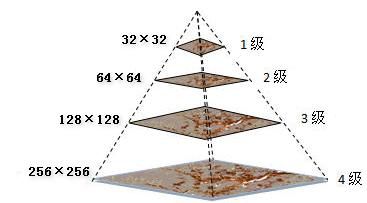
高斯与拉普拉斯金字塔

**一 实验原理**

**一.一** **图像金字塔**

图像金字塔是为了以多分辨率来解释图像而诞生的一种方法。一幅图像的金字塔是以一系列以金字塔形状排列的分辨率初步降低的图像的集合。如图一所示：



图一：图像金字塔

金字塔有如下的特征：

1.金字塔的底部是待处理图像的高分辨率的表示，顶部是低分辨率的表示。一般来说最下方也就是第零层是原始图片。

2.当向上层移动的时候，尺寸和分辨率都会降低。在本次实验中使用二倍下采样，即图片边长缩小为原来的二分之一。

3.金字塔不停迭代使边长不断减半，当得到图像时迭代终止，得到所有层图像，即一个完整的金字塔。

**一.二 高斯金字塔**

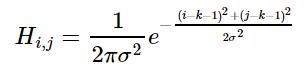
在金字塔向上层迭代的过程中一共有两步，第一步对图像进行一次滤波，第二步删去图像的偶数行和列从而使图像尺寸减半。这里应注意要先滤波再降采样，因为降采样会减少信息，如果先降采样再滤波那么滤波得到的信息就不是原层图像的信息了。在第一步中，如果使用一个高斯核对图像滤波，那么最后得到的金字塔就是高斯金字塔。

在图像高斯平滑中，对图像进行平均时不同位置的像素被赋予了不同的权重。这些权重值的分布符合高斯分布。高斯分布相比均匀滤波能够更好的保存图像特征。图二上为一个3×3的高斯模板。不过在本次实验中为了进行拉普拉斯图与高斯图的还原，全程使用float64格式保存数据以保有更多细节，所以使用了浮点数形式的高斯模板如图二下。

下面说明一下高斯滤波器中对高斯分布（正态分布）的使用。一个二维的高斯函数如下：



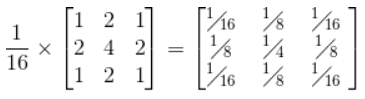
其中为点的坐标，在图像处理中认为是整数；是标准差。对高斯函数进行离散化，得到的高斯函数值作为模板的系数。当窗口模板的大小为 (2k+1)×(2k+1)，模板中各个元素值的计算公式如下：



这样计算出来的模板有两种形式：小数和整数。

小数形式的模板，就是直接计算得到的值没有经过处理，即图二下。

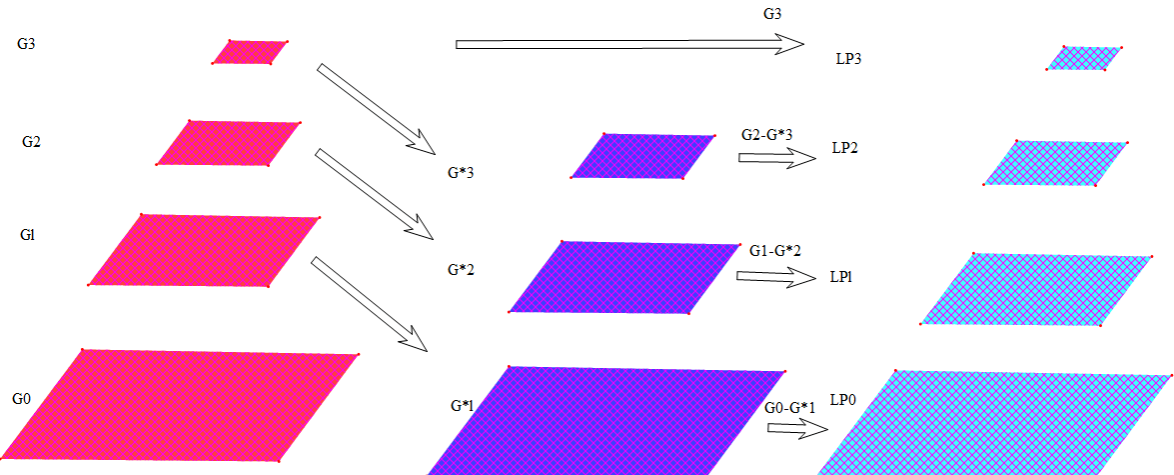
整数形式的需要进行归一化处理，将模板左上角的值归一化为1，使用整数模板时需要在模板的前加一系数，系数为模板所有值的和的倒数如图二上。高斯金字塔的整体效果见图四下。



图二：一个3×3的高斯模板

**一.三 拉普拉斯金字塔**

拉普拉斯金字塔的生成与高斯金字塔相关，其具体操作如图三。对于层的拉普拉斯金字塔，先将层的高斯金字塔插值至层高斯金字塔的大小，然后做一个高斯滤波，再用层的高斯金字塔减去滤波后的图像，得到层的拉普拉斯金字塔。由于高斯金字塔的每层拥有不同频率的信息，当一层减去下一层时，相当于一个低通分量减去一个更高层的低通分量，所得的结果就是带通滤波，这也是为什么拉普拉斯金字塔又称为带通金字塔的原因。



图三：拉普拉斯金字塔

**一.四 重建**

由于拉普拉斯金字塔的每一层都是相邻层高斯金字塔相减得到的，所以可以通过高斯金字塔和拉普拉斯金字塔中某一层的相加得到更高分辨率的高斯金字塔。其大致效果如图四上，图四下是高斯金字塔，图四下最左侧图为第零层高斯图即为原图。





图四：重建图与高斯图

**一.五 负值处理**

因为拉普拉斯金字塔是高斯金字塔相减得到，所以拉普拉斯图像的像素值并不总是正数。由于在图像重建的过程中应尽量保持所有信息，故不对负数进行处理，而在显示拉普拉斯金字塔时往往难以显示负数像素和大于255的像素，需要将数值范围规范化。

对于小于零的部分有两种处理方式。第一种是直接置为0，这种方法会导致拉普拉斯图亮度非常低，如图五上。第二种方法是检测图像中像素的最小值，若是负数，则所有像素加上该负数的绝对值，这种方法能够得到亮度适中的拉普拉斯图，如图五下。





图五：置零与加绝对值

对于大于255的部分也有两种方式，其一是截断，将大于255的值置为255。其二是按比例缩放，即将最大值置为255，其余所有值按照该比例等比缩小或放大。方法一见图六上，方法二见图六下。这两种方法没有明显差别，原因可能是在高斯图的运算过程中高斯核相加为1，保持能量不变，故没有大于255的值，同时相减也可能将大于255的值减至255以下。





图六：截断与按比例缩放

**二 结果展示**

**原图：**



**高斯金字塔：**

这里金字塔有7层，由于第零层为原图共八个图像。



**拉普拉斯金字塔：**



**拉普拉斯金字塔显示版：**



**重建：**





**三 分析**

为了更好的测试实验效果，我选取了视频11分钟时猫的图片，截图作为原始图。

之前的叙述中都使用了彩色版本，灰度图版本如图七。从上到下依次为重建图、高斯金字塔、拉普拉斯金字塔、负值处理后的拉普拉斯金字塔。



图七：灰度图下的金字塔

可以看到在边缘部分亮度稍微变亮，这是由于在边缘处理时采用了镜像对称的方法。当使用加权方法时不会出现这种现象如图八。从上到下依次为彩色图的重建图、高斯金字塔、拉普拉斯金字塔、负值处理后的拉普拉斯金字塔。以及灰度图的上述图像。





图八：使用加权方法处理边界的图像金字塔。

**四 代码**

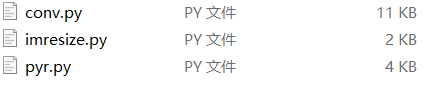
代码共分为三个部分如图九，使用python编程。

依赖库：

Numpy 数据存储与操作

Imageio 图像读取

matplotlib.pyplot 图像显示



图九：代码结构

第一个是pyr.py，它是主要函数部分，主要提供以下函数：

gskernel(size,sigma=1.0)，获得一个高斯核，默认sigma为1。

getGpyr(im)获得一个图片的高斯金字塔。

lpyr(gsim)获得一个图片的拉普拉斯金字塔。

beforeshow(im)将拉普拉斯金字塔进行负值和过255值处理。

listImShow(im)显示list类型保存的图像序列。

第二个是conv.py（第三次作业），它提供卷积功能，函数如下：

Convolve(I, F,way=2,iw=0, ih=0, fw=0, fh=0)卷积，第一个参数为图像，第二个参数为卷积核，第三个参数为边缘处理方法，1为置零，2为镜像，3为加权，由于加权方法对含有负值的卷积核难以解释正确性，故默认使用第二种方法。为了拥有更大的泛用性，这里对偶数长度卷积核和奇数长度卷积核进行了分别操作，同时采用了三种边缘处理方法，导致代码较长请见谅。

show(im)显示一张图片（自动判断灰度或彩色）

第三个是imresize.py（第一次作业），负责对图像进行上采样（双线性插值）

imresize(ori\_im, tar\_size)第一个参数为图像，第二个参数为目标尺寸。

**代码pyr.py**

import numpy as np

import conv

import imageio

import imresize

import matplotlib.pyplot as plt

def listImShow(im):

plt.figure()

for i in range(1,len(im)+1):

plt.subplot(1,len(im),i)

if len(np.shape(im[0]))==2:

plt.imshow(im[i-1].astype(np.int),cmap=plt.cm.gray)

else:

plt.imshow(im[i-1].astype(np.int))

plt.xticks([])

plt.yticks([])

plt.show()

plt.close()

def listImWrite(im,path,t='\_'):

for i in range(len(im)):

# print(path+t+str(i)+'.jpg')

imageio.imwrite(path+t+str(i)+'.jpg', im[i].astype(np.uint8))

def gskernel(size,sigma=1.0):

gskernel=np.zeros((size,size),np.float64)

for i in range (size):

for j in range (size):

norm=pow(i-1,2)+pow(j-1,2)

gskernel[i,j]=np.exp(-norm/(2\*pow(sigma,2)))

s=np.sum(gskernel)

kernel=gskernel/s

return kernel

def gdown(im):

for i in range(int(im.shape[0]/2)):

for j in range(int(im.shape[1]/2)):

im[i][j]=im[2\*i][2\*j]

return im

#输入k层高斯，返回k+1层高斯

def gup(im):

F=gskernel(3,1)

# for i in F:

# print(i)

t=conv.Convolve(im,F)#高斯滤波

t=gdown(t)#下采样

return t[0:int(im.shape[0]/2),0:int(im.shape[1]/2)]

def getGpyr(im):

gsim=[]

gsim.append(im.astype(np.uint8))

tim=im

n=0

while tim.shape[0]>1:

tim=gup(tim)

gsim.append(tim)

n+=1

return gsim

def lpyr(gsim):

lplcim=[]

for i in range(len(gsim)-1):

# t=gsim[i]-imresize.imresize(gsim[i+1],gsim[i].shape)

t=gsim[i]-conv.Convolve(imresize.imresize(gsim[i+1],gsim[i].shape),gskernel(3,1),3)

# tmin=t.min()

# t=t+abs(tmin)

# tmax=t.max()

# t=t/tmax\*255.0

lplcim.append(t)

return lplcim

def yanzheng(g,l):

huanyuan=[]

for i in range(len(l)):

t=l[i]+conv.Convolve(imresize.imresize(g[i+1],g[i].shape),gskernel(3,1),3)

huanyuan.append(t)

listImShow(huanyuan)

return huanyuan

def beforeshow(im):

for i in range(len(im)):

tmin=im[i].min()

# if tmin<0:

# for j in range(len(im[i])):

# for k in range(len(im[i][j])):

# for l in range(len(im[i][j][k])):

# if im[i][j][k][l]<0:

# im[i][j][k][l]=0

im[i]=im[i]+abs(tmin)

tmax=im[i].max()

if tmax>255:

im[i]=im[i]/tmax\*255.0

return im

if \_\_name\_\_=='\_\_main\_\_':

path='C:/Users/19041/Desktop/图像处理/图像处理作业/实验一/'

name='1.jpg'

im = imageio.imread(path+name,as\_gray=1).astype('float64')

g=[]

l=[]

huanyuan=[]

lshow=[]

g=getGpyr(im)

l=lpyr(g)

listImWrite(g,path,'g')

listImWrite(l,path,'l')

huanyuan=yanzheng(g,l)

listImShow(g)

listImShow(l)

lshow=beforeshow(l)

listImShow(lshow)

listImWrite(l,path,'showl')

listImWrite(huanyuan,path,'reBuild')

**代码conv.py**

import numpy as np

import matplotlib.pyplot as plt

import imageio

def show(im):

#通过判断形状控制是否灰度显示

if len(im.shape) == 2:

plt.ion()

plt.imshow(im,cmap='gray')

plt.show()

plt.close()

else:

plt.ion()

plt.imshow(im)

plt.show()

plt.close()

return 0

def getFmid(F,h,w,i,j):

tF=np.zeros(F.shape).astype('float64')

fh=F.shape[0]

fw=F.shape[1]

bh=int((fh-1)/2)

bw=int((fw-1)/2)

#对行进行重新加权

if i-bh<0:

up=abs(i-bh)

# print(up)

for tj in range(fw):

tsall=F[:,tj].sum()

tstemp=F[up:fh,tj].sum()

#计算

# print('局部和',tsall,tstemp)

for ti in range(fw):

if ti>=up:

if tstemp!=0:

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

else:

tF[ti][tj]=0

else:

tF[ti][tj]=0

F=tF

if i+bh>h-1:

down=fh-(i+bh-h+1)-1

#print(down)

for tj in range(fw):

tsall=F[:,tj].sum()

tstemp=F[0:down+1,tj].sum()

#计算

# print('局部和',tsall,tstemp)

for ti in range(fw):

if ti<=down:

if tstemp!=0:

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

else:

tF[ti][tj]=0

else:

tF[ti][tj]=0

F=tF

if j-bw<0:

# print(tF)

l=abs(j-bw)

for ti in range(fh):

tsall=sum(F[ti])

tstemp=F[ti][l:fw].sum()

#计算

for tj in range(fw):

if tj>=l:

if tstemp!=0:

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

else:

tF[ti][tj]=0

else:

tF[ti][tj]=0

#列加权

F=tF

if j+bw>w-1:

r=fw-(j+bw-w+1)-1

# print(r)

for ti in range(fh):

tsall=sum(F[ti])

tstemp=F[ti,0:r+1].sum()

#计算

for tj in range(fw):

if tj<=r:

if tstemp!=0:

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

else:

tF[ti][tj]=0

else:

tF[ti][tj]=0

#列加权

F=tF

return tF

#由于只在边界生效，所以当输入不在边界时认为产生错误调用返回全空

def getF(F,h,w,i,j):

tF=np.zeros(F.shape).astype('float')

fh=F.shape[0]

fw=F.shape[1]

#对行进行重新加权

if j-fw<-1:

l=fw-j-1

for ti in range(fh):

tsall=sum(F[ti])

tstemp=0.0

for tstj in range(l,fw):

tstemp+=F[ti][tstj]

#计算

for tj in range(l,fw):

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

#列加权

F=tF

#print(F)

if i+fh>h:

down=fh-(i+fh-h)

#print(down)

for tj in range(fw):

tsall=0.0

tstemp=0.0

for tsti in range(fh):

tsall+=F[tsti][tj]

for tsti in range(down):

tstemp+=F[tsti][tj]

#计算

#print('局部和',tsall,tstemp)

for ti in range(fw):

if ti<down and tj+j-(fw-1)>=0:

tF[ti][tj]=tsall/tstemp\*F[ti][tj]

else:

tF[ti][tj]=0

return tF

def Convolve(I, F,way=2,iw=0, ih=0, fw=0, fh=0):

iw=I.shape[1]

ih=I.shape[0]

fw=F.shape[1]

fh=F.shape[0]

O=np.zeros(I.shape).astype(np.float64)

t=np.zeros(I[0][0].shape)

if F.shape[0]%2==0:

# print(I.shape,F.shape,way)

#第一种

if way==1:

for i in range(ih):

for j in range(iw):

t=0

for fi in range(fw):

for fj in range(fh):

if i+fi<ih and i+fi>-1 and \

j-fw+1+fj<iw and j-fw+1+fj>-1:

t+=I[i+fi][j-fw+1+fj]\*F[fi][fj]

else:

t+=0

O[i][j]=t

#第二种

if way==2:

for i in range(ih):

for j in range(iw):

t=0

for fi in range(fh):

for fj in range(fw):

#若行爆

if i+fi>=ih:

ti=(ih-1)-abs(ih-1-i-fi)

else:

ti=i+fi

#若列爆

if j-fw+1+fj<0:

tj=abs(j-fw+1+fj)

else:

tj=j-fw+1+fj

t+=I[ti][tj]\*F[fi][fj]

O[i][j]=t

#第三种

if way==3:

for i in range(ih):

for j in range(iw):

t=0

#若在边界

if i+fh-1>ih-1 or j-(fw-1)<0:

tF=getF(F,ih,iw,i,j,)

for fi in range(fh):

for fj in range(fw):

if i+fi<ih and j-fw+1+fj>-1:

t+=I[i+fi][j-fw+1+fj]\*tF[fi][fj]

else:

for fi in range(fh):

for fj in range(fw):

t+=I[i+fi][j-fw+1+fj]\*F[fi][fj]

O[i][j]=t

else:

#中心对齐

bh=int((fh-1)/2)

bw=int((fw-1)/2)

tshape=list(I.shape)

tshape[0]=ih+fh-1

tshape[1]=iw+fw-1

Y = np.zeros(tshape).astype(np.float64)

Y[bh:ih+bh,bw:iw+bw]=I

if way==1:

if len(tshape)==3:

s=np.zeros((3))

for i in range(O.shape[0]):

for j in range(O.shape[1]):

s[0]=(Y[i:i+fh,j:j+fw,0]\*F).sum()

s[1]=(Y[i:i+fh,j:j+fw,1]\*F).sum()

s[2]=(Y[i:i+fh,j:j+fw,2]\*F).sum()

O[i][j] = s

else:

for i in range(O.shape[0]):

for j in range(O.shape[1]):

O[i][j]=(Y[i:i+fh,j:j+fw]\*F).sum()

#第二种

if way==2:

for i in range(bh):

Y[i,0:-1]=Y[2\*bh-1-i,0:-1]

Y[ih+i,0:-1]=Y[ih-1-i,0:-1]

for i in range(bw):

Y[0:-1,i]=Y[0:-1,2\*bh-1-i]

Y[0:-1,iw+i]=Y[0:-1,iw-1-i]

if len(tshape)==3:

s=np.zeros((3))

for i in range(O.shape[0]):

for j in range(O.shape[1]):

s[0]=(Y[i:i+fh,j:j+fw,0]\*F).sum()

s[1]=(Y[i:i+fh,j:j+fw,1]\*F).sum()

s[2]=(Y[i:i+fh,j:j+fw,2]\*F).sum()

O[i][j] = s

else:

for i in range(O.shape[0]):

for j in range(O.shape[1]):

O[i][j]=(Y[i:i+fh,j:j+fw]\*F).sum()

#第三种

if way==3:

if len(tshape)==3:

s=np.zeros((3))

for i in range(ih):

for j in range(iw):

#若在边界

if i-bh<0 or i+bh>ih-1 or j-bw<0 or j+bw>iw-1 :

tF=getFmid(F,ih,iw,i,j,)

s[0]=(Y[i:i+fh,j:j+fw,0]\*tF).sum()

s[1]=(Y[i:i+fh,j:j+fw,1]\*tF).sum()

s[2]=(Y[i:i+fh,j:j+fw,2]\*tF).sum()

else:

s[0]=(Y[i:i+fh,j:j+fw,0]\*F).sum()

s[1]=(Y[i:i+fh,j:j+fw,1]\*F).sum()

s[2]=(Y[i:i+fh,j:j+fw,2]\*F).sum()

O[i][j] = s

else:

for i in range(ih):

for j in range(iw):

#若在边界

if i-bh<0 or i+bh>ih-1 or j-bw<0 or j+bw>iw-1 :

tF=getFmid(F,ih,iw,i,j,)

s=(Y[i:i+fh,j:j+fw]\*tF).sum()

else:

s=(Y[i:i+fh,j:j+fw]\*F).sum()

O[i][j] = s

# O=O.clip(0,255)

return O.astype(np.float64)

def getim():

m=4

n=4

t=np.zeros((m,n)).astype('uint8')

for i in range(m):

for j in range(n):

t[i][j]=i+j

# if i == int(m/2):

# t[i][j]=127

# else:

# t[i][j]=30

print(t)

return t

if \_\_name\_\_=='\_\_main\_\_':

path='C:/Users/19041/Desktop/图像处理/图像处理作业/图像处理练习3/'

im=imageio.imread(path+'1.jpg')

F1=np.array(

[

[1/4,1/4],

[1/4,1/4]

]

)

F2=np.array(

[

[1/9,1/9,1/9],

[1/9,1/9,1/9],

[1/9,1/9,1/9]

]

)

F3=np.array(

[

[-1/8,-1/8,-1/8],

[-1/8,1,-1/8],

[-1/8,-1/8,-1/8]

]

)

F4=np.array(

[

[-1,-1,-1],

[-1, 8,-1],

[-1,-1,-1]

]

)

F=F2#调整这里选择核

#im = cv2.cvtColor(im,cv2.COLOR\_BGR2GRAY)#控制转变为灰度图

#im=getim()#调整这里获得构造图像

show(im)

#opencv

# tcv=cv2.filter2D(im,-1,F)

# print('cv2:')

#print(tcv)

# show(tcv)

#我的程序

way=2#调整这里选择边缘处理方式，默认为第二种

t=Convolve(im,F,way)

#print(t.shape)

print('my')

#print(t)

show(t)

print('相减')

# show(t-tcv)

#保存

iname='1\_'+str(way)+'.jpg'

imageio.imwrite(path+iname,t)

inamecv='1\_'+'cv.jpg'

# imageio.imwrite(path+inamecv,tcv)

**代码imresize.py**

import numpy as np

def imresize(ori\_im, tar\_size):

ori\_size=np.shape(ori\_im)

tar\_im=np.zeros(tar\_size)

ratio\_i=tar\_size[0]/ori\_size[0]

ratio\_j=tar\_size[1]/ori\_size[1]

for i in range(len(tar\_im)):

for j in range(len(tar\_im[i])):

ori\_i=i/ratio\_i

ori\_j=j/ratio\_j

x1=int(ori\_i)

y1=int(ori\_j)

if x1==ori\_i and y1==ori\_j:

tar\_im[i][j]=ori\_im[x1][y1]

continue

b=ori\_i-x1

a=ori\_j-y1

t1=(1-a)\*(1-b)\*ori\_im[x1][y1]

t2=a\*(1-b)\*ori\_im[x1][y1] if y1+1>=ori\_size[1] else a\*(1-b)\*ori\_im[x1][y1+1]

t3=(1-a)\*b\*ori\_im[x1][y1] if x1+1>=ori\_size[0] else (1-a)\*b\*ori\_im[x1+1][y1]

t4=a\*b\*ori\_im[x1][y1] if x1+1>=ori\_size[0] or y1+1>=ori\_size[1] else a\*b\*ori\_im[x1+1][y1+1]

tar\_im[i][j]=t1+t2+t3+t4

return tar\_im.astype('float64')

def main(path\_work,num\_im,big\_ratio,small\_ratio,\*arg):

for i in range(num\_im):

im = imageio.imread(path\_work+eval("arg["+str(i)+"]"))

ori\_size=im.shape

out\_im=imresize(im,ori\_size,[int(ori\_size[0]\*big\_ratio),int(ori\_size[1]\*big\_ratio)])

imageio.imwrite(path\_work+"out\_big\_"+eval("arg["+str(i)+"]"), out\_im)

out\_im=imresize(im,ori\_size,[int(ori\_size[0]\*small\_ratio),int(ori\_size[1]\*small\_ratio)])

imageio.imwrite(path\_work+"out\_small\_"+eval("arg["+str(i)+"]"), out\_im)

if \_\_name\_\_ == '\_\_main\_\_':

main('C:/Users/19041/Desktop/图像处理/图像处理练习1/',2,1.5,0.75,'CARTOON.jpg','flowergray.jpg')