opencv学习笔记python实现 图像金字塔（高斯金字塔与拉普拉斯金字塔）

发布于2018-09-28 21:36:06阅读 2.3K0

使用拉普拉斯金字塔时，图像必须是2^n\*2\*m

使用拉普拉斯金字塔先要知道高斯金字塔

;

这两种过程是图片缩小与放大

缩小   reduce =  高斯模糊 + 降采样（pyrDown）

放大    expand = 扩大（升采样/pyrUp） + 卷积

#-\*-coding:utf-8 -\*-

import cv2 as cv

#高斯金字塔

def pyramid\_image(image):

cv.imshow("yuan",image)

level = 3#金字塔的层数

temp = image.copy()#拷贝图像

pyramid\_images = []

for i in range(level):

dst = cv.pyrDown(temp)

pyramid\_images.append(dst)

cv.imshow("pyramid"+str(i), dst)

temp = dst.copy()

return pyramid\_images

#拉普拉斯金字塔

def lpls\_image(image):

pyramid\_images = pyramid\_image(image)

level = len(pyramid\_images)

for i in range(level-1, -1, -1):#数组下标从0开始 i从金字塔层数-1开始减减

if (i-1)<0:#原图

expand = cv.pyrUp(pyramid\_images[i])

lpls = cv.subtract(image, expand)

cv.imshow("lpls\_%s" % i, lpls)

else:

expand = cv.pyrUp(pyramid\_images[i])

lpls = cv.subtract(pyramid\_images[i-1], expand)

cv.imshow("lpls\_%s" % i, lpls)

img = cv.imread("d://work//1.jpg")

lpls\_image(img)

cv.waitKey(0)

cv.destroyAllWindows()

# 高斯金字塔和拉普拉斯金字塔

# 一、图像金字塔原理

图像缩小（先高斯模糊，再降采样，需要一次次重复，不能一次到底）

图像扩大（先扩大，再卷积或者使用拉普拉斯金字塔）

图像金字塔是图像中多尺度表达的一种，最主要用于图像的分割，是一种以多分辨率来解释图像的有效但概念简单的结构。简单来说，图像金字塔就是用来进行图像缩放的。

进行图像缩放可以用图像金字塔，也可以使用resize函数进行缩放，后者效果更好。这里只是对图像金字塔做一些简单了解。

# 二、金字塔类型：高斯和拉普拉斯

①高斯金字塔：用于下采样。高斯金字塔是最基本的图像塔。  
原理：首先将原图像作为最底层图像G0（高斯金字塔的第0层），利用高斯核（5\*5）对其进行卷积，然后对卷积后的图像进行下采样（去除偶数行和列）得到上一层图像G1，  
将此图像作为输入，重复卷积和下采样操作得到更上一层图像，反复迭代多次，形成一个金字塔形的图像数据结构，即高斯金字塔。

②拉普拉斯金字塔：用于重建图像，也就是预测残差，对图像进行最大程度的还原。比如一幅小图像重建为一幅大图，  
原理：用高斯金字塔的每一层图像减去其上一层图像上采样并高斯卷积之后的预测图像，得到一系列的差值图像即为 LP 分解图像。

# 三、采样类型：上采样和下采样

①上采样：就是图片放大（所谓上嘛，就是变大），使用PryUp函数。   
上采样步骤：先将图像在每个方向放大为原来的两倍，新增的行和列用0填充，再使用先前同样的内核与放大后的图像卷积，获得新增像素的近似值。

②下采样：就是图片缩小（所谓下嘛，就是变小），使用PryDown函数。下采样将步骤：先对图像进行高斯内核卷积 ，再将所有偶数行和列去除。

总之，上、下采样都存在一个严重的问题，那就是图像变模糊了，因为缩放的过程中发生了信息丢失的问题。要解决这个问题，就得用拉普拉斯金字塔。

# 四、使用高斯金字塔实现下采样****pyrDown****

## 1、函数

**pyrDown降采样**

def pyrDown(src, dst=None, dstsize=None, borderType=None): # real signature unknown; restored from \_\_doc\_\_ --->dst

opencv的pyrDown函数**先对图像进行高斯平滑，然后再进行降采样**（将图像尺寸行和列方向缩减一半）。

src参数表示输入图像。

dst参数表示输出图像，它与src类型、大小相同。

dstsize参数表示降采样之后的目标图像的大小。

* 它是有默认值的，如果我们调用函数的时候不指定第三个参数，那么这个值是按照  Size((src.cols+1)/2, (src.rows+1)/2)  计算的。而且不管你自己如何指定这个参数，一定必须保证满足以下关系式：|dstsize.width \* 2 - src.cols| ≤ 2; |dstsize.height \* 2 - src.rows| ≤ 2。
* 也就是说降采样的意思其实是把图像的尺寸缩减一半，行和列同时缩减一半。

borderType参数表示表示图像边界的处理方式。

## 2、代码实现

[复制代码](javascript:void(0);)

import cv2 as cv

import numpy as np

def demo(image):

level = 3

temp = image.copy()

cv.imshow('temp',temp)

demo\_image = []　　　　#一共有三个元素，元素类型是ndarray，shape分别为(288, 512, 3)，(144, 256, 3)，(72, 128, 3)

for i in range(level):

# 先对图像进行高斯平滑，然后再进行降采样（将图像尺寸行和列方向缩减一半）

dst = cv.pyrDown(temp)

demo\_image.append(dst)

cv.imshow('demo%d'%i,dst)

temp = dst.copy()

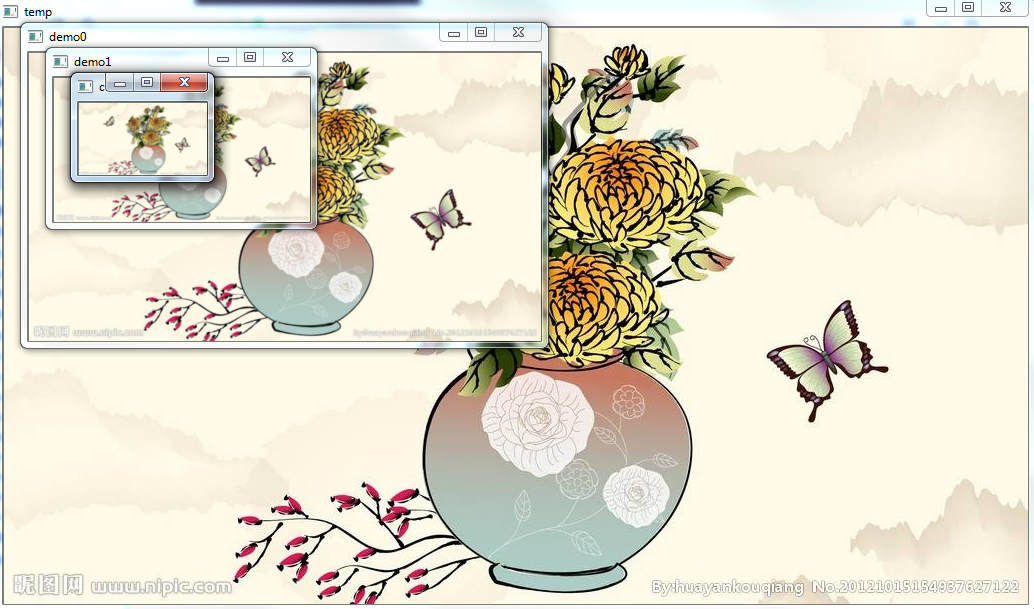
print(demo\_image.shape())

src = cv.imread('1.jpg')

demo(src)

cv.waitKey(0)

cv.destroyAllWindows()[复制代码](javascript:void(0);)



# 五、使用拉普拉斯金字塔

## 1、pyrUp上采样

def pyrUp(src, dst=None, dstsize=None, borderType=None): # ---> dst

opencv的pyrUp函数先对图像进行升采样（将图像尺寸行和列方向增大一倍），然后再进行高斯平滑。  
  
src参数表示输入图像。  
  
dst参数表示输出图像，它与src类型、大小相同。

dstsize参数表示降采样之后的目标图像的大小。

* 在默认的情况下，这个尺寸大小是按照 Size(src.cols\*2, (src.rows\*2) 来计算的。如果你自己要指定大小，那么一定要满足下面的条件：
* |dstsize.width - src.cols \* 2| ≤ (dstsize.width mod 2);  //如果width是偶数，那么必须dstsize.width是src.cols的2倍
* |dstsize.height - src.rows \* 2| ≤ (dstsize.height mod 2);

borderType参数表示表示图像边界的处理方式。

## 2、代码实现

[复制代码](javascript:void(0);)

import cv2 as cv

import numpy as np

def demo(image):

level = 3

temp = image.copy()

cv.imshow('temp',temp)

demo\_image = []

for i in range(level):

# 先对图像进行高斯平滑，然后再进行降采样（将图像尺寸行和列方向缩减一半）

dst = cv.pyrDown(temp)

demo\_image.append(dst)

cv.imshow('demo%d'%i,dst)

temp = dst.copy()

return demo\_image

def lap(image):

lap\_image = demo(image) #拉普拉斯需要用到高斯金字塔结果

level = len(lap\_image)

for i in range(level-1,-1,-1): #从后向前2,1,0

if (i-1) < 0 :

expand = cv.pyrUp(lap\_image[i],dstsize=image.shape[:2]) #先上采样

lapls = cv.subtract(image,expand) #使用高斯金字塔上一个减去当前上采样获取的结果，才是拉普拉斯金字塔

else:

expand = cv.pyrUp(lap\_image[i],dstsize=lap\_image[i-1].shape[:2])

lapls = cv.subtract(lap\_image[i-1],expand)

cv.imshow('lapls%s'%i,lapls)

src = cv.imread('1\_512\_512.jpg') #图片大小是512\*512

cv.imshow('input',src)

lap(src)

cv.waitKey(0)

cv.destroyAllWindows()[复制代码](javascript:void(0);)



### 尝试直接输出expand不去subtract，会发现变模糊了

cv.imshow("lapls%s"%i,expand)

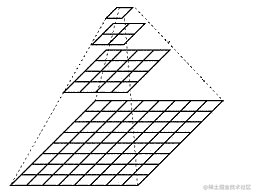


## ****我们选用的图片大小必须是2^n大小，或者是一个宽高相等的图片！！！****

# Opencv学习笔记：图像金字塔

# 一、高斯金字塔

## 1.数学原理讲解

标准意义上的高斯金字塔指的是不同分辨率的同一张图像所组成的图像结构，金字塔从上往下生成，图片的分辨率不断增大，称作上采样；金字塔从下往上生成，图片的分辨率不断减下，称作下采样（从下往上依次为G\_0、G\_1、G\_2....G\_N*G*0​、*G*1​、*G*2​....*GN*​层）。 

### i.下采样

下采样是对图像进行缩小的过程，期间图像分辨率不断降低，图片的信息在不断丢失，从下一层往上一层生成的过程分为两步：

1. 对底层图像进行高斯卷积，通常选用归一化5×5高斯卷积和进行卷积；
2. 删除所有的偶数行和偶数列，图片大小变为原来1/4； 示例代码如下：

**int** **main**()

{

Mat srcImg = imread("E:\\material\\assassin.jpeg");

**if** (srcImg.empty())

{

cout << "图片读取错误";

**return** -1;

}

imshow("原始图片", srcImg);

vector<Mat> tempImg;

tempImg.push\_back(srcImg);

**for** (**int** k = 0;k < 4;k++)

{

Mat tempimg= Mat::zeros(cvFloor(tempImg[k].size[0] /2), cvFloor(tempImg[k].size[1] / 2), srcImg.type());

tempImg.push\_back(tempimg);

Mat dstImg = Mat(tempImg[k].size(), tempImg[k].type());

GaussianBlur(tempImg[k], dstImg, Size(5, 5), 0, 0);

**for** (**int** i = 0;i < tempImg[k+1].size[0];i++)

{

**for** (**int** j = 0;j < tempImg[k+1].size[1];j++)

{

tempImg[k+1].at<Vec3b>(i, j) = dstImg.at<Vec3b>(2 \* i, 2 \* j);

}

}

**char** windowname[20];

sprintf\_s(windowname, "%s%d%s", "第", k+1, "次缩小.png");

imshow(windowname, tempImg[k+1]);

imwrite(windowname , tempImg[k + 1]);

}

**while** ((**char**)waitKey(0) != 'q');

**return** 0;

}

复制代码

结果：







在这里插入图片描述

### ii.上采样

下采样是对图像进行放大的过程，期间图像的分辨率不断升高，从上一层到下一层的生成过程分为两步：

1. 将图像在行和列上扩大两倍，隔一补一，新增的行和列用0来填充；
2. 使用与“下采样”相同的卷积核乘以4与放大后的图像进行卷积；

示例代码如下：

**int** **main**()

{

Mat srcImg = imread("第4次缩小.png");

**if** (srcImg.empty())

{

cout << "图片读取错误";

**return** -1;

}

imshow("原始图片", srcImg);

vector<Mat> tempImg;

tempImg.push\_back(srcImg);

**for** (**int** k = 0;k < 4;k++)

{

Mat tempimg= Mat::zeros(cvFloor(tempImg[k].size[0]\*2), cvFloor(tempImg[k].size[1]\*2), srcImg.type());

**for** (**int** i = 0;i < tempImg[k].size[0];i++)

{

**for** (**int** j = 0;j < tempImg[k].size[1];j++)

{

tempimg.at<Vec3b>(2\*i, 2\*j) = tempImg[k].at<Vec3b>(i, j);

}

}

Mat dstImg = Mat(tempimg.size(), tempimg.type());

GaussianBlur(tempimg, dstImg, Size(5, 5), 0, 0);

dstImg = 4 \* dstImg;

tempImg.push\_back(dstImg);

**char** windowname[20];

sprintf\_s(windowname, "%s%d%s", "第", k+1, "次放大.png");

imshow(windowname, tempImg[k+1]);

imwrite(windowname , tempImg[k + 1]);

}

**while** ((**char**)waitKey(0) != 'q');

**return** 0;

}

实验结果：

在这里插入图片描述

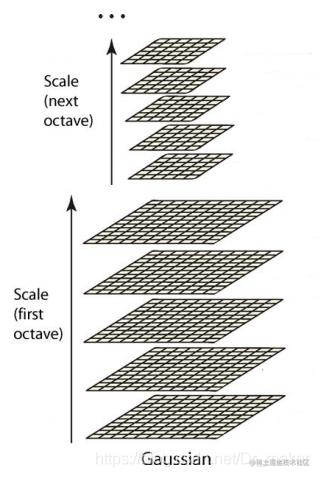




 需要注意的是下采样和上采样并不是互逆的两个过程，这一点可以从上采样之后的图片与下采样中相应图片的对比发现，因为下采样过程中消失了很多的图片信息。

## 2.Sift尺度中的高斯金字塔

高斯金字塔在sift（尺度不变特征变换)中的概念和前面的介绍有所不同，但其原理基本一致。在sift中，高斯金字塔分为组和层，每一组金字塔包含若干层。这一金字塔结构如下：

1. 将原图扩大1倍之后作为第一组的第一层，将第一组第一层经高斯卷积之后得到第二层，高斯卷积函数(其中\sigma*σ*取1.6)： G(x,y)={{1}\over{2\pi\sigma^2}}e^{-{{(x-x\_0)^2-(y-y\_0)^2}\over{2\sigma^2}}}*G*(*x*,*y*)=2*πσ*21​*e*−2*σ*2(*x*−*x*0​)2−(*y*−*y*0​)2​
2. 对第一组第一层进行高斯卷积，但高斯卷积函数中的\sigma*σ*要乘以一个系数k，所得结果作为第一组第二层；
3. 后续重复步骤二对上一层进行高斯卷积，高斯卷积中\sigma*σ*继续乘以系数k,最终得到第一组的N层图像，每一层尺寸都是原图的2倍，且都是对前一层进行高斯卷积得到，只是参数\sigma*σ*不同，依次为\sigma*σ*，k\sigma*kσ*、k^2\sigma*k*2*σ*....k^{N-2}\sigma*kN*−2*σ*.
4. 将第一组的倒数第三层 进行比例因子为2的降采样，得到尺寸为其1/4的图片作为第二组的第一层，重复步骤1～3得到第二组N层；
5. 重复步骤4得到M组N层高斯金字塔； 

## 3.差分金字塔

而得到了sift中的高斯金字塔有什么用呢？这就需要引入差分金字塔的概念了，差分金字塔是在高斯金字塔的基础上构建的，对应M组N层高斯金字塔有M组N-1层差分金字塔，其每组第一层都是由高斯金字塔中相应组第二层减第一层得到；每组第二层由高斯金字塔中相应组第三层减第二层得到，后续同理。对所得所用差分金字塔图像进行归一化，我们可以获取在**任何尺度变换及模糊程度下图片仍旧保持不变的特征**，这就是我们想要获取的稳定特征。

## 4.高斯金字塔在图像缩放中的应用

Opencv给出了相应的下采样和上采样函数供我们对图像进行缩放。

**下采样函数pyrDown()函数原型：**

pyrDown( InputArray src, OutputArray dst,**const** Size& dstsize = Size(),

**int** borderType = BORDER\_DEFAULT );

复制代码

第一、二个参数为输入输出图像，二者应该有相同的尺寸及类型；

第三个参数为希望输出图像变成的尺寸，有默认值Size (（src.cols+1)/2,(src.rows+1)/2),如果是自行设定的话，需要满足:

|dstsize.width-src.cols|\le2∣*dstsize*.*width*−*src*.*cols*∣≤2

|dstsize.height-src.rows|\le2∣*dstsize*.*height*−*src*.*rows*∣≤2

猜测这是针对与原图长或者宽为奇数时，可以在下采样第二步选择少去除一行（列）或者多去除一行（列），例如原图为长为51时，输出图像长度可选25或者26； 第四个参数为边界填充类型；

**上采样函数pyrUp函数原型：**

pyrUp( InputArray src, OutputArray dst,**const** Size& dstsize = Size(),

**int** borderType = BORDER\_DEFAULT );

复制代码

除第三个参数外与pyrDown()没有区别，pyrUp第三个参数也为输出图像尺寸，默认Size (src.cols\*2,src.rows\*2)，如果自行设定，需要满足：

|dstsize.width-src.cols\*2|\le(dstsize.width\ mod \ 2)∣*dstsize*.*width*−*src*.*cols*∗2∣≤(*dstsize*.*width* *mod* 2)

|dstsize.height-src.rows\*2|\le(dstsize.height\ mod \ 2)∣*dstsize*.*height*−*src*.*rows*∗2∣≤(*dstsize*.*height* *mod* 2)

猜测是为了适应不同的插值方式，像是原图长为10时，可以选择第10行后面不插也可以选择插，这样输出就会是20或者21；

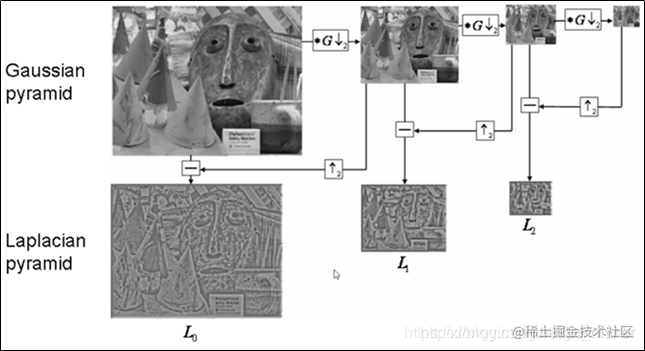
# 二、拉普拉斯金字塔

## 1.数学原理

拉普拉斯金字塔也是基于高斯金字塔得到的，对应N层高斯金字塔就有N-1层拉普拉斯金字塔，其数学公式为：

L\_i=G\_i-pyrUp(pyrDown(G\_i))*Li*​=*Gi*​−*pyrUp*(*pyrDown*(*Gi*​))

即拉普拉斯金字塔第一层是将高斯金字塔的第一层减去高斯金字塔第二层上采样之后得到。



**参考文献**

[Sift中尺度空间、高斯金字塔、差分金字塔（DOG金字塔）、图像金字塔](https://link.juejin.cn/?target=https%3A%2F%2Fblog.csdn.net%2Fdcrmg%2Farticle%2Fdetails%2F52561656)

[SIFT解析（一）建立高斯金字塔](https://link.juejin.cn/?target=https%3A%2F%2Fblog.csdn.net%2Fhonpey%2Farticle%2Fdetails%2F8639617)

[数字图像处理(21): 图像金字塔(高斯金字塔 与 拉普拉斯金字塔)](https://link.juejin.cn/?target=https%3A%2F%2Fblog.csdn.net%2Fzaishuiyifangxym%2Farticle%2Fdetails%2F90167880%3Fops_request_misc%3D%25257B%252522request%25255Fid%252522%25253A%252522159737154519725219903109%252522%25252C%252522scm%252522%25253A%25252220140713.130102334..%252522%25257D%26request_id%3D159737154519725219903109%26biz_id%3D0%26utm_medium%3Ddistribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v3~pc_rank_v3-2-90167880.pc_ecpm_v3_pc_rank_v3%26utm_term%3D%25E5%259B%25BE%25E5%2583%258F%25E9%2587%2591%25E5%25AD%2597%25E5%25A1%2594%26spm%3D1018.2118.3001.4187%232.1%2520%25E5%259F%25BA%25E7%25A1%2580%25E7%2590%2586%25E8%25AE%25BA)

# Python OpenCV 之图像金字塔，高斯金字塔与拉普拉斯金字塔

。

## 基础知识铺垫

学习图像金字塔，发现网上的资料比较多，检索起来比较轻松。

图像金字塔是一张图像多尺度的表达，或者可以理解成一张图像不同分辨率展示。

金字塔越底层的图片，像素越高，越向上，像素逐步降低，分辨率逐步降低。

### 高斯金字塔

我们依旧不对概念做过多解释，第一遍学习应用，应用，毕竟 365 天的周期，时间长，后面补充理论知识。

高斯金字塔用于向下采样，同时它也是最基本的图像塔。

在互联网检索原理，得到最简单的说明如下：

将图像的最底层（高斯金字塔的第 0 层），例如高斯核（5x5）对其进行卷积操作，这里的卷积主要处理掉的是偶数行与列，然后得到金字塔上一层图像（即高斯金字塔第 1 层），在针对该图像重复卷积操作，得到第 2 层，反复执行下去，即可得到高斯金字塔。

每次操作之后，都会将 M×N 图像变成 M/2 × N/2 图像，即减少一半。

还有实测中发现，需要用图像的宽和高一致的图片，并且宽高要是 2 的次幂数，例如，8 像素，16 像素，32 像素等等，一会你也可以实际测试一下。

图像金字塔应用到的函数有 cv2.pyrDown() 和 cv2.pyrUp() 。

### cv2.pyrDown 与 cv2.pyrUp 函数原型

通过 help 函数得到函数原型如下：

pyrDown(src[, dst[, dstsize[, borderType]]]) -> dst

pyrUp(src[, dst[, dstsize[, borderType]]]) -> dst

两个函数原型参数一致，参数说明如下：

* src ：输入图像；
* dst ： 输出图像；
* dstsize ： 输出图像尺寸，默认值按照 ((src.cols+1)/2, (src.rows+1)/2) 计算。

关于两个函数的补充说明：

* cv2.pyrDown 从一个相对高分辨率的大尺寸的图像上构建一个金字塔，运行之后的结果是，图像变小，分辨率降低（下采样）；
* cv2.pyrUp 是一个上采样的过程，尽管相对尺寸变大，但是分辨率不会增加，图像会变得更模糊。

测试代码如下：

import cv2 as cv

src = cv.imread("./testimg.jpeg")

print(src.shape[:2])

cv.imshow("src", src)

# 向下采样

dst = cv.pyrDown(src)

print(dst.shape[:2])

cv.imshow("dst", dst)

# 再次向下采样

dst1 = cv.pyrDown(dst)

print(dst1.shape[:2])

cv.imshow("dst1", dst1)

cv.waitKey()

运行代码之后，得到三张图片，大小依次减小，分辨率降低。



20210131213917372[1].png

通过上面运行得到的最小图，在执行向上采样之后，图片会变的模糊，这也说明上采样和下采样是非线性处理，它们是不可逆的有损处理，因此下采样后的图像是无法还原的，即使放大图片也会变模糊（后面学习到拉普拉斯金字塔可以解决该问题）。

# 向上采样

dst2 = cv.pyrUp(dst1)

print(dst2.shape[:2])

cv.imshow("dst2", dst2)



再总结一下上采样和下采样的步骤：

1. 上采样：使用 cv2.pyrUp 函数， 先将图像在每个方向放大为原来的两倍，新增的行和列用 0 填充，再使用先前同样的内核与放大后的图像卷积，获得新增像素的近似值;
2. 下采样：使用 cv2.pyrDown 函数，先对图像进行高斯内核卷积 ，再将所有偶数行和列去除。

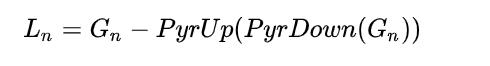
## 拉普拉斯金字塔（Laplacian Pyramid， LP）

拉普拉斯金字塔主要用于重建图像，由上文我们已经知道在使用高斯金字塔的的时候，上采样和下采样会导致图像细节丢失。

拉普拉斯就是为了在放大图像的时候，可以预测残差，何为残差，即小图像放大的时候，需要插入一些像素值，在上文直接插入的是 0，拉普拉斯金字塔算法可以根据周围像素进行预测，从而实现对图像最大程度的还原。

学习到原理如下：用高斯金字塔的每一层图像，减去其上一层图像上采样并高斯卷积之后的预测图像，得到一系列的差值图像即为 LP 分解图像（其中 LP 即为拉普拉斯金字塔图像）。

关于拉普拉斯还存在一个公式（这是本系列课程第一次书写公式），其中 L 为拉普拉斯金字塔图像，G 为高斯金字塔图像



使用下面的代码进行测试。

import cv2 as cv

src = cv.imread("./testimg.jpeg")

print(src.shape[:2])

cv.imshow("src", src)

# 向下采样一次

dst = cv.pyrDown(src)

print(dst.shape[:2])

cv.imshow("dst", dst)

# 向上采样一次

dst1 = cv.pyrUp(dst)

print(dst1.shape[:2])

cv.imshow("dst1", dst1)

# 计算拉普拉斯金字塔图像

# 原图 - 向上采样一次的图

laplace = cv.subtract(src, dst1)

cv.imshow("laplace", laplace)

cv.waitKey()

运行结果如下，相关的图像已经呈现出来，重点注意最右侧的图片。



这个地方需要注意下，如果你使用 cv.subtract(src, dst1) 函数，得到的是上图效果，但是在使用还原的时候会发现问题，建议直接使用 - 完成，匹配公式，修改代码如下：

# cv.subtract(src, dst1)

laplace = src - dst1

代码运行效果如下。



20210201082447927[1].png

学习过程中发现这样一段话：图像尺寸最好是 2 的整次幂，如 256,512 等，否则在金字塔向上的过程中图像的尺寸会不等，这会导致在拉普拉斯金字塔处理时由于不同尺寸矩阵相减而出错。

这个我在实测的时候发现确实如此，例如案例中使用的图像，在向下采样 2 次的时候，图像的尺寸就会发生变化，测试代码如下：

import cv2 as cv

src = cv.imread("./testimg.jpeg")

print(src.shape[:2])

cv.imshow("src", src)

# 向下采样1次

dst1 = cv.pyrDown(src)

print(dst1.shape[:2])

cv.imshow("dst", dst1)

# 向下采样2次

dst2 = cv.pyrDown(dst1)

print(dst1.shape[:2])

cv.imshow("dst2", dst2)

# 向上采样1次

up\_dst1 = cv.pyrUp(dst2)

print(up\_dst1.shape[:2])

cv.imshow("up\_dst1", up\_dst1)

# 计算拉普拉斯金字塔图像

# 采样1次 - 向上采样1次的图

laplace = dst1 - up\_dst1

cv.imshow("laplace", laplace)

cv.waitKey()

注意 print(up\_dst1.shape[:2]) 部分的输出如下：

(710, 400)

(355, 200)

(355, 200)

(356, 200)

如果在该基础上使用拉普拉斯图像金字塔，就会出现如下错误

Sizes of input arguments do not match

在总结一下拉普拉斯图像金字塔的执行过程：

* 向下采样：用高斯金字塔的第 i 层减去 i+1 层做上采样的图像，得到拉普拉斯第 i 层的图像；
* 向上采样：用高斯金字塔的 i+1 层向上采样加上拉普拉斯的第 i 层，得到第 i 层的原始图像。

向下采样上面的代码已经实现了，但是拉普拉斯向上采样还未实现，完善一下代码如下，为了代码清晰，我们将变量命名进行修改。

import cv2 as cv

src = cv.imread("./testimg\_rect.jpeg")

print(src.shape[:2])

cv.imshow("src", src)

# 高斯金字塔第 0 层

gus0 = src # 原图

# 高斯金字塔第 1 层

gus1 = cv.pyrDown(gus0)

# 高斯第 2 层

gus2 = cv.pyrDown(gus1)

# 拉普拉斯金字塔第 0 层

lap0 = gus0 - cv.pyrUp(gus1)

# 拉普拉斯金字塔第 1 层

lap1 = gus1 - cv.pyrUp(gus2)

# 显示拉普拉斯第一层代码

cv.imshow("laplace", lap1)

cv.waitKey()

下面用修改好的代码完成还原图片的操作。

import cv2 as cv

src = cv.imread("./testimg\_rect.jpeg")

print(src.shape[:2])

cv.imshow("src", src)

# 高斯金字塔第 0 层

gus0 = src # 原图

# 高斯金字塔第 1 层

gus1 = cv.pyrDown(gus0)

# 高斯第 2 层

gus2 = cv.pyrDown(gus1)

# 拉普拉斯金字塔第 0 层

lap0 = gus0 - cv.pyrUp(gus1)

# 拉普拉斯金字塔第 1 层

lap1 = gus1 - cv.pyrUp(gus2)

rep = lap0 + cv.pyrUp(lap1 + cv.pyrUp(gus2))

gus\_rep = cv.pyrUp(cv.pyrUp(gus2))

cv.imshow("rep", rep)

cv.imshow("gus\_rep", gus\_rep)

cv.waitKey()

以上代码最重要的部分为下面两句：

rep = lap0 + cv.pyrUp(lap1 + cv.pyrUp(gus2))

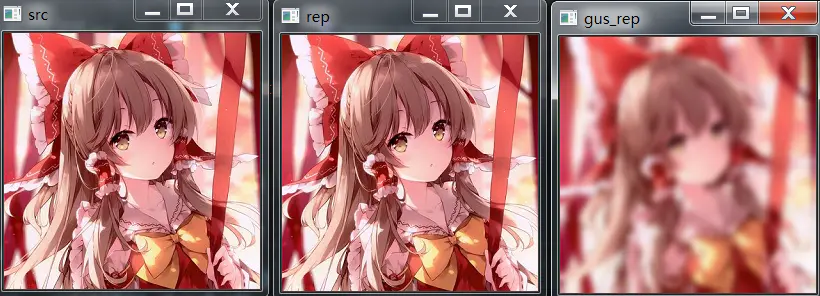
gus\_rep = cv.pyrUp(cv.pyrUp(gus2))

第一行代码中 lap1 + cv.pyrUp(gus2) 即文字公式 【用高斯金字塔的 i+1 层向上采样加上拉普拉斯的第 i 层，得到第 i 层的原始图像】的翻译。

第二行代码是使用直接向上采样，最终得到的是损失细节的图像。

上述代码运行的结果如下，通过拉普拉斯可以完美还原图像。

学习本案例之后，你可以在复盘本文开始部分的代码，将其进行修改。



20210201081348164[1].png

最后在学习一种技巧，可以直接将两幅图片呈现，代码如下：

import cv2 as cv

import numpy as np

src = cv.imread("./testimg\_rect.jpeg")

print(src.shape[:2])

cv.imshow("src", src)

# 向下采样1次

down\_dst1 = cv.pyrDown(src)

print(down\_dst1.shape[:2])

cv.imshow("dst", down\_dst1)

# 向上采样1次

up\_dst1 = cv.pyrUp(down\_dst1)

print(up\_dst1.shape[:2])

cv.imshow("up\_dst1", up\_dst1)

res = np.hstack((up\_dst1, src))

cv.imshow('res', res)

cv.waitKey()

运行之后，通过 np.hstack((up\_dst1, src)) 函数，将两个图像矩阵合并，实现效果如下：



20210201074426473[1].png