NUS-ISS
Vision Systems

Module 4 - Foundations of computer vision system (3) - Global feature and representation, part 1

Dr. Tan Jen Hong
Lecturer & Consultant
Institute of System Science
National University of Singapore
issjht@nus.edu.sg

Learning objectives

- Perform image filtering
- Perform edge detection
- Extract objects from image
- Draw contour around objects

Undesirable effect from uint8

- Let's investigate uint8, declare two variables, assumed already import numpy as np

```
> A = np.uint8(5)
> B = np.uint8(8)
> C = A - B
```

- What is the output of C?

- The way numpy / python handles the above situation is problematic in image processing

- Thus, many times we need to convert uint8 to float32

Handling something beyond boundaries ...

- Many times, the values of the output of a processing exceeds the usual range
- The values either exceed 255 or go into negative (for integer)
- The values either exceed 1 or below 0 (for float)
- There is a need to rescale the values back to the desired range
- We call this process normalization

Handling something beyond boundaries ...

- Let x denote the pixel values in an image; let oldMin and oldMax denote the minimum and the maximum values respectively in the image

- let x_n denote the normalized value; let newMin and newMax denote the desired minimum and maximum values

- The formula to rescale / re-normalize:

$$\frac{x - \text{oldMin}}{\text{oldMax} - \text{oldMin}} = \frac{x_n - \text{newMin}}{\text{newMax} - \text{newMin}}$$

Handling something beyond boundaries ...

- Re-arrange and we get:

$$x_n = \frac{x - \text{oldMin}}{\text{oldMax} - \text{oldMin}}\left(\text{newMax} - \text{newMin}\right) + \text{newMin}$$

- Exercise: Create a function with the below signature to normalize an image (the `img` must be a float)

```
> def imgNormalize(img, minv=0, maxv=1):
```

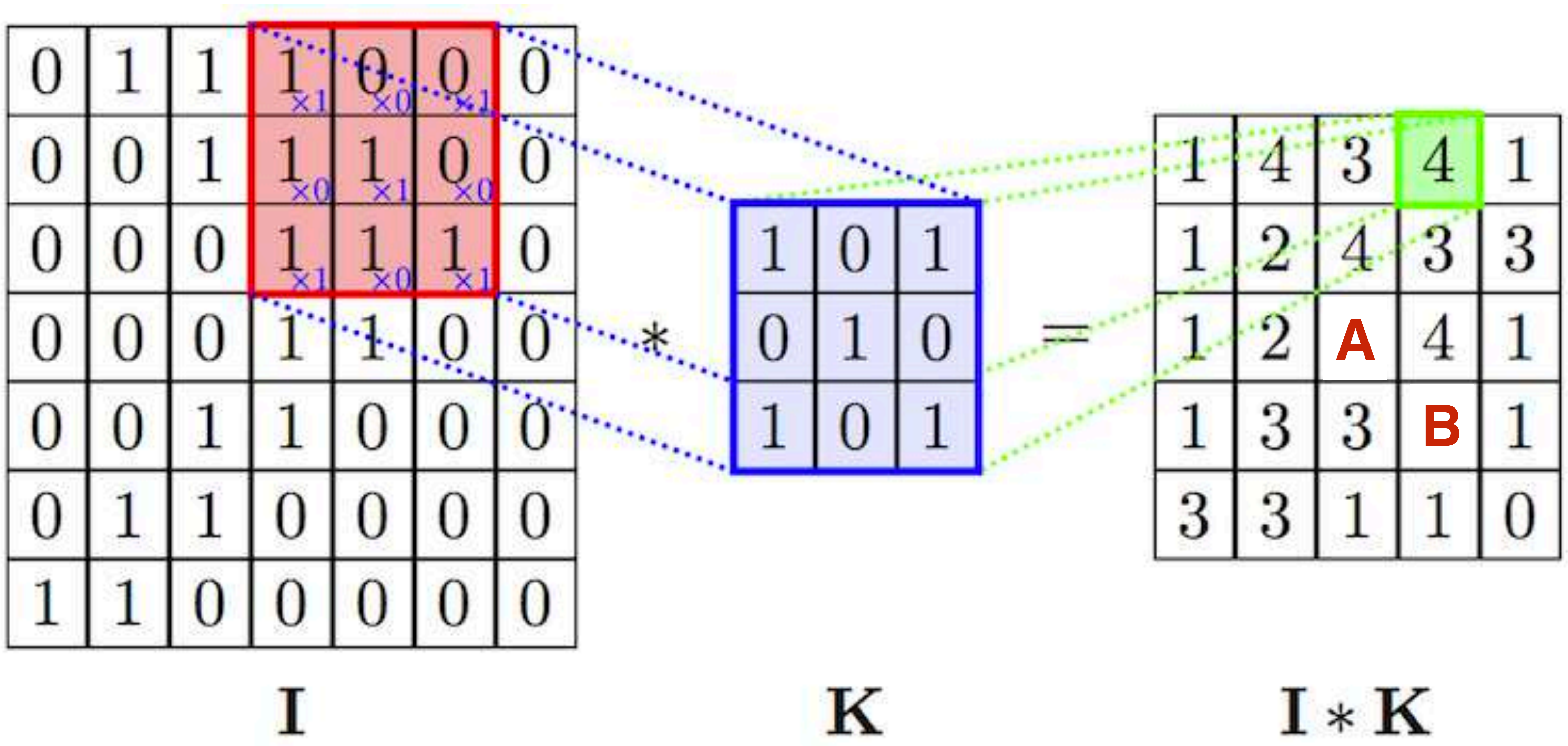
Image filtering

- In image processing, filtering is used to strengthen preferred features or weaken unwanted features
- We use filtering to smooth or sharpen an image; also, to enhance edges in image
- Filtering is achieved through convolution; it involves a kernel and the image of interest
- The design of the kernel determines the filtering output

Source: https://www.star-spain.com/en/blog/transittermstar-nxt-tooltips/filtering-data-records-termstar-nxt

2D Convolution

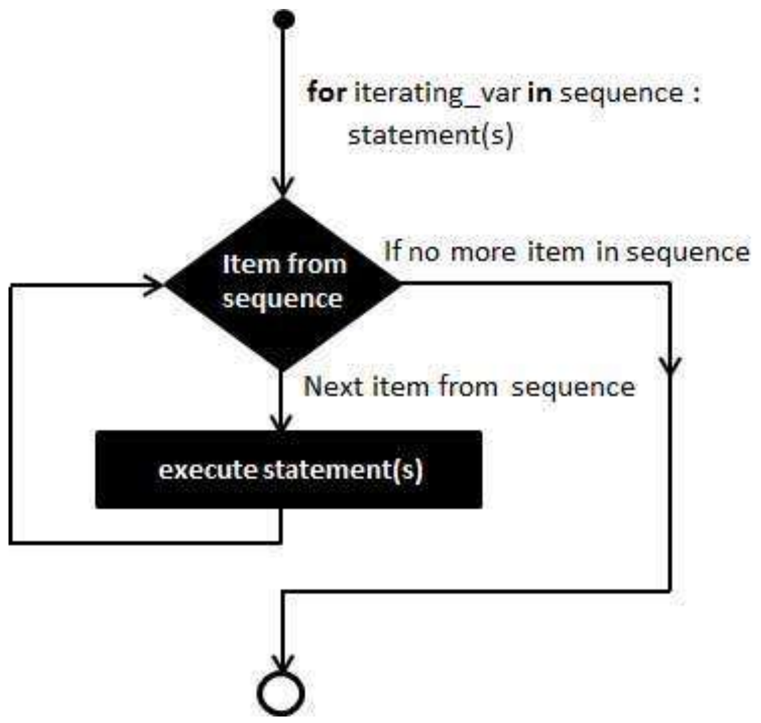
- What is the value of A and B?



Source: https://www.researchgate.net/figure/An-example-of-convolution-operation-in-2D-2_fig3_324165524

Note on 2D Convolution

- Avoid writing convolution code in python
- for loops in python are very slow; the calculation takes much longer time
- Always use library functions; they are optimized
- If not, write C code



Source: https://www.tutorialspoint.com/python/python_for_loop.htm

Some convolutions

Original

$$\begin{bmatrix} \bullet 0 & \bullet 0 & \bullet 0 \\ \bullet 0 & \bullet 1 & \bullet 0 \\ \bullet 0 & \bullet 0 & \bullet 0 \end{bmatrix} = ?$$

Original

$$\begin{bmatrix} \bullet 0 & \bullet 0 & \bullet 0 \\ \bullet 0 & \bullet 0 & \bullet 1 \\ \bullet 0 & \bullet 0 & \bullet 0 \end{bmatrix} = ?$$

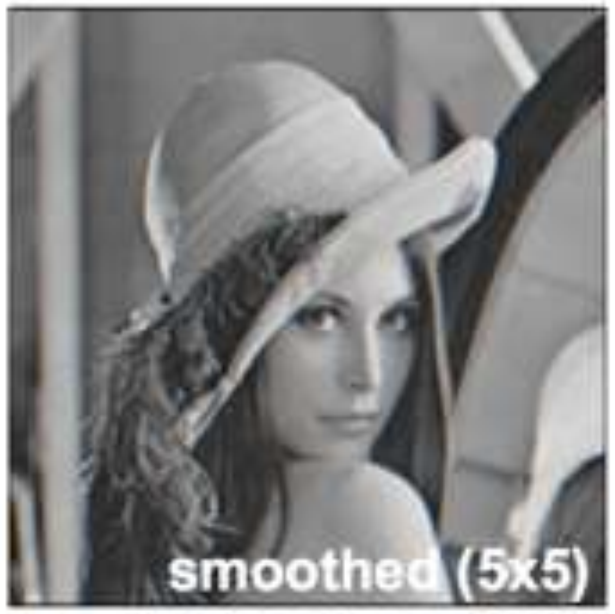
Original

$$\frac{1}{9}\begin{bmatrix} \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \end{bmatrix} = ?$$

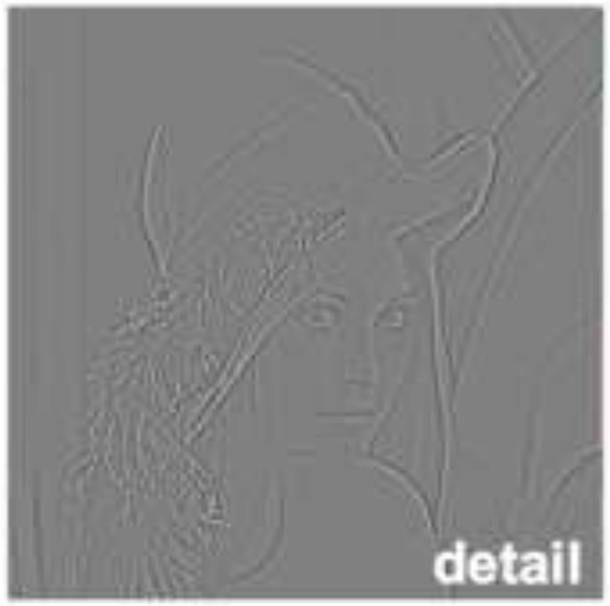
Source: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Some combinations

–



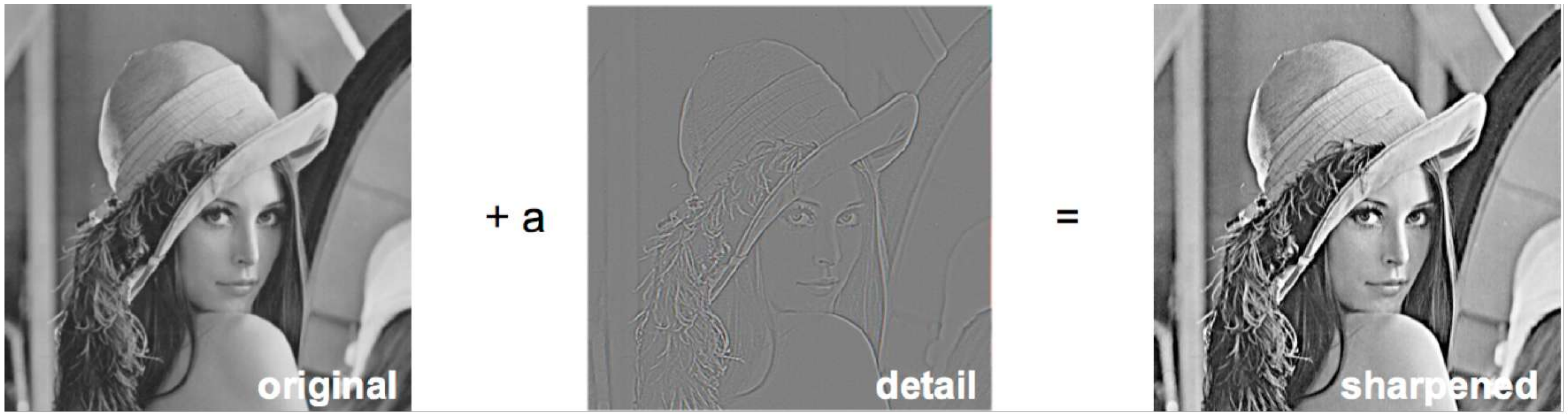
=



$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Source: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Some combinations



$$\begin{bmatrix} \bullet 0 & \bullet 0 & \bullet 0 \\ \bullet 0 & \bullet 1 & \bullet 0 \\ \bullet 0 & \bullet 0 & \bullet 0 \end{bmatrix} + \begin{bmatrix} \bullet 0 & \bullet 0 & \bullet 0 \\ \bullet 0 & \bullet 1 & \bullet 0 \\ \bullet 0 & \bullet 0 & \bullet 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \end{bmatrix} = \begin{bmatrix} \bullet 0 & \bullet 0 & \bullet 0 \\ \bullet 0 & \bullet 2 & \bullet 0 \\ \bullet 0 & \bullet 0 & \bullet 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \end{bmatrix}$$

Source: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Basic filtering

dr3.png

- Load the image and the necessary libraries

```
> import cv2
> import numpy as np
> import matplotlib.pyplot as plt

> dr3 = cv2.imread('dr3.png')
```

- Create the kernel

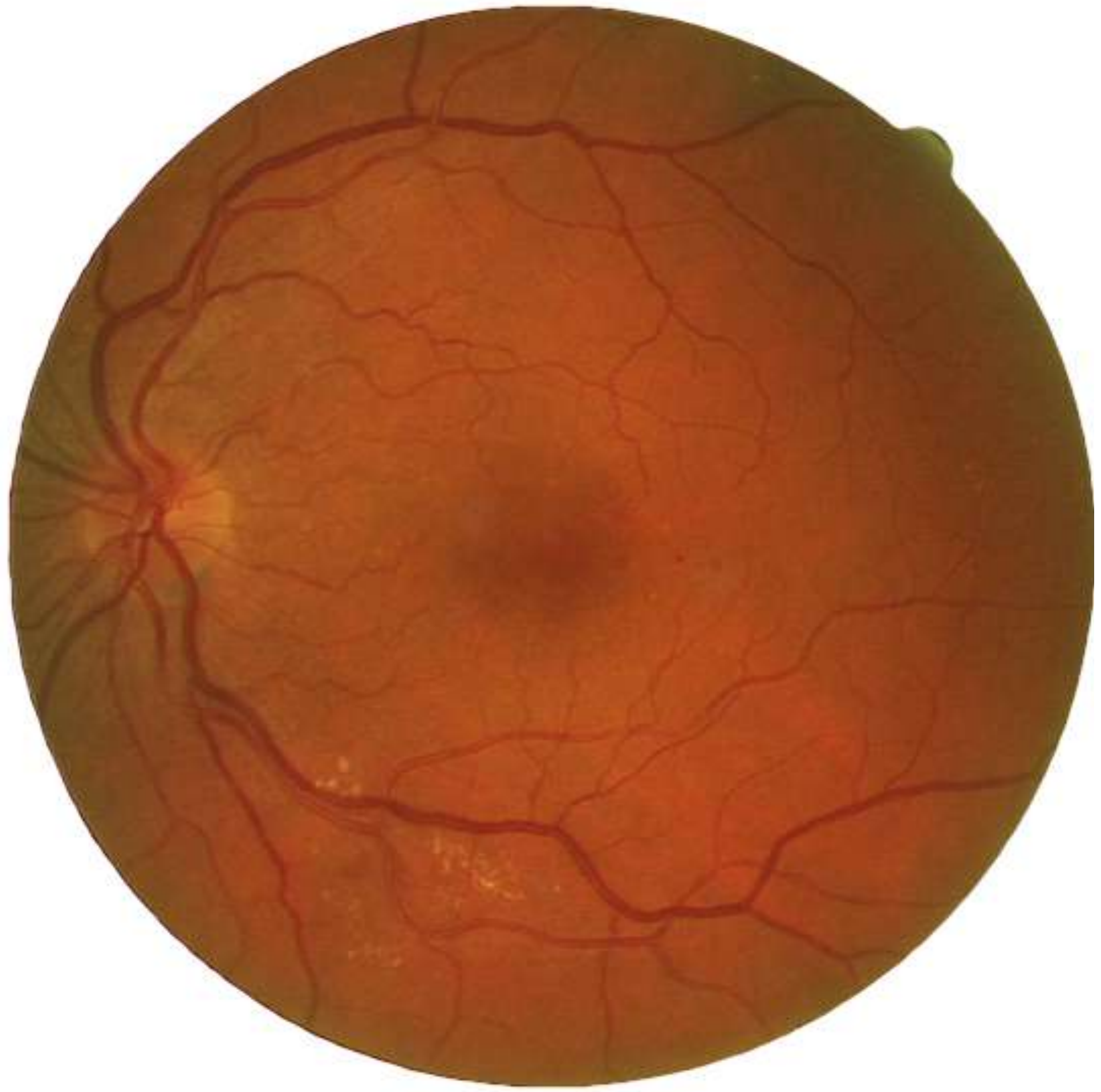
```
> knl = np.ones((7,7),
                      np.float32)/49
```

- Do the filtering

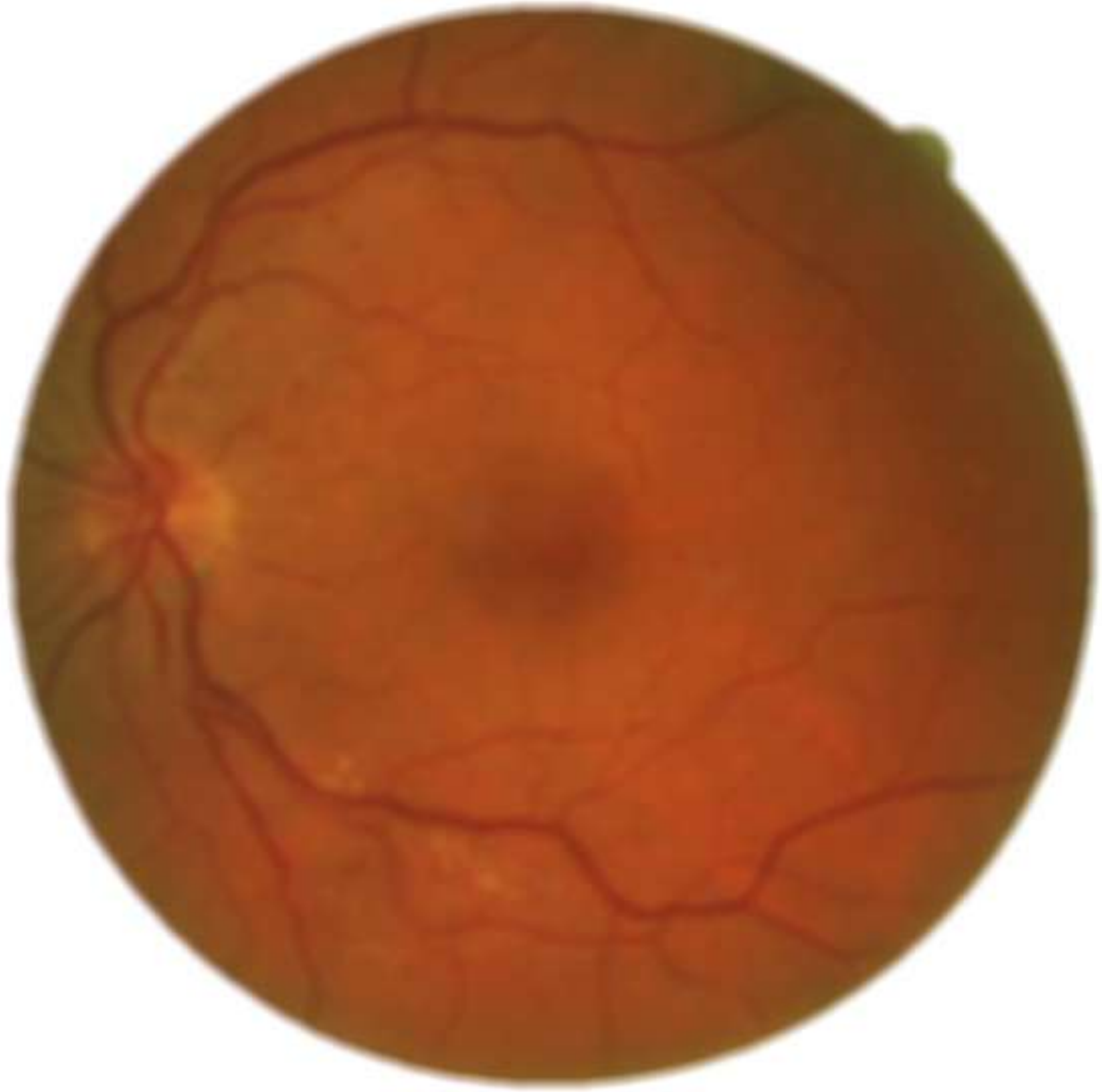
```
> bsc = cv2.filter2D(dr3,
  intensity depth, just set to -1   -1,
                             kernel   knl)
```

Basic filtering

Original



Filtered



Mean filtering

- Mean filter reduces intensity variation among pixels
- Commonly used to reduce noise in images
- For a kernel of size j x j, each entry in the matrix shares the same value of 1/(j x j)
- Example: a 5 x 5 kernel for a mean filter looks like

$$\begin{bmatrix} \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \\ \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} & \frac{1}{25} \end{bmatrix}$$

$$\frac{M + E + A + N}{4}$$

Source: http://www.webquest.hawaii.edu/kahihi/mathdictionary/M/mean.php

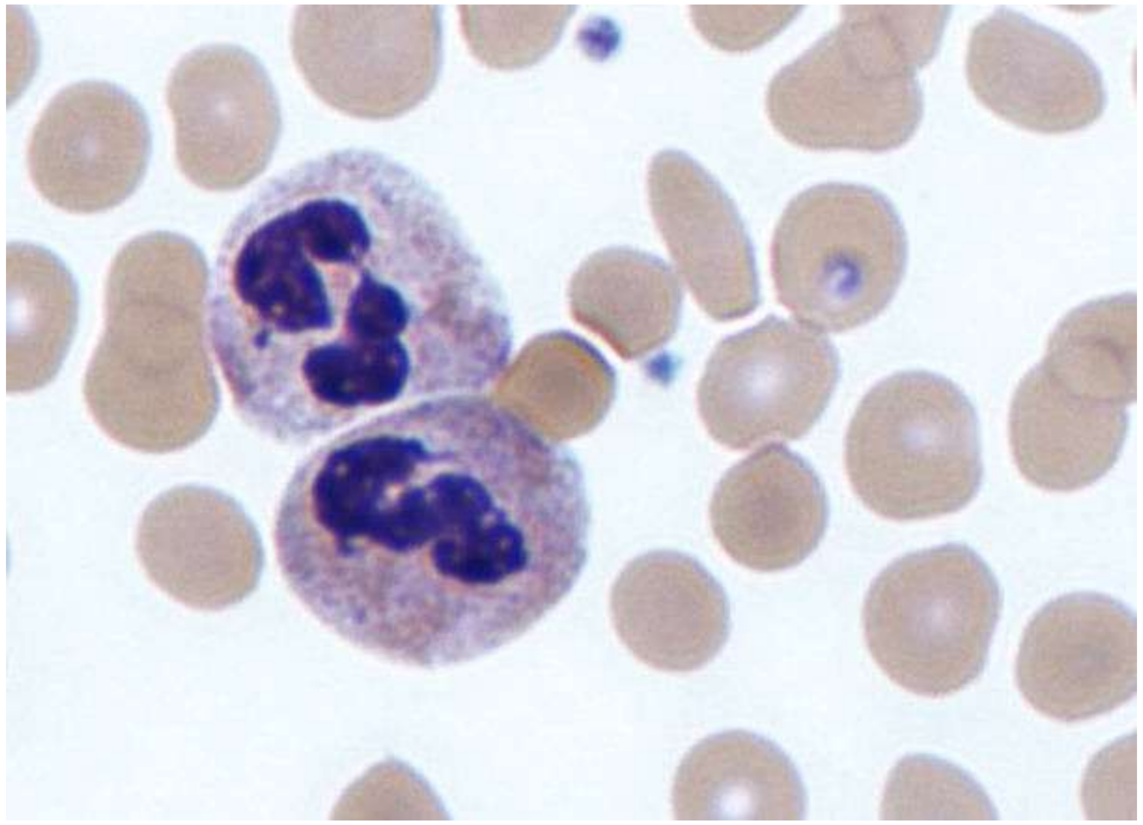
Mean filtering

- In opencv, mean filtering is simply by

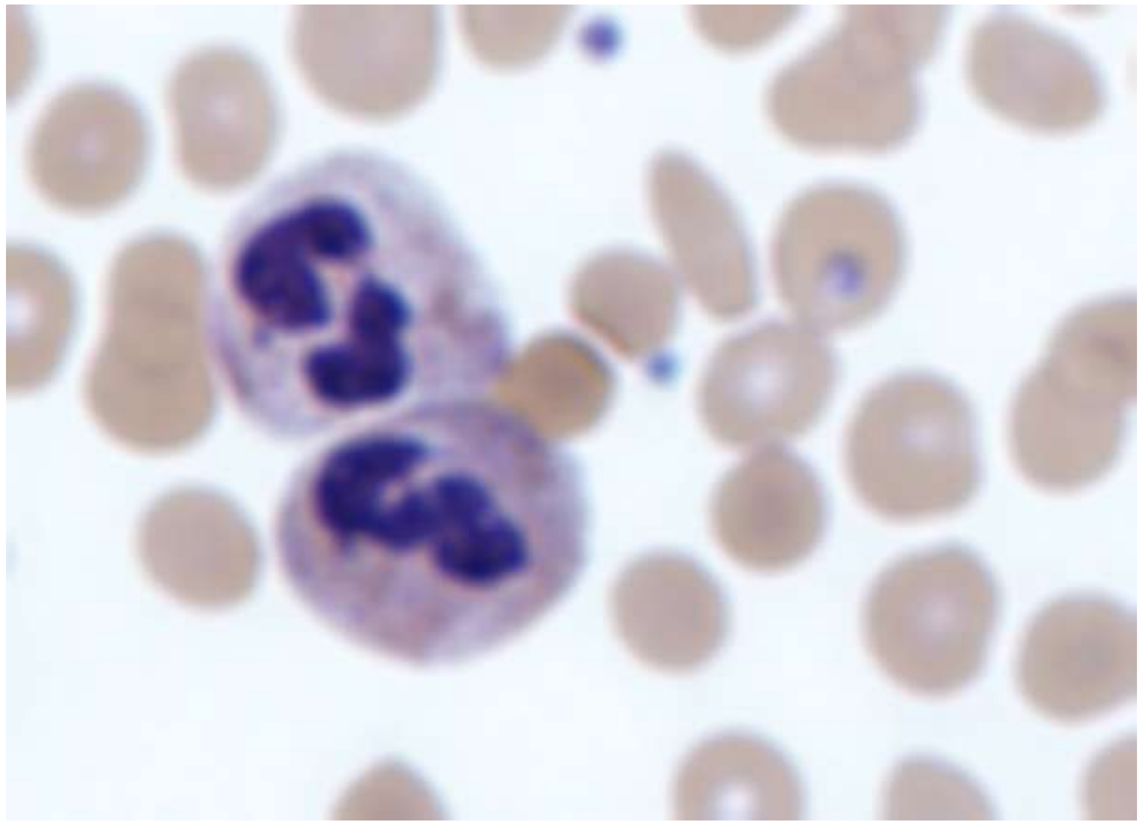
```
> nuc  = cv2.imread('nuc.jpg')
> mflt = cv2.blur(nuc,(11,11))
```

kernel size

nuc



mflt

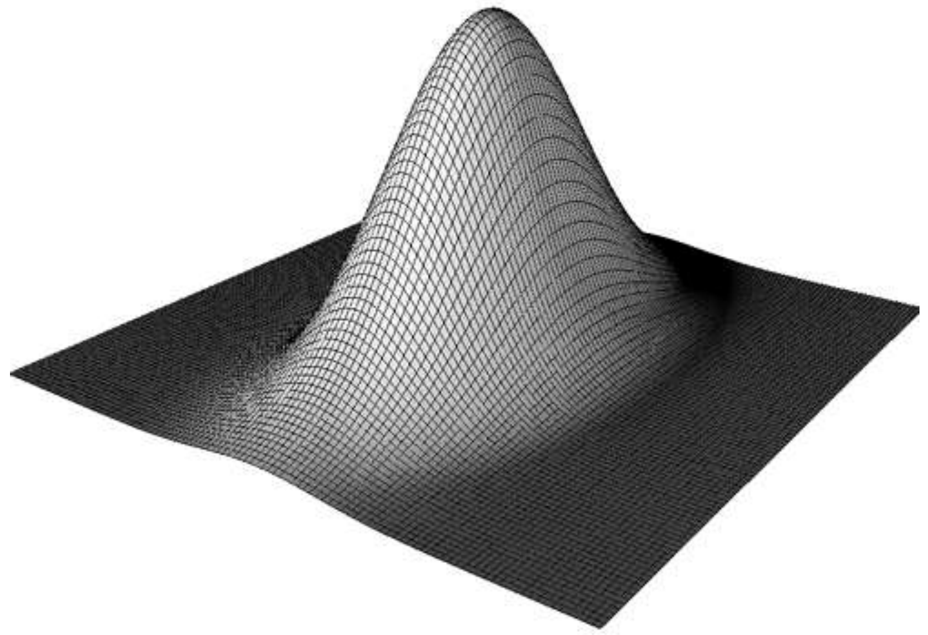


Gaussian filtering

- Gaussian filter: non-uniform low pass filter
- Commonly used to remove noise and detail
- The formula for 2D kernel (isotropic):

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

- When we use Gaussian filter, sigma is the key
- The kernel size should be around 6 x sigma (3 sigma for each side)



Source: https://www.teamliquid.net/forum/starcraft-2/142211-sc2-ladder-analysis-part-2

Gaussian filtering

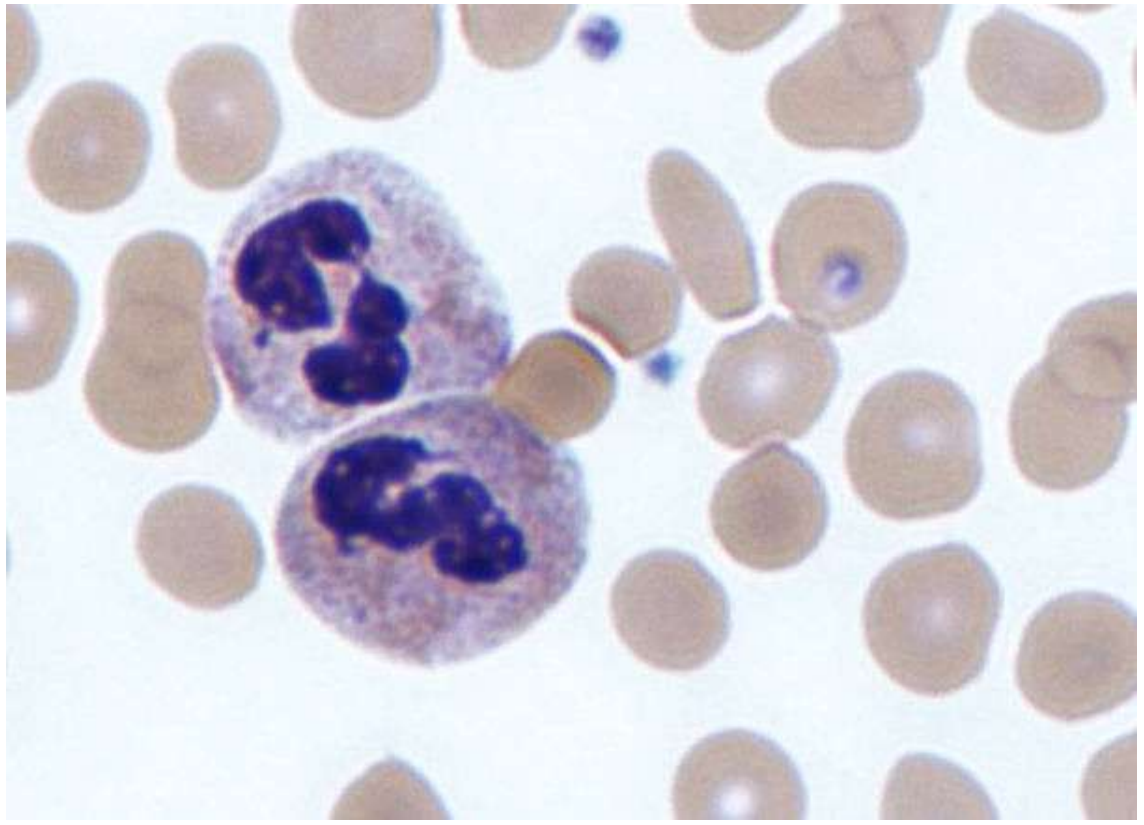
- In opencv, Gaussian filtering is simply by

```
> gaus= cv2.GaussianBlur(nuc,
                         (11,11),
                         0)
```

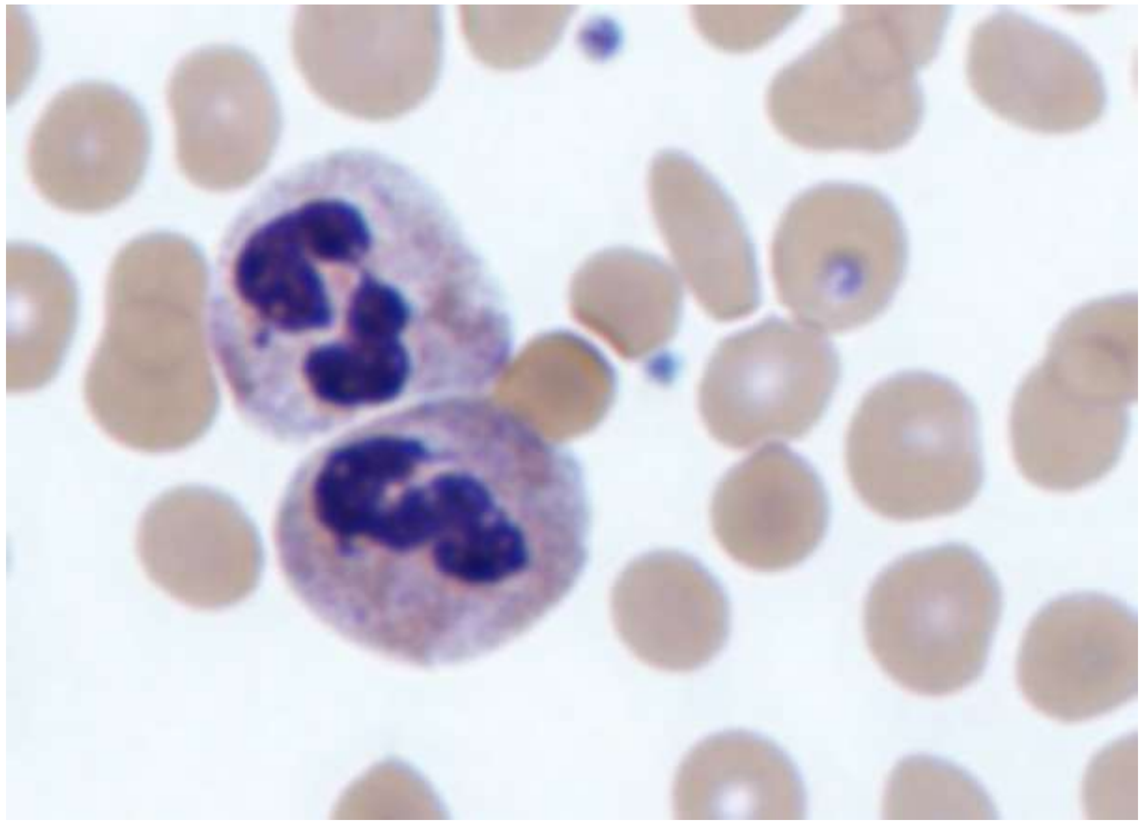
kernel size

sigmaX, set to 0, so that opencv will calculate the sigmaX and sigmaY based on the kernel size

nuc



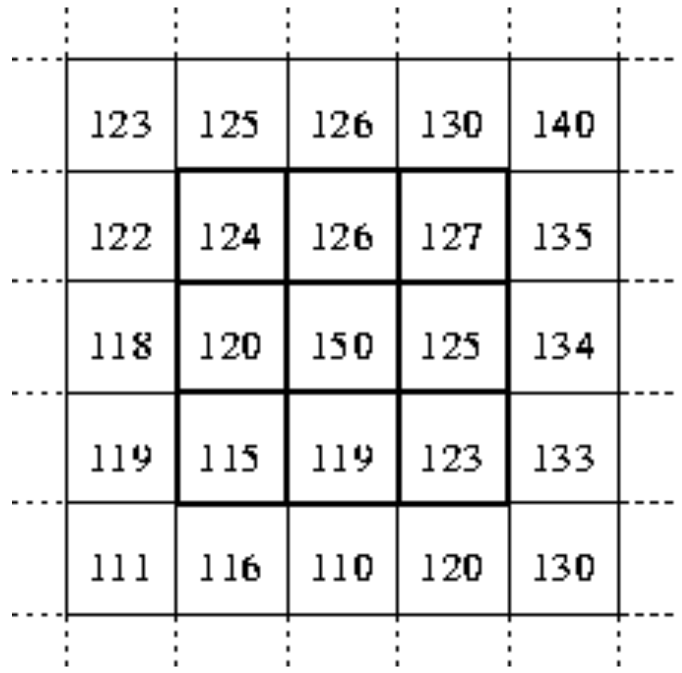
gaus



Median filtering

- Median filter: it considers the neighbourhood around a pixel, order the values, determine the median value and output the median value

- Commonly used to remove noise and detail; do better job than mean filter



123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

Neighbourhood values:

115, 119, 120, 123, 124,
125, 126, 127, 150

Median value: 124

Source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm

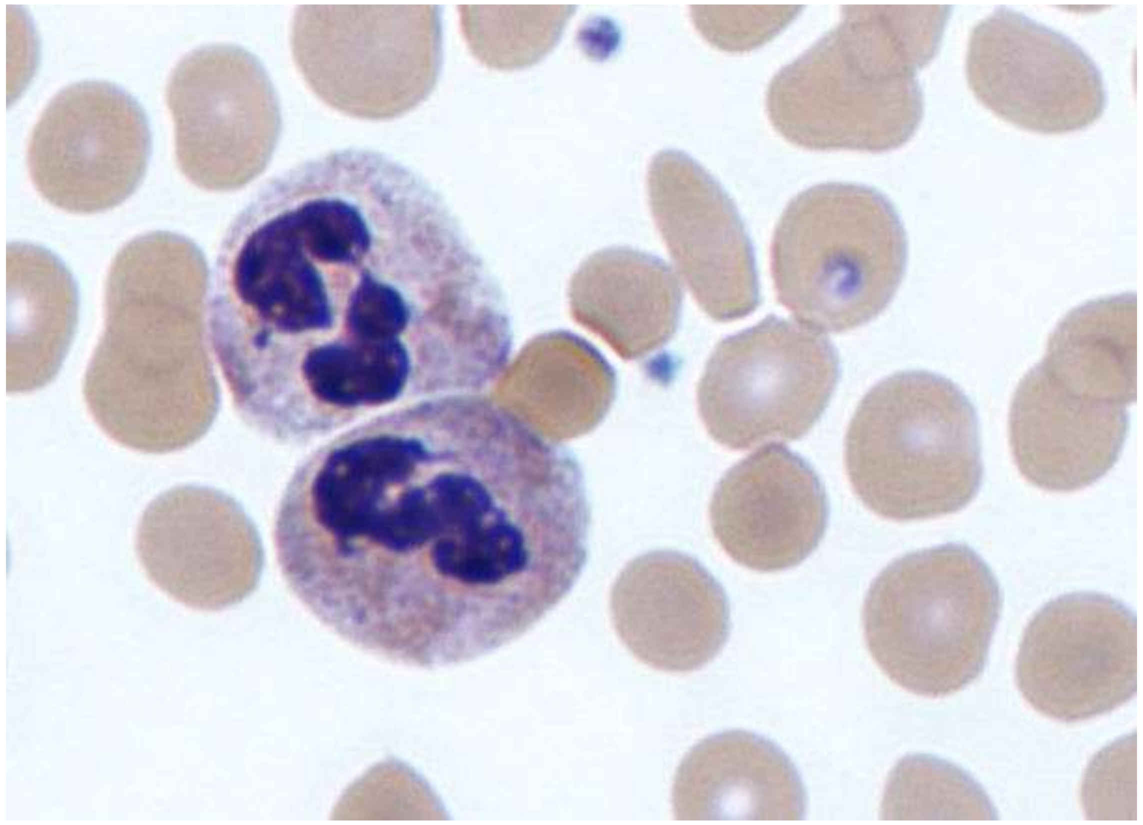
Median filtering

- In opencv, Median filtering is simply by

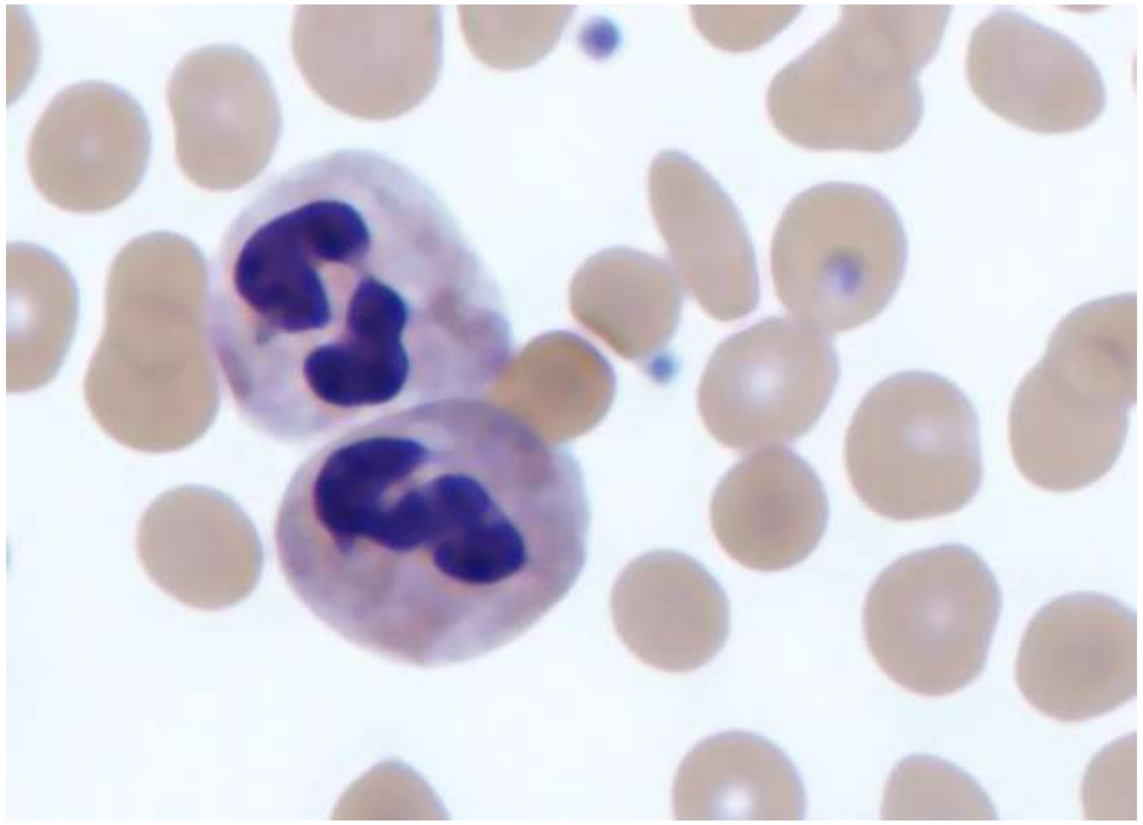
```
> medf= cv2.medianBlur(nuc,11)
```

kernel size

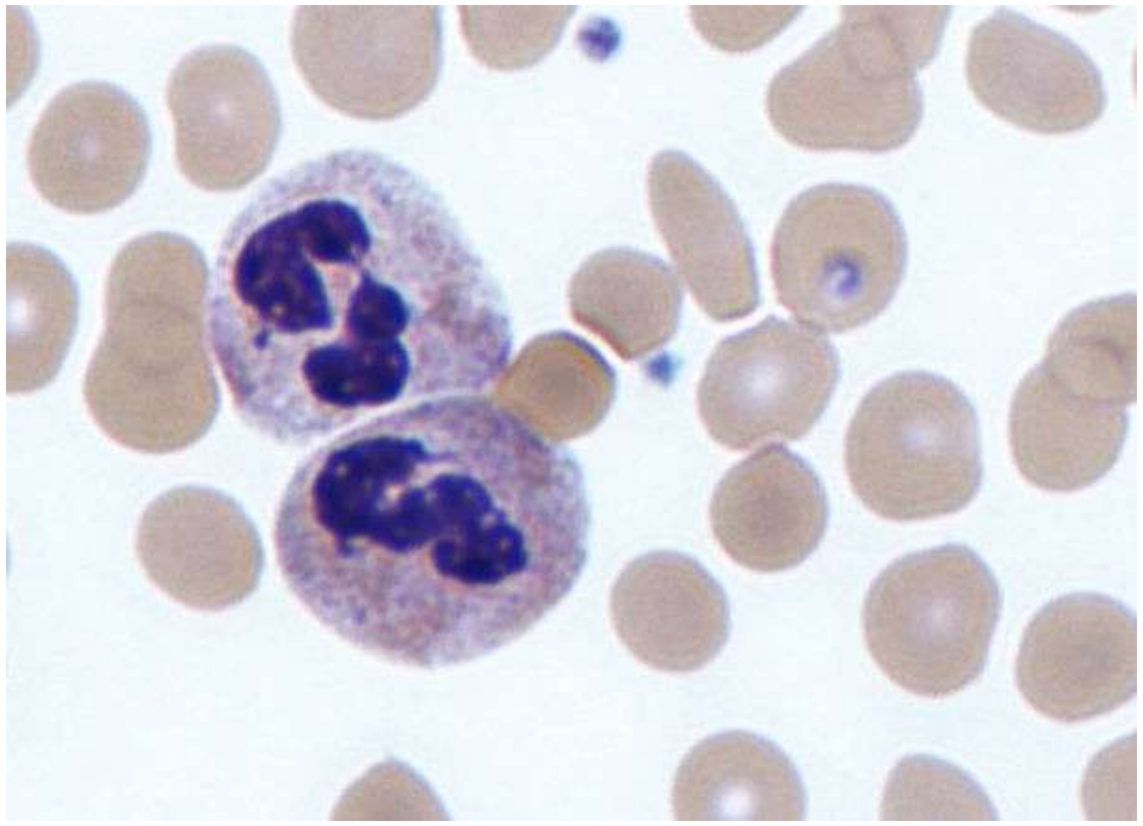
nuc



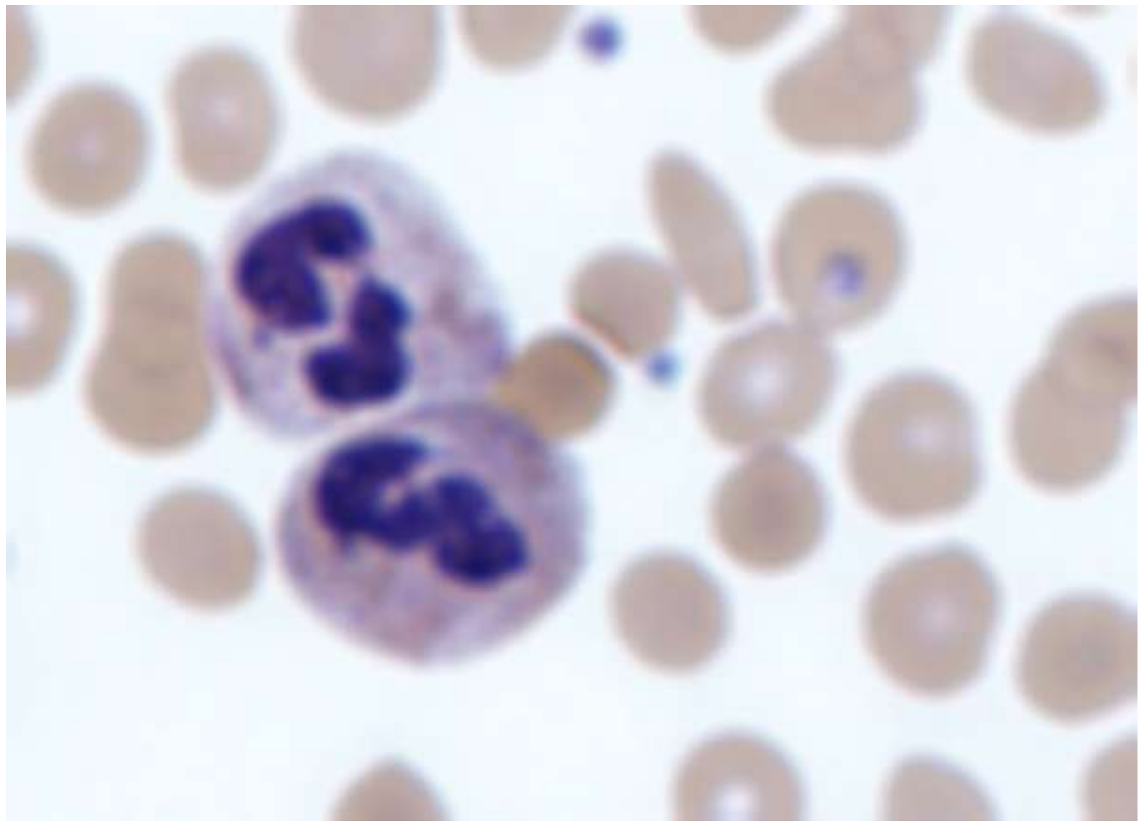
medf



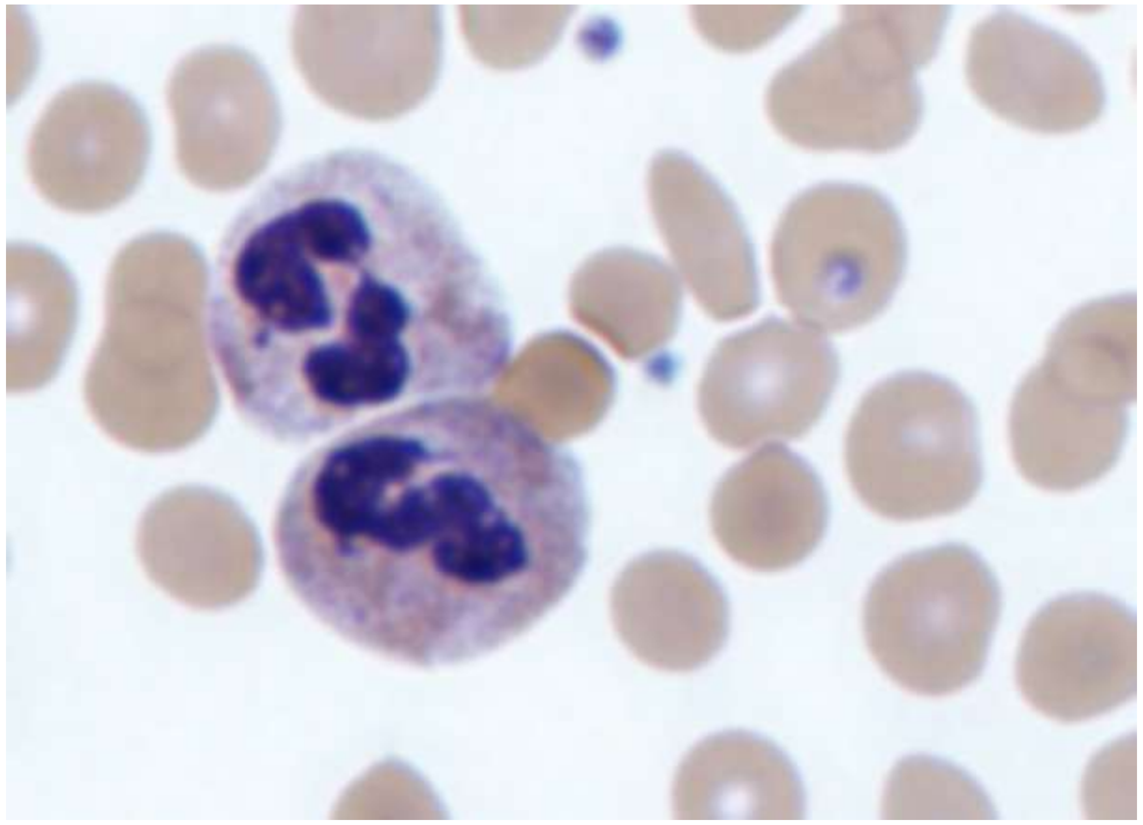
nuc



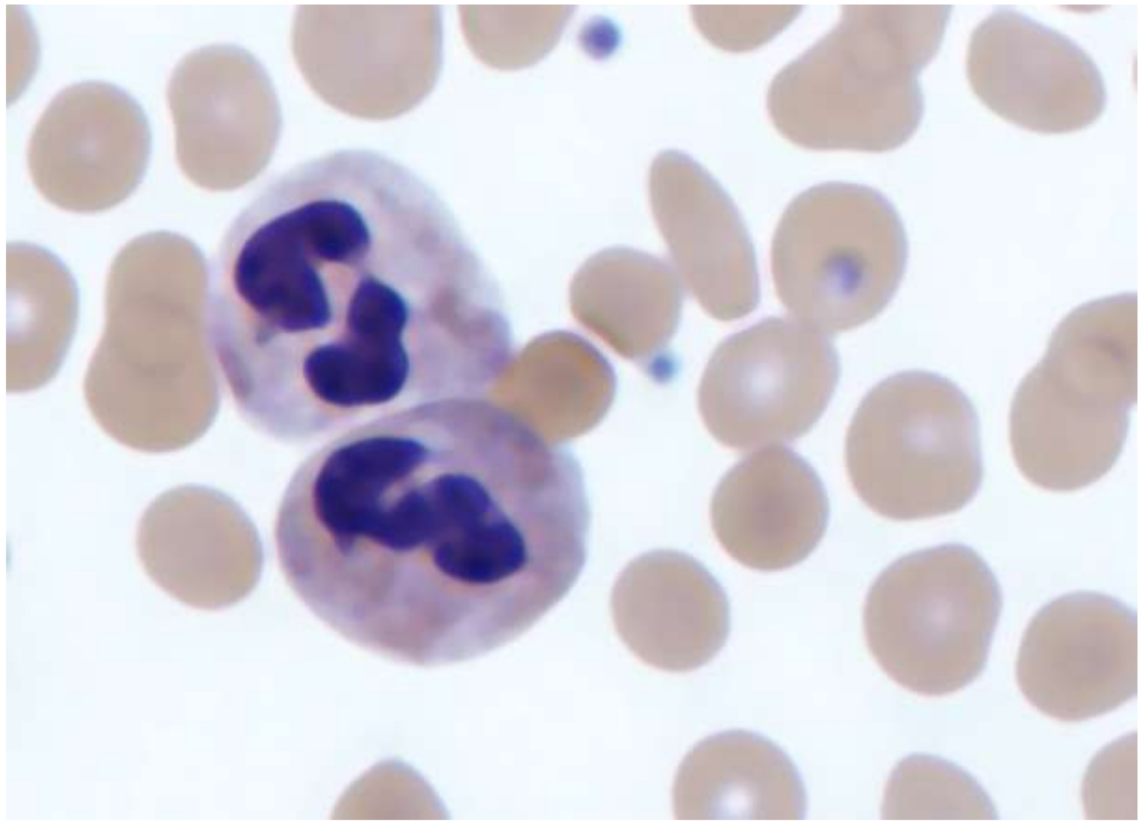
mflt



gauss



medf



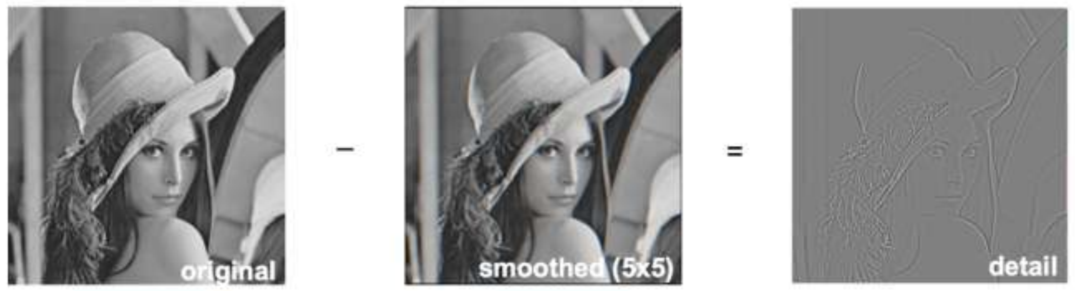
How about sharpen the image?

- Create the mean filter kernel and perform the mean filtering

```
> krn  = np.ones((11,11),np.float32)/121
> mnuc = cv2.filter2D(nuc,-1,krn)
```

- Perform the subtraction and plot the output

```
> detail = (np.float32(nuc) - np.float32(mnuc))/255
> cv2plt(detail)
```

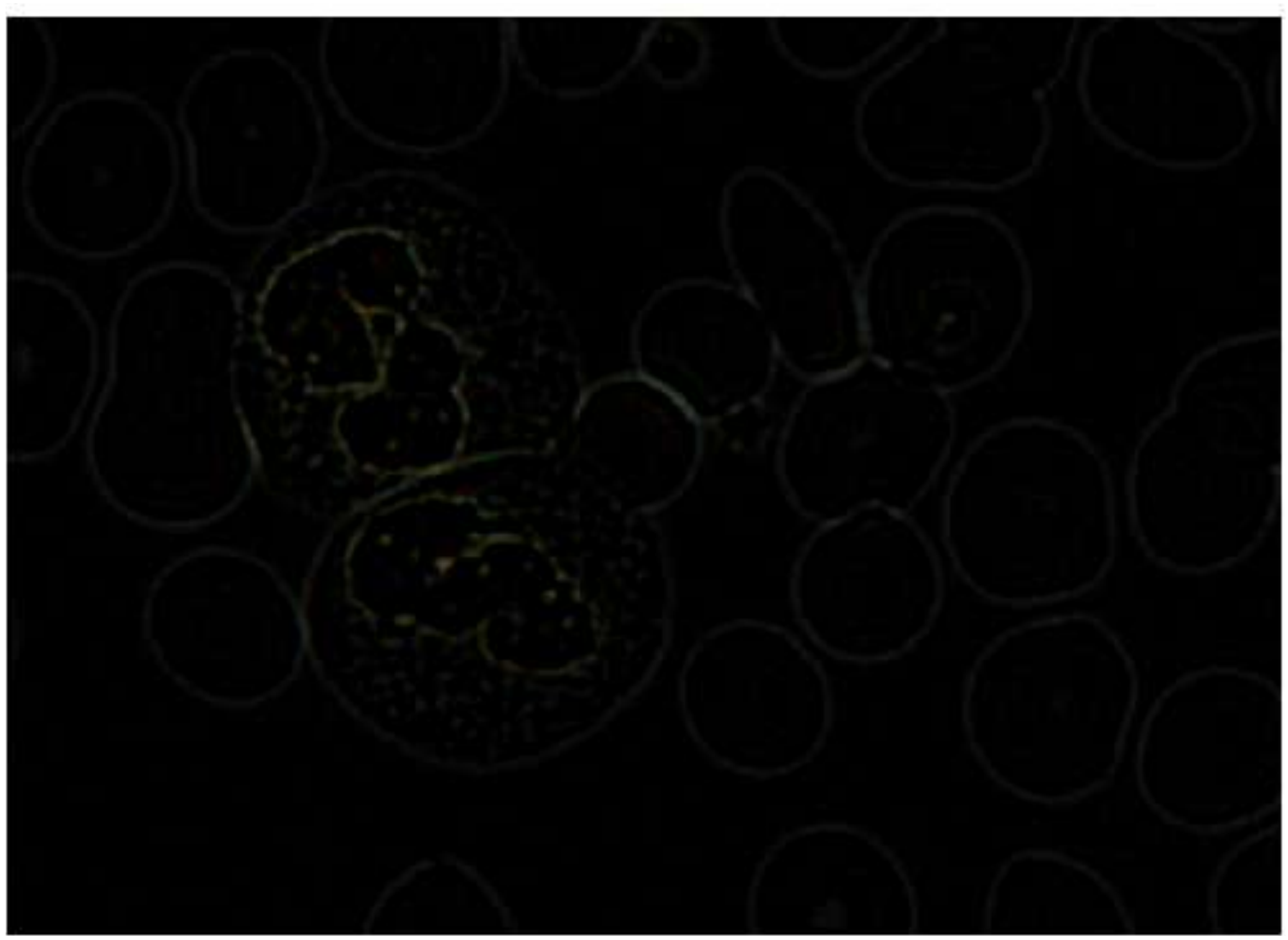


$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Source: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

How about sharpen the image?

- Can't see anything ....
- Let's check the maximum and minimum value
- Max: 0.29;   Min: -0.27



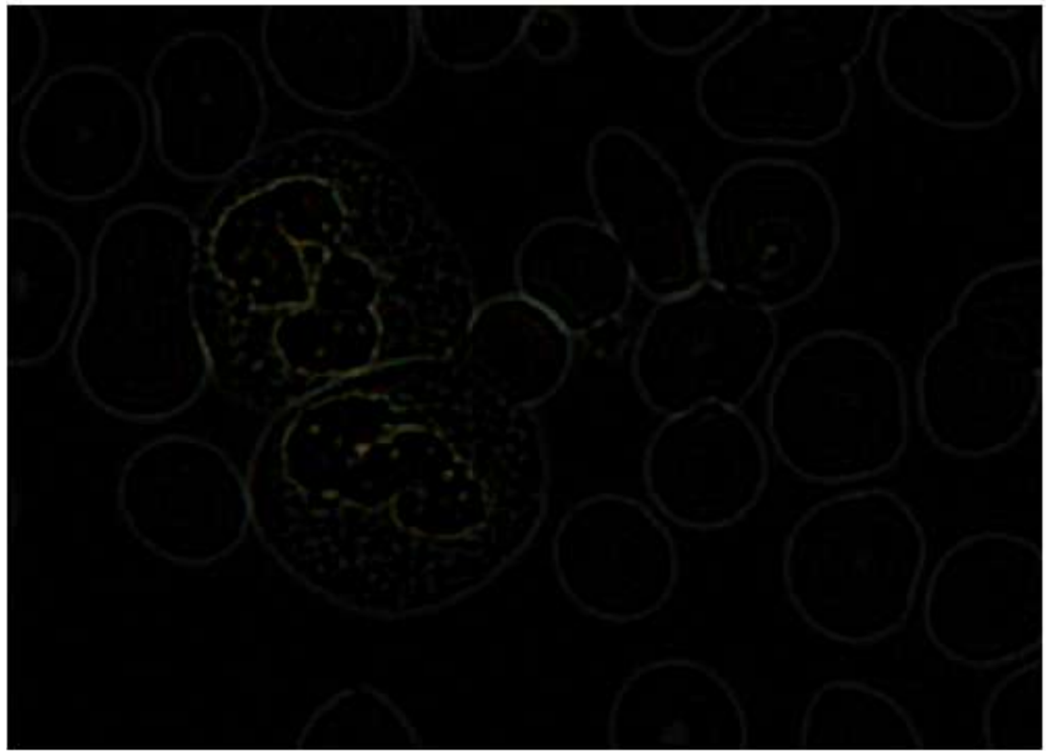
How about sharpen the image?

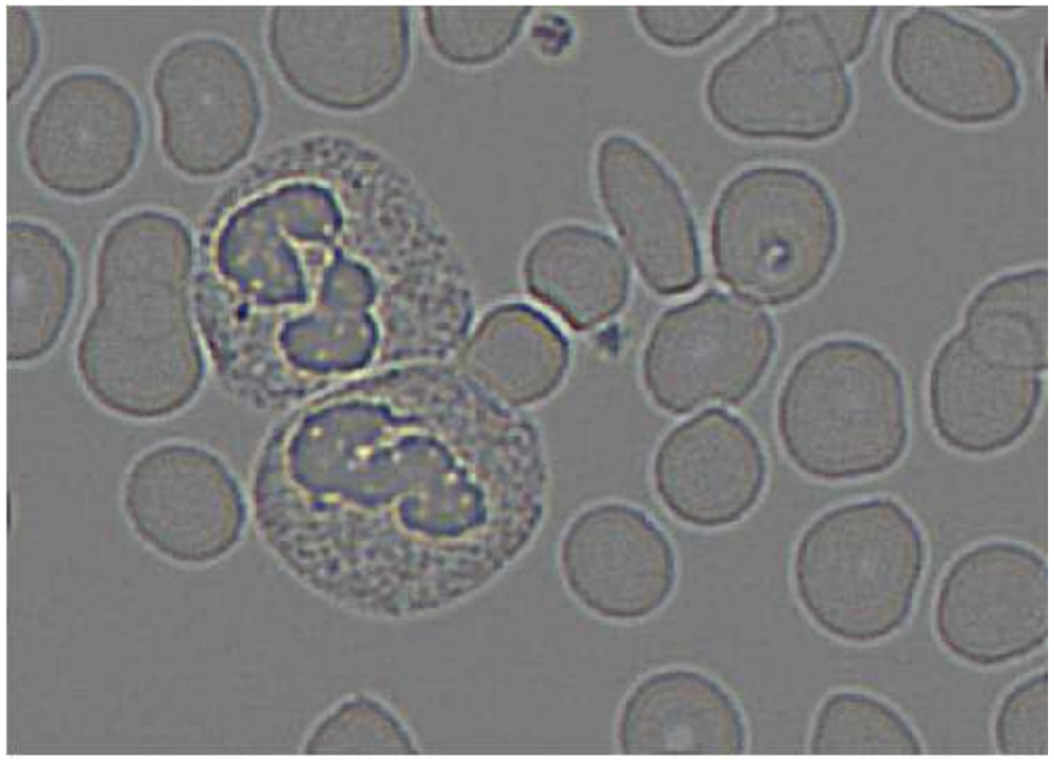
- Rescale the image, use the normalization function

```
def imgNormalize(img, minv=0, maxv=1):
```

- The code to achieve that:

```
> normalizedDetail = imgNormalize(detail)
> plt.figure()
> cv2plt(normalizedDetail)
```





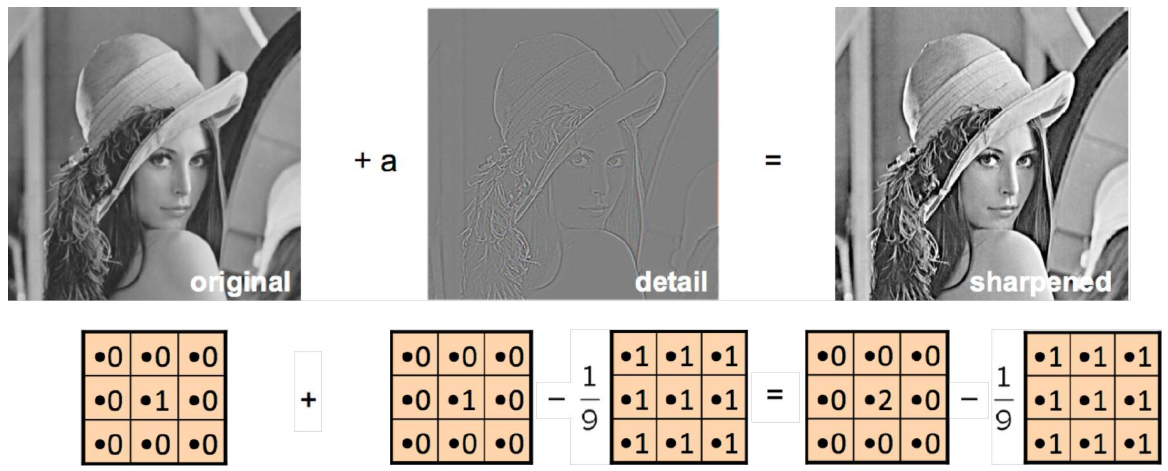
Sharpen the nuc

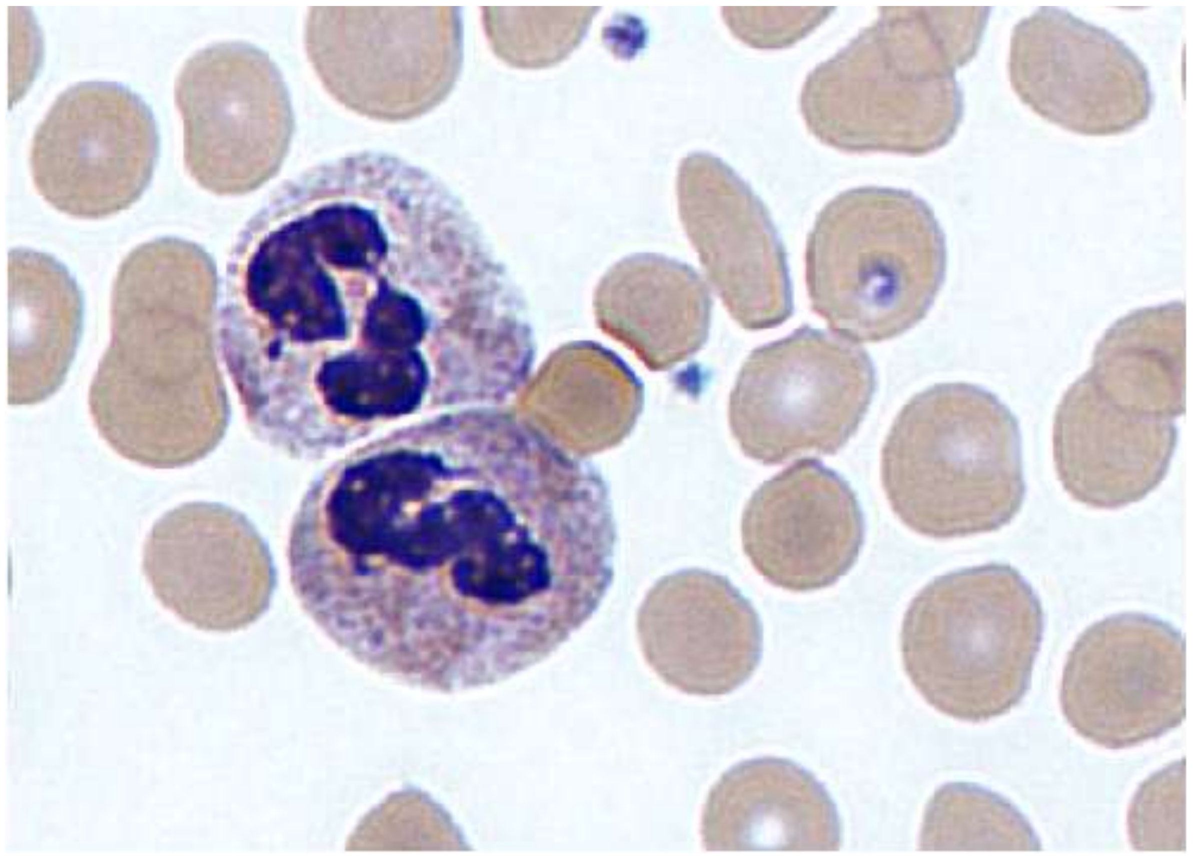
- To sharpen the image, we do

```
> sharpen = (np.float32(nuc)*2 - np.float32(mnuc))/255
```

- And plot the output:

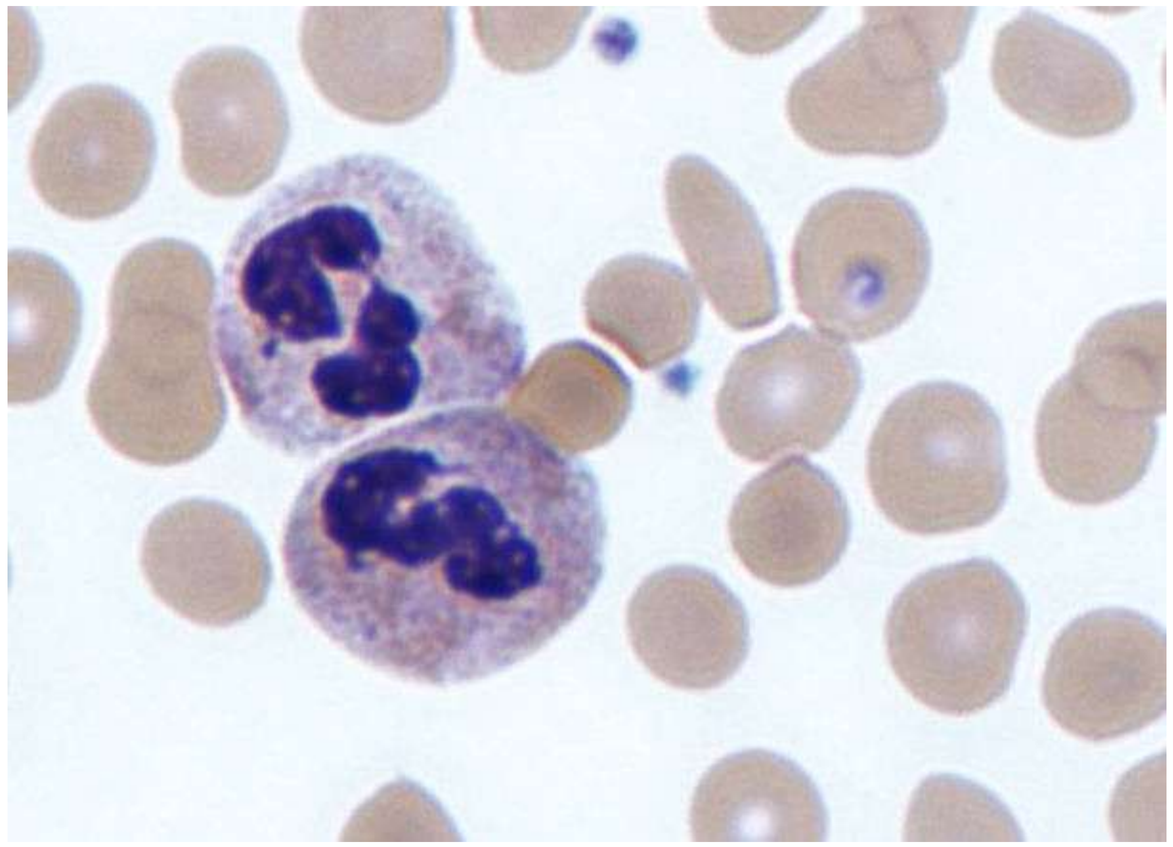
```
> plt.figure()
> cv2plt(normalizedDetail)
```

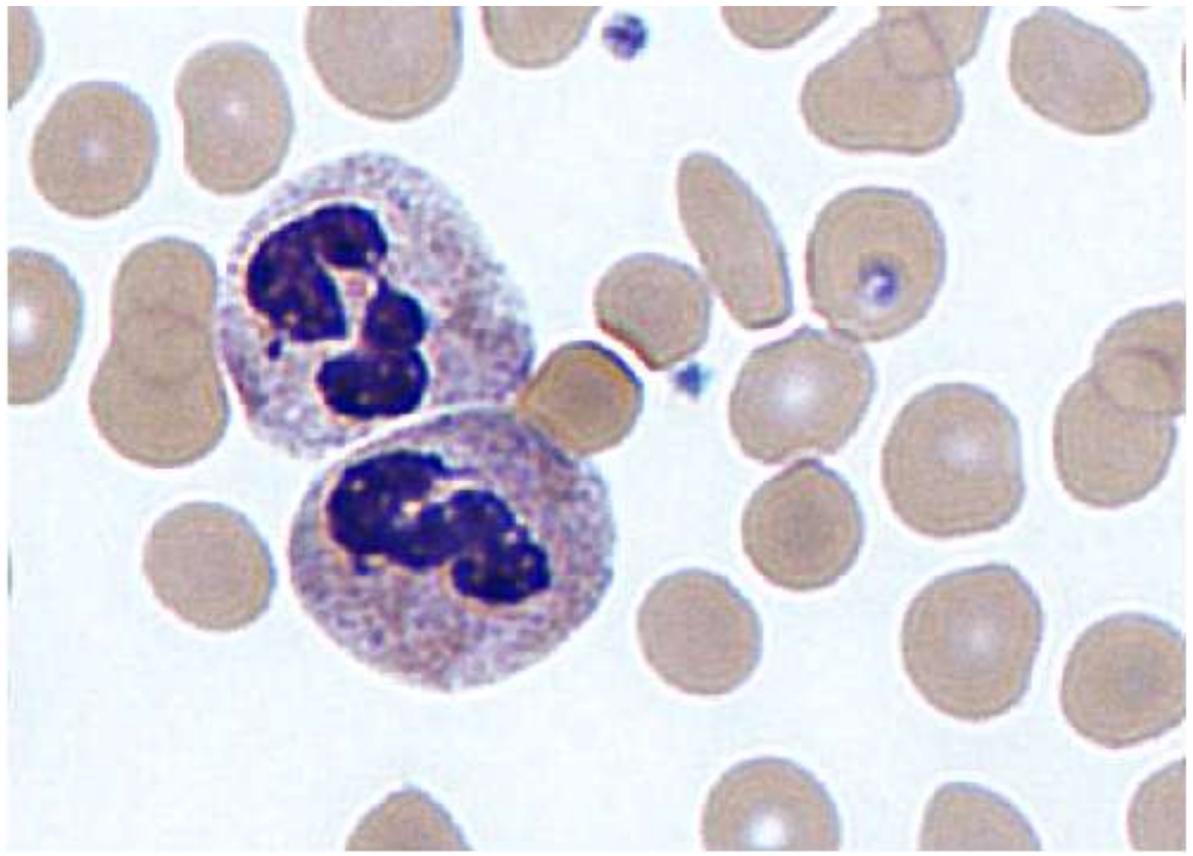




Before and after

- The effect of sharpening: Sharpen the boundaries but also introduce noise in the image

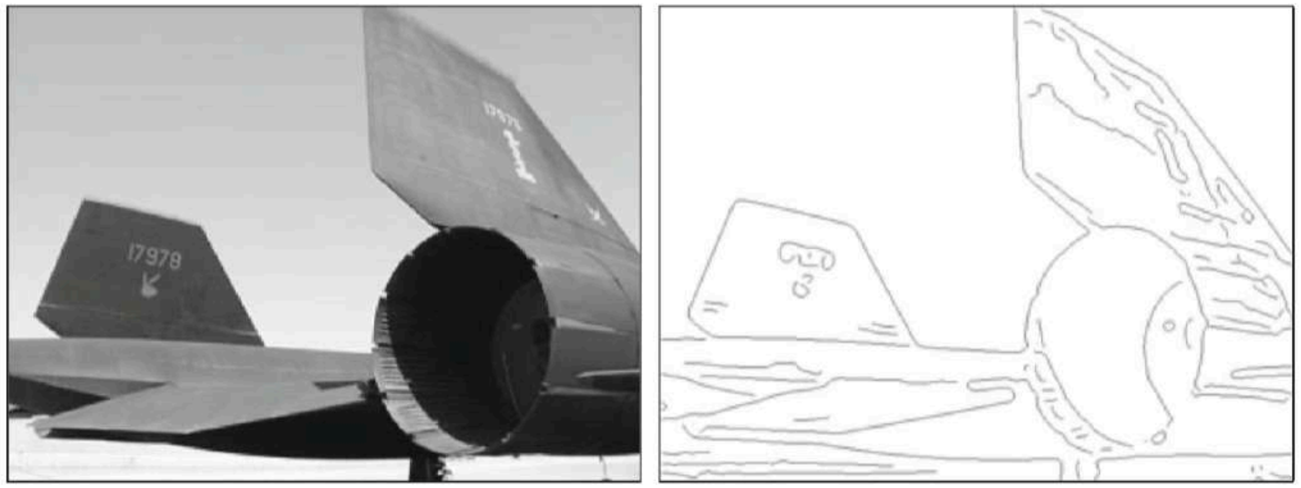




Edge detection

Edges and contours

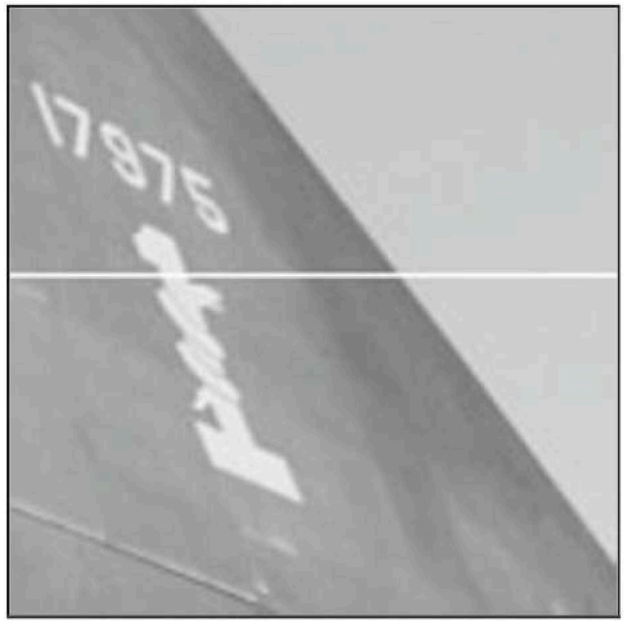
- Local changes in intensity or colour, such as edges and contours, are important for visual perception
- In human visual system, a few lines in caricature and illustration are often sufficient to describe an object and a scene
- This shows how much edges and contours can inform with our visual system

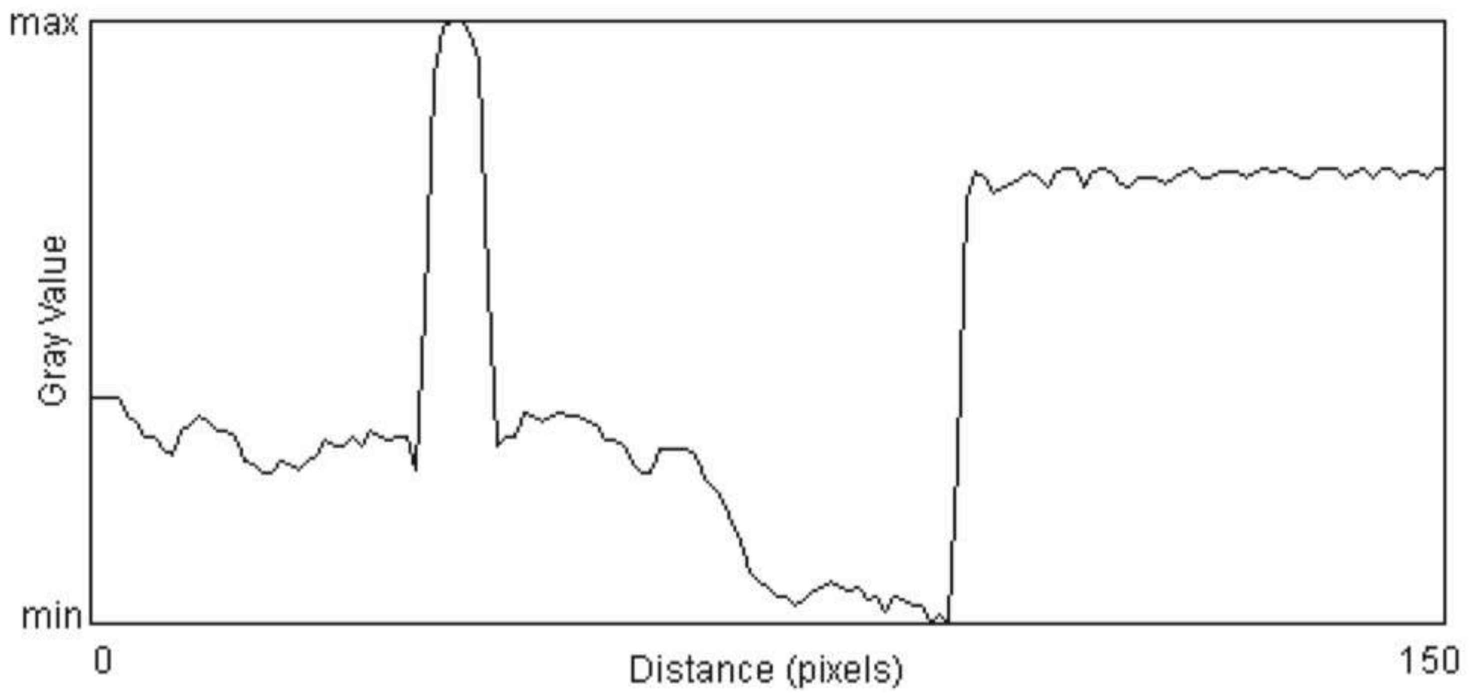


Source: Digital Image Processing by Burger and Burge, 2016

Edges and contours

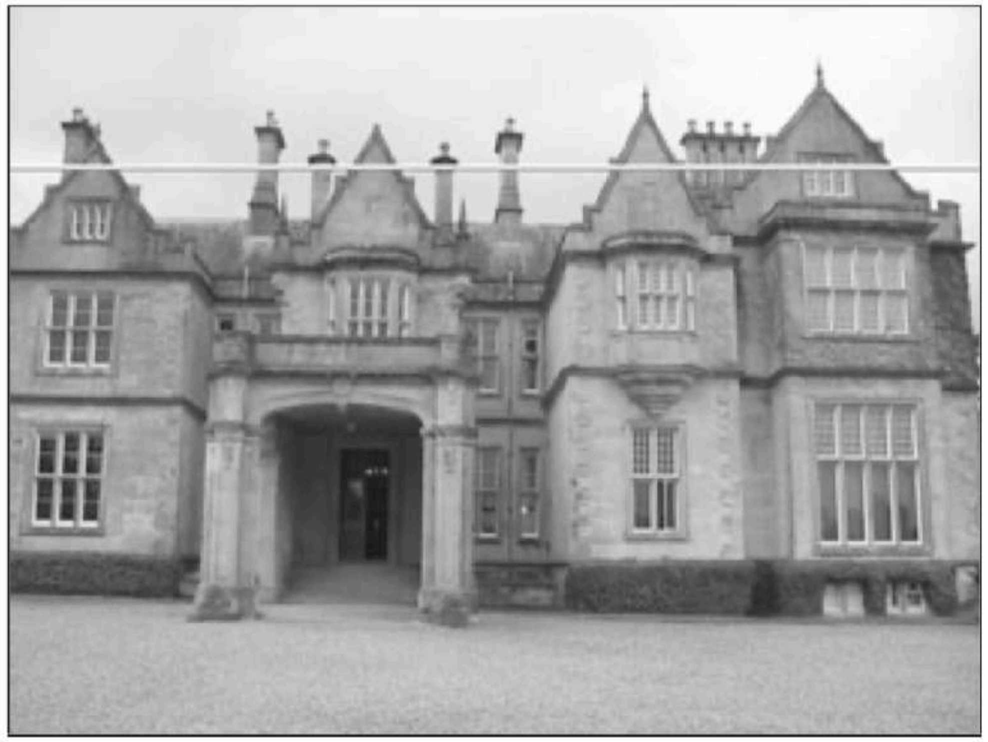
- Edges are usually located at areas where the change in intensity values is large

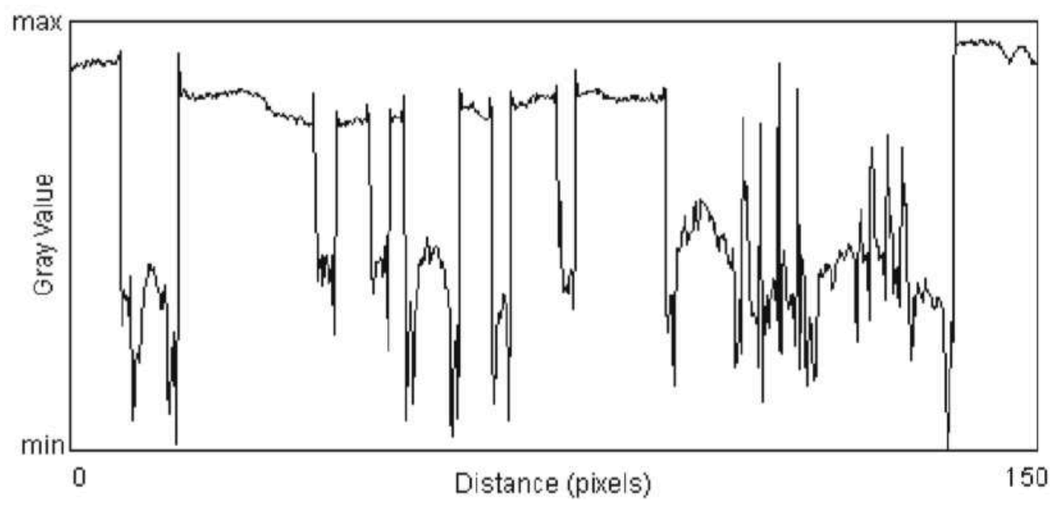




Source: Digital Image Processing by Burger and Burge, 2016

Edges and contours





Source: Digital Image Processing by Burger and Burge, 2016

- Edges are usually located at sites where the change in intensity values is large, i.e. the gradient at the site is large

- Reality: Not all the large changes in intensity values are the edges we are interested in

- Key to good edge detection: get only the edges we want, and eliminate the rest

Edge detection

- Strategies to edge detection: Calculate the gradient in x-direction and y-direction

- The calculation of gradient generally involves 2D convolution with specific kernels. Assume I is the image, and the kernel is H_x and H_y

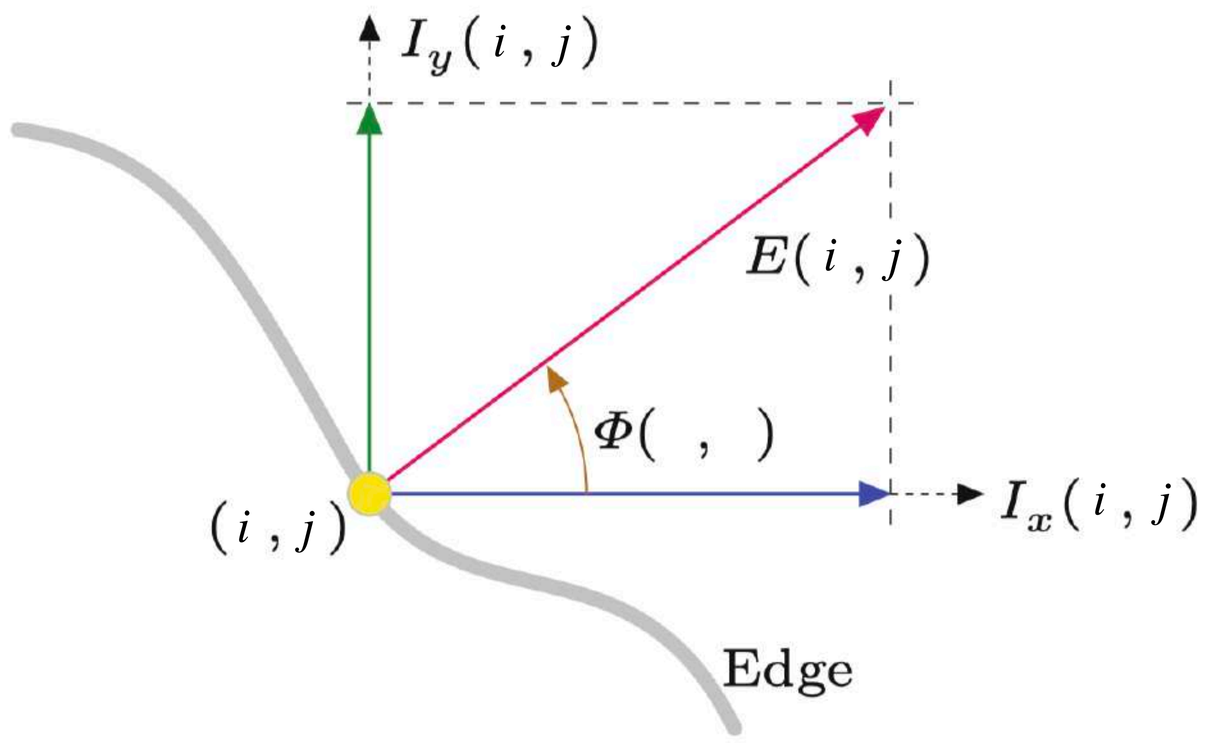
$$I_x = I * H_x \qquad I_y = I * H_y$$

- I_x and I_y is the gradient in x and y direction respectively, and the local edge strength is given by

$$E(i,j) = \sqrt{I_x^2(i,j) + I_y^2(i,j)}$$

- The local edge orientation angle is given by

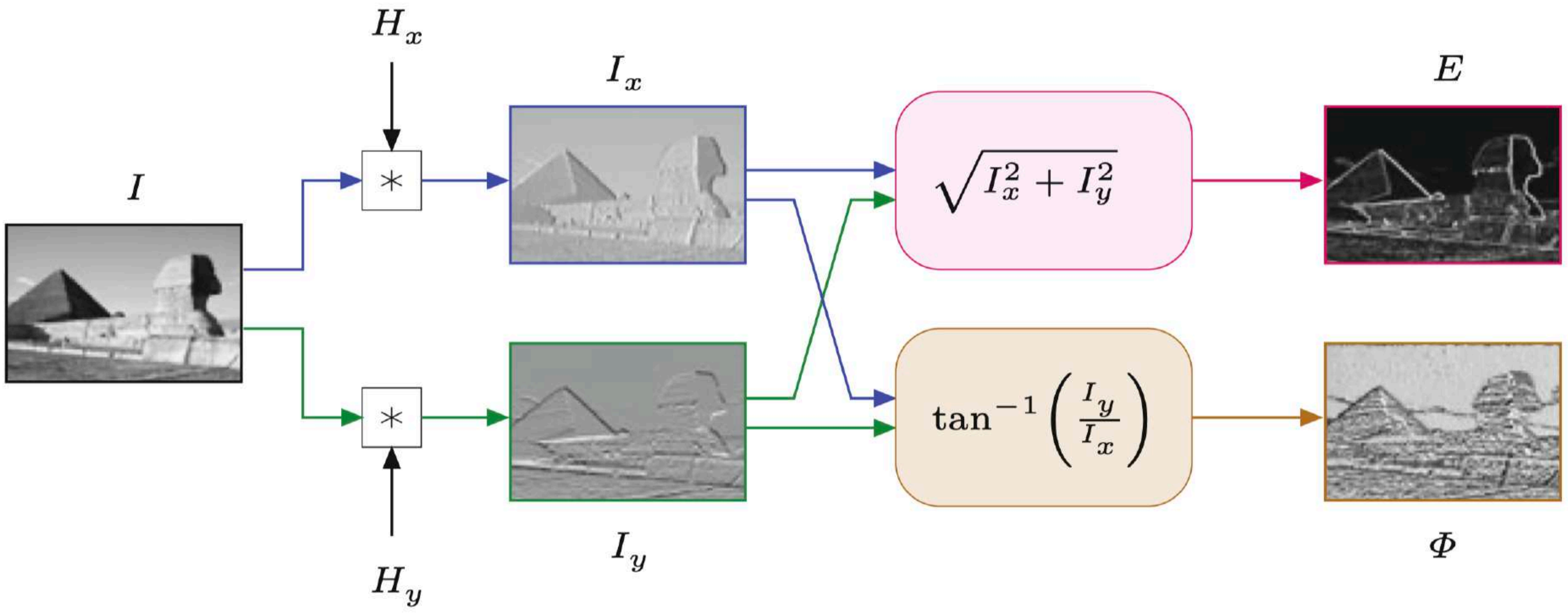
$$\Phi(i,j) = \tan^{-1}\left(\frac{I_y(i,j)}{I_x(i,j)}\right)$$



Source: Digital Image Processing by Burger and Burge, 2016

Edge detection

- The flow of getting the edges are as below



Source: Digital Image Processing by Burger and Burge, 2016

Edge detection

- Some of the common kernels / operators to get edges are Prewitt, Roberts and Sobel operators

- Roberts operators are one of the simplest and oldest methods; Prewitt and Sobel are almost similar

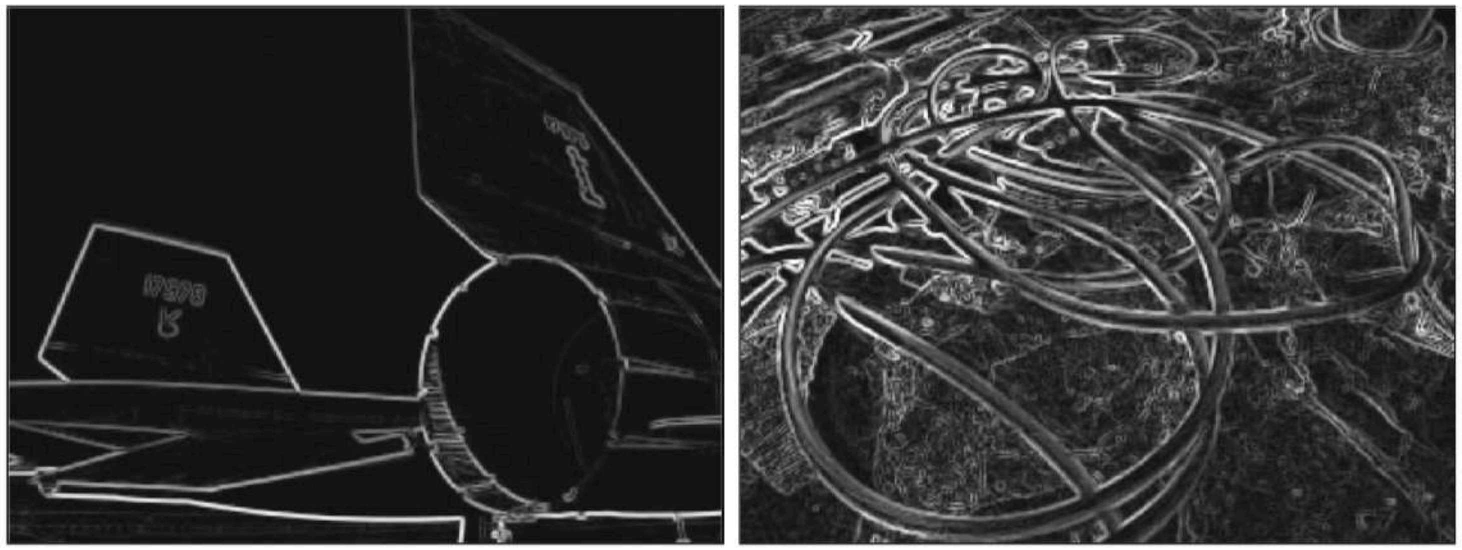
- The kernels for Sobel operator:

$$H_x^{\mathrm{S}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^{\mathrm{S}} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

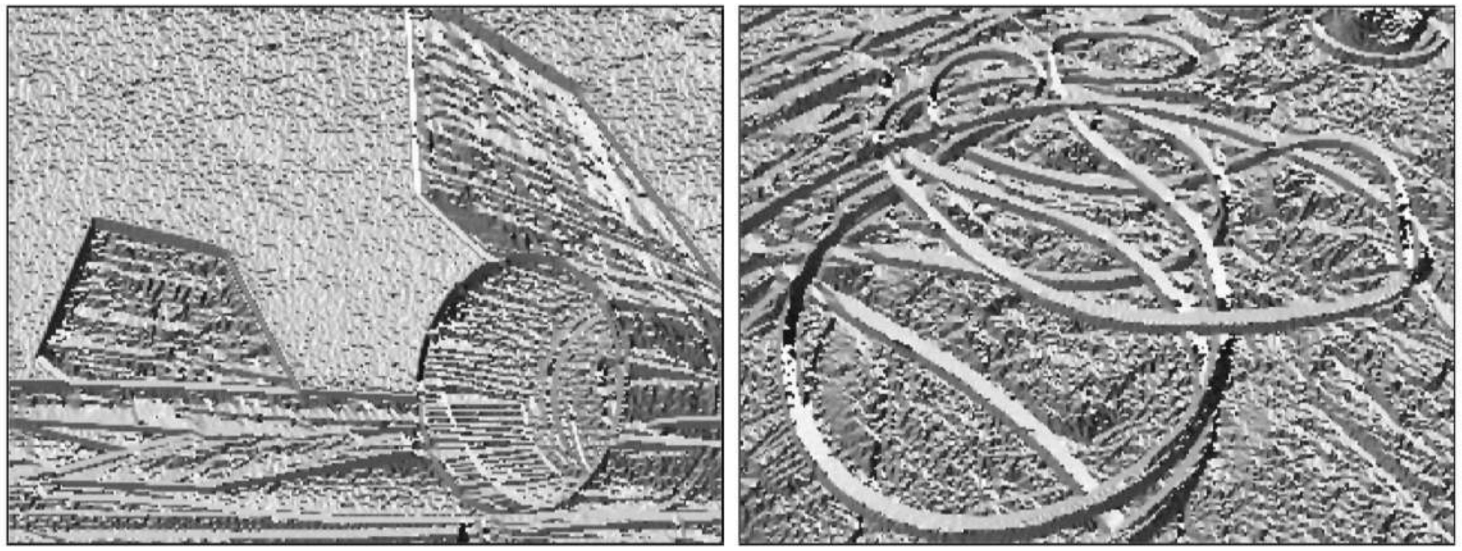
Sobel edge operators

original image

local edge strength

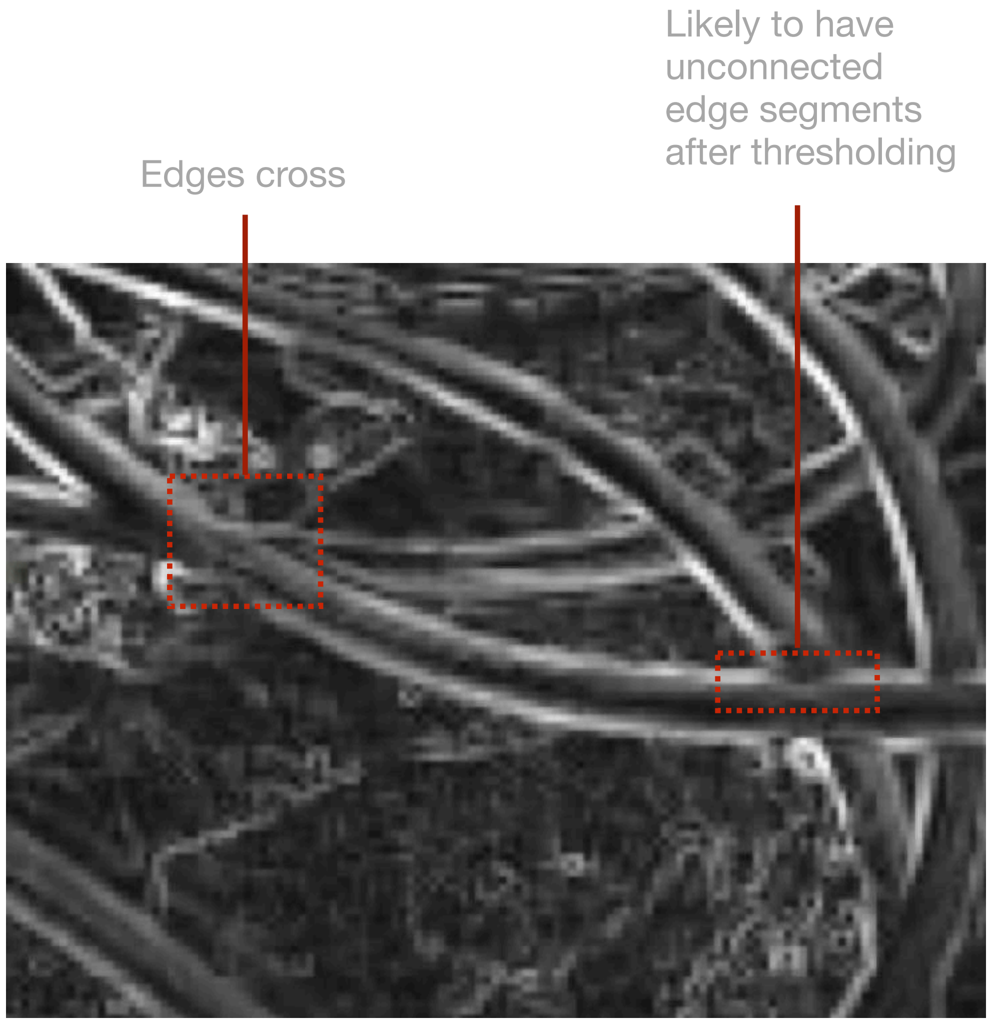


local edge orientation
(The edge orientation estimated by Sobel is relatively inaccurate)



Source: Digital Image Processing by Burger and Burge, 2016

After gradient, what's next?



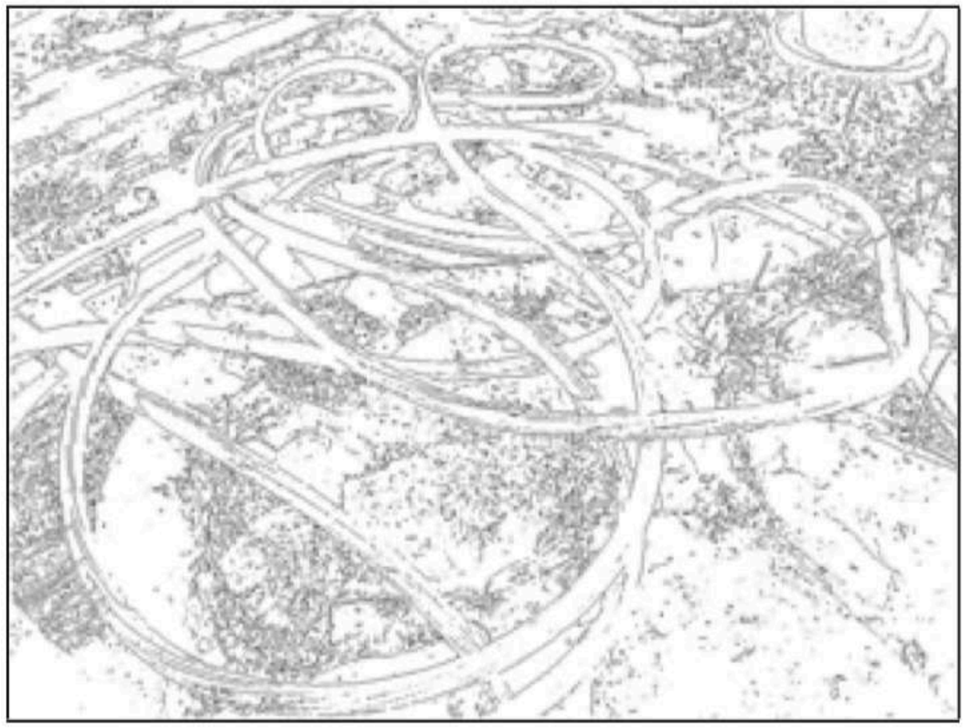
Source: Digital Image Processing by Burger and Burge, 2016

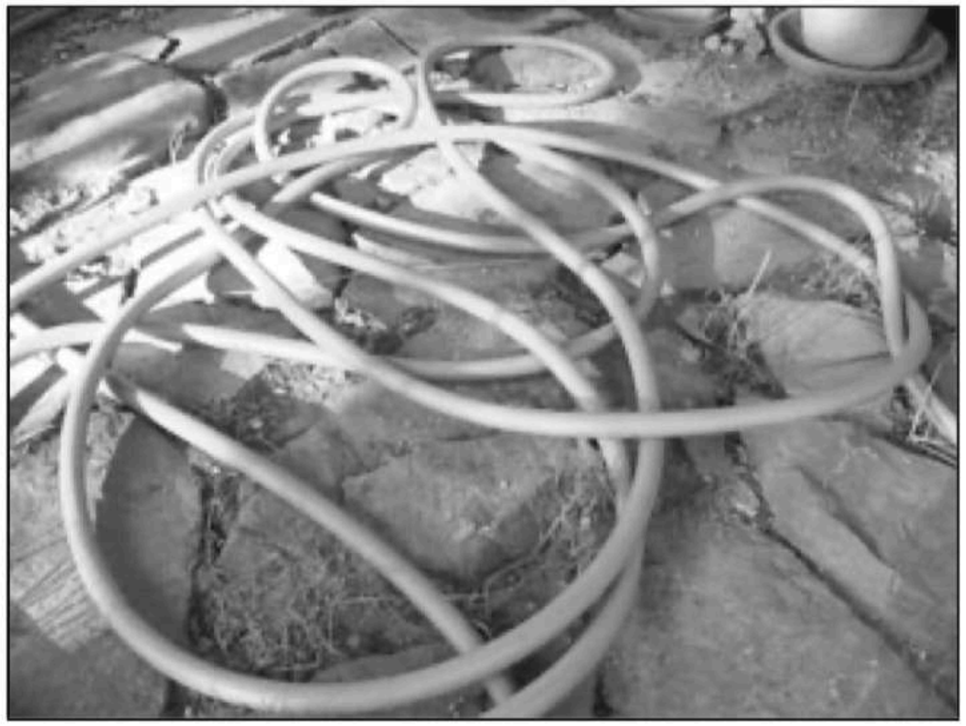
- Edge operator generally generates values that indicate the likelihood of a pixel to be part of edge
- The problem: How to judge if a pixel is part of edge?
- Use of thresholding? This strategy often produces many unconnected edge segments
- Tracing edges pixel by pixel? This strategy does not work well in scenario where edges cross, or a single edge branches into several direction
- Solution? Canny edge detection can help, but still can't fully solve the problems

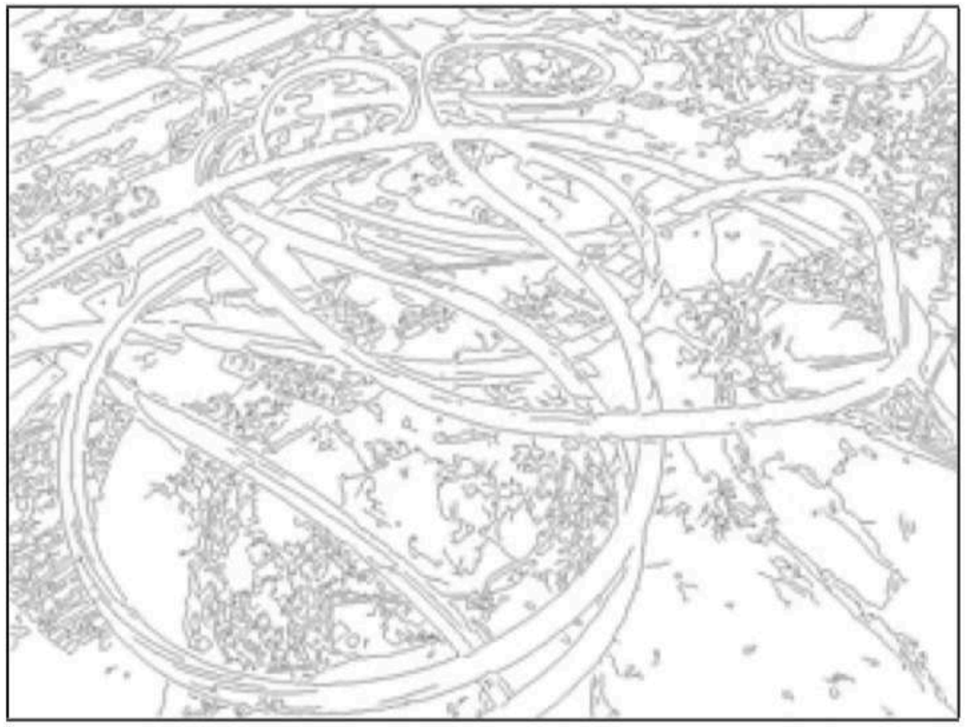
Canny edge detection

- There are four steps in Canny edge detection
- 1. Noise reduction: remove noise with 5 x 5 Gaussian filter
- 2. Calculate intensity gradient: Filter image with Sobel kernel in both horizontal and vertical direction. In image processing, Sobel kernel is often used to calculate the range of change in pixels along x-direction and y-direction
- 3. Non-maximum suppression: check if an edge point is a local maximum in its neighbourhood in the direction of gradient
- 4. Hysteresis thresholding: everything above maxVal is edge; every thing below minVal is not edge; For those in between, see connections

Canny edge detection


Sobel operator


Source: Digital Image Processing by Burger and Burge, 2016

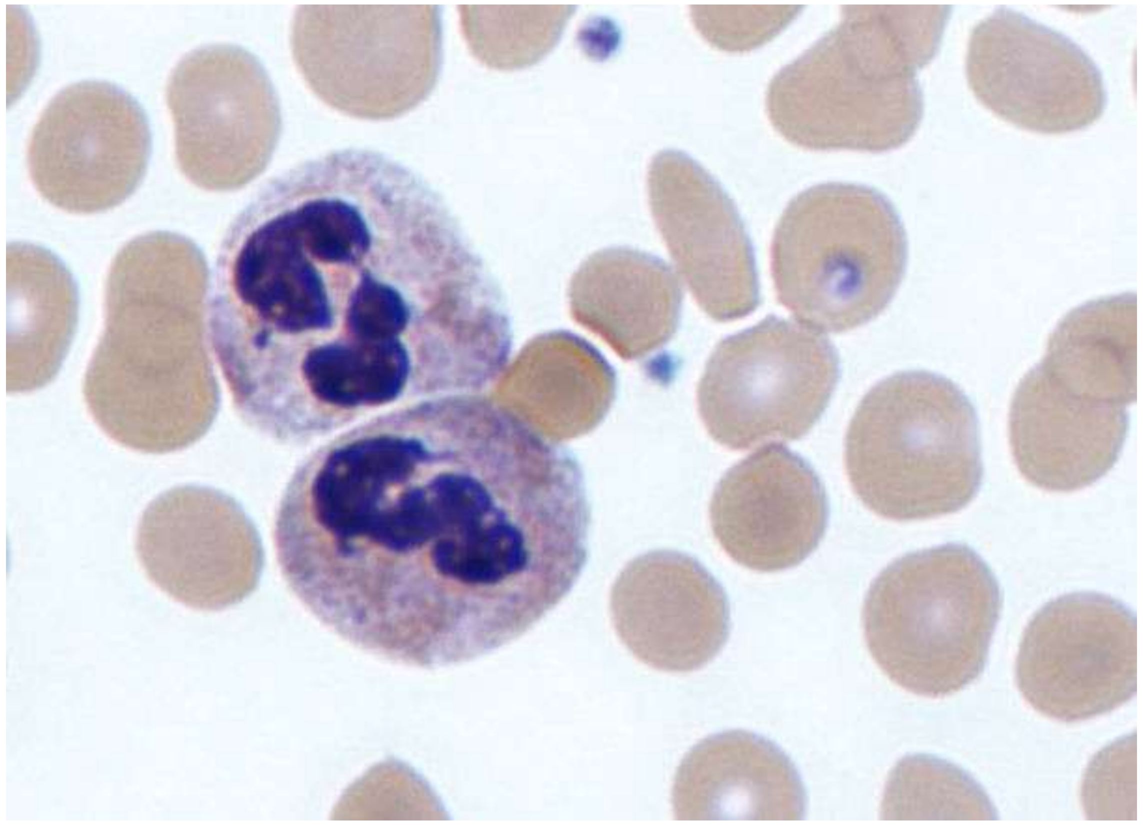

Canny edge detection

Canny edge detection

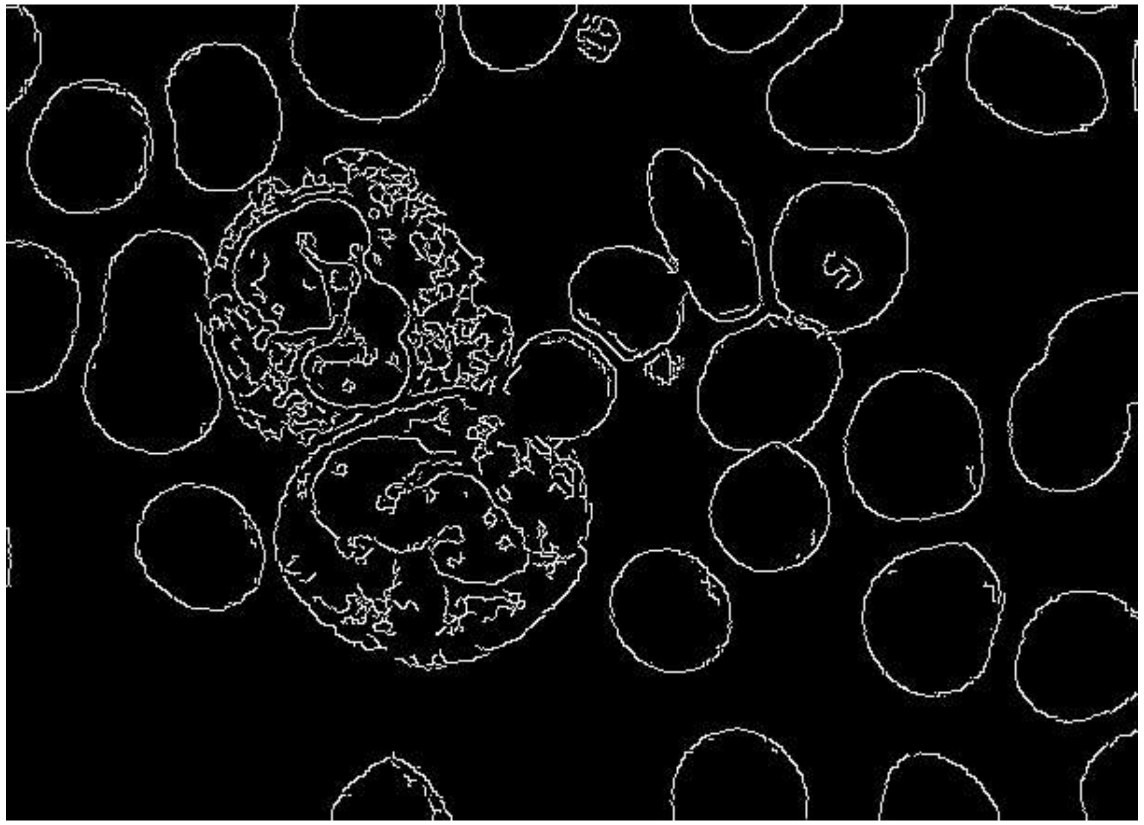
- In opencv, canny edge detection is simply done by

```
> cann= cv2.Canny(nuc,
minVal               31,
maxVal               127,
Sobel kernel size    apertureSize=3)
```

nuc



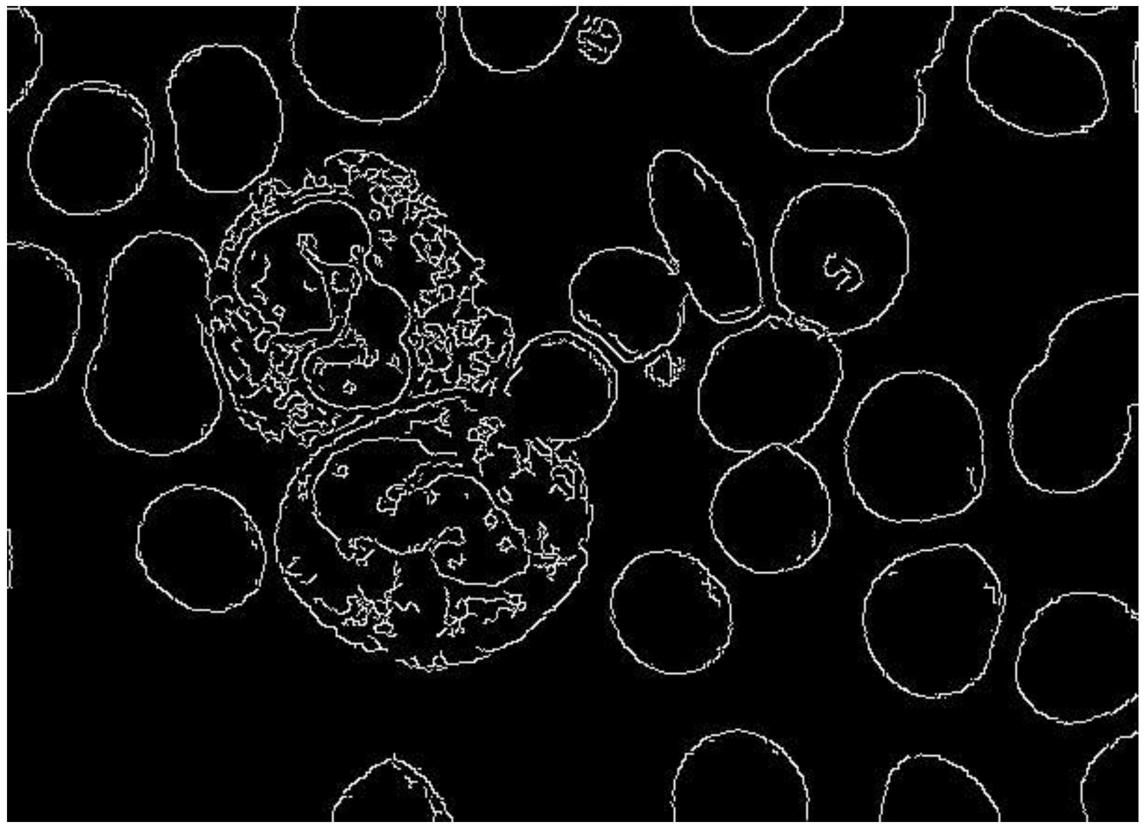
cann



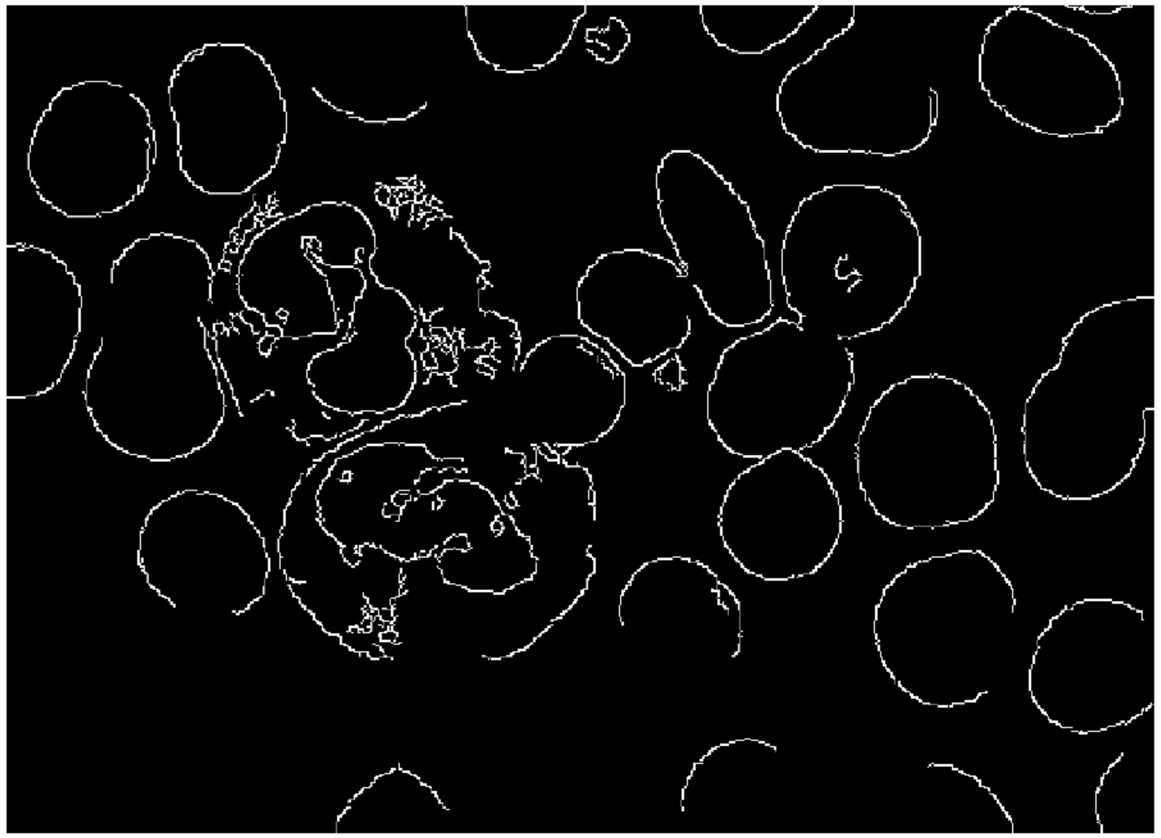
Canny edge detection

- The effect of different maxVal

maxVal = 127

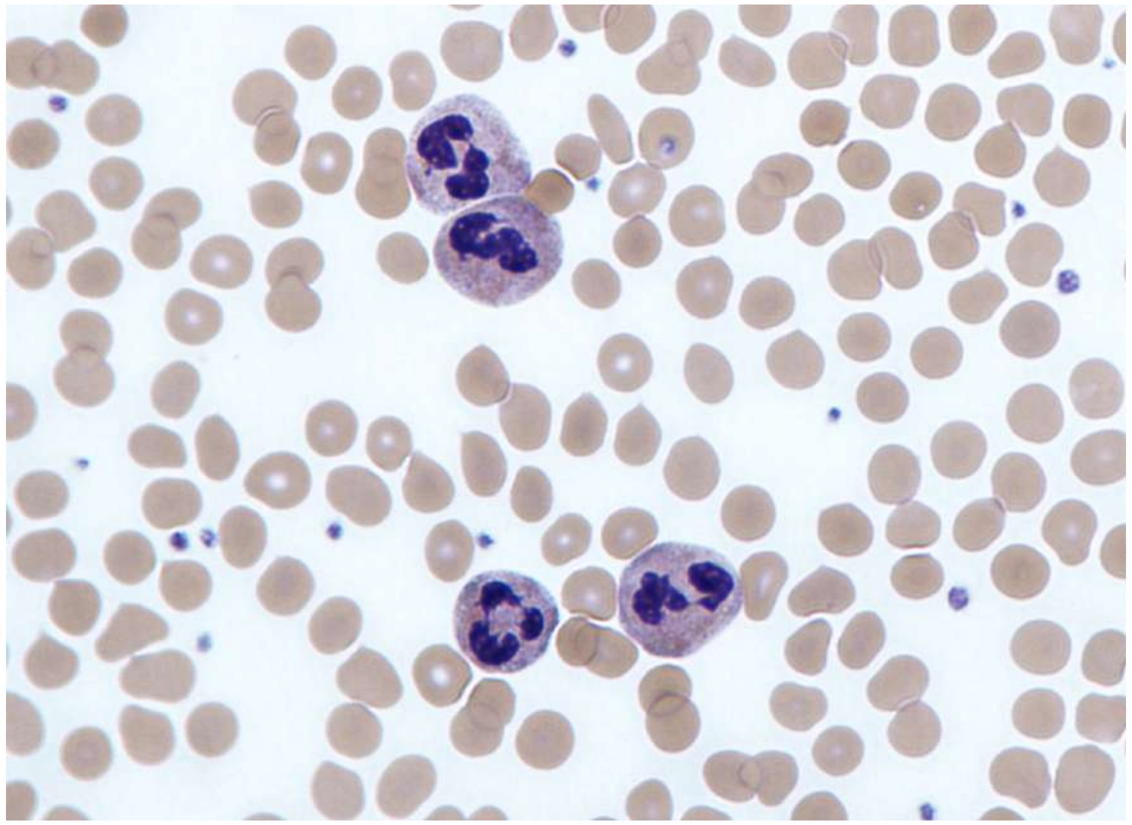


maxVal = 191

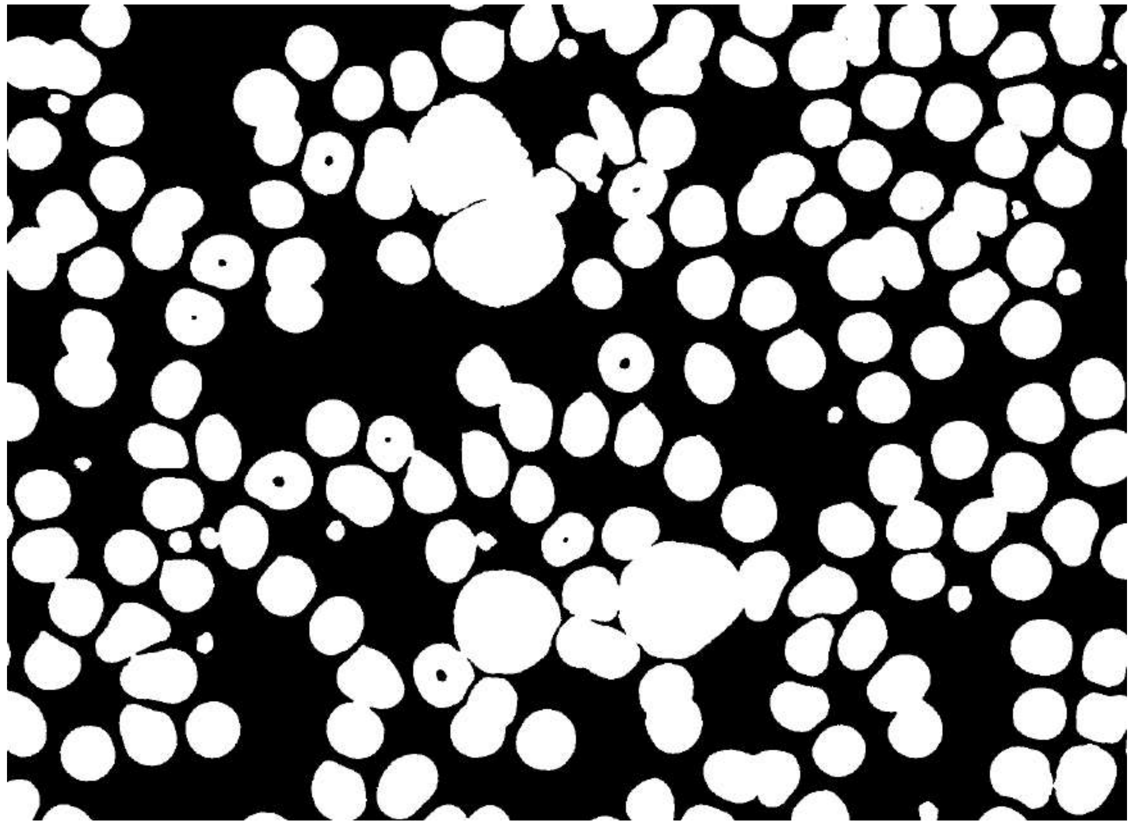


Problem: How to isolate region of interest?

neu.jpg



wks2_2_e.jpg



Region isolation

- Load the images

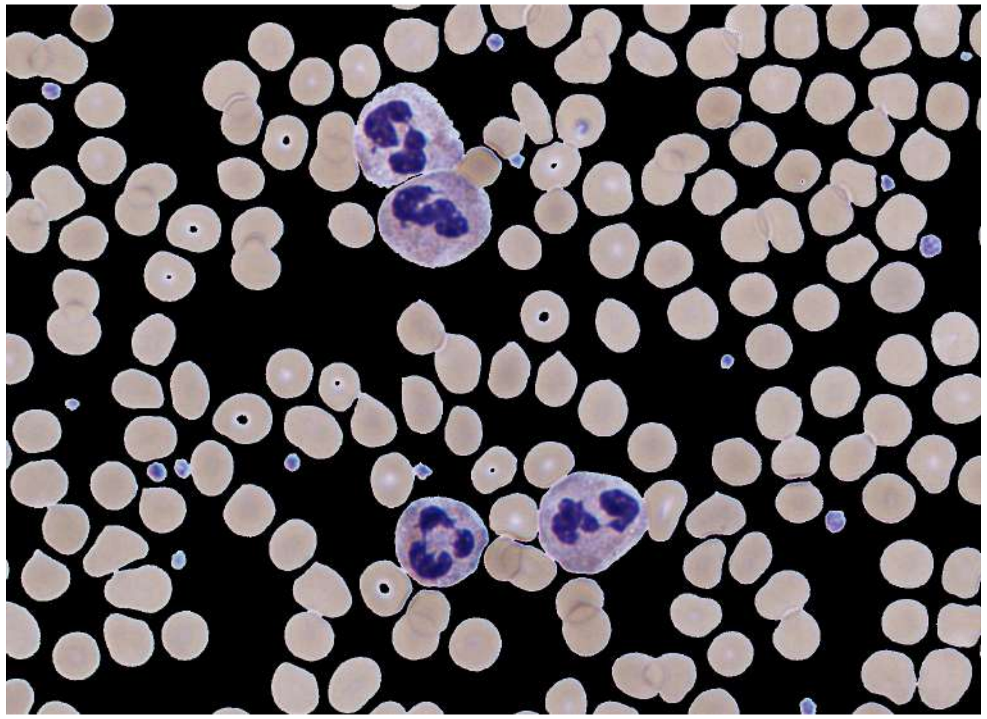
```
> neu = cv2.imread('neu.jpg')
> msk = cv2.imread('wks2_2_e.jpg')
```

- Process the mask

```
> msk = np.float32(msk[:,:,0])/255
> msk = np.round(msk)
```

- Do the isolation

```
> iso = cv2.bitwise_and(neu,     source 1
               source 2 neu,
                   mask mask=np.uint8(msk))
```



Draw boundaries

- To draw boundaries, first go and find boundary contour

```
> ctrs= cv2.findContours(np.uint8(msk),     mask
     contour retrieval modes  cv2.RETR_EXTERNAL,
contour approximation methods  cv2.CHAIN_APPROX_SIMPLE)
```

- `ctrs` is a tuple with 2 items. First item is a list of contour; second item is a hierarchy matrix

```
> ctrs= ctrs[0]
```

- Draw the contours

```
                                          > cv2.drawContours(neu,   image to be drawn
                                                             ctrs,  list of contours
contour index; -1 indicate all contours to be drawn          -1,
                             colour of the contours          (191,191,255),
                                          thickness          5)
```

Draw contours