[OpenCV]未来视觉8-表格切割识别

2019-11-13

opencv pen

这段时间工作真心比较忙，而空余时间，都在在研究补全一些opencv技术的基础，近来在做一个OpenCV的表格扫描以及文字识别的功能，是一个非常好的应用机会。

估计很多人都会对大大鼎鼎的OpenCV有一定的听说。它很多应用在生活中，例如pdf扫描，身份证识别，车牌识别，人脸识别等，估计都听说过这些功能。这些功能都可以使用OpenCV来实验。
本篇是以表格识别为主题。
网上估计找不到一篇比这篇更加全面和整齐了。
这里是以方正的表格作为检测，排除那些不规则表格，例如圆角那些。
来说一下步骤吧（这是针对于真简单的基础表格，就都是方方正正的表格，无法识别一些特殊表格，例如超级课程表这些，但是第三方百度ocr可以识别出一定的范围的坐标值，你通过这些坐标值处理起来也是很费时间的。
一.边框检测
1.灰度二值化
2.findContours获得外边框
3.approxPolyDP多边形逼近，去除相近的点，而且只留下四边形
4.areaContours获得边框面积获得最大的面积的边框，并记录为长方形
二.表格检测
1.直线检测，包括横线和竖线，需要使用到腐蚀和膨胀
2.横竖线叠加形成表格图，得到表格坐标和横纵交点
三.表格分割
1.findContours获得边框树状图，注意获得边框树状图是无序的。
2.排除最大的边框和面积较少的边框
3.移除无效范围（横竖向排序）
4.通过边框来圈定一些感兴趣的区域。
四.优化表格识别
1.外框判定优化，对表格切割有优化效果(使用膨胀)
2.感兴趣区域边框优化，便于文字扫描(颜色优化)
3.返回边框有序按竖向或横向排列(对整理有优化)
4.区域列表大小调整
以上就是我这段时间考究过的一些过程和步骤。

以下的代码基本都可以使用SmartCropper的源码来说明，因为Smart_Cropper已经很好的实现了边框识别，这里先说的基本的识别，最基本的边框和过滤。而Smart_Cropper还加入了tensorflow机器学习，来进一步完善边框识别。

一.边框识别

1. 灰度二值化

```
1  //需要将图片从RGB格式，转为BGR格式（Opencv的Mat对象就是需要这样排列的）
2  Mat srcBitmapMat;
3  Mat bgrData(srcBitmapMat.rows,srcBitmapMat.cols,CV_8UC3);
4  cvtColor(srcBitmapMat,bgrData,COLOR_RGBA2BGR);
5
6  Mat Scanner::preprocessedImage(Mat &image, int cannyValue, int blurValue) {
7      Mat grayMat; //灰度化，将彩色图转为灰度图
8      cvtColor(image, grayMat, COLOR_BGR2GRAY);
```

```
    if (isHisEqual) {
        equalizeHist(grayMat, grayMat);
    }

    Mat blurMat; //高斯模糊降噪，Size()高斯内核，降噪的意思可以理解为去除干扰点。
    GaussianBlur(grayMat,blurMat,Size(blurValue,blurValue),0);

    Mat cannyMat; //Canny算法边缘检测
    Canny(blurMat,cannyMat,50,cannyValue,3);

    Mat thresholdMat; //图像二值化
    threshold(cannyMat,thresholdMat,0,255,THRESH_OTSU);
    return thresholdMat;
}

 //图片膨胀一定的范围，更好确认边框
Mat dilateMat;
Mat element = getStructuringElement(MORPH_RECT,Size(2,2));
dilate(scanImage,dilateMat,element);
```

以下是一些参照资料

GaussianBlur高斯模糊函数使用

Canny原理

threshold图像二值化函数的使用

膨胀dilate腐蚀erode原理

2.findContours获得外边框，通过面积排序来获取最外则边框

```
 //边框数据
 vector<vector<Point>> contours;
//提取边框
 /*
        * 通过 findContours 找轮廓
        *
        * 第一个参数，是输入图像，图像的格式是8位单通道的图像，并且被解析为二值图像（即图中
        * 第二个参数，是一个 MatOfPoint 数组，在多数实际的操作中即是STL vectors的STL vector，
        * 第三个参数，hierarchy，这个参数可以指定，也可以不指定。如果指定的话，输出hierarchy，
        * 第四个参数，轮廓的模式，将会告诉OpenCV你想用何种方式来对轮廓进行提取，有四个可选
        *     CV_RETR_EXTERNAL （0）： 表示只提取最外面的轮廓；
        *     CV_RETR_LIST （1）： 表示提取所有轮廓并将其放入列表；
        *     CV_RETR_CCOMP （2）:表示提取所有轮廓并将组织成一个两层结构，其中顶层轮廓是
        *     CV_RETR_TREE （3）： 表示提取所有轮廓并组织成轮廓嵌套的完整层级结构。
        * 第五个参数，见识方法，即轮廓如何呈现的方法，有三种可选的方法：
        *     CV_CHAIN_APPROX_NONE （1）： 将轮廓中的所有点的编码转换成点；
        *     CV_CHAIN_APPROX_SIMPLE （2）： 压缩水平、垂直和对角直线段，仅保留它们的端
        *     CV_CHAIN_APPROX_TC89_L1 （3）or CV_CHAIN_APPROX_TC89_KCOS（4）： 应
        * 第六个参数，偏移，可选，如果是定，那么返回的轮廓中的所有点均作指定量的偏移
        */
findContours(dilateMat,contours,RETR_EXTERNAL,CHAIN_APPROX_NONE);
//按面积排序
sort(contours.begin(),contours.end(),sortByArea);

//面积排序
static bool sortByArea(const vector<Point> &v1, const vector<Point> &v2){
    //fabs浮点绝对值
    //opencv中contourArea返回轮廓真实面积
```

```
}
```

3.approxPolyDP多边形逼近，去除相近的点，而且只留下四边形

```
 vector<Point> outDp;
//多边形逼近
  approxPolyDP(Mat(contour),outDp,0.01 * arc, true);
 //去掉相近的点
 vector<Point> selectedPoints = selectPoints(outDp);
 if(selectedPoints.size() != 4){
     //如果帅选出来后不是四边形
      continue;
 } else{
    //计算最外围矩形面积

}
```

4.areaContours获得边框面积获得最大的面积的边框，返回四个边框点，需要通过左上，右上，又下左下排序。

```
int widthMin = selectedPoints[0].x;
                int widthMax = selectedPoints[0].x;
                int heightMin = selectedPoints[0].y;
                int heightMax = selectedPoints[0].y;
                for (int k = 0; k < 4; ++k) {
                    if (selectedPoints[k].x < widthMin){
                        widthMin = selectedPoints[k].x;
                    }
                    if (selectedPoints[k].x > widthMax){
                        widthMax = selectedPoints[k].x;
                    }
                    if (selectedPoints[k].y < heightMin){
                        heightMin = selectedPoints[k].y;
                    }
                    if (selectedPoints[k].y > heightMax){
                        heightMax = selectedPoints[k].y;
                    }
                }

                //选择区域外围矩形面积
                int selectArea = (widthMax - widthMin) * (heightMax - heightMin);
                int imageArea = scanImage.cols * scanImage.rows;
                if (selectArea < (imageArea / 20)){
                    result.clear();
                    //帅选出来区域太小
                    continue;
                } else{
                    result = selectedPoints;
                    if (result.size() !=4 ){
                        Point2f p[4];
                        p[0] = Point2f(0,0);
```

```
                    result.push_back(p[0]);
                    result.push_back(p[1]);
                    result.push_back(p[2]);
                    result.push_back(p[3]);
                }
                for (Point &p:result) {
                    p.x *=resizeScale;
                    p.y *=resizeScale;
                }
                // 按左上，右上，右下，左下排序
                return sortPointClockwise(result);
            }
```

这里已经获取了最外层矩形的边框特征点，那么就可以进行裁剪。这里有一个讨巧的地方，表格一定是占据图片最大的地方。

这里还需要注意的是图片裁剪的时候，还要需要注意的是图像矫正，有可能边框是四边形，并不是一定是规则的矩形来的。

```
Mat transform = getPerspectiveTransform(srcTriangle, dstTriangle);
warpPerspective(srcBitmapMat, dstBitmapMat, transform, dstBitmapMat.size());
```

表格检测

对于表格检测，网上都有一些python都代码可以参照，但是基本只能翻译过来了，然后以下代码基本都是基于个人编写的，如果有有更优化的方法，大家也可以指导一下。

1.直线检测，包括横线和竖线，需要使用到腐蚀和膨胀

```
//灰度图
    Mat grayMat;
    cvtColor(srcMat,grayMat,CV_BGR2GRAY);
    //自动阀值二值化
    //~gray反色
    Mat thresholdMat;
    adaptiveThreshold(~grayMat,thresholdMat,255,CV_ADAPTIVE_THRESH_MEAN_C,THRESH_B

    //使用二值化后的图像来获取表格横纵的线
    Mat horizontalMat = thresholdMat.clone();
    Mat verticalMat = thresholdMat.clone();

    int scale = 20; //值越大，检测到的直线越多
    int horizontalSize = horizontalMat.cols / scale;

    //为了获取横向的表格线，设置腐蚀和膨胀的操作区域为一个比较大的横向直条
    Mat horizontalStructure = getStructuringElement(MORPH_RECT,Size(horizontalSize,1));

    //线腐蚀后膨胀
    erode(horizontalMat,horizontalMat,horizontalStructure,Point(-1,-1));
    dilate(horizontalMat,horizontalMat,horizontalStructure,Point(-1,-1));

    //获取纵向直线
    int verticalSize = verticalMat.rows/scale;
    Mat verticalStructure = getStructuringElement(MORPH_RECT,Size(1,verticalSize));
    erode(verticalMat,verticalMat,verticalStructure,Point(-1,-1));
    dilate(verticalMat,verticalMat,verticalStructure,Point(-1,-1));
```

2.横竖线叠加形成表格图，得到表格坐标和横纵交点

```
    //横纵交汇表格
    Mat mask = horizontalMat + verticalMat;

    //横纵焦点
    Mat joints;
    bitwise_and(horizontalMat,verticalMat,joints);
```

这里为何还要自己做横竖直接检测，检测交点，而不是直接进行角检测呢?
因为很有可能有些表格因为扫描问题，出现缺失角点，这种怎么办呢，只能通过横竖线补全，然后再检测横纵焦点了。

三.表格分割

关于表格分割，其作用可想而知，如果你想对一个表格，进行文字检测，你总不能直接检测吧，检测出来的文字混乱且无序。表格分割，才能确定区域坐标和对应的图片内容。

1.findContours获得边框树状图，注意获得边框树状图是无序的。

```
 vector<Vec4i> hierarchy;
 vector<vector<Point>> contours;
findContours(mask,contours,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_SIMPLE,Point(0,0))
```

2.排除最大的边框和面积较少的边框

```
//检测并移除面积最大的一项，就是外边框
    double maxArea = 0;
    int maxIndex =0;
    for (int i = 0; i < contours.size(); ++i) {
        double area = contourArea(contours[i]);
        if (area > maxArea) {
            maxArea = area;
            maxIndex = i;
        }
    }

    if (maxArea > 0){
        contours.erase(contours.begin() + maxIndex);
    }
//    string areaTxt;
    for(size_t i =0;i < contours.size();i++){

        //获取区域的面积，如果小于某个值就忽略，代表杂线不是表格
        double area = contourArea(contours[i]);
        if (area < 100){
            continue;
        }

        /*
            * approxPolyDP 函数用来逼近区域成为一个形状，true值表示产生的区域为闭合区域。比如
            *
            * MatOfPoint2f curve：像素点的数组数据。
```

```
            */
        approxPolyDP(Mat(contours[i]),contours_poly[i],3,true);
        boundRect[i] = boundingRect(contours_poly[i]);
        //将矩形画在原图
//        rectangle(srcMat,boundRect[i].tl(),boundRect[i].br(),Scalar(0,255,0),1,8,0);
    }
```

3.移除无效范围（横竖向排序）。

```
//移除无效范围
    vector<Rect>::iterator it;
    for(it=boundRect.begin();it!=boundRect.end();){
        if(it->width == 0 && it->height == 0)
            it= boundRect.erase(it);   //删除元素，返回值指向已删除元素的下一个位置
        else
            ++it;   //指向下一个位置
    }
```

4.通过边框来圈定一些感兴趣的区域。

上述步骤得到的边框，加入边框颜色为白色，方便之后做的处理（例如文字识别），这一步非别要。

```
vector<Rect> boundRect = cutTableRect(isVertical);

    for (int i = 0; i < boundRect.size(); ++i) {
        //将裁剪到的边框置成宽度为3的白色，方便用于之后的文字识别
        Mat roi = srcMat(boundRect[i]).clone();
        rectangle(roi,Point2f(0,0),Point2f(roi.cols,roi.rows),Scalar(255,255,255),3,8,0);

        //保存这片区域
        rois.push_back(roi);
    }
```

四.优化表格识别

1.外框判定优化，对表格切割有优化效果(使用膨胀)

（1）使用的膨胀和腐蚀去除干扰点。

（2）使用正视矫正

（3）通过添加直线来补全边界

2.感兴趣区域边框优化，便于文字扫描(颜色优化)

这个不知道怎么区分

3.返回边框有序按竖向或横向排列(对整理有优化)

（1）因为返回边框是无序的，需要自己手动排序

```
//横向排列器
static bool sortByHorizontal(const Rect &r1, const Rect &r2){
    //x坐标大的交换位置,加入误差值判断
    if (r1.y < r2.y && fabs(r1.y-r2.y)>=3){
        return true;
    } else if (r1.y == r2.y){
        return r1.x < r2.x && fabs(r1.x-r2.x)>=3;
    } else{
        return false;
    }
}

//纵向排列器
static bool sortByVertical(const Rect &r1, const Rect &r2){
    //x坐标大的交换位置
    if (r1.x < r2.x && fabs(r1.x-r2.x)>=3){
```

```
20      } else{
21          return false;
22      }
23  }
```

```
1   4.区域列表大小调整
```

了解了这些之后，你估计对OpenCV的简单运用已经有一定的了解了。

至于源码，只能联系本人了。

Opengl

Android组件化

码农家园