南工骁鹰视觉组培训

OpenCV入门

培训内容

- OpenCV概述
- OpenCV数据结构介绍
- 颜色空间转换
- 绘图函数
- 二值化
- 滤波
- 形态学操作
- 边缘检测
- 轮廓查找
- 霍夫变换

OpenCV概述

- OpenCV是一个基于Apache2.0许可(开源)发行的跨平台计算机视觉和机器学习软件库，可以运行在Linux、Windows、Android和Mac OS操作系统上。它轻量级而且高效——由一系列 C 函数和少量 C++ 类构成，同时提供了Python、Ruby、MATLAB等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。
- 编程语言
  - OpenCV基于C++实现，同时提供python, Ruby, Matlab等语言的接口。OpenCV-Python是OpenCV的Python API，结合了OpenCV C++API和Python语言的最佳特性。
- 跨平台
  - OpenCV可以在不同的系统平台上使用，包括Windows，Linux，OS，X，Android和iOS。基于CUDA和OpenCL的高速GPU操作接口也在积极开发中。
- 活跃的开发团队
  - 自从第一个预览版本于2000年公开以来，目前已更新至OpenCV4.6.0。
- 丰富的API
  - 完善的传统计算机视觉算法，涵盖主流的机器学习算法，同时添加了对深度学习的支持。

OpenCV基本数据类型

Point

- 成员变量
  - x
  - y
  - z　(Point3i,Point3f)　i:int,f:float
- 构造方法
  - Point_ ()
Point3_ ()
  - Point_ (_Tp _x, _Tp _y)
Point3_ (_Tp _x, _Tp _y, _Tp _z)
  - Point_ (const Point_ &pt)

Rect

- 成员变量
  - x(左上角顶点x)
  - y(左上角顶点y)
  - width
  - height
- 构造方法
  - Rect_ ( )
  - Rect_ (const Rect_ &r)
  - Rect_ (_Tp _x, _Tp _y, _Tp _width, _Tp _height)
  - Rect_ (const Point_< _Tp > &pt1, const Point_< _Tp > &pt2)
- 常用成员函数
  - bool empty ( ) const
  - Size_< _Tp > size ( ) const

Mat对象

- 构造方法
  - Mat()
  - Mat(int rows, int cols, int type)
  - Mat(Size size, int type)
- 赋值方法
  - Mat img = imread(const string &filename)
- 通道分离
  - split(Mat src, vector<Mat> channels);
  - Mat b = channels.at(0);//g, r同理
- 通道合并
  - merge(vector<Mat> channels, Mat dst)

Mat对象

- 常用成员函数
  - void copyTo(Mat mat,Mat mask)
  - Mat clone() //完全复制
  - void convertTo(Mat dst, int type) //转换类型
    - int type() //类型(CV_8UC3)
    - 8:8位(16,32,64)
    - U：无符号整型(S：有符号整型；F：单精度浮点型)
    - C3：3通道(C1，C2，C3，C4，C(n))
  - int channels() //通道数
  - int depth() //像素类型(CV_8S，8U，16U，16S，32S，32F，64F)
  - bool empty() //是否为空
  - Size size() //图像尺寸(width，height)

Mat对象

- Mat的内存结构

	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,...	n, m

	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

颜色空间、颜色识别、二值化

颜色空间转换

- cvtColor(Mat src, Mat dst, int code, int dstCn = 0)
- code: CV_BGR2GRAY //空间a 2(to) 空间b
- dstCn:转换后图像的通道数，如果为0则根据src和code来确定
- 颜色空间: BGR, GRAY, HSV, HLS…

课堂练习

- 用cvtColor将图片转换到不同颜色空间，并用imshow显示

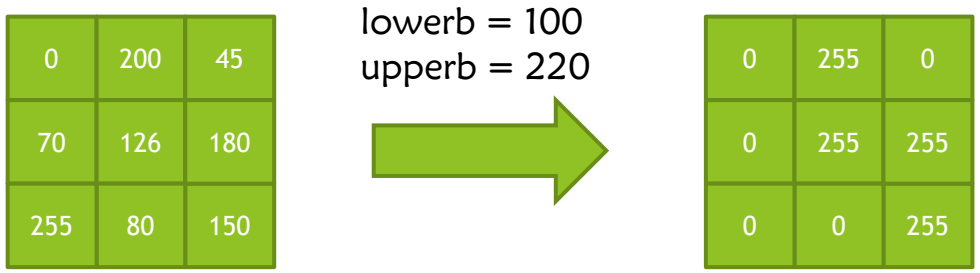
课堂练习

- 用cvtColor将图片转换到不同颜色空间，并用imshow显示

```cpp
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace cv;

#include <iostream>
#include <string>

int main() {
    Mat img, gray, hsv, hls;
    img = imread("../img/img1.jpg");
    if (img.empty()) {
        std::cout << "error" << std::endl;
        return -1;
    }
    cvtColor(img, gray, COLOR_BGR2GRAY);
    imshow("GRAY", gray);
    cvtColor(img, hsv, COLOR_BGR2HSV);
    imshow("HSV", hsv);
    cvtColor(img, hls, COLOR_BGR2HLS);
    imshow("HLS", hls);

    waitKey(0);

    return 0;
}
```

颜色识别

- 颜色范围筛选
  - inRange(Mat src, Scalar lowerb, Scalar upperb, Mat dst)
  - 若像素值在两个Scalar之间，则为255，否则为0
  - 中间两个参数也可是Mat类型
  - dst：输出8位单通道二值图像
  - 一般将RGB图像转换为HSV图像（HSV对光照的敏感度更低）



图像二值化

- Threshold(Mat src, Mat dst, double thresh, double maxval, int type)
  - type（阈值类型）
  - THRESH_BINARY//常用，大于thresh设为maxval，否则为0
  - THRESH_BINARY_INV//大于thresh设为0， 否则为maxval
- AdaptiveThreshold(Mat src, Mat dst, double maxval, int adaptiveMethod, int type, int blockSize, double C)
  - type可选前面2种，blockSize为适应算法取样邻域尺寸(3,5,7,…)
  - adaptiveMethod:
    - ADAPTIVE_THRESH_MEAN_C//邻域平均值减C
    - ADAPTIVE_THRESH_GAUSSIAN_C//邻域高斯均值减C

调参技巧

- createTrackbar(const string &trackbarname, const string &winname, int *value, int count, Trackbarcallback func, void *userdata = 0)
  - value为trackbar对应值的地址
  - count:value最大值
  - 回调函数: void func(int, void*)
  - void *userdata:传入数据

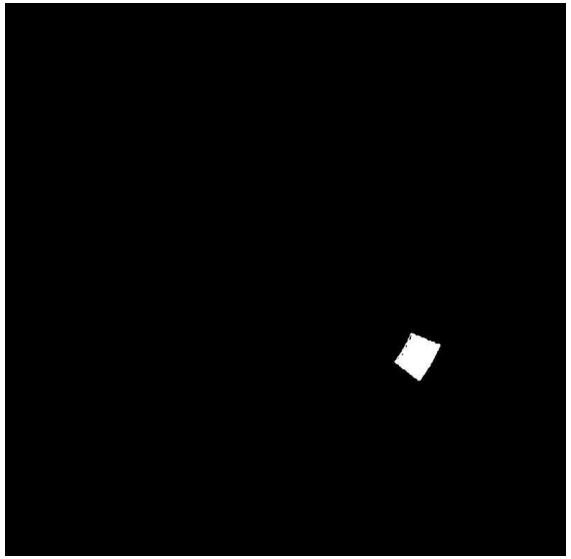
课堂练习

- 使用inrange实现在终端实时输出当前value值

```cpp
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace cv;

#include <iostream>
#include <string>

void callBack(int value, void *) {
    std::cout << "value:" << value << std::endl;
}

int main() {
    int value = 0;
    namedWindow("test", WINDOW_AUTOSIZE);
    createTrackbar("value", "test", &value, 100, callBack);
    waitKey(0);

    return 0;
}
```

课堂练习

- 用inrange筛选图片中的不同色彩，最好使用trackbar
  - (参考范围(170,240,84), (179,247,135))
- 通过二值化提取出图片中的字体



课堂练习

- 用inrange筛选图片img1中的不同色彩(可以使用trackbar)

```cpp
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

using namespace cv;

#include <iostream>
#include <string>

int hmin = 0;
int hmax = 255;
//饱和度
int smin = 0;
int smax = 255;
//亮度
int vmin = 0;
int vmax = 255;

//回调函数
void callBack(int, void *data) {
    Mat dst;
    Mat hsv = *(Mat *)data;
    GaussianBlur(hsv, dst, Size(5, 5), 3, 3);
    inRange(dst, Scalar(hmin, smin, vmin), Scalar(hmax, smax, vmax), dst);
    imshow("inrange", dst);
}
int main() {
    //输入图像
    Mat img, hsv;
    img = imread("../img/img1.jpg");
    if (img.empty()) {
        std::cout << "error" << std::endl;
        return EXIT_FAILURE;
    }
    std::vector<Mat> hsvSplit;
    cvtColor(img, hsv, COLOR_BGR2HSV);
    split(hsv, hsvSplit);
    equalizeHist(hsvSplit[2], hsvSplit[2]);
    merge(hsvSplit, hsv);
    //定义窗口
    namedWindow("inrange", WINDOW_AUTOSIZE);
    namedWindow("img", WINDOW_AUTOSIZE);
    //调节色相 H
    createTrackbar("hmin", "img", &hmin, 180, callBack, &hsv);
    createTrackbar("hmax", "img", &hmax, 180, callBack, &hsv);
    //调节饱和度 S
    createTrackbar("smin", "img", &smin, 255, callBack, &hsv);
    createTrackbar("smax", "img", &smax, 255, callBack, &hsv);
    //调节亮度 V
    createTrackbar("vmin", "img", &vmin, 255, callBack, &hsv);
    createTrackbar("vmax", "img", &vmax, 255, callBack, &hsv);
    imshow("img",img);
    waitKey(0);
    return 0;
}
```

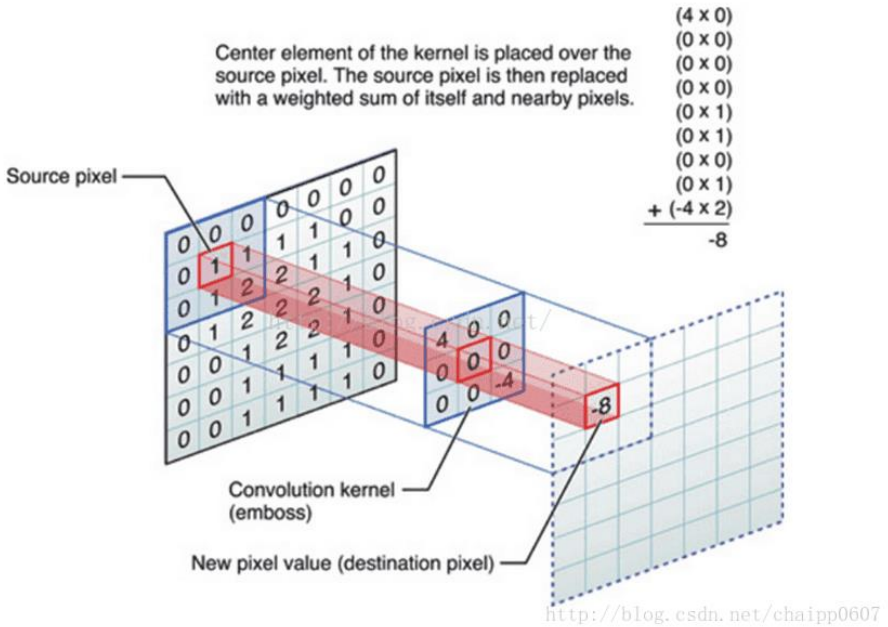
课堂练习

- 通过二值化提取出图片img2中的字体

```cpp
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>
#include <string>

using namespace cv;

void callBack(int thresh, void *data) {
    Mat gray = *(Mat *)data;
    Mat dst;
    threshold(gray, dst, thresh, 255, THRESH_BINARY_INV);
    imshow("gray", gray);
    imshow("dst", dst);
}

int main() {
    Mat img, gray,dst;
    int thresh;
    img = imread("../img/img2.jpg");
    if (img.empty()) {
        std::cout << "error" << std::endl;
        return -1;
    }
    namedWindow("dst", WINDOW_AUTOSIZE);
    namedWindow("gray", WINDOW_AUTOSIZE);
    cvtColor(img, gray, COLOR_BGR2GRAY);
    imshow("gray", gray);
    // createTrackbar("threshold", "gray", &thresh, 255, callBack, &gray);
    threshold(gray, dst, 115, 255, THRESH_BINARY_INV);
    imshow("dst", dst);
    waitKey(0);
    return 0;
}
```

卷积和滤波

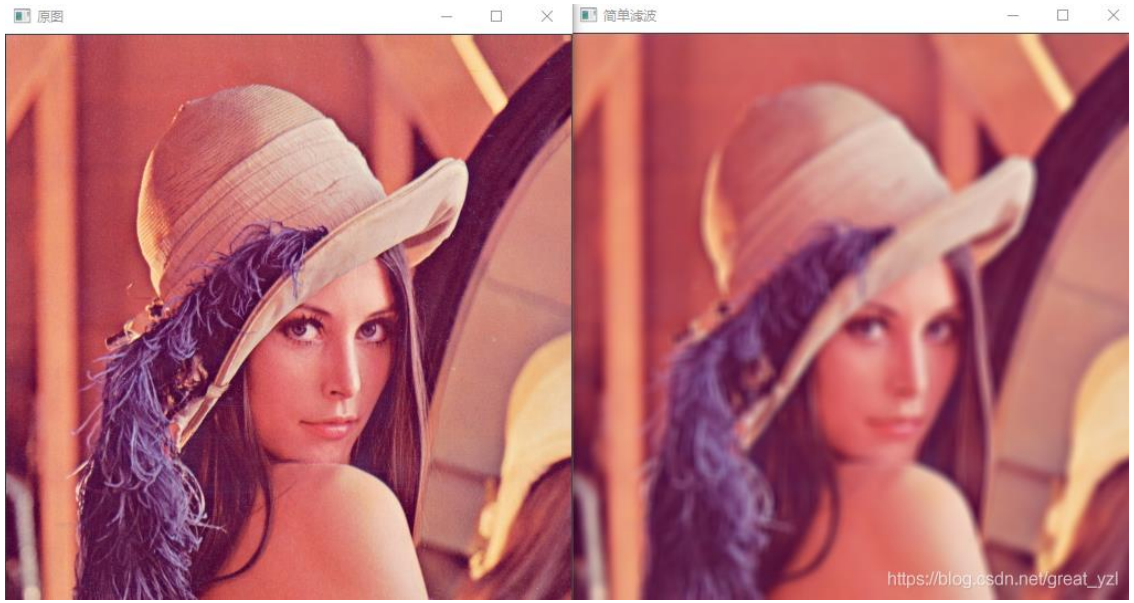
图像卷积



- 用一个小矩阵（卷积核kernel）覆盖在图像上方，与图像对应元素相乘并求和，结果赋到其中一个像素点（锚点，一般为kernel中点对应的地方）处。之后从左到右，从上到下移动，重复上述操作。
- kernel的大小一般为奇数
- 对于滤波后的结构，可能会出现负数或者大于255的数值，需要将其截断到0至255
- 若不对图像边缘进行处理（一般为扩充），目标图像的宽和高会变小

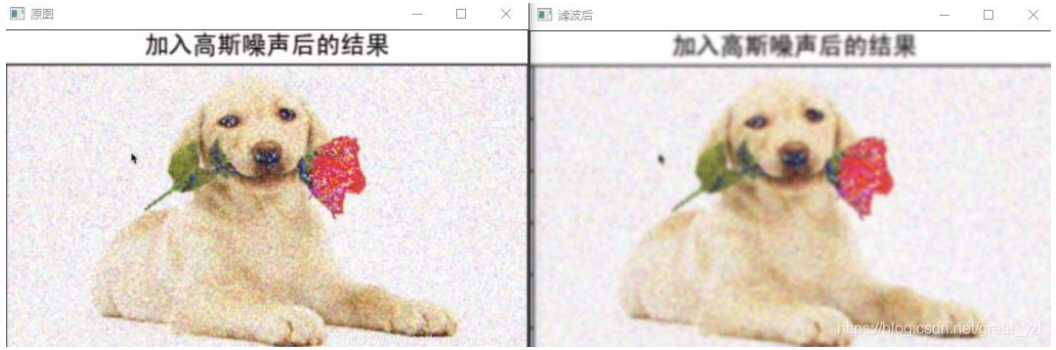
滤波方法

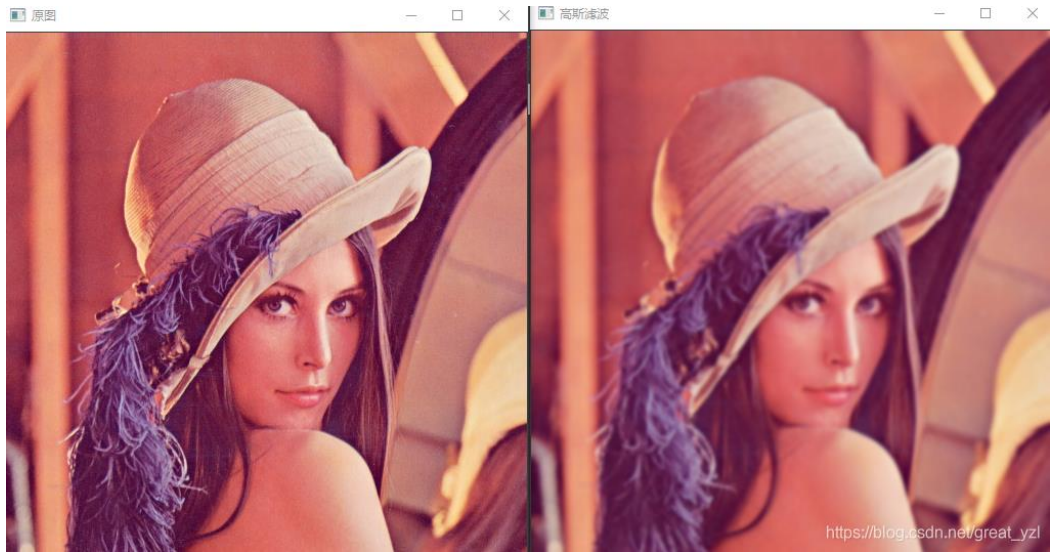
- 均值滤波
  - blur(Mat src, Mat dst, Size(xRadius, yRadius), Point(-1, -1))//size：kernel大小
  - kernel所有位置与图像对应点的值相乘相加后取平均值赋予锚点



滤波方法

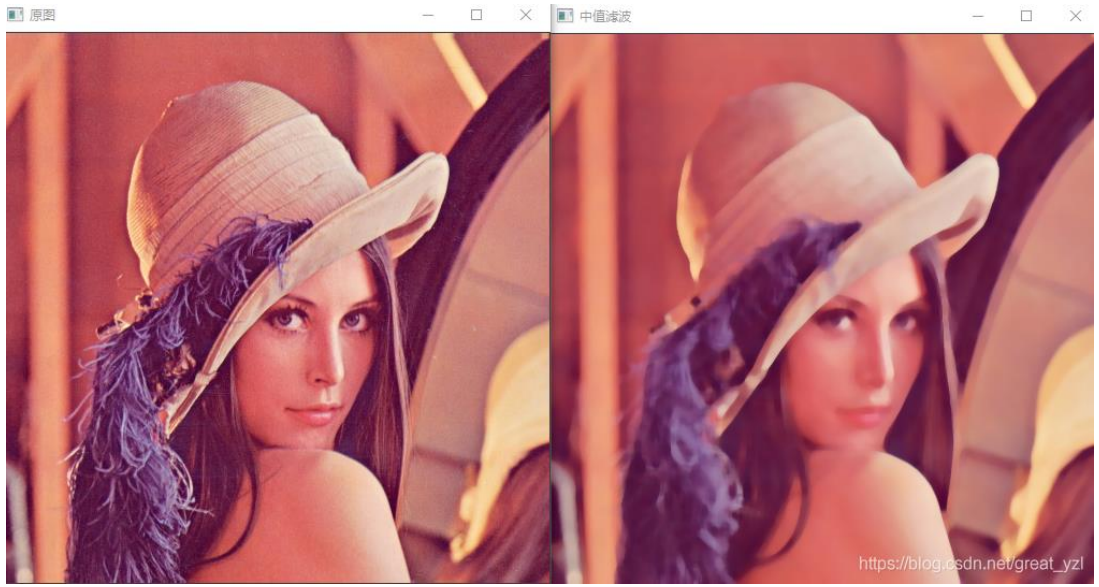
- 高斯滤波
  - GaussianBlur(Mat src, Mat dst, Size ksize, double sigmaX, double sigmaY = 0, int borderType=BORDER_DEFAULT)
  - 高斯加权均值赋予锚点





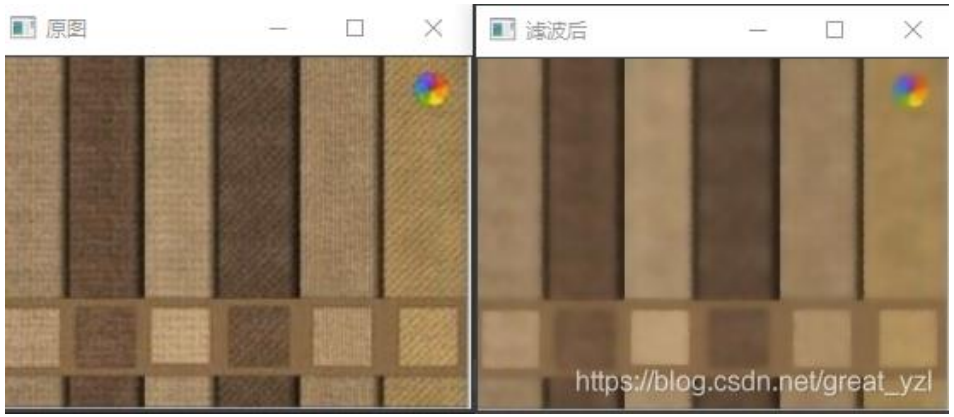
滤波方法

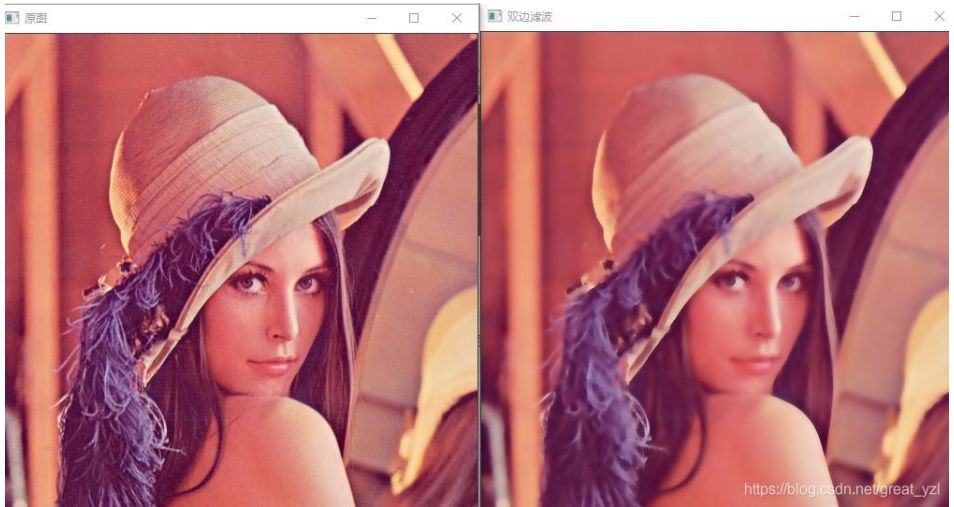
- 中值滤波
  - medianBlur(Mat src, Mat dst, Size ksize)
  - 用中值代替锚点，可有效抑制椒盐噪声



滤波方法

- 双边滤波
  - `bilateralFilter(Mat src, Mat dst, int d, double sigmaColor, double sigmaSpace, int borderType=4)`
  - d为邻域半径，sigmaColor为多少差值内的像素会被计算，sigmaSpace在d大于0时无效，d= -1时根据它来计算d的值

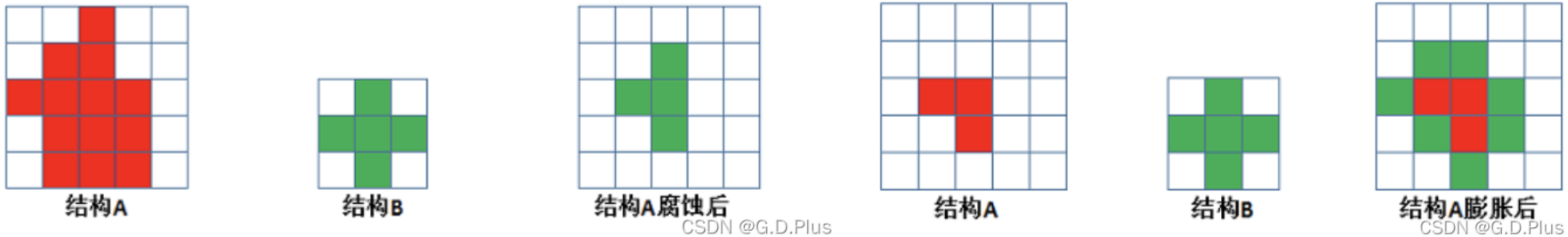




形态学操作

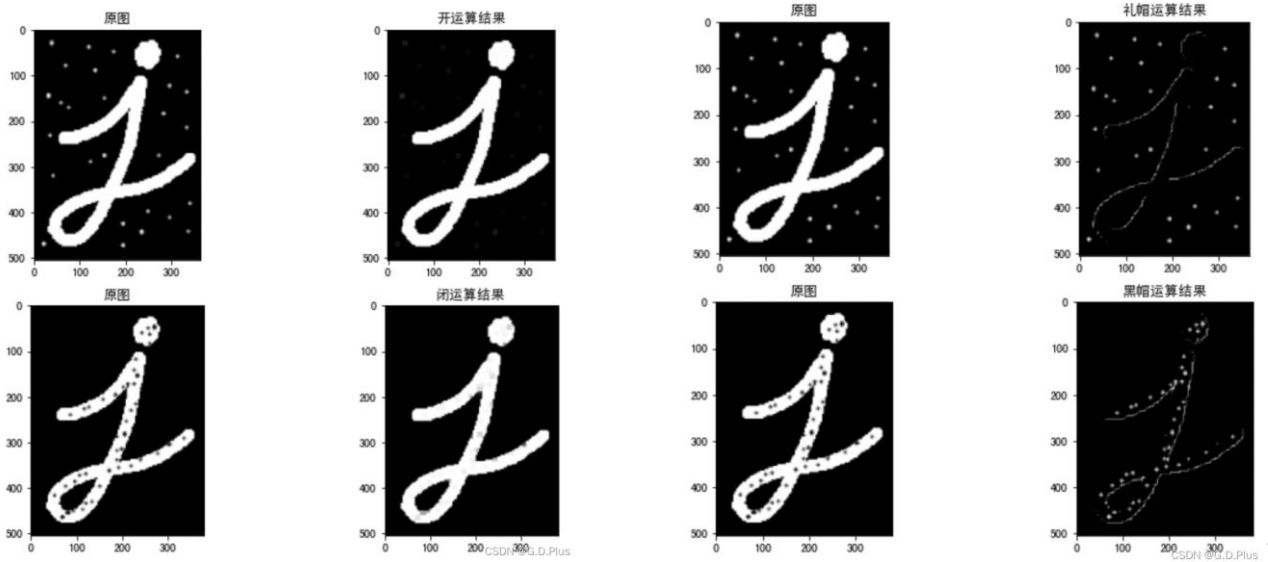
形态学操作

- 获取结构元素
  - getStructuringElement(int shape, Size ksize, Point anchor = Point(-1, -1))
  - shape: MORPH_RECT, MORPH_CROSS, MORPH_ELLIPSE
- 腐蚀
  - erode(Mat src, Mat dst, Mat kernel, Point anchor = Point(-1, -1), int iterations = 1)
- 膨胀
  - dilate(Mat src, Mat dst, Mat kernel, Point anchor = Point(-1, -1), int iterations = 1)



形态学操作

- 形态学变换
  - morphologyEx(Mat src, Mat dst, int op, Mat kernel, Point anchor = Point(-1, -1), int iterations = 1)
  - op:操作类型（背景为黑色）
    - MORPH_OPEN: 开操作（先腐蚀后膨胀），可去除小对象
    - MORPH_CLOSE: 闭操作（先膨胀后腐蚀），可填补小洞
    - MORPH_TOPHAT: 顶帽（原图与开操作的差值），保留小对象
    - MORPH_BLACKHAT: 黑帽（闭操作与原图的差值），保留小洞
    - MORPH_GRADIENT: 形态学梯度（膨胀减腐蚀）



课堂练习

- 通过形态学操作降噪

课堂练习

- 通过形态学操作降噪

```
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>
#include <string>

using namespace cv;

int main() {
    Mat img, gray,dst;
    int thresh;
    img = imread("../img/img2.jpg");
    if (img.empty()) {
        std::cout << "error" << std::endl;
        return -1;
    }
    cvtColor(img, gray, COLOR_BGR2GRAY);
    imshow("gray", gray);
    threshold(gray, dst, 115, 255, THRESH_BINARY_INV);
    Mat kernel = getStructuringElement(MORPH_CROSS,Size(9,9));
    erode(dst,dst,kernel);
    dilate(dst,dst,kernel);
    imshow("dst", dst);
    waitKey(0);
    return 0;
}
```

课后作业

作业一：提取音符

- 将图片中的五线谱去除，保留音符
  - 提交时间：10月14日24:00前
  - 提交方式：提交到邮箱1310813307@qq.com
  - 提交格式：将代码打包在压缩包中，压缩包命名为姓名_学号.zip

图形绘制

图形绘制

- void line(Mat& Point pt1, Point pt2, const Scalar& color, int thickness = 1, int lineType = LINE_8, int shift = 0)
  - color:颜色
  - thickness:直线的粗细
  - lineType: 画笔类型
  - shift:偏移量
- void rectangle(Mat& img, Rect rec, const Scalar& color, int thickness = 1,int lineType = LINE_8, int shift = 0)
- void circle(Mat& img, Point center, int radius, const Scalar& color, int thickness = 1,int lineType = LINE_8, int shift = 0)

图形绘制

- void ellipse(Mat& img, Point center, Size axes, double angle, double startAngle, double endAngle, const Scalar& color, int thickness = 1, int lineType = LINE_8, int shift = 0);
  - box:椭圆中心
  - axes:椭圆的尺寸
  - angle:椭圆的角度
  - startAngle：画椭圆的开始角度
  - endAngle:画椭圆的结束角度
- void fillConvexPoly(Mat& img, const Point* pts, int npts, const Scalar& color, int lineType = LINE_8, int shift = 0)
  - pts:多边形的顶点集合
  - npts:多边形的顶点个数

边缘检测

边缘检测

- Sobel算子：在x，y方向求导，得到2个方向的梯度图像
  - Sobel(Mat src, Mat dst, int ddepth, int dx, int dy, int ksize, double scale = 1, double delta = 0, int borderType = 4)

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

$$G = \sqrt{G_x^2 + G_y^2}$$

$$G = |G_x| + |G_y|$$

- Scharr算子(Sobel改进版)
  - Scharr(参数和Sobel一样)

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} \quad G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

- 拉普拉斯(Laplacian)算子
  - 原理：二阶导为0处变化最大
  - Laplacian(Mat src, Mat dst, int ddepth, int ksize, double scale = 1, double delta = 0, int borderType = 4)

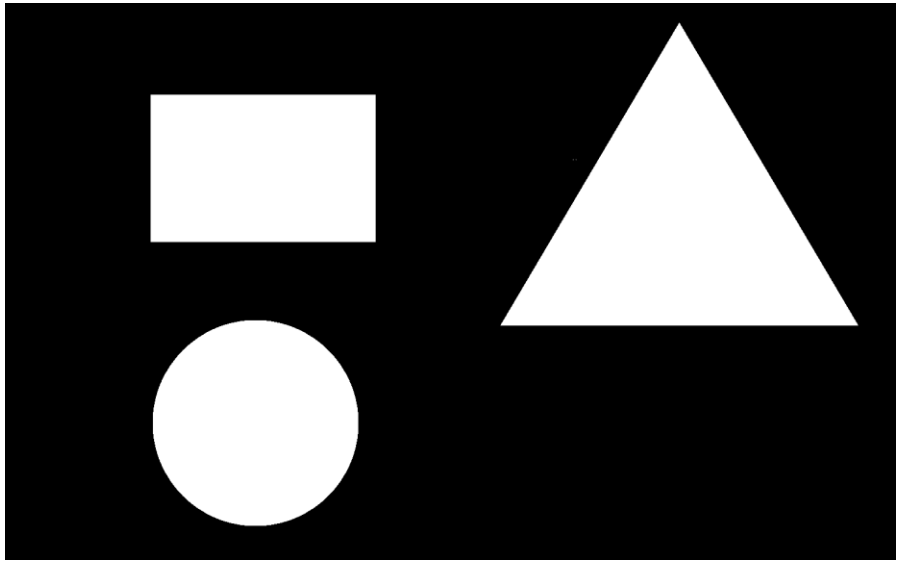
$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

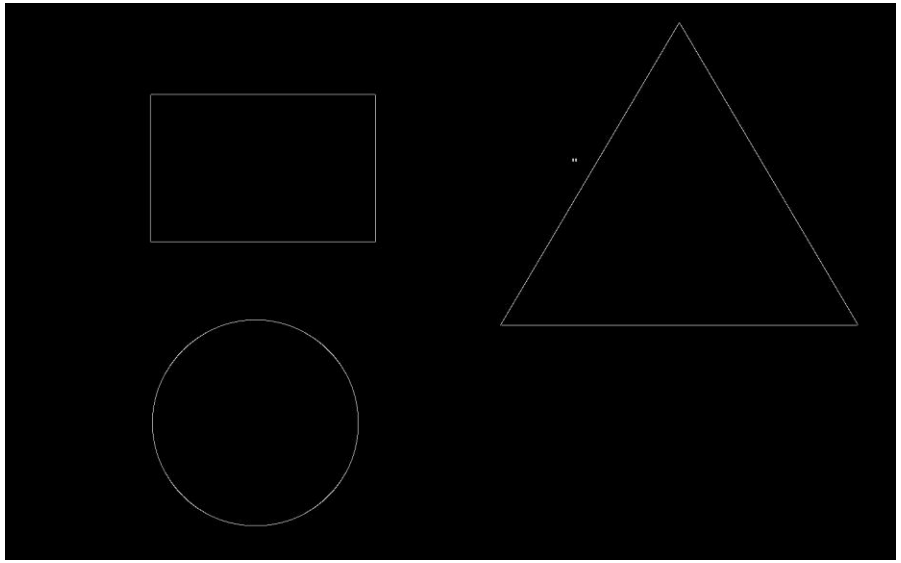
边缘检测

- Canny边缘检测
  - Canny(Mat src, Mat dst, double threshold1, double threshold2, int aptertureSize = 3, bool L2gradient = false)//aptertureSize: Sobel算子大小，L2gradient：梯度计算方式
  - 预处理:cvtColor转灰度（输入图像为8位单通道）
  - 算法内部流程：
    - 高斯模糊(OpenCV中不包含这一步)
    - 计算梯度-Sobel
    - 非最大信号抑制
    - 高低阈值处理：高于T2保留，低于T1丢弃，高于T1且与高于T2的部分连接则保留

课堂练习

- 使用canny得到图像中的边缘





课堂练习

- 使用canny得到图像中的边缘

```
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>
#include <string>
using namespace cv;

int main() {
    Mat img, dst;
    std::vector<std::vector<Point>> contours;
    std::vector<Vec4i> hierarchy;
    img = imread("../img/img4.jpg");
    if (img.empty()) {
        std::cout << "error" << std::endl;
        return -1;
    }
    Canny(img, dst, 20, 200);
    imshow("canny", dst);
```

轮廓寻找及筛选

轮廓检测

- 寻找轮廓
  - findContours(Mat binImg, vector<vector<Point>> contours, vector<Vec4i> hierarchy, int mode, int method, Point offset = Point())
  - binImg：输入二值化图像
  - contours：轮廓点集
  - hierarchy：拓扑结构
  - mode：轮廓返回模式
    - RETR_TREE：返回所有轮廓和拓扑结构
    - RETR_EXTERNAL：只返回最外层轮廓
    - RETR_LIST：返回所有轮廓但不建立拓扑结构
  - method：轮廓发现方法，常用CHAIN_APPROX_SIMPLE
- 画轮廓
  - drawContours(binImg, contours, contourIdx, color, thickness, lineType, hierarchy, maxlevel, offset)

轮廓检测

- 连通域检测
  - int connectedComponents(Mat image, Mat labels, int connectivity, int ltype, int ccltype)
  - 返回连通域数N，其中第0个是背景，即一共有N-1个轮廓（1~N-1）
  - image：输入8位单通道图像
  - labels：输出标记图，size和image一样，连通域所在像素值为该连通域的ID（0-N）
  - connectivity：8或4，一般取8
  - ltype：输出label的type，可选CV_32S（默认）或CV_16U
  - ccltype：检测算法（可选）

轮廓检测

- 连通域检测(带参数)
  - int connectedComponentsWithStats(Mat image, Mat labels, Mat stats, Mat centroids, int connectivity, int ltype, int ccltype)
  - stats：N*5，CV_32S的矩阵，表示每个连通域的外接矩形与面积

Enumerator	
CC_STAT_LEFT Python: cv.CC_STAT_LEFT	The leftmost (x) coordinate which is the inclusive start of the bounding box in the horizontal direction
CC_STAT_TOP Python: cv.CC_STAT_TOP	The topmost (y) coordinate which is the inclusive start of the bounding box in the vertical direction.
CC_STAT_WIDTH Python: cv.CC_STAT_WIDTH	The horizontal size of the bounding box.
CC_STAT_HEIGHT Python: cv.CC_STAT_HEIGHT	The vertical size of the bounding box.
CC_STAT_AREA Python: cv.CC_STAT_AREA	The total area (in pixels) of the connected component.

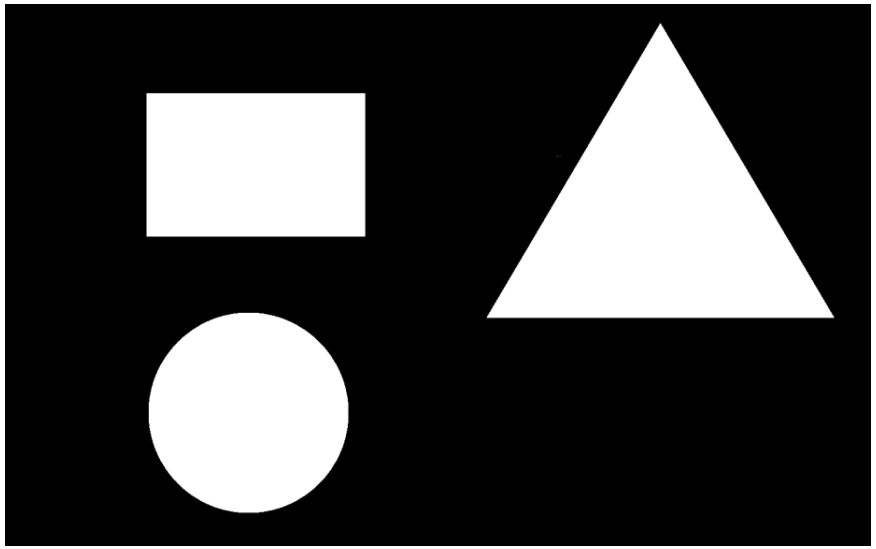
  - stats.at<int>(i, CC_STAT_LEFT) //第i个连通域左上角的x坐标
  - centroids：N*2，CV_32S的矩阵，表示每个连通域的质心

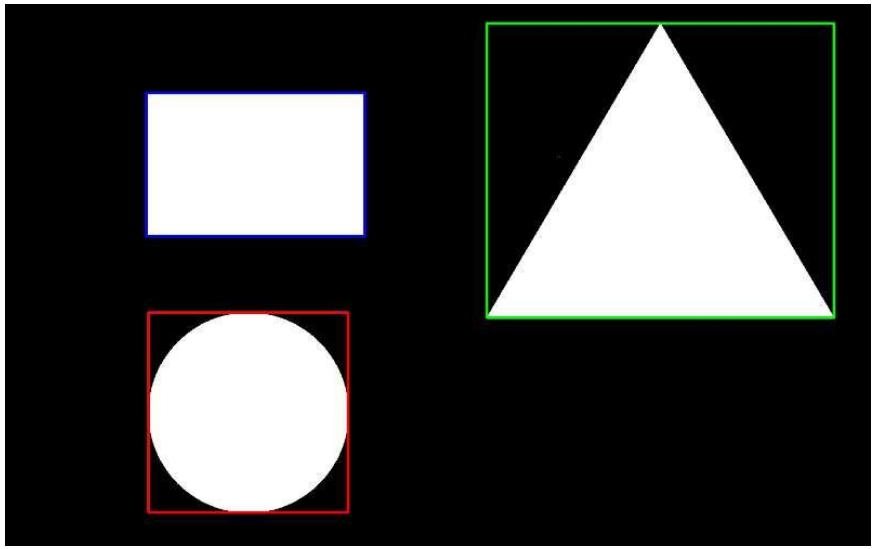
轮廓筛选

- 目标过滤
  - 基于面积
    - double contourArea(vector<Point> contour, bool oriented = false)
    - 利用connectedComponentsWithStat获得的stat中的面积信息
  - 基于宽高比……
- 外接矩形
  - RotatedRect minAreaRect(vector<Point> points)
    - 绘制方法：直线连接
    - RotatedRect rRect; Point vertices[4]; rRect.points(vertices);
    - for(int i = 0; i < 4; i++) line(image, vertices[i], vertices[(i+1)%4], color,…)
  - Rect boundingRect(vector<Point> points)

课堂练习

- 筛选不同轮廓，矩形用蓝色框出，圆形用红色框出，三角形用绿色框出





课堂练习

- 筛选不同轮廓，矩形用蓝色框出，圆形用红色框出，三角形用绿色框出

```cpp
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>
#include <string>
using namespace cv;

int main() {
    Mat img, dst;
    std::vector<std::vector<Point>> contours;
    std::vector<Vec4i> hierarchy;
    img = imread("../img/img4.jpg");
    if (img.empty()) {
        std::cout << "error" << std::endl;
        return -1;
    }
    Canny(img, dst, 20, 200);
    imshow("canny", dst);
    findContours(dst, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
    for (int i = 0; i < contours.size(); ++i) {
        Rect fit = boundingRect(contours[i]);
        double area_rect = float(fit.width) * float(fit.height);
        double scale_rect = float(fit.width) / fit.height;
        std::cout << area_rect << "----" << scale_rect << std::endl;
        if (area_rect > 80000 && area_rect < 150000 && scale_rect > 1.2 && scale_rect < 1.8) {
            rectangle(img, fit, Scalar(255, 0, 0), 4);
        } else if (area_rect > 200000 && area_rect < 400000 && scale_rect > 0.8 && scale_rect < 1.2) {
            rectangle(img, fit, Scalar(0, 255, 0), 4);
        } else if (area_rect > 80000 && area_rect < 150000 && scale_rect > 0.9 && scale_rect < 1.1) {
            rectangle(img, fit, Scalar(0, 0, 255), 4);
        }
    }
    imshow("img", img);
    waitKey(0);

    return 0;
}
```

霍夫变换

霍夫变换

- 霍夫直线变换
  - HoughLinesP(Mat src, vector<Vec4i> lines, double rho, double theta, int threshold, double minLineLength = 0, double maxLineGap = 0)
    - src：输入8位单通道图像
    - lines：vector<Vec4i> Vec4i (x1, y1, x2, y2)
    - rho：像素扫描精度，一般取1
    - theta：角度的精度，一般取CV_PI/180，即1度
    - threshold：霍夫空间中交点数阈值，一般取10（越大越精确）
    - minLineLength：最小直线长度
    - maxLineGap：最大直线间隔（大于这个判断为2条直线）

霍夫变换

霍夫圆变换

- HoughCircles(Mat src, vector<Vec3f> circles, int method, double dp, double minDist, double param1, double param2, int minRadius, int maxRadius)
  - src：输入8位单通道图像
  - circles：vector<Vec3f> Vec3f(a,b,r)
  - method：检测方法，取HOUGH_GRADIENT
  - dp：累加器大小与原图大小的比值，一般取1
  - minDist：区分同心圆的最小距离
  - param1：canny的高阈值（低阈值为一半）
  - param2：中心点累加阈值（霍夫空间交点数），一般30到50
  - minRadius，maxRadius：最小和最大半径

课后作业

- **作业二：识别金矿**
  - 给定一张图片，利用OpenCV识别图中的金矿石，并且用矩形框选出，保存为dst.jpg
  - 提交时间：10月14日24:00前
  - 提交方式：提交到邮箱1310813307@qq.com
  - 提交格式：将代码以及dst.jpg打包在压缩包中，压缩包命名为姓名_学号.zip