Assignment 2
Filters
Parts 1-3

CMPUT206

Prof. Nilanjan Ray

TA: Bernal Manzanilla

Intro: Create your own 2D convolution filtering script in Python

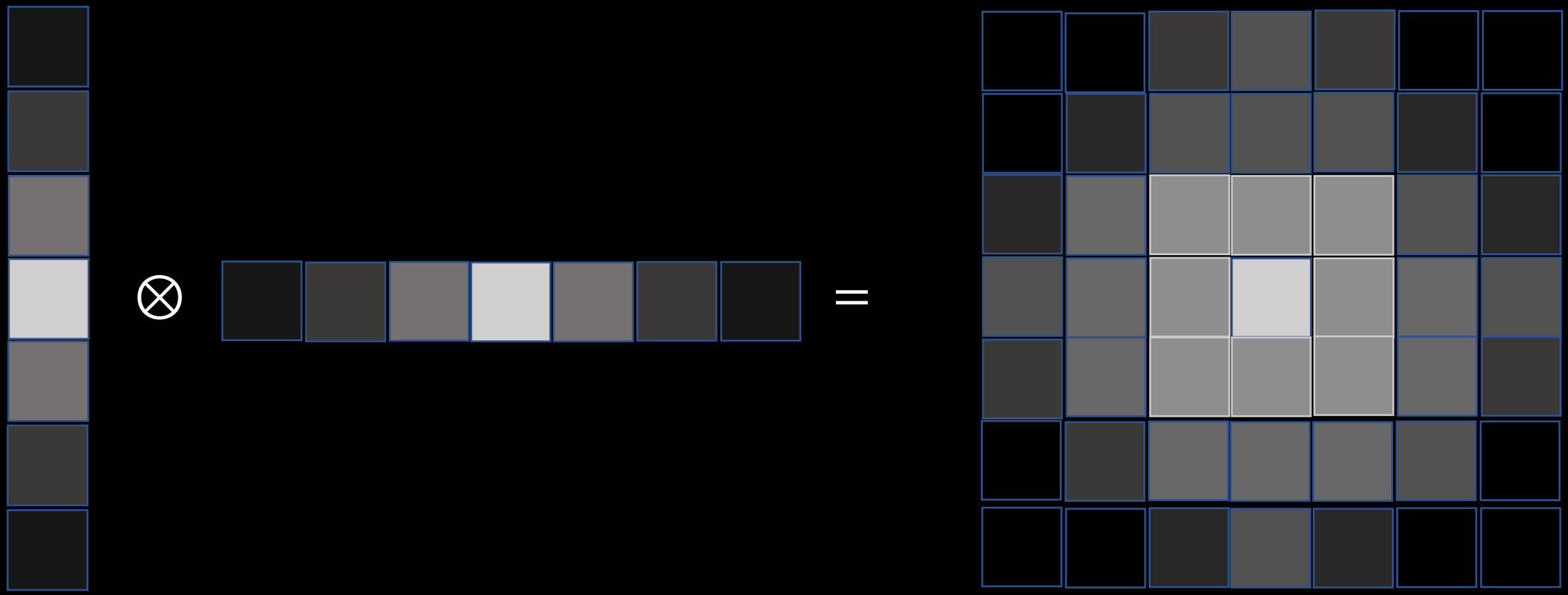
- Read a grayscale image moon.png
- Filter the grayscale image with the following filters
- Display results
- Create a 3-by-3 matrix and write your own code to implement these filters: (You cannot use any built in functions from any library for this.)

Intro: Create your own 2D convolution filtering script in Python

- Pad image with strips of width equal to the filter size.
- Flip kernel horizontally and vertically.
- “Overlap” kernel on the top left corner of the image.
- Mutiply Kernel with image pixel to pixel
- Sum
- Slide to the right.
- Once you hit the rightmost side of the image, do a line break (i.e. slide Kernel down and place it again on the left).

Intro: Filter kernel.

$$k^T \quad * \quad k \quad = \quad \mathrm{K}$$



Intro: Pad. DIY or calling a command....

```
img_padded = np.pad(img, pad_width=1, mode=?)
```

Calling a padding function...

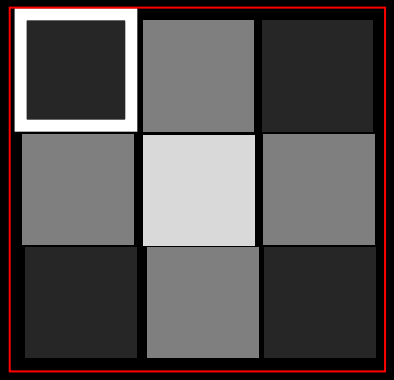
```
img_padded =np.zeros(ImIn.shape[0]+Filtersize[0], ImIn.shape[1]+Filtersize[1])
```

Then copy the pixels you need into the padding area....

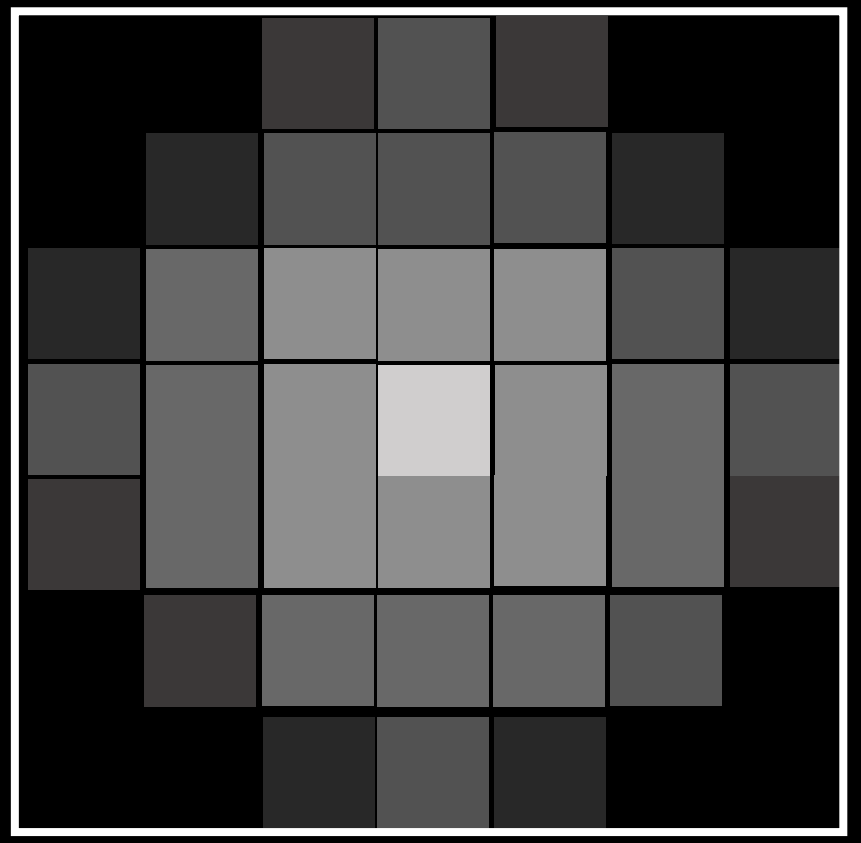
https://numpy.org/doc/stable/reference/generated/numpy.zeros.html

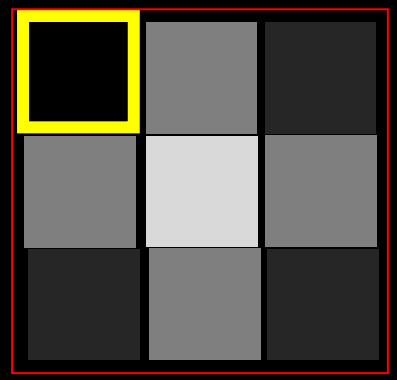
Intro: 2D convolution visually

3 by 3 filter (kernel)

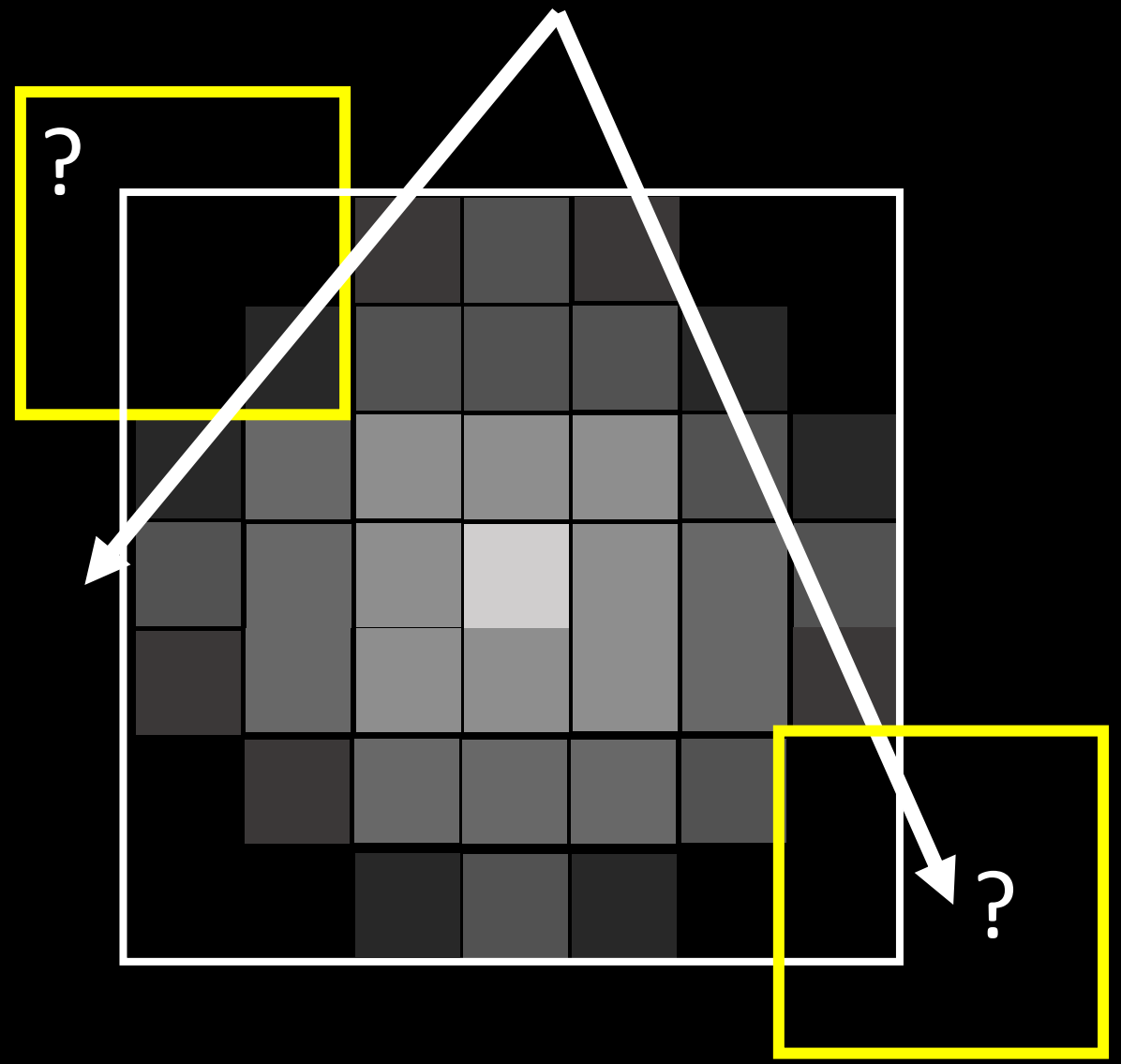


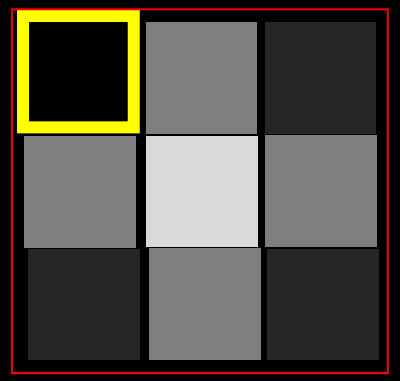
Image



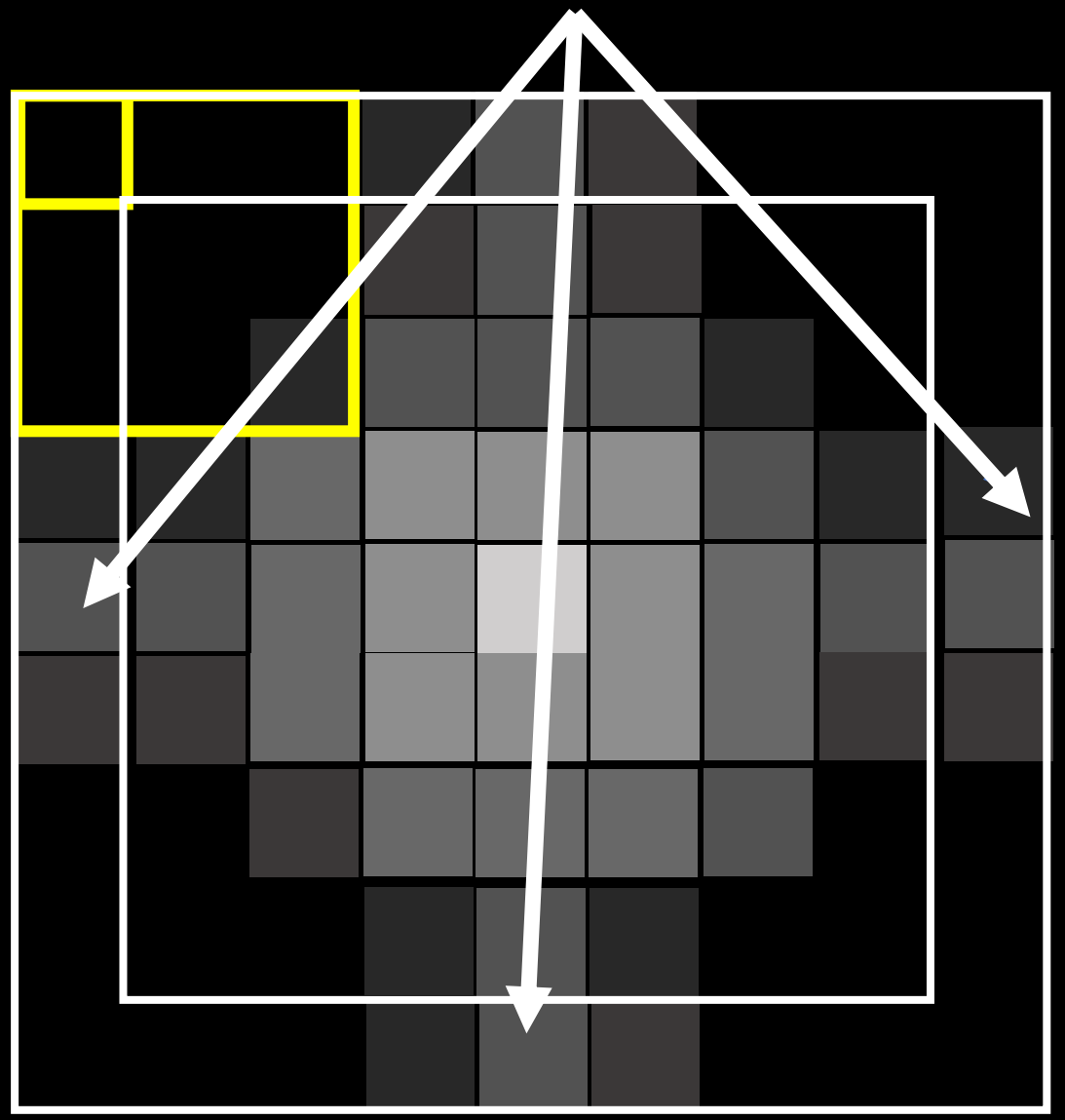


Entension (copy edges...) Padding





Entension (copy edges...) Padding



Intro: Pad


50
100
150
200
250

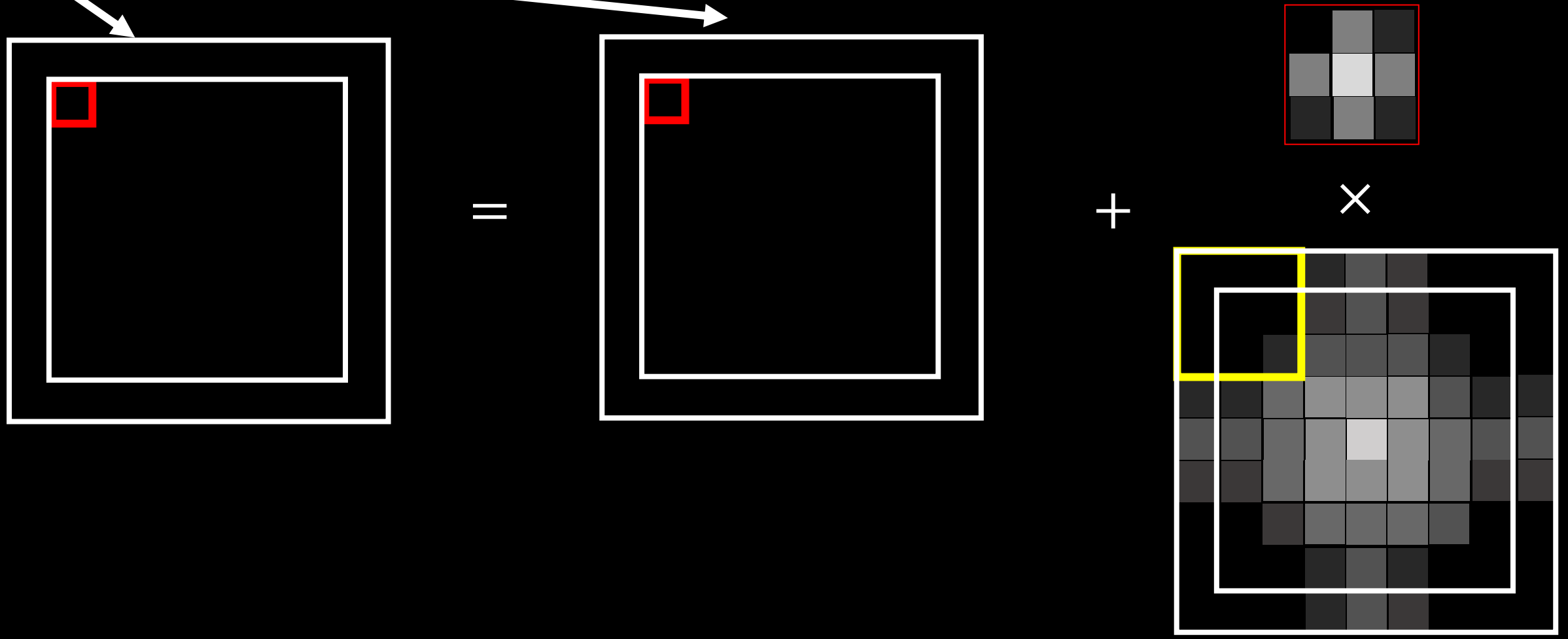

50
100
150
200
250


100
150
200
250
30


50
100
150
200

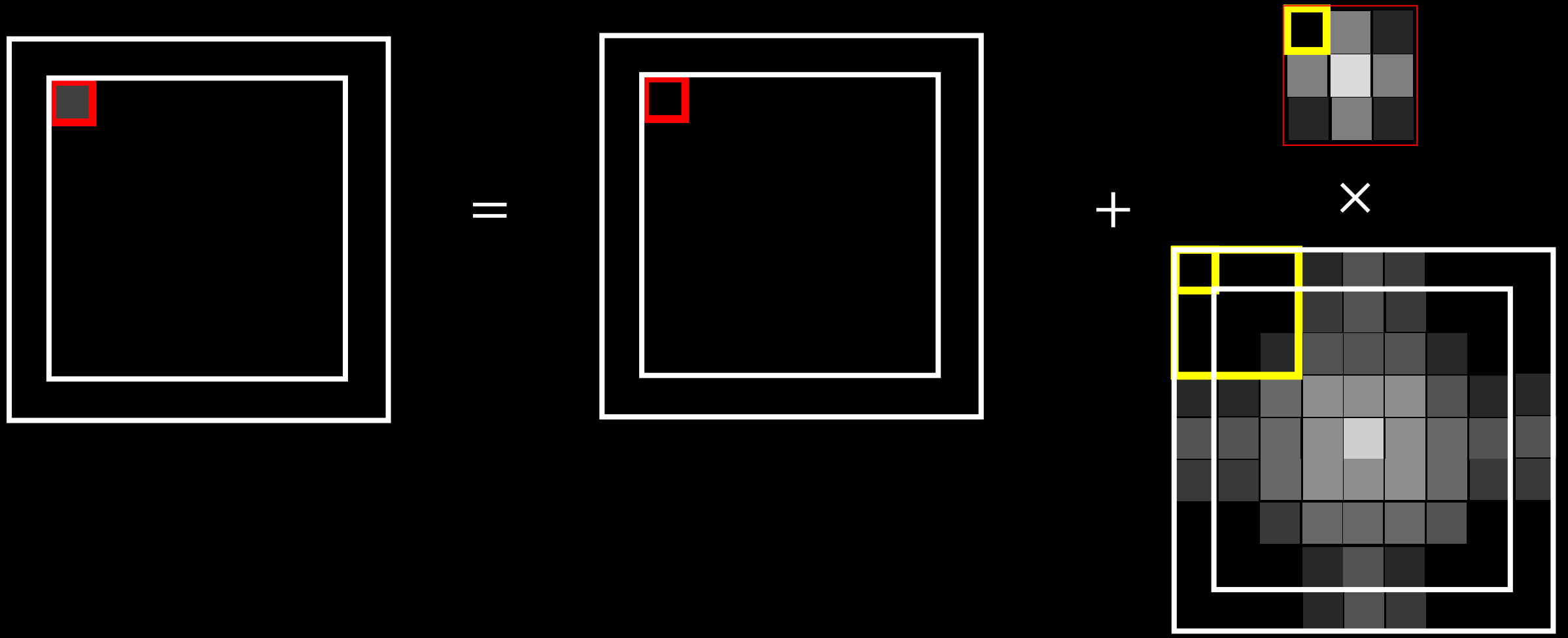
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\boldsymbol{kcol}, \boldsymbol{krow}) \times ImIn(row + kcol, col + krow)$$



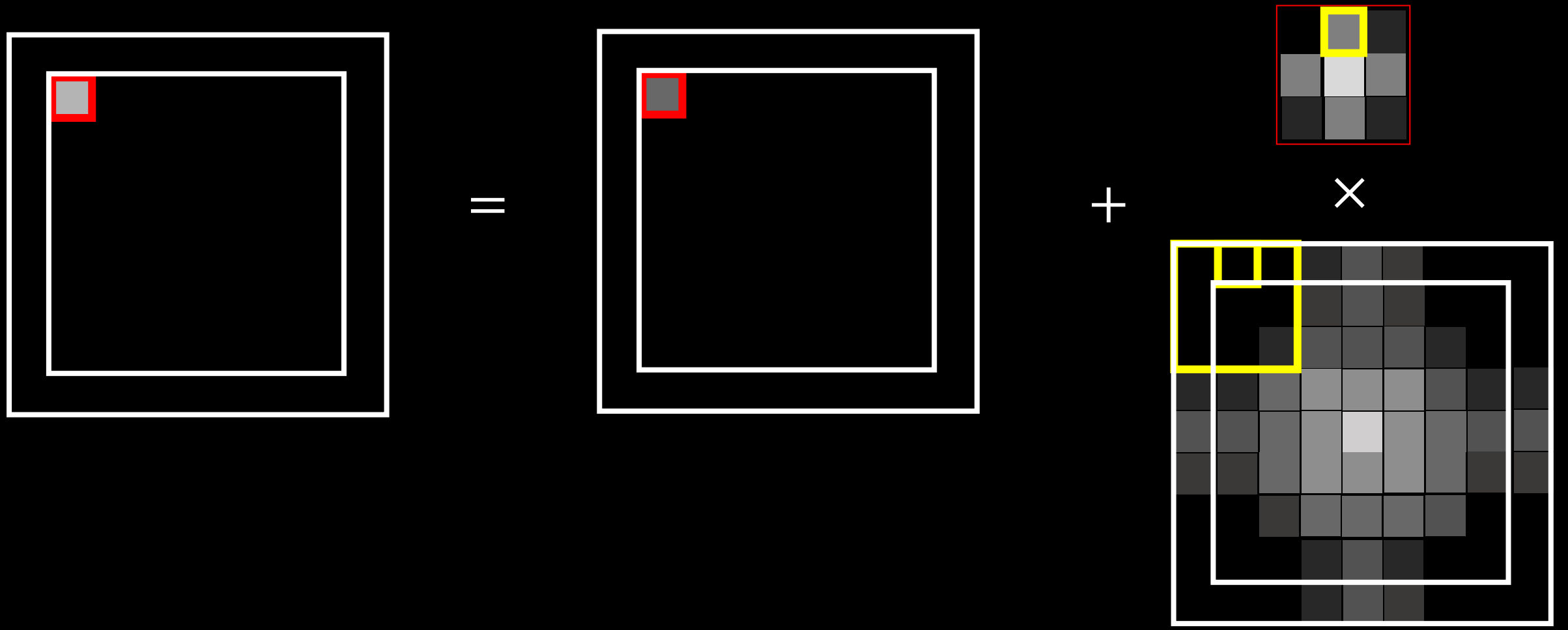
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{1}, \mathbf{1}) \times ImIn(row + \mathbf{1}, col + \mathbf{1})$$



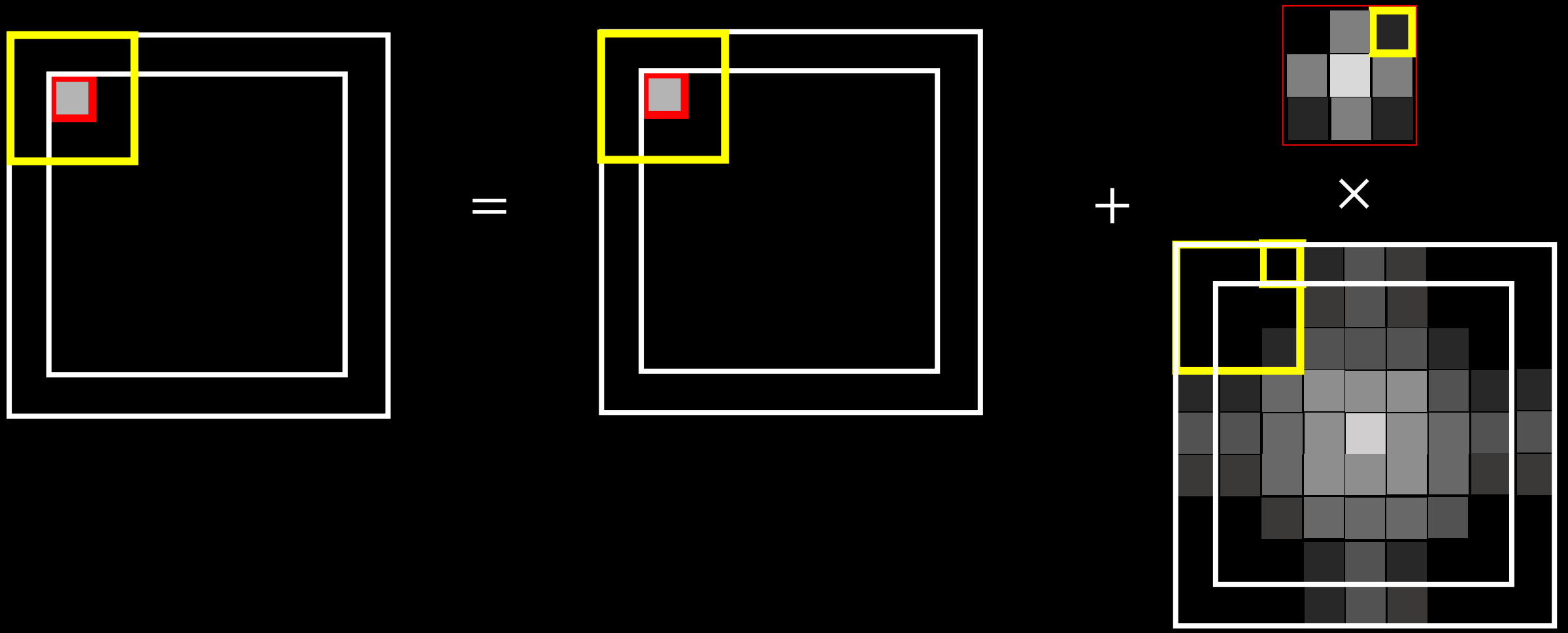
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{1}, \mathbf{2}) \times ImIn(row + 1, col + \mathbf{2})$$



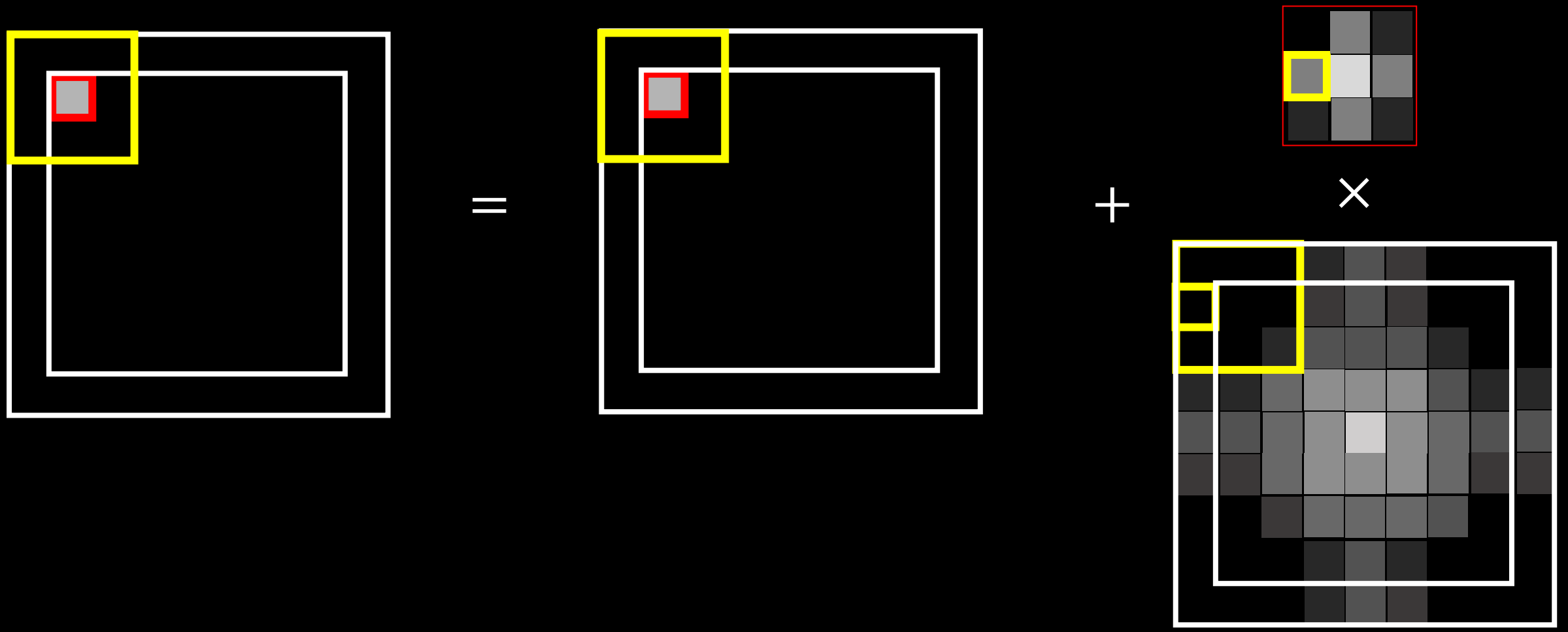
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{1}, \mathbf{3}) \times ImIn(row + \mathbf{1}, col + \mathbf{3})$$



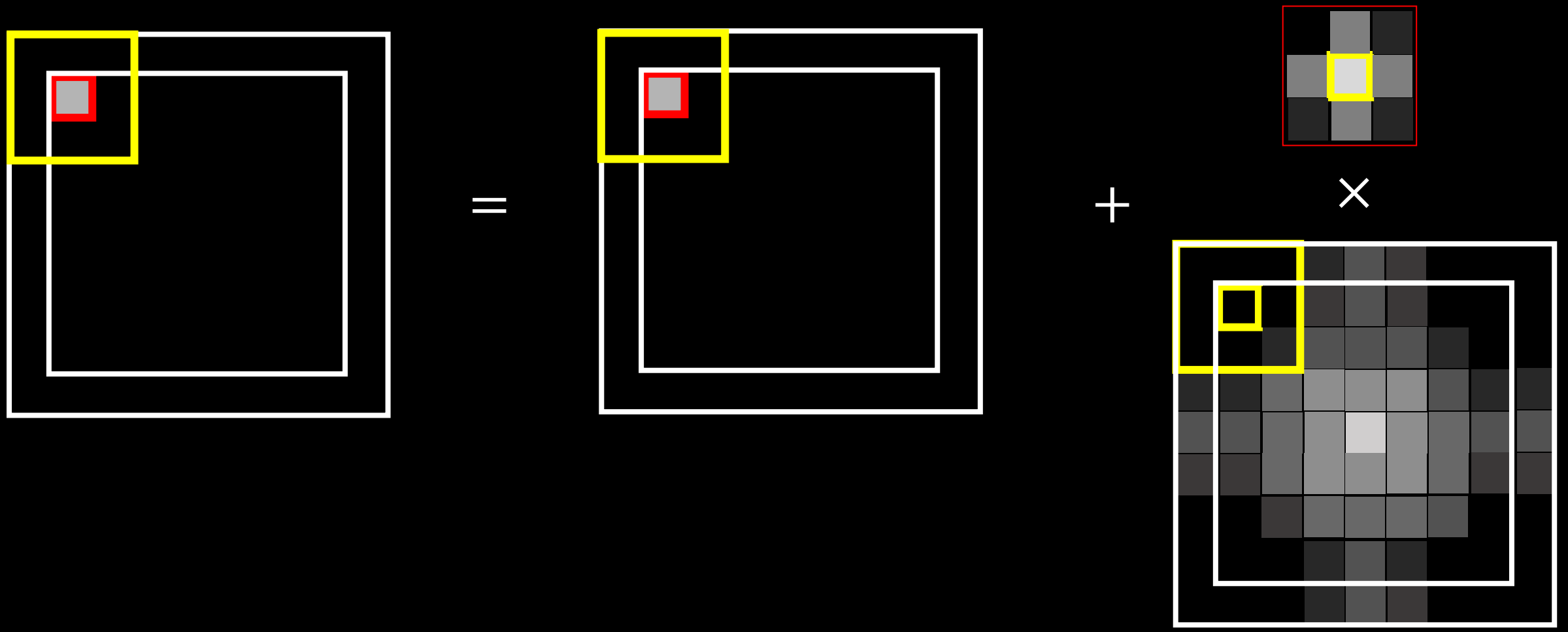
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{2}, \mathbf{1}) \times ImIn(row + \mathbf{2}, col + \mathbf{1})$$



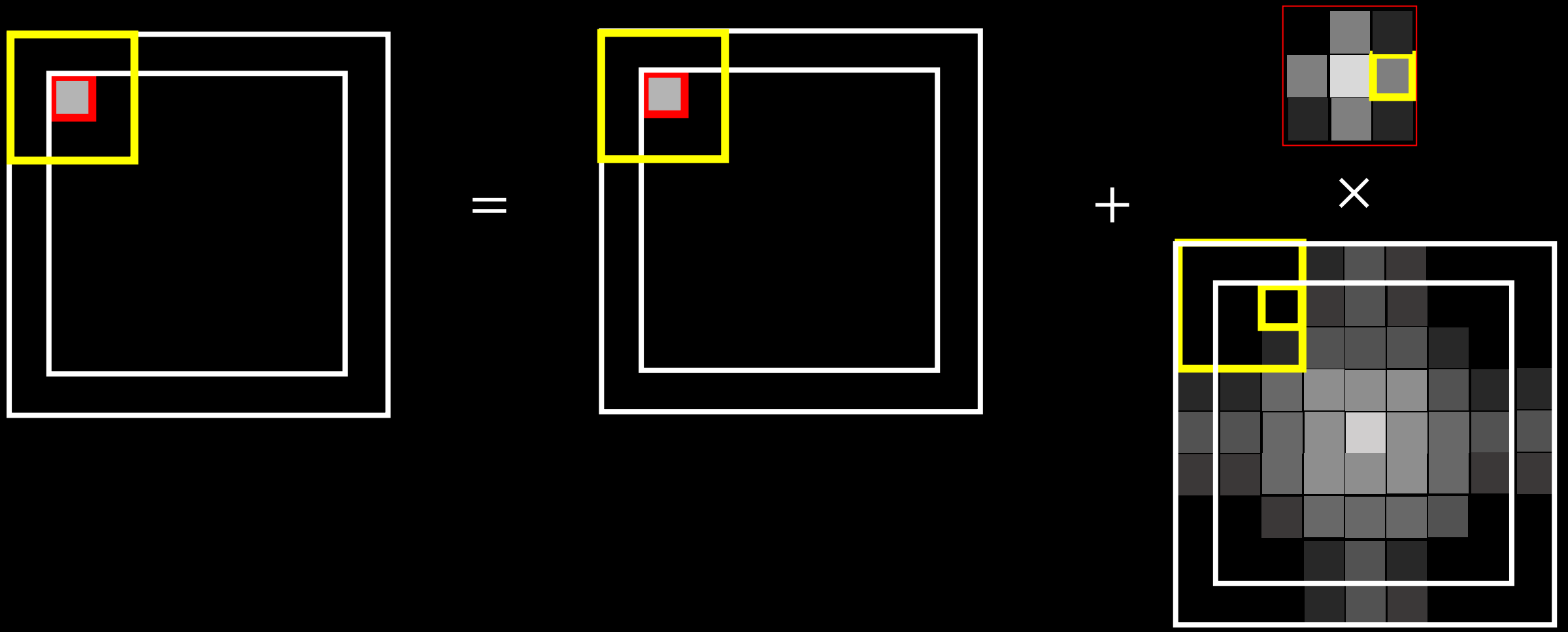
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{2}, \mathbf{2}) \times ImIn(row + \mathbf{2}, col + \mathbf{2})$$



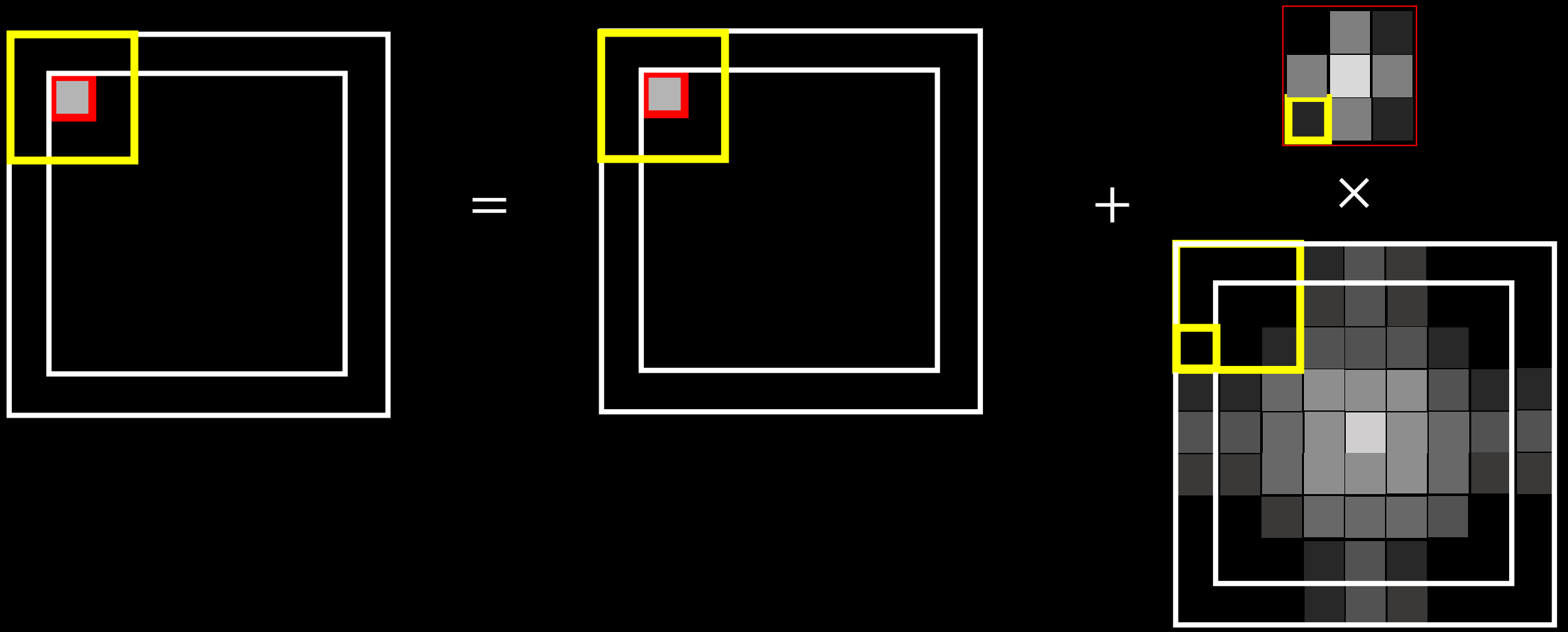
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{2}, \mathbf{3}) \times ImIn(row + \mathbf{2}, col + \mathbf{3})$$



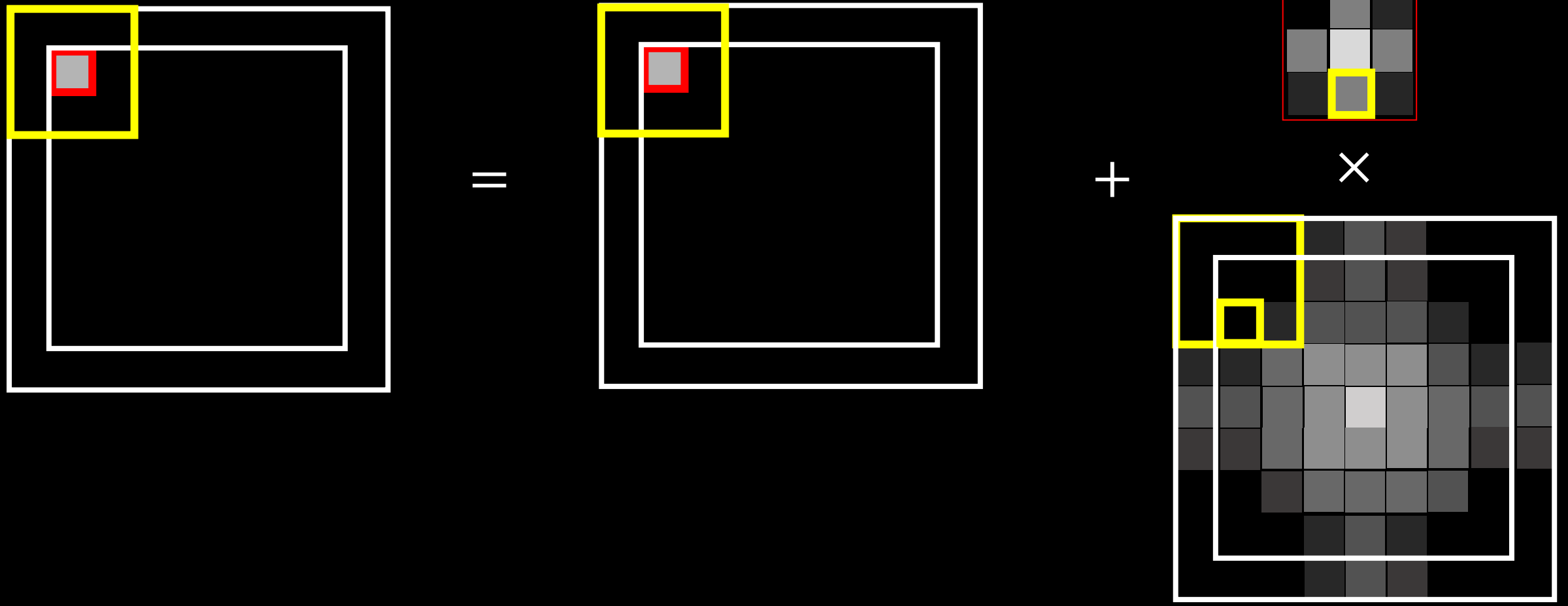
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{3}, \mathbf{1}) \times ImIn(row + \mathbf{3}, col + \mathbf{1})$$



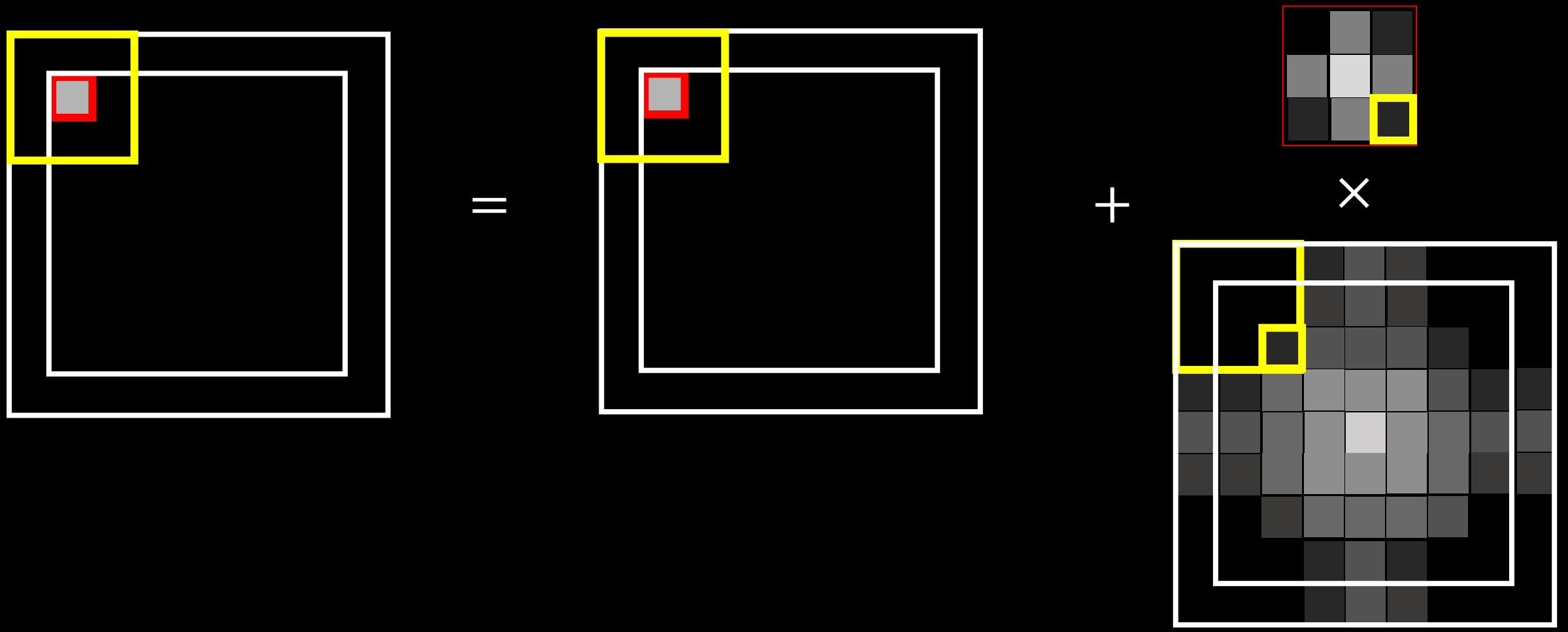
$$\text{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\text{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{3}, \mathbf{2}) \times ImIn(row + \mathbf{3}, col + \mathbf{2})$$



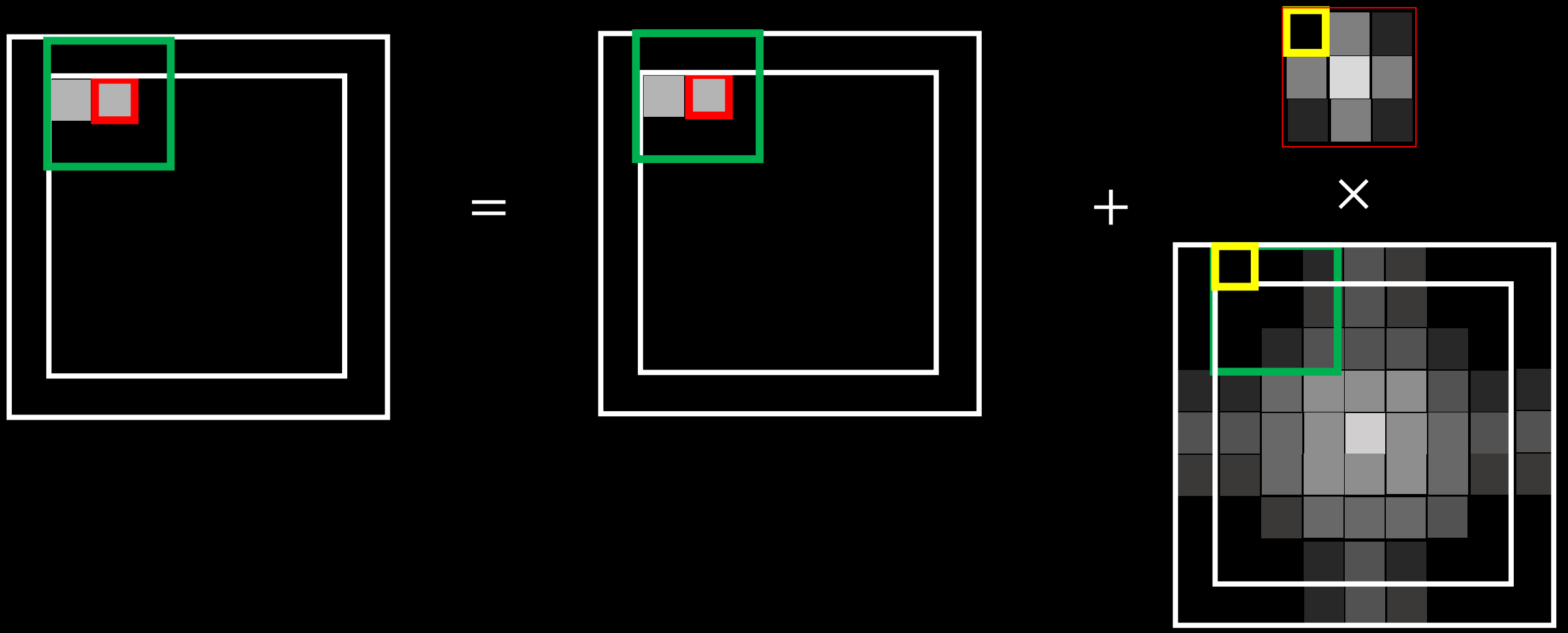
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{3}, \mathbf{3}) \times ImIn(row + \mathbf{3}, col + \mathbf{3})$$



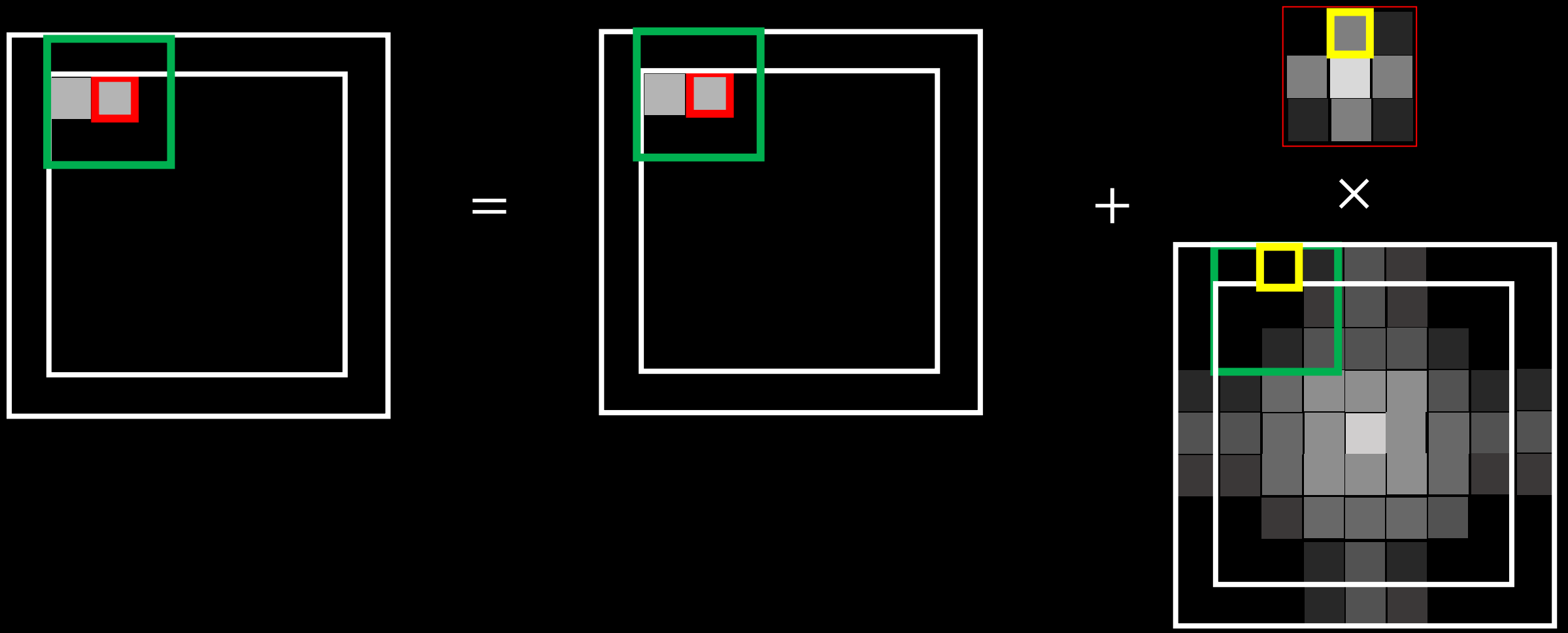
$$\text{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\text{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(2,1) \times ImIn(row + 2, col + 1)$$



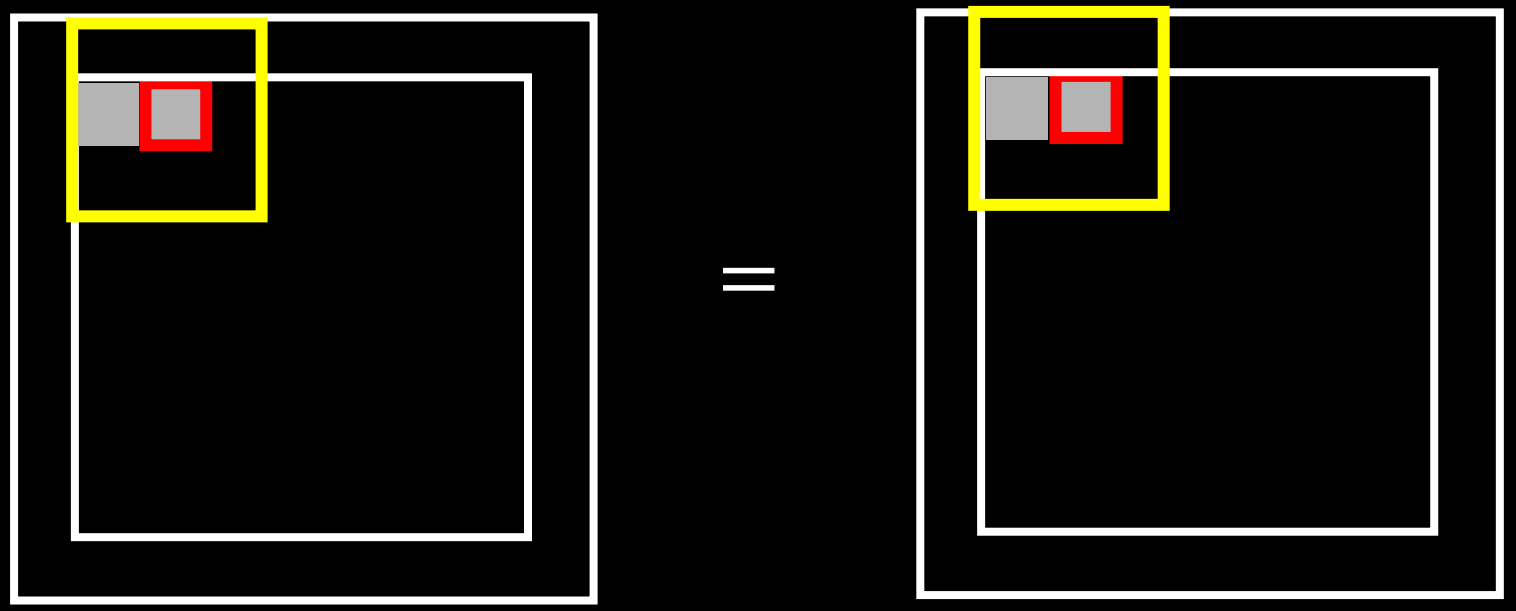
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{2}, \mathbf{1}) \times ImIn(row + \mathbf{2}, col + \mathbf{1})$$



$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

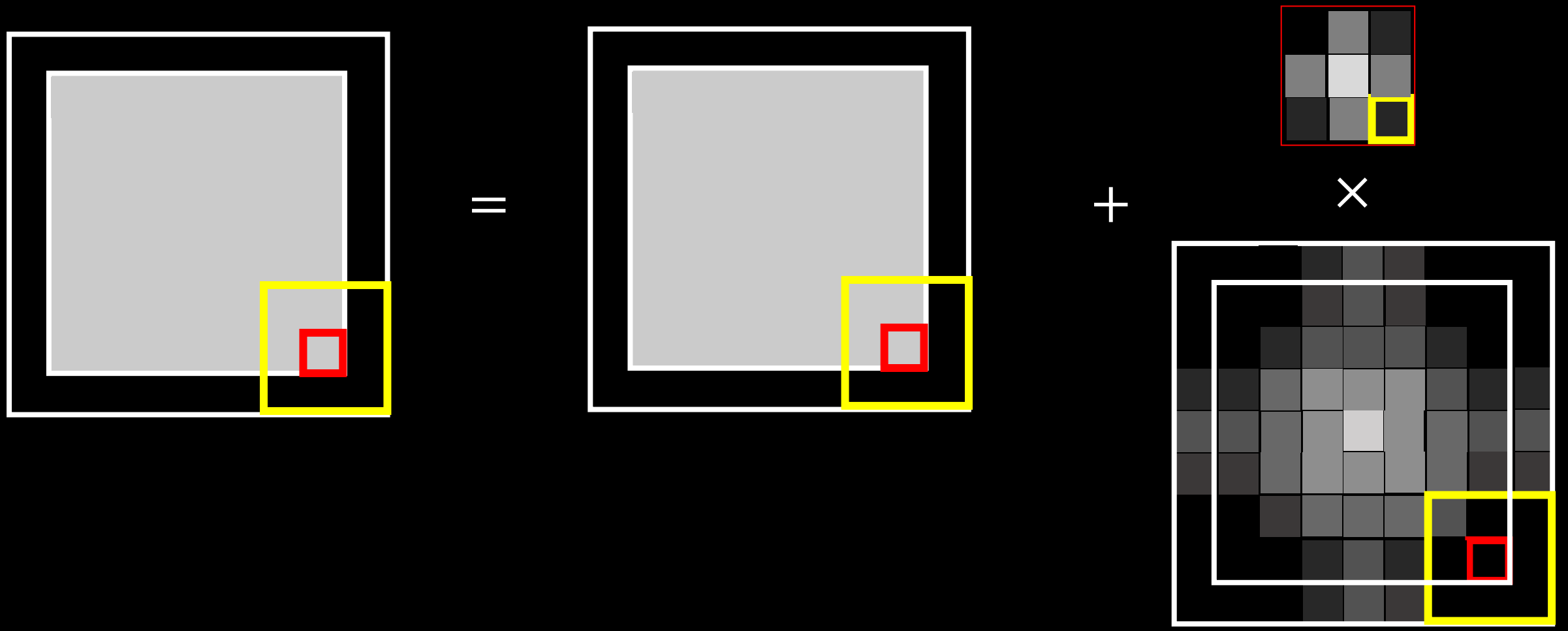
$$\mathrm{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{2}, \mathbf{1}) \times ImIn(row + \mathbf{2}, col + \mathbf{1})$$



and so on… until…

$$\text{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) =$$

$$\text{ImOut}\left(row + \frac{Kcols - 1}{2} + 1, col + \frac{Krows - 1}{2} + 1\right) + k(\mathbf{2}, \mathbf{1}) \times ImIn(row + \mathbf{2}, col + \mathbf{1})$$



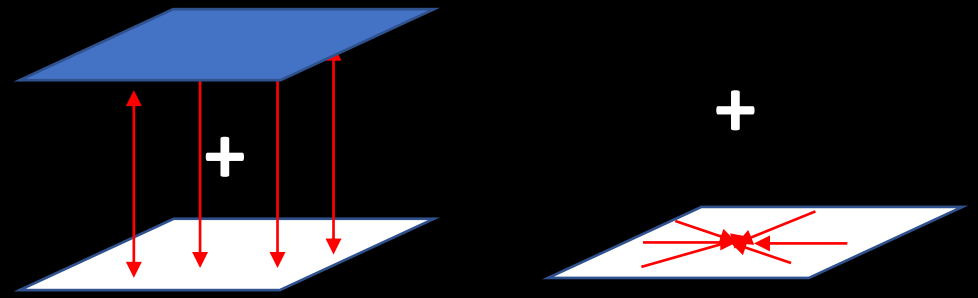
Part 1: Create your own 2D convolution filtering script in Python

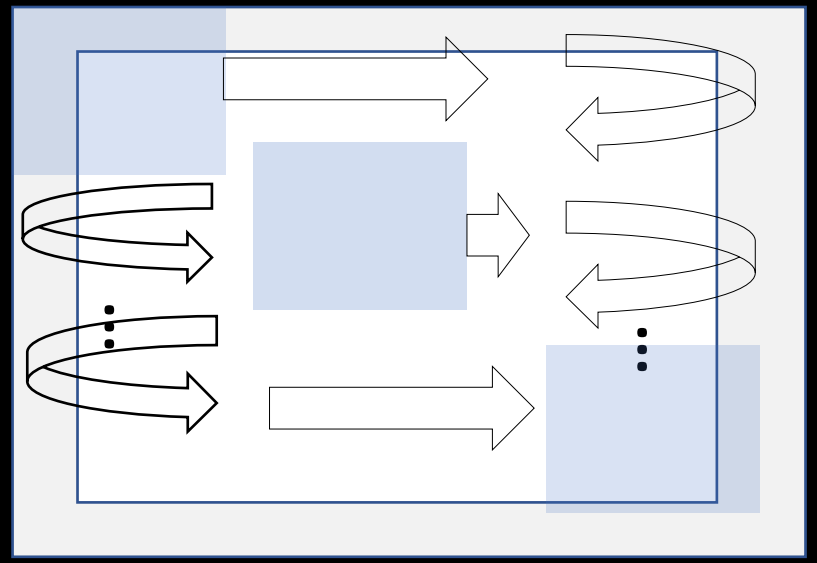
Block 1: Filter:

1. **Pad image**
2. **Perform pointwise multiplication of each element of filter and image patch**
3. **Sum and assign output center pixel position in image patch.**

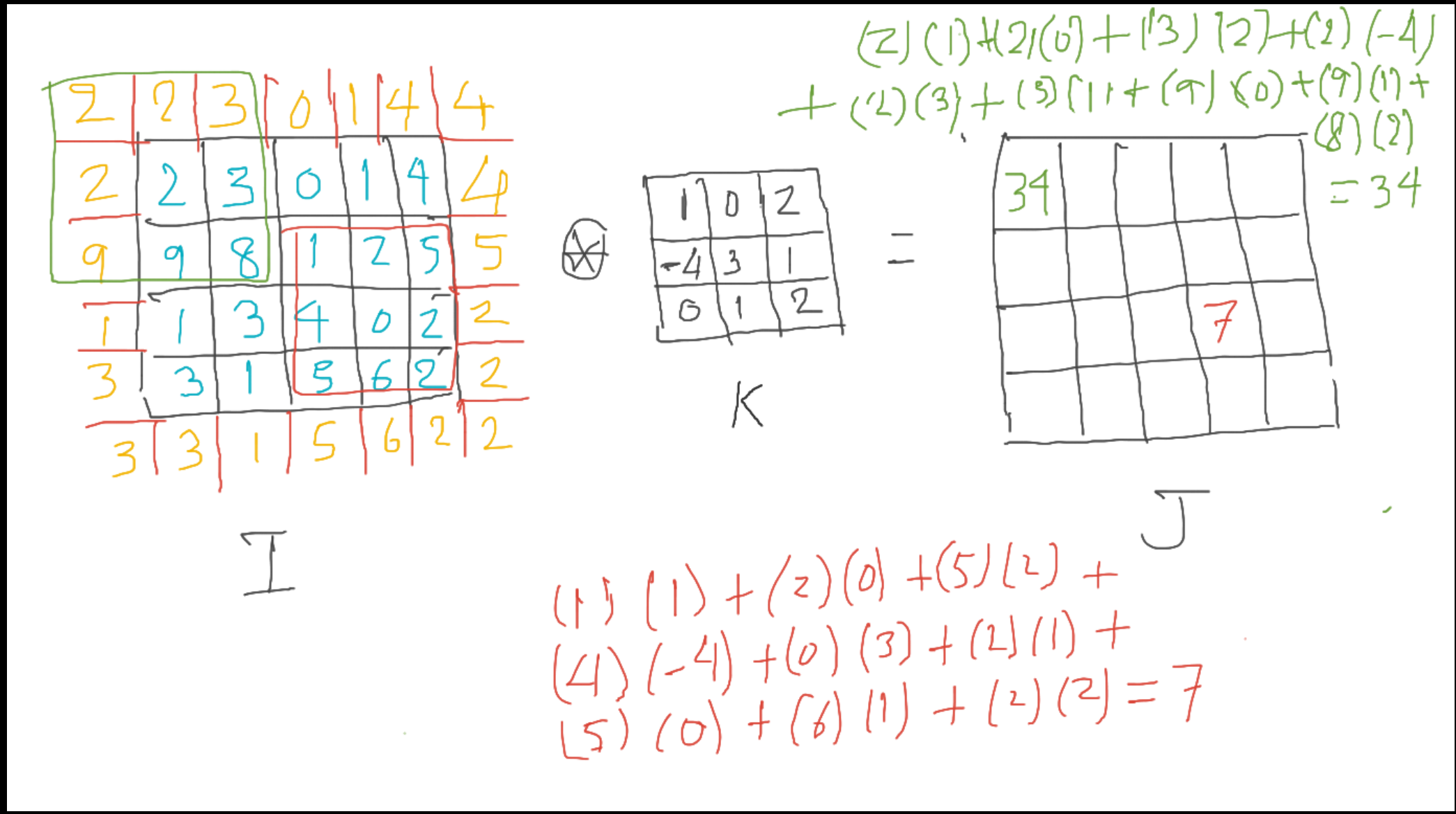
Block 2: Apply filter to every pixel in the image:

A. **Slide filter to the right until you hit right edge of image.**
B. **Slide one pixel down and return to the left most margin and repeat block 1.**





Intro: 2D Convolution recap



Intro: 2D Convolution recap

Inputs: Image I, filter matrix K; shape of K is (2h+1, 2w+1)
Output: Image J

Step 1: Pad h rows on the top and the bottom and w columns at the left and the right of image I; Let (H, W) be the shape of the padded image I

Step 2: Initialize output image J to all zeros; J has shape (H, W)

Step 3: for i from h to H-h-1
for j from w to W-w-1
for m from -h to h
for n from -w to w
J[i, j] += I[i+m, j+n] * K[m+h, n+w]

Step 4: Strip padding from J, so that J has shape (H-2h, W-2w). Return J.

Intro: Overlap, Slide and Multiply

```
for row in ROWS
  for col in COLS
    for Krow from -(KROWS-1)/2 to -(KROWS-1)/2 +1
      for Kcol in -(KCOLS-1)/2 to -(KCOLS-1)/2 +1
      Imout(row,col)=Kernel(krow+KROWS,kcol+KCOLS)*ImIn(row+krow,col+kcol)
```

Part 1: Create your own 2D convolution filtering script in Python

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

- Remember: two fundamental operations for filtering:
- Adding (integrate) is smoothing.
- Subtracting (differentiate) is sharpening.

Part 1: Create your own 2D convolution filtering script in Python

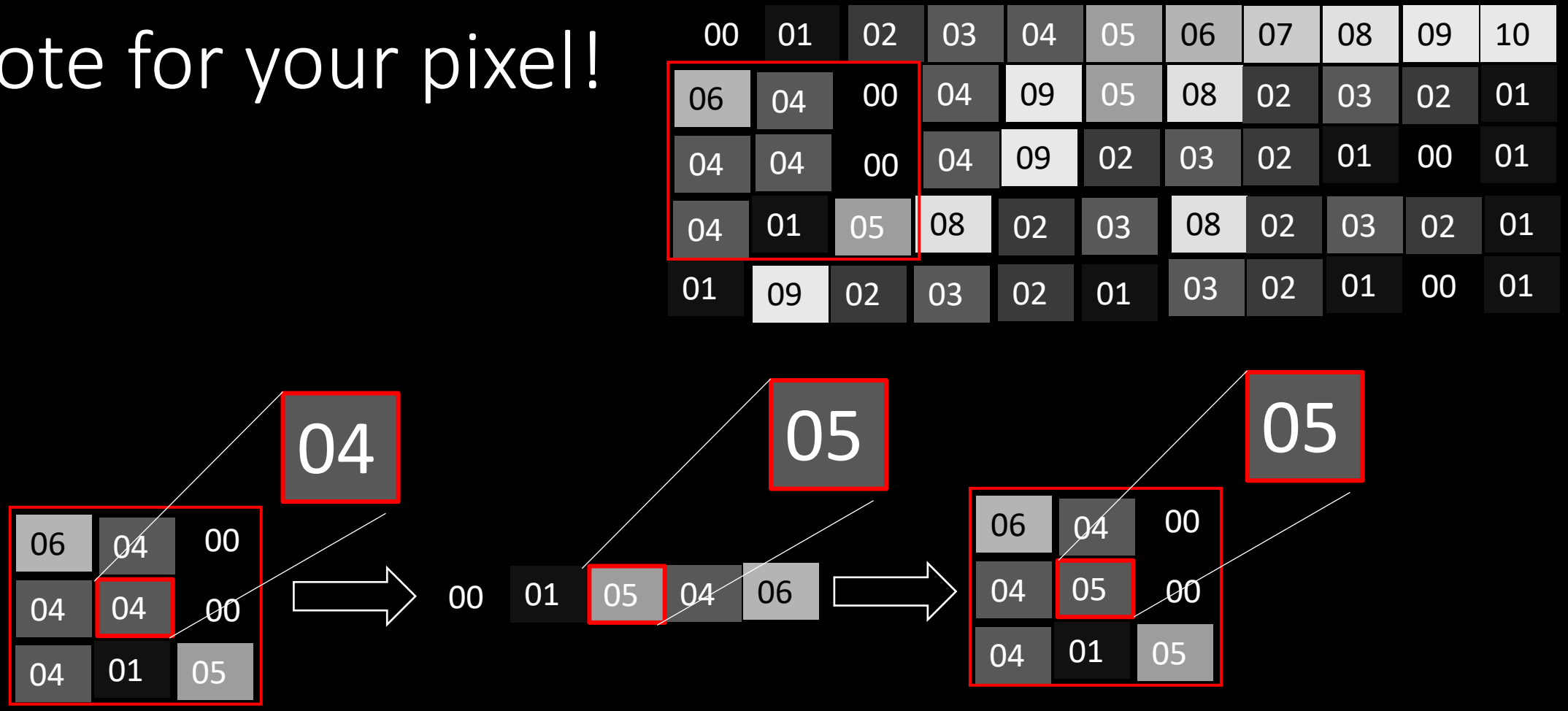
Iterate along rows
  Iterate along columns
  Write: ImOut(row+(Frows-1)/2+1,col+(Fcols-1)/2+1)=**sum(**Filter(1:Frows,1:Fcols)*ImageInPatch(row:row+Frows,col:col+Fcols))

Part 2:Median and Gaussian filters

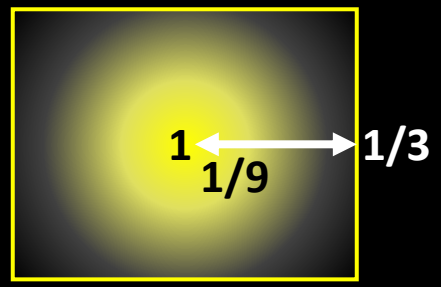
Read ***" noisy.jpg"*** corrupted with salt and pepper noise.

- Apply a median filter to remove the noise.
- Apply a Gaussian filter to the same noisy image.
- You can use any scikit-image function you like.
- Which filter was more successful?

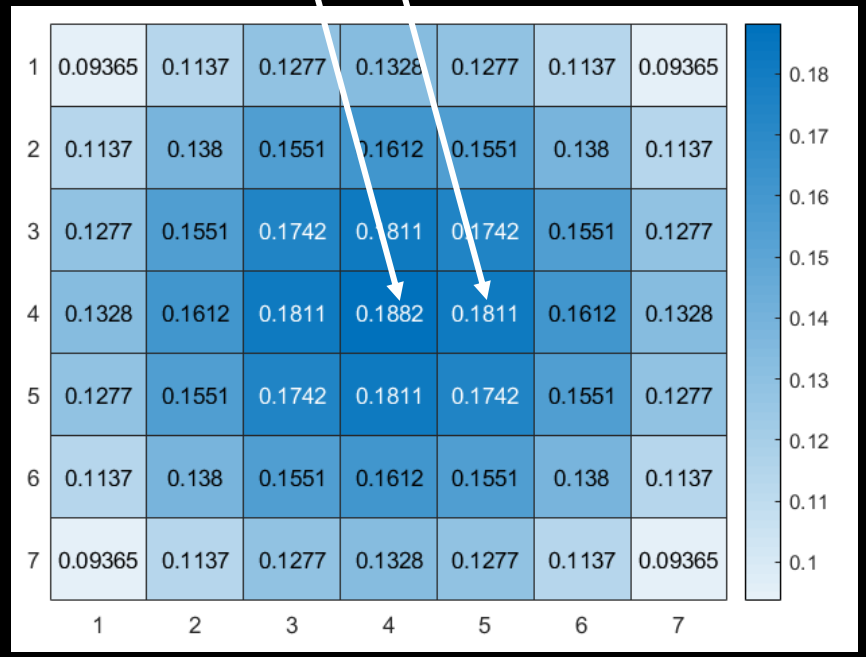
Part 2:Median…
Vote for your pixel!



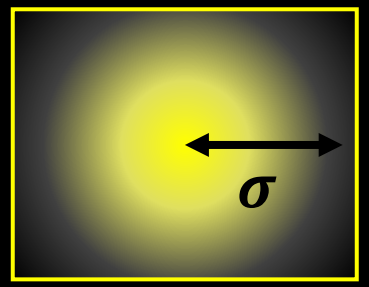
Part 2:Gaussian... merge your pixels!



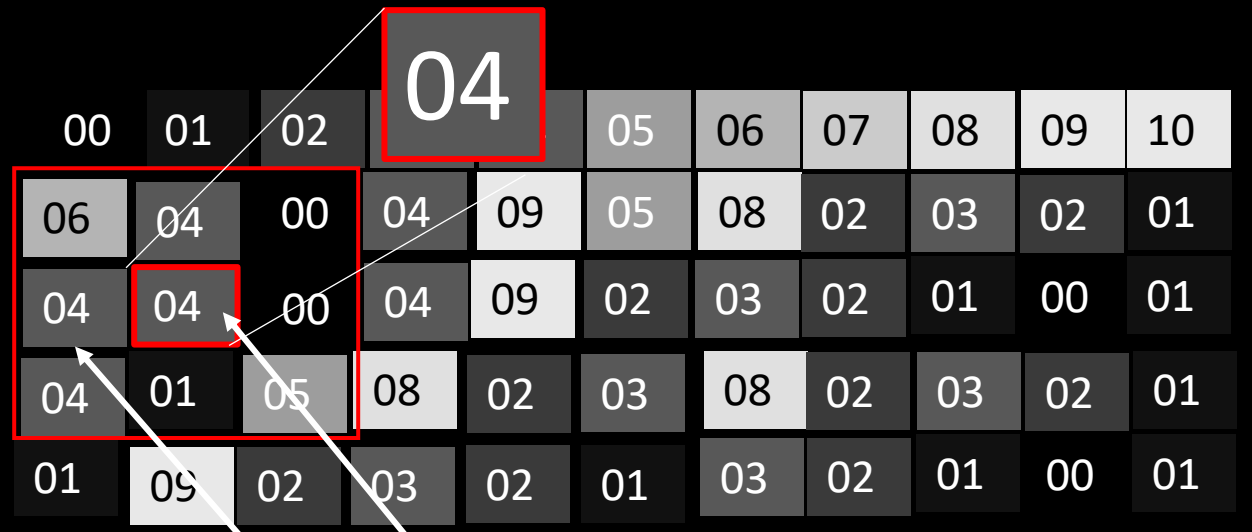
00	01	02	03	04	05	06	07	08	09	10
06	04	00	04	09	05	08	02	03	02	01
04	04	00	04	09	02	03	02	01	00	01
04	01	05	08	02	03	08	02	03	02	01
01	09	02	03	02	01	03	02	01	00	01

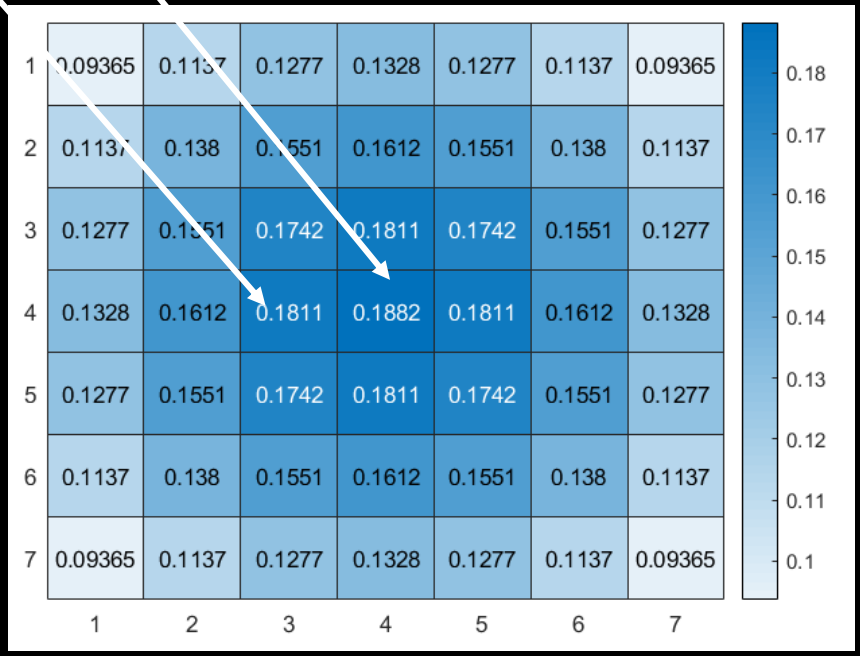


Part 2:Gaussian… merge your pixels!



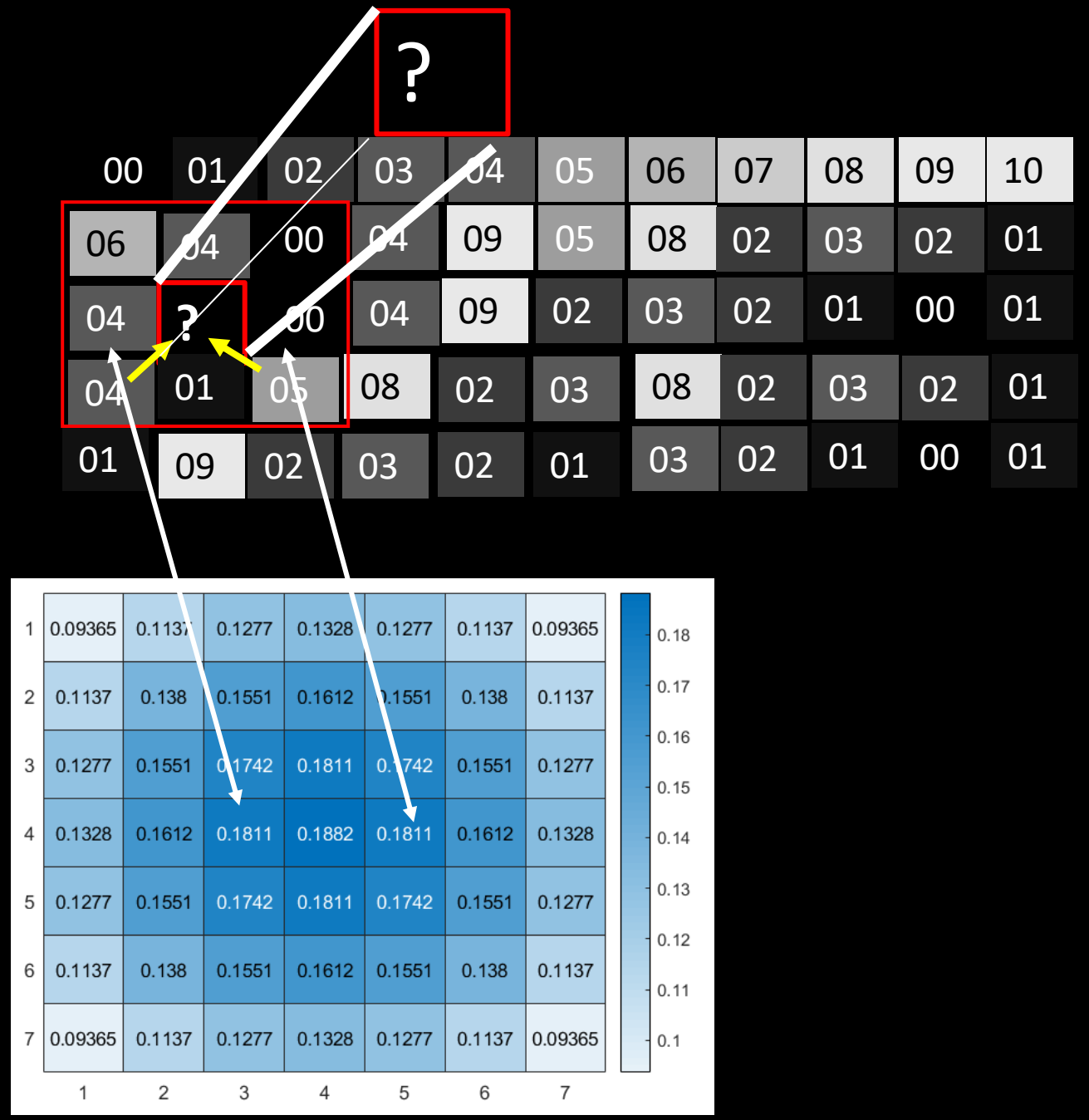
$$G = \frac{1}{2\pi\sigma^2} exp\left(-\frac{((i-i_0)^2 + (j-j_0)^2)}{2\sigma^2}\right)$$





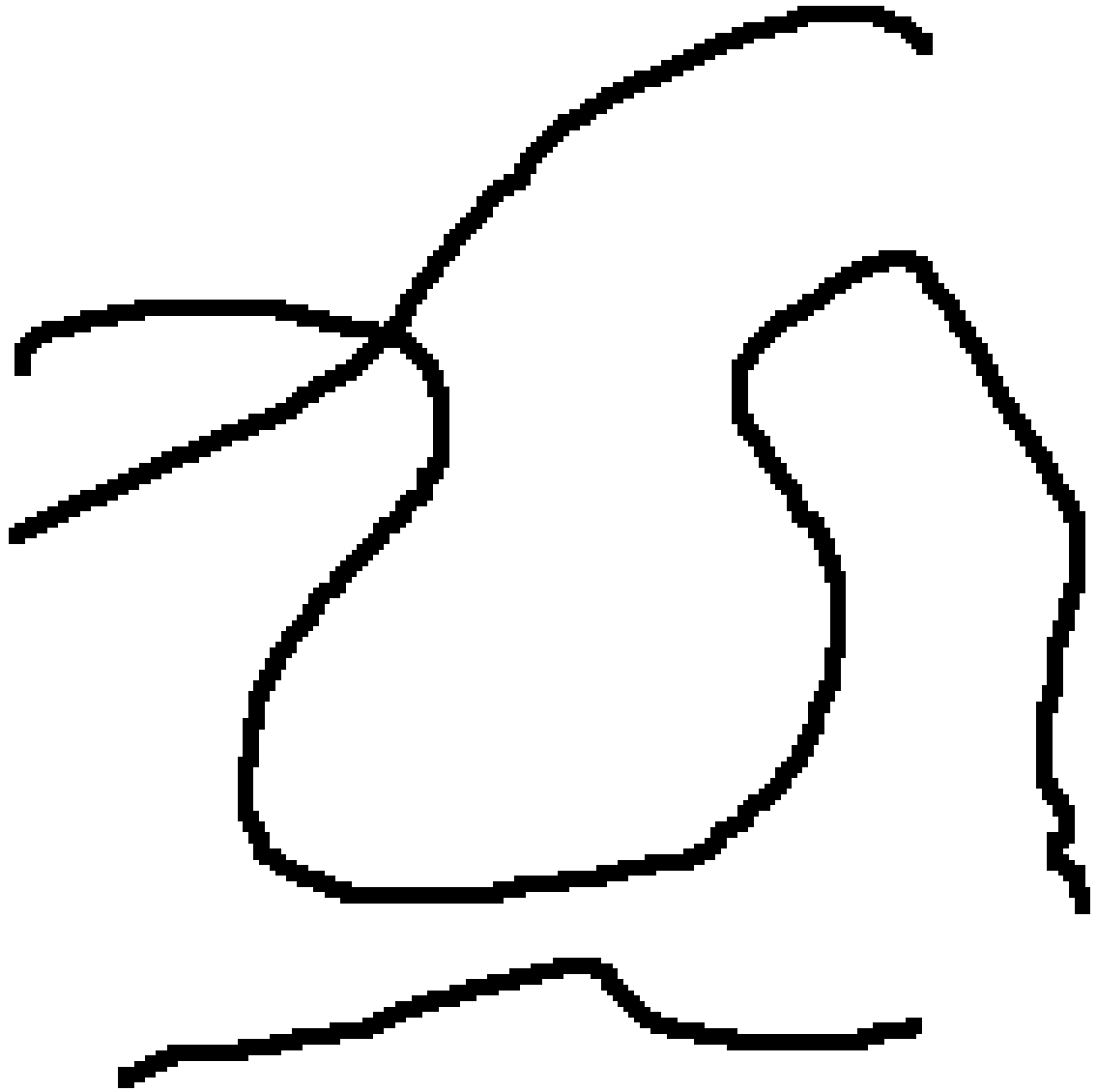
Part 3: Inpainting

1. Blurr image to fill-in voids.
2. Copy “good” pixels
3. Repeat process



Part 3: Inpainting

Part 3: Inpainting



Part 3: Inpainting

Part 3: Inpainting

Algorithm

- Input: Damaged image, I and Mask U
- Output: Repaired image,J
- Step 1: J = I
- Repeat
  - J = GaussianSmooth(J) # Smooth damaged image
  - J(U) = I(U) # Copy good pixels

Let's look at the demo

Assignment 2
Edges
Parts 4-6

CMPUT206

Prof. Nilanjan Ray

TA: Bernal Manzanilla

Parts 4-6: Subtracting is sharpening. Summing is smoothing.

- Taking differences of neighbouring pixels, is a discrete equivalent of a derivative, which brings up amplitudes of edges.

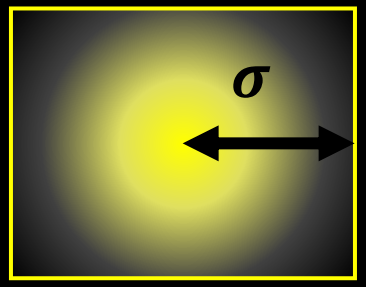
- Differences increase noise.

- Summing neighboring pixels is smoothing.

- Combination of summing filters and difference filters control noise and allow to sharpen or identify edges, lines and corners in natural images.

$$\frac{d(Image)}{dx} = \mathrm{Dx} = (\mathbf{1/2}, \mathbf{-1/2})$$

$$\frac{d(Image)}{dy} = \mathrm{Dx}^T = Dy = \begin{pmatrix} \mathbf{1/2} \\ \mathbf{-1/2} \end{pmatrix}$$



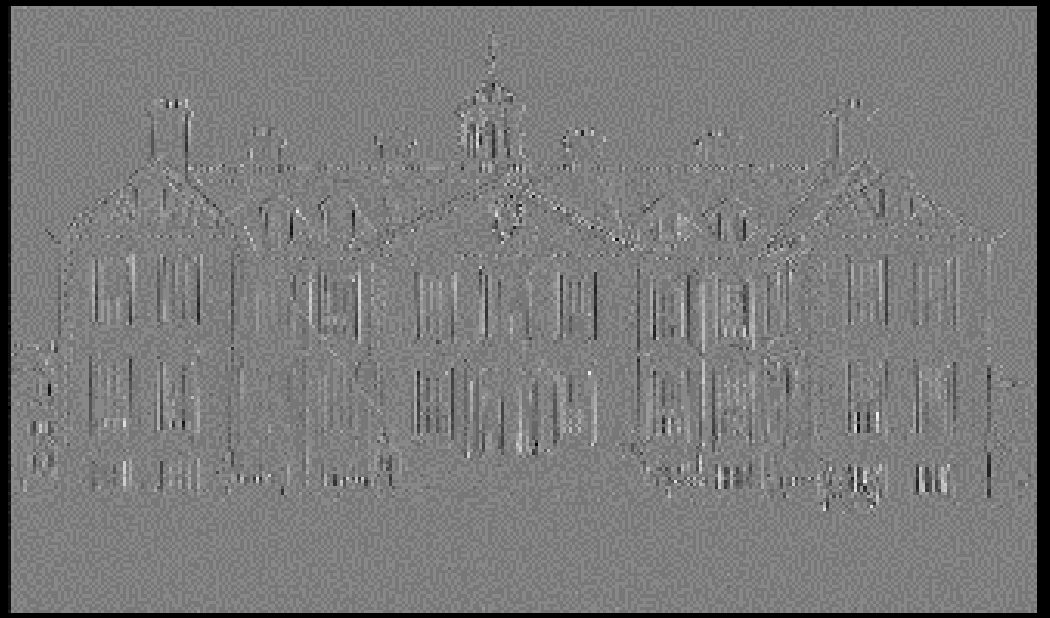
$$\mathrm{G} = \frac{1}{2\pi\sigma^2} exp(-\frac{((i-i_0)^2 + (j-j_0)^2)}{2\sigma^2})$$

$$Sx=\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

Part 4: Sobel operator

$$Sy=\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

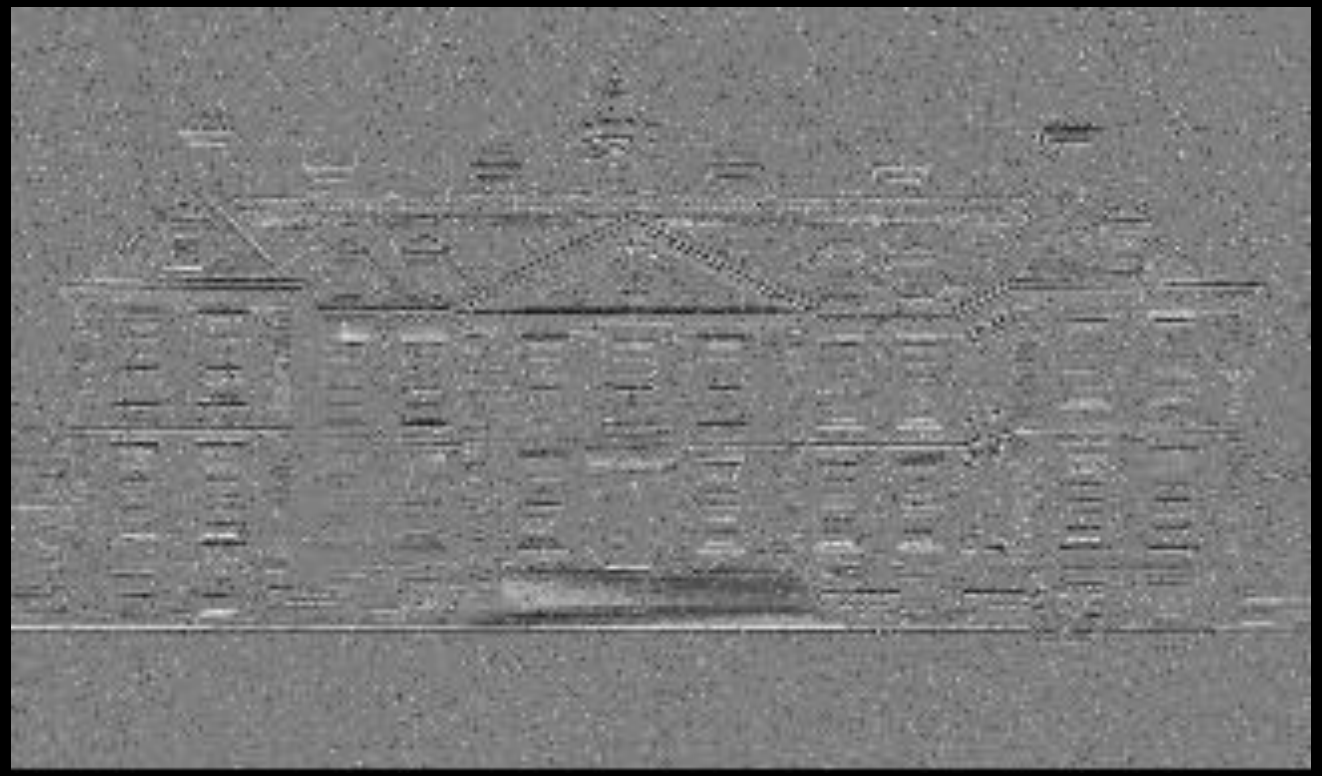
$$Sx=\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$



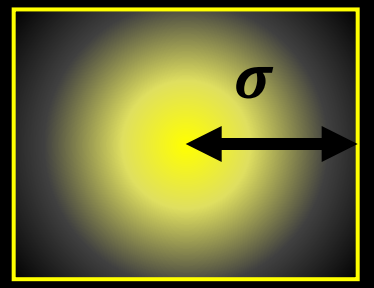
Part 4: Sobel operator

$$Sy=\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Part 5: Canny



Part 5: Canny filter. Smooth, calculate gradient and apply lower bound threshold.



$$G = \frac{1}{2\pi\sigma^2} exp\left(-\frac{((i-i_0)^2 + (j-j_0)^2)}{2\sigma^2}\right)$$

$$I_{smooth} = G * I \Rightarrow I_{x/y} = D_{x/y} * I_{smooth}$$

$$D_x = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{2} & \mathbf{0} & \mathbf{-2} \\ \mathbf{1} & \mathbf{0} & \mathbf{-1} \end{pmatrix}$$

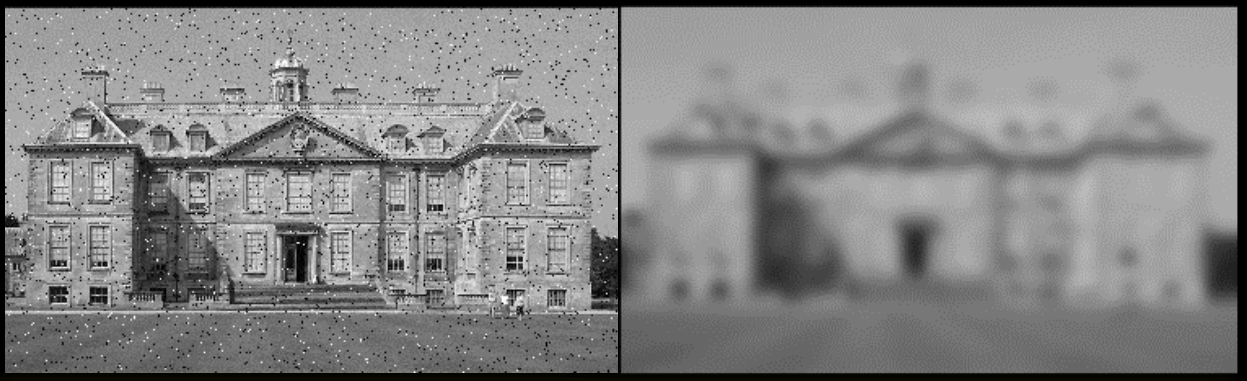
$$D_y = \begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{-1} & \mathbf{-2} & \mathbf{-1} \end{pmatrix}$$

Part 5: Canny

Canny → **Smooth** → **gradient** → **Threshold**

I

Part 5: Canny



Canny → **Smooth** → **gradient** → **Threshold**

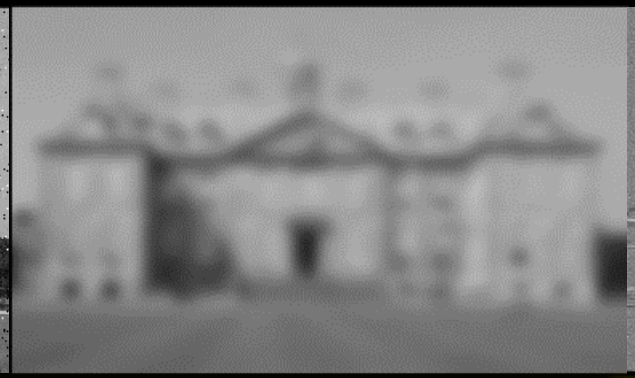
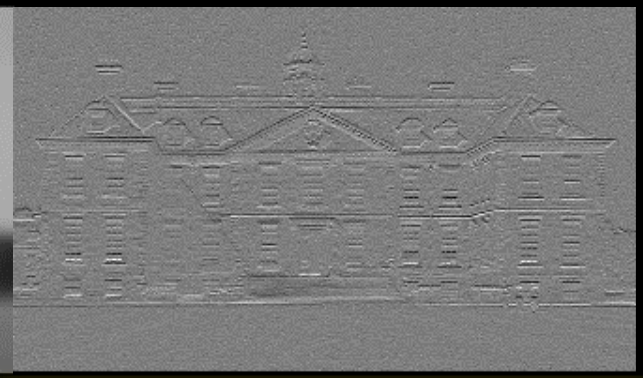
$$I \Rightarrow I_{smooth} = G * I$$

Part 5: Canny

Canny → **Smooth** → **gradient** → **Threshold**

$$I \Rightarrow I_{smooth} = G * I \Rightarrow I_{x/y} = D_{x/y} * I_{smooth}$$

Part 5: Canny


Canny → **Smooth** → **gradient** → **Threshold**

$$I \Rightarrow I_{smooth} = G * I \Rightarrow I_{x/y} = D_{x/y} * I_{smooth} \Rightarrow I_{x/y} \begin{cases} 0 & I_{x/y} < t \\ I_{x/y} & I_{x/y} \geq t \end{cases}$$