第一次实验报告：混合图像

姓名：伍锵伊 学号：18030100285 完成日期：2020.9.14

1. 问题描述

创建图像滤波函数，分别对不同的两个图像进行低通和高通滤波，以得到低频平滑图像和高频图像，两个图像相加得到混合图像。高频往往在感知中占主导地位，但在远处，只能看到信号的低频(平滑)部分。通过将一幅图像的高频部分与另一幅图像的低频部分混合，可以得到一幅混合图像，在不同的距离产生不同的解释。本次实验，通过编写图像滤波函数得到混合图像的效果。

1. 程序设计
2. 所编写的主要函数：
   1. Cross\_correlation\_2d():

*def* cross\_correlation\_2d(img, kernel):  
 *if len*(img.shape) == 2:  
 m\_l = kernel.shape[0]  
 n\_l = kernel.shape[1]  
 origin\_x\_l = img.shape[0]  
 origin\_y\_l = img.shape[1]  
 img\_a\_l = np.zeros((origin\_x\_l+*int*((m\_l-1)),origin\_y\_l+*int*((n\_l-1))))  
 *for* i *in range*(origin\_x\_l):  
 *for* j *in range*(origin\_y\_l):  
 img\_a\_l[i+*int*((m\_l-1)/2),j+*int*((n\_l-1)/2)] += img[i,j]  
 x = img\_a\_l.shape[0]  
 y = img\_a\_l.shape[1]  
 G1\_l = np.zeros((x,y))  
 *for* i *in range*(*int*((m\_l-1)/2),x-*int*((m\_l-1)/2)):  
 *for* j *in range*(*int*((n\_l-1)/2),y-*int*((n\_l-1)/2)):  
 *for* a *in range*(0,m\_l):  
 *for* b *in range*(0,n\_l):  
 G1\_l[i,j] += img\_a\_l[i-*int*((m\_l-1)/2)+a,j-*int*((n\_l-1)/2)+b]\*kernel[a,b]  
 G1\_a\_l = np.zeros((x-(m\_l-1),y-(n\_l-1)))  
 *for* i *in range*(x-(m\_l-1)):  
 *for* j *in range*(y-(n\_l-1)):  
 G1\_a\_l[i,j] = G1\_l[i+*int*((m\_l-1)/2),j+*int*((n\_l-1)/2)]  
 *return* G1\_a\_l  
 *elif len*(img.shape) == 3:  
 G1\_s = []  
 img\_b,img\_g,img\_r = cv2.split(img)  
 img\_s = [img\_b,img\_g,img\_r]  
 *for* img\_l *in* img\_s:  
 m = kernel.shape[0]  
 n = kernel.shape[1]  
 origin\_x = img\_l.shape[0]  
 origin\_y = img\_l.shape[1]  
 img\_a = np.zeros((origin\_x + *int*((m - 1)), origin\_y + *int*((n - 1))))  
 *for* i *in range*(origin\_x):  
 *for* j *in range*(origin\_y):  
 img\_a[i + *int*((m - 1) / 2), j + *int*((n - 1) / 2)] += img\_l[i, j]  
 x = img\_a.shape[0]  
 y = img\_a.shape[1]  
 G1 = np.zeros((x, y))  
 *for* i *in range*(*int*((m - 1) / 2), x - *int*((m - 1) / 2)):  
 *for* j *in range*(*int*((n - 1) / 2), y - *int*((n - 1) / 2)):  
 *for* a *in range*(0, m):  
 *for* b *in range*(0, n):  
 G1[i, j] += img\_a[i - *int*((m - 1) / 2) + a, j - *int*((n - 1) / 2) + b] \* kernel[a, b]  
 G1\_a = np.zeros((x - (m - 1), y - (n - 1)))  
 *for* i *in range*(x - (m - 1)):  
 *for* j *in range*(y - (n - 1)):  
 G1\_a[i, j] = G1[i + *int*((m - 1) / 2), j + *int*((n - 1) / 2)]  
 G1\_s.append(G1\_a)  
 G1\_1 = cv2.merge([G1\_s[0],G1\_s[1],G1\_s[2]])  
 *return* G1\_1  
  
 *# TODO-BLOCK-BEGIN  
 raise Exception*("TODO in hybrid.py not implemented")  
 *# TODO-BLOCK-END*

* 1. Convolve\_2d():

*def* convolve\_2d(img, kernel):  
 kernel\_y = np.flipud(kernel)  
 nkernel = np.fliplr(kernel\_y)  
 G2\_a = cross\_correlation\_2d(img,nkernel)  
 *return* G2\_a  
 *# TODO-BLOCK-BEGIN  
 raise Exception*("TODO in hybrid.py not implemented")  
 *# TODO-BLOCK-END*

* 1. Gaussian\_blur\_kernel\_2d():

*def* gaussian\_blur\_kernel\_2d(sigma, height, width):  
 G\_sigma = np.zeros((height,width))  
 center\_width = *int*(math.floor(width/2))  
 center\_height = *int*(math.floor(height/2))  
 *for* i *in range*(height):  
 *for* j *in range*(width):  
 x = *abs*(j-center\_width)  
 y = *abs*(i-center\_height)  
 coe = 1/(2\*math.pi\*(sigma\*\*2))  
 index = -(x\*\*2+y\*\*2)/(2\*(sigma\*\*2))  
 G\_sigma[i,j] = coe\*math.exp(index)  
 k = G\_sigma.sum()  
 G\_sigma = G\_sigma / k  
 *return* G\_sigma  
  
 *# TODO-BLOCK-BEGIN  
 raise Exception*("TODO in hybrid.py not implemented")  
 *# TODO-BLOCK-END*

* 1. Low\_pass():

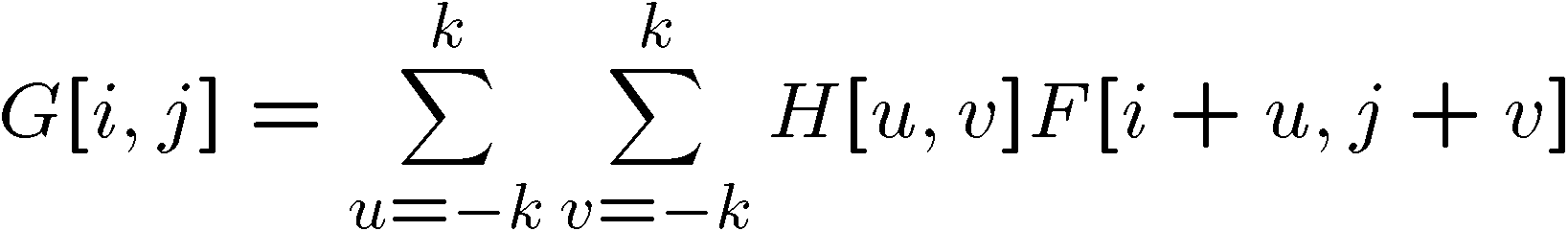
*def* low\_pass(img, sigma, size):  
 kernel = gaussian\_blur\_kernel\_2d(sigma, size, size)  
 *if len*(img.shape) == 3:  
 img\_b,img\_g,img\_r = cv2.split(img)  
 low\_pass\_img\_b = convolve\_2d(img\_b,kernel)  
 low\_pass\_img\_g = convolve\_2d(img\_g, kernel)  
 low\_pass\_img\_r = convolve\_2d(img\_r, kernel)  
 low\_pass\_img\_1 = cv2.merge([low\_pass\_img\_b,low\_pass\_img\_g,low\_pass\_img\_r])  
 *return* low\_pass\_img\_1  
  
 *elif len*(img.shape) == 2:  
 low\_pass\_img = convolve\_2d(img,kernel)  
 *return* low\_pass\_img  
  
 *# TODO-BLOCK-BEGIN  
 raise Exception*("TODO in hybrid.py not implemented")  
 *# TODO-BLOCK-END*

* 1. High\_pass():

*def* high\_pass(img, sigma, size):  
 high\_pass\_img = img - low\_pass(img,sigma,size)  
 *return* high\_pass\_img  
  
 *# TODO-BLOCK-BEGIN  
 raise Exception*("TODO in hybrid.py not implemented")  
 *# TODO-BLOCK-END*

1. 详细算法描述
2. Cross\_correlation\_2d()函数

互相关的定义由下列计算式给出：

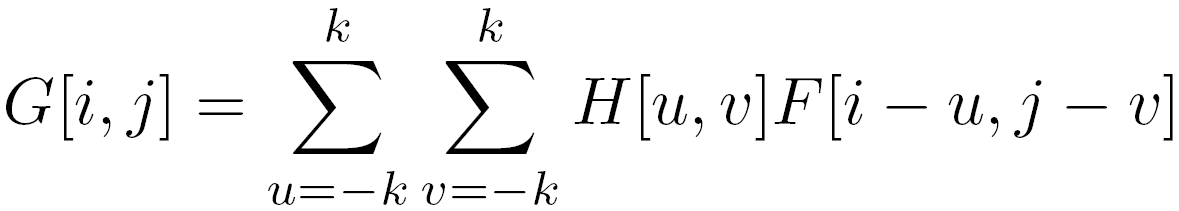


其中，H表示权重核，u,v表示权重核的大小，F是边缘补零后的图像，G表示相关后的输出图像。

函数采用上述计算式的特点，运用四次循环进行加和计算，外两层循环表示权重核在图像上的移动，内两层循环表示核与图像的每次互相关，在循环中，值得注意的是，图像矩阵与核矩阵x,y坐标的差异，需要通过核的大小确定互相关中心的实际位置，互相关操作后，得到的输出图像仍为扩充补零后的图像，需要单独将原始图像大小的输出图像从中取出，最终得到的输出才是互相关的最终输出图像。函数分为两部分，第一部分处理单通道灰度图像，第二部分处理三通道彩色图像，彩色图像的处理与灰度图像处理相似，只需将彩色图像分为三个灰度图像即可，分别处理后再重新融合。

1. Convolve\_2d()函数

卷积与互相关极为相似，卷积计算式如下：

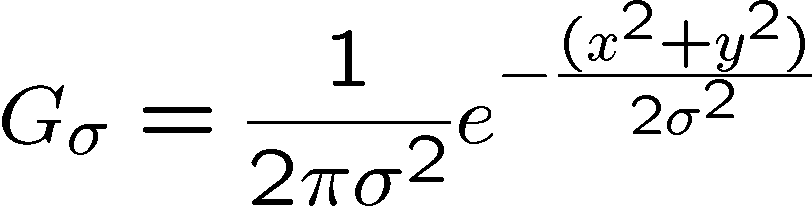


卷积实际意义上，只需要将互相关的权重核进行翻转，便可以得到卷积的输出图像，而互相关函数也是为卷积函数做铺垫。

利用NumPy库中的函数对权重核进行翻转，然后调用互相关函数，实现卷积操作。

1. Gaussian\_blur\_kernel\_2d()函数

高斯滤波是一种特殊的加权平均，其计算式如下：



计算式中，x,y分别表示像素点里核中心的垂直距离与水平距离，通过调用math库中的函数，可以进行指数计算。值得注意的是，在计算得出输出高斯核后，需要对高斯核进行归一化处理。

1. Low\_pass()函数

低通滤波简单来说，就是高斯核与图像的卷积，通过高斯滤波函数计算得到高斯核，以高斯核为权重核，调用卷积函数将核与图像进行卷积，得到低通滤波图像。

函数分为两部分，第一部分是对三通道彩色图像的处理，第二部分是对单通道灰度 图像的处理。

1. High\_pass()函数

高通滤波图像就是原图像与低通滤波图像的差值。

1. 调试分析

sigma = 20 kernel = 13 running time约等于 8 分钟

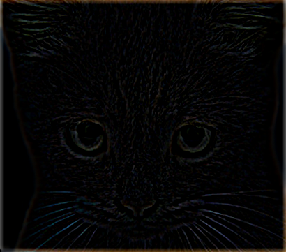
本次实验中，分为两部分python文件，一个为hybrid.py文件，用来编写所有图像处理的相关函数，一个为MainImage.py文件，用来调用相关函数，进行实际的图像处理。本次实验测试中，我给定了sigma和kernel的值大小，sigma = 20，kernel = 13，实验效果明显，图像混合效果较好，但实验编译时间很长，执行一次代码时间在8分钟左右。当改变kernel值时，kernel值越小，混合效果越差。test.py文件为老师给的函数测试文件，编译皆通过无误。

1. 测试结果

low\_pass\_img:



high\_pass\_img:



hybrid\_img:



1. 心得体会

本次实验中，我们被要求禁止使用opencv库中指定函数，所以在这次实验过程中，我详细编写并推敲了滤波函数的每一步原理，也在编译错误中，一步步改进代码，仔细思考了原理的实现机制和方法，我收获颇多，但本次实验中，代码优化效果较差，编译代码的时间较长，时间复杂度较高，且尚未找到合适的简化思路，之后，我会进一步思考计算方式，优化算法，降低时间复杂度，提高编译效率。