填充上下部分
5. k = -k;
6. cv::copyMakeBorder(\_img, \_img, k/2, k - k/2, 0, 0, cv::BORDER\_CONSTANT, 0);
7. }

如果k > 0（即高度大于宽度），在图像左右两侧填充黑色边界，使其宽度等于高度：

1. cv::copyMakeBorder(\_img, \_img, 0, 0, k/2, k - k/2, cv::BORDER\_CONSTANT, 0);

 cv::copyMakeBorder函数用于在图像边缘填充边界。参数分别为：

* 1. 输入图像\_img
  2. 输出图像\_img
  3. 上下边界填充0
  4. 左右边界填充k/2和k - k/2
  5. 边界类型为cv::BORDER\_CONSTANT，即填充常数值
  6. 填充值为0，即黑色

如果k <= 0（即宽度大于或等于高度），在图像上下两侧填充黑色边界，使其高度等于宽度：

1. k = -k;
2. cv::copyMakeBorder(\_img, \_img, k/2, k - k/2, 0, 0, cv::BORDER\_CONSTANT, 0);

 更新k为其绝对值

 cv::copyMakeBorder函数在上下两侧填充黑色边界

**3）缩放图像到指定大小：**

使用OpenCV的cv::resize函数将图像缩放到指定的28x28大小：

1. cv::resize(\_img, \_img, cv::Size(28, 28));

#### 1.3滤波、边缘、二值化

实现步骤：

• 双边滤波：去除了图像中的噪声，同时保留了边缘，增强了图像的平滑度。

• 灰度化：将彩色图像转换为灰度图像，简化后续的处理步骤，并减少计算复杂度。

• 形态学边缘检测：通过闭运算和作差提取图像的边缘，增强了图像的轮廓特征。

• 二值化：将图像转换为二值图像，便于后续的目标识别和分析。

**1）双边滤波：**

1. bilateralFilter(idcard, img, 7, 10, 5);             // 双边滤波
2. idcard = img.clone();                               // 保存双边滤波后的原图
3. **if** (DEBUG)
4. {
5. imshow("predeal\_0\_bilateraFilter", img);
6. }

·bilateralFilter函数对图像进行双边滤波。

·双边滤波在保留边缘的同时平滑图像，有效地去除噪声。

·7 是过滤器的直径，10 是空间域的标准差，5 是色彩域的标准差。

·滤波后的图像被克隆并保存为 idcard。



**2）灰度化：**

1. cvtColor(img, img, COLOR\_BGR2GRAY);
2. **if** (DEBUG)
3. {
4. imshow("predeal\_1\_gray", img);
5. }



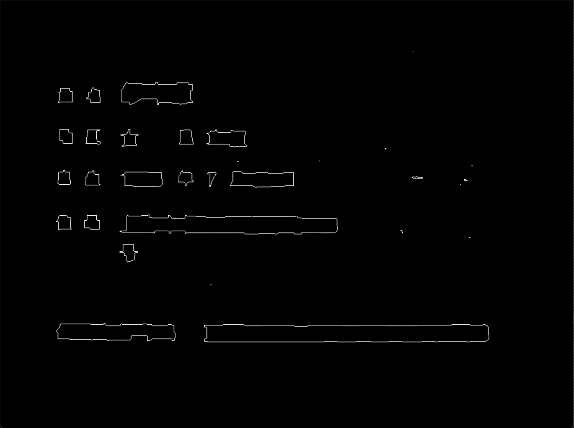
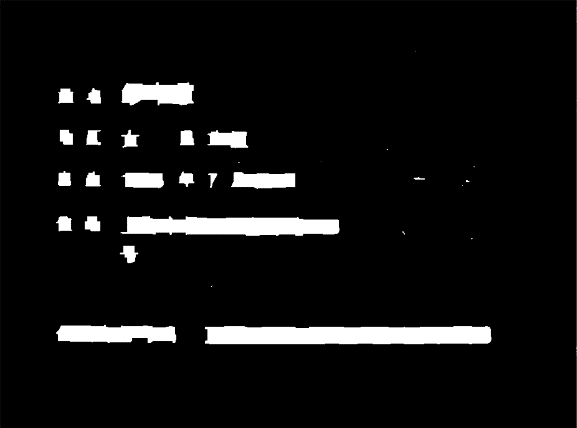
**3）形态学边缘检测：**

1. Mat tmp = img.clone();                              // 临时存储空间
2. morphologyEx(                                       // 形态学运算
3. img,                                            // 输入图像
4. tmp,                                            // 输出图像
5. MORPH\_CLOSE,                                    // 指定闭运算 连通破碎区域
6. getStructuringElement(MORPH\_RECT, Size(7, 7))   // 获取结构核
7. );
8. img = tmp - img;                                    // 作差
10. **if** (DEBUG)
11. {
12. imshow("predeal\_2\_close", tmp);
13. imshow("predeal\_3\_gray - close", img);
14. }
16. tmp.release();                                      // 释放临时空间

1.创建一个克隆的临时图像 tmp。

2.使用 morphologyEx 函数进行形态学运算，指定 MORPH\_CLOSE 操作，MORPH\_CLOSE 操作用于闭合小的孔洞，连通破碎的区域。使用 getStructuringElement 函数获取一个大小为 7x7 的矩形结构元素。

3.将形态学运算后的结果与原图做差，以提取边缘信息。



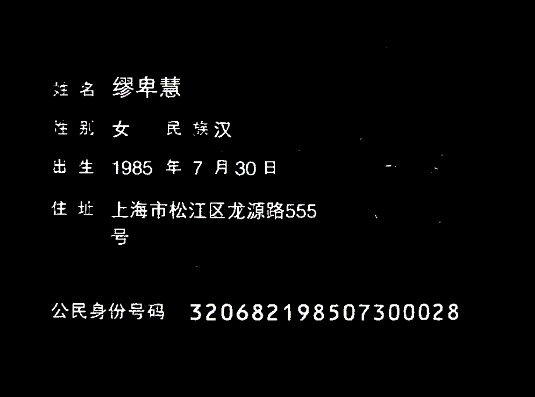
**4）二值化：**

1. threshold(img, img, 0, 255, THRESH\_OTSU);           // 二值化
2. **if** (DEBUG)
3. {
4. imshow("predeal\_4\_threshold\_otsu", img);
5. }

 使用 threshold 函数进行二值化操作，指定 THRESH\_OTSU 方法。

 THRESH\_OTSU 方法自动计算全局阈值，以分割前景和背景。

 二值化后的图像，前景像素值为255，背景像素值为0。



### 2.SVM模型训练

支持向量机（Support Vector Machine，SVM）是一种监督学习算法，用于分类和回归分析。核心思想是找到一个最佳的超平面来分隔不同类别的数据点。在分类问题中，SVM的目标是找到一个能够最大化类别间距离（margin）的超平面，这意味着找到一个能够将不同类别的数据点有效分开的决策边界。

1. 最大化间隔：SVM试图找到一个决策边界（超平面），使得两个不同类别的数据点到该超平面的距离尽可能远。这个距离被称为间隔，SVM的目标是最大化这个间隔，这样可以提高模型的泛化能力。

2. 支持向量：在SVM中，距离超平面最近的那些数据点被称为支持向量。这些支持向量决定了最终的超平面的位置，因此SVM的决策边界是通过支持向量来定义的。

3. 核技巧：在处理非线性可分的数据时，SVM使用核技巧将数据映射到高维空间中，使得数据在该空间中线性可分。常用的核函数包括线性核、多项式核、高斯核等。

4. 正则化参数：SVM中通常会引入正则化参数来平衡间隔的最大化和误分类点的惩罚，以避免过拟合。

首先通过Card::setTrain 函数实现了一个完整的 SVM 模型训练过程，包括数据加载与预处理、模型配置与训练、模型保存。

1. Mat trainImages;
2. vector<**int**> trainLabels;
3. svm = SVM::create();
4. svm->setType(SVM::C\_SVC);
5. svm->setKernel(SVM::LINEAR);
6. svm->setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER, 100, 1e-6));

首先完成数据结构初始化，接着通过以下代码进行SVM 模型的创建和配置：SVM::create() 创建一个 SVM 实例。

setType(SVM::C\_SVC) 设置 SVM 类型为 C-SVC（支持向量分类）。

setKernel(SVM::LINEAR) 设置核函数为线性核。

setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER, 100, 1e-6)) 设置终止条件为最大迭代 100 次或精度 1e-6。

1. **for** (auto x : classes)
2. {
3. getFileNames(trainDataFolderPath + to\_string(x), files);
4. **for** (auto file : files)
5. {
6. img = imread(file, 0);
7. **if** (img.empty()) **continue**;
9. threshold(img, img, 10, 255, THRESH\_OTSU);
10. img.convertTo(img, CV\_32FC1);
11. trainImages.push\_back(img.reshape(0, 1));
12. trainLabels.push\_back(x);
13. }
14. files.clear();
15. }
16. img.release();

使用 getFileNames 函数获取每个类别文件夹中的所有图片文件名。对每个文件名，读取图像并进行预处理（灰度化、二值化、转换数据类型、展平）。将预处理后的图像数据和对应标签添加到 trainImages 和 trainLabels 中。

1. svm->train(trainImages, ROW\_SAMPLE, trainLabels);
2. svm->save(trainDataFolderPath + "svm.xml");
3. svm->clear();

使用 svm->train 方法进行模型训练。训练完成后，将模型保存为 svm.xml 文件。

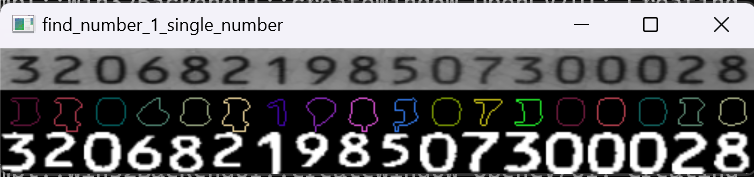
### 3.数字识别、校验、分析

这里主要分为两个部分：Card::predict 函数用于识别号码，Card::correct 函数用于校验识别结果是否正确。

**1）识别**

1. **for** (auto itr = idcardNumbers.begin(); itr != idcardNumbers.end(); itr++)
2. {
3. itr->convertTo(img, CV\_32FC1);
4. **float** res = svm->predict(img.reshape(0, 1));
5. predictNumber.push\_back(res);
6. }

遍历保存的号码图像（idcardNumbers），对每个图像进行处理和预测。将每个图像转换为 CV\_32FC1 类型的单通道浮点图像。使用 SVM 模型对转换后的图像进行预测，将预测结果存储到 predictNumber 中。



**2）校验**

1. **int** sum = 0;
2. vector<**int**> W = { 7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2, 1 };
3. **for** (**int** i = 0; i < W.size(); i++)
4. sum += predictNumber[i] \* W[i];
5. sum %= 11;
6. **if** (
7. sum == 1
8. || (sum == 10 && predictNumber.back() == 10)
9. || mapPlace.find(predictNumber[0]\*10 + predictNumber[1]) != mapPlace.end()
10. )
11. **return** 0;
12. **else**
13. **return** 1;

使用权重向量 W 计算加权和，然后对 11 取余。权重向量 W 是身份证号码校验码计算的一部分。

校验规则：

* 如果余数为 1，校验通过。
* 如果余数为 10 并且最后一位是 10（即 'X'），校验通过。
* 如果前两位（即省市代码）在 mapPlace 中找到，校验通过。

**3）分析**

首先通过Card 类构造函数初始化身份证号码前两位数字对应的省份信息

1. Card::Card()
2. {
3. // 身份证号前两位数字
4. vector<**int**> key = {
5. 11, 12, 13, 14, 15,
6. 21, 22, 23,
7. 31, 32, 33, 34, 35, 36, 37,
8. 41, 42, 43,
9. 44, 45, 46,
10. 51, 52, 53, 54, 50,
11. 61, 62, 63, 64, 65,
12. 71, 81, 82
13. };
14. vector<string> value = {
15. "北京","天津","河北","山西","内蒙古",
16. "辽宁","吉林","黑龙江",
17. "上海","江苏","浙江","安徽","福建","江西","山东",
18. "河南","湖北","湖南",
19. "广东","广西","海南",
20. "四川","贵州","云南","西藏","重庆",
21. "陕西","甘肃","青海","宁夏","新疆",
22. "台湾","香港","澳门"
23. };
25. **for** (**int** i = 0; i < key.size(); i++)
26. mapPlace[key[i]] = value[i];
27. }

key 和 value 两个向量分别存储了身份证号码前两位数字及其对应的省份名称。构造函数通过遍历 key 向量，将 key[i] 和 value[i] 作为键值对存储在 mapPlace 映射中。

1. string provinceCode = txt.substr(0, 2);
2. string province = mapPlace[stoi(provinceCode)];

provinceCode 从身份证号码 txt 中提取前两位字符，表示省份代码。 mapPlace[stoi(provinceCode)] 使用 provinceCode 从 mapPlace 映射中获取对应的省份名称。

1. string birthYear = txt.substr(6, 4);
2. string birthMonth = txt.substr(10, 2);
3. string birthDay = txt.substr(12, 2);

birthYear 提取身份证号码中第 7 到 10 位字符，表示出生年份。

birthMonth 提取第 11 和 12 位字符，表示出生月份。

birthDay 提取第 13 和 14 位字符，表示出生日期。

1. **char** genderCode = txt[16];
2. string gender = (genderCode - '0') % 2 == 0 ? "Female" : "Male";

genderCode 提取身份证号码中第 17 位字符，表示性别代码。

(genderCode - '0') % 2 == 0 ? "Female" : "Male" 根据性别代码的奇偶性确定性别，偶数为女性 ("Female")，奇数为男性 ("Male")。

### 4.界面与整合

这个部分定义了一个 IDCardDetect\_QtClass 类，该类继承自 QMainWindow。在Qt界面中实现了图像上传与显示、身份证号信息的提取、图片的切换功能。

#### 4.1 类关系

IDCardDetect\_QtClass: 这是一个继承自 QMainWindow 的类，代表主窗口。该类包含了用户界面的定义和功能的实现。

Ui::IDCardDetect\_QtClass: 这是由 Qt Designer 生成的 UI 类，包含了用户界面元素的定义。通过调用 ui.setupUi(this)，将 UI 元素设置到主窗口中。

Card: 这是一个自定义的类，用于处理身份证识别的逻辑。在本代码中，该类包含了用于识别身份证的方法和一些配置信息。

#### 4.2 Qt 信号与槽机制

1. connect(ui.pushButton, &QPushButton::clicked, **this**, &IDCardDetect\_QtClass::on\_pushButton\_clicked);
2. connect(ui.pushButton\_3, &QPushButton::clicked, **this**, &IDCardDetect\_QtClass::on\_pushButton\_3\_clicked);
3. connect(ui.pushButton\_2, &QPushButton::clicked, **this**, &IDCardDetect\_QtClass::on\_pushButton\_2\_clicked);

以上代码将三个按钮的 clicked 信号连接到相应的槽函数。当按钮被点击时，对应的槽函数会被调用。

#### 4.3 功能和原理

在构造函数中，初始化了一些成员变量和界面

1. IDCardDetect\_QtClass::IDCardDetect\_QtClass(QWidget\* parent)
2. : QMainWindow(parent), currentIndex(-1)
3. {
4. ui.setupUi(**this**);
5. scene = **new** QGraphicsScene(**this**);
6. ui.graphicsView->setScene(scene);
8. // 初始化 Card 类的配置
9. card.setPicFolderPath("E:\\Data\\ID\_Card\_Recognize\\test");
10. card.setTrainDataFolderPath("E:\\Data\\ID\_Card\_Recognize\\reference\\idcard-opencv-svm\\idcard\\data\\trainData\\");
11. card.setSavePath("E:\\Data\\ID\_Card\_Recognize\\save\\");
12. card.setTrain("FALSE");
13. card.setDebug("FALSE");
15. // 连接按钮点击信号到槽函数
16. connect(ui.pushButton, &QPushButton::clicked, **this**, &IDCardDetect\_QtClass::on\_pushButton\_clicked);
17. connect(ui.pushButton\_3, &QPushButton::clicked, **this**, &IDCardDetect\_QtClass::on\_pushButton\_3\_clicked);
18. connect(ui.pushButton\_2, &QPushButton::clicked, **this**, &IDCardDetect\_QtClass::on\_pushButton\_2\_clicked);
19. }

on\_pushButton\_clicked 槽函数用于打开文件对话框，使用 OpenCV 读取图像并存储在 images 和 imagePaths 中

1. **void** IDCardDetect\_QtClass::on\_pushButton\_clicked()
2. {
3. // 打开文件对话框
4. QFileDialog dialog(**this**);
5. dialog.setFileMode(QFileDialog::ExistingFiles);
6. dialog.setNameFilter(tr("Image Files (\*.png \*.jpg \*.bmp)"));
7. **if** (dialog.exec() != QDialog::Accepted)
8. **return**;
10. QStringList fileNames = dialog.selectedFiles();
12. // 清空图像列表并重置
13. images.clear();
14. imagePaths.clear();
15. currentIndex = -1;
16. scene->clear();
18. // 使用 OpenCV 读取图像并存储
19. **for** (**const** QString& fileName : fileNames)
20. {
21. cv::Mat img = cv::imread(fileName.toStdString());
22. **if** (!img.empty())
23. {
24. images.push\_back(img);
25. imagePaths.push\_back(fileName.toStdString());
26. }
27. }
29. // 确保图像和路径顺序一致
30. **if** (!images.empty())
31. {
32. currentIndex = 0;
33. displayImage(currentIndex);
34. }
35. }

displayImage 函数用于在 QGraphicsView 中显示当前索引的图像：

1. **void** IDCardDetect\_QtClass::displayImage(**int** index)
2. {
3. **if** (index < 0 || index >= images.size())
4. **return**;
6. qDebug() << "Displaying image at index: " << index;
8. cv::Mat img = images[index];
9. cv::Mat rgbImg;
10. cv::cvtColor(img, rgbImg, cv::COLOR\_BGR2RGB);
12. QImage qImg(rgbImg.data, rgbImg.cols, rgbImg.rows, rgbImg.step, QImage::Format\_RGB888);
14. QGraphicsPixmapItem\* item = **new** QGraphicsPixmapItem(QPixmap::fromImage(qImg));
15. scene->clear();
16. scene->addItem(item);
17. ui.graphicsView->fitInView(item, Qt::KeepAspectRatio);
19. // 显示身份证识别信息
20. std::string filePath = imagePaths[index];
21. displayIDCardInfo(filePath);
22. }

displayIDCardInfo 函数用于显示识别出的身份证信息：

1. **void** IDCardDetect\_QtClass::displayIDCardInfo(**const** std::string& filePath)
2. {
3. std::string txt;
4. **int** res = card.identify(filePath, txt);
6. **if** (res == 0)
7. {
8. // 获取省份信息
9. std::string provinceCode = txt.substr(0, 2);
10. std::string province = card.getMapPlace().at(stoi(provinceCode));
12. // 获取出生日期信息
13. std::string birthYear = txt.substr(6, 4);
14. std::string birthMonth = txt.substr(10, 2);
15. std::string birthDay = txt.substr(12, 2);
17. // 获取性别信息
18. **char** genderCode = txt[16];
19. std::string gender = (genderCode - '0') % 2 == 0 ? "Female" : "Male";
21. // 设置 QLabel 文本
22. QString labelText = QString("ID Number: %1\nProvince: %2\nBirthdate: %3-%4-%5\nGender: %6")
23. .arg(QString::fromStdString(txt))
24. .arg(QString::fromStdString(province))
25. .arg(QString::fromStdString(birthYear))
26. .arg(QString::fromStdString(birthMonth))
27. .arg(QString::fromStdString(birthDay))
28. .arg(QString::fromStdString(gender));
29. ui.label->setText(labelText);
30. }
31. **else**
32. {
33. ui.label->setText("Error identifying ID card.");
34. }
35. }

通过槽函数 on\_pushButton\_3\_clicked 和 on\_pushButton\_2\_clicked 来切换显示的图像：

1. **void** IDCardDetect\_QtClass::on\_pushButton\_3\_clicked()
2. {
3. **if** (currentIndex > 0)
4. {
5. currentIndex--;
6. displayImage(currentIndex);
7. }
8. }
10. **void** IDCardDetect\_QtClass::on\_pushButton\_2\_clicked()
11. {
12. **if** (currentIndex < images.size() - 1)
13. {
14. currentIndex++;
15. displayImage(currentIndex);
16. }
17. }

## 三、最终效果



## 四、总结

该项目主要通过OpenCV结合SVM支持向量机对身份证号码进行了提取，对于提取到的身份证号码进一步分析得到相关信息，同时结合Qt界面实现多张图片的上传与显示。在实际测试的26张图片中有4张未能正确识别，初步判定为以下原因导致：

1. 对于光照的鲁棒性不够强

2. 身份证轮廓不够完整未能正确提取

3. 没有针对角度问题做透视变换、矫正

后续还可针对以上及界面进行进一步优化。