CV HomeWork 2 实验报告

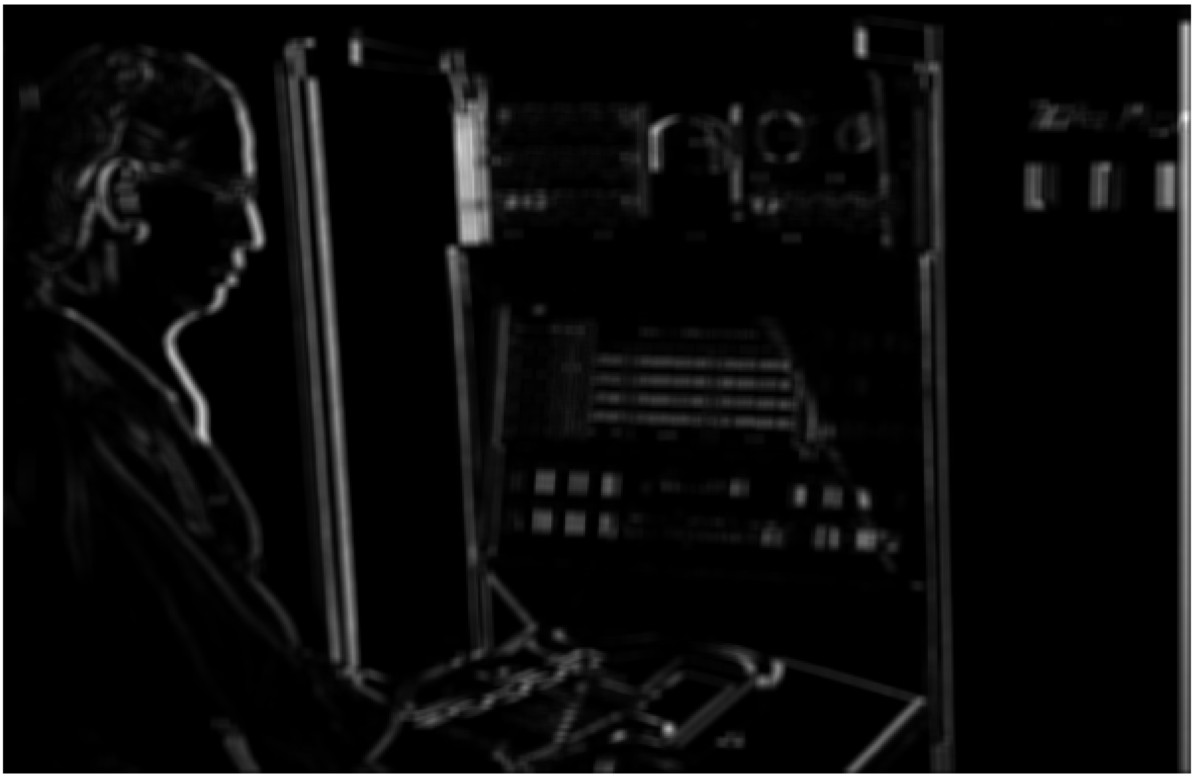
Task 1：

corner_score的代码如下：

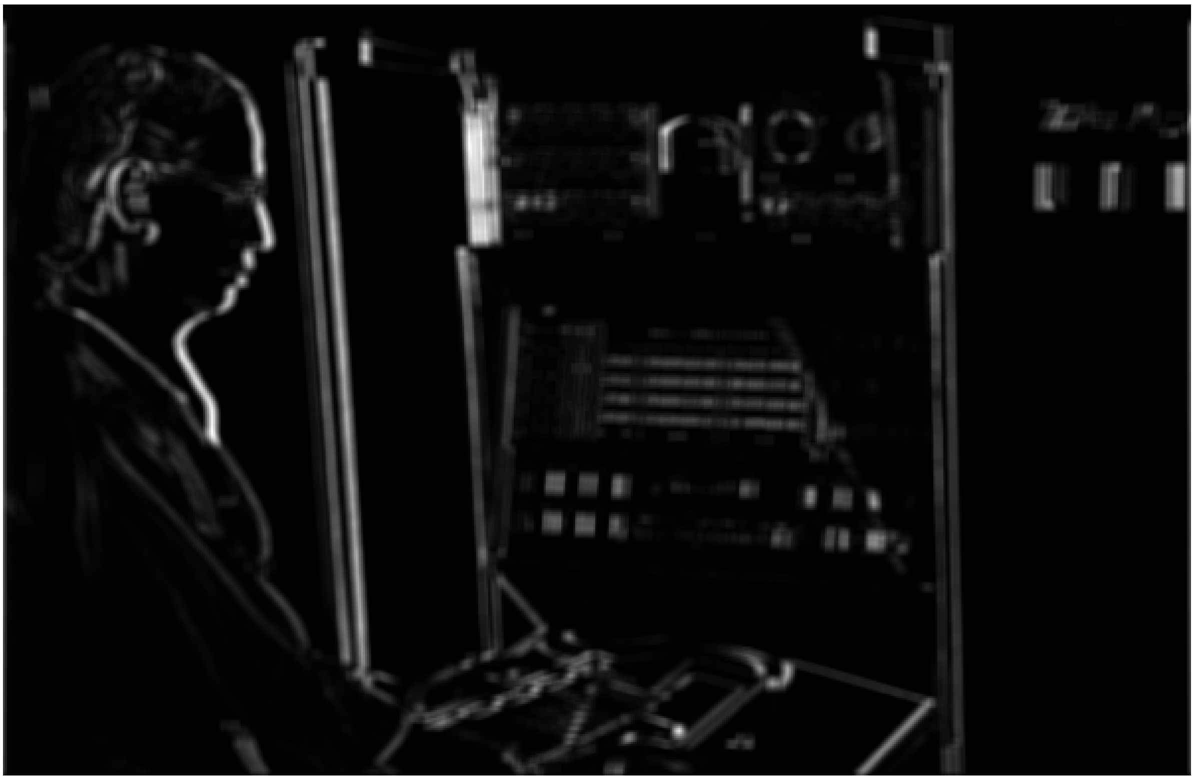
```
def corner_score(image, u=5, v=5, window_size=(5, 5)):
    """
    Given an input image, x_offset, y_offset, and window_size,
    return the function E(u,v) for window size W
    corner detector score for that pixel.
    Use zero-padding to handle window values outside of the image.

    Input- image: H x W
           u: a scalar for x offset
           v: a scalar for y offset
           window_size: a tuple for window size

    Output- results: a image of size H x W
    """
    # np.set_printoptions(threshold=np.inf)
    output = np.zeros(shape=(image.shape[0],image.shape[1]))
    I = np.zeros(shape=(image.shape[0] + window_size[0],image.shape[1] +
window_size[1]))
    I[(int)(window_size[0]/2): (int)(window_size[0]/2 + image.shape[0]), (int)
(window_size[1]/2):(int)(window_size[1]/2 + image.shape[1])] = image
    I_u_v = np.zeros(shape=(I.shape[0],I.shape[1]))
    for i in range(I.shape[0]):
        for j in range(I.shape[1]):
            if i + u >= I.shape[0] or j + v >= I.shape[1]:
                I_u_v[i,j] = 0
            else:
                I_u_v[i,j] = I[i + u, j + v]
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            output[i,j] = np.sum(np.square(I_u_v[i:(int)(i + window_size[0]),j:
(int)(j + window_size[1])] - I[i:(int)(i + window_size[0]),j:(int)(j +
window_size[1])]))
    return output
```

运行结果图如下：

u = 0, v = 5, window size (5, 5)

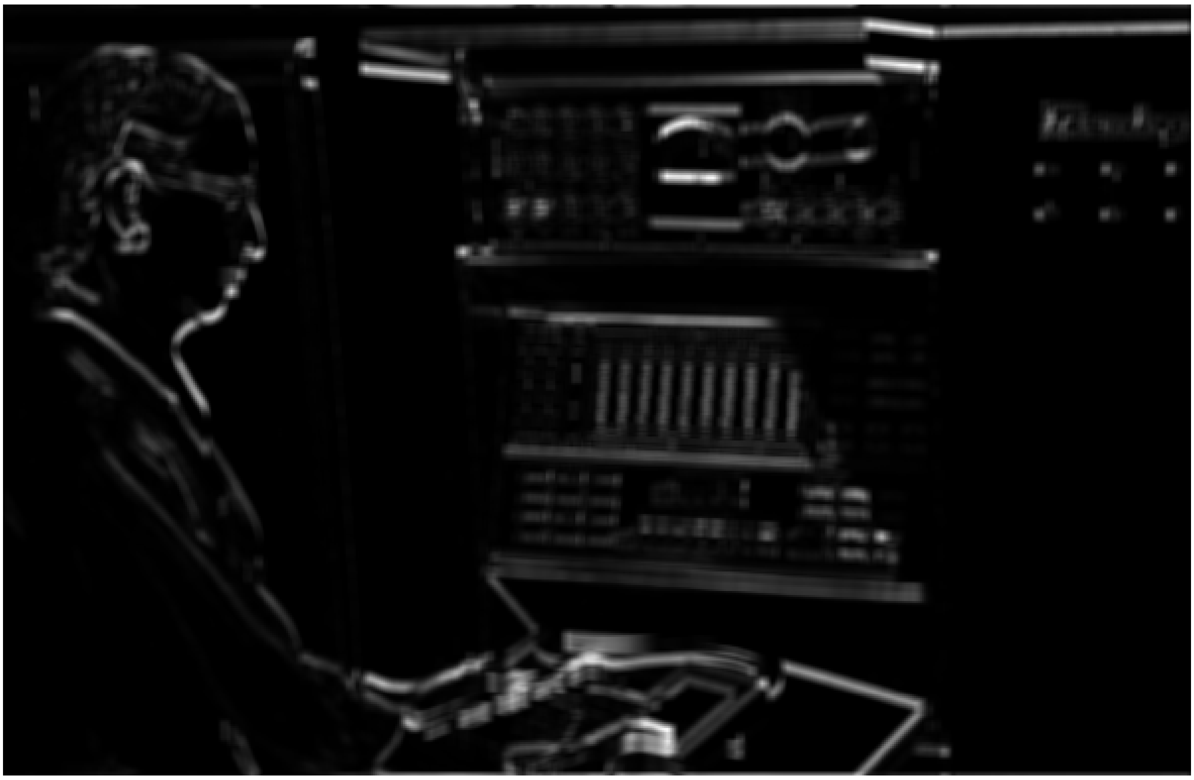


u = 0, v = -5, window size (5, 5)



u = 5, v = 0, window size (5, 5)

u = -5, v = 0, window size (5, 5)

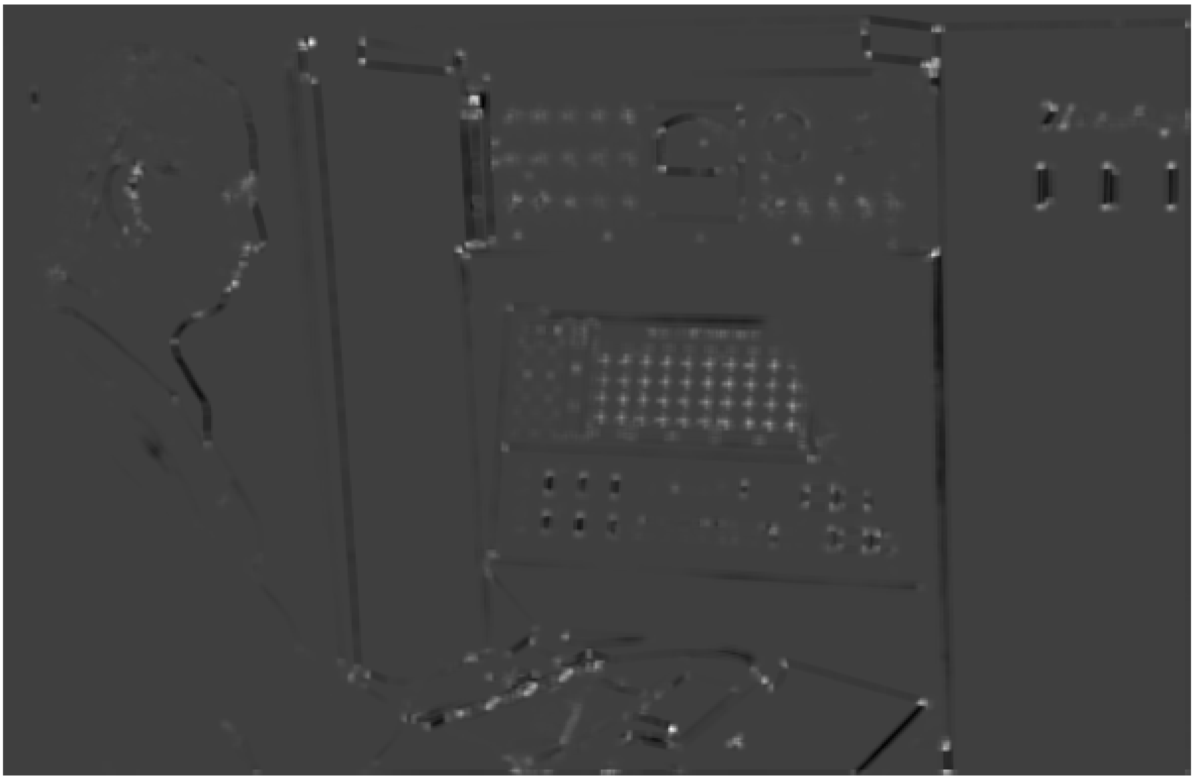


Task 2:

harris_detector函数代码如下：

```python
def harris_detector(image, window_size=(5, 5)):
    """
    Given an input image, calculate the Harris Detector score for all pixels
    You can use same-padding for intensity (or 0-padding for derivatives)
    to handle window values outside of the image.

    Input- image: H x W
    Output- results: a image of size H x W
    """
    # compute the derivatives
    alpha = 0.06
    kx = np.array([[-1,0,1]]) * 0.5
    ky = np.transpose(kx)
    Ix = scipy.ndimage.convolve(image, kx, mode = 'constant')
    Iy = scipy.ndimage.convolve(image, ky, mode = 'constant')

    Ixx = Ix * Ix
    Iyy = Iy * Iy
    Ixy = Ix * Iy

    # For each image location, construct the structure tensor and calculate
    # the Harris response
    k_gauss = np.ones(window_size)
    M = np.zeros((image.shape[0], image.shape[1], 3))
    M[:,:,0] = scipy.ndimage.convolve(Ixx,k_gauss, mode = 'constant')
    M[:,:,1] = scipy.ndimage.convolve(Ixy,k_gauss, mode = 'constant')
    M[:,:,2] = scipy.ndimage.convolve(Iyy,k_gauss, mode = 'constant')

    R =  M[:,:,0]*M[:,:,2] - M[:,:,1]**2 - alpha*((M[:,:,0]+M[:,:,2]))**2
    response = R

    return response
```

运行结果图如下：

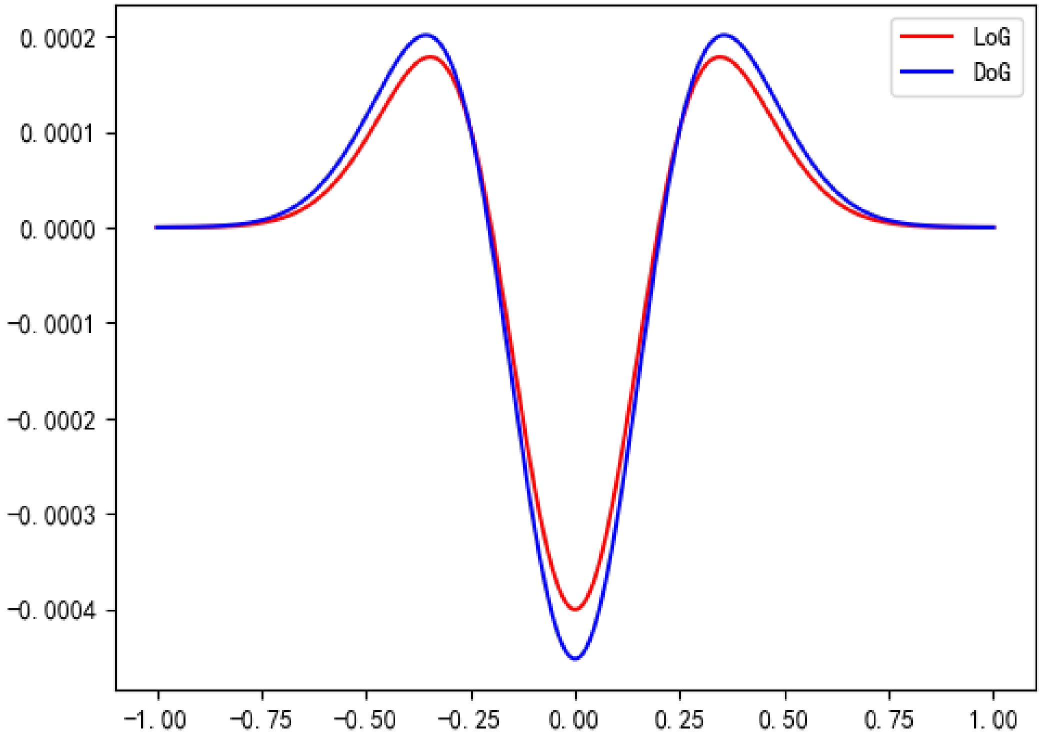


Task 3:

比较LoG与DoG的代码如下：

```
data = np.load('log1d.npz')
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
x = np.linspace(-1,1,data['log50'].shape[0])
y1 = data['log50']
y2 = data['gauss53'] - data['gauss50']
plt.plot(x,y1,color='red',label='LoG')
plt.plot(x,y2,color='blue',label='DoG')

plt.legend()
plt.savefig('./LoG_vs_DoG.png')
```

运行结果如下：



两者有差异的原因主要在于两者计算方式的不同，分别如下：

LoG 的表达式如下：

$$L = \sigma^2(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma))$$

DoG的表达式如下：

$$DoG = G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2\nabla^2 G$$

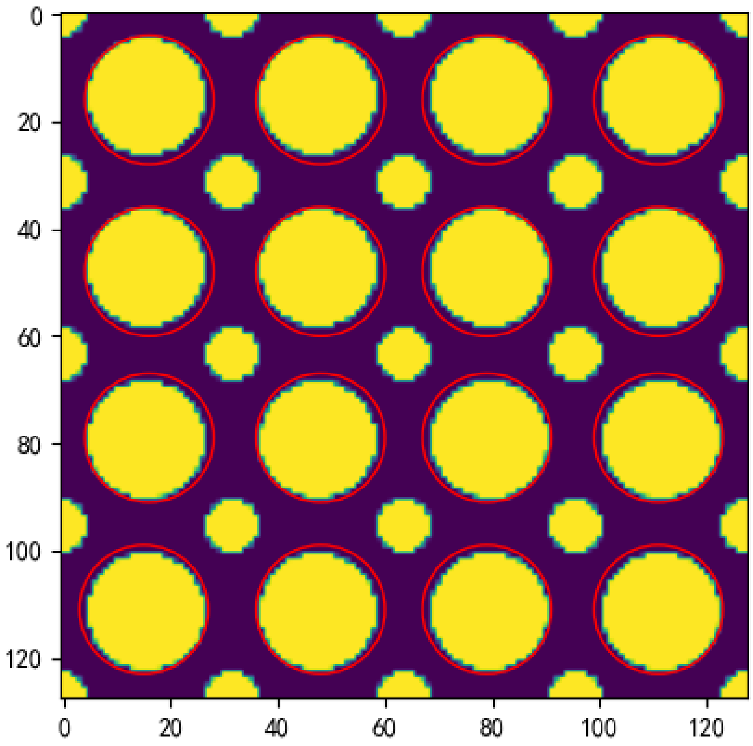
Task 4:

gaussian_filter函数代码如下：

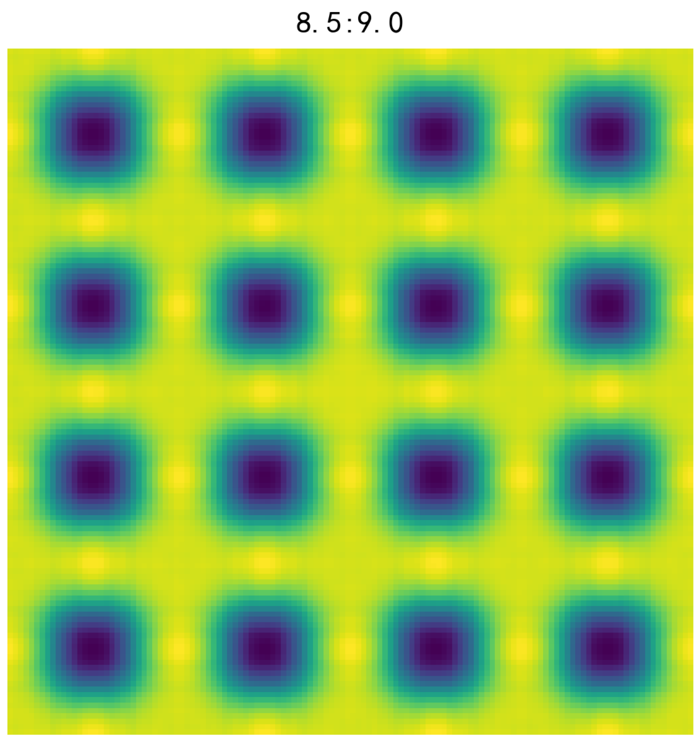
```
def gaussian_filter(image, sigma):
    """
    Given an image, apply a Gaussian filter with the input kernel size
    and standard deviation
    Input
      image: image of size HxW
      sigma: scalar standard deviation of Gaussian Kernel

    Output
      Gaussian filtered image of size HxW
    """
    H, W = image.shape
    # -- good heuristic way of setting kernel size
    kernel_size = int(2 * np.ceil(2 * sigma) + 1)
    # Ensure that the kernel size isn't too big and is odd
    kernel_size = min(kernel_size, min(H, W) // 2)
    if kernel_size % 2 == 0:
        kernel_size = kernel_size + 1
    # TODO implement gaussian filtering of size kernel_size x kernel_size
    # Similar to Corner detection, use scipy's convolution function.
    # Again, be consistent with the settings (mode = 'reflect').

    kernel = np.zeros(shape=(kernel_size,kernel_size),dtype=np.float)
    radius = kernel_size // 2
    for i in range(-radius, radius + 1):
        for j in range(-radius, radius + 1):
            v = 1.0 / (2 * np.pi * sigma ** 2) * np.exp(-1.0 / (2 * sigma ** 2) * (i ** 2 + j ** 2))
            kernel[i + radius, j + radius] = v
    kernel = kernel / np.sum(kernel)
    output = scipy.ndimage.convolve(image, kernel, mode = 'reflect')
    return output
```

运行结果如下图：

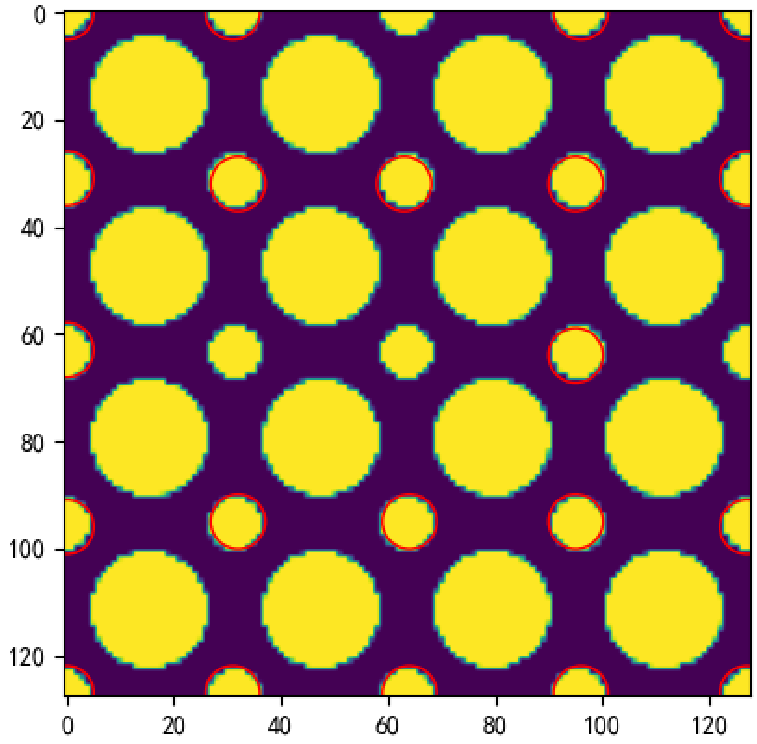
polka_large.png



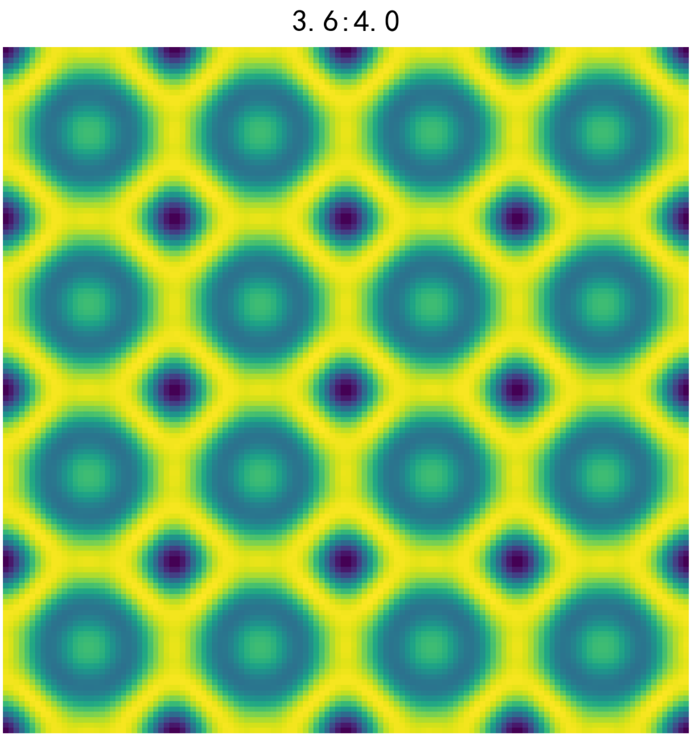
polka_large_DoG.png



polka_small.png



polka_small_DoG.png



在large的图片里面，共识别出**16个大圆**；在small图片中共识别出**21个小圆**

Task 5:

代码如下:

```
cell_names = ['001cell','002cell','004cell','005cell']
for cell_name in cell_names:
    cell_image = read_img('./cells/' + cell_name + '.png')
    sigma_1, sigma_2 = 3.6, 4
    gauss_1 = gaussian_filter(cell_image,sigma_1)
    gauss_2 = gaussian_filter(cell_image,sigma_2)
    DoG_Cell = gauss_2 - gauss_1
    maxima = find_maxima_for_cell(DoG_Cell, k_xy=10)
    visualize_scale_space(DoG_Cell, sigma_1, sigma_2 / sigma_1,
                          './cell_detections/' + cell_name + '_DoG.png')
    visualize_maxima(cell_image, maxima, sigma_1, sigma_2 / sigma_1,
                     './cell_detections/' + cell_name + '.png')
```

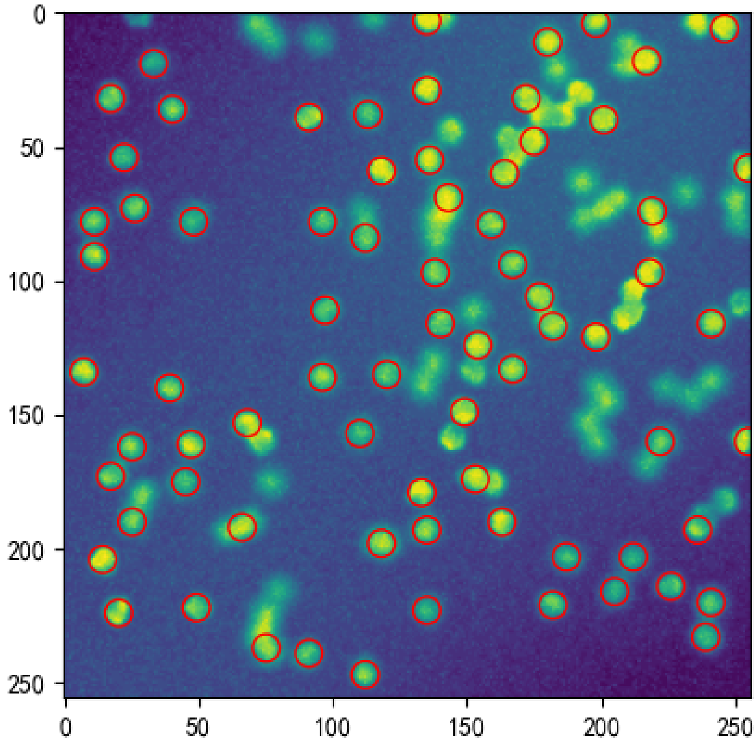
在这个部分我没有使用自带的`find_maxima`函数，而是自己在这个的基础上改进了下。因为识别出的点正确的和错误的相差很大，取选出点的平均值可以将他们很好地分开，因此将选出的圆圈计算一个平均值，只保留低于平均值的圆圈，即为`find_maxima_for_cell`函数，具体如下

```
def find_maxima_for_cell(scale_space, k_xy=5, k_s=1):
    """
    Extract the peak x,y locations from scale space for cell detections

    author
        leihao

    Input
```

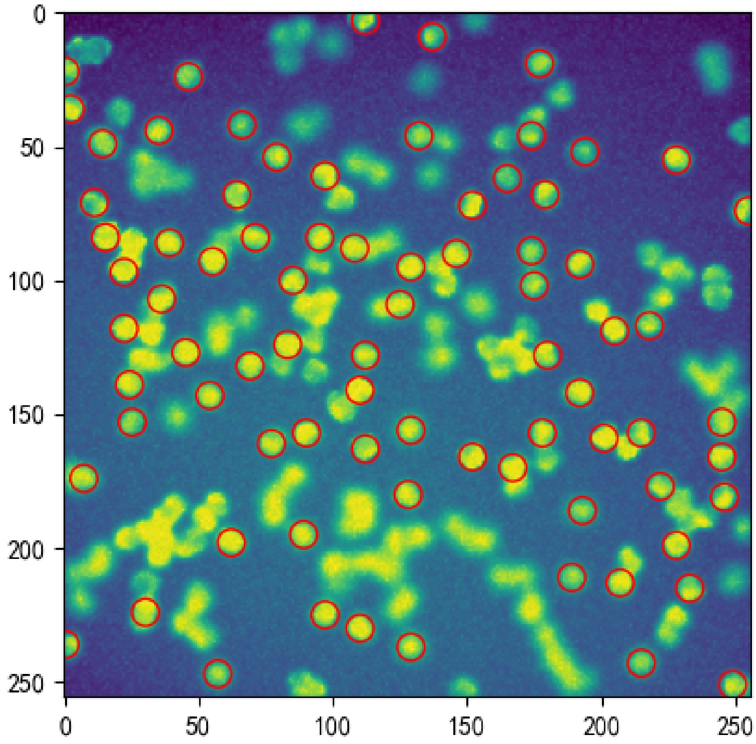
```
      scale_space: Scale space of size HxWxS
      k: neighborhood in x and y
      ks: neighborhood in scale

    Output
      list of (x,y) tuples; x<W and y<H
    """
    if len(scale_space.shape) == 2:
        scale_space = scale_space[:, :, None]

    H, W, S = scale_space.shape
    maxima = []
    maxima_data = []
    for i in range(H):
        for j in range(W):
            for s in range(S):
                # extracts a local neighborhood of max size
                # (2k_xy+1, 2k_xy+1, 2k_s+1)
                neighbors = scale_space[max(0, i - k_xy):min(i + k_xy + 1, H),
                                        max(0, j - k_xy):min(j + k_xy + 1, W),
                                        max(0, s - k_s):min(s + k_s + 1, S)]
                mid_pixel = scale_space[i, j, s]
                num_neighbors = np.prod(neighbors.shape) - 1
                # if mid_pixel > all the neighbors; append maxima
                if np.sum(mid_pixel < neighbors) == num_neighbors:
                    maxima.append((i, j, s))
                    maxima_data.append(mid_pixel)
    maxima_final = []
    mean_maxima_data = sum(maxima_data) / len(maxima_data)
    for i in range(len(maxima_data)):
        if maxima_data[i] < mean_maxima_data:
            maxima_final.append(maxima[i])
    return maxima_final
```

识别效果大大提升，运行结果如下：

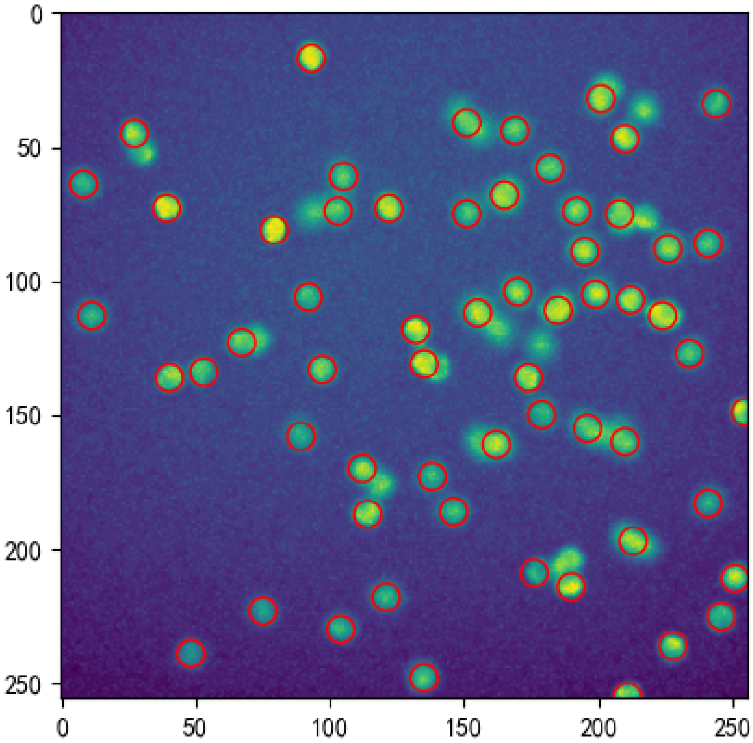
001cell.png



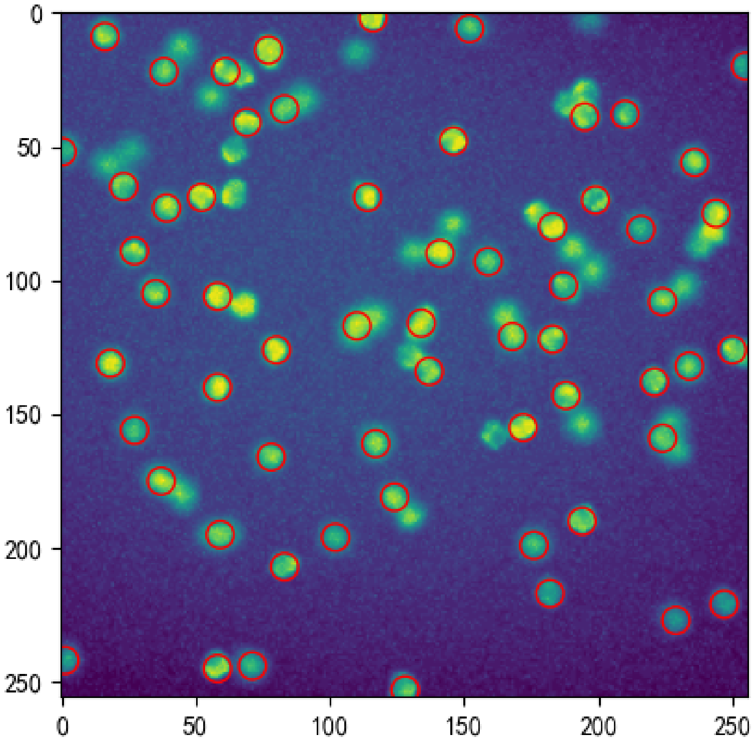
002cell.png



004cell.png


0
50
100
150
200
250
0
50
100
150
200
250

005cell.png


0
50
100
150
200
250
0
50
100
150
200
250