1.1 直方图均衡化(15 分)
假设您已对数字图像进行直方图均衡。直方图均衡化的第二次通过(在直方图均衡图像上)会产生与第一次传递完全相同的结果吗？证明你的答案。

会的，
直方图均衡化的结果只与灰度值出现的概率有关，所以经过变换之后，灰度值的相对大小不会发生改变，也就是说，当一次均衡化之后，整个图像就被调整成为了“均衡”状态，再次均衡化也不会改变
举例：
令映射是由 r 映射到 s，图像共分为 L 个灰度级别,若原图像灰度级 r 与新图像灰度级 s 为一一对应的关系
则 s= P(r)*(L−1)
那么此时，在新图像中 P(s)=P(r)，对新图像次做直方图均衡化，令映射为 s−>t
则 t=P(s)*(L−1)=P(r)*(L−1)=s
所以第二次直方图均衡化后像素灰度级与第一次相同
接下来看一下原图像中多个灰度级映射到新图像的一个灰度级的情况，令映射为 r1,r2−>s
则 s=(P(r1)+P(r2))*(L−1)
此时在新图像中 P(s)=P(r1)+P(r2),那么对新图像再次做直方图均衡化，令映射为 s−>t
则 t=P(s)*(L−1)=(P(r1)+P(r2))*(L−1)=s
第二次均衡化之后与第一次一样相同
从上面两个例子我们可以看出来，当我们做第一次均衡化过后，新图像中每个灰度值出现的概率等于原图像对应的像素概率(和)，新的灰度值为它之前的灰度值概率求和再乘(L−1)
所以第二次直方图均衡化的时候我们得到的“新灰度值”一样是由同样的概率来进行计算，所以得出来的结果永远与第一次相等
而且不止第二次，第 3，4，…n 次均衡化结果也不会改变
1.2 空间滤波(20 分)
考虑 4×4 灰度图像和 3×3filter：
(1) 将灰度图像与给定的具有零填充的filter 卷积，并显示结果(其大小应为 4×4)(7 分)

原矩阵
[80 10 20 90]
[170 8 240 15].
[20 150 163 40]
[12 34 40 70]

变换因子
[1 1 1]
[0 0 0]
[−1 −1 −1]

既然要卷积，那么就要先把变换因子上下左右进行旋转
变换后的矩阵
[−1 −1 −1]
[0 0 0]
[1 1 1]
卷积的时候要将我们所进行变换的像素点与充当变换因子的矩阵中心对齐
卷积后的结果
[178 418 263 255]
[80 223 233 93]
[−132 −332 −119 −145]
[−170 −333 −353 −203]

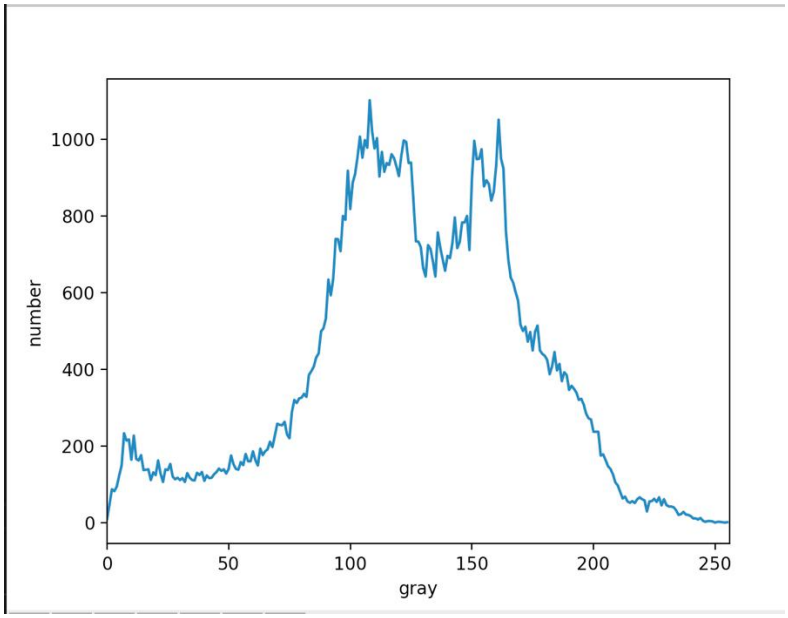
(2)讨论卷积结果中值的意义(5 分)
卷积得到的结果矩阵中的每一个值，都是由原矩阵中对应位置以及其相邻元素的值按照一定的权重综合而成，可以说卷积结果中中矩阵每一个元素都按照变换因子的权重包含了原矩阵对应元素
(3)根据您的知识，讨论给定的filter 的一些应用(3 分)
给定的 filter 有点类似于 Sobel 算子，它可以被用作工业检测中常用的梯度处理，用于检测产品缺陷
(4)如果另一个输入图像和零填充的输入输出为下列，请输入相应的 3×3 大小的数据。(5 分)
通过给出的输入输出，以及卷积因子的形式(还是要先对其进行上下左右旋转)
可得方程组

$$c+b+a=-2$$
$$d+c+b+a=-1$$
$$d+c+a=-2$$
$$c+b+e+a=-1$$
$$a+b+c+d+e=0$$
$$e+d+c+a=-1$$
$$e+c+b=-2$$
$$d+c+b+e=-1$$
$$d+c+e=-2$$

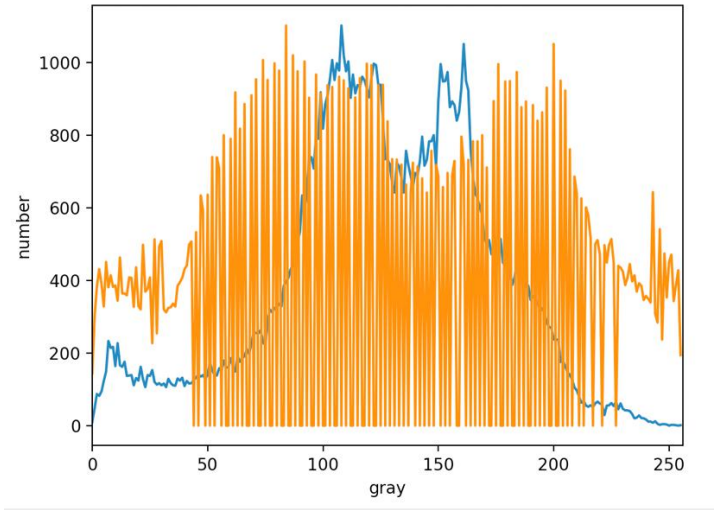
最终解得

$$a=1$$
$$b=1$$
$$c=-4$$
$$d=1$$
$$e=1$$

2.1
写一个在灰度图像上应用直方图均衡化的函数。函数原型是 “equalize_hist(input_img) 返回一个灰度图像，该图像的直方图近似为
对于报告，请加载您的输入图像，并使用您的程序
（1）计算并显示其直方图。在报表上手动粘贴直方图。注意：
您必须自己计算直方图，但是可以使用第三方 API 进行显示。 （5 分）



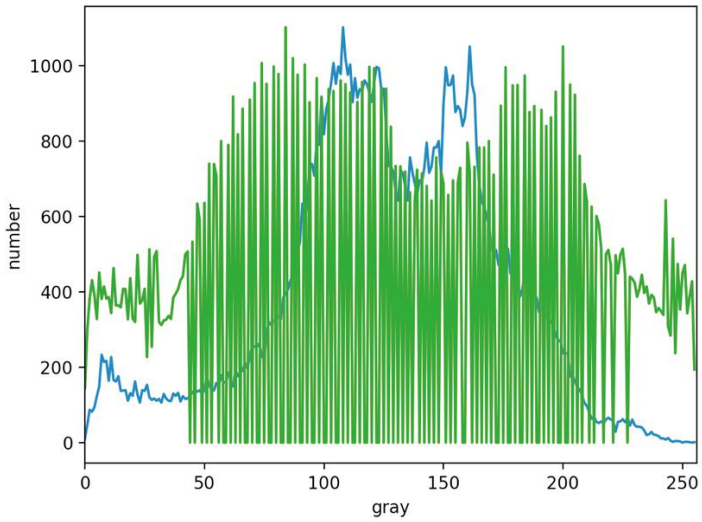
（2）均衡直方图，然后将直方图均衡化后的图像和相应的直方图粘贴到报告中。 （10 分）



蓝色为原图像，橙色为均衡化之后的图像
均衡化之后的图像为

（3）再次均衡直方图，然后将生成的直方图粘贴到报告中。直方图均衡化的第二遍是否会产生与第一遍完全相同的结果？为什么？ （10 分）

完全一样
在此均衡后的直方图为



上面的图像是原图像直方图，两次均衡化之后的直方图，可见两次均衡化之后的直方图完全重合
为此，可以进一步验证两幅图像是否完全相同

```
def dif(image1,image2):
    height = image1.shape[0]
    width = image1.shape[1]
    num = 0
    for i in range(0,height):
        for j in range(0,width):
            if(image1[i,j]!=image2[i,j]):
                num+=1
    return num
```

这个函数用来区分两幅图像不同的像素数目，返回 0 时证明两幅图像完全相同

```
if(dif(b,c)==0):
    print('same')
else:
    print('not same')
```

b,c 为两次均衡化之后的图像
输出结果为

```
dongchenyudeMacBook-Pro:~ dongchenyu$ python3 -u "/Users/dongchenyu/Documents/数字图像处理/hw2.1.py"
same
```

可得两次均衡化之后的图像完全相同
原因在第一题中有所提及
直方图均衡化的结果只与灰度值出现的概率有关，所以经过变换之后，灰度值的相对大小不会发生改变，也就是说，当一次均衡化之后，整个图像就被调整成为了“均衡”状态，再次均衡化也不会改变

（4）请在不到 2 页的时间内详细讨论如何实现直方图均衡操作，即“ equalize_hist”函数。 部分。由于您的验证码也已提交，因此请不要在报告中广泛复制/粘贴您的验证码。 （10 点）

做直方图均衡化，首先要将直方图画出来，画出直方图，就要统计 0—255 各个灰度值出现的次数以及所占据的比例

```
for i in range(h):
    for j in range(w):
        gray = image[i,j]
        num[gray] = num[gray]+1
```

num 是一个长度为 256 的一维数组，num[x]代表灰度为 x 的像素的个数
统计完之后就可以用 python 自带的绘图工具绘制出直方图
均衡化的本质就是建立一个灰度值的映射关系，原图像大的灰度值 x 对应新图像的灰度值 y 二这个映射关系与原图像中每个灰度值所占的比例以及它们的累积有关

```
percent = [0]*256
pdf = [0]*256
trans = [0]*256
```

这三个都是长度为 256 的数组
percent：每个灰度值所占的百分比
pdf：灰度值的累积值
trans：映射关系 trans[1]=2 表示原图灰度值 1 对应新图像灰度值 2
然后求 percent 和 pdf

```
for i in range(0,256):
    percent[i]=num[i]/sum
pdf[0]=percent[0]
for i in range(1,256):
    pdf[i]=pdf[i-1]+percent[i]
```

求出来 pdf 之后根据 pdf 中的值乘上 255，就能得出原图像中的灰度值所对应的新的灰度值，灰度值都是整数，而我们求出来的有可能是小数，所以对其进行四舍五入，python 里面暂时没查到有什么四舍五入的函数，int()函数只是向下兼容，所以在这里我自己操作了一下

```
if(temp-int(temp)>=0.5):
    temp=int(temp)+1
else:
    temp=int(temp)
trans[i]=temp
```

这样就达成了四舍五入的目的
最后再按照映射关系将原图像映射到新图像上

```
for i in range(0,h):
    for j in range(0,w):
        result[i,j]=trans[image[i,j]]
```

空间滤波

编写一个在灰度图像上执行空间滤波的函数。功能原型是

1） 分别使用 3×3、5×5 和 7×7 平均滤波器对输入图像进行平滑处理。将您的三个结果粘贴到报告上。 （9 分）

在这里使用的 3*3，5*5，7*7 的矩阵中的数字全是 1

结果如下(从上到下依次是 3*3 5*5 7*7)

可以看到从上到下图像依次“变糊”了

（2）使用 3×3 拉普拉斯滤波器（使用图 2 中指定的变体）锐化输入图像。

教科书第 3.37（b）条），然后粘贴结果。此外，简述了为什么可以使用拉普拉斯滤波器进行锐化(6 分)

原因：拉普拉斯算子矩阵是[1,1,1][1,−8,1][1,1,1],我们可以看出某个像素点经过拉普拉斯变换之后的值取决于它的值与周围的值的差，也就是它周围的像素到它的变化程度，而如果对于一个像素点，它周围的点的灰度值到它本身变化幅度很大，那么它本身所在的位置就可能是边缘

而通过拉普拉斯算子变换之后，若一个像素点的灰度值低于/高于其周围的像素点，那变换之后这个点的灰度值会被进一步的降低/提高，从而加强了对比度，实现了图像边缘的“锐化“

（3）执行高增益滤波（即 g（x，y）= f（x，y）+ k * g mask（x，y），有关其他详细信息，请参见教科书的方程（3.6-9））在您的输入图像上。该过程的平均部分应使用图 2 中的滤波器完成。 教科书第 3.32（a）条。如图所示，选择一个 k（等式（3.6-9）中的权重）。写下所选的 k，然后将结果粘贴到报告上。 （5 分）

在这里选择 k=2

（4）在不到 2 页的时间内详细讨论如何实现空间过滤操作，即“ filter2d”功能。 （10 点）

均值滤波：

均值滤波用的因子是元素都为 1 的方形矩阵，所以卷积的话免了旋转矩阵那一步了

这里以 5*5 为例

首先要拓展矩阵

5*5 的话需要上下左右各拓展两行/列出来

```
res1 = np.zeros([height+4,width+4],np.uint8)
temp1 = np.zeros([height+4,width+4],np.uint8)
fin1 = np.zeros([height,width],np.uint8)
```

然后将原图像中的像素填充到这个拓展矩阵的中心部分

```
for i in range(2,height+2):
    for j in range(2,width+2):
        temp1[i,j]=image[(i-2),(j-2)]
```

接下来是计算每一个像素点经过矩阵变换之后的灰度值
由于矩阵中所有元素都是 1
所以只要把对应的像素点和变换因子的中心对其，然后把拓展矩阵中每个元素加起来，在除以 25，最后将其转换成整数，这样拓展矩阵的中心就是我们要的灰度值，

```
for i_ in range(i-2,i+3):
    for j_ in range(j-2,j+3):
        result+=image[i_,j_]
result=result/25
result=int(result)
```

最后将其还原出来，防止边缘出现黑色(拓展的 0)

```
for i in range(2,height+2):
    for j in range(2,width+2):
        res1[i,j]=cal_ave55(temp1,filter1,i,j)
        fin1[i-2][j-2]=res1[i,j]
```

fin1 就是最后的结果
滤波大体就是这么个流程，3*3，7*7 跟上面的其实差不多
拉普拉斯锐化跟这个流程也差不多，但是要注意的是按照给出的算子算出来的不是最后的结果，要用原图减去我们得出来的结果，单纯按照给出的拉普拉斯算子算出来的结果相当于给图像进行了“描边”，以及像素点的灰度值会出现负数，我处理的比较粗糙，小于 0 的直接按照 0 去算了
而高提升滤波则需要有模板，在这里用了原图与均值滤波之后的差值作为模板，再用原图加上系数*模板，其中系数大于 1，在这里我取了 2
在这里我对于差值小于 0 的点处理的比较粗糙，小于 0 的我直接按 0 去算了
最后用原图加上模板*系数的时候，大于 255 的我也按照 255 去算了
在这里的处理就不是很细致了
高提升滤波主要涉及的是图像的加减，加法和减法的核心代码如下

```
for i in range(0,height):
    for j in range(0,width):
        res[i,j]=int(img1[i,j])+int(img2[i,j])
        if(res[i,j]>255):
            res[i,j]=255
```

```
for i in range(0,height):
    for j in range(0,width):
        res[i,j]=int(img1[i,j])-int(img2[i,j])
        res[i,j]=int(2*res[i,j])
        if(res[i,j]<0):
            res[i,j]=0
```

减法那里乘 2 我就提前处理了