Intelligent Control Systems

Image Processing (2)
— Filtering and Geometric Transforms —

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/

Sample codes for this week

- Open https://github.com/shingo-kagami/ic.git
- Click the green button “Code” and click “Download Zip”
- Copy the files whose names start from ic02*** to C:¥ic2022¥sample

If you are a Git user, you may simply run:

```
cd C:¥ic2022¥sample
git pull
```

Taxonomy of Image Processing

input	output	example
image	image (2-D data)	color conversion, edge detection, smoothing, coordinate transforms, …
	feature values (1-D vector, scalar values, etc.)	histogram, position, object label, …

Image to Image

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

point operation

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

(thresholding, pixel value conversion, color 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

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

point operation

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

local operation / neighboring operation

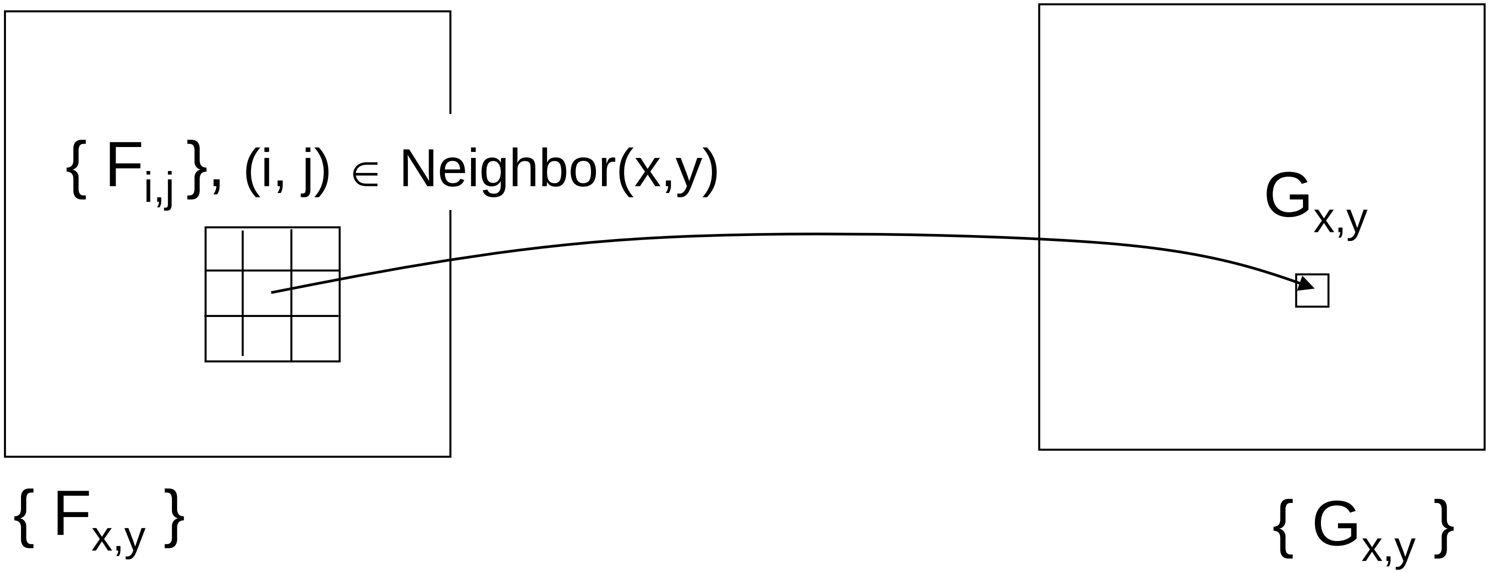
$G_{i,j}$ depends on pixels within some neighborhood of $F_{i,j}$

global operation

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

Local operation example: Spatial Filter

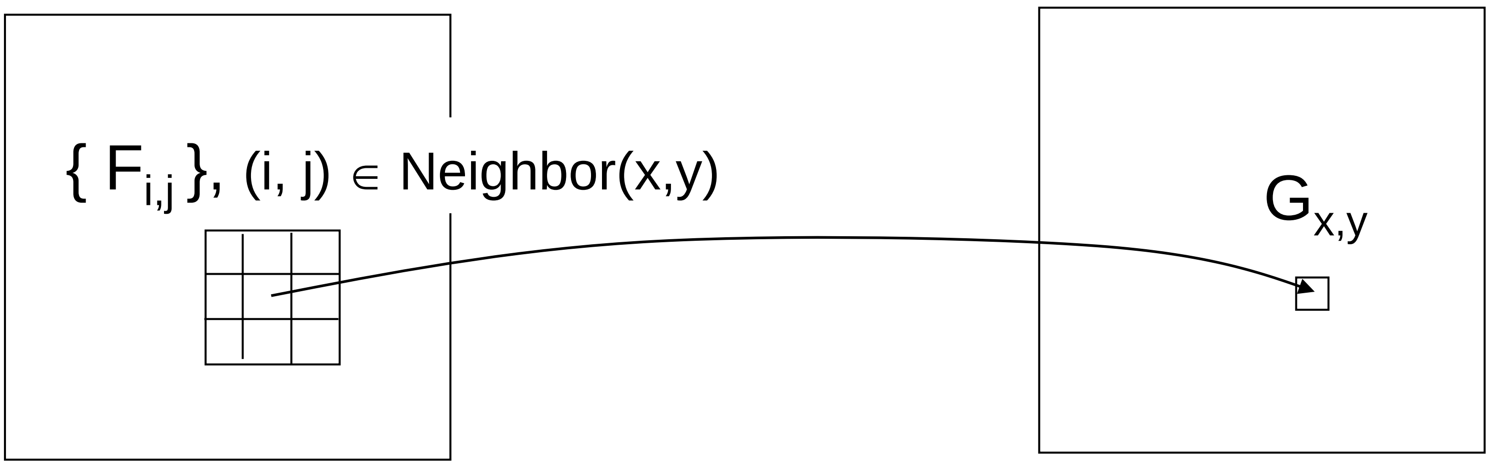
$G_{x,y}$ depends on some neighborhood (e.g. 3×3, 5×5 pixels, etc.) of the point of interest (x,y)



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

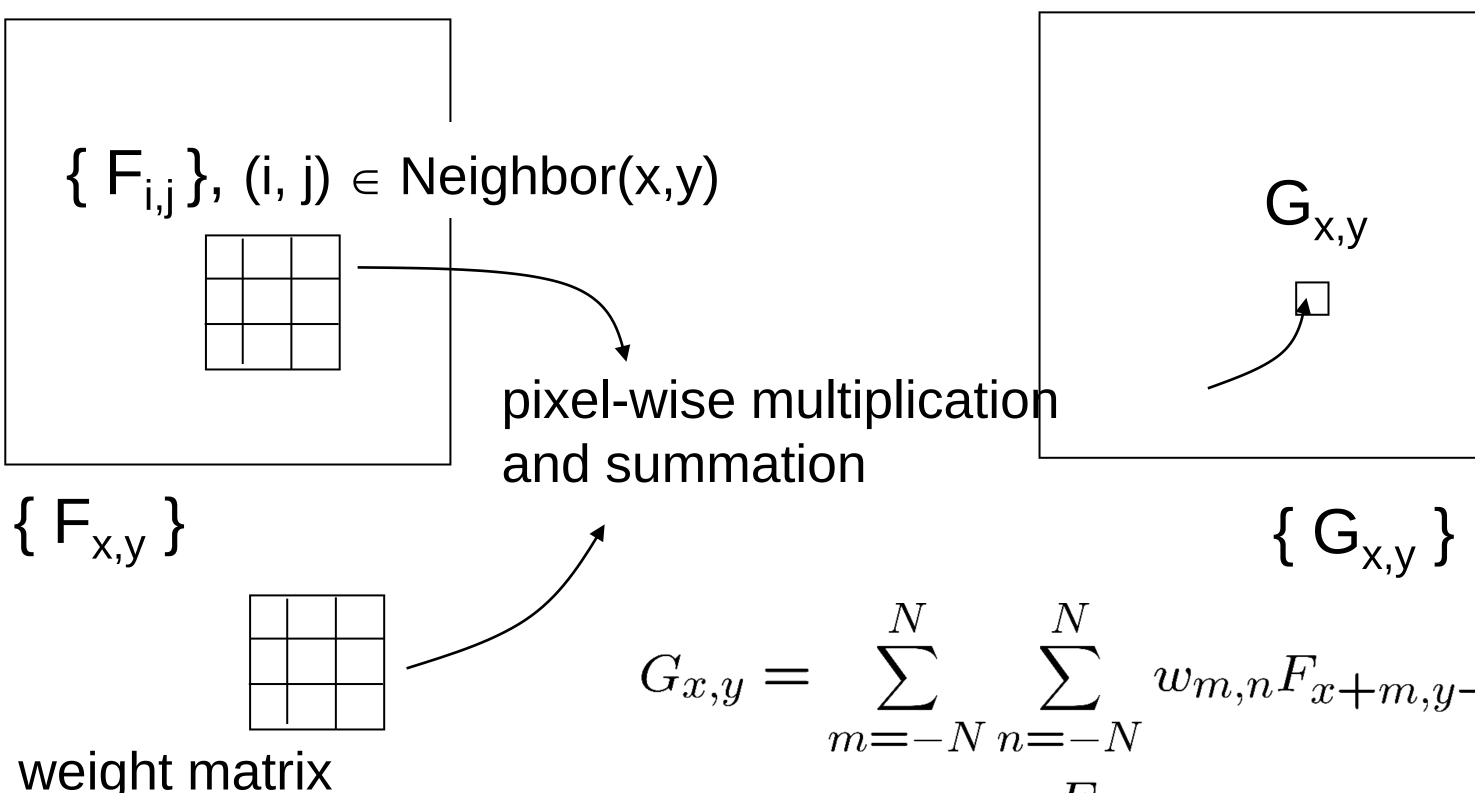
(mean)

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 (possibly 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

- When computational cost matters (and historically it often mattered), integer computation is favored. In such cases, care must be taken for overflow of values

Implementation of 3x3 filtering

ic02_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] = 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

Generic function for linear spatial filter

```
cv2.filter2D()
```

Dedicated functions for well-known filters

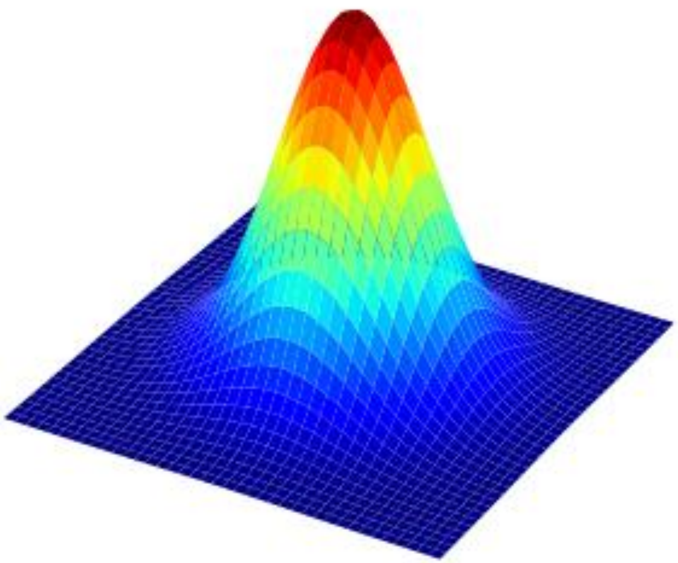
```
cv2.GaussianBlur()
cv2.Sobel()
cv2.Laplacian()
…
```

Nonlinear filters

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

Gaussian: most widely used smoothing kernel

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



- Appropriate for smoothing both in theory and in practice in many aspects. see e.g. [Lindeberg 1994] [Florack 1992]
- Discretized in space for digital computation
- Weight values are sometimes rounded to integer (favoring computational cost)
- Amount of smoothing can be controlled by parameter σ (Note that large σ requires large matrix size)

Frequency-domain understanding

$$G_{x,y} = \sum_{m=-N}^{N} \sum_{n=-N}^{N} w_{m,n} F_{x+m,y+n}$$
$$= w_{-x,-y} * F_{x,y} \xrightarrow{\mathcal{F}} \mathcal{F}\left[w_{-x,-y}\right] \cdot \mathcal{F}\left[F_{x,y}\right]$$

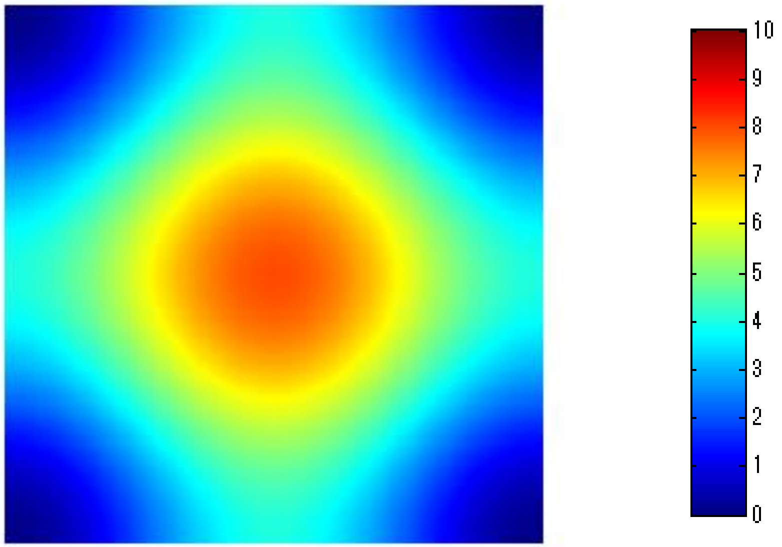
$\mathcal{F}[\cdot]$: 2-D discrete Fourier transform

ic02_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 gradient
- (approximated by finite difference)

$$\frac{\partial}{\partial x} F_{x,y} \simeq \frac{F_{x+1,y} - F_{x-1,y}}{2}$$

$$\frac{1}{2}\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

in x direction

$$\frac{1}{2}\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

in y direction

- Often combined with smoothing

$$\frac{1}{2}\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

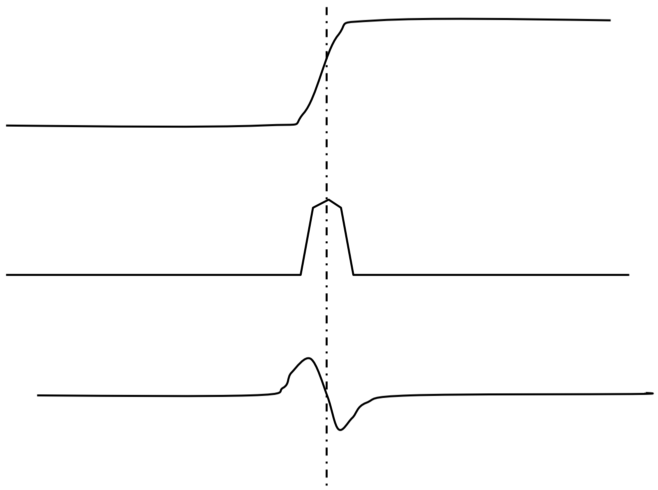
Sobel filter in x direction

$$\frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

in y direction

Edge detection by 2nd order derivative

- Edge = zero crossing of 2nd order derivative
- Laplacian $\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is the lowest-order isotropic differential operator (i.e. equally responds to edges in any direction)



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

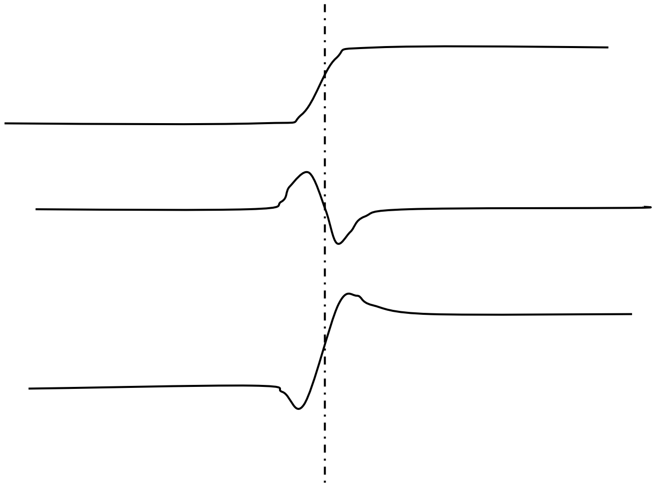
$$\frac{\partial^2}{\partial x^2} F_{x,y} \simeq \frac{\frac{F_{x+1,y} - F_{x,y}}{1} - \frac{F_{x,y} - F_{x-1,y}}{1}}{1}$$

$$= F_{x+1,y} - 2F_{x,y} + F_{x-1,y}$$

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

=

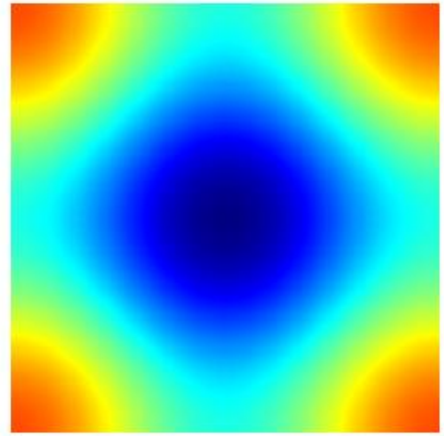
0	-1	0
-1	5	-1
0	-1	0

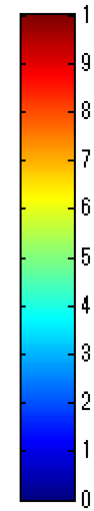
Frequency-domain visualization

Laplacian:

0	1	0
1	-4	1
0	1	0

$$\xrightarrow{\mathcal{F}}$$

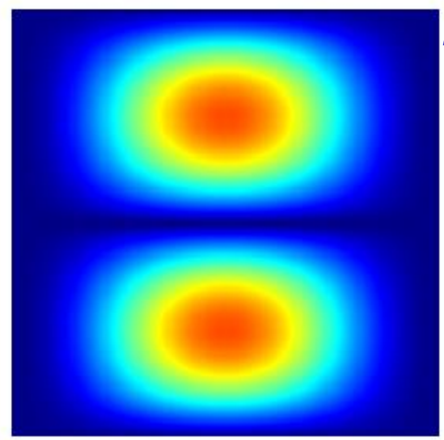




Sobel:

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

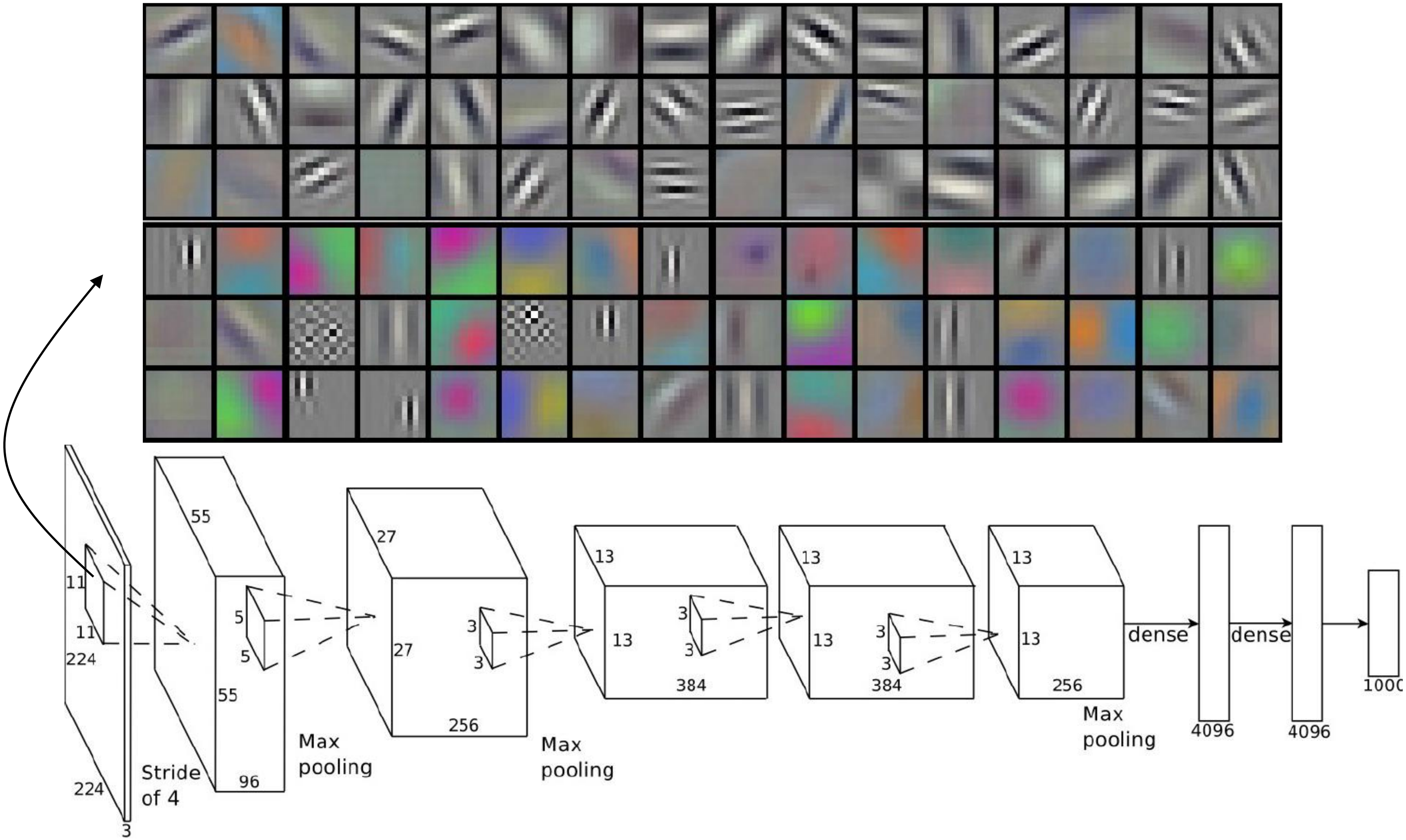
$$\xrightarrow{\mathcal{F}}$$



DC in y direction

highest frequency
in y direction

Deep Convolutional Neural Networks



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

Image to Image

image $\{ F_{x,y} \}$ $\longrightarrow$ 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 $F_{i,j}$

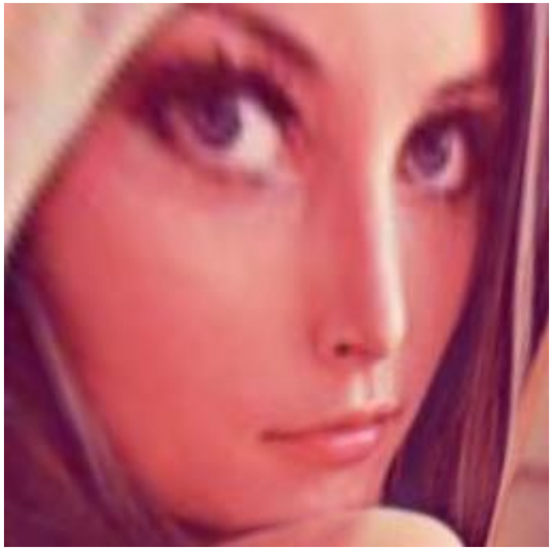
global operation

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

Global operation example: Warping

ic02_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

Translation

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

Rigid Transform

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

Similarity Transform

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

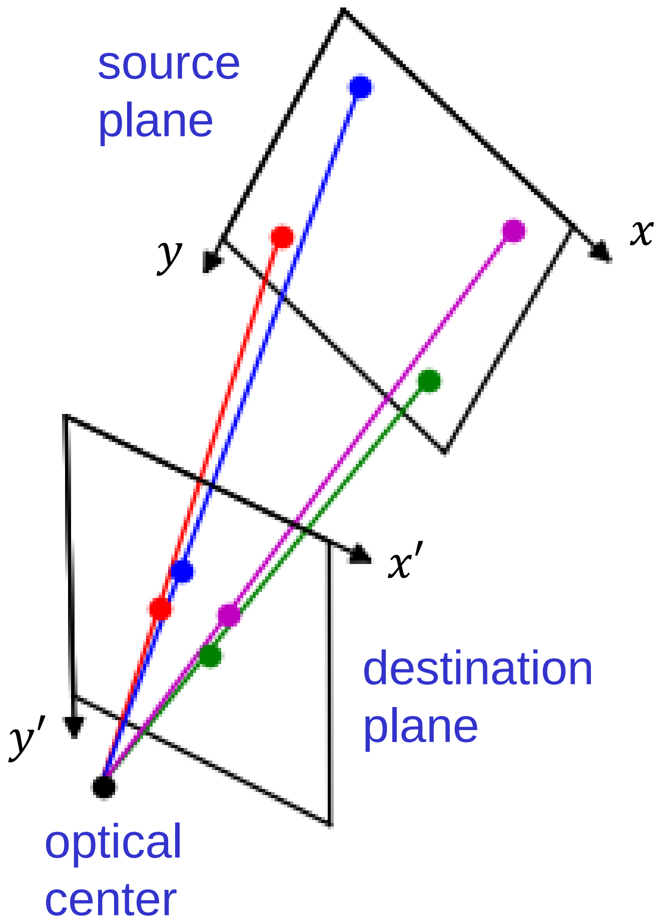
Affine Transform

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

Homography Transform
(Projective Transform, Perspective Transform, Collineation)

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

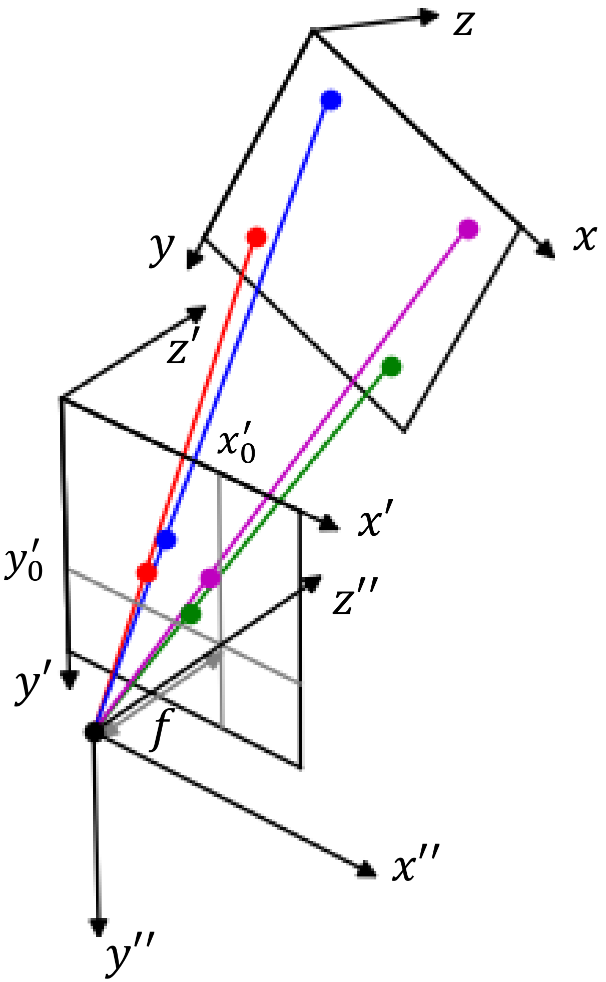
Homography Transform



$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \propto \underbrace{\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}}_{H} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Represents perspective mapping from a plane to a plane in 3D space
- Note the proportionality sign (instead of equality sign); the matrix-vector product on the right-hand side must be normalized such that 3rd entry becomes 1
- Therefore, *H* multiplied by an arbitrary scale factor gives the same mapping as *H*
- Because it has 8 degrees of freedom, *H* is computed when n (n ≥ 4) corresponding points are given

Understanding Homography



(x, y)

$\downarrow$

(x'', y'', z'')

$\downarrow$

(x', y')

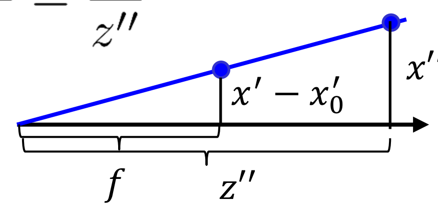
rotation　　　　translation

$$\begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \cancel{\boldsymbol{r}_3} \end{pmatrix} \begin{pmatrix} x \\ y \\ \cancel{0} \end{pmatrix} + \boldsymbol{t}$$

$$= \begin{pmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

3x3 matrix

$$\frac{x' - x'_0}{f} = \frac{x''}{z''}, \quad \frac{y' - y'_0}{f} = \frac{y''}{z''}$$



or equivalently,

$$z'' \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & x'_0 \\ 0 & f & y'_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x'' \\ y'' \\ 1 \end{pmatrix} = \boldsymbol{H} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

x''-y''-z'': parallel to x'-y'-z' and has origin at optical center

Homography Warping by OpenCV

ic02_warp.py:

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

H = cv2.getPerspectiveTransform(src_pnts, dest_pnts)
result = cv2.warpPerspective(frame, H, dest_size)
```

4 points [x, y] ’s

4 points [x’, y’]’s

Implementation of Homography Warp

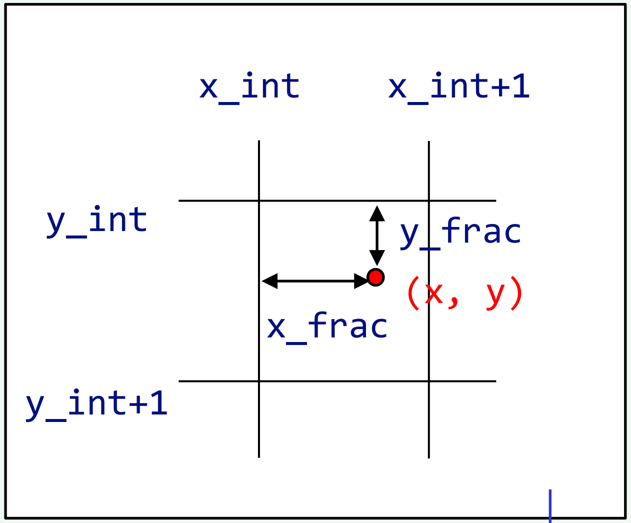
```
for j in range(height):
    x0 = Hinv[0, 1] * j + Hinv[0, 2]
    y0 = Hinv[1, 1] * j + Hinv[1, 2]
    w0 = Hinv[2, 1] * j + Hinv[2, 2]
    for i in range(width):
        x = Hinv[0, 0] * i + x0
        y = Hinv[1, 0] * i + y0
        w = Hinv[2, 0] * i + w0

        if w != 0.0:
            x, y = x / w, y / w
        else:
            x, y = 0.0, 0.0
        x = min(swidth - 1.0, max(0.0, x))
        y = min(sheight - 1.0, max(0.0, y))
        x_int, y_int = int(x), int(y)
        x_frac, y_frac = x - x_int, y - y_int

        weight = np.array([(1 - x_frac) * (1 - y_frac), x_frac * (1 - y_frac),
                           (1 - x_frac) * y_frac, x_frac * y_frac])
        dest[j, i] = (weight[0] * src[y_int, x_int]
                      + weight[1] * src[y_int, x_int + 1]
                      + weight[2] * src[y_int + 1, x_int]
                      + weight[3] * src[y_int + 1, x_int + 1])
```

$x' = H x$

normalized by 3rd entry

boundary check

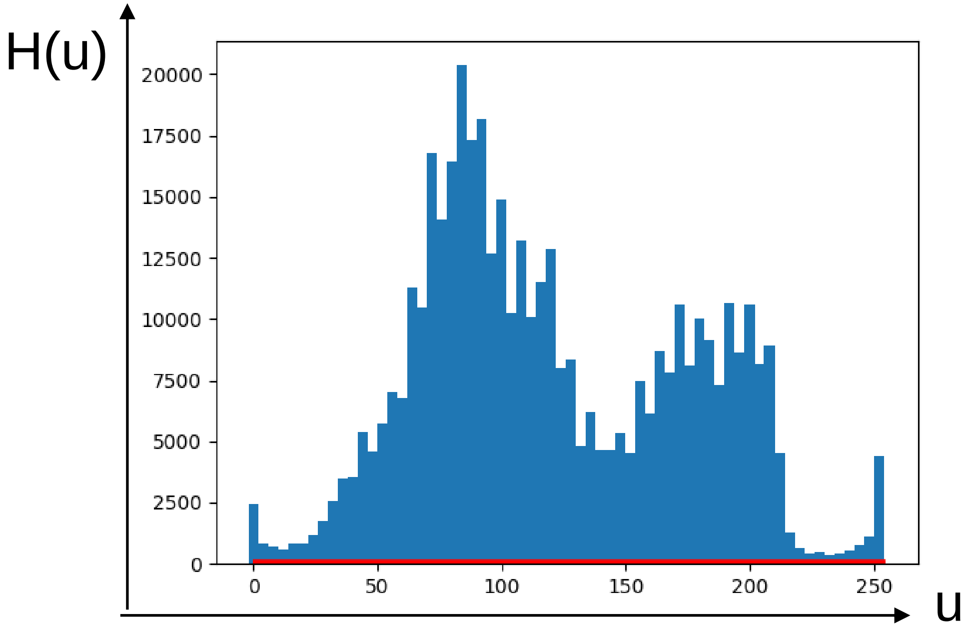
integer and fractional parts



Taxonomy of Image Processing

input	output	example
image	image (2-D data)	color conversion, edge detection, smoothing, coordinate transforms, …
	feature values (1-D vector, scalar values, etc.)	histogram, position, object label, …

Histogram of Pixel Values



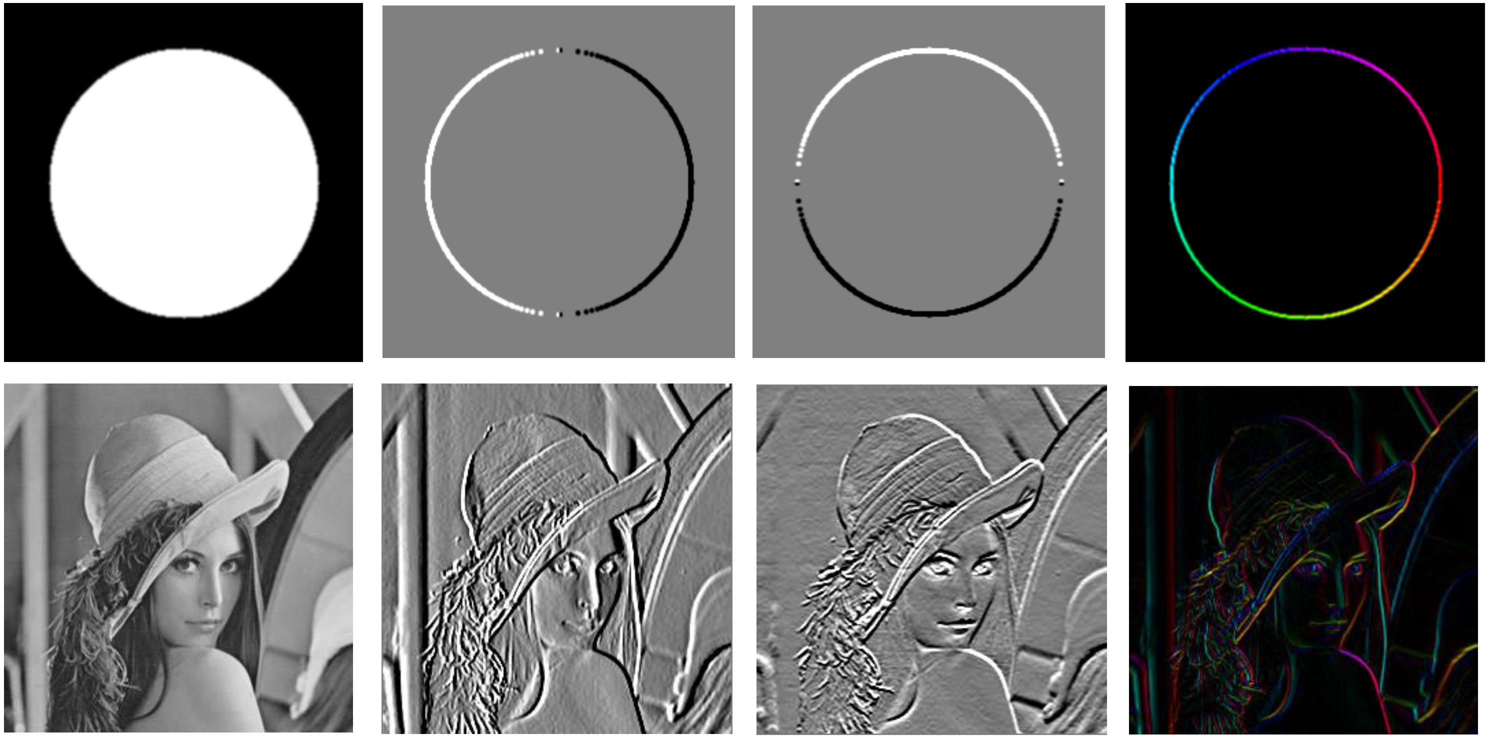
$$\boldsymbol{H} = \{H_u\}_{u=1,2,\cdots,m}, \quad H_u = \sum_{\boldsymbol{x} \in S(u)} 1$$

where $S(u)$ is a set of pixels having values belonging to the bin u

If you run ic02_histogram.py from Spyder environment, you may need to execute in IPython console to open the graph plot window:

```
%matplotlib auto
```

Histogram of Gradient Orientations



I I_x I_y

Hue $\propto \mathrm{atan2}(\mathrm{I_y}, \mathrm{I_x}) + \pi$

Sat = 1, Value $\propto \sqrt{I_x^2 + I_y^2}$

Histogram of orientations of pixel value gradients reflects the overall edge structure of the image

Implementation of Histogram Computation

ic02_histogram.py:

```
    hist_val = np.zeros(n_bins)

    for j in range(height):
        for i in range(width):
            at_bin = int(src[j, i] * n_bins / 256)
            hist_val[at_bin] += 1
```

source image (8-bit unsigned)

ic02_histogram_orient_grad.py:

```
    hist = np.zeros(n_bins, dtype=np.float32)

    for j in range(height):
        for i in range(width):
            m = mag[j, i]
            a = angle[j, i]
            if a == 2 * math.pi:
                ## prevent at_bin from getting out of [0, n_bins-1] range
                a = 0.0
            at_bin = int(a * n_bins / (2 * math.pi))
            hist[at_bin] += m
```

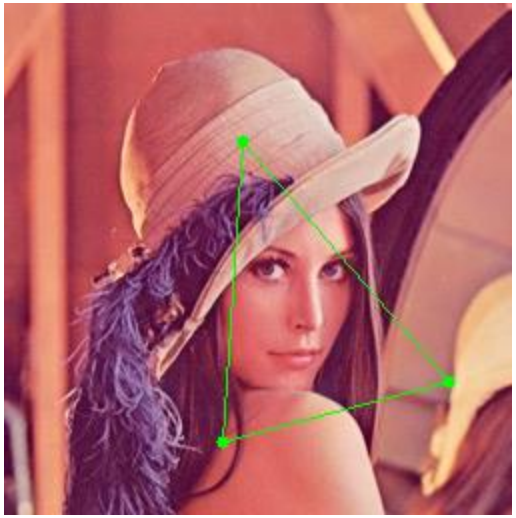
magnitude of gradient at pixel (i, j)

angle of gradient at pixel (i, j)

Exercises (Not Assignments)

Confirm that $\frac{1}{8}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$ is a rough approximation of 2D Gaussian function with $\sigma = 0.6$

Copy and modify ic02_warp.py to apply affine transform instead of homography transform using OpenCV functions cv2.getAffineTransform and cv2.warpAffine. Note that three pairs of corresponding points are needed (instead of four pairs needed in homography).



References

- 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)
- A. Hornberg eds.: Handbook of Machine Vision, Wiley-VCH, 2006.

- 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.

- T. Lindeberg: Scale-space theory: A basic to ol for analysing structures at different scales, J. Applied Statistics, vol. 21, no. 2, pp.225-270, 1994.
- L. M. J. Florack, B. M. T. Romeny, J. J. Koenderink and M. A. Viergever: Scale and the Differential Structure of Images, Image and Vision Computing, vol.10, no.6, pp.376-388, 1992.