

计算机视觉-HW1

xTryer

2024/9/26

1 Short questions

1.1 Homogeneous coordination

1.1.1

$$\frac{x_i}{w} = \frac{x'_i}{1}, (w \neq 0)$$
$$x'_i = \frac{x_i}{w}$$

1.1.2

- 使用齐次坐标可以统一方便地以矩阵形式来表示平移、旋转、缩放等变换比如平移 (x,y,z) 作平移 (t_x, t_y, t_z) 可以简单地表示为

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 将矩阵形式的变换统一起来后，我们就能通过一系列的矩阵运算获得最后的变换矩阵
- 齐次坐标允许我们表示无穷远点，处理特定情况
- 齐次坐标的形式使插值结果的计算更为简洁，方便我们对图像进行处理

1.2 Dolly Zoom

Dolly Zoom 实际上就是在改变焦距的同时调整相机的位置，从而实现一种主视物不变但是背景剧烈变化的效果。主要有两种途径可以实现这种效果

- 将相机靠近物体的同时增大 FOV
- 将相机远离物体的同时减小 FOV

通过这些方法，可以保证特定物体在镜头中大小不变，但是在背景上的物体则在发生剧烈变化。

2 Camera parameters from the image

2.1 Find the vanishing line and calculate the camera height H



图 1: T2-1

根据测量, vanishing_line 的像素 y 坐标为 1763, 花坛顶部的像素 y 坐标为 1949, 花坛底部的像素 y 坐标为 2143

$$\frac{h}{H} = \frac{2143 - 1949}{2143 - 1763}$$
$$H = h \cdot \frac{380}{194} \approx 1.489m$$

2.2 Write down the intrinsic parameter matrix and derive the camera focal length f.

$$M_{intrinsic} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$c_x = \frac{4096pixels}{2} = 2048pixels$$

$$c_y = \frac{3072pixels}{2} = 1536pixels$$

$$w \cdot \begin{bmatrix} 1892 \\ 1949 \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -13.4 \\ 0 & 1 & 0 & -4.5 \\ 0 & 0 & 1 & -1.489 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 3.74 \\ 0.76 \\ 1 \end{bmatrix}$$

3 Image filtering and subsampling

3.1 Image Filtering

3.1.1 cross_correlation_2d

cross_correlation 实现：遍历图片上每一个像素点，对每一个像素点与其周围的像素求加权和，覆盖的像素与加权的权重由传入的 kernel 来决定。

```

1 def cross_correlation_2d(img_array,kernel_array):
2     img_height = img_array.shape[0]
3     img_width = img_array.shape[1]
4     kernel_height,kernel_width = kernel_array.shape
5     output_img_array = np.zeros_like(img_array,dtype=np.float32)
6     k_h = kernel_height//2
7     k_w = kernel_width//2
8
9     for i in range(0,img_height):
10        for j in range(0,img_width):
11            for k_i in range(-k_h,k_h+1):
12                for k_j in range(-k_w,k_w+1):
13                    cur_i = i+k_i
14                    cur_j = j+k_j
15                    if 0 <= cur_i < img_height and 0 <= cur_j < img_width:
16                        output_img_array[i][j] +=
17                            kernel_array[k_h+k_i][k_w+k_j]*img_array[cur_i][cur_j]
18
19    return output_img_array

```

代码解释：img_array 为传入图片的像素组（一般是 $m \times n \times 3$ 的三通道图片），格式为 np.array，kernel_array 则是传入的核矩阵，为 $n \times n$ 的矩阵（ n 一般为奇数），然后我们进行 cross_correlation 的计算

$$output_img(i, j) = \sum_{k,l} input_img(i + k, j + l) kernel(k, l)$$

对于范围之外的像素则做简单的跳过处理

3.1.2 convolve_2d

convolve_2d 实现：convolve 与 correlation 的区别在于当周围像素点与 kernel 进行加权计算时，计算公式变为了：

$$output_img(i, j) = \sum_{k,l} input_img(i - k, j - l) kernel(k, l)$$

因此我们只需要将传入的 kernel 进行一个 flip 转置得到一个新的 kernel，然后再进行 cross_correlation 操作即可

```
1 def convolve_2d(img_array,kernel_array):  
2     return cross_correlation_2d(img_array,np.flip(kernel_array))
```

代码解释：使用 numpy 库中的 flip 操作实现转置，其他与 cross_correlation 定义相同

3.2 Gaussian Blur

Gaussian Blur 实现：Gaussian Blur 实现的核心是生成一个 Gaussian Kernel，在二维情况下，高斯函数表达式为：

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

其中 x, y 在我们计算核函数时为对应点到 kernel 中心点的向量坐标，换言之，其实是 d_x, d_y

由于我们最后总要保持 Kernel 的 sum 为 1，所以我们在计算时可以忽略掉前面的系数：

$$G'(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

```
1 def gauss_2d_calculate(k_sigma,center_point,tar_point):  
2     distance_ = np.linalg.norm(center_point-tar_point)  
3     return np.exp(-(distance_*distance_)/(2*k_sigma*k_sigma))  
4  
5 def gaussian_blur_kernel_2d(k_size,k_sigma=1):  
6     gauss_kernel = np.zeros((k_size,k_size), dtype=np.float32)  
7     mean_ = k_size//2  
8     for i in range(0,k_size):  
9         for j in range(0,k_size):  
10             gauss_kernel[i,j] =  
11                 gauss_2d_calculate(k_sigma,np.array([mean_,mean_]),np.array([i,j]))  
12     return gauss_kernel/np.sum(gauss_kernel)
```

代码解释： k_size 表示生成的 Gaussian Kernel 为 $k_size \times k_size$ 大小，我们默认标准差为 1，均值为 0

3.3 Low Pass Filters

Low Pass Filters 实现：调用 Gaussian Kernel 生成一个对应大小的高斯核在生成完 Gaussian Kernel 之后，我们就可以调用 convolve 函数利用高斯核对图像进行卷积操作。

```
1 def low_pass(img_array,k_size,k_sigma=1):  
2     gauss_kernel = gaussian_blur_kernel_2d(k_size,k_sigma)  
3     return convolve_2d(img_array,gauss_kernel)
```

调用函数对图片进行低通滤波：

```
1 img_read = Image.open('Lena.png')
```

```
2     img_pixel_array = np.array(img_read)
3
4     # gaussian_blur_img_array = gaussian_blur_2d(img_pixel_array,3)
5     # gaussian_blur_img = Image.fromarray(np.uint8(gaussian_blur_img_array))
6     # gaussian_blur_img.save('lena_gaussian.png')
7
8     low_pass_img_array = low_pass(img_pixel_array,3)
9     low_pass_img = Image.fromarray(np.uint8(low_pass_img_array))
10    low_pass_img.save('lena_lowpass.png')
```

效果展示：



(a) bq_Original

(b) bq_Low_Pass

图 2: _Low_Pass_Show_bq



(a) Lena_Original

(b) Lena_Low_Pass

图 3: _Low_Pass_Show_Lena

3.4 Image subsampling/downsampling

Simple Down Sampling 实现：只取横纵坐标同时为奇数/偶数的像素点。

```

1 def image_subsampling(img_array):
2     img_height = img_array.shape[0]
3     img_width = img_array.shape[1]
4     img_channels_num = img_array.shape[2]
5
6     tar_height = img_height//2
7     tar_width = img_width//2
8     ret_img = np.zeros((tar_height,tar_width,img_channels_num))
9
10    for i in range(0,tar_height):
11        for j in range(0,tar_width):
12            ret_img[i,j]=img_array[i*2,j*2]
13
14    return ret_img

```

调用函数对图片进行 DownSample:

```

1 img_read = Image.open('Vangogh.png')
2 img_pixel_array = np.array(img_read)
3
4 subsampling_img_array=image_subsampling(img_pixel_array)
5 subsampling_img = Image.fromarray(np.uint8(subsampling_img_array))
6 subsampling_img.save('Vangogh_downsample.png')

```

效果展示:

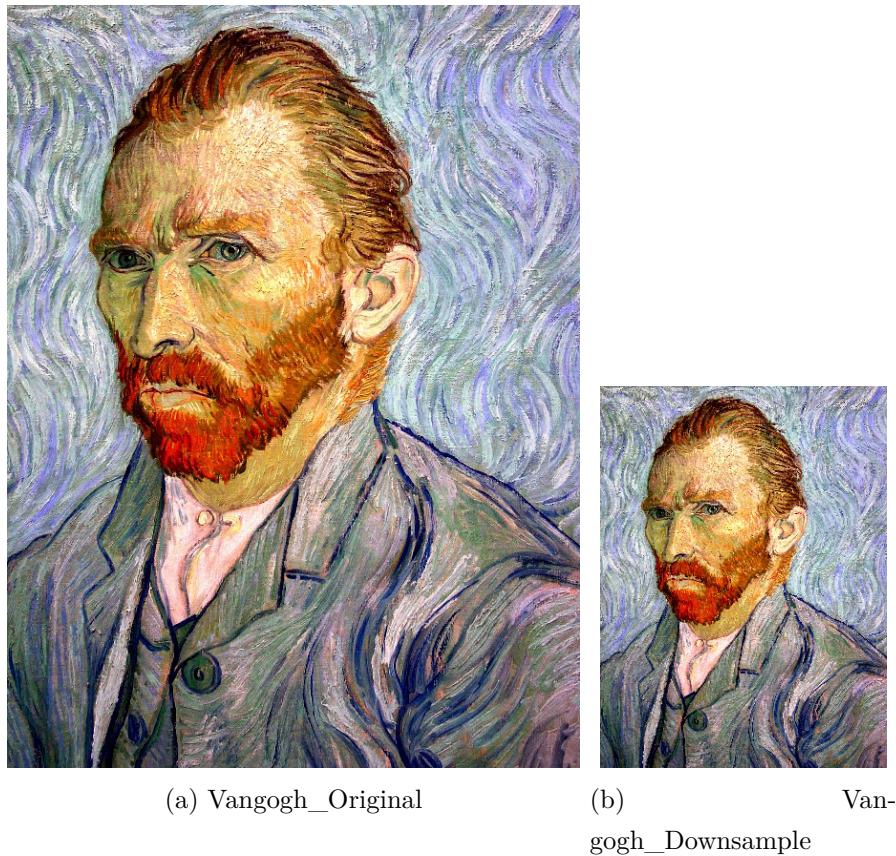


图 4: DownSample_Show_Vangogh



图 5: DownSample_Show_Lena

3.5 Gaussian pyramid

Gaussian Pyramid 实现: 若想得到 $\frac{1}{2^k}$ 级别的图像, 则只需对 $\frac{1}{2^{k-1}}$ 级别的图像进行一次 Gaussian Blur 然后再进行一次将图像缩减为一半的 DownSampling 即可。

```

1 def gaussian_pyramid(img_array,pyramid_level):
2     pyramid =[img_array]
3     cur_img = img_array
4     for i in range(0,pyramid_level):
5         cur_img = low_pass(cur_img,3)
6         cur_img = image_subsampling(cur_img)
7         pyramid.append(cur_img)
8
return pyramid

```

代码解释:pyramid_level 是一个正整数, 表示生成 $\frac{1}{2^0}, \frac{1}{2^1}, \frac{1}{2^2}, \dots, \frac{1}{2^{pyramid_level}}$ 的图像

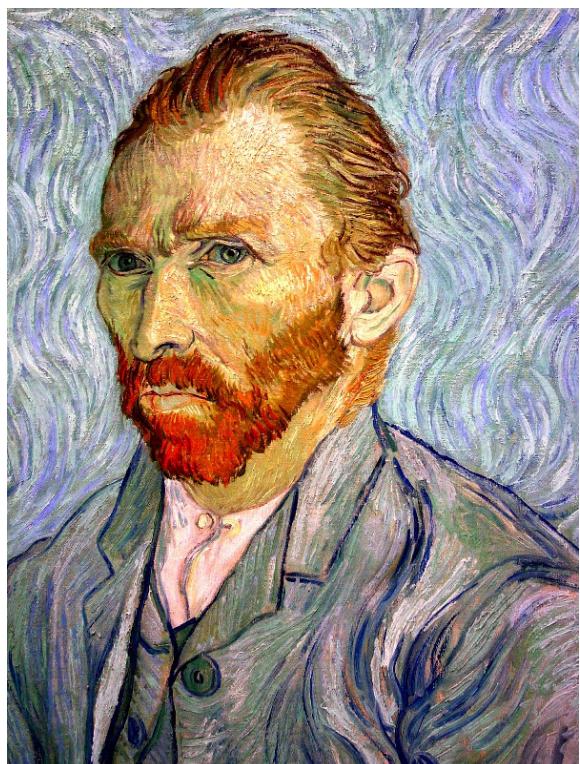
调用函数对图片求 Gaussian Pyramid:

```

1 img_read = Image.open('Vangogh.png')
2 img_pixel_array = np.array(img_read)
3
4 pyramid_level = 3
5 gaussian_pyramid_img_array = gaussian_pyramid(img_pixel_array,pyramid_level)
6 for i in range(0,pyramid_level+1):
7     pyramid_img_ =Image.fromarray(np.uint8(gaussian_pyramid_img_array[i]))
8     pyramid_img_.save(f'Vangogh_pyramid_{pow(2,i)}.png')

```

效果展示:

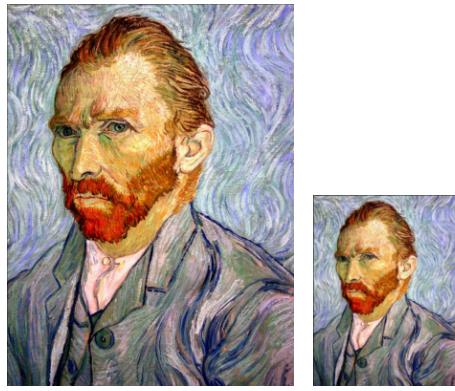


(a) Vangogh_pyramid_1



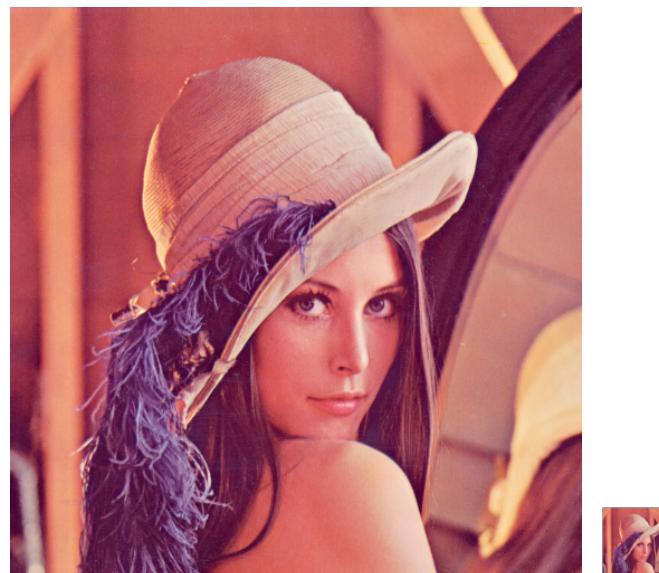
(b)
Van-
gogh_pyramid_8

图 6: Gaussian_Pyramid_Vangogh



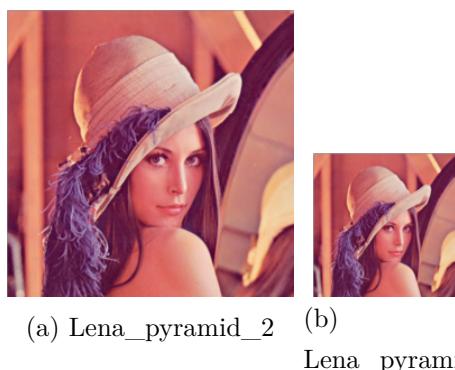
(a) Vangogh_pyramid_2 (b) Van-
gogh_pyramid_4

图 7: Gaussian_Pyramid_Vangogh



(a) Lena_pyramid_1 (b)
Lena_pyramid_8

图 8: Gaussian_Pyramid_Lena



(a) Lena_pyramid_2 (b)
Lena_pyramid_4

图 9: Gaussian_Pyramid_Lena



(a) bq_pyramid_1

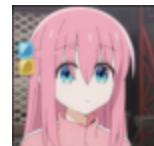


(b)
bq_pyramid_8

图 10: Gaussian_Pyramid_bq



(a) bq_pyramid_2



(b)
bq_pyramid_4

图 11: Gaussian_Pyramid_bq