**一、实验目的和要求**

1.熟练掌握OpenCV中图像点运算基本方法；

2.熟练掌握OpenCV中图像代数运算基本方法；

3.熟练掌握OpenCV中图像几何运算基本方法。

**二、实验主要仪器和设备**

1.计算机；2.Python与Opencv库；3.实验所需数据。

**三、实验方法和步骤**

1.打开计算机，启动Python运行环境；

2.采集实验所需图像和视频数据，并学习上述例题；

3.完成两幅图像的加、减、乘、除四种代数运算，并显示运算结果；

4.完成图像的尺寸变化、旋转和剪切操作，并显示操作结果；

5.完成图像的仿射变换操作，并显示操作结果；

6.使用非线性变换函数对一幅彩色图像进行点运算，并与使用线性变换函数进行点运算的结果相比较；

7.记录和整理实验报告。

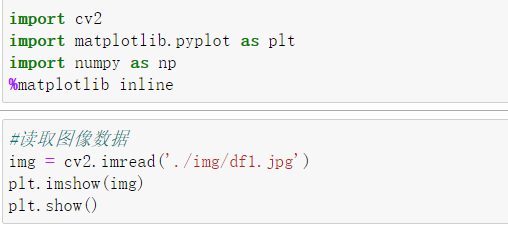
**四、实验结果分析**

**（实验源代码、实验结果图、实验分析比较、结论）**

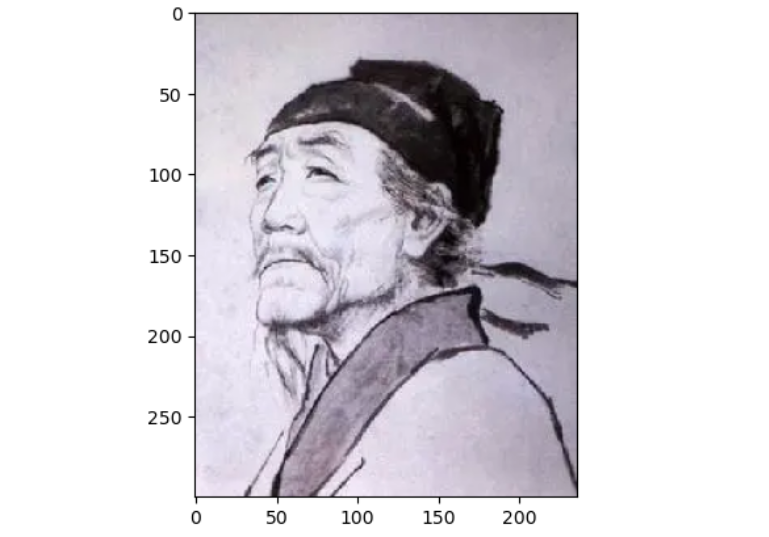
**1.完成两幅图像的加、减、乘、除四种代数运算，并显示运算结果；**

读取图像数据；

实验源代码：

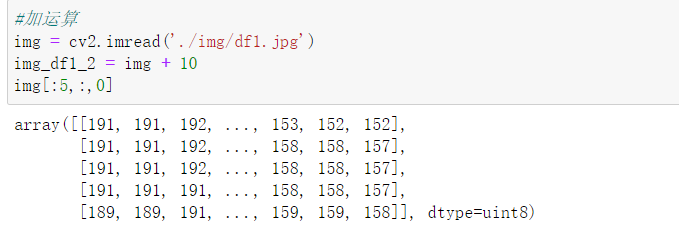


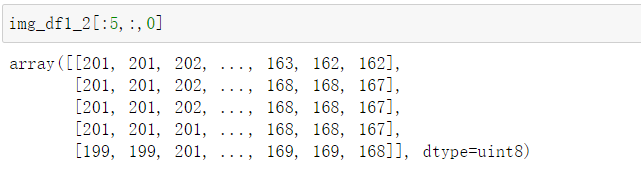
实验结果图：

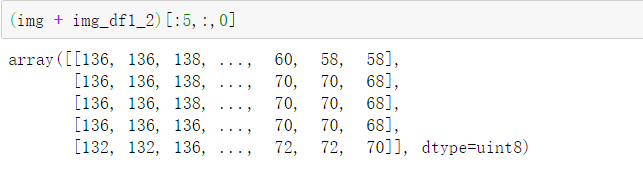


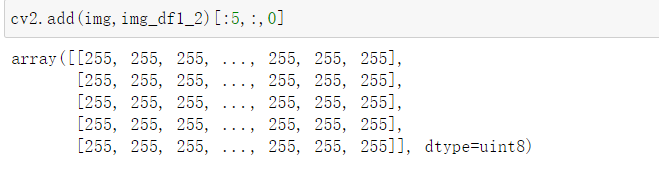
加运算；

实验源代码：



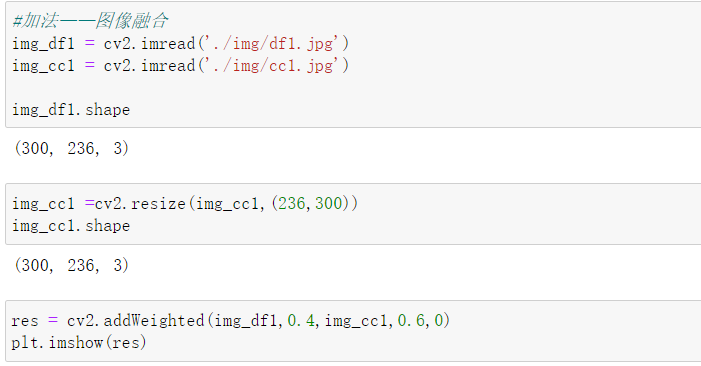
、



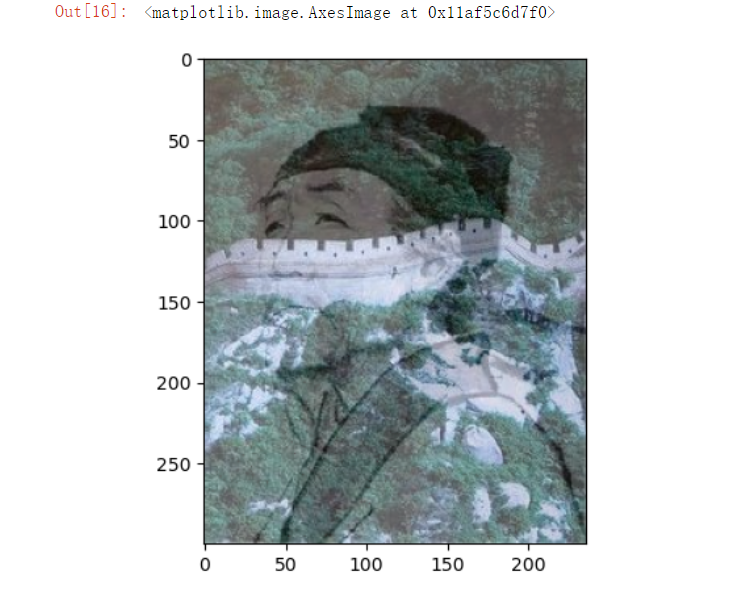


加法——图像融合；

实验源代码：

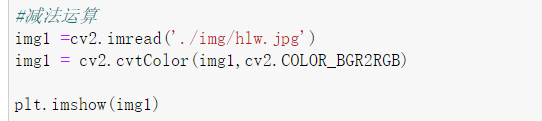


实验效果图：

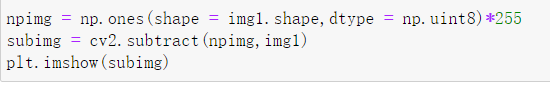


减法运算；

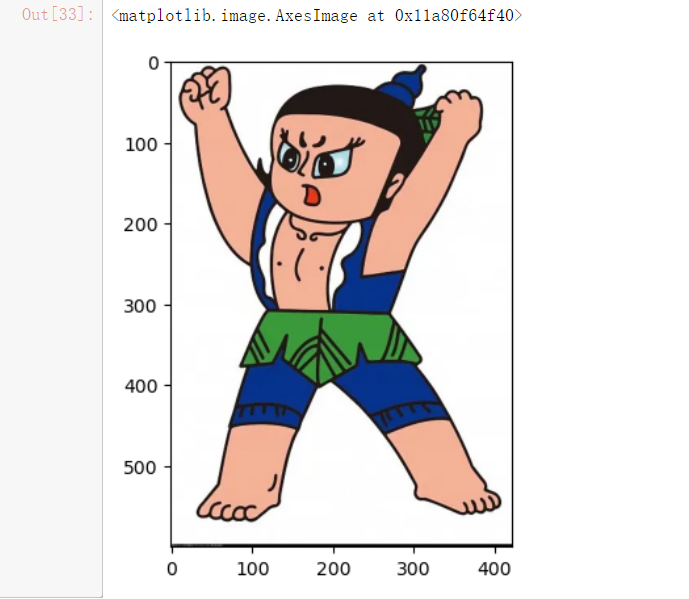
实验源代码：

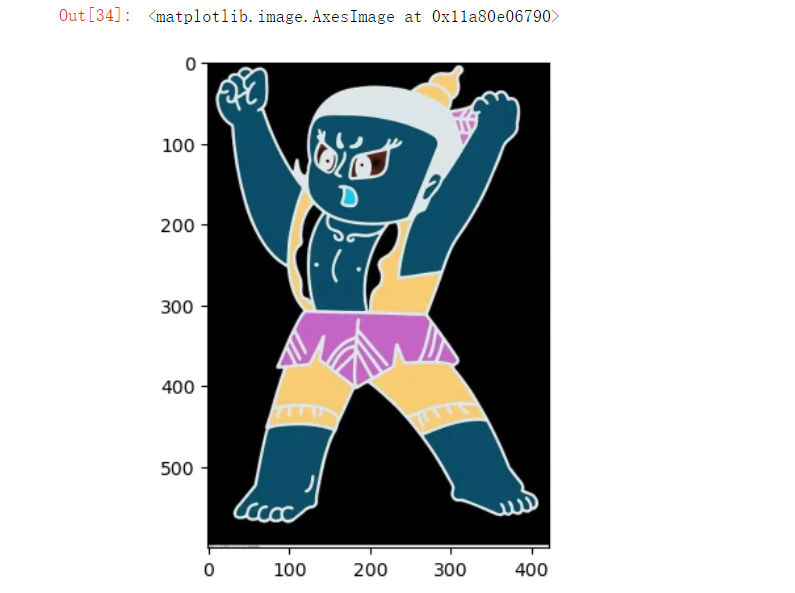


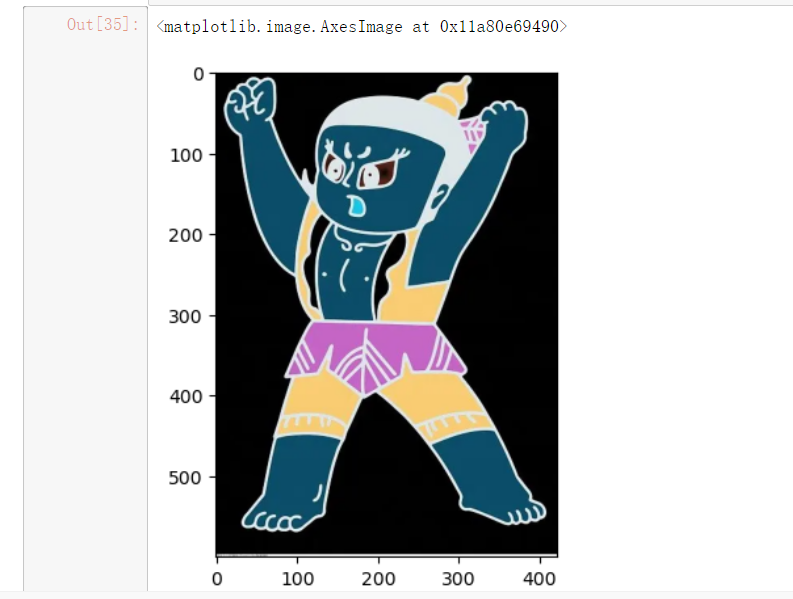
IMG_256



实验效果图：

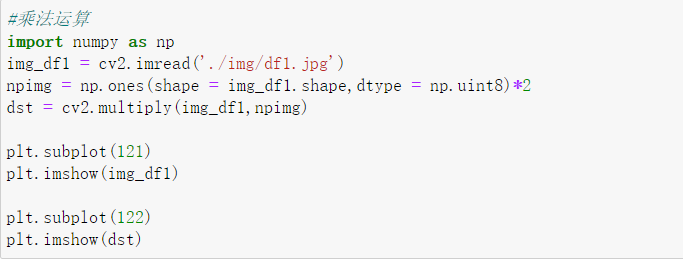




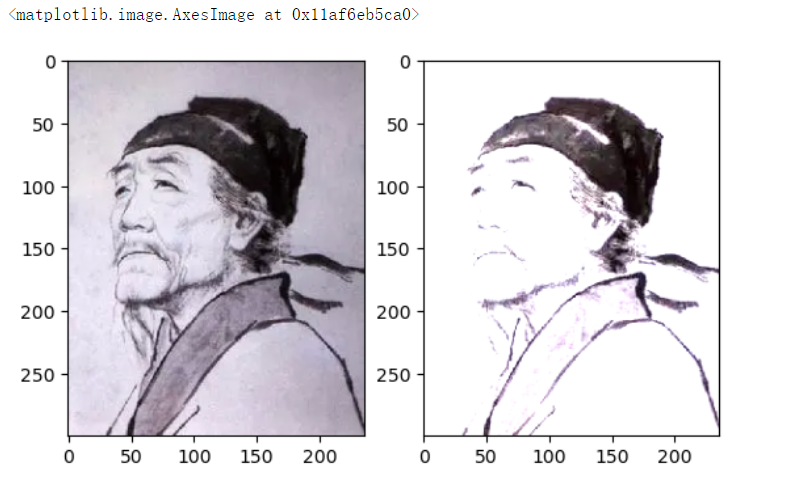


乘法运算；

实验源代码：

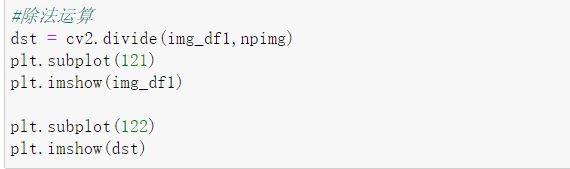


实验效果图：



除法运算；

实验源代码：

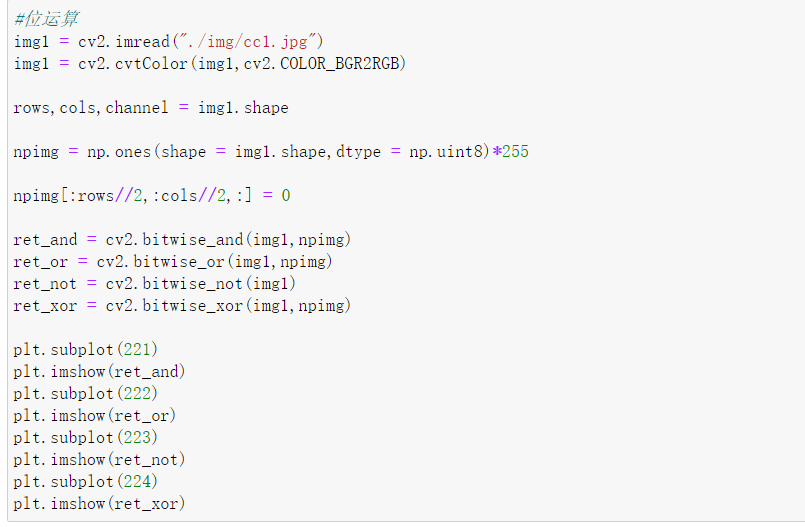


实验效果图：

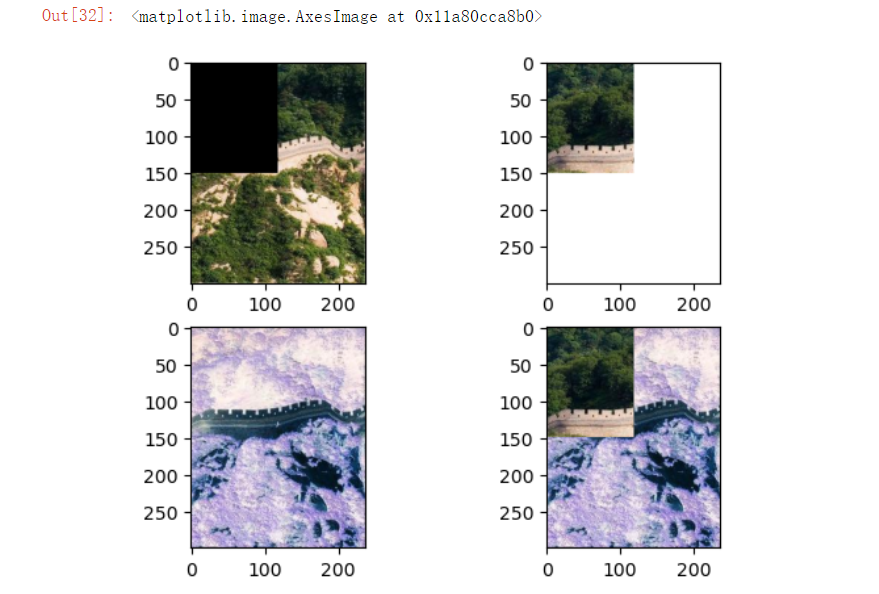


位运算；

实验源代码：



实验效果图：



实验分析比较与结论：

加法运算这段代码使用了OpenCV库加载一张名为“df1.jpg”的图像，并对该图像进行了一系列加法操作。

(img + img\_df1\_2)[:5,:,0]: 将变量img和img\_df1\_2中对应像素值相加，生成一个新的图像，再取该图像前5行、所有列、第1个通道的像素值。

减法运算这段代码使用了OpenCV和Matplotlib库来对一张名为“hlw.jpg”的图像进行减法运算，并进行可视化展示。

img1 = cv2.cvtColor(img1,cv2.COLOR\_BGR2RGB): 将img1从BGR颜色空间转换为RGB颜色空间。

img2 = 255 - img1: 对img1进行减法运算，将255减去每个像素的值，得到新的图像，并将其赋值给变量img2。

我们需要注意的是，在进行加减法运算时，如果两个像素值相加减超过了像素值的最大取值范围（如在8位无符号整型图像中，最大像素值为255），则会发生截断现象，即超过最大取值范围或最小范围的部分会被截断，导致图像颜色变化。因此，在使用加减法运算时，需要注意像素值的取值范围，避免出现不必要的颜色失真。

乘法运算这段代码使用了OpenCV、NumPy和Matplotlib库来对一张名为“df1.jpg”的图像进行乘法运算。

npimg = np.ones(shape=img\_df1.shape, dtype=np.uint8) \* 2: 使用NumPy库创建一个与img\_df1大小相同、像素值都为2的图像，并将其赋值给变量npimg。这个数组用于和img\_df1进行乘法运算。

dst = cv2.multiply(img\_df1, npimg): 对img\_df1和npimg进行乘法运算，得到新的图像，并将其赋值给变量dst。

plt.subplot(121), plt.imshow(img\_df1): 使用Matplotlib库的subplot和imshow函数将原始图像img\_df1显示在第一个子图中。

plt.subplot(122), plt.imshow(dst): 使用Matplotlib库的subplot和imshow函数将乘法运算结果dst显示在第二个子图中。

在这段代码中，使用了OpenCV库的multiply函数来进行乘法运算。multiply函数将两个图像对应像素的值相乘，返回一个新的图像。由于这里npimg中每个像素值都为2，因此对于原图像img\_df1中的每个像素，相应位置上的像素值都会乘以2。这样做可以使图像变得更加明亮。

除法运算这段代码使用了OpenCV和Matplotlib库对一张名为“df1.jpg”的图像进行除法运算。

dst = cv2.divide(img\_df1, npimg): 对img\_df1和npimg进行除法运算，得到新的图像，并将其赋值给变量dst。

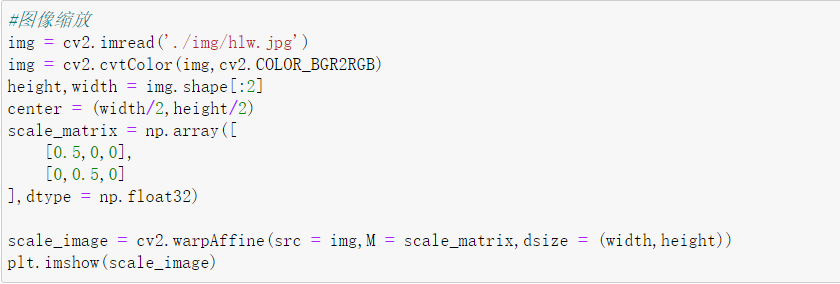
在这段代码中，使用了OpenCV库的divide函数来进行除法运算。divide函数将两个图像对应像素的值相除，返回一个新的图像。由于这里除数npimg中每个像素值都为2，因此对于原图像img\_df1中的每个像素，相应位置上的像素值都会除以2。这样做可以使图像变得更加暗淡。

需要注意的是，在使用除法运算时，如果除数中有像素值为0，将会导致除法运算出现除以0的情况，产生错误。因此，在使用除法运算时，需要注意避免除数中存在0值。

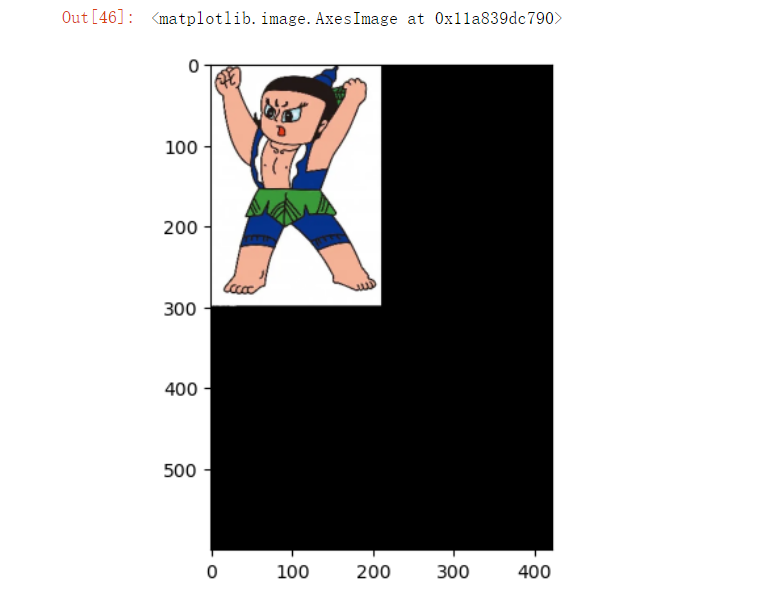
**2.完成图像的尺寸变化、旋转和剪切操作，并显示操作结果；**

图像缩放；

实验源代码：

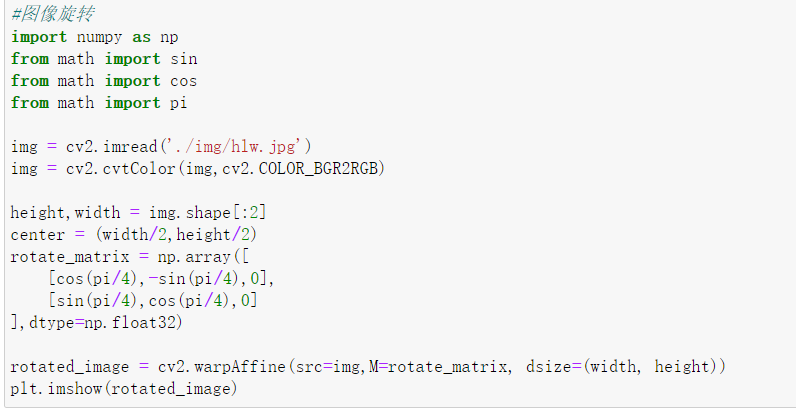


实验效果图：

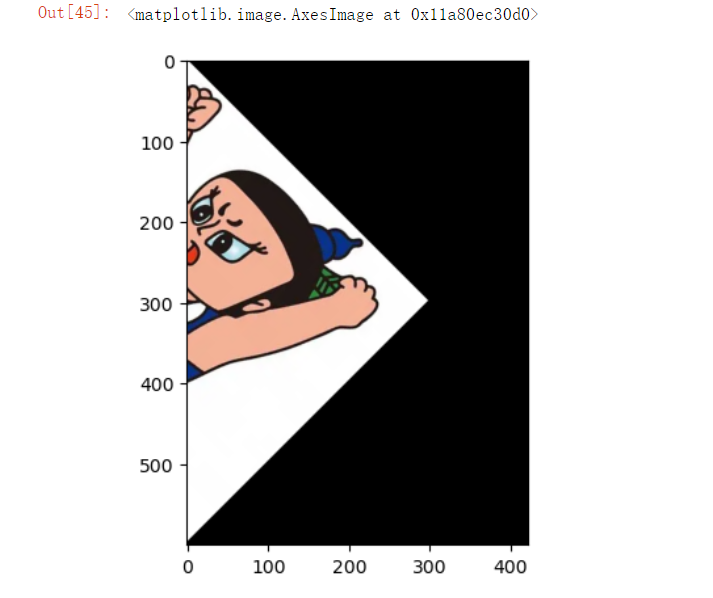


图像旋转；

实验源代码：

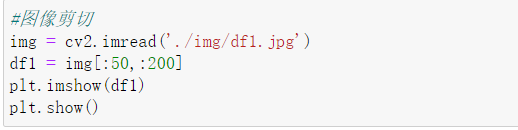


实验效果图：

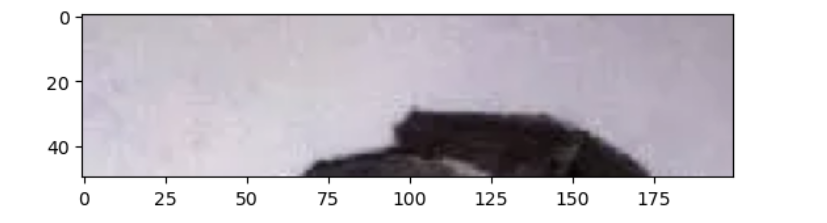


图像剪切；

实验源代码：



实验结果图：



实验分析比较与结论：

图像缩放这段代码具体实现过程如下：

读取一张图像，使用cv2.imread()函数，返回的图像是一个NumPy数组。

将图像从BGR颜色空间转换为RGB颜色空间，使用cv2.cvtColor()函数，因为Matplotlib使用RGB颜色空间。

获取图像的高度和宽度，使用img.shape[:2]。

计算缩放矩阵，使用np.array()函数构建一个2x3的矩阵，其中前两列分别表示x和y方向的缩放因子，最后一列表示平移向量。

调用cv2.warpAffine()函数实现仿射变换，参数src表示原始图像，M表示变换矩阵，dsize表示输出图像的大小。

使用Matplotlib的plt.imshow()函数显示缩放后的图像。

该代码中使用的缩放矩阵为等比例缩放矩阵，即x和y方向的缩放因子相同。

图像旋转这段代码调用cv2.warpAffine()函数实现仿射变换，参数src表示原始图像，M表示变换矩阵，dsize表示输出图像的大小。

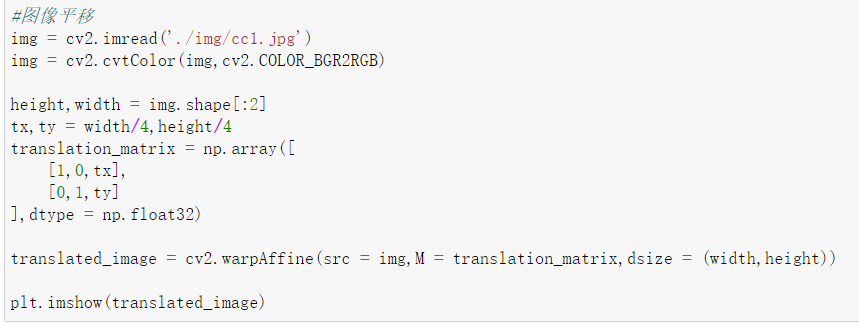
该代码中使用的旋转矩阵为顺时针旋转45度的矩阵，即旋转角度为π/4。

**3.完成图像的仿射变换操作，并显示操作结果；**

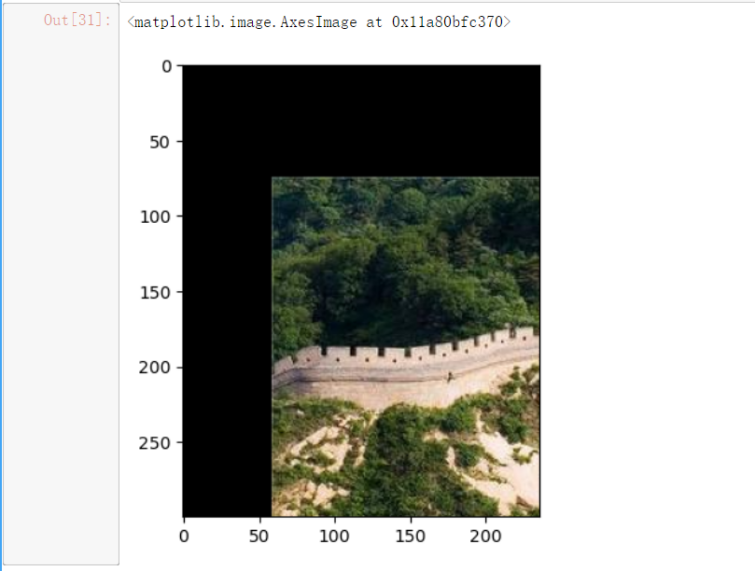
**（包括平移、缩放、旋转、翻转、错切）**

图像平移

实验源代码：



实验效果图：

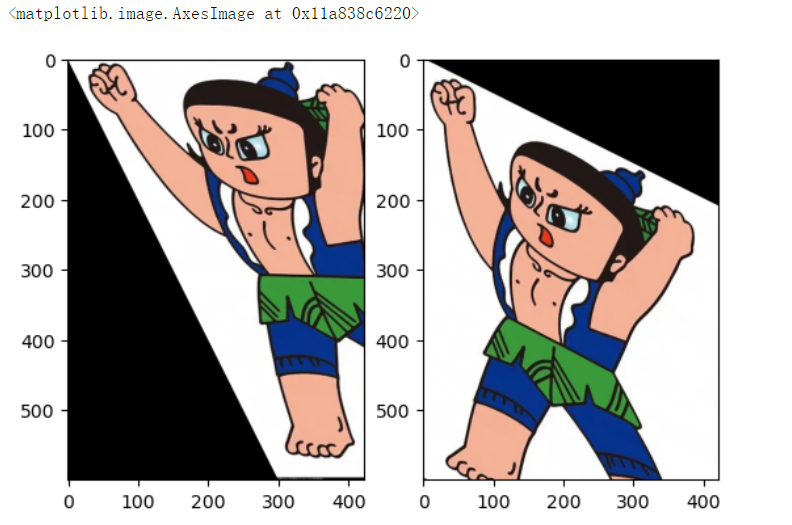


图像错切；

实验源代码：



实验效果图：

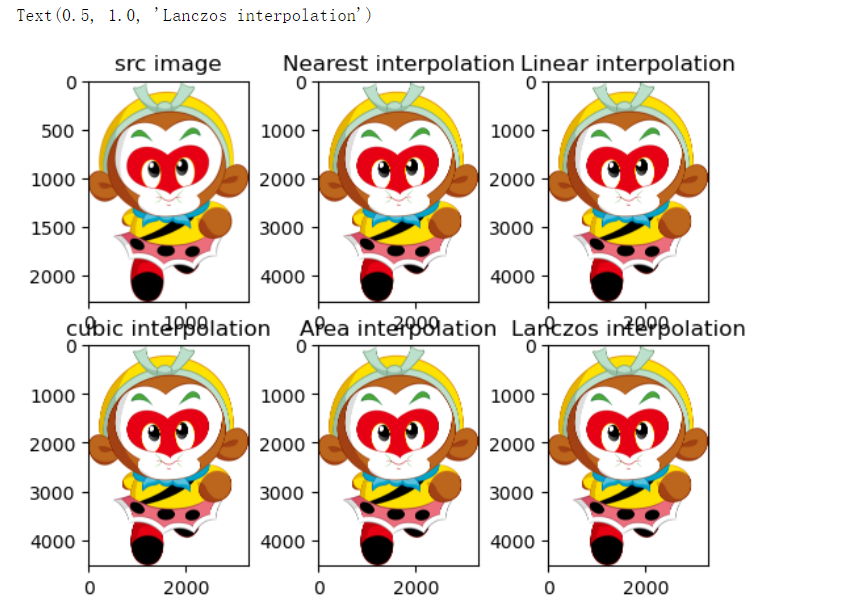


图像插值；

实验源代码：



实验效果图：



实验分析比较与结论：

图像平移这段代码获取图像的高度和宽度，使用img.shape[:2]。

计算平移矩阵，使用np.array()函数构建一个2x3的矩阵，其中第三列表示平移向量。

调用cv2.warpAffine()函数实现仿射变换，参数src表示原始图像，M表示变换矩阵，dsize表示输出图像的大小。

该代码中使用的平移矩阵将图像沿x和y方向平移了width/4和height/4个像素，即向右下方平移了图像。

错切变换，它是一种基于仿射变换的变换方式。在错切变换中，平行于X或Y轴的线在变换后不再平行于相应的轴，而是沿着一定角度倾斜，从而改变了线的方向。

代码中，使用了两个矩阵分别进行垂直和水平方向的错切变换。矩阵中的参数可以通过调节来控制变换程度和方向。最后使用cv2.warpAffine函数将变换应用于原始图像。

通过观察可视化结果，可以看到原始图像被垂直和水平方向的直线拉伸，形成了一种扭曲的效果。

图像差值这段代码演示了不同插值方法对图像进行缩放的效果。插值是一种用于计算未知函数在已知点处的近似值的方法，对于图像缩放来说，它可以用于计算图像中不存在的像素值。OpenCV提供了几种插值方法：

INTER\_NEAREST：最近邻插值。它将目标像素的值设置为最接近该像素的源像素的值。这种插值方法的速度很快，但会产生明显的锯齿状边缘。

INTER\_LINEAR：线性插值。它使用目标像素周围的四个源像素的加权平均值作为目标像素的值。这种插值方法速度较快，而且通常可以得到不错的结果。

INTER\_CUBIC：立方插值。它使用目标像素周围的16个源像素的加权平均值作为目标像素的值。这种插值方法会更加平滑，但也会更加耗时。

INTER\_AREA：区域插值。它计算目标像素覆盖的源像素的平均值，而不考虑周围的源像素。这种插值方法比较适用于缩小图像，但在放大图像时可能会产生较差的结果。

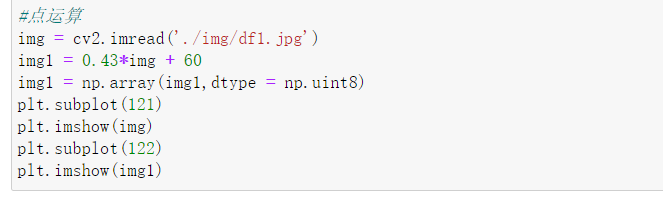
INTER\_LANCZOS4：Lanczos插值。它使用一个具有4个支点的Lanczos核函数来计算目标像素的值。这种插值方法通常可以得到较好的结果，但比其他插值方法更加耗时。

在本代码中，对一张原始图像进行了五种插值方法的演示。可以看出，不同插值方法对图像缩放的效果是有所不同的。

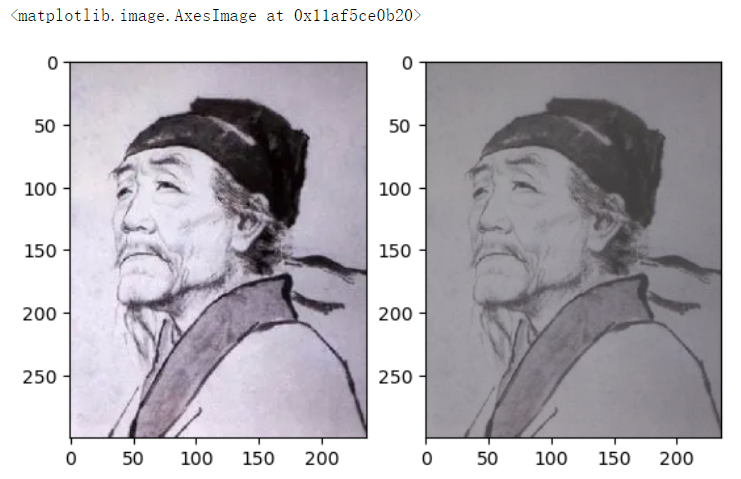
**4.使用非线性变换函数对一幅彩色图像进行点运算，并与使用线性变换函数进行点运算的结果相比较。**

点运算；

实验源代码：



实验效果图：



实验分析比较与结论：

这段代码实现了图像的点运算，将原始图像进行了亮度调整。具体来说，代码加载了一张图像，然后使用公式 img1 = 0.43\*img + 60 对每个像素进行了计算，得到新的图像 img1。这个公式实际上对原始图像中每个像素的灰度值进行了一个线性变换，使得整个图像整体变亮了，增加了对比度和亮度。

最后，代码将原始图像和处理后的图像进行了可视化，使用了 Matplotlib 库中的 subplot 和 imshow 函数将两张图像放在同一个窗口中显示出来。左边是原始图像，右边是亮度调整后的图像。

**五、实验总结与思考**

数字图像与视频处理是计算机视觉领域的基础，是许多高级图像和视频处理技术的基础。在数字图像与视频处理中，我们需要了解图像和视频的基本概念以及其处理的基本操作。在实验中，我们学习了图像和视频的读取、显示、存储、缩放、剪切、旋转、翻转、加减乘除等基本操作，并使用了Python的OpenCV和Matplotlib库进行实现。

通过本次实验，我对数字图像和视频的基本操作有了更深入的理解。我了解了如何读取和显示图像和视频，如何对它们进行基本的处理，以及如何将它们存储到文件中。我还学习了如何在Python中使用OpenCV和Matplotlib库对图像和视频进行处理，这些库提供了许多方便的函数和工具，使得图像和视频处理变得更加简单和高效。

此外，在本次实验中，我还学习了如何使用Jupyter Notebook进行交互式编程和结果展示。通过使用Jupyter Notebook，我可以方便地在一个笔记本中组织代码、注释、可视化结果和说明，使得整个实验过程更加清晰和易于理解。

总之，本次实验让我对数字图像和视频的基本操作有了更深入的了解，也让我熟悉了Python中的OpenCV和Matplotlib库的使用。这些基础知识对于我进一步学习和掌握计算机视觉领域的高级技术将非常重要。