Intelligent Control Systems

Image Processing (2)
— Filtering, Geometric Transforms and Colors —

Shingo Kagami

Graduate School of Information Sciences,

Tohoku University

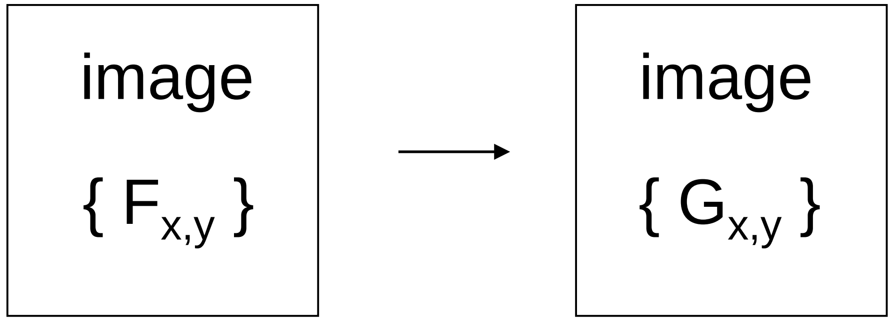
swk(at)ic.is.tohoku.ac.jp

http://www.ic.is.tohoku.ac.jp/ja/swk/

Taxonomy

input	output	example
image	image (2-D data)	image-to-image conversion
	1-D data	projection, histogram
	scalar values	position, object label

Image to Image



point operation

$G_{i,j}$ depends only on $F_{i,j}$ (thresholding, pixel value conversion, …)

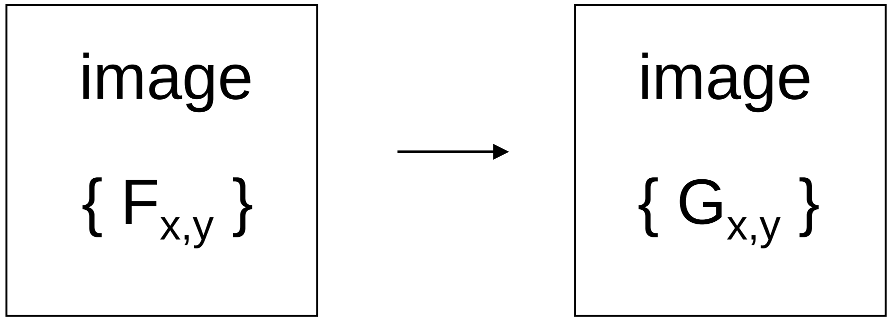
local operation / neighboring operation

$G_{i,j}$ depends on pixels within some neighborhood of $F_{i,j}$

global operation

$G_{i,j}$ depends on almost all the pixels in $\{ F_{i,j} \}$

Image to Image



point operation

$G_{i,j}$ depends only on $F_{i,j}$

local operation / neighboring operation

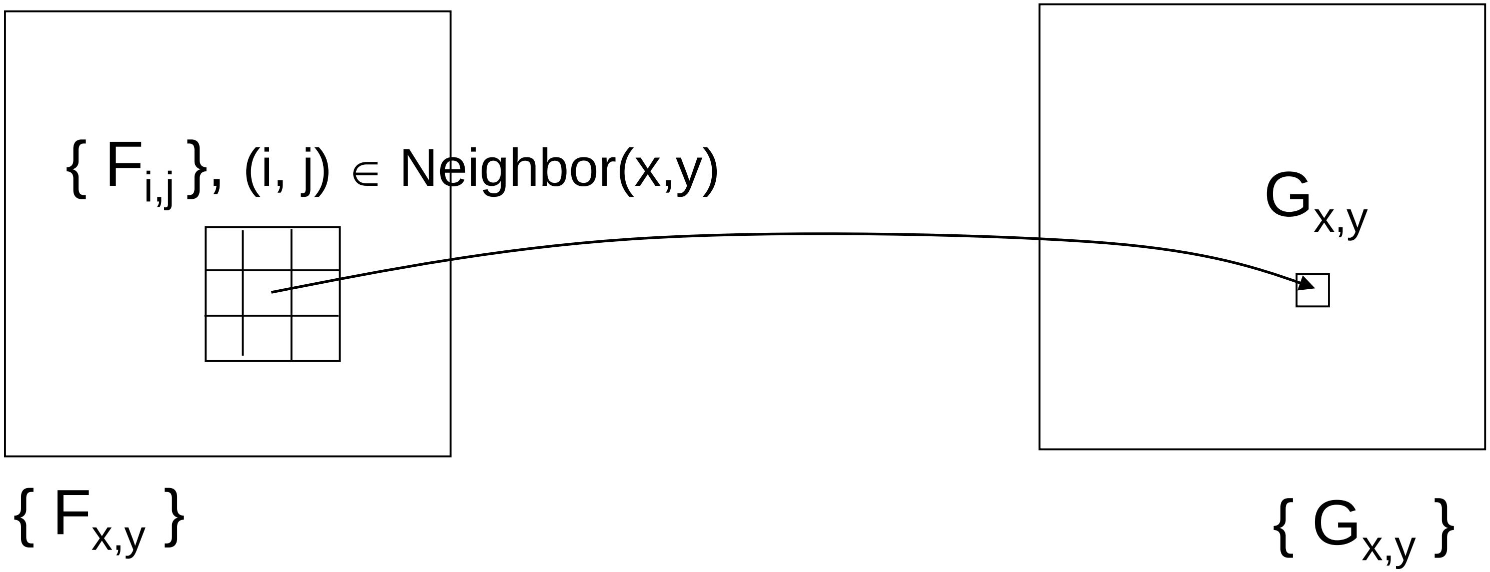
$G_{i,j}$ depends on pixels within some neighborhood of $F_{i,j}$

global operation

$G_{i,j}$ depends on almost all the pixels in $\{ F_{i,j} \}$

Local operation example: Spatial Filter

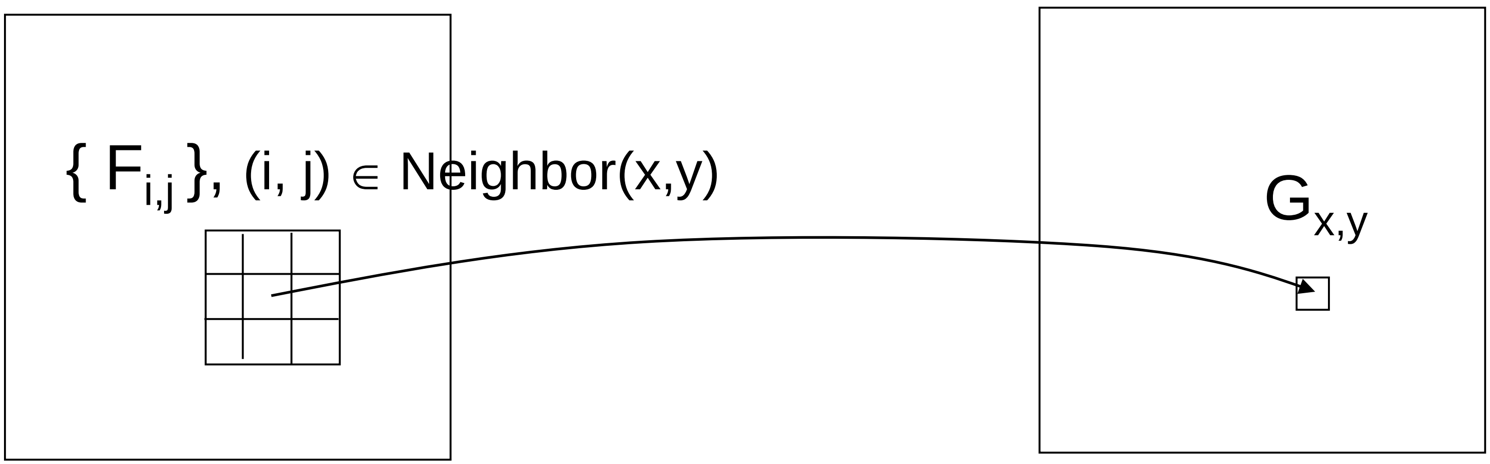
$G_{x,y}$ depends on some neighborhood (e.g. 3×3, 5×5 pixels, etc.) of the point of interest (x,y)



Typical examples: smoothing, edge detection

Important Example: Smoothing

- Output at (x, y): some representative value of the set of neighbor pixels around (x, y), e.g. mean, weighted mean, median
- Used for: e.g. noise reduction, scale-space processing



1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

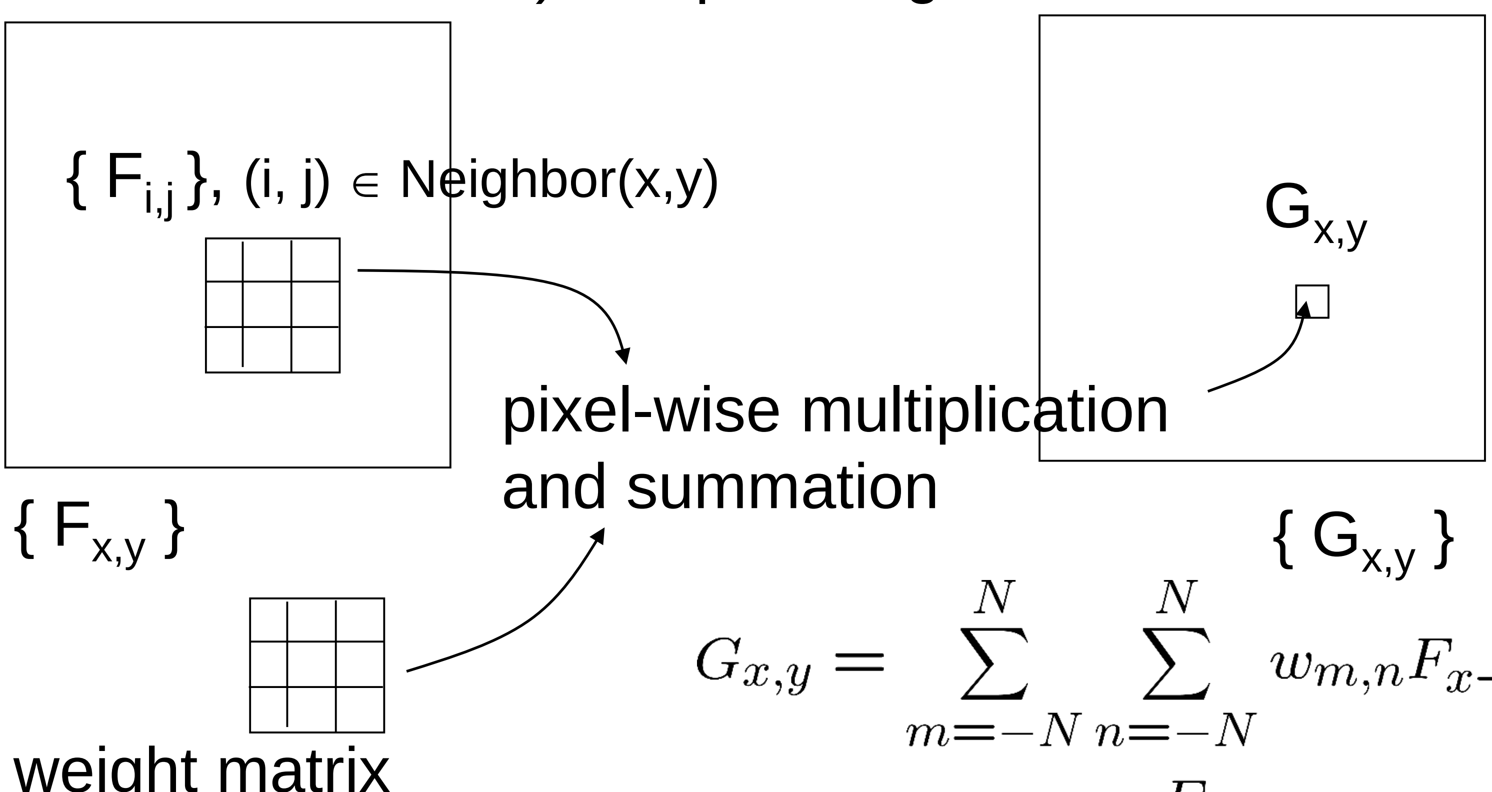
(mean)

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

(weighted mean)

Linear Spatial Filtering

- Smoothing with (weighted) mean is an example of linear spatial filtering (while smoothing with median is nonlinear)
- Computed by convolving a weight matrix (filter coefficients, filter kernel, or mask) to input image



$$G_{x,y} = \sum_{m=-N}^{N} \sum_{n=-N}^{N} w_{m,n} F_{x+m,y+n}$$
$$= w_{-x,-y} * F_{x,y}$$

Examples of 3x3 smoothing weight matrices

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

1/9

1	1	1
1	1	1
1	1	1

1/10	1/10	1/10
1/10	1/5	1/10
1/10	1/10	1/10

=

1/10

1	1	1
1	2	1
1	1	1

0	1/8	0
1/8	1/2	1/8
0	1/8	0

=

1/8

0	1	0
1	4	1
0	1	0

Implementation of 3x3 filtering

02_filter3x3.py:

```
weight = 1.0/8 * np.array([[0, 1, 0],
                           [1, 4, 1],
                           [0, 1, 0]])

for j in range(1, height - 1):
    for i in range(1, width - 1):

        sum = 0.0
        for n in range(3):
            for m in range(3):
                sum += weight[n, m] * src[j + n - 1, i + m - 1]
        dest[j, i] = int(saturate(sum))
```

Generates `[1, 2, …, height - 2]` (a lazy way of boundary handling)

Unlike the mathematical definition, the center coordinate of weight is not (0, 0) but (1, 1)

OpenCV functions for common filters

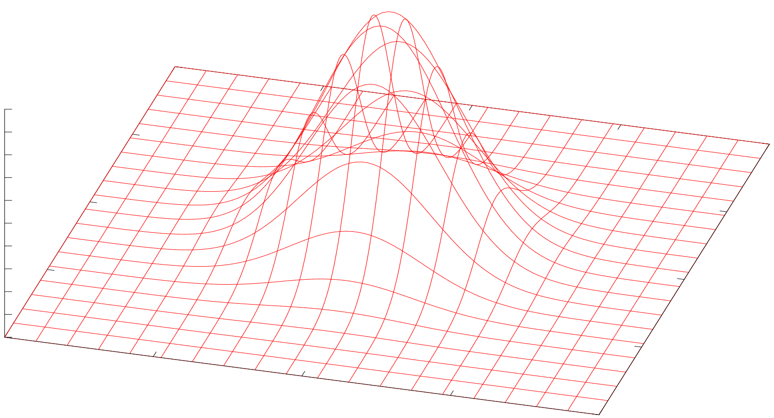
```
cv2.filter2D()

cv2.GaussianBlur()
cv2.Sobel()
cv2.Laplacian()
…

cv2.medianBlur()
cv2.dilate()
cv2.erode()
…
```

Gaussian: most widely used smoothing kernel

$$g_\sigma(x,y) = \frac{1}{\sqrt{2\pi}\sigma}\exp\{-\frac{x^2}{2\sigma^2}\} \cdot \frac{1}{\sqrt{2\pi}\sigma}\exp\{-\frac{y^2}{2\sigma^2}\}$$

$$= \frac{1}{2\pi\sigma^2}\exp\{-\frac{x^2+y^2}{2\sigma^2}\}$$

separable in x and y



- Discretized in space for computation
- Coefficient values are sometimes rounded to integer (for efficiency)
- Amount of smoothing can be controlled by parameter σ (large σ requires large matrix size)

Frequency-domain understanding

$$G_{x,y} = \sum_{m=-N}^{N} \sum_{n=-N}^{N} w_{m,n} F_{x+m,y+n}$$

$$= w_{-x,-y} * F_{x,y} \quad \xrightarrow{\mathcal{F}} \quad \mathcal{F}\left[w_{-x,-y}\right] \cdot \mathcal{F}\left[F_{x,y}\right]$$

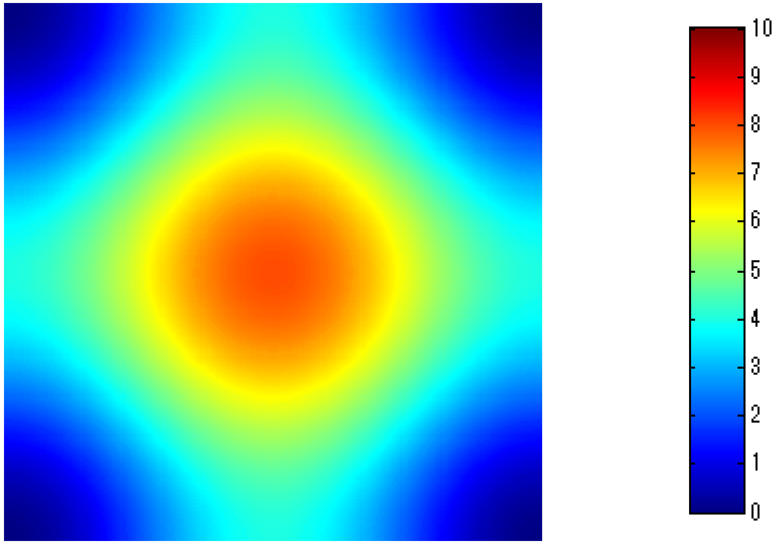
$\mathcal{F}[\cdot]$: 2-D discrete Fourier transform

02_filter3x3_fft.py:

0	1	0
1	4	1
0	1	0

(zero-padded to 256x256 and)

$\xrightarrow{\mathcal{F}}$



Recall: Fourier transform of Gaussian function is Gaussian

Edge Detection

- Spatial differentiation (approximated by finite difference)

0	0	0
-1	0	1
0	0	0

1st order diff. in x direction

0	-1	0
0	0	0
0	1	0

1st order diff. in y direction

- Often combined with smoothing:

-1	0	1
-2	0	2
-1	0	1

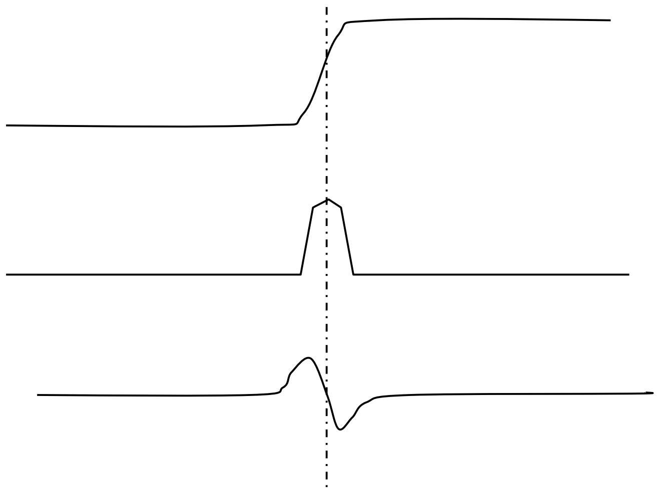
Sobel filter in x direction

-1	-2	-1
0	0	0
1	2	1

Sobel filter in y direction

Edge detection by 2^{nd} order derivative

- Edge = zero crossing of 2^{nd} order derivative
- Laplacian $\partial^2/\partial x^2 + \partial^2/\partial y^2$ is the lowest-order isotropic differential operator
  - does not depend on direction of edges

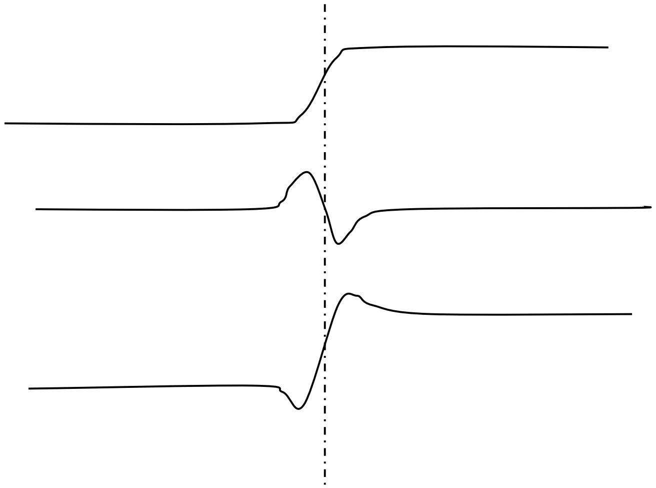


- Laplacian operator is realized by adding 2^{nd} order differentials $f_{i+1} - 2 f_i + f_{i-1}$ of x and y directions

0	1	0
1	–4	1
0	1	0

Sharpening

Subtract the Laplacian image from the original image to yield an edge-enhanced image



0	0	0
0	1	0
0	0	0

–

0	1	0
1	–4	1
0	1	0

=

0	-1	0
-1	5	-1
0	-1	0

Frequency-domain visualization

Laplacian:

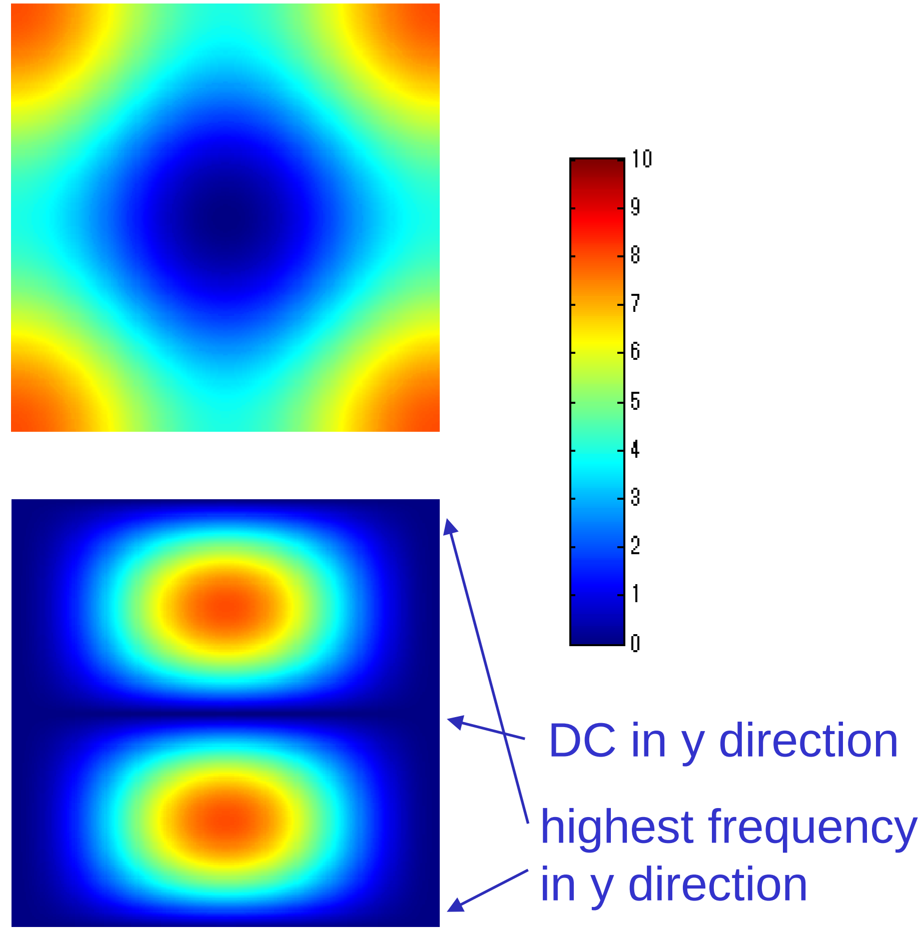
0	1	0
1	-4	1
0	1	0

$\xrightarrow{\mathcal{F}}$

Sobel:

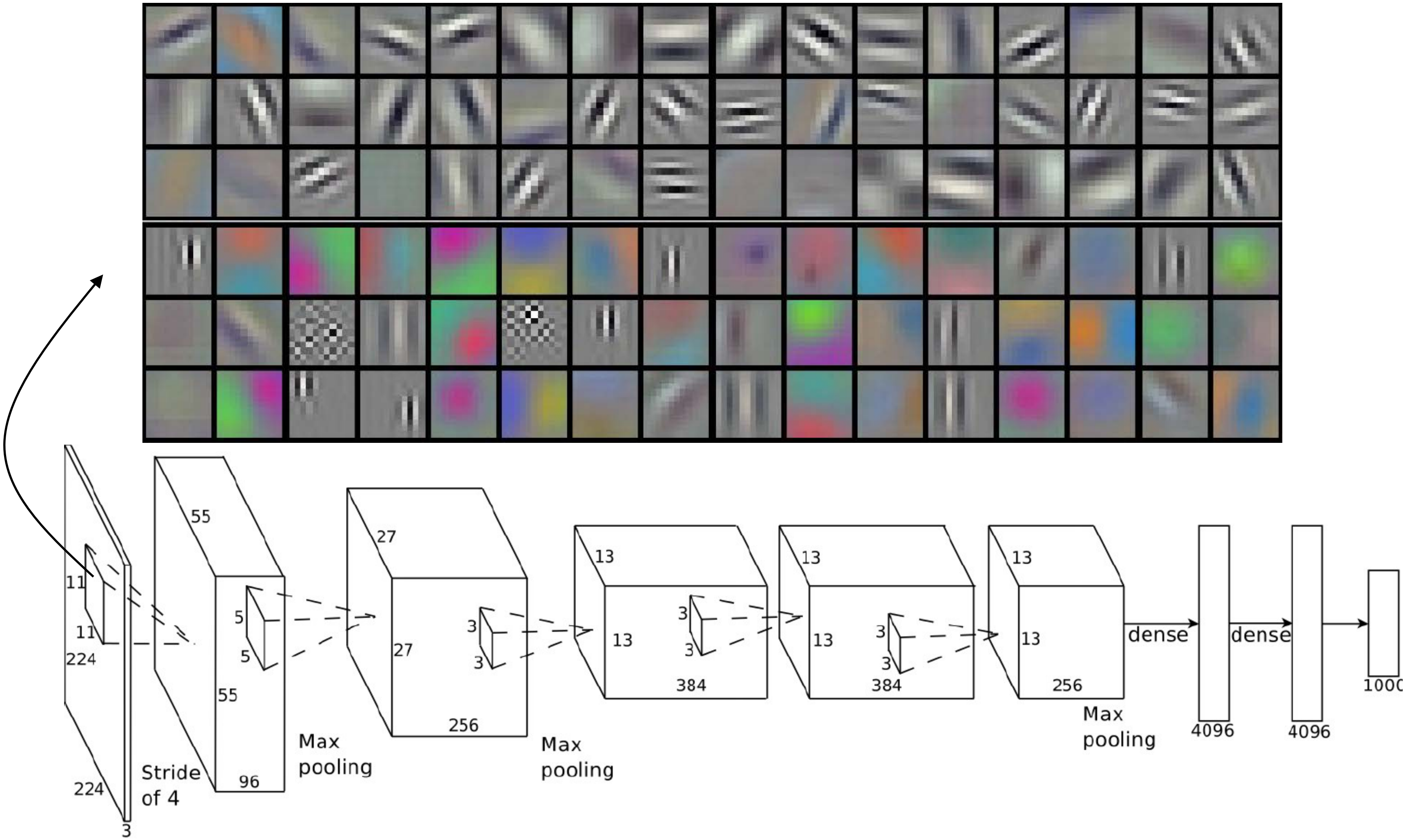
1	2	1
0	0	0
-1	-2	-1

$\xrightarrow{\mathcal{F}}$



Q: Why is Sobel a band-pass filter instead of high-pass?

Deep Convolutional Neural Networks



Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.

Image to Image

image $\{ F_{x,y} \}$ → image $\{ G_{x,y} \}$

point operation

$G_{i,j}$ depends only on $F_{i,j}$

local operation / neighboring operation

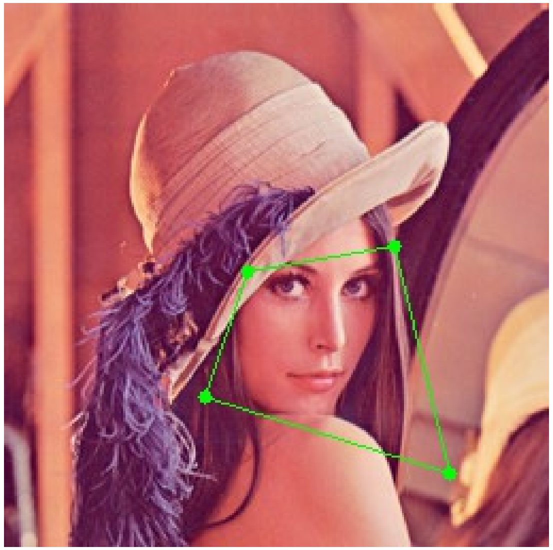
$G_{i,j}$ depends on pixels within some neighborhood of $F_{i,j}$

global operation

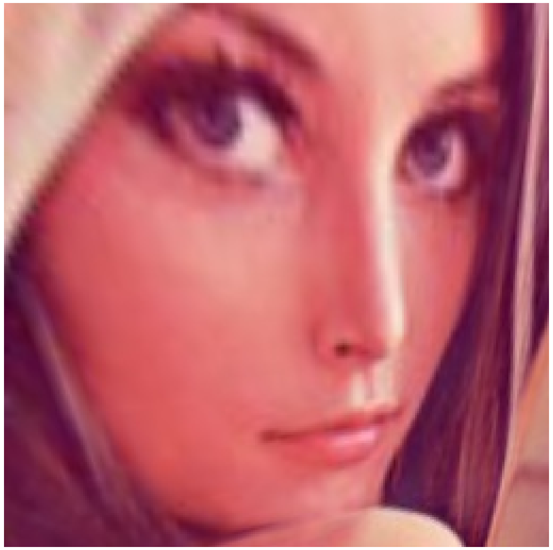
$G_{i,j}$ depends on almost all the pixels in $\{ F_{i,j} \}$

Global operation example: Warping

02_warp.py:



$\{ F_{x,y} \}$



$\{ G_{x,y} \}$

- $G_{x,y}$ is sampled from $F_{x',y'}$ where (x', y') is determined from (x, y)

Important Geometric Transforms

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Translation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha\cos\theta & -\alpha\sin\theta & t_x \\ \alpha\sin\theta & \alpha\cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Similarity Transform

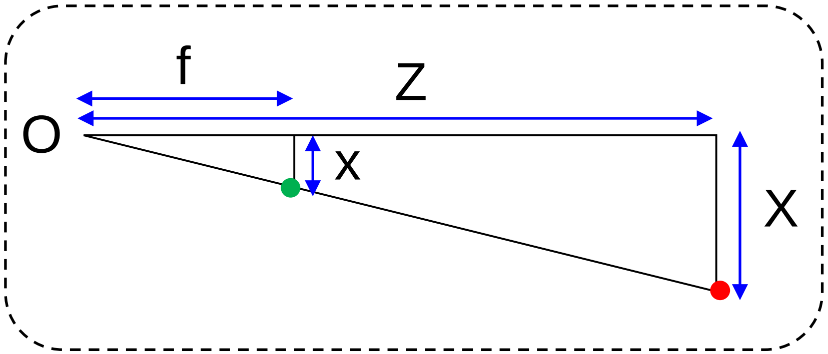
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

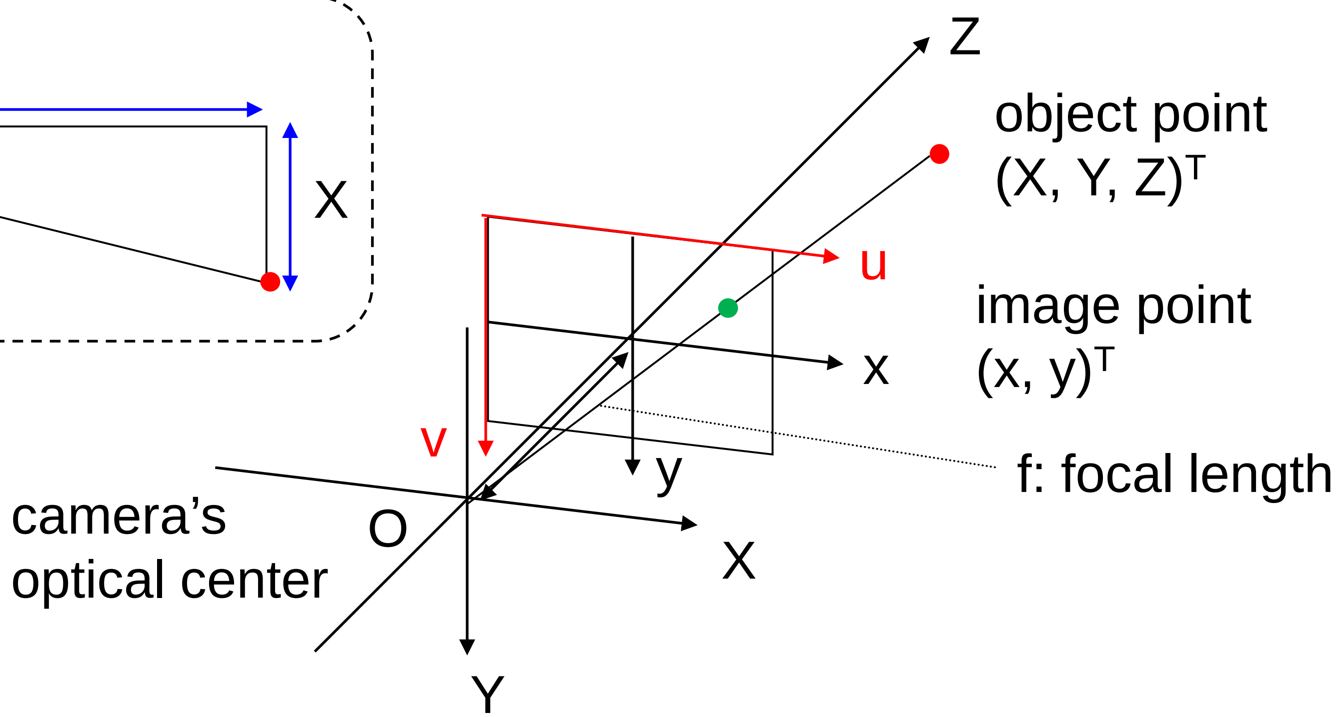
Affine Transform

$$s\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Homography Transform
(Perspective Transform)
(Collineation)

Understanding Homography (1/3)





$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix}$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} k_u x + c_u \\ k_v y + c_v \end{pmatrix}$$

X-Y-Z: camera coordinate frame
x-y: (normalized) image coordinate frame

u-v: image coordinate
(pixel coordinate) frame

Understanding Homography (2/3)

By substituting $\begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix}$ into $\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} k_u x + c_u \\ k_v y + c_v \end{pmatrix}$

$$Z\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fk_u & 0 & c_x \\ 0 & fk_v & c_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = A\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

A: 3x3 camera intrinsic matrix

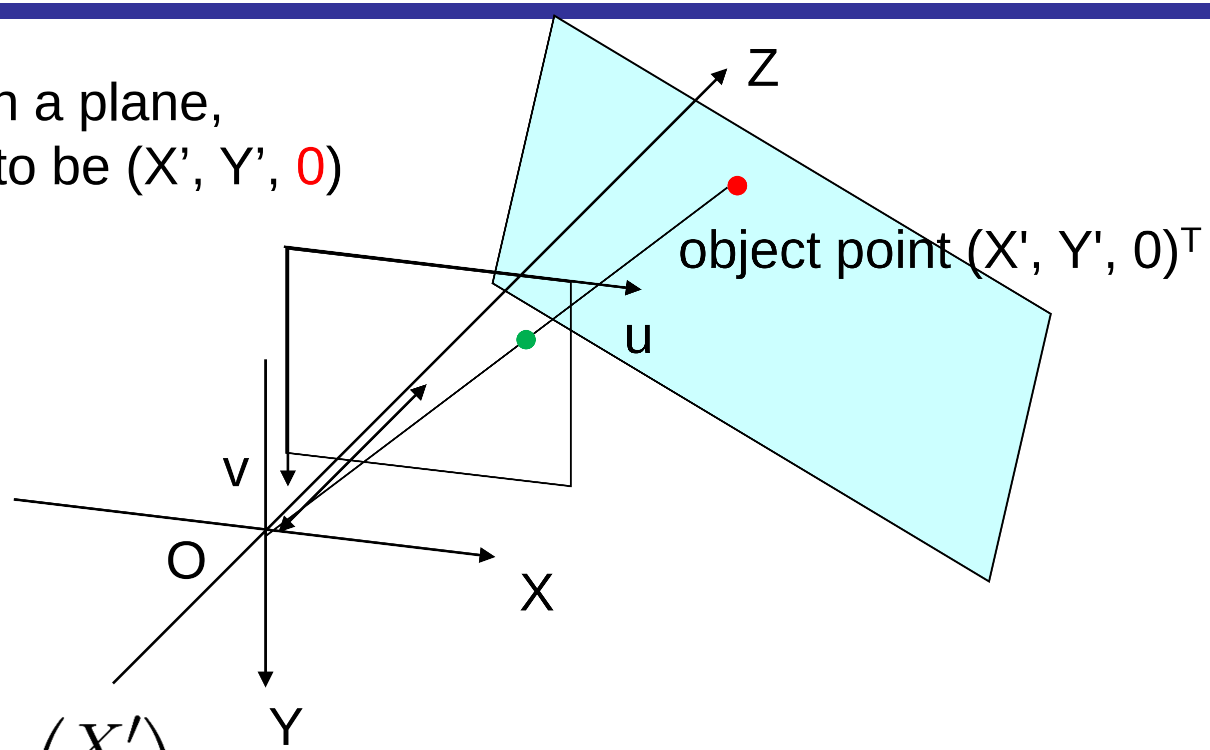
For an arbitrary object coordinate frame X'-Y'-Z',

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} R & | & \boldsymbol{t} \end{pmatrix}\begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix}$$

R: 3x3 rotation matrix
t: 3D translation vector

Understanding Homography (3/3)

When the object point is on a plane,
its coordinate is assumed to be (X’, Y’, 0)
without loss of generality



$$Z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = A \begin{pmatrix} R & | & \boldsymbol{t} \end{pmatrix} \begin{pmatrix} X' \\ Y' \\ 0 \\ 1 \end{pmatrix} = H \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix}$$

- *H* determines bijective mapping between (u, v) and (X’, Y’)
- *H* is computed when n ($n \geq 4$) corresponding points are given

Homography Warping by OpenCV

02_warp.py:

```
src_pnts = np.array([[100, 100],
                     [200, 100],
                     [200, 200],
                     [100, 200]],
                    np.float32)
dest_size = 256
dest_pnts = np.array([[0, 0],
                      [dest_size - 1, 0],
                      [dest_size - 1, dest_size - 1],
                      [0, dest_size - 1]],
                     np.float32)

H = cv2.getPerspectiveTransform(src_pnts, dest_pnts)

output = cv2.warpPerspective(input, H, (dest_size, dest_size))
```

4 points [X, Y]'s

4 points [X', Y']'s

Image to Image

image $\{ F_{x,y} \}$ → image $\{ G_{x,y} \}$

point operation

$G_{i,j}$ depends only on $F_{i,j}$ (thresholding, pixel value conversion, …)

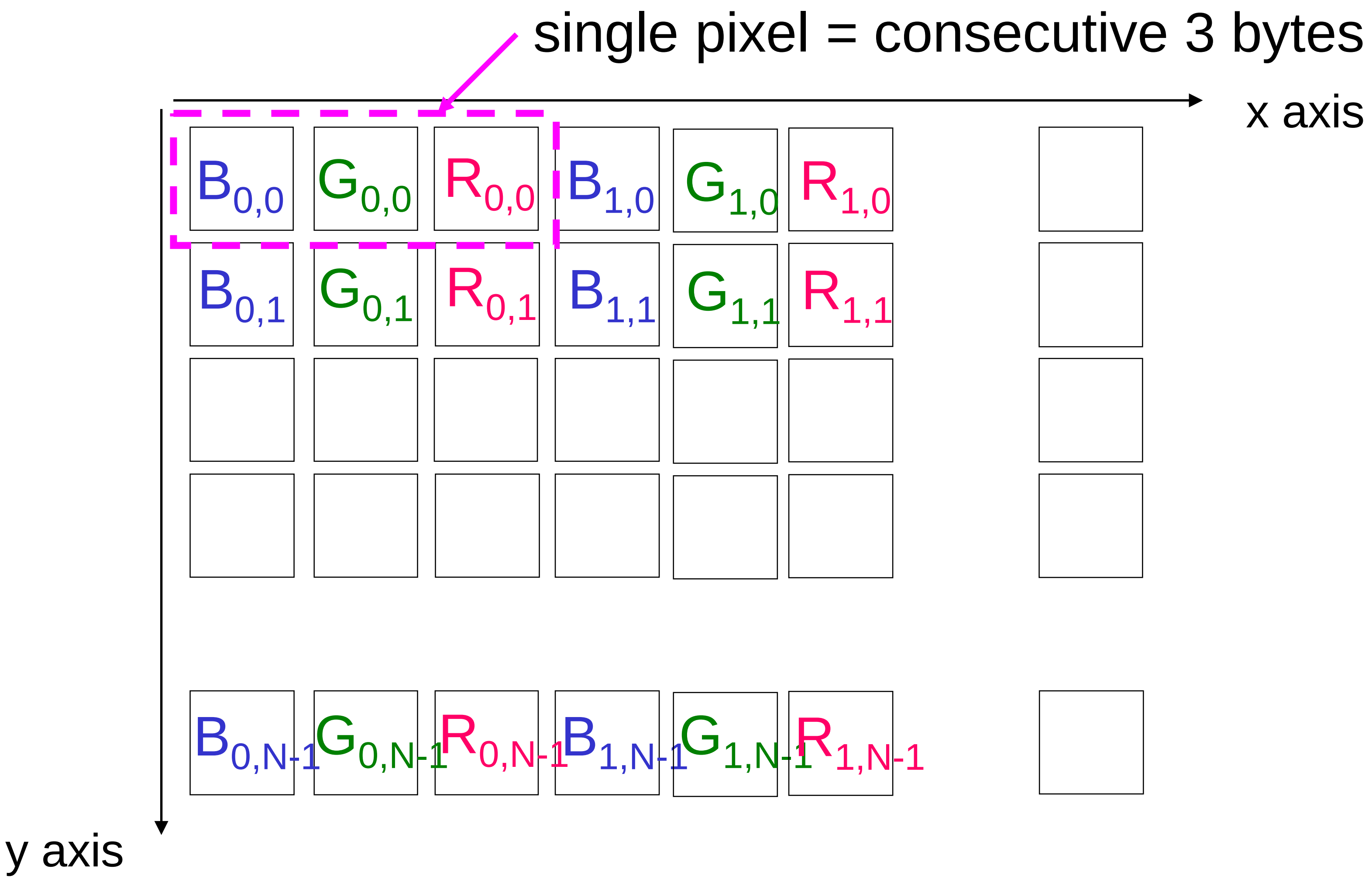
local operation / neighboring operation

$G_{i,j}$ depends on pixels within some neighborhood of $F_{i,j}$

global operation

$G_{i,j}$ depends on almost all the pixels in $\{ F_{i,j} \}$

Color Image Representation



Representation in OpenCV for Python (NumPy)

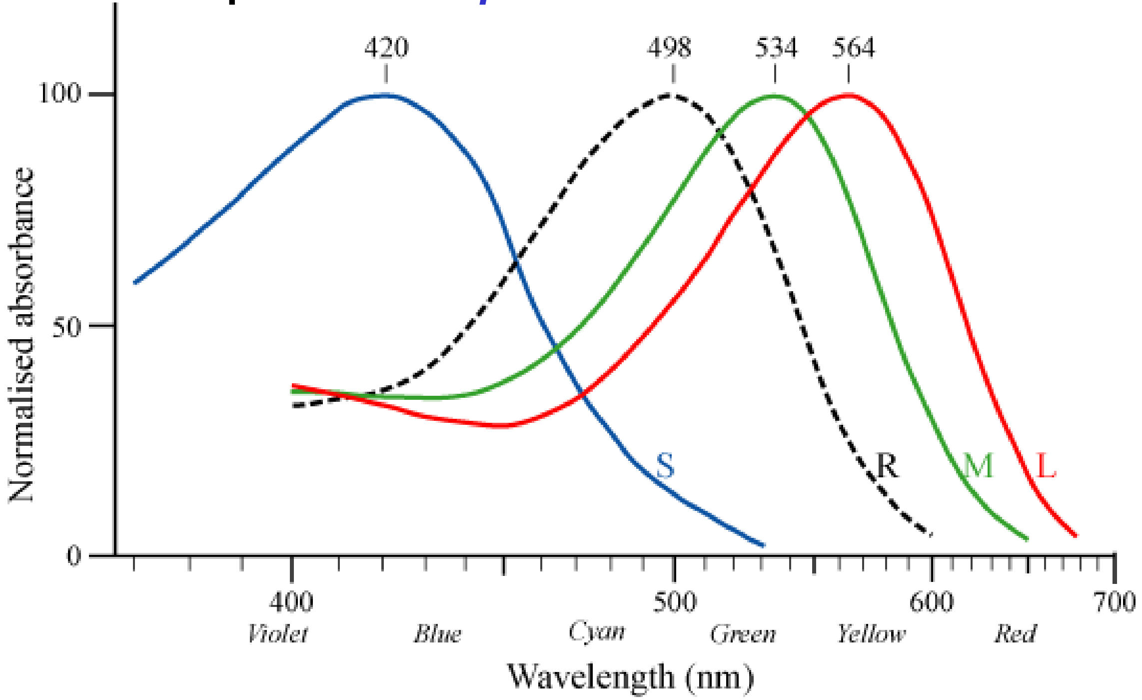
(Y, X) array with 3 channels (or, (Y, X, 3) tensor) is used

```
fruits = cv2.imread('fruits.jpg')
fruits.shape
  -> (480, 512, 3)

fruits[100, 100]
  -> array([ 52,  98, 116], dtype=uint8)
fruits[100, 100, 0]
  -> 52
```

RGB Color Space

Why R, G, and B?

- Our eyes have three types of wavelength-sensitive cells (cone cells)
  - cf. rod cells
- So, the color space we *perceive* is three-dimensional



http://commons.wikimedia.org/wiki/File:Cone-response.png

Other Color Spaces

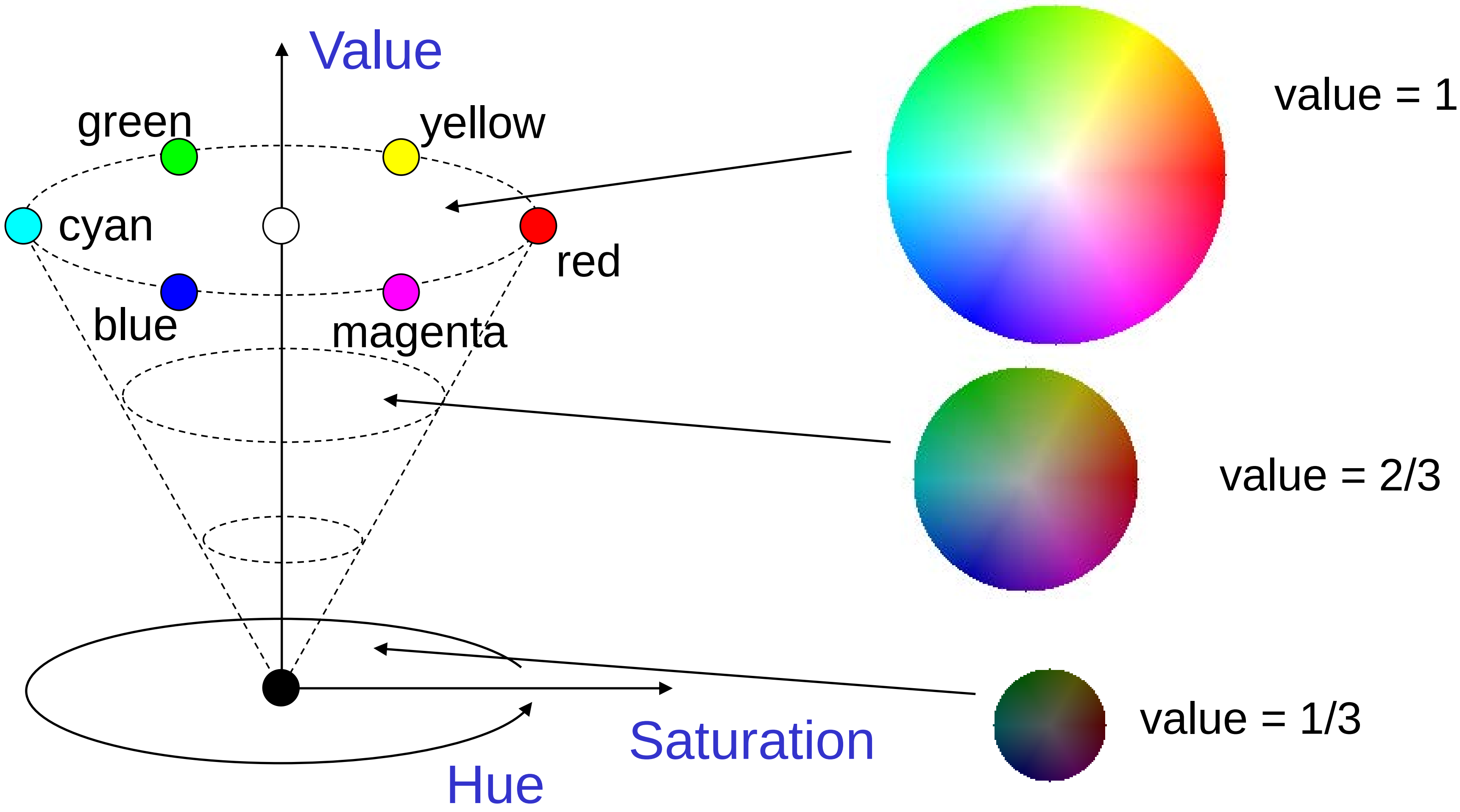
XYZ, L*a*b, L*u*v
    defined by CIE (Commission Internationale de l'Eclairage)
YIQ, YUV, YCbCr
    used in video standards (NTSC, PAL, …)
HSV (HSI, HSL)
    based on Munsell color system

cf. CMY, CMYK (for printing; subtractive color mixture)

```
output = cv2.cvtColor(input, cv2.COLOR_BGR2HSV)
```

HSV Color Space

02_hsv_illustrated.py
02_convert_color.py



Common Definition: $0 \leq \text{Hue} \leq 360$, $0 \leq \text{Saturation} \leq 1$, $0 \leq \text{Value} \leq 1$

OpenCV (uint8): $0 \leq \text{Hue} \leq 180$, $0 \leq \text{Saturation} \leq 255$, $0 \leq \text{Value} \leq 255$

References

Reference manuals for OpenCV and NumPy are in:

- https://docs.opencv.org/
- http://www.numpy.org/

- R. Szeliski: Computer Vision: Algorithms and Applications, Springer, 2010. (コンピュータビジョン, アルゴリズムと応用, 共立出版, 2013)
- A. Kaehler, G. Bradski: Learning OpenCV 3, O’Reilly, 2017. (詳解 OpenCV 3, オライリー・ジャパン, 2018)
- ディジタル画像処理編集委員会, ディジタル画像処理, CG-ARTS協会, 2015.