# 数字图像处理

## 作业 2：基于图像的校园卡智能终端键盘磨损程度评估

## 一. 项目简介

校园内随处可见的校园卡智能终端机，包含了自主充值、缴费、圈存转账、绑定银行卡等各种实用功能，为广大在校师生提供了巨大的便利。在用户操作终端机器时，涉及到输入金额、输入密码等操作时必然要使用机器上的键盘。一般来说，按键被按下的次数越多，该按键的磨损程度就会越大。本项目的内容就是根据实拍的键盘图像来评估不同按键的磨损程度。

## 二. 设计文档

本次实验采用的磨损前键盘图像和磨损后键盘图像分别如下：


提取其中的若干按键作为样本：

磨损前：


磨损后：


根据人眼的判断，这四个按键的磨损程度大小关系是：

$$\text{按键 6} \approx \text{按键 5} > \text{按键 9} > \text{按键 7}$$

下文将以此为标准来评判算法的准确性。

**下面考虑以三种方式实现磨损程度的评估。**

方式一：对于 Otsu 二值化后的图像进行总灰度值的比较

先将磨损前和磨损后的图像分别进行中值滤波去除噪声，再 Otsu 二值化：

磨损前：


磨损后：


对磨损前和磨损后的二值图像进行灰度值相减，得到的差值的绝对值大小反应了磨损程度。利用此方法得到的四个按键的差值绝对值依次是：634649，9949，745656，586055。得到的磨损程度大小关系是：

按键 7 > 按键 5 > 按键 9 > 按键 6

与人眼观察的结论相比，误差较大。

分析：经过二值化后只能比较字符处的磨损，如数字和字母，而有些磨损并非发生在字符上而是发生在金属色的空白区域，所以会引起较大误差。

算法核心代码：

```
# 通过计算灰度值之差来比较两幅图像相似度
def getdiff(img1,img2):
    # 定义边长
    Sidelength=300
    # 缩放图像
    img1 = cv2.resize(img1, (Sidelength, Sidelength), interpolation=cv2.INTER_CUBIC)
    img2 = cv2.resize(img2, (Sidelength, Sidelength), interpolation=cv2.INTER_CUBIC)
    avg1 = avg2 = 0
```

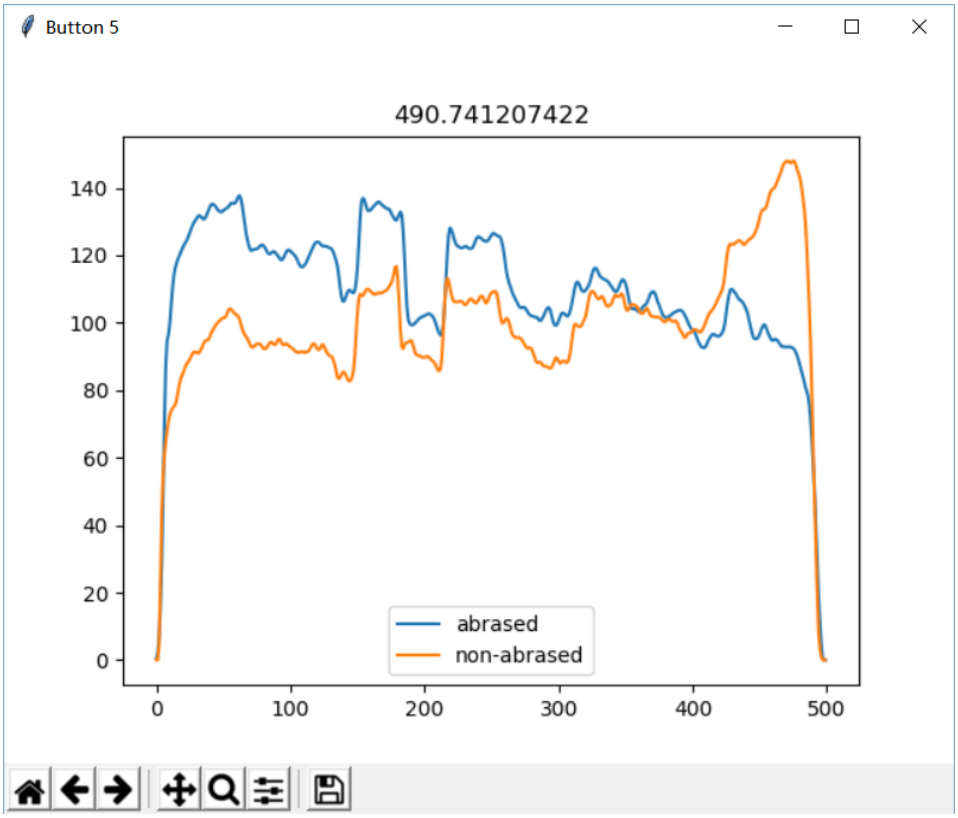
```
    # 计算像素灰度值总和
    for i in range(Sidelength):
        avg1 += sum(img1[i])
        avg2 += sum(img2[i])
    # 计算像素灰度值总差值
    return abs(avg1-avg2)

# 对图像先高斯滤波再 Otsu 二值化
def Otsu(path):
    img = cv2.imread(path,0)
    blur = cv2.GaussianBlur(img,(5, 5),0)
    ret, th = cv2.threshold(blur,0,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    return th
```

## 方式二：利用改进的哈希算法评估图象相似度

### 步骤：

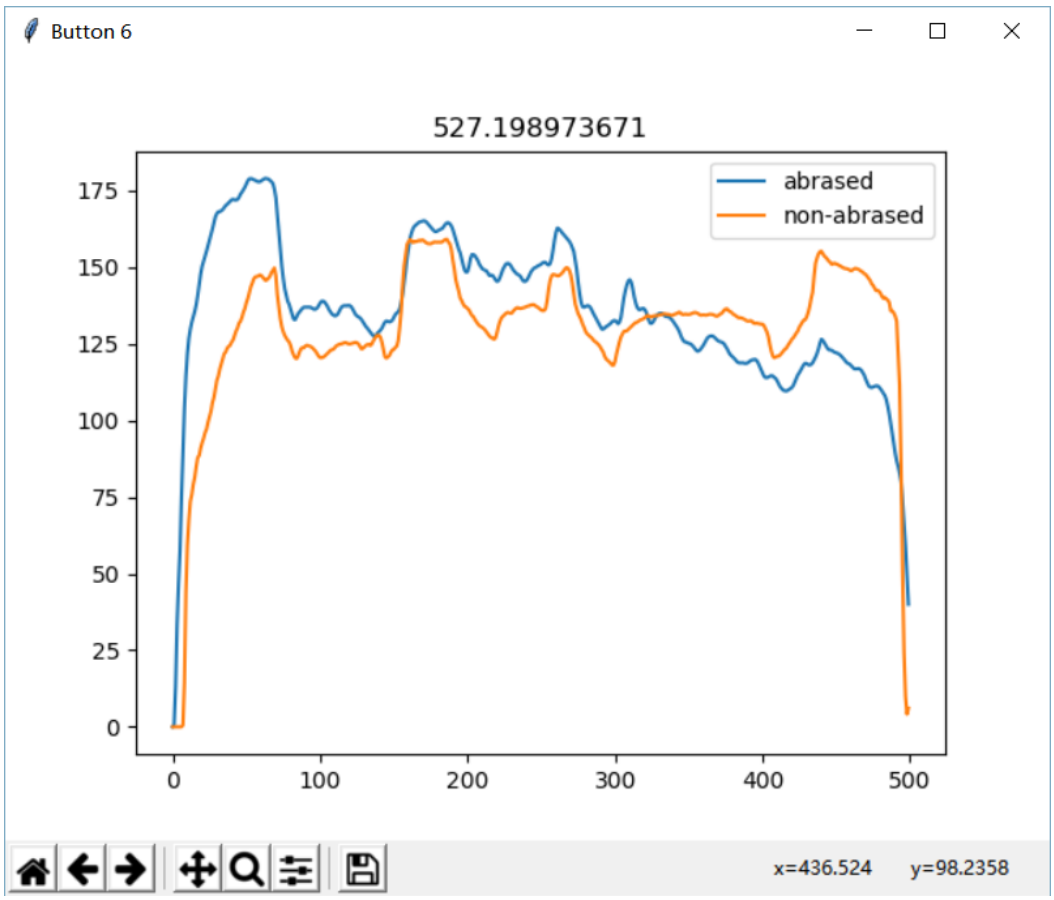
1. **缩放：** 将需要处理的图片所放到指定尺寸，缩放后图片大小由图片的信息量和复杂度决定。譬如，一些简单的图标之类图像包含的信息量少，复杂度低，可以缩放小一点。风景等复杂场景信息量大，复杂度高就不能缩放太小，容易丢失重要信息。根据需求弹性地缩放。在效率和准确度之间维持平衡。
2. **灰度化**：通常对比图像相似度和颜色关系不是很大，所以处理为灰度图，减少后期计算的复杂度。如果有特殊需求则保留图像色彩。
3. **计算平均值**： 分别依次计算并记录图像每行像素点的平均值，每一个平均值对应着一行的特征，所有行的平均值序列构成一幅图像的特征。
4. **平均值作差**： 分别依次计算并记录两幅图像每行像素点的平均值的差，若每行的平均值差都相近，说明每行图像的色调都相差相近的数值，即表示两幅图像总体特征相似。
5. **计算方差**：对得到的差值列表计算方差。一组数据方差的大小可以判断稳定性，方差越小，像素平均值的差的大小在每一行的分布越稳定，两幅图像就越相似。

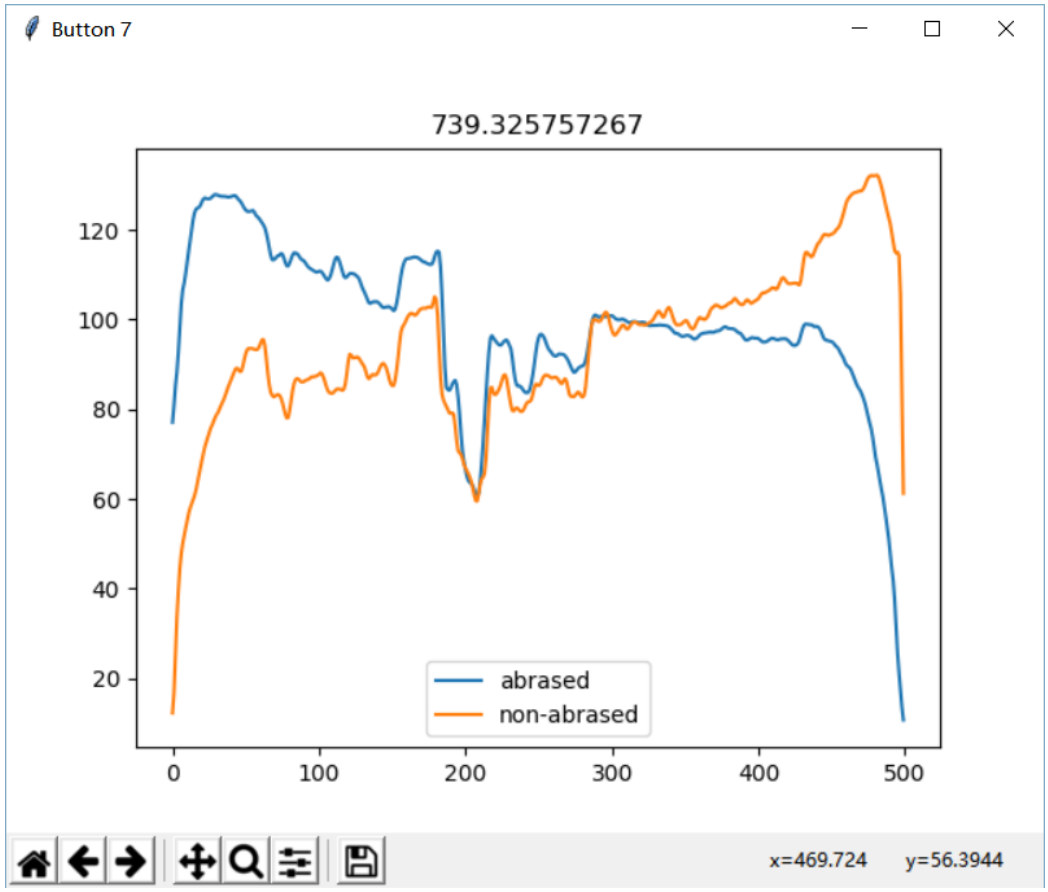
### 运行演示：

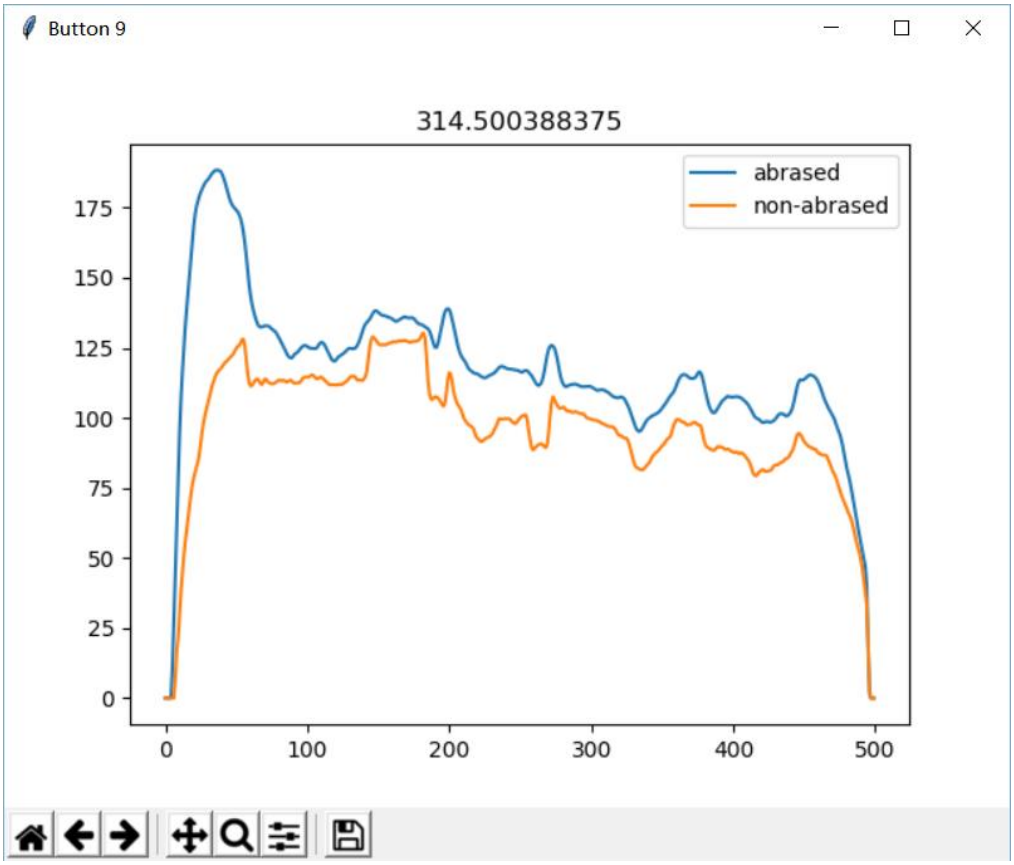



上图所示是按键 5 的灰度平均值—像素行分布图。原图像先被缩放成 500 × 500 的大小，再转灰度图像，计算每一行的平均灰度值。横坐标是行的序号，从 1 到 500 是图像的不同行，横坐标确定时，纵坐标反映了该行的灰度平均值大小。黄线代表磨损前图像，蓝线代表磨损后图像。该算法实质上比较的是两条曲线形状的相似程度，曲线形状越相似，则图片越相似，这种相似程度用曲线差值的方差来模拟，图表上方的数字即计算得到的方差，方差越小，相似程度越高。

按键 6、按键 7、按键 9 的图表依次如下：


Button 6
527.198973671
abrased
non-abrased
175
150
125
100
75
50
25
0
0
100
200
300
400
500
x=436.524
y=98.2358


Button 7
739.325757267
120
100
80
60
40
20
abrased
non-abrased
0
100
200
300
400
500
x=469.724
y=56.3944




根据该算法计算的方差得到的磨损程度大小关系是：

按键 7 > 按键 6 > 按键 5 > 按键 9

对比人眼观察的结论，该算法对按键 7 的磨损程度判断出现错误。

算法核心代码：

```
#获取一幅图像的每行像素平均值
def getdiff(img):
    #定义边长
    Sidelength=500
    #缩放图像
    img=cv2.resize(img,(Sidelength,Sidelength),interpolation=cv2.INTER_CUBIC)
    #灰度处理
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    #avglist 列表保存每行像素平均值
    avglist=[]
    #计算每行均值，保存到 avglist 列表
    for i in range(Sidelength):
        avg=sum(gray[i])/len(gray[i])
```

```
            avglist.append(avg)
    return avglist

# 计算两个图像曲线的差值的方差
def getss(list1,list2):
    newlist = []
    # 计算两个图像每行像素的均值的差，存入 newlist
    for i in range(len(list1)):
        avg = list1[i]-list2[i]
        newlist.append(avg)
    # 计算像素均值差的平均值 avg
    avg = sum(newlist)/len(newlist)
    # 计算像素均值差的方差 ss
    ss = 0
    for i in newlist:
        ss += (i-avg)*(i-avg)/len(newlist)
    return str(ss)

#显示两张图片的各行的像素均值曲线图，横坐标代表行数，纵坐标代表均值大小，表头显示像素均值差的方差
def show(path1,path2):
    plt.figure('Button 9')
    plt.plot(range(500),getdiff(cv2.imread(path1)),marker="",label="abrased")
    plt.plot(range(500),getdiff(cv2.imread(path2)),marker="",label="non-abrased")
    plt.title(getss(getdiff(cv2.imread(path1)),getdiff(cv2.imread(path2))))
    plt.legend()
    plt.show()
```

## 方式三：利用 SSIM 比较图象相似度

**SSIM（structural similarity）**结构相似性，也是一种全参考的图像质量评价指标，它分别从亮度、对比度、结构三方面度量图像相似性。

$$l(X,Y)=\frac{2\mu_X\mu_Y+C_1}{\mu_X^2+\mu_Y^2+C_1} \qquad c(X,Y)=\frac{2\sigma_X\sigma_Y+C_2}{\sigma_X^2+\sigma_Y^2+C_2} \qquad s(X,Y)=\frac{\sigma_{XY}+C_3}{\sigma_X\sigma_Y+C_3}$$

其中 ux、uy 分别表示图像 X 和 Y 的均值，σX、σY 分别表示图像 X 和 Y 的方差，σXY 表示图像 X 和 Y 的协方差，即

$$\mu_X = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X(i,j) \qquad \sigma_X^2 = \frac{1}{H \times W - 1} \sum_{i=1}^{H} \sum_{j=1}^{W} (X(i,j) - \mu_X)^2$$

$$\sigma_{XY} = \frac{1}{H \times W - 1} \sum_{i=1}^{H} \sum_{j=1}^{W} ((X(i,j) - \mu_x)(Y(i,j) - \mu_y))$$

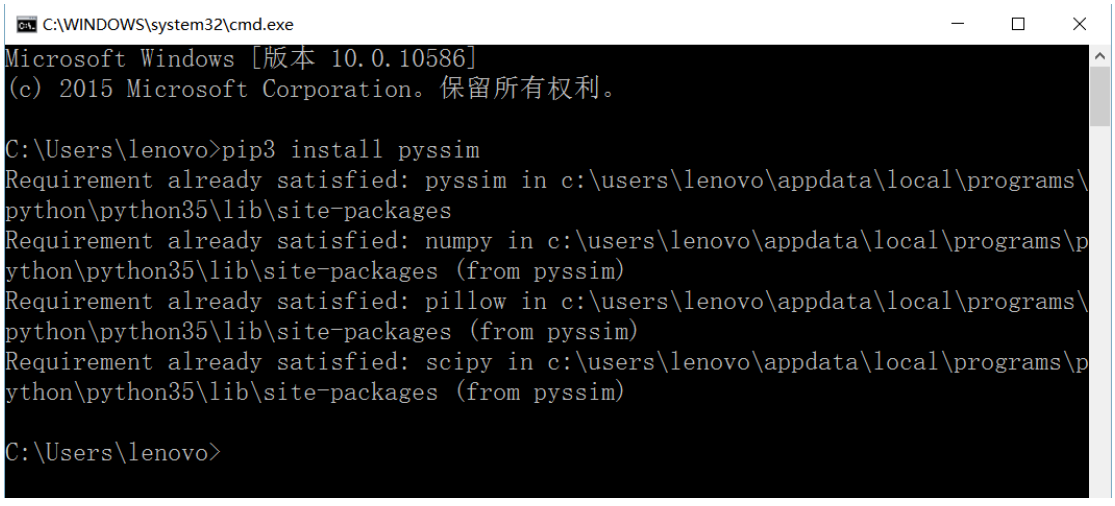
C1、C2、C3 为常数，为了避免分母为 0 的情况，通常取 C1=(K1*L)^2, C2=(K2*L)^2, C3=C2/2，一般地 K1=0.01, K2=0.03, L=255. 则

$$SSIM(X,Y) = l(X,Y) \bullet c(X,Y) \bullet s(X,Y)$$

SSIM 取值范围[0,1]，值越大，表示图像失真越小。

由于 Python 自带计算 SSIM 的包 pyssim，计算 SSIM 可以直接在控制台窗口完成，演示如下：

1. 安装 pyssim 包 (pip install pyssim)




2. 进入图片所在路径（cd 图片路径）

```
C:\Users\lenovo>cd C:\Users\lenovo\PycharmProjects\untitled

C:\Users\lenovo\PycharmProjects\untitled>
```

3. 计算两张图片的 SSIM（pyssim 图 1 名称 图 2 名称）

```
C:\Users\lenovo\PycharmProjects\untitled>pyssim 5d.jpg 5u.jpg
0.5461417

C:\Users\lenovo\PycharmProjects\untitled>pyssim 6d.jpg 6u.jpg
0.62357

C:\Users\lenovo\PycharmProjects\untitled>pyssim 7d.jpg 7u.jpg
0.6174604

C:\Users\lenovo\PycharmProjects\untitled>pyssim 9d.jpg 9u.jpg
0.6458274

C:\Users\lenovo\PycharmProjects\untitled>
```

根据该算法计算的方差得到的磨损程度大小关系是：

按键 5 > 按键 7 > 按键 6 > 按键 9

对比人眼观察的结论，该算法对按键 7 的磨损程度判断出现错误。