CSC420 Assignment #1

Tingfeng Xia (1003780884)

Tuesday 6th October, 2020

Contents

1 Part I: Theory

1.1 Q1) LTI Systems and Convolution

Consider a Linear Time-Invariant system T, which for impulse signal

$$\delta(n) = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{else} \end{cases} \tag{1.1}$$

is such that $T[\delta(n)] = h(n)$. We wish to show that for any input signal $x(n)$, $T[x(n)] = h(n) * x(n)$, i.e. the output is the impulse response convolved with itself. We start with the following

$$(\ddagger) = \sum_t x(t)\delta(n-t) \tag{1.2}$$

but since $\delta(n-t) = 0$ precisely when $n = t$, we can rewrite

$$\begin{aligned}(\ddagger) &= \sum_t x(n)\delta(n-t) & (1.3)\\ &= x(n)\sum_t \delta(n-t) & (1.4)\end{aligned}$$

but $\delta(n-t)$ only evaluates to 1 once for all possible t, which is when $n = t$. So $\sum_t \delta(n-t) = 1$, and thus

$$(\ddagger) = \sum_t x(t)\delta(n-t) = x(n) \tag{1.5}$$

Then,

$$T[x(n)] = T\left[\sum_t x(t)\delta(n-t)\right] \tag{1.6}$$

where we notice that

1. summation is a linear operation so we can bring T into the integral, and
2. the multiplicand $x(t)$ inside the summation is just a constant real number (evaluated value of $x(\cdot)$ at t), and
3. the transformation is time invariant meaning that $T[\delta(n)] = h(n)$ implies $T[\delta(n-t)] = h(n-t)$

$$\begin{aligned} T[x(n)] &= \sum_t x(t)T\left[\delta(n-t)\right] && (1.7)\\ &= \sum_t x(t)h(n-t) && (1.8)\\ &= x(n) * h(n) = h(n) * x(n) && (1.9)\end{aligned}$$

where the last step utilizes the commutativity of convolution operation. This concludes the proof. ■

1.2 Q2) Polynomial Multiplication and Convolution

Suppose that we have two polynomials

$$f(x) = \sum_{i=0}^{m} a_i x_i \quad \text{and} \quad g(x) = \sum_{j=0}^{n} b_j x^j \tag{1.10}$$

Then, in general we have

$$f(x)g(x) = \sum_{i=0}^{m}\sum_{j=0}^{n} a_i b_i x^i x^j \tag{1.11}$$

we also assume, without loss of generality, that $m \geq n$. We use the change of variable $k = i+j$, then

$$\begin{aligned} f(x)g(x) &= \sum_{k=0}^{m+n} \sum_{j=\max\{0,k-m\}} a_{k-j} b_j x^{k-j} x^j && (1.12)\\ &= \sum_{k=0}^{m+n} \underbrace{\left[\sum_{j=\max\{0,k-m\}} a_{k-j} b_j\right]}_{\dagger} x^k && (1.13)\end{aligned}$$

where we notice that † is exactly the same as the convolution between vector forms of $f(x)$ and $g(x)$, and this concludes the proof. ■

1.3 Q3) Laplacian Operator

Let $(x, y) \in \mathbb{R}^2$, and we compute the rotated version of (x, y) using the rotation matrix, i.e.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta + y\sin\theta \\ y\cos\theta - x\sin\theta \end{bmatrix} \tag{1.14}$$

Our final goal is to show that

Notation shorthand used here: $\partial_x f = \frac{\partial f}{\partial x}$ and $\partial_{xy} f = \frac{\partial}{\partial x}\frac{\partial f}{\partial y}$

$$\partial_{xx} I + \partial_{yy} I = \partial_{uu} I + \partial_{vv} I \tag{1.15}$$

We first compute $\partial_x I$,

$$\partial_x I = \frac{\partial I}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial I}{\partial v}\frac{\partial v}{\partial x} \tag{1.16}$$

$$= \frac{\partial I}{\partial u}\cos\theta - \frac{\partial I}{\partial v}\sin\theta \tag{1.17}$$

then,

$$\partial_{xx} I = \partial_x(\partial_x I) = \underbrace{\frac{\partial}{\partial x}\frac{\partial I}{\partial u}}_{(1)}\cos\theta - \underbrace{\frac{\partial}{\partial x}\frac{\partial I}{\partial v}}_{(2)}\sin\theta \tag{1.18}$$

and due to space limitations, we work on (1) and (2) separately below. Recall that Clairaut's Theorem states that for any second partial derivative $\partial_{xy} f = \partial_{yx} f$, then,

$$(1) = \partial_{xu} I = \partial_{ux} I = \frac{\partial}{\partial u}\frac{\partial I}{\partial x} \tag{1.19}$$

$$= \frac{\partial}{\partial u}\left(\frac{\partial I}{\partial u}\cos\theta - \frac{\partial I}{\partial v}\sin\theta\right) \tag{1.20}$$

$$= \partial_{uu} I\cos\theta - \partial_{uv} I\sin\theta \tag{1.21}$$

and

$$(2) = \partial_{xv} I = \partial_{vx} I = \frac{\partial}{\partial v}\frac{\partial I}{\partial x} \tag{1.22}$$

$$= \frac{\partial}{\partial v}\left(\frac{\partial I}{\partial u}\cos\theta - \frac{\partial I}{\partial v}\sin\theta\right) \tag{1.23}$$

$$= \partial_{vu} I\cos\theta - \partial_{vv} I\sin\theta \tag{1.24}$$

Thus

$$\partial_{xx} I = (\partial_{uu} I\cos\theta - \partial_{vu} I\sin\theta)\cos\theta - (\partial_{vu} I\cos\theta - \partial_{vv} I\sin\theta)\sin\theta \tag{1.25}$$

We do the similar to try to find $\partial_{yy} I$ by starting from $\partial_y I$,

$$\partial_y I = \frac{\partial I}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial I}{\partial v}\frac{\partial v}{\partial y} = \partial_u I\sin\theta + \partial_v I\cos\theta \tag{1.26}$$

then,

$$\partial_{yy} I = \partial_y(\partial_y I) = \underbrace{\frac{\partial}{\partial y}\frac{\partial I}{\partial u}}_{(3)} \sin\theta + \underbrace{\frac{\partial}{\partial y}\frac{\partial I}{\partial v}}_{(4)} \cos\theta \tag{1.27}$$

We expand (3) and (4),

$$(3) = \partial_{yu} I = \partial_{uy} I \tag{1.28}$$
$$= \frac{\partial}{\partial u}\left(\frac{\partial I}{\partial u}\sin\theta + \frac{\partial I}{\partial v}\cos\theta\right) \tag{1.29}$$
$$= \partial_{uu} I \sin\theta + \partial_{uv} I \cos\theta \tag{1.30}$$

and

$$(4) = \partial_{yv} I = \partial_{vy} I \tag{1.31}$$
$$= \frac{\partial}{\partial u}\left(\frac{\partial I}{\partial u}\sin\theta + \frac{\partial I}{\partial v}\cos\theta\right) \tag{1.32}$$
$$= \partial_{vu} I \sin\theta + \partial_{vv} \cos\theta \tag{1.33}$$

thus,

$$\partial_{yy} I = (\partial_{uu} I \sin\theta + \partial_{uv} I \cos\theta)\sin\theta + (\partial_{uv} I \sin\theta + \partial_{vv} I \cos\theta)\cos\theta \tag{1.34}$$

All together, we have

$$\partial_{xx} I + \partial_{yy} I = \partial_{uu} I \cos^2\theta \tag{1.35}$$
$$- \partial_{vu} I \sin\theta\cos\theta \tag{1.36}$$
$$- \partial_{vu} I \cos\theta\sin\theta \tag{1.37}$$
$$+ \partial_{vv} I \sin^2\theta \tag{1.38}$$
$$+ \partial_{uu} I \sin^2\theta \tag{1.39}$$
$$+ \partial_{uv} I \sin\theta\cos\theta \tag{1.40}$$
$$+ \partial_{uv} I \sin\theta\cos\theta \tag{1.41}$$
$$+ \partial_{vv} I \cos^2\theta \tag{1.42}$$
$$= \partial_{uu} I \cos^2\theta + \partial_{vv} I \sin^2\theta + \partial_{uu} I \sin^2\theta + \partial_{vv} I \cos^2\theta \tag{1.43}$$
$$= \partial_{uu} I(\sin^2\theta + \cos^2\theta) + \partial_{vv} I(\sin^2\theta + \cos^2\theta) \tag{1.44}$$
$$= \partial_{uu} I + \partial_{vv} I \tag{1.45}$$

which is what we wanted to show in Equation 1.15, and this concludes the proof. ■

2 Part II: Applications

2.1 Q4) Edge Detection

2.1.1 Step I - Gaussian Blurring

Please check my implementation of `get_gaussian_kernel(size=3, sigma=1.)` inside file `a1_code.ipynb` submitted along with this report. Figure 1 is the visualization with appropriate parameters.

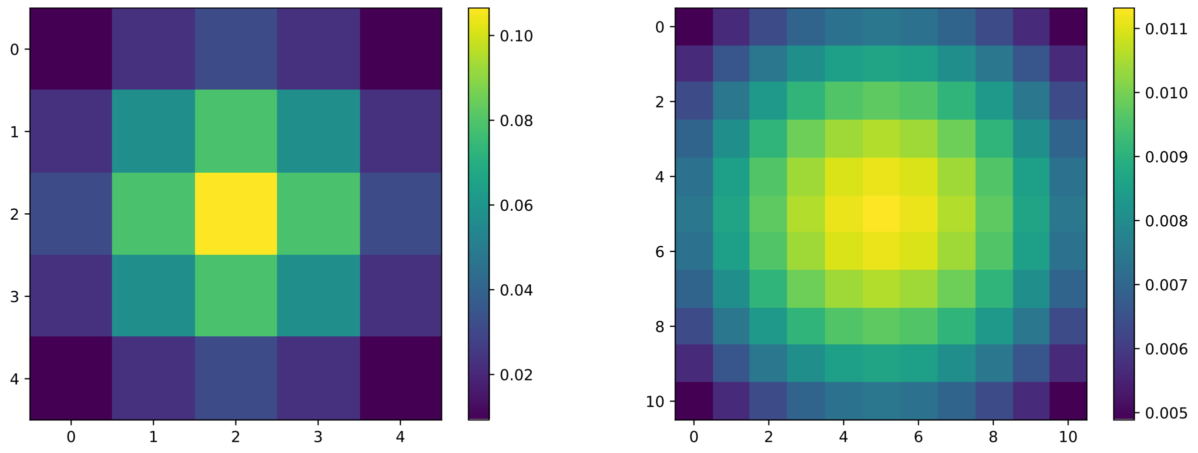

Figure 1: Question 4 Step I; Visualizations of Gaussian kernels generated, ***(left)*** size $= 5 \times 5$, $\sigma = 1.6$, and ***(right)*** size $= 11 \times 11, \sigma = 6$

2.1.2 Step II - Gradient Magnitude

Please check my implementations inside `a1_code.ipynb` submitted along with this report. Below are descriptions of functions written for this question:

- `flip_both_sides(kernel)` takes in a kernel and flips the kernel along both axis (horizontally and vertically)
- `pad_image(image, pad_sizex, pad_sizey)` pads the image with zeros around. For $m \times n$ image input, the output is $(m + 2\texttt{pad\_sizey}) \times (n + 2\texttt{pad\_sizex})$
- `handle_kernel_even(kernel)` handles kernel to make sure they have odd side lengths. It does so by padding the kernel with zeros at appropriate sides, for example

$$\mathcal{K} = \begin{bmatrix} -1 & 0 \end{bmatrix} \quad \text{will be transformed} \quad \Longrightarrow \quad \mathcal{K}' = \begin{bmatrix} 0 & -1 & 0 \end{bmatrix} \tag{2.1}$$

- `conv_full(image, kernel)` convolves image with kernel. Handles padding with correct size and flipping of kernel before convolution. Using vectorized inner product between matrices to aid speeding up the computation the convolution result. For input image size $(m \times n)$ the output should also be size $(m \times n)$
- `calc_grad_magnitude(image, gx=..., gy=...)` calculates the finite difference derivative of image through convolution with `gx` and `gy`. For image size of $(m \times n)$, returns the gradient magnitude matrix of size $(m \times n)$ by convoluting the image with `gx` and `gy`.

2.1.3 Step III - Threshold Algorithm

Please check my implementation of `auto_threshold(image_grad_mag, eps=0.1)` inside file `a1_code.ipynb` submitted along with this report.

2.1.4 Step IV - Test

Please check `get_edges(filename=..., kernel=..., thres_eps=...)` inside file `a1_code.ipynb` submitted along with this report.

Example Outputs Feeding the function with images provided and one image of my own choice, produced Figures 2, 3, and 4.

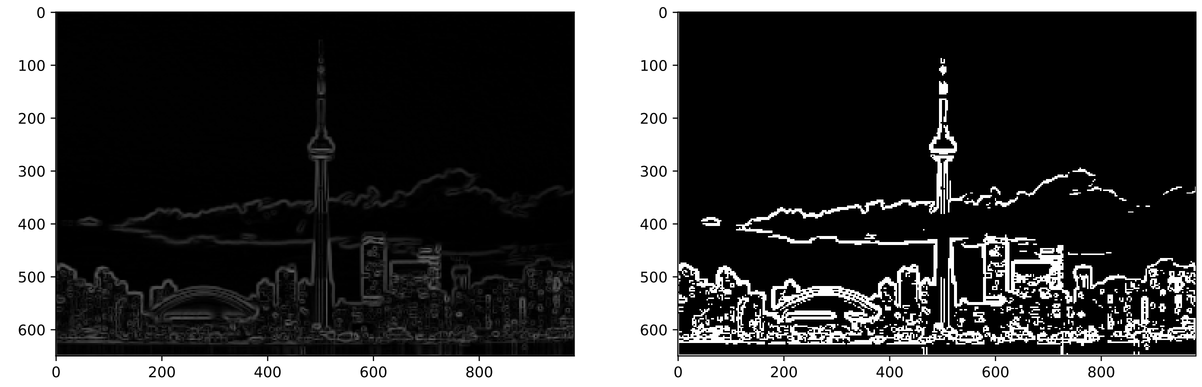

Figure 2: Question 4 Step IV; ***(left)*** Gradient strength `Q4_image_1.jpg` through convolution with Sobel filters, and ***(right)*** thresholded binarized result of gradient strength.

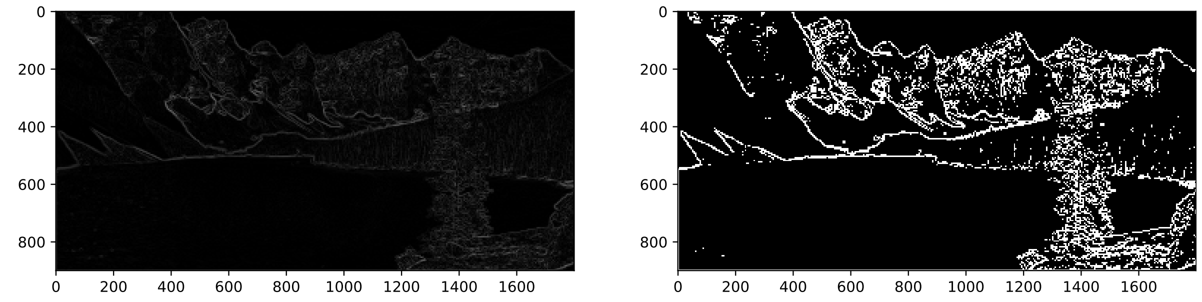

Figure 3: Question 4 Step IV; ***(left)*** Gradient strength `Q4_image_2.jpg` through convolution with Sobel filters, and ***(right)*** thresholded binarized result of gradient strength.

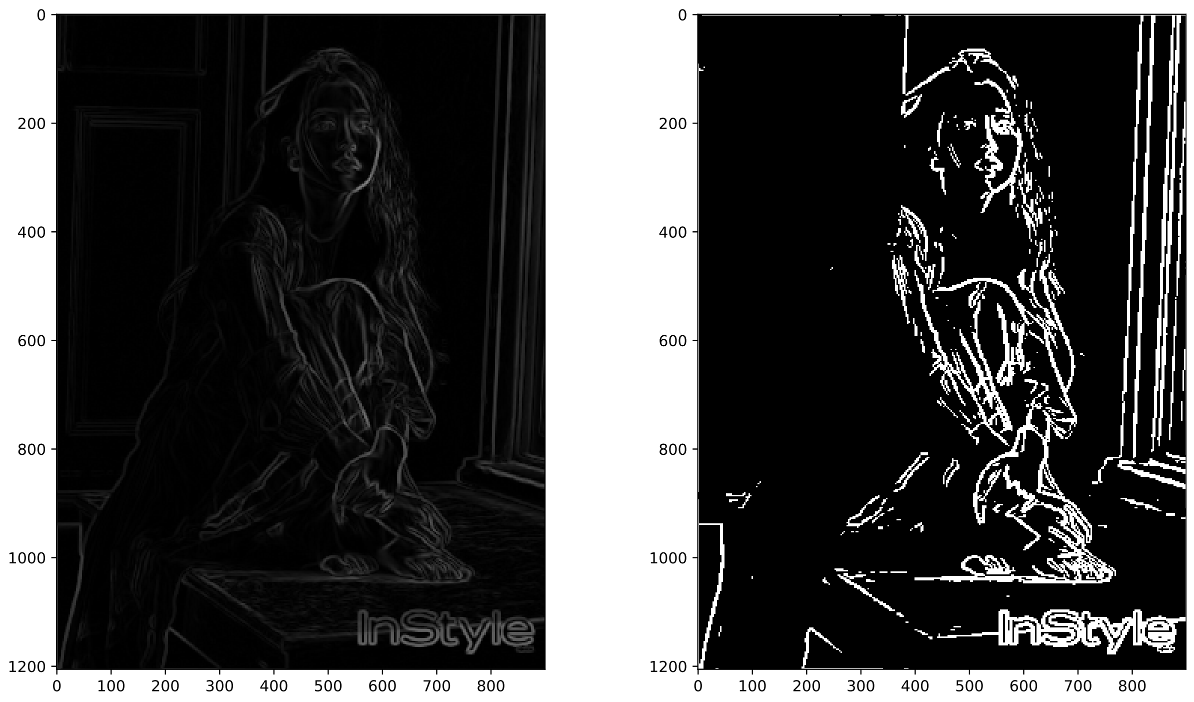

Figure 4: Question 4 Step IV; ***(left)*** Gradient strength `Q4_kim_jisoo.jpeg` through convolution with Sobel filters, and ***(right)*** thresholded binarized result of gradient strength.

Analysis

- The algorithm was able to neglect gradual changes in image, for example the sky in the Toronto skyline picture. So this is good.
- In images with wide range of light intensity, the edges in part with low amount of light is not detected. For example in the lake image, the edges on the left most mountain is not detected. I think this is due to the fact that our thresholding algorithm only has one global thresholding value for the image. For images of this type, segmenting image into parts and apply thresholding separately may help.
- For my choice of image `Q4_kim_jisoo.jpeg`, the edges high lighted seems not too fluid (they are somewhat discrete), and I believe this could be improved using, for example, Hysteresis Thresholding discussed during lecture.

2.2 Q5) Connected-Component Labelling

Please check my implementation of `connected_comp_lab(image)` inside file `a1_code.ipynb` submitted along with this report. The function takes in an image (NumPy array of binarized image), and returns a matrix of the same size as input consisting of label assigned for each pixel. Note that 0 means unlabelled, and corresponds to background.

2.3 Q6) Count the Number of Cells

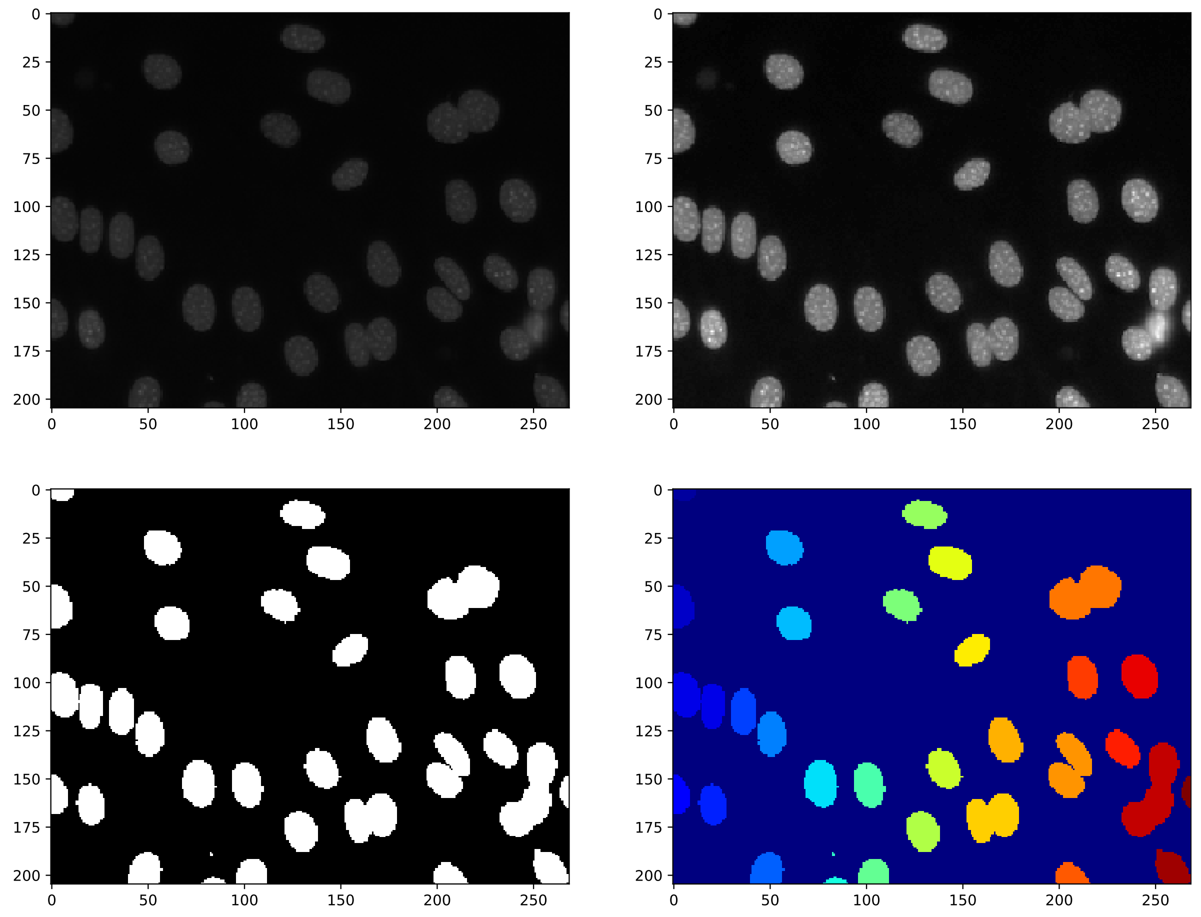

Figure 5: Question 6 Count the Number of Cells; ***(top left)*** Original cell image, ***(top right)*** Grey scale cell image, ***(bottom left)*** binarized cell image using auto thresholding, and ***(bottom right)*** colour coded cells - each blob has a different colour and corresponds to a different label . **The algorithm reports an estimated final result of 32 cells in the image.**

Please check my implementation of `report_num_cells(filename=...)` inside file `a1_code.ipynb` submitted along with this report. The function does the following:

- Read the image as a NumPy array
- convert the colour into grey scale
- threshold the grey scale image into matrix of 0 and 255 only with the `auto_threshold` implemented earlier in Question 4) Step IV
- run the Connected Component Labelling algorithm written in Question 5) on the binarized image
- plot the intermediate and final results for visualization and report final count of cells.

Figure 5 shows the visualizations produced. **The algorithm reports an estimated final result of 32 cells in the image.**

Analysis The estimate result is pretty good. I counted myself and had a result of 36 cells, which is quite close to the result of 32 produced.

We clearly can see that some cells got connected together during the thresholding, and we want to avoid this. (e.g. the pair at the bottom left corner) The ones that overlap will be harder to deal with but we can at least try to avoid the connection between originally disjunct ones. I think one way to do so is to detect edges first, sine the contrast of back back ground between the cells and the white colour of cell would create a large gradient strength.

Near bottom to the left, there is an extremely small blob, and it is also recognized as a cell. We should limit that only blobs bigger than a threshold (containing more than ... pixels) to be cells.