《数字图像处理》课程设计

题 目：图像处理综合应用

目录

[1、设计内容及要求： 1](#_Toc7118395)

[2、实验内容： 1](#_Toc7118396)

[实验一：RGB图像底片效果 1](#_Toc7118397)

[实验二：灰度增强 2](#_Toc7118398)

[2.1 图片的灰度化： 2](#_Toc7118399)

[2.2对灰度图像进行二值化处理 3](#_Toc7118400)

[实验三：边缘检测 5](#_Toc7118401)

[实验四： 浮雕效果 7](#_Toc7118402)

[实验五：颜色翻转 8](#_Toc7118403)

[实验六、爬取某高校页面的验证码，进行验证码识别 10](#_Toc7118404)

# 1、设计内容及要求：

（1）、独立设计方案，实现对图像的五种以上处理（比如：底片化效果、灰度增强、图像复原和去噪、图像边缘检测、木刻效果等等）。

# 2、实验内容：

## 实验一：RGB图像底片效果

为了加快处理速度，在图像处理算法中，往往需要把彩色图像转换为灰度图像。

**0x00. 灰度图**

灰度数字图像是每个像素只有一个采样颜色的图像，这类图像通常显示为从最暗黑色到最亮的白色的灰度。

灰度图像与黑白图像不同，在计算机图像领域中黑白图像只有黑白两种颜色，灰度图像在黑色与白色之间还有许多级的颜色深度。

在RGB模型中，如果R=G=B时，则彩色表示一种灰度颜色，其中R=G=B的值叫灰度值。

**0x01. 灰度化的方法**

分量法

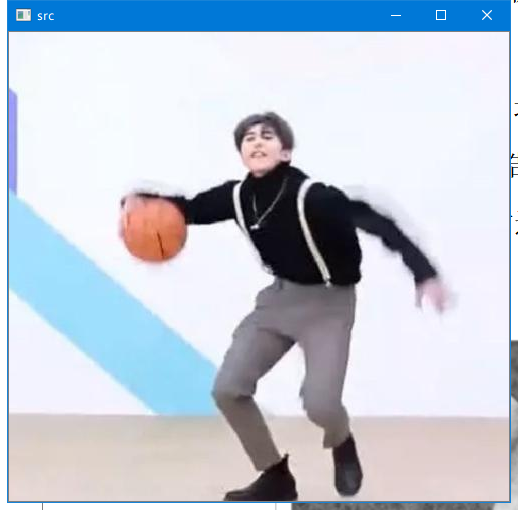
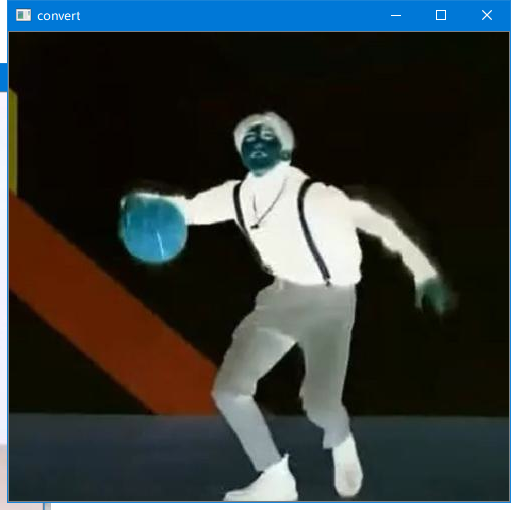
将彩色图像中的三分量的亮度作为三个灰度图像的灰度值，可根据应用需要选取一种灰度图像。

F1(i,j) = R(i,j)   
F2(i,j) = G(i,j)   
F3(i,j) = B(i,j)

代码实现：

1. # coding:utf-8
3. '''''
4. RGB图像底片效果
5. '''
7. **import** cv2
9. src = cv2.imread(r'img/cxk.jpg')
11. dist = 255 - src
13. cv2.imshow('src', src)
14. cv2.imshow('convert', dist)
16. cv2.waitKey()
17. cv2.destroyAllWindows()

运行结果：

## 实验二：灰度增强

一张图片是由像素点矩阵构成，我们对图片进行操作即为对图片的像素点矩阵进行操作。我们只要在这个像素点矩阵中找到这个像素点的位置，比如第x行，第y列，所以这个像素点在这个像素点矩阵中的位置就可以表示成（x，y）,因为一个像素点的颜色由红、绿、蓝三个颜色变量表示(R,G,B)，所以我们通过给这三个变量赋值，来改变这个像素点的颜色。

### 2.1 图片的灰度化：

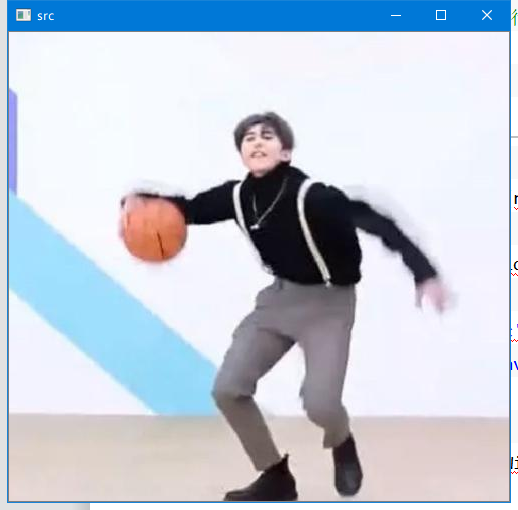
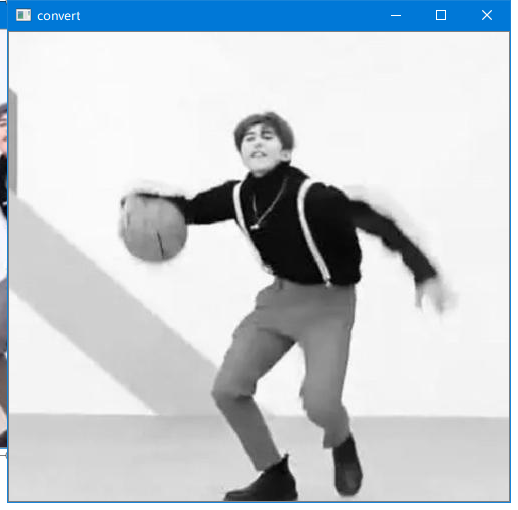
灰度化：将一个像素点的三个颜色变量相等，R=G=B，此时该值称为灰度值

直接调用opencv中的函数，读入的图片可以与代码文件放在一起这样可以省略输入图片路径。

代码实现：

1. # coding:utf-8
3. '''''
4. 读入正常图像并进行灰度化处理
5. '''
7. **import** cv2
8. #读入原始图像
9. img=cv2.imread(r'img/cxk.jpg')
10. #灰度化处理
11. gray=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)
13. cv2.imshow('src', img)
14. cv2.imshow('convert', gray)
16. cv2.waitKey()
17. cv2.destroyAllWindows()

运行效果：

### 2.2对灰度图像进行二值化处理

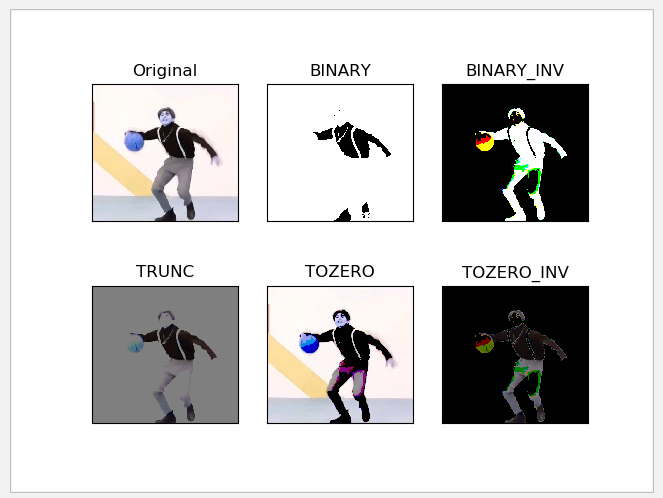
二值化处理：将一个像素点的值突出为0,255，使得图片呈现黑白两种颜色。在灰度图像中像素值在0~255,二值化后图像中像素值为0或255。

CV\_THRESH\_BINARY, //表示如果当前像素点的灰度值大于阈值则将输出图像的对应位置像素值置为255，否则为0

代码实现：

1. # coding:utf-8
3. '''''
4. 读入正常图像并进行灰度化处理
5. '''
7. **import** cv2
8. **import** matplotlib.pyplot as plt
9. #读入原始图像
10. img=cv2.imread(r'img/cxk.jpg')
11. #灰度化处理
12. gray=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)
14. #二值化处理
15. ret,thresh1=cv2.threshold(gray,50,255,cv2.THRESH\_BINARY)
17. ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV)
18. ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH\_TRUNC)
19. ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO)
20. ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO\_INV)
21. titles = ['Original','BINARY','BINARY\_INV','TRUNC','TOZERO','TOZERO\_INV']
22. images = [img,thresh1,thresh2,thresh3,thresh4,thresh5]
23. **for** i **in** range(6):
24. plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
25. plt.title(titles[i])
26. plt.xticks([]),plt.yticks([])
27. plt.show()
28. cv2.waitKey()
29. cv2.destroyAllWindows()

运行效果：



## 实验三：边缘检测

边缘检测是图像处理和计算机视觉中的基本问题，边缘检测的目的是标识数字图像中亮度变化明显的点。图像属性中的显著变化通常反映了属性的重要事件和变化。边缘检测是特征提取中的一个研究领域。

图像边缘检测大幅度地减少了数据量，并且剔除了可以认为不相关的信息，保留了图像重要的结构属性。有许多方法用于边缘检测，它们的绝大部分可以划分为两类：基于查找一类和基于零穿越的一类。基于查找的方法通过寻找图像一阶导数中的最大值和最小值来检测边界，通常是将边界定位在梯度最大的方向。基于零穿越的方法通过寻找图像二阶导数零穿越来寻找边界，通常是Laplacian过零点或者非线性差分表示的过零点。

如果将边缘认为是一定数量点亮度发生变化的地方，那么边缘检测大体上就是计算这个亮度变化的导数。

（一）检测方法

边缘检测的方法大致可分为两类：基于搜索和基于零交叉

基于搜索的边缘检测方法首先计算边缘强度，通常用一阶导数表示，例如梯度模，然后，用计算估计边缘的局部方向，通常采用梯度的方向，并利用此方向找到局部梯度模的最大值。

基于零交叉的方法找到由图像得到的二阶导数的零交叉点来定位边缘，通常用拉普拉斯算子或非线性微分方程的零交叉点。

滤波作为边缘检测的预处理通常是必要的，通常采用高斯滤波。

（二）Sobel边缘检测算子

Sobel边缘检测算法比较简单，实际应用中效率比canny边缘检测效率要高，但是边缘不如Canny检测的准确，但是很多实际应用的场合，sobel边缘却是首选，Sobel算子是高斯平滑与微分操作的结合体，所以其抗噪声能力很强，用途较多。尤其是效率要求较高，而对细纹理不太关系的时候。算子的模板为：

Sobel算子是一种带有方向的过滤器，openCV中Sobel算子的函数为cv2.Sobel ()。

1. Sobel\_x\_or\_y = cv2.Sobel(src, ddepth, dx, dy, dst, ksize, scale, delta, borderType)

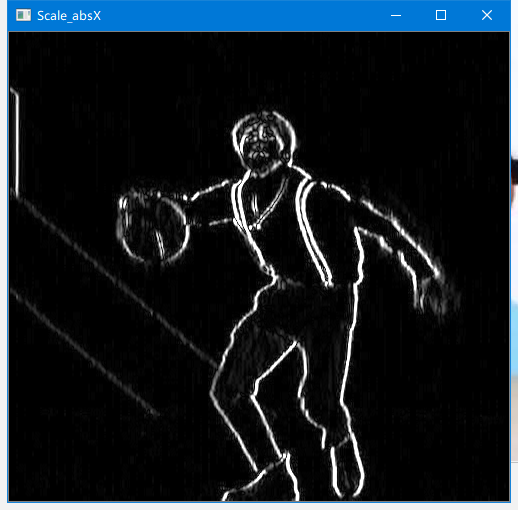
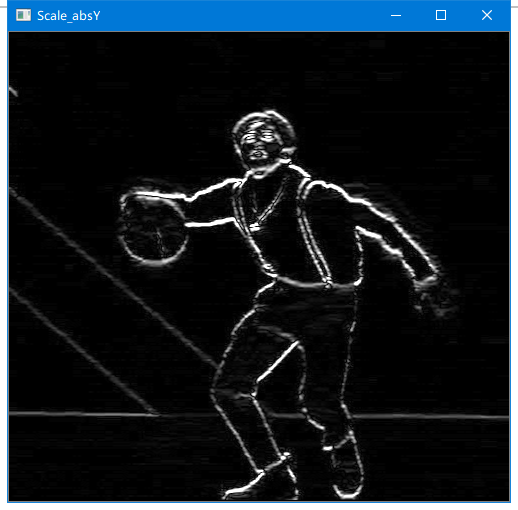
dst及dst之后的参数都是可选参数。

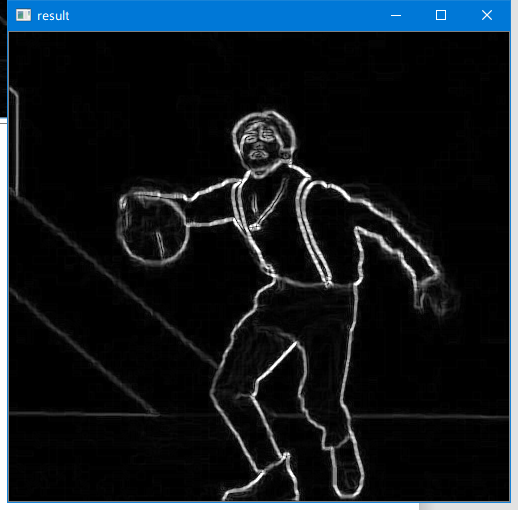
第一个参数是传入的图像，第二个参数是图像的深度，dx和dy指的是求导的阶数，0表示这个方向上没有求导，所填的数一般为0、1、2。ksize是Sobel算子的大小，即卷积核的大小，必须为奇数1、3、5、7。如果ksize=-1，就演变成为3x3的Scharr算子，scale是缩放导数的比例常数，默认情况为没有伸缩系数。borderType是判断图像边界的模式，这个参数默认值为cv2.BORDER\_DEFAULT。

代码实现：

1. # coding:utf-8
3. '''''
4. 边缘检测
5. '''
7. **import** cv2
8. # Sobel边缘检测算子
9. img = cv2.imread('img/cxk.jpg', 0)
10. x = cv2.Sobel(img, cv2.CV\_16S, 1, 0)
11. y = cv2.Sobel(img, cv2.CV\_16S, 0, 1)
12. # cv2.convertScaleAbs(src[, dst[, alpha[, beta]]])
13. # 可选参数alpha是伸缩系数，beta是加到结果上的一个值，结果返回uint类型的图像
14. Scale\_absX = cv2.convertScaleAbs(x)  # convert 转换  scale 缩放
15. Scale\_absY = cv2.convertScaleAbs(y)
16. result = cv2.addWeighted(Scale\_absX, 0.5, Scale\_absY, 0.5, 0)
17. cv2.imshow('img', img)
18. cv2.imshow('Scale\_absX', Scale\_absX)
19. cv2.imshow('Scale\_absY', Scale\_absY)
20. cv2.imshow('result', result)
21. cv2.waitKey(0)
22. cv2.destroyAllWindows()

运行效果：



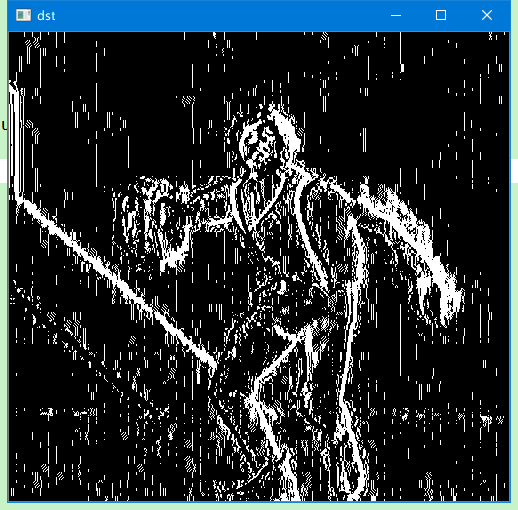
## 实验四： 浮雕效果

浮雕/雕刻算法实质是很简单地，即：对图像的每一个点进行卷积处理。假设原图像为X,处理后的图像为Y；浮雕算法核kernel矩阵定义为：[1 , 0, 0 ; 0, 0,  0; 0, 0, -1]. 那么，对于坐标为(i,j)点，浮雕效果图的算法为Y(i,j) = X(i+1,j+1)-X(i-1,j-1) + 128。当然，X,Y的取值均在0~255之间。雕刻算法核kernel矩阵定义为：[1, 0;  0, -1]。那么，对于坐标为(i,j)点，其浮雕效果图的算法为Y(i,j) = X(i,j) - X(i-1,j-1) + 128。当然，X,Y的取值均在0~255之间。

代码展示：

1. # coding:utf-8
3. '''''
4. 浮雕效果
5. '''
7. **import** cv2
8. **import** numpy as np
9. gray=cv2.imread('img/cxk.jpg',0)
10. imgInfo=gray.shape
11. height=imgInfo[0]
12. weight=imgInfo[1]
14. dst=np.zeros((height,weight,1),np.uint8)
15. **for** i **in** range(0,height):
16. **for** j **in** range(0,weight-1):
17. gray0=gray[i,j]
18. gray1=gray[i,j+1]
19. newp=gray0-gray1+150
20. **if** newp>255:
21. newp=255
22. **else**:
23. newp=0
24. dst[i,j]=newp
25. cv2.imshow('dst',dst)
26. cv2.waitKey(0)

运行效果：



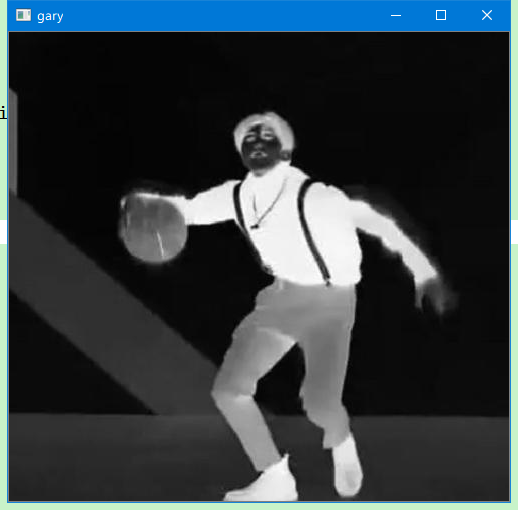
## 实验五：颜色翻转

灰度图像素值为0到255，若当前的像素值为i,翻转过后为255-i

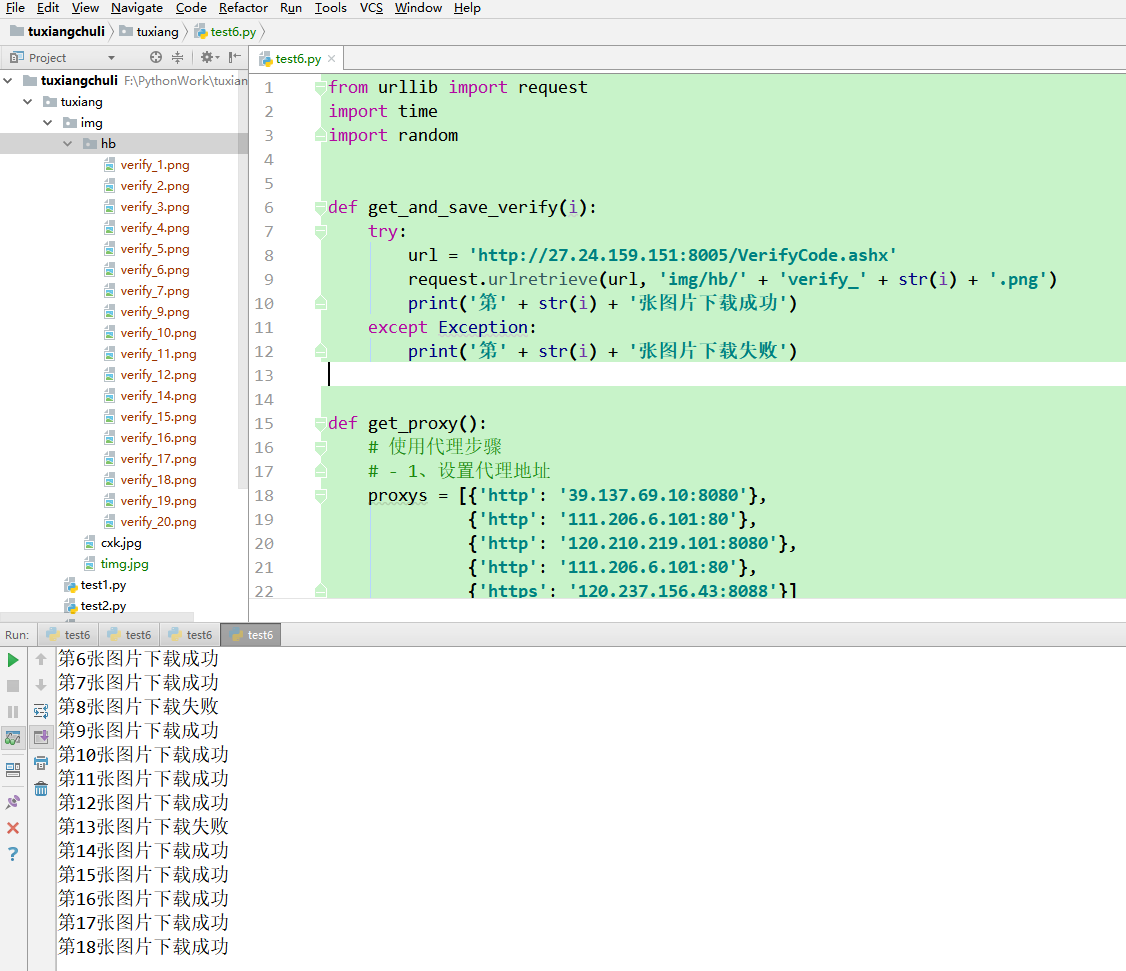
代码实现：

1. **import** cv2
2. **import** numpy as np
3. #读取彩色原图
4. src=cv2.imread('img/cxk.jpg',1)
5. gray=cv2.cvtColor(src,cv2.COLOR\_BGR2GRAY)
6. img\_info=src.shape
7. image\_height=img\_info[0]
8. image\_weight=img\_info[1]
9. dst=np.zeros((image\_height,image\_weight,1),np.uint8)
10. **for** i **in** range(image\_height):
11. **for** j **in** range(image\_weight):
12. grayPixel=gray[i][j]
13. dst[i][j]=255-grayPixel
14. cv2.imshow('gary',dst)
15. cv2.waitKey(0)

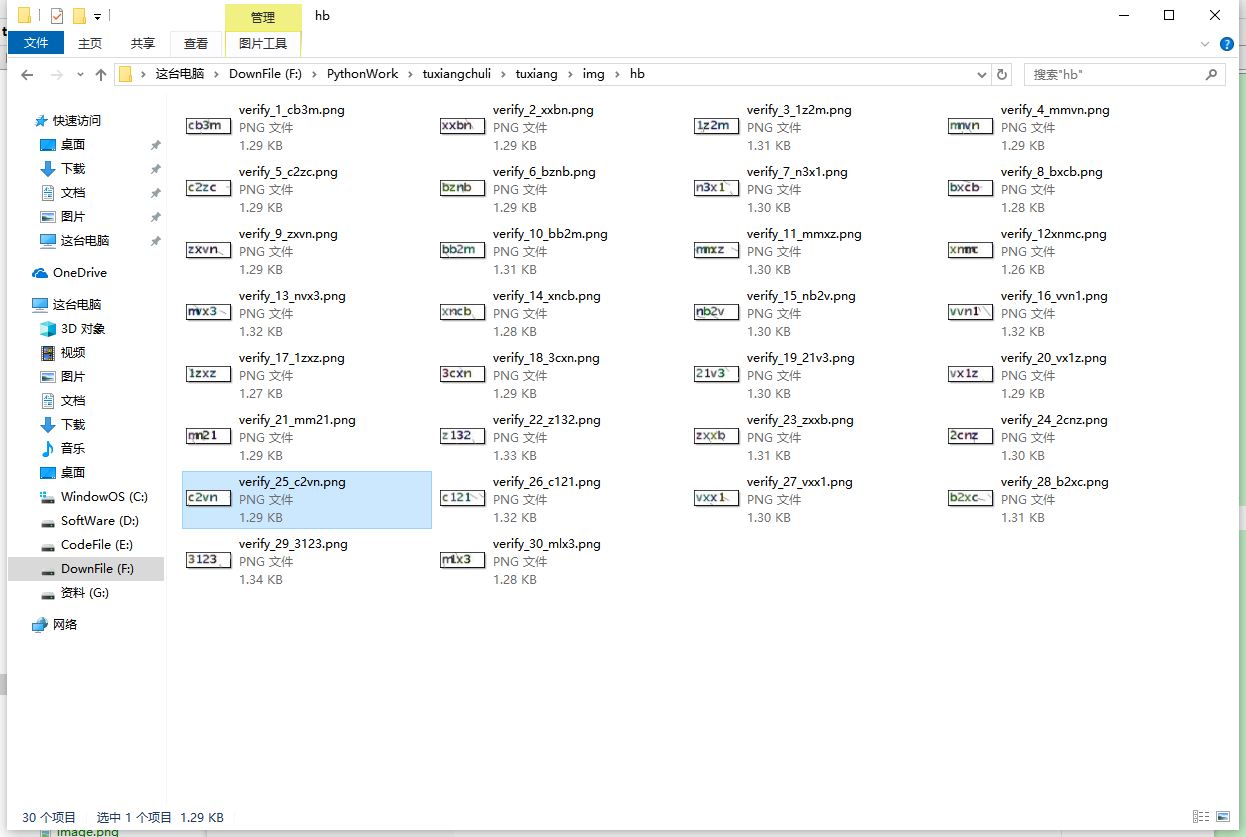
运行效果：

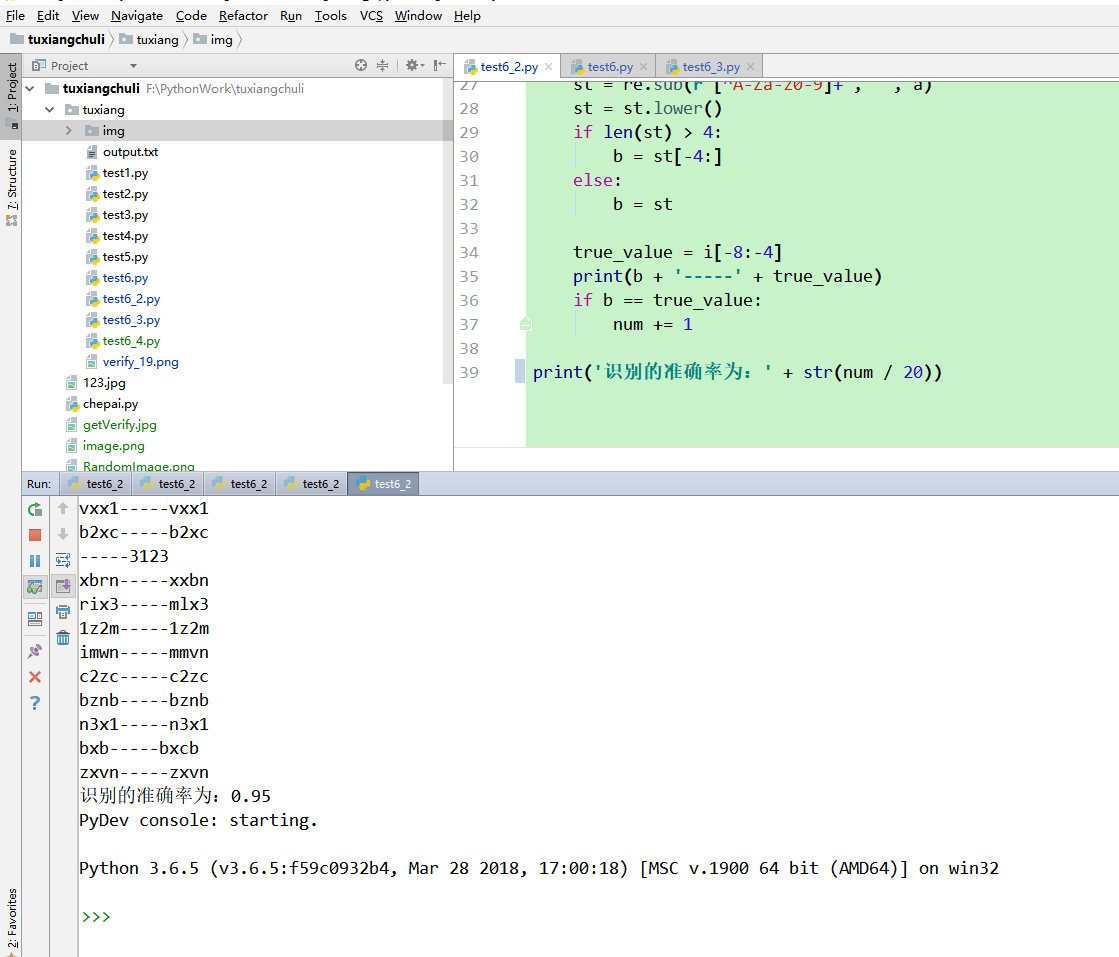


## 实验六、爬取某高校页面的验证码，进行验证码识别



对验证码图片手动信息标注，将图片上的验证信息放入图片的名称内，便于后续测试





爬取代码：

使用代理不断访问该网址获取验证码图片，并保存为png格式文件；

1. **from** urllib **import** request
2. **import** time
3. **import** random

6. **def** get\_and\_save\_verify(i):
7. **try**:
8. url = 'http://jwxt.qlu.edu.cn/verifycode.servlet'
9. request.urlretrieve(url, 'img/hb/' + 'verify\_' + str(i) + '.png')
10. **print**('第' + str(i) + '张图片下载成功')
11. **except** Exception:
12. **print**('第' + str(i) + '张图片下载失败')

15. **def** get\_proxy():
16. # 使用代理步骤
17. # - 1、设置代理地址
18. proxys = [{'http': '39.137.69.10:8080'},
19. {'http': '111.206.6.101:80'},
20. {'http': '120.210.219.101:8080'},
21. {'http': '111.206.6.101:80'},
22. {'https': '120.237.156.43:8088'}]
23. # - 2、创建ProxyHandler
24. proxy = random.choice(proxys)
25. proxy\_handler = request.ProxyHandler(proxy)
26. # - 3、创建Opener
27. opener = request.build\_opener(proxy\_handler)
28. # - 4、导入Opener
29. request.install\_opener(opener)

32. **if** \_\_name\_\_ == '\_\_main\_\_':
33. **for** i **in** range(1, 101):
34. get\_proxy()
35. time.sleep(random.randint(1, 4))
36. get\_and\_save\_verify(i)

验证码识别代码：

利用Tesseract-OCR进行图像信息识别，并将图像的识别结果与藏在图片文件名中的标签进行比对，测试识别的准确率。

**降噪处理：**

分别用高斯滤波、中值滤波和双边滤波对图像进行降噪处理，并不断调整参数，确定出对应方法的最优参数

1. blur = cv2.GaussianBlur(img, (3, 3), 0)  # 高斯滤波函数
2. blur = cv2.medianBlur(img, 3)  # 中值滤波函数
3. blur = cv2.bilateralFilter(img, 3, 560, 560)  # 双边滤波函数

数据清洗：

在双边滤波降噪处理的基础上再对识别结果进行数据清洗，将会提高识别的准确率；

1. # 对结果的处理
2. st = re.sub(r'[^A-Za-z0-9]+', '', a)
3. st = st.lower()
4. **if** len(st) > 4:
5. b = st[-4:]
6. **else**:
7. b = st

整个识别代码

1. **import** pytesseract
2. **import** cv2
3. **import** os
4. **import** numpy as np
5. **import** re

8. path = 'img/hb/'
10. file\_name = []
11. **for** k **in** os.walk(path):
12. file\_name = k[-1]
14. **print**('识别值' + '-----' + '真实值')
15. num = 0
16. **for** i **in** file\_name:
17. img = cv2.imdecode(np.fromfile(path + i, dtype=np.uint8), 1)
19. # 对数据的处理
20. # blur = cv2.GaussianBlur(img, (3, 3), 0)  # 高斯滤波函数
21. # blur = cv2.medianBlur(img, 3)  # 中值滤波函数
22. blur = cv2.bilateralFilter(img, 3, 560, 560)  # 双边滤波函数 560：0.28
24. a = pytesseract.image\_to\_string(blur)
26. # 对结果的处理
27. st = re.sub(r'[^A-Za-z0-9]+', '', a)
28. st = st.lower()
29. **if** len(st) > 4:
30. b = st[-4:]
31. **else**:
32. b = st
34. true\_value = i[-8:-4]
35. **print**(b + '-----' + true\_value)
36. **if** b == true\_value:
37. num += 1
39. **print**('识别的准确率为：' + str(num / 20))

识别结果：

