Chapter

3

Image Processing

1. Spatial Filtering
2. Edge Detection
3. Histogram Equalization
4. OCR of Hand-written Data using kNN
5. Face Detection (Haar-cascade Detection)

Spatial Filtering

❖ 공간 영역 처리

- 영상의 한 픽셀과 이웃한 픽셀 값을 이용한 공간 영역 연산
- 입력 픽셀과 이웃한 각 픽셀에 가중치를 곱한 합을 출력 픽셀로 생성
- 공간 마스크 필터링(Spatial Mask Filtering)
  - 가중치 마스크(mask)를 원 영상에 회선 처리(Convolution Processing)

$$g[y][x] = \sum_{j=-w}^{w} \sum_{i=-w}^{w} m[j][i]\, f[y+j][x+i]$$

- g[y][x] : 회선 처리로 출력한 픽셀
- f[y+j][x+j] : 입력 영상의 픽셀
- m[j, i] : 가중치 마스크

- 마스크 (mask)
  - 이웃 픽셀의 가중치 값을 표현,
  - 윈도우(window), 템플릿(template), 커널(kernel), 필터(filter)

Spatial Filtering

❖ Convolution Processing

I_1	I_2	I_3
I_4	I_5	I_6
I_7	I_8	I_9

(a) 입력 영상

M_1	M_2	M_3
M_4	M_5	M_6
M_7	M_8	M_9

(b) 회선 마스크

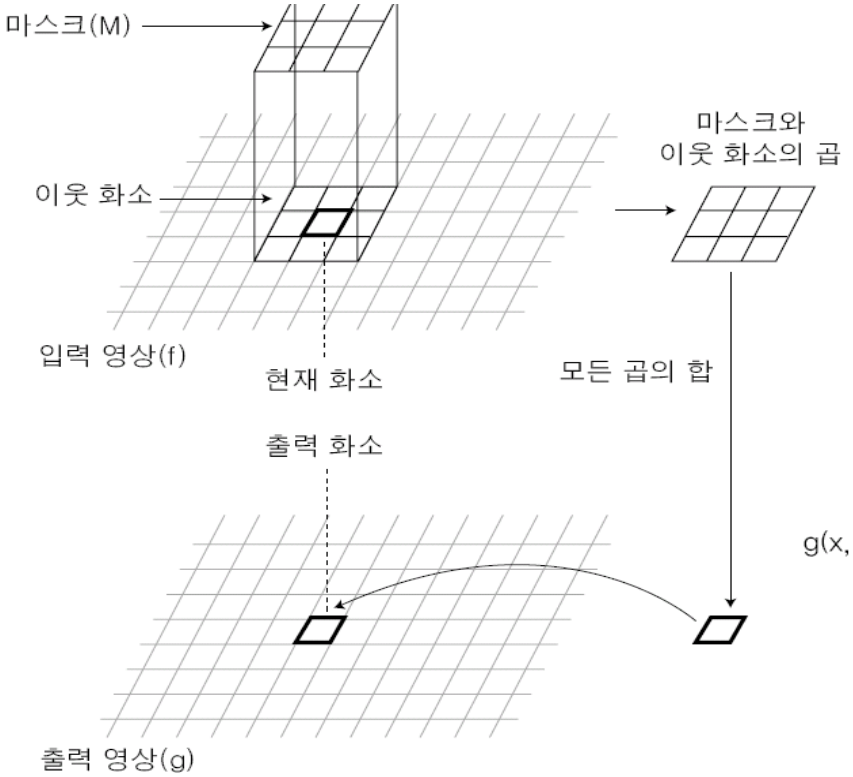
출력 픽셀 값 : $O_5 =$

$$I_1 \times M_1 + I_2 \times M_2 + I_3 \times M_3 + I_4 \times M_4 + I_5 \times M_5 + I_6 \times M_6 + I_7 \times M_7 + I_8 \times M_8 + I_9 \times M_9$$

회선 기법으로 출력 화소 생성

Spatial Filtering

❖ Convolution Processing



$$g(x,y)=M(0,0)\times f(x-1,y-1)+M(0,1)\times f(x-1,y)+M(0,2)\times f(x-1,y+1)+M(1,0)\times f(x,y-1)+M(1,1)\times f(x,y)+M(1,2)\times f(x,y+1)+M(2,0)\times f(x+1,y-1)+M(2,1)\times f(x+1,y)+M(2,2)\times f(x+1,y+1)$$

디지털 영상에서 회선을 처리하는 과정

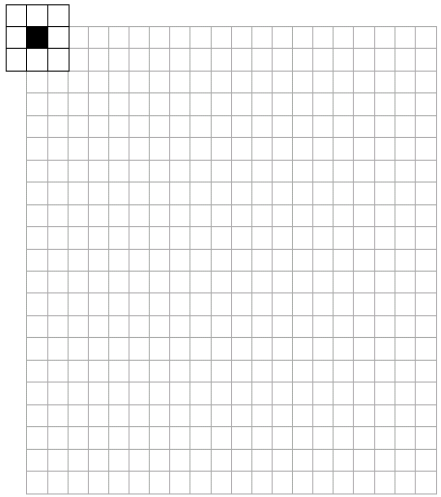
Spatial Filtering

❖ Convolution Processing 수행방법

(1)가중치를 포함한 마스크를 이동하면서 회선처리 수행

(2) 회선마스크가 영상의 왼쪽 위 픽셀에서 오른쪽으로 한 픽셀씩 차례로 이동하면서 처리하여 새로운 결과 픽셀을 생성

(3)한 줄에서의 회선 수행이 끝나면, 다음 줄로 이동하여 다시 한 픽셀씩 오른쪽으로 이동하면서 차례로 수행



회선 수행이 시작되는 위치

Spatial Filtering

❖ Convolution 선의 경계 부분 처리

- 회선처리에서는 이웃 픽셀이 필요한데 영상의 경계부분에는 이웃픽셀이 없기 때문에 경계 부분 처리가 필요
- 경계 부분 처리 방법

(1)경계를 0로 확장하여 회선 수행

- 회선 마스크에 대응되는 경계를 벗어나는 부분을 0으로 가정해서 회선을 수행하는 방법
- 원본 영상의 가장자리를 확장하여 0로 설정 후 처리결과에서 확장 부분을 제거하는 방법

(2)중첩 부분에서만 회선 수행

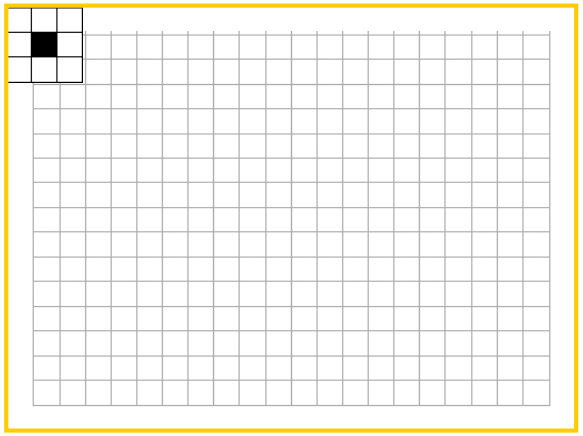
- 회선 마스크와 영상이 완전하게 중첩되는 위치에서 회선을 시작하도록 하는 방법
- 회선 마스크의 크기가 3×3이면 모든 회선 마스크의 요소와 영상의 화소가 중첩되는 영상 위치 (1, 1)에서 회선을 시작
- 경계부분은 회선처리 없이 입력 영상과 같은 픽셀 값을 복사해서 사용

(3)영상의 크기를 조정하여 회선 수행

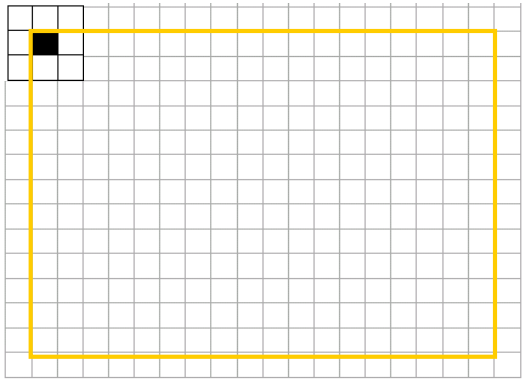
- 영상의 시작과 끝부분이 연결된 것으로 처리하는 방법

Spatial Filtering

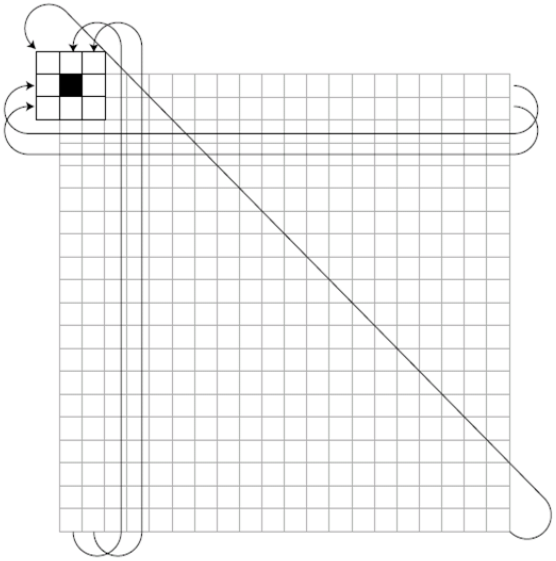
❖ Convolution의 경계 부분 처리 방법



(1)테두리를 확장 0로 확장한 처리 방법



(2)경계부분 픽셀을 제외한 처리 방법



(3)영상의 끝부분을 연결한 처리 방법

Spatial Filtering

❖ 공간 영역 처리 기술

- 연산 시 사용되는 가중치 마스크에 따라 결과영상의 특성이 결정
- 블러링(Blurring), 샤프닝(Sharpening), 경계선 검출(Edge Detection), 잡음 제거(Noise Removal) 등의 기술이 있음.
- Blurring
  - 영상을 흐리게(부드럽게) 하는 처리
- Sharpening
  - 영상을 세밀하게 하는 처리

Spatial Filtering

❖ Blurring

- 영상의 세밀한 부분을 흐리게 하거나 부드럽게 하여 잡음을 제거하는 기술
- 픽셀 값의 변화율을 낮출 수 있는 가중치 마스크 사용
- Mean Filter
  - 주변 픽셀 값들에 대한 평균값으로 출력 필셀 값을 계산
  - 마스크는 모든 계수가 양수로 전체 합은 1 로 구성

$\frac{1}{9}\times$

1	1	1
1	1	1
1	1	1

$\frac{1}{25}\times$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

평균값 필터 마스크 (3X3, 5X5)

Spatial Filtering

❖ Blurring 처리 예

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

평균값필터 마스크

90	90	90	90	90	90	90
90	90	90	90	90	90	90
90	90	255	255	255	90	90
90	90	255	255	255	90	90
90	90	255	255	255	90	90
90	90	90	90	90	90	90
90	90	90	90	90	90	90

(a)원본영상

90	90	90	90	90	90	90
90	108	127	145	127	108	90
90	127	163	200	163	127	90
90	145	200	255	200	145	90
90	127	163	200	163	127	90
90	108	127	145	127	108	90
90	90	90	90	90	90	90

(b)블러링영산

Spatial Filtering

❖ Blurring

입력 영상

3x3 mean filter 적용

5x5 mean filter 적용

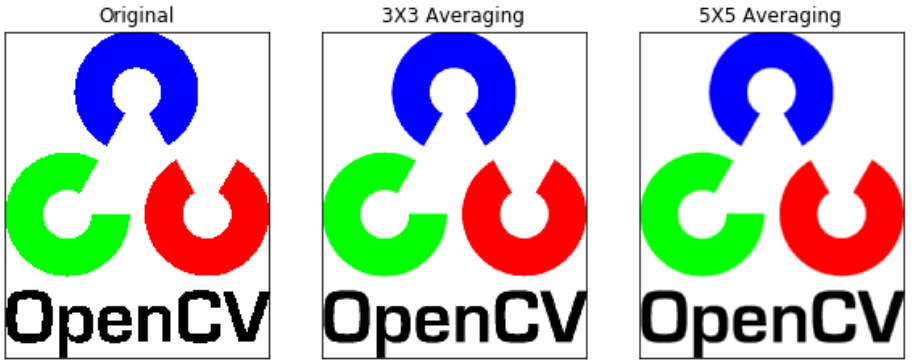
Spatial Filtering

❖ Blurring in OpenCV

- cv2.filter2D() 함수로 kernel(filter)을 이미지에 Convolution처리
- cv.blur()함수로 Blurring

```
img = cv2.imread('images/opencv-logo.png')
print("img.shape=", img.shape)
kernel1 = np.ones((3,3),np.float32)/9   #3X3 커널  생성
kernel2 = np.ones((5,5),np.float32)/25  #5X5 커널  생성

dst1 = cv2.filter2D(img,-1,kernel1)       #convolution
dst2 = cv2.filter2D(img,-1,kernel2)       #convolution

plt.figure(figsize=(10,8))
plt.subplot(131),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(dst1),plt.title('3X3 Averaging')
plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(dst2),plt.title('5X5 Averaging')
plt.xticks([]), plt.yticks([])
plt.show()
```

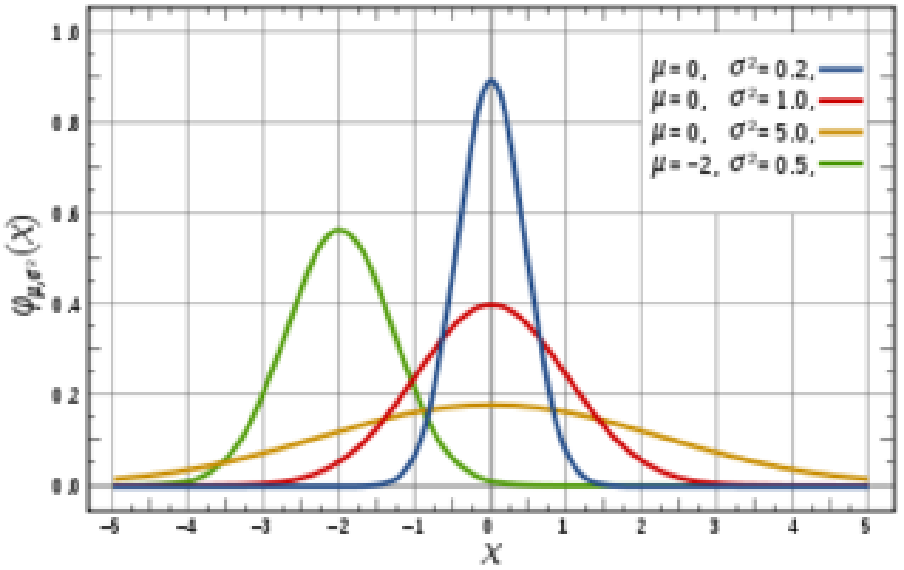
```
blur1 = cv.blur(img,(3,3))
blur2 = cv.blur(img,(5,5))
```



Spatial Filtering

❖ 가우시안(Gaussian) 필터링

- 자연현상을 가장 잘 표현한 함수
- 평균을 기준으로 좌우 대칭 형태
- 양끝으로 갈수록 수치가 낮아지는 종 모양
- 정규분포에서 평균(μ)을 0로 간주하고 표준편차(σ)로 유도한 분포의 함수식
- 가우시안 잡음을 제거하는 블러링 마스크로 이용
- 표준편차(σ)가 클수록 블러링 효과가 큼



Spatial Filtering

❖ 가우시안(Gaussian) 필터링

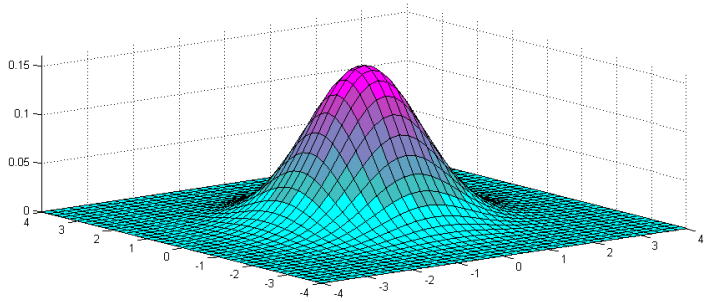
- 2차원 가우시안 필터 마스크(σ = 1.0)

-4 σ <= x, y <= 4 σ (mask size = 8 σ +1, 9X9)

0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0002	0.0011	0.0018	0.0011	0.0002	0.0000	0.0000
0.0000	0.0002	0.0029	0. 0131	0.0215	0.0131	0.0029	0.0002	0.0000
0.0000	0.0011	0.0131	0.0585	0.0965	0.0585	0.0131	0.0011	0.0000
0.0001	0.0018	0.0215	0.0965	0.1592	0.0965	0.0215	0.0018	0.0001
0.0000	0.0011	0.0131	0.0585	0.0965	0.0585	0.0131	0.0011	0.0000
0.0000	0.0002	0.0029	0. 0131	0.0215	0.0131	0.0029	0.0002	0.0000
0.0000	0.0000	0.0002	0.0011	0.0018	0.0011	0.0002	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0001	0.0000	0.0000	0.0000	0.0000

- 2차원 가우시안 함수

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



평균=(0,0) , 표준편차(σ) = 1.0

Spatial Filtering

❖ Gaussian blurring in OpenCV

- cv2.GaussianBlur ()
- sigmaX값에 따른 출력 결과 확인

```
img = cv2.imread('images/noise1.bmp')
dst1 = cv.GaussianBlur(img,(5,5),0) #sigma=0
dst1 = cv.GaussianBlur(img,(5,5),1) #sigma=1
dst3 = cv.GaussianBlur(img,(5,5),5) #sigma=5

plt.imshow(img)
plt.title('Original')
plt.figure(figsize=(14,8))
plt.subplot(131),plt.imshow(dst1),plt.title('GaussianKerne sigma0')
plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(dst2),plt.title('GaussianKerne sigma1')
plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(dst3),plt.title('GaussianBlur sigma5')
plt.xticks([]), plt.yticks([])
```

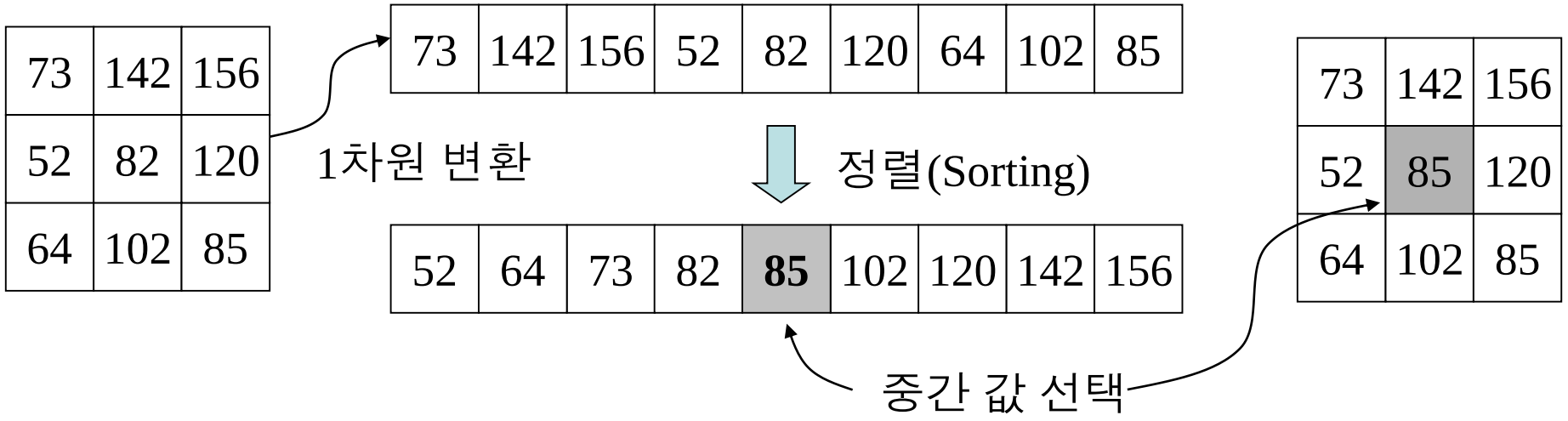
QUIZ!

1. ‘noise1.bmp’ 이미지에 대해 5X5 mean filtering결과와 Gaussian filtering (sigma 5)결과를 비교하세요.

Spatial Filtering

❖ 미디언(Median) 필터

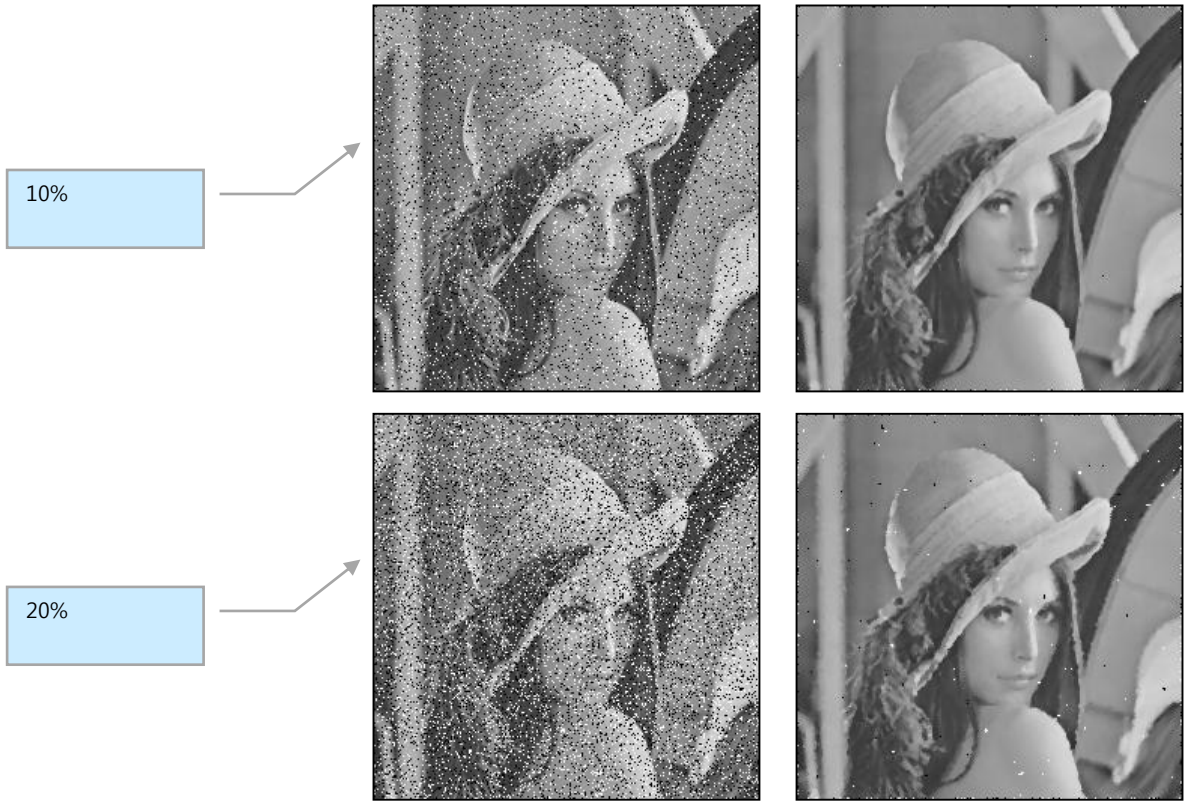
- 잡음 제거를 위한 비선형 필터
- 입력 영상의 (x,y) 좌표 주변 픽셀들의 값들을 오름 또는 내림 차순으로 정렬하여 그 중앙에 있는 픽셀 값을 사용



Spatial Filtering

❖ 미디언(Median) 필터

- 소금&후추(salt & pepper noise) 잡음이 포함된 영상에 미디언 필터를 적용한 결과



Spatial Filtering

❖ Median Blurring in OpenCV

- cv.medianBlur () 함수 이용

```
img1 = cv2.imread('images/lenna_SPN10.bmp')
img2 = cv2.imread('images/lenna_SPN20.bmp')
dst1 = cv2.medianBlur(img1, 3)
dst2 = cv2.medianBlur(img2, 3)

plt.figure(figsize=(8,8))
plt.subplot(221),plt.imshow(img1),plt.title('Original N10')
plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(dst1),plt.title('Median filtering N10')
plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.imshow(img2),plt.title('Original N20')
plt.xticks([]), plt.yticks([])
plt.subplot(224),plt.imshow(dst2),plt.title('Median filtering N20')
plt.xticks([]), plt.yticks([])
```

Spatial Filtering

❖ Sharpening

- 영상의 경계부분을 강조하여 영상을 선명하게 하는 영상처리 기법
- 픽셀 값의 변화율을 높일 수 있는 가중치 마스크 사용

−1	−1	−1
−1	9	−1
−1	−1	−1

(a) 샤프닝 회선 마스크 1

(8방향 마스크)

0	−1	0
−1	5	−1
0	−1	0

(b) 샤프닝 회선 마스크 2

(4방향 마스크)

Spatial Filtering

❖ Sharpening 처리 예

0	−1	0
−1	5	−1
0	−1	0

10	10	10	10	10	10	10
10	10	10	10	10	10	10
10	10	50	50	50	10	10
10	10	50	50	50	10	10
10	10	50	50	50	10	10
10	10	10	10	10	10	10
10	10	10	10	10	10	10

(a) 원본 영상

10	10	10	10	10	10	10
10	10	0	0	0	10	10
10	0	130	90	130	0	10
10	0	90	50	90	0	10
10	0	130	90	130	0	10
10	10	0	0	0	10	10
10	10	10	10	10	10	10

(b) 샤프닝 영상

Spatial Filtering

❖ Sharpening in OpenCV

```
img = cv2.imread('images/lenna_g.bmp')
kernel1 = np.full((3,3),-1, np.float32)   #3X3 8방향  Sharpening 마스크 생성
kernel1[1,1] = 9

imgB = cv.GaussianBlur(img,(5,5),5) #sigma=5

dst1 = cv2.filter2D(img,-1,kernel1)       #convolution
dst2 = cv2.filter2D(imgB,-1,kernel1)       #convolution

plt.figure(figsize=(10,8))
plt.subplot(221),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(dst1),plt.title('Sharpening')
plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.imshow(imgB),plt.title('Blur')
plt.xticks([]), plt.yticks([])
plt.subplot(224),plt.imshow(dst2),plt.title('Sharpening')
plt.xticks([]), plt.yticks([])
plt.show()
```

Edge Detection

❖ 그래디언트(Gradients)

- 2차원 공간에서 정의된 함수 f(x, y)가 있을 때 이 함수의 x축 방향 미분과 y축 방향 미분을 한꺼번에 벡터로 표현한 것

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = f_x \mathbf{i} + f_y \mathbf{j}$$

- 그래디언트는 벡터이므로 크기(magnitude)와 방향(phase) 성분으로 표현

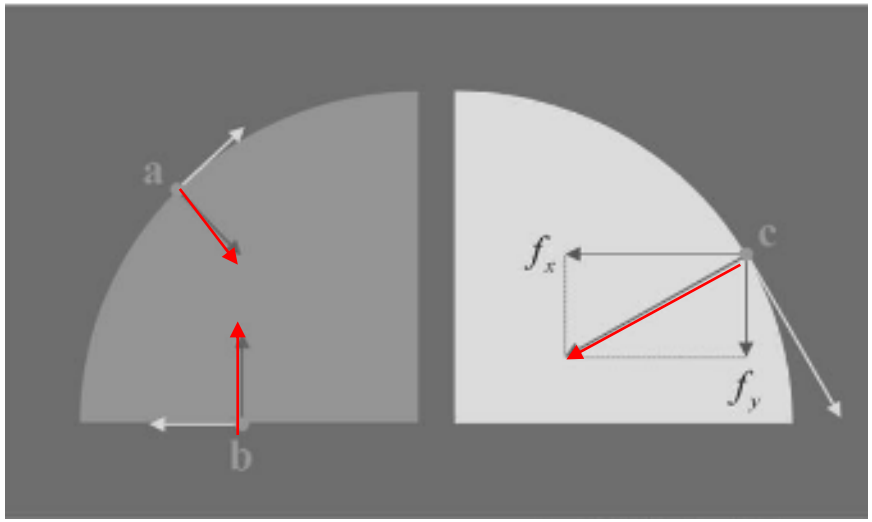
$$\|\nabla f\| = \sqrt{f_x^2 + f_y^2} \qquad \theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$$

  - 그래디언트 방향: 변화 정도가 가장 큰 방향
  - 그래디언트 크기 : 변화율 세기

Edge Detection

❖ 그래디언트(Gradients)

- a, b, c 세 점에서의 그래디언트
- 빨간색 화살표: 그래디언트 방향과 크기
- 흰색 화살표 : 그래디언트 방향에 수직, 에지 방향
- 에지 : 그래디언트 크기가 임계값보다 큰 경우



https://thebook.io/006939/ch09/01/01-06/

Edge Detection

❖ Sobel mask filtering

- X축 미분은 수평선을 미분하여 수직성분만 남음
- Y축 미분은 수직선을 미분하여 수평성분만 남음

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

(a)

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

(b)

(a) x축 방향으로의 편미분을 구하는 소벨 마스크

(b) y축 방향으로의 편미분을 구하는 소벨 마스크

Edge Detection

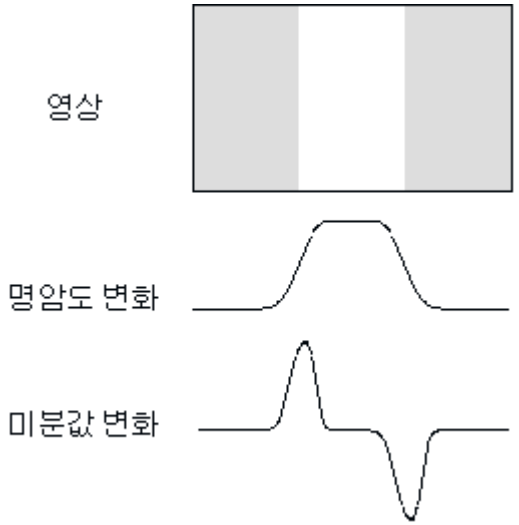
❖ Laplacian

- 영상의 2차 미분을 이용하여 경계부분을 추출
- X축, y축에 대한 2차 미분의 합으로 계산

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$= \left[f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) \right]$$

$$- 4 f(x, y)$$



0	1	0
1	-4	1
0	1	0

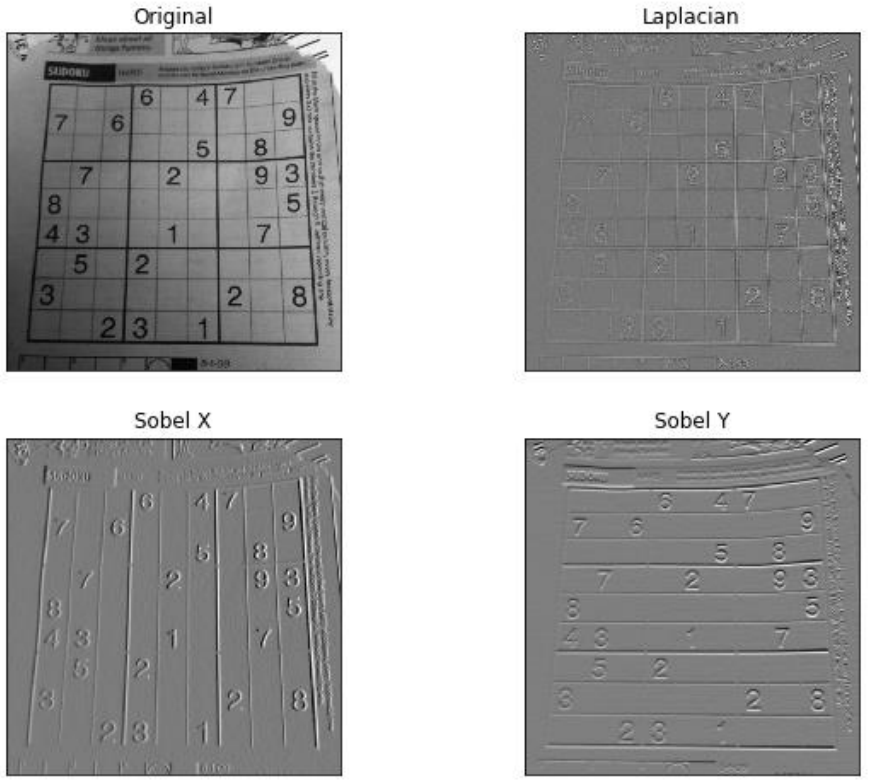
1	1	1
1	-8	1
1	1	1

4방향, 8방향 라플라시안 필터 마스크

Edge Detection

❖ Edge Detection in OpenCV

```
img = cv2.imread('images/sudoku.png', 0)
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  #  x방향 편미분, 수직성분 추출
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)  # y반향 편미분, 수평성분 추출

plt.figure(figsize=(10,8))
plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```



Edge Detection

❖ Canny Edge Detection

(1)Noise Reduction

- 이미지의 Noise를 제거, 5x5의 Gaussian filter 사용

(2)Edge Gradient Detection

- 이미지에서 Gradient의 방향과 강도를 계산
- 경계부분에서 미분값(밝기의 변화량)이 크므로 Edge 후보로 추출

(3)Non-maximum Suppression

- 이미지의 pixel을 모두 scan하여 Edge가 아닌 pixel은 제거.

(4)Hysteresis Thresholding

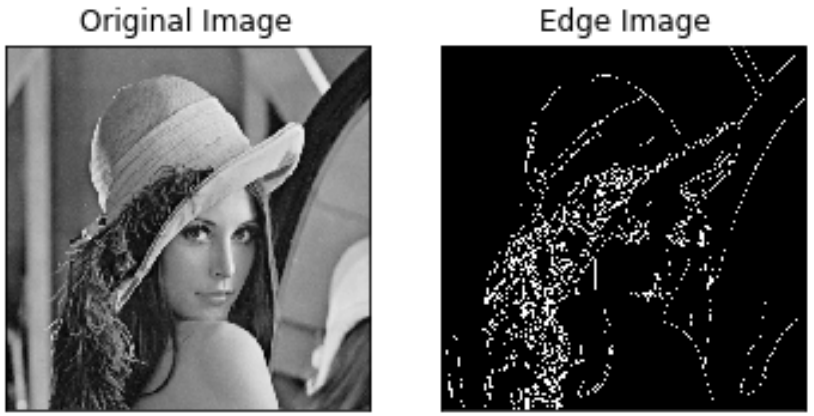
- Edge 후보들이 진짜 edge인지 판별
- 임계값을 max, min을 설정하여 max 이상은 강한 Edge, min과 max사이는 약한 edge로 설정
- 약한 edge는 강한edge와 연결이 되어 있으면 edge로 판단하고, 아니면 제거

Edge Detection

❖ Canny Edge Detection in OpenCV

- cv2.Canny(image, threshold_min , threshold_max)

```
img = cv2.imread('images/lenna.bmp',0)
edges = cv2.Canny(img,80,240)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```



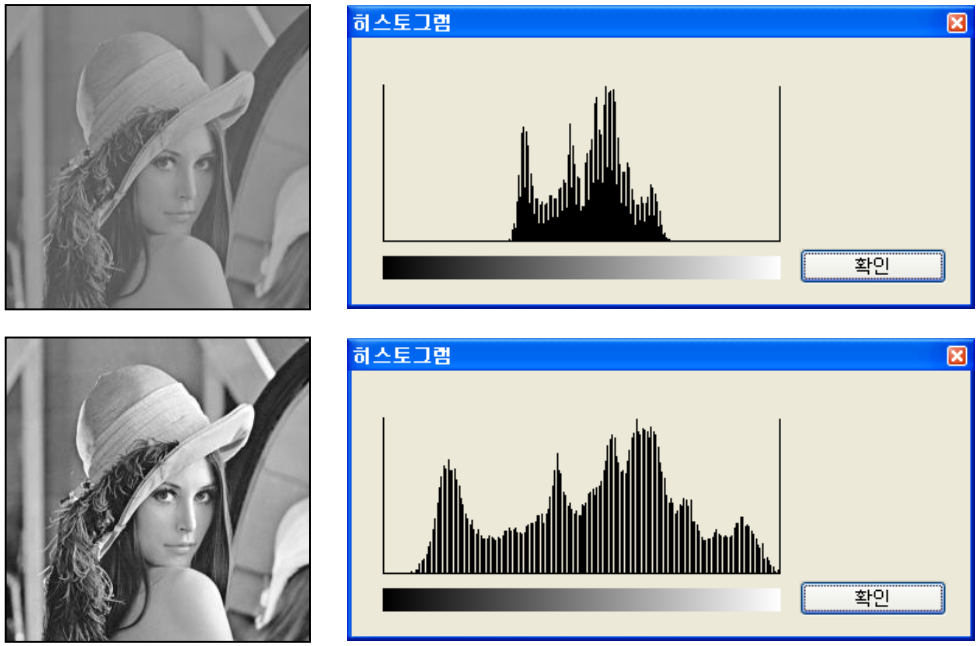
QUIZ!

1. ‘lenna.bmp’를 가우시안 블러링을 처리 한 후 Sobel, Laplacian , Canny Edge Detection 처리 결과를 확인하세요.

Histogram Equalization

❖ 히스토그램 균일화, 평활화(Histogram Equalization)

- 이미지의 contrast(대비)를 향상시키기 위해 히스토그램을 스트레칭

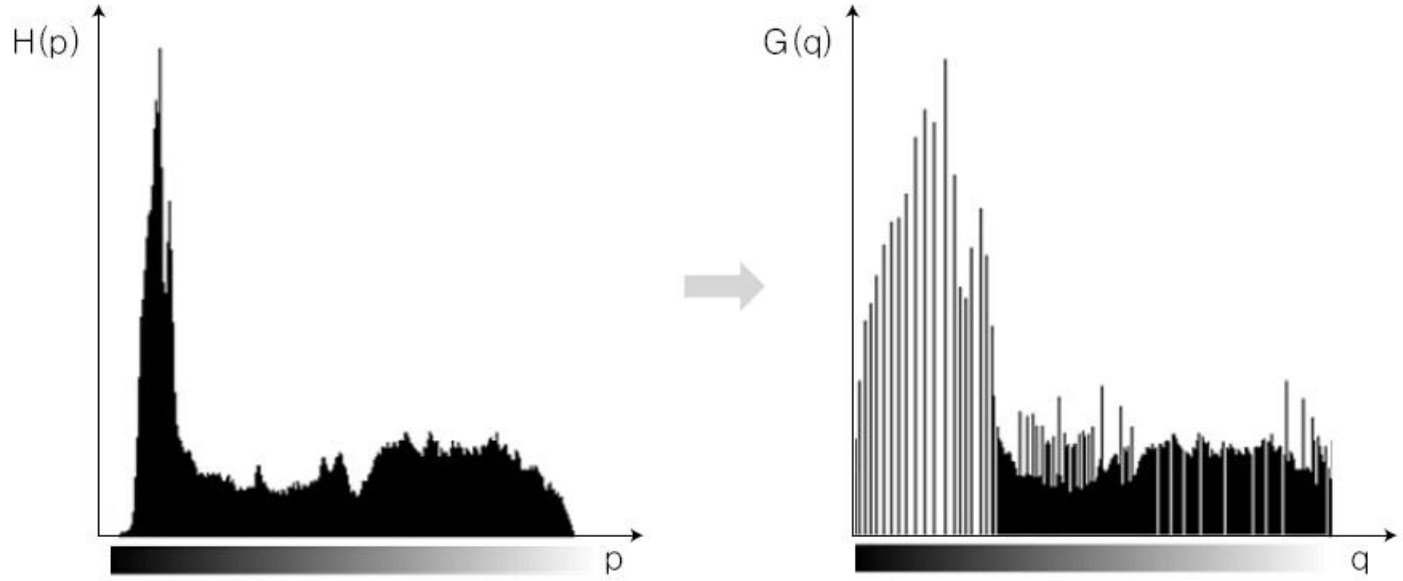


Histogram Equalization

❖ 히스토그램 평활화

- 영상의 밝기 분포를 재분배하여 명암 대비(Intensity Contrast)를 최대화하는 기법
- 히스토그램의 분포를 균등하게 변환
  - 영상의 히스토그램을 명암 값 전 구간에서 일정하게 분포되게 하는 방법
- 정규화된 누적분포함수 이용

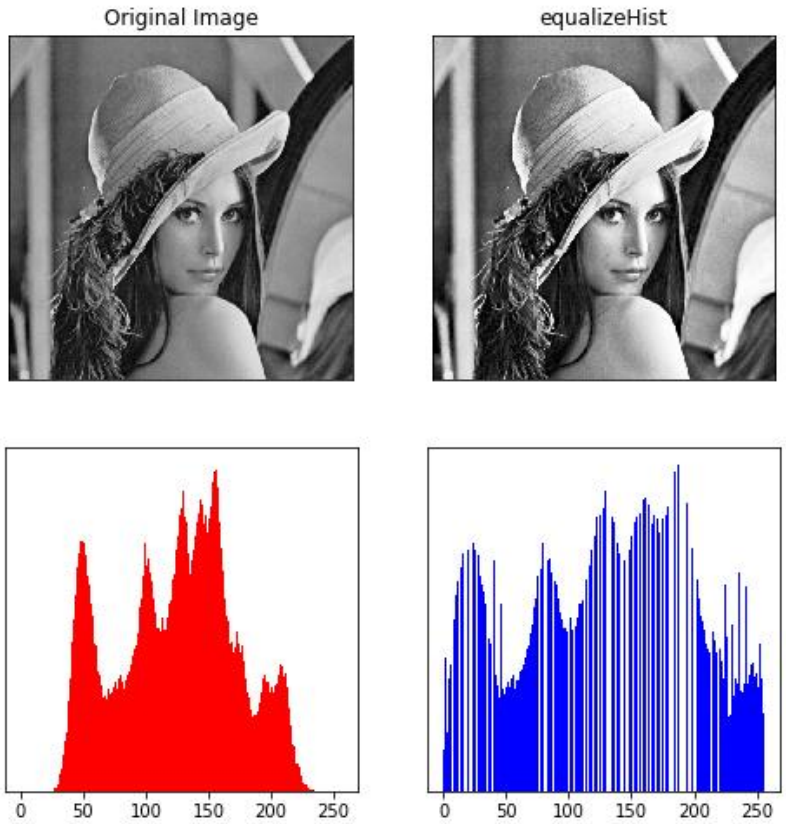
$$s = T(r) = \int_0^r p_r(\tau)\, d\tau$$



Histogram Equalization

- Histograms Equalization in OpenCV
  - equalizeHist()함수 이용

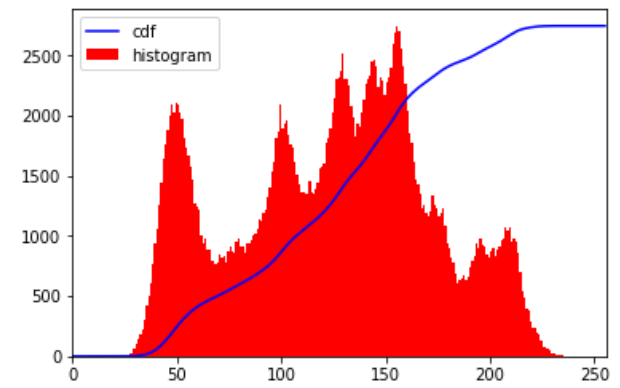
```
img = cv.imread('images/lenna.bmp',0)
equ = cv.equalizeHist(img)

plt.figure(figsize=(8,8))
plt.subplot(221),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(equ,cmap = 'gray')
plt.title('equalizeHist'), plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.yticks([])
plt.subplot(224),plt.hist(equ.flatten(),256,[0,256], color = 'b')
plt.yticks([])
```



Histogram Equalization

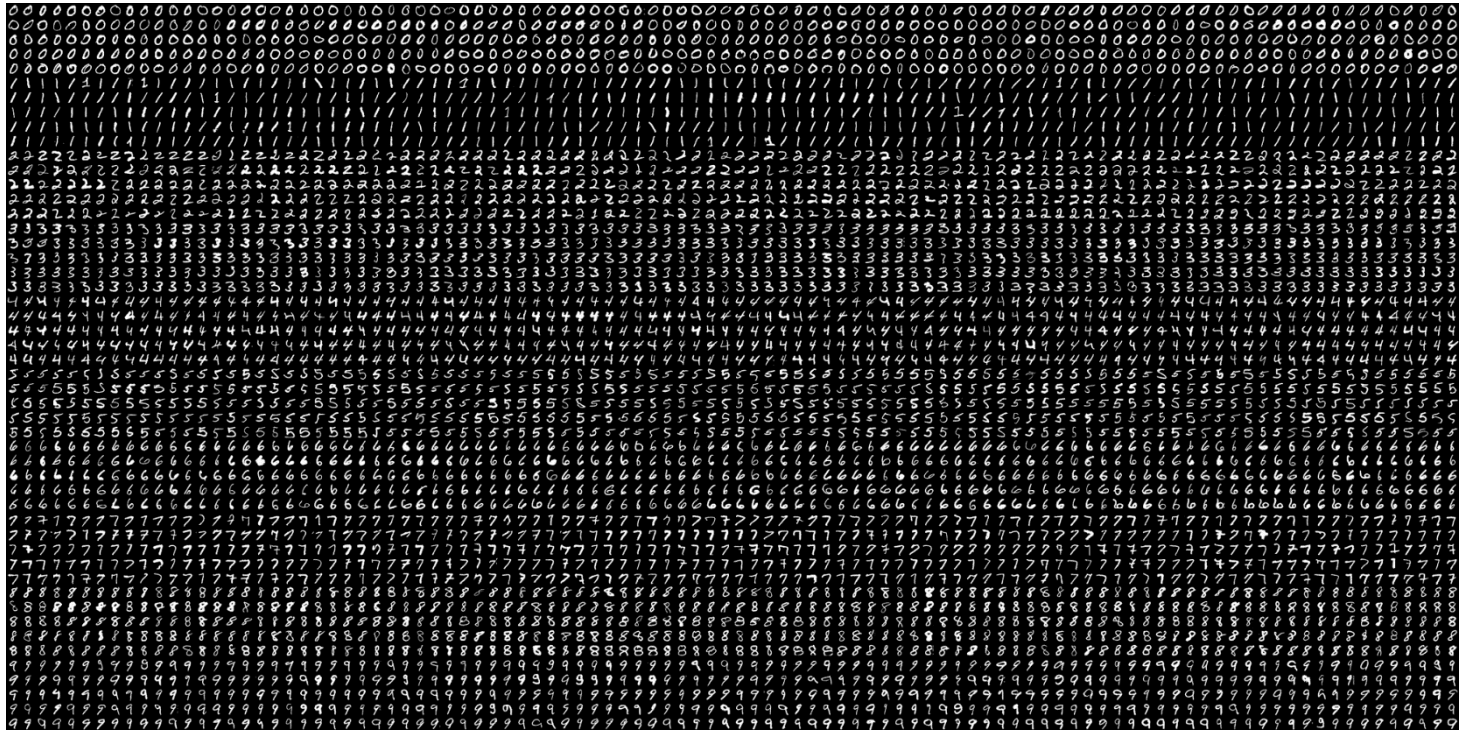
- Histograms Equalization in OpenCV
  - 알고리즘 구현

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('images/lenna.bmp',0)

#(1)히스토그램을 구한다
hist,bins = np.histogram(img.flatten(),256,[0,256])
#(2)정규화된 누적분포함수를 구한다.
cdf = hist.cumsum()   #누적합, 누적분포함수(CDF)
cdf_normalized = cdf * float(hist.max()) / cdf.max() #누적합에 최대밝기값을 곱하여 픽셀개수로 나누어 정규화

plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()

#(3)CDF 균일화 처리
cdf_m = np.ma.masked_equal(cdf,0)       #cdf에서 값이 0인 부분 제외
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min()) #균일화 처리
cdf = np.ma.filled(cdf_m,0).astype('uint8')  #cdf에서 마스크부분을 0으로 채우기

#(4)정규화된 cdf 값으로 영상의 픽셀 값을 변환
img2 = cdf[img]
```



Knn 필기체 손글씨 인식

- 5000 개의 필기 숫자 (각 숫자에 대해 500)가 있는 이미지 'digital.png' 제공



Knn 필기체 손글씨 인식

```
import numpy as np
import cv2 as cv

img = cv.imread('images/digits.png')
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
print('gray.shape=', gray.shape)

# 5000 셀을 100개씩 50줄로 분리
# 각 셀은 20x20
cells = [np.hsplit(row,100) for row in np.vsplit(gray,50)]

# Numpy array로 생성  (50,100,20,20)
x = np.array(cells)
print('x.shape=', x.shape)

#training data, test data 분리
train = x[:,:50].reshape(-1,400).astype(np.float32) # Size = (2500,400)
test = x[:,50:100].reshape(-1,400).astype(np.float32) # Size = (2500,400)

# train, test data의 라벨 생성
k = np.arange(10)
train_labels = np.repeat(k,250)[:,np.newaxis]  #0-9, 250번 반복하여 값을 저장
test_labels = train_labels.copy()
```

Knn 필기체 손글씨 인식

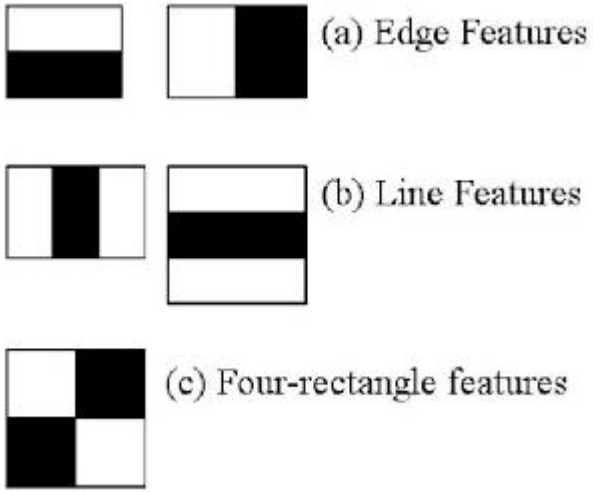
```
# kNN 초기화
knn = cv.ml.KNearest_create()
#training data로 학습
knn.train(train, cv.ml.ROW_SAMPLE, train_labels)
#5-NN으로 test와 가까운 이웃에 대한 라벨을 결정
ret,result,neighbours,dist = knn.findNearest(test,k=5)

# 분류 정확도 확인
matches = result==test_labels
correct = np.count_nonzero(matches)
accuracy = correct*100.0/result.size
print( accuracy )
```

```
gray.shape= (1000, 2000)
x.shape= (50, 100, 20, 20)
91.76
```

Face Detection

❖ Haar-cascade Detection in OpenCV

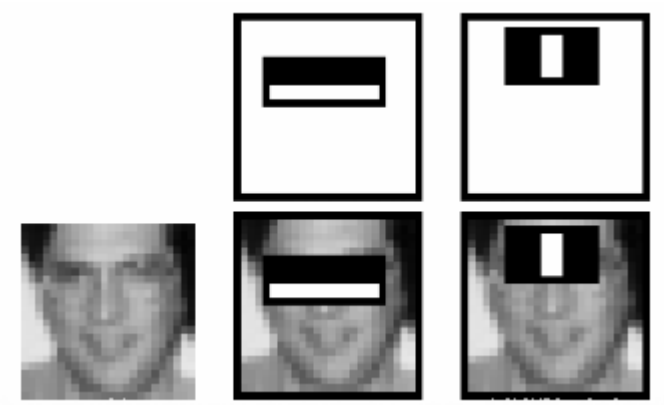
- 2001년에 비올라(P. Viola)와 존스(M. Jones)가 발표한 부스팅(boosting) 기반의 캐스케이드 분류기(cascade classifier) 알고리즘을 이용
  - 영상을 24×24 크기로 정규화한 후, 하르 필터(Haar-like filter) 집합으로부터 특징 정보를 추출하여 얼굴 여부를 판별
  - 하르 필터
    - 흑백 사각형이 서로 붙어 있는 형태로 구성된 필터
    - 24×24 영상에서 만들 수 있는 필터



Face Detection

❖ Haar-cascade Detection in OpenCV

- 사람의 정면 얼굴 형태가 전형적으로 밝은 영역(이마, 미간, 볼 등)과 어두운 영역(눈썹, 입술 등)이 정해져 있기 때문에 하르 필터로 구한 특징 값은 얼굴을 판별에 유용한 특징



- 캐스케이드 구조를 사용하여 얼굴이 아닌 영역은 필터링에서 제외시켜 처리속도를 향상

Face Detection

❖ Haar-cascade Detection in OpenCV

- 사전 학습된 분류기를 xml 파일로 제공

XML 파일 이름	검출 대상
haarcascade_frontalface_default.xml haarcascade_frontalface_alt.xml haarcascade_frontalface_alt2.xml haarcascade_frontalface_alt_tree.xml	정면 얼굴 검출
haarcascade_profileface.xml	측면 얼굴 검출
haarcascade_smile.xml	웃음 검출
haarcascade_eye.xml haarcascade_eye_tree_eyeglasses.xml haarcascade_lefteye_2splits.xml haarcascade_righteye_2splits.xml	눈 검출
haarcascade_frontalcatface.xml haarcascade_frontalcatface_extended.xml	고양이 얼굴 검출
haarcascade_fullbody.xml	사람의 전신 검출
haarcascade_upperbody.xml	사람의 상반신 검출
haarcascade_lowerbody.xml	사람의 하반신 검출
haarcascade_russian_plate_number.xml haarcascade_licence_plate_rus_16stages.xml	러시아 자동차 번호판 검출

https://thebook.io/006939/ch13/02-05/

Face Detection

❖ Haar-cascade Detection in OpenCV

```python
def detect_face(src):
    classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    if classifier.empty():
        print('XML load failed!')
        sys.exit()

    faces = classifier.detectMultiScale(src)

    for (x, y, w, h) in faces:
        cv2.rectangle(src, (x, y), (x + w, y + h), (255, 0, 255), 2)

    cv2.imshow('src', src)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

```python
def detect_eyes(src):

    face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    eye_classifier = cv2.CascadeClassifier('haarcascade_eye.xml')

    if face_classifier.empty() or eye_classifier.empty():
        print('XML load failed!')
        sys.exit()

    faces = face_classifier.detectMultiScale(src)

    for (x1, y1, w1, h1) in faces:
        cv2.rectangle(src, (x1, y1), (x1 + w1, y1 + h1), (255, 0, 255), 2)

        faceROI = src[y1:y1 + h1, x1:x1 + w1]
        eyes = eye_classifier.detectMultiScale(faceROI)

        for (x2, y2, w2, h2) in eyes:
            center = (int(x2 + w2 / 2), int(y2 + h2 / 2))
            cv2.circle(faceROI, center, int(w2 / 2), (255, 0, 0), 2, cv2.LINE_AA)

    cv2.imshow('src', src)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

```python
src = cv2.imread('images/bp.jpg')
if src is None:
        print('Image load failed!')
        sys.exit()
#얼굴 검출
detect_face(src)
#눈 검출
detect_eyes(src)
```

Face Detection

❖ Haar-cascade Detection in OpenCV

수고하셨습니다.