# 《数字图像处理》实验报告

姓名：景恺丰 学号：24122365

实验3

1. **任务1** 
   1. **核心代码：**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15 | **def** gamma\_with\_LUT(img,gamma):      lut**=**np.zeros(256,np.uint8)#创建查找表  **for** index **in** range(256):          lut[index]**=**np.clip(((index**/**255.0)**\*\***gamma)**\***255.0,0,255).astype(np.uint8)          #gamma变换  #边界检测保护      im\_gamma**=**cv.LUT(img,lut)#用opencv自带函数进行映射  **return** im\_gamma  **def** gamma\_without\_LUT(img,gamma):      im\_gamma**=**np.array(np.clip(((img**/**255.0)**\*\***gamma)**\***255.0,0,255),np.uint8)      #直接使用gamma变换  **return** im\_gamma  im\_light**=**cv.imread("light.tif",1)  im\_dark**=**cv.imread("dark.jpg",1)  gamma\_dark**=**5#gamma>1使亮图变暗  gamma\_light**=**0.5#gamma<1使暗图变亮 |

* 1. **实验结果截图**

****

图1: light.tif变暗



图2:dark.jpg变亮

* 1. **实验小结**

本实验要求自己实现Gamma变换，采用2种不同版本，使用查找表和不使用查找表，比较二者的效率差异。

Gamma变换的计算公式为：output=((input/255.0)\*\*gamma)\*255可直接利用公式带入进行计算。其中Gamma值大于1时会使图变暗，Gamma值小于1时会使图变亮。

用查找表实现需要先创建表，然后利用循环语句遍历表对单个值进行计算，然后再用cv2自带函数进行映射。

实际运行代码中添加了计数器来比较两者效率，结果如下：

im\_light\_with\_LUT 's time : 0.005415201187133789

im\_light\_without\_LUT 's time : 0.014034032821655273

比较分析发现，使用LUT明显比不使用LUT快，图像更加清晰，效率更高。

1. **任务2** 
   1. **核心代码：**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22 | **def** self\_equalize\_hist(img\_gray):      img\_gray**=**np.clip(img\_gray,0,255).astype(np.uint8)      #hist,bins = np.histogram(img\_gray.flatten(),256,[0,256])#一直报错，怀疑是numpy的兼容性问题      pixels **=** img\_gray.flatten()#用flatten(）——将图像展为一维数组      hist**=**np.zeros(256,dtype**=**np.int64)  **for** x **in** pixels:          hist[x]**=**hist[x]**+**1      #不依赖函数计算hist值      cdf **=** hist.cumsum()#计算累计分布函数      cdf\_min**=**cdf.min()      cdf\_max**=**cdf.max()      cdf\_normalized**=**((cdf**-**cdf\_min)**\***255.0**/**(cdf\_max**-**cdf\_min)).astype(np.uint8)#对累计分布函数进行归一化，确保灰度为整形  **return** cdf\_normalized[img\_gray]    **def** hist\_plot(img,title):      img\_gray**=**np.clip(img,0,255).astype(np.uint8)      #plt.hist(img.flatten(),256,[0,256])一直报错，怀疑是兼容性问题      pixels **=** img\_gray.flatten()      hist**=**np.bincount(pixels,None,256).astype(np.int64)      plt.bar(np.arange(len(hist)),hist)      plt.title(title)      plt.show()#绘制直方图 |

* 1. **实验结果截图**

****

图3-1 dark从左到右分别为原图，自己实现的，cv自带函数实现的

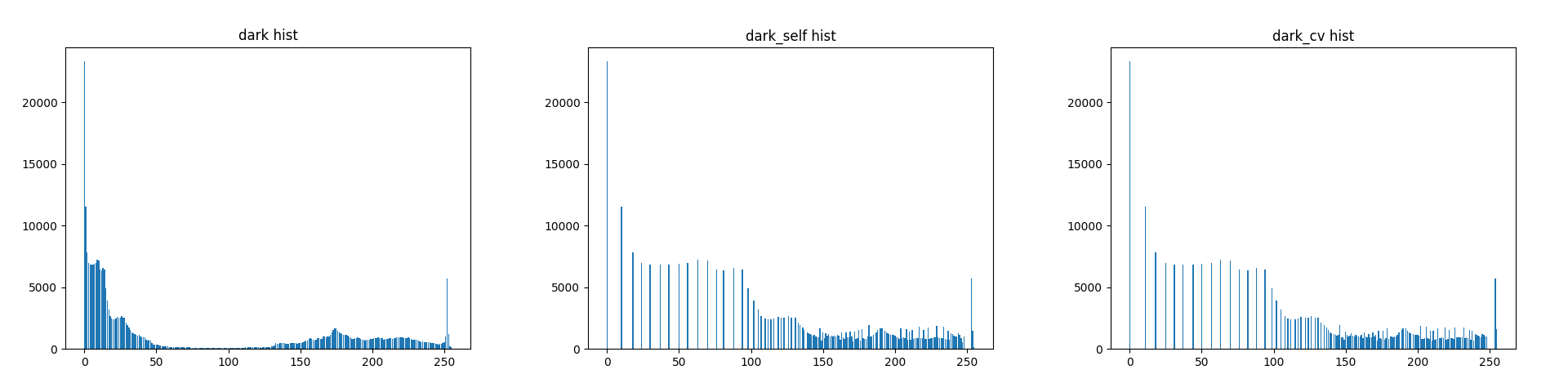


图3-2 从左到右分别为原图的，自己实现图的，cv自带函数实现图的直方图



图4-1 hill从左到右分别为原图，自己实现的，cv自带函数实现的

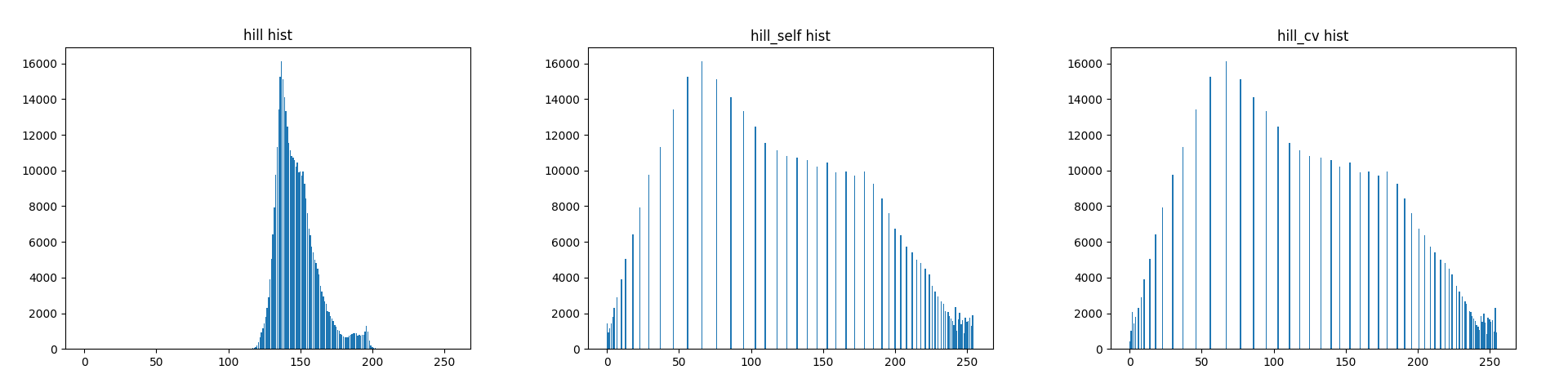


图4-2 从左到右分别为原图的，自己实现图的，cv自带函数实现图的直方图

图5-1 baby从左到右分别为原图，自己实现的，cv自带函数实现的

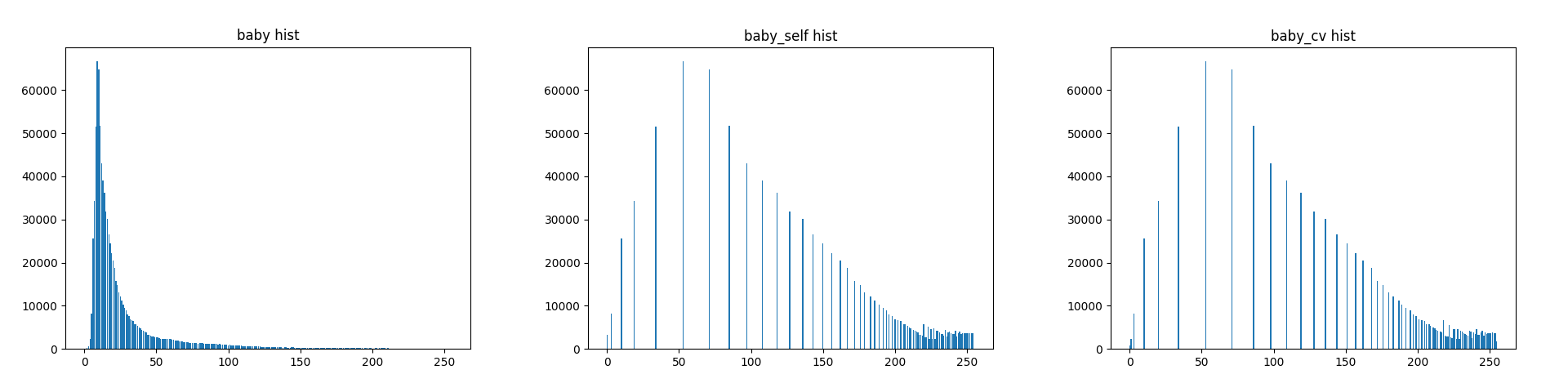


图5-2 从左到右分别为原图的，自己实现图的，cv自带函数实现图的直方图

* 1. **实验小结**

本任务要求自己实现直方图均衡化，绘出变换前后的直方图，并和OpenCV中对应函数进行效果对比。

为实现任务，先要计算直方图，用flatten() 方法将其展平为一维数组，得到每个元素表示对应区间中的像素数量——hist。之后再计算累积分布函数（CDF）。CDF 是对频率分布函数（PDF）的积分，它表示每个像素值在原始图像中出现的概率。CDF 可以通过对 PDF 进行累加计算得到。最后需要对其进行归一化，构建新的灰度级映射函数：Sk = (L - 1) \* C(rk)（其中rk 为某个像素值，Sk 是新的灰度值，范围仍在 [0,L−1] 之间），该公式表示累积分布函数的归一化映射，将原始图像的灰度值范围拉伸至整个灰度级范围。

在实现过程中发现hist,bins = np.histogram(img\_gray.flatten(),256,[0,256])语句一直报错，怀疑是numpy的兼容性问题（绘制直方图时也发现此问题）。

实验结果发现，自行实现的效果较好，和cv2自带函数实现的结果差不多。

根据所得直方图可发现，方图均衡化调整了图像的灰度级分布，使得图像的像素值更加均匀地分布在整个灰度范围内，从而改善图像的整体视觉效果。

由直方图可证明自行实现的效果较好，和cv2自带函数实现的结果差不多。实验较为成功。

1. **任务3** 
   1. **核心代码：**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22 | **def** get\_filter(img,kernel,i,j):      #获取点周围与算子大小对应的图像区域      half\_kernel**=**kernel**//**2 #将卷积整除      start\_i**=**np.clip(i**-**half\_kernel,0,img.shape[0]**-**1)      start\_j**=**np.clip(j**-**half\_kernel,0,img.shape[1]**-**1)      end\_i**=**np.clip(i**+**kernel**-**half\_kernel,0,img.shape[0])      end\_j**=**np.clip(j**+**kernel**-**half\_kernel,0,img.shape[1])#边界检测  **return** img[start\_i:end\_i,start\_j:end\_j]  **def** mean\_filtering(img,kernel):      im\_filtered **=** np.zeros(img.shape,np.uint8)  **for** i **in** range(img.shape[0]):  **for** j **in** range(img.shape[1]):  **for** k **in** range(img.shape[2]):                  im\_filtered[i,j,k] **=** np.mean(get\_filter(img[:,:,k],kernel,i,j))  **return** im\_filtered #计算点周围灰度的平均数  **def** median\_filtering(img,kernel):      im\_filtered **=** np.zeros(img.shape,np.uint8)  **for** i **in** range(img.shape[0]):  **for** j **in** range(img.shape[1]):  **for** k **in** range(img.shape[2]):                  im\_filtered[i,j,k] **=** np.median(get\_filter(img[:,:,k],kernel,i,j))  **return** im\_filtered #计算点周围灰度的中值 |

* 1. **实验结果截图**

****

图6 Mona从左到右分别为原图，均值滤波，中值滤波

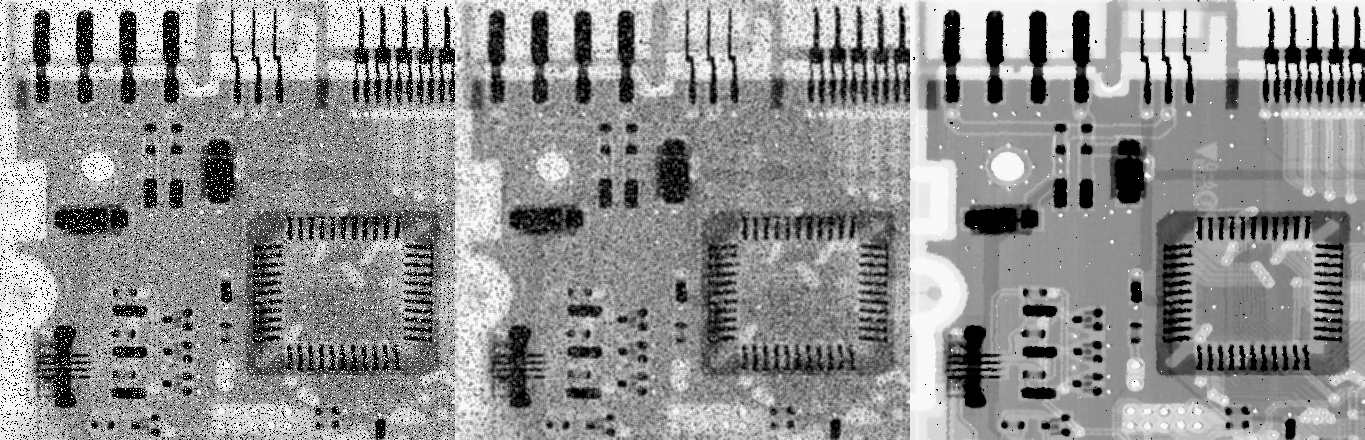
****

图7 pcb从左到右分别为原图，均值滤波，中值滤波

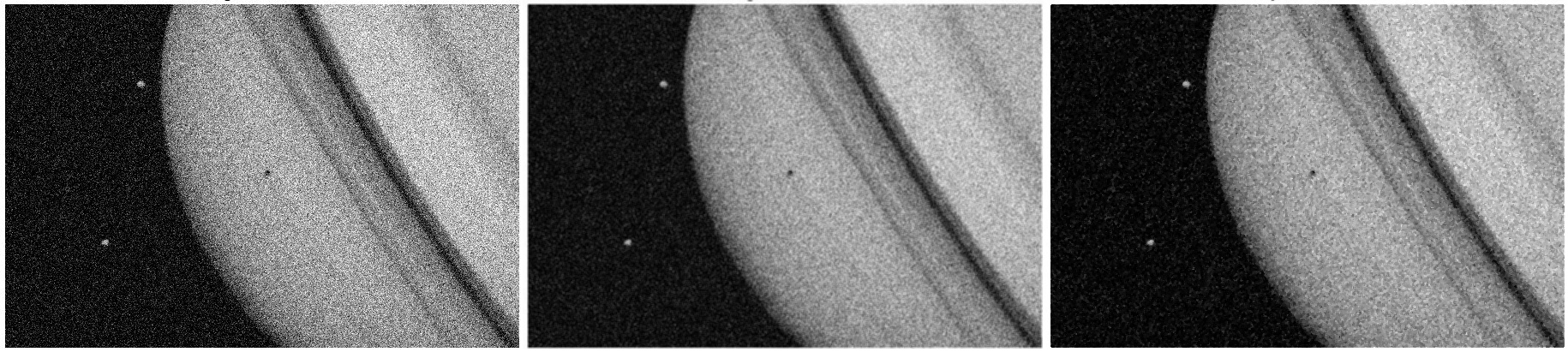
****

图8 space从左到右分别为原图，均值滤波，中值滤波

* 1. **实验小结**

本实验要求是自己实现均值滤波，中值滤波。

均值和中值的两种实现的基本思想过程都是先设定卷积大小（如：3\*3），而后遍历每一个像素点，获取该像素点周围的卷积大小的图片区域，计算其均值/中值，替换到原来的像素位置上。

实验结果显示均值和中值滤波可以使得图片可以消除图片噪点使图片变得更加清晰。其中在pcb中实现的效果最为明显，mona有一定效果，而space中实现的效果不太明显。

1. **任务4**
   1. **核心代码：**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | # 设置拉普拉斯算子  # 中心为-4，四周为1  laplacian **=** np.array([[0, 1, 0],                        [1, **-**4, 1],                        [0, 1, 0]])  **def** get\_filter(img, laplacian, i, j):      ##获取点周围与算子大小对应的图像区域      kernel\_size **=** laplacian.shape[0]      half\_kernel **=** kernel\_size **//** 2      start\_i **=** i **-** half\_kernel      start\_j **=** j **-** half\_kernel      end\_i **=** i **+** half\_kernel **+** 1      end\_j **=** j **+** half\_kernel **+** 1  **return** img[start\_i:end\_i, start\_j:end\_j]  **def** laplacian\_filtering(img, laplacian):      ##使用拉普拉斯算子对图像进行锐化      ##公式: g(x,y) = f(x,y) - ∇²f(x,y) (当算子中心为负时)      ##转换为浮点型进行计算，防止溢出      im\_sharpened **=** np.zeros\_like(img.astype(np.float32))  **for** i **in** range(1, img.shape[0] **-** 1):  **for** j **in** range(1, img.shape[1] **-** 1):  **for** k **in** range(img.shape[2]):                      roi **=** get\_filter(img[:, :, k], laplacian, i, j)                      lap\_val **=** np.sum(roi **\*** laplacian)                      im\_sharpened[i, j, k] **=** img[i, j, k] **-** lap\_val      im\_sharpened **=** np.clip(im\_sharpened, 0, 255).astype(np.uint8)  **return** im\_sharpened |

* 1. **实验结果截图**

**res_blurry_moon_sharpened**

图9 blurry\_moon从左到右分别为原图，自己实现的

* 1. **实验小结**

本实验的要求是自己实现拉普拉斯算子，对图像blurry\_moon进行锐化。

为实现实验，需要自己先设置拉普拉斯算子（本质为一个3\*3的卷积核）。本实验设置的拉普拉斯算子中心为-4，四周为1，因此当实现锐化时，使用公式： g(x,y) = f(x,y) - ∇²f(x,y)进行计算。

实现拉普拉斯算子先设定卷积大小（如：3\*3），而后遍历每一个像素点，获取该像素点周围的卷积大小的图片区域，根据公式计算其值，替换到原来的像素位置上。

实验结果发现图片边界明显变得更加清晰，实现的锐化效果显著。