数字图像处理作业报告一

学号：71194506019 姓名：姜志刚 专业：计算机技术

题目

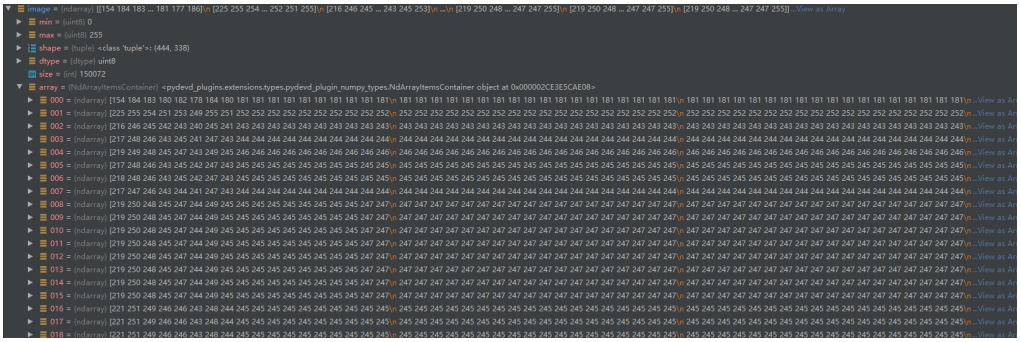
打开图像，显示图像，存储图像；对一张图像进行缩放，观察其分辨率，降低灰度分辨率。

打开、显示、存储图片

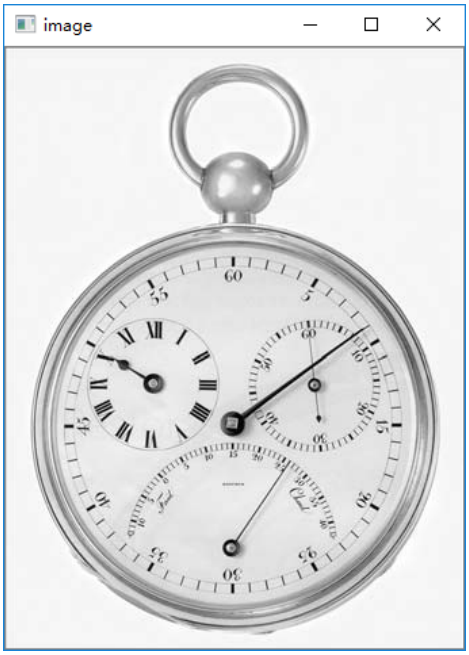
用Python版OpenCV实现

```
image = cv2.imread("images/2_20_a.jpg", cv2.IMREAD_UNCHANGED)
cv2.imwrite("images/full.jpg", image, [int(cv2.IMWRITE_JPEG_QUALITY), 100])
cv2.imshow("image", image)
cv2.waitKey(0)
```

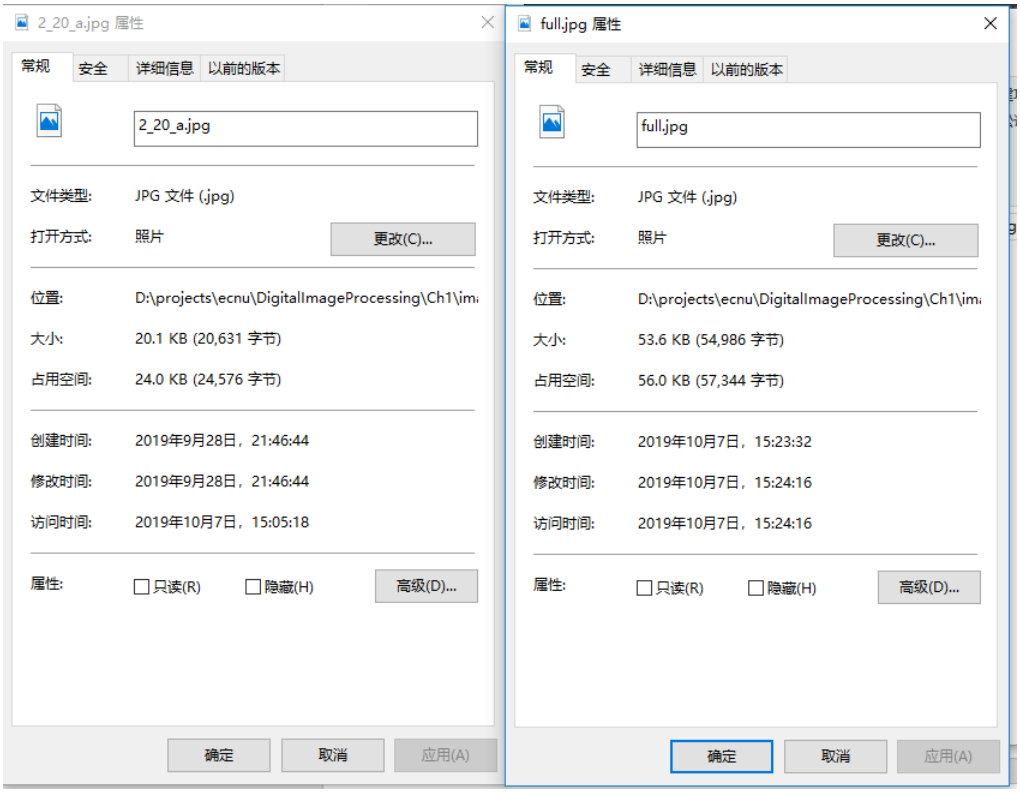
打开图片后，可以看图片由矩阵构成，shape显示了图片的行高和列宽，打开的是一幅灰度图片，因此只有一个通道



显示图片:



保存：



未经压缩，该图片占444 * 338 * 8bit = 444 * 338 B = 146.5KB
原图在磁盘占20.1KB
保存的图片占53.6KB

图像缩放

最近领域插值

h, w, h_0, w_0 分别为原图高、宽，和目标高、宽
$g(x, y)$为新图像x行y列的灰度等级，$f(x, y)$为原图像

$$g(x, y) = f(\frac{xh}{h_0}, \frac{yw}{w_0})(1)$$

或从原图像对应目标图像

$$g(\frac{xh_0}{h}, \frac{yw_0}{w}) = f(x, y)(2)$$

在(1)式中，假如 $\frac{h}{h_0} > 1$ 缩小图像
在(2)式中，若g(x, y)为原图像，f(x, y)为新图像，则 $\frac{h_0}{h} < 1$ 放大图像
出现小数时，取整。

Python关键代码：

```
def nearest(img, size):
    """
    Nearest neighbor interpolation
    :param img: source image
    :param size: (height, width)
    :return: destination image
    """
    re = np.zeros([size[0], size[1], 1], np.uint8)
    for x in range(size[0]):
        for y in range(size[1]):
```

```
            new_x = int(x * (img.shape[0] / size[0]))
            new_y = int(y * (img.shape[1] / size[1]))
            re[x, y] = img[new_x, new_y]
    return re
```

双线性插值

对于一个目的像素，设置坐标通过反向变换得到的浮点坐标为$(i+u, j+v)$ (其中i、j均为浮点坐标的整数部分，u、v为浮点坐标的小数部分，是取值[0,1)区间的浮点数)，则这个像素得值 $f(i+u, j+v)$ 可由原图像中坐标为$ (i,j), (i+1,j), (i,j+1), (i+1,j+1)$所对应的周围四个像素的值决定，即：

$$g(x,y)=f(\frac{xh}{h_0},\frac{yw}{w_0})=f(i+u,j+v)=(1-u)(1-v)f(i,j)+(1-u)vf(i,j+1)+u(1-v)f(i+1,j)+u$$

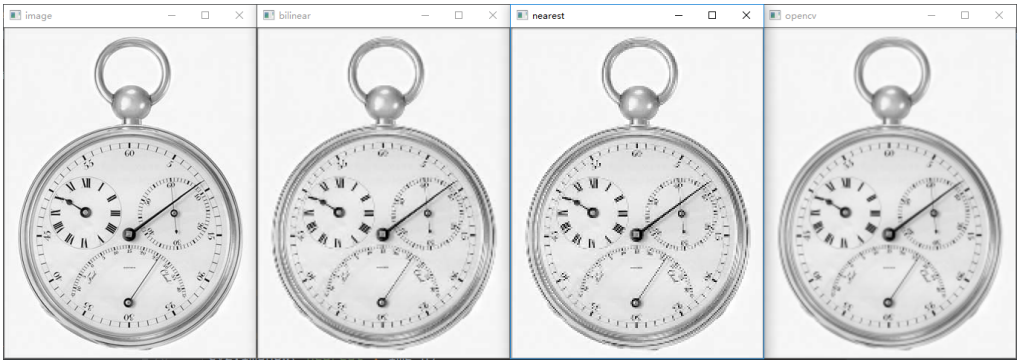
出现小数时考虑周围情况，离哪个点进，那个点发挥的作用就更大。

Python关键代码：

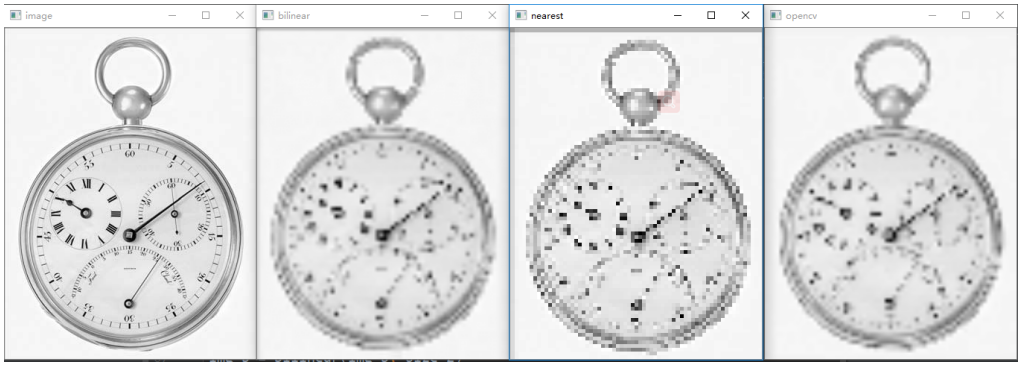
```
def bilinear(img, size):
    """
    Bilinear interpolation
    :param img: source image
    :param size: (height, width)
    :return: destination image
    """
    re = np.zeros([size[0], size[1], 1], np.uint8)
    for x in range(size[0]):
        for y in range(size[1]):
            new_x = x * (img.shape[0] / size[0])
            new_y = y * (img.shape[1] / size[1])
            i = int(new_x)
            j = int(new_y)
            u = new_x - i
            v = new_y - j
            if i + 1 >= img.shape[0]:
                i = img.shape[0] - 2
            if j + 1 >= img.shape[1]:
                j = img.shape[1] - 2
            # f(i+u,j+v)=(1-u)(1-v)f(i,j)+(1-u)vf(i,j+1)+u(1-v)f(i+1,j)+uvf(i+1,j+1)
            re[x, y] =  (1-u)*(1-v)*img[i, j]
             + (1-u)*v*img[i, j+1]
             +  u*(1-v)*img[i+1, j]
             +  u*v*img[i+1, j+1]
    return re
```

结果：

从左到右
图一：原图
图二：用最近领域插值缩放0.5，再扩大2倍，恢复原图大小
图三：用双线性插值缩放0.5，再扩大2倍，恢复原图大小
图四：OpenCV.resize缩放0.5，再扩大2倍，恢复原图大小



缩放倍数为0.2，再恢复原大小时



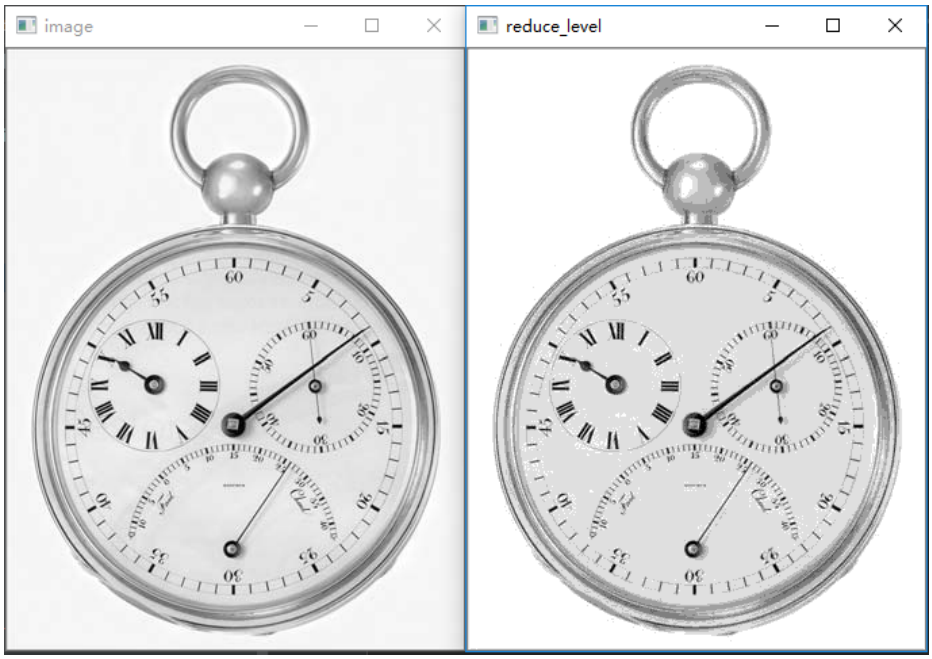
OpenCV的效过较好，双线性插值其次，最近领域插值效果非常差。

降低灰度分辨率

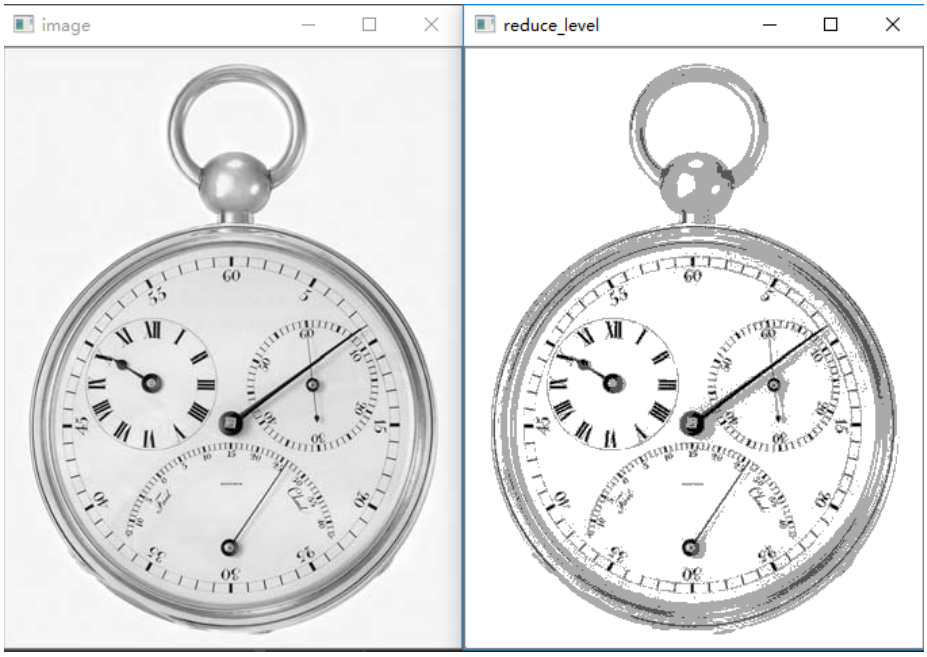
下面函数先量化到level+1级，再量化至256级以显示

```
def reduce_intensity_levels(img, level):
    img = cv2.copyTo(img, None)
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            si = img[x, y]
            ni = int(level * si / 255 + 0.5) * (255 / level)
            img[x, y] = ni
    return img
```

8灰度级图像



4灰度级图像



出现假轮廓

2灰度级图像（2值化 0~127置0 128~256置255）