

영상처리 Report

Week2

영상의 입출력

영상 입출력 코드

- 지정한 컬러 영상 파일을 회색조(grayscale) 형태로 읽어서 화면에 출력
- bmp 및 jpg로 저장을 수행
 - .bmp는 미압축, 미가공 파일로 파일 크기가 큰 고품질 이미지 파일 포맷
 - .jpg는 손실 압축 방법으로 매우 높은 해상도의 이미지를 처리할 때 품질이 저하될 수 있음

```
In [2]: # 초기화
import cv2
from matplotlib import pyplot as plt
import numpy as np

def plt_show(title, path, is_gray=False):
    plt.axis('off')
    if is_gray:
        plt.imshow(path, cmap='gray')
    else:
        plt.imshow(path)
    plt.title(title)
    plt.show()
```

```
In [3]: img = cv2.imread('images/netero.jpg', cv2.IMREAD_GRAYSCALE)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt_show('Image', img, True)
cv2.imwrite('output/netero_1.bmp', img)
cv2.imwrite('output/netero_2.jpg', img)
```

Image



```
Out[3]: True
```

영상 불러오기 imread 함수

- 래스터 그래픽 영상 파일 형식들을 메모리로 읽어들임
 - 파일 시그니처(file signature)를 읽어 적절한 코덱을 결정하고, 압축 해제된 영상 데이터 구조에 필요한 메모리 할당과 같은 복잡한 작업들을 처리
 - 확장자가 아닌 파일 시그니처를 읽어 파일의 포맷을 분석
-
- 함수명 : `cv2.imread(fileName, flags)`
 - 매개변수
 - `fileName (str)` : 읽어들일 영상 파일의 경로(상대 경로 또는 절대 경로 이용)
 - `flags (int)` : 영상 파일을 읽을 때 옵션
 - `cv2.IMREAD_COLOR` : 컬러로 읽어 들이기 (3채널 BGR 영상) (1)
 - `cv2.IMREAD_GRAYSCALE` : 회색조로 읽어 들이기 (0)
 - `cv2.IMREAD_UNCHANGED` : alpha channel까지 포함하여 읽기 (-1)
 - `cv2.IMREAD_UNCHANGED` : alpha channel까지 포함하여 읽기 (-1)
 - `cv2.IMREAD_REDUCED_GRAYSCALE_2` : 1채널 ½크기 grayscale 영상
 - `cv2.IMREAD_REDUCED_GRAYSCALE_4` : 1채널 ¼크기 grayscale 영상
 - `cv2.IMREAD_REDUCED_GRAYSCALE_8` : 1채널 ⅛크기 grayscale 영상
 - `cv2.IMREAD_REDUCED_COLOR_2` : 3채널 ½크기 BGR영상
 - `cv2.IMREAD_REDUCED_COLOR_4` : 3채널 ¼크기 BGR영상

- `cv2.IMREAD_REDUCED_COLOR_8` : 3채널 1/8크기 BGR영상
- 리턴값 : image 객체 행렬 (numpy.ndarray)

```
In [13]: # cv2.IMREAD_COLOR: 컬러로 읽어 들이기 (3채널 BGR 영상) (1)
IMREAD_COLOR = cv2.imread('images/netero.jpg', cv2.IMREAD_COLOR)
IMREAD_COLOR = cv2.cvtColor(IMREAD_COLOR, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_COLOR', IMREAD_COLOR)

# cv2.IMREAD_GRAYSCALE: 회색조로 읽어 들이기 (0)
IMREAD_GRAYSCALE = cv2.imread('images/netero.jpg', cv2.IMREAD_GRAYSCALE)
IMREAD_GRAYSCALE = cv2.cvtColor(IMREAD_GRAYSCALE, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_GRAYSCALE', IMREAD_GRAYSCALE, True)

# cv2.IMREAD_UNCHANGED: alpha channel까지 포함하여 읽기 (-1)
IMREAD_UNCHANGED = cv2.imread('images/netero.jpg', cv2.IMREAD_UNCHANGED)
IMREAD_UNCHANGED = cv2.cvtColor(IMREAD_UNCHANGED, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_UNCHANGED', IMREAD_UNCHANGED)

# cv2.IMREAD_REDUCED_GRAYSCALE_2: 1채널 ½크기 grayscale 영상
IMREAD_REDUCED_GRAYSCALE_2 = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_GRAYSCALE_2)
IMREAD_REDUCED_GRAYSCALE_2 = cv2.cvtColor(IMREAD_REDUCED_GRAYSCALE_2, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_REDUCED_GRAYSCALE_2', IMREAD_REDUCED_GRAYSCALE_2, True)

# cv2.IMREAD_REDUCED_GRAYSCALE_4: 1채널 ¼크기 grayscale 영상
IMREAD_REDUCED_GRAYSCALE_4 = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_GRAYSCALE_4)
IMREAD_REDUCED_GRAYSCALE_4 = cv2.cvtColor(IMREAD_REDUCED_GRAYSCALE_4, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_REDUCED_GRAYSCALE_4', IMREAD_REDUCED_GRAYSCALE_4, True)

# cv2.IMREAD_REDUCED_GRAYSCALE_8: 1채널 1/8크기 grayscale 영상
IMREAD_REDUCED_GRAYSCALE_8 = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_GRAYSCALE_8)
IMREAD_REDUCED_GRAYSCALE_8 = cv2.cvtColor(IMREAD_REDUCED_GRAYSCALE_8, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_REDUCED_GRAYSCALE_8', IMREAD_REDUCED_GRAYSCALE_8, True)

# cv2.IMREAD_REDUCED_COLOR_2: 3채널 1/2크기 BGR영상
IMREAD_REDUCED_COLOR_2 = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_COLOR_2)
IMREAD_REDUCED_COLOR_2 = cv2.cvtColor(IMREAD_REDUCED_COLOR_2, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_REDUCED_COLOR_2', IMREAD_REDUCED_COLOR_2)

# cv2.IMREAD_REDUCED_COLOR_4: 3채널 1/4크기 BGR영상
IMREAD_REDUCED_COLOR_4 = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_COLOR_4)
IMREAD_REDUCED_COLOR_4 = cv2.cvtColor(IMREAD_REDUCED_COLOR_4, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_REDUCED_COLOR_4', IMREAD_REDUCED_COLOR_4)

# cv2.IMREAD_REDUCED_COLOR_8: 3채널 1/8크기 BGR영상
IMREAD_REDUCED_COLOR_8 = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_COLOR_8)
IMREAD_REDUCED_COLOR_8 = cv2.cvtColor(IMREAD_REDUCED_COLOR_8, cv2.COLOR_BGR2RGB)
plt.show('IMREAD_REDUCED_COLOR_8', IMREAD_REDUCED_COLOR_8)
```

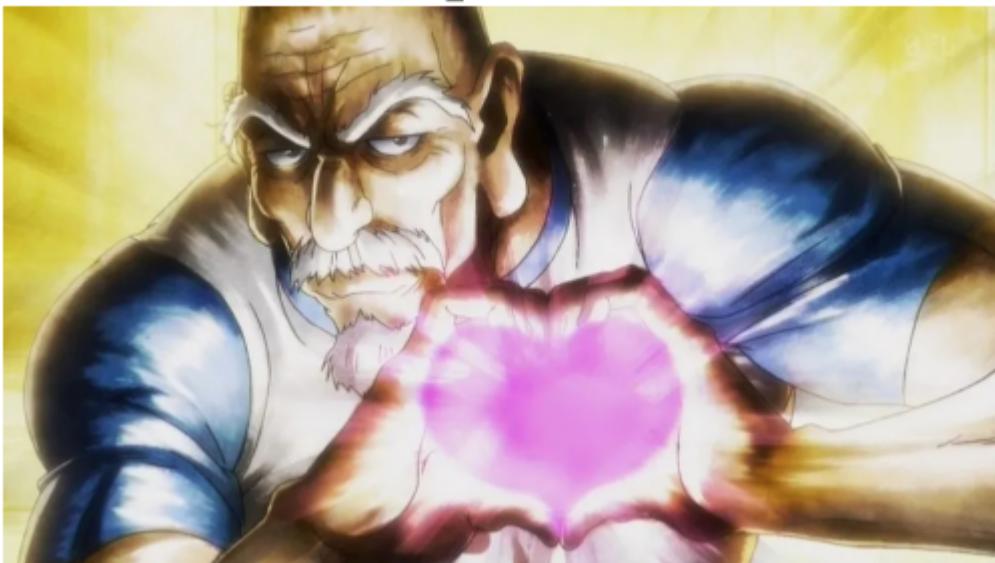
IMREAD_COLOR



IMREAD_GRAYSCALE



IMREAD_UNCHANGED



IMREAD_REDUCED_GRAYSCALE_2



IMREAD_REDUCED_GRAYSCALE_4



IMREAD_REDUCED_GRAYSCALE_8



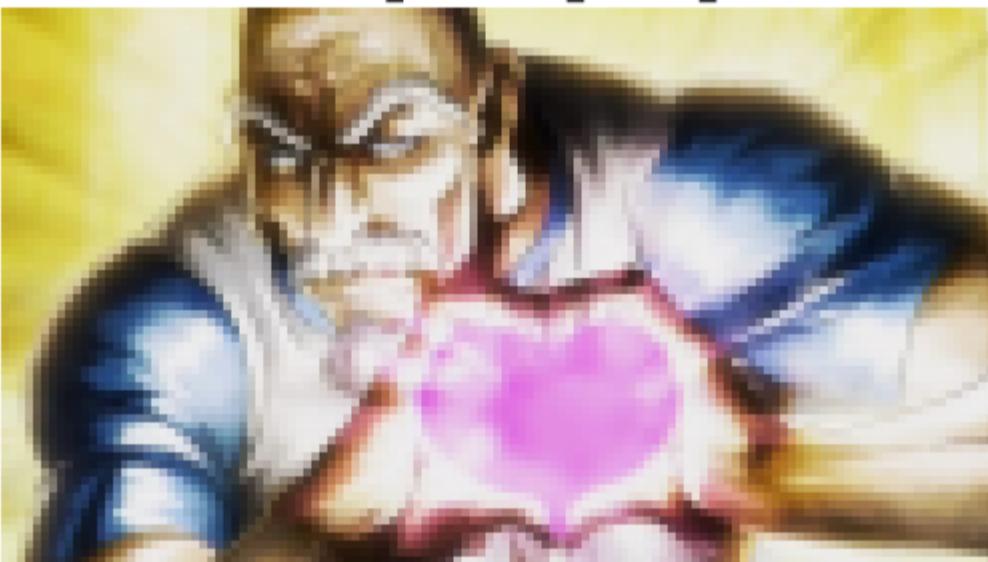
IMREAD_REDUCED_COLOR_2



IMREAD_REDUCED_COLOR_4



IMREAD_REDUCED_COLOR_8



영상 저장 imwrite 함수

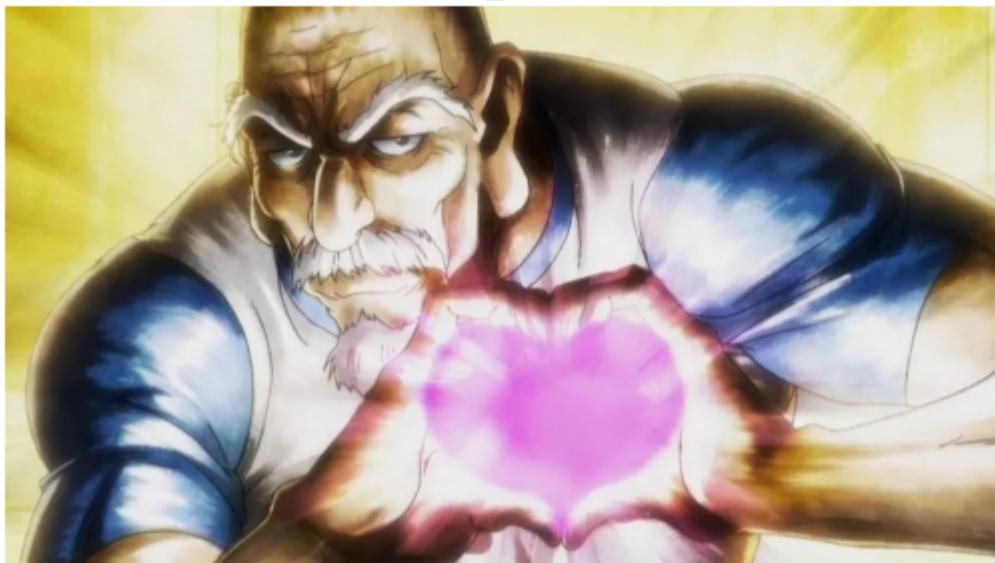
- 영상을 레스터 그래픽 영상 파일 형식으로 저장함
- 저장하는 파일의 형식은 파일 확장자에 따라서 결정됨
 - .jpg / .bmp / .png
- 함수명 : `cv2.imwrite(fileName, image, flags)`
- 매개변수
 - `fileName (str)` : 저장할 영상 파일의 경로(상대 경로 또는 절대 경로 이용)
 - `image (numpy.ndarray)` : 저장할 영상 데이터
 - `flags (int)` : 영상 파일을 저장할 때 옵션
- 리턴값 : `True` 또는 `False (bool)`

```
In [4]: img = cv2.imread('./images/netero.jpg', cv2.IMREAD_UNCHANGED)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
cv2.imwrite('./output/netero.jpg', img)
cv2.imwrite('./output/netero.bmp', img)
cv2.imwrite('./output/netero.png', img)
jpg_img = cv2.imread('./output/netero.jpg', cv2.IMREAD_UNCHANGED)

plt.show('jpg_img', jpg_img)
bmp_img = cv2.imread('./output/netero.bmp', cv2.IMREAD_UNCHANGED)
plt.show('bmp_img', bmp_img)
```

```
png_img = cv2.imread('./output/netero.png', cv2.IMREAD_UNCHANGED)
plt.show('png_img', png_img)
```

jpg_img



bmp_img



png_img



영상 화면 출력 imshow 함수

- 영상을 지정한 윈도우에 출력
- "윈도우 이름"에 따라서 출력되는 윈도우가 결정

일치하는 윈도우 이름이 없으면 새로운 윈도우를 생성하여 출력

- 함수명 : `cv2.imshow(window_name, image)`
- 매개변수
 - `window_name (str)` : 윈도우 창의 Title 내용
 - `image (numpy.ndarray)` : 출력할 영상 데이터
- 리턴값 : N/A

```
In [15]: img = cv2.imread('./images/netero.jpg', cv2.IMREAD_GRAYSCALE)
cv2.imshow("image", img)
cv2.waitKey(0)
```



영상정보 출력

- 입력 영상에 대한 넓이, 높이 채널 수 정보 출력

```
In [16]: # color image
img = cv2.imread('images/netero.jpg', cv2.IMREAD_UNCHANGED)
h, w, c = img.shape
print(f'height : {h}, width : {w}, channel : {c}')

#grayscale image
img = cv2.imread('images/netero.jpg', cv2.IMREAD_GRAYSCALE)
h, w = img.shape
print(f'height : {h}, width : {w}')

height : 562, width : 1000, channel : 3
height : 562, width : 1000
```

waitkey 함수

- 지정된 시간 동안 사용자가 키를 입력할 때까지 프로그램을 대기
- 대기하는 시간은 밀리 초(milliseconds) 단위로서 1,000이 1초에 해당
- 함수명 : `cv2.waitKey(delay)`
- 매개변수
 - `delay (int)` : 대기할 시간 (밀리 초 단위) delay가 0보다 작거나 같은 경우 무한 (infinite) 대기를 의미
- 리턴값 : 사용자가 입력한 키 값 (int)

```
In [ ]: img = cv2.imread('./images/netero.jpg', cv2.IMREAD_UNCHANGED)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.show('Image', img)

# cv2.waitKey(1000)
key = cv2.waitKey(0)
print(f'key = {key} ({chr(key)})')
```

Image



이진화 코드 : threshold() 함수

- threshold 함수는 입력한 영상 데이터에 대해 임계값 필터링을 수행하여 이진영상을 생성
- 함수 : `cv2.threshold(src, thresh, maxval, type)
- 매개변수
 - src (numpy.ndarray) : 입력 영상 데이터
 - thresh : 임계값 (threshold value)
 - maxval : THRESH_BINARY & THRESH_BINARY_INV에서 사용할 최대값
 - type : 임계값 필터링 종류 (Threshold Type)
- 리턴값 : 출력 영상데이터 (numpy.ndarray)

type	THRESH_BINARY Python: cv.THRESH_BINARY	$dst(x, y) = \begin{cases} maxval & \text{if } src(x, y) > thresh \\ 0 & \text{otherwise} \end{cases}$
	THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > thresh \\ maxval & \text{otherwise} \end{cases}$
	THRESH_TRUNC Python: cv.THRESH_TRUNC	$dst(x, y) = \begin{cases} threshold & \text{if } src(x, y) > thresh \\ src(x, y) & \text{otherwise} \end{cases}$
	THRESH_TOZERO Python: cv.THRESH_TOZERO	$dst(x, y) = \begin{cases} src(x, y) & \text{if } src(x, y) > thresh \\ 0 & \text{otherwise} \end{cases}$
	THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > thresh \\ src(x, y) & \text{otherwise} \end{cases}$

```
In [5]: src = cv2.imread('./images/netero.jpg', cv2.IMREAD_GRAYSCALE)
src = cv2.cvtColor(src, cv2.COLOR_BGR2RGB)
plt.show('origin_img', src, True)

_, THRESH_BINARY = cv2.threshold(src, 160, 255, cv2.THRESH_BINARY)
plt.show('THRESH_BINARY', THRESH_BINARY, True)

_, THRESH_BINARY_INV = cv2.threshold(src, 160, 255, cv2.THRESH_BINARY_INV)
plt.show('THRESH_BINARY_INV', THRESH_BINARY_INV, True)

_, THRESH_TRUNC = cv2.threshold(src, 160, 255, cv2.THRESH_TRUNC)
plt.show('THRESH_TRUNC', THRESH_TRUNC, True)

_, THRESH_TOZERO = cv2.threshold(src, 160, 255, cv2.THRESH_TOZERO)
plt.show('THRESH_TOZERO', THRESH_TOZERO, True)

_, THRESH_TOZERO_INV = cv2.threshold(src, 160, 255, cv2.THRESH_TOZERO_INV)
plt.show('THRESH_TOZERO_INV', THRESH_TOZERO_INV, True)
```

origin_img



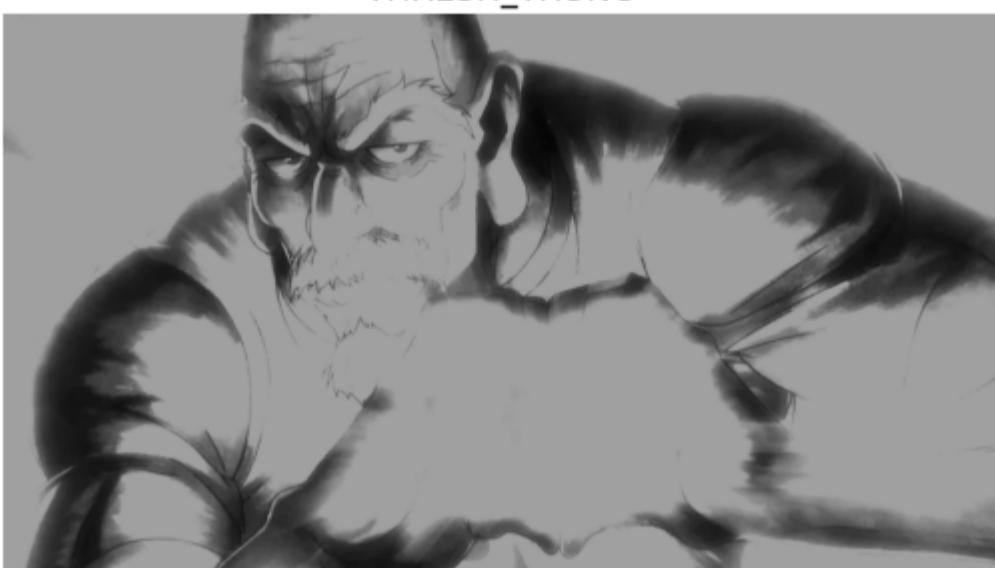
THRESH_BINARY



THRESH_BINARY_INV



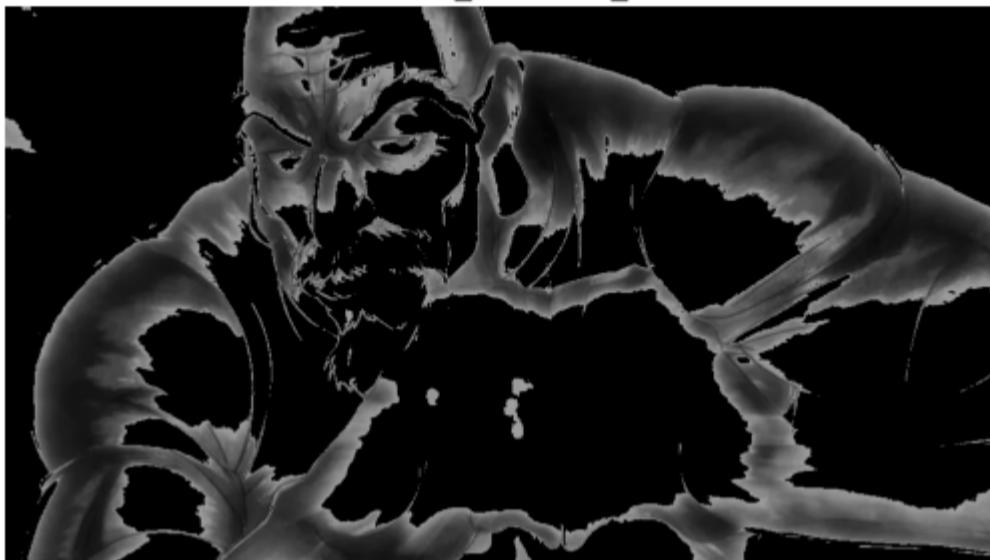
THRESH_TRUNC



THRESH_TOZERO



THRESH_TOZERO_INV



Week 4

기하학적 변환

기본 변환 - Scaling

크기변환 - 확대 및 축소

- 영상의 크기를 변경하는 처리
- OpenCV에서는 cv2.resize() 함수를 사용하여 크기를 변환
- 픽셀값 보간 과정에서는 최소 이웃 보간법(nearest neighbor interpolation), 양방향 선형 보간법(bilinear interpolation), Cubic 보간법(cubic interpolation) 방법들을 많이 사용

영상 크기 변환 함수 : resize() 함수

- 입력된 영상 데이터에 대하여 크기를 변환
- 함수명 : `cv2.resize(src, dsize, fx, fy, interpolation)`
- 매개변수
 - `src (numpy.ndarray)` : 변환시킬 영상 데이터
 - `dsize` : 가로, 세로, 형태의 튜플 ex) (100, 200)
 - `fx` : 가로 크기의 배수, 2배로 크게 하려면 2, 반으로 줄이려면 0.5
 - : 세로 크기의 배수, 2배로 크게 하려면 2, 반으로 줄이려면 0.5
 - `interpolation` : 보간법 방식
 - `cv2.INTER_NEAREST` : 최소 근접 보간
 - `cv2.INTER_LINEAR` : 양방향 선형 보간
 - `cv2.INTER_CUBIC` : 양방향 선형 보간
 - `cv2.INTER_AREA` : 픽셀 리샘플링 보간
 - `cv2.INTER_LANCZOS4` : Lanczos 보간
 - `cv2.INTER_LINEAR_EXACT` : Bit exact bilinear interpolation
 - `cv2.INTER_NEAREST_EXACT` : Bit exact nearest neighbor interpolation

- 리턴값 : 출력 영상 데이터 (numpy.ndarray)

```
In [6]: ori_img = cv2.imread('images/netero.jpg', cv2.IMREAD_UNCHANGED)
ori_img = cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB)

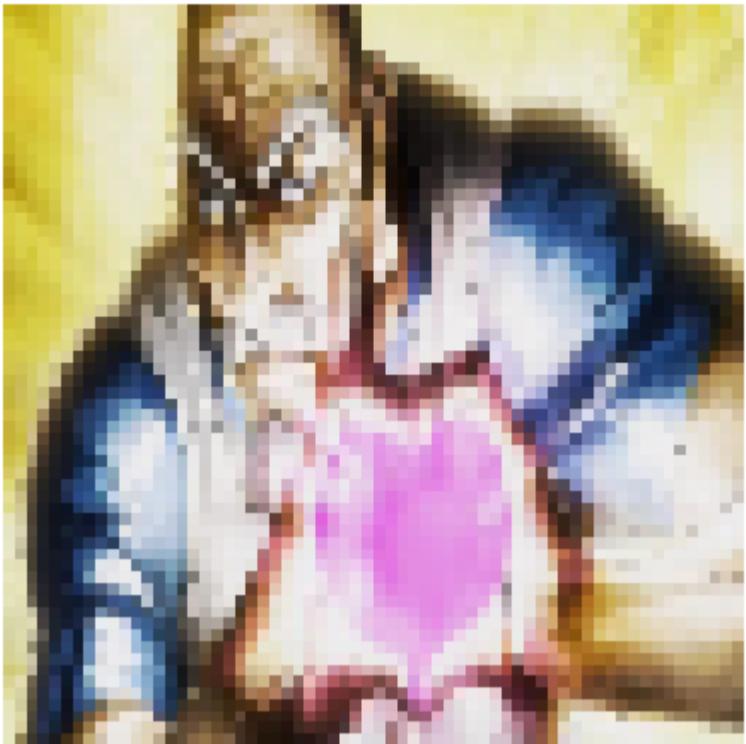
rs_img = cv2.resize(ori_img, (64, 64), interpolation=cv2.INTER_LINEAR)
plt.show('128x64 image', rs_img)

z_img1 = cv2.resize(rs_img, (512, 512), interpolation=cv2.INTER_NEAREST)
z_img2 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_NEAREST)
z_img3 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_CUBIC)
```

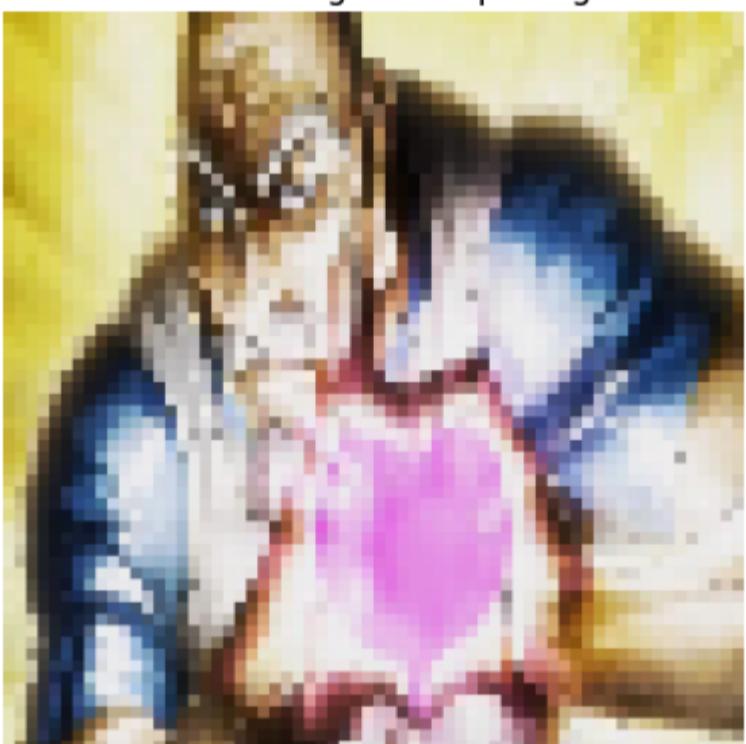
```
z_img4 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_AREA)
z_img5 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_LANCZOS4)
z_img6 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_LINEAR_EXACT)
z_img7 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_NEAREST_EXACT)

plt.show('Nearest neighbor intp image', z_img1)
plt.show('Bilinear intp image', z_img2)
plt.show('Cubic intp image', z_img3)
plt.show('Area intp image', z_img4)
plt.show('lanczos intp image', z_img5)
plt.show('linear_exact intp image', z_img6)
plt.show('nearest_exact intp image', z_img7)
```

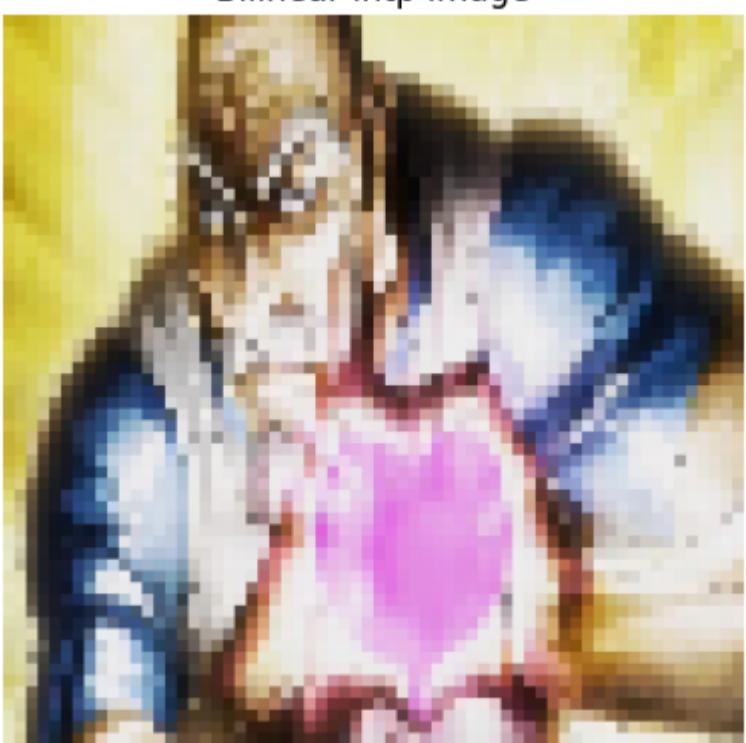
128x64 image



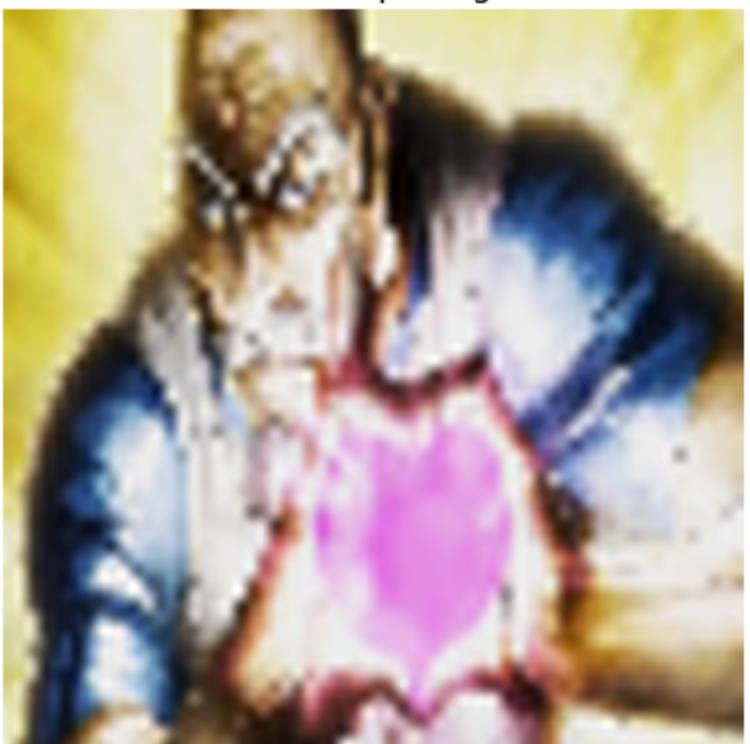
Nearest neighbor intp image



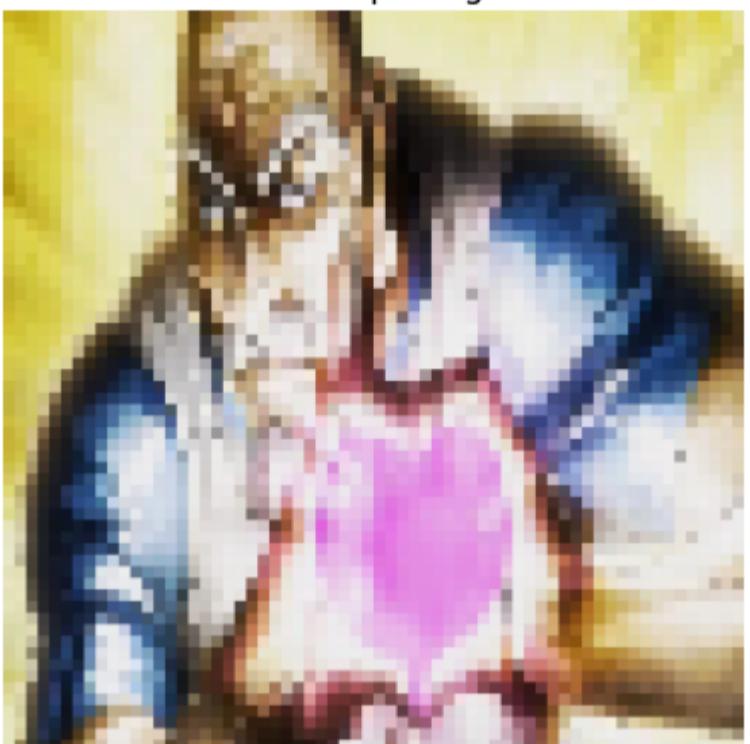
Bilinear intp image



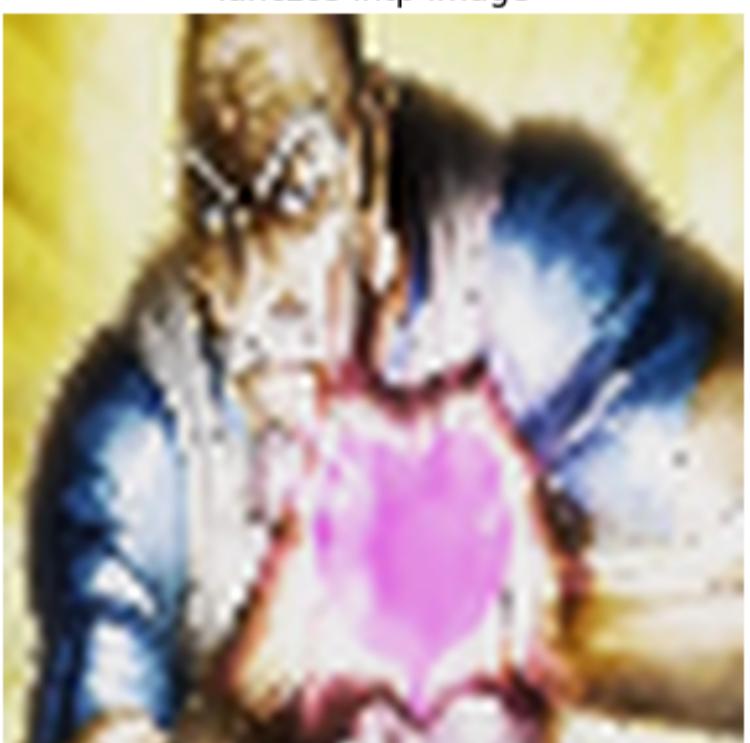
Cubic intp image



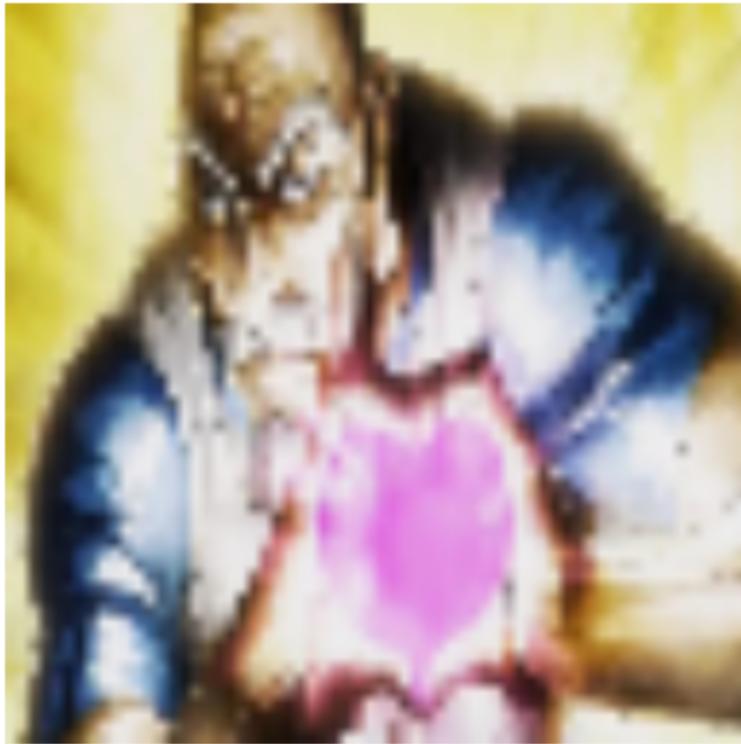
Area intp image



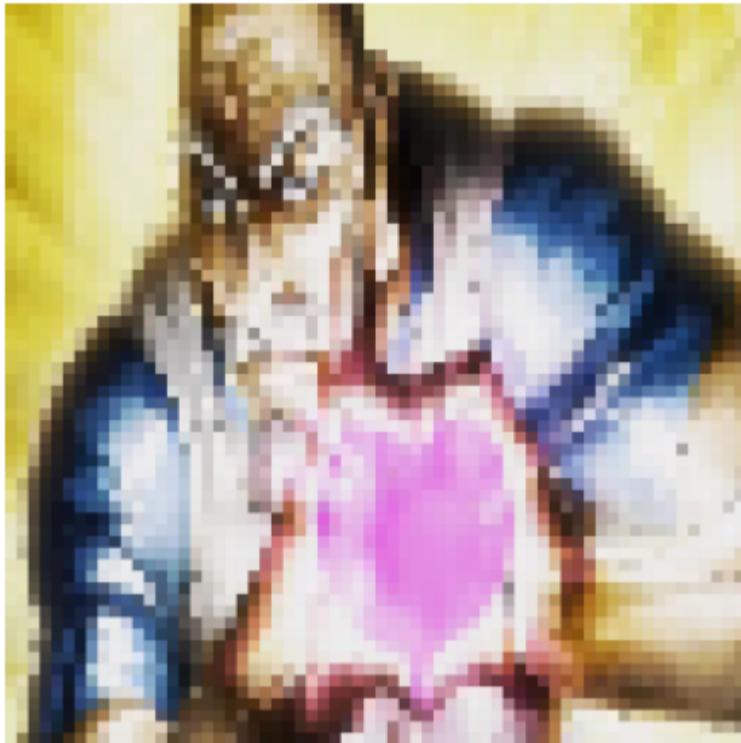
lanczos intp image



linear_exact intp image



nearest_exact intp image



기본 변환 - Translation

이동 변환(Translation)

- 영상을 이동시키는 처리
- 정수형 이동시에는 보간이 불필요함 (실수형 이동시에 필요)
- OpenCV에서는 cv2.warpAffine() 함수를 사용하여 이동 변환

영상 어파인(Affine) 변환 함수 : warpAffine() 함수

- 입력된 영상 데이터에 대하여 지정한 어파인 변환 행렬에 따른 변환을 수행
- 변환 행렬이 이동 행렬일 경우 이동 변환을 수행

- 함수명 : `cv2.warpAffine(src, M, dsize, dst, flags, borderMode, borderValue)`
- 매개변수
 - `src` (`numpy.ndarray`) : 변환시킬 영상 데이터
 - `M` : (2x3) 변환 행렬
 - `dsize` : 가로, 세로, 형태의 튜플 ex) (100, 200)
 - `dst` : 출력 영상 데이터 (dszie 크기로 조정됨)
 - `flags` : 보간 방법과 역변환 방법 설정
 - `borderMode` : 픽셀 외삽(extrapolation) 방법 BorderTypes
 - `cv2.BORDER_CONSTANT` : iiii|abcdefgh|iiii 여기서 i는 지정한 `borderValue`
 - `cv2.BORDER_REPLICATE` : aaaaaa|abcdefghijklhhhhhh 경계 복사
 - `cv2.BORDER_REFLECT` : fedcba|abcdefghijkl|hgfedcb 경계부터 반사
 - `cv2.BORDER_WRAP` : cdefgh|abcdefghijkl|abcdefg 소실된 영역을 이동 변환
 - `borderValue` : BORDER_CONSTANT 경우 사용할 상수값 (기본값은 0)
- 리턴값 : 출력 영상 데이터 (`numpy.ndarray`)

```
In [7]: ori_img = cv2.imread("images/netero.jpg", cv2.IMREAD_UNCHANGED)
ori_img = cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB)
plt.show('Original image', ori_img)
```

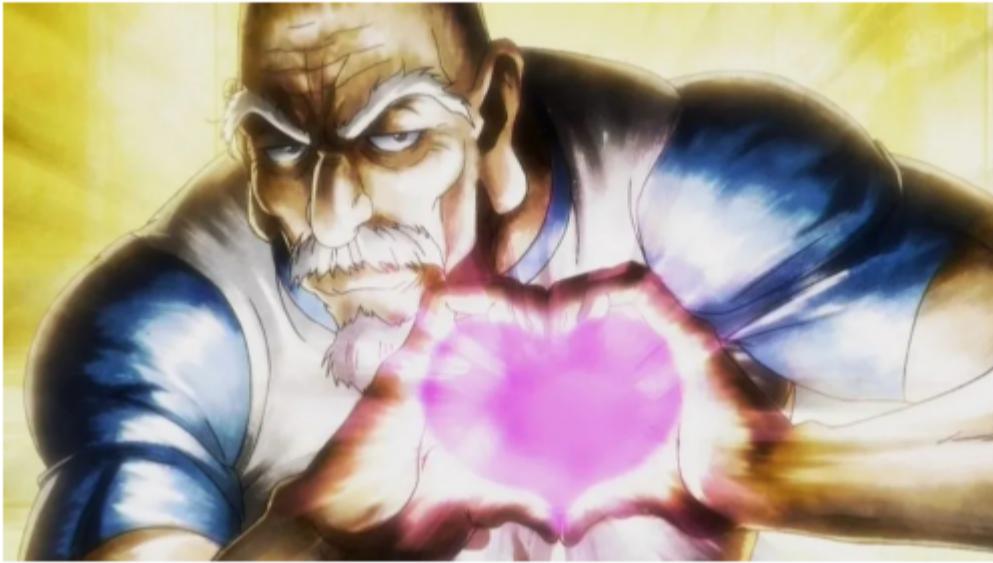
```

rows, cols = ori_img.shape[:2] # channel 여부 무시
Mat = np.float32([[1, 0, 30], [0, 1, 60]])
t_image1 = cv2.warpAffine(ori_img, Mat, (cols, rows))
# 이동된 여백 지정된 색으로
t_image2 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                         borderMode=cv2.BORDER_CONSTANT,
                         borderValue=(255,255,255))
# 이동된 여백 경계면 늘이기
t_image3 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                         borderMode=cv2.BORDER_REPLICATE)
# 이동된 여백 반전으로 채우기
t_image4 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                         borderMode=cv2.BORDER_REFLECT)
# 이동된 여백 자투리 사진으로 채우기
t_image5 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                         borderMode=cv2.BORDER_WRAP)

plt.show('Translation image - default', t_image1)
plt.show('Translation image - BORDER_CONSTANT', t_image2)
plt.show('Translation image - BORDER_REPLICATE', t_image3)
plt.show('Translation image - BORDER_REFLECT', t_image4)
plt.show('Translation image - BORDER_WRAP', t_image5)

```

Original image



Translation image - default



Translation image - BORDER_CONSTANT



Translation image - BORDER_REPLICATE



Translation image - BORDER_REFLECT



Translation image - BORDER_WRAP



기본 변환 - Rotation

회전 변환 (Rotation)

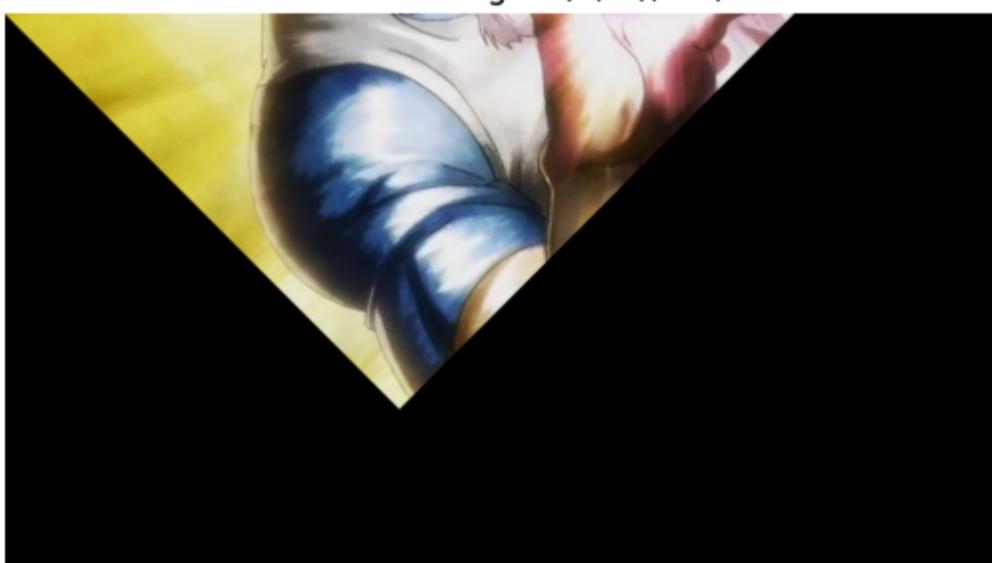
- 영상을 회전시키는 처리
- OpenCV에서는 cv2.warpAffine() 함수를 사용하여 회전 변환

```
In [8]: ori_img = cv2.imread("images/netero.jpg", cv2.IMREAD_UNCHANGED)
ori_img = cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB)
plt.show('Original image', ori_img)
rows, cols = ori_img.shape[:2] # channel 여부 무시
# getRotationMatrix2D((센터 좌표 type: tuple), (회전각도 type: float), (확대률 type: float))
Mat1 = cv2.getRotationMatrix2D((0, 0), 45, 1.0)
Mat2 = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1.0)
Mat3 = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1.0)
r_image1 = cv2.warpAffine(ori_img, Mat1, (cols, rows))
r_image2 = cv2.warpAffine(ori_img, Mat2, (cols, rows),
borderMode=cv2.BORDER_REPLICATE)
r_image3 = cv2.warpAffine(ori_img, Mat2, (cols, rows),
borderMode=cv2.BORDER_DEFAULT)
r_image4 = cv2.warpAffine(ori_img, Mat3, (cols, rows))
plt.show('Rotation image - (0, 0), 45', r_image1)
plt.show('Rotation image - (w/2, h/2), 45 - replicate)', r_image2)
plt.show('Rotation image - (w/2, h/2), 45 - default)', r_image3)
plt.show('Rotation image - (w/2, h/2), 90', r_image4)
```

Original image



Rotation image - (0, 0), 45



Rotation image - (w/2, h/2), 45 - replicate



Rotation image - (w/2, h/2), 45 - default



Rotation image - (w/2, h/2), 90



기본 변환 - Flip

대칭(Flip) 변환 또는 반사(Reflection) 변환

- 기하학적으로 반사(reflection) 변환 형태로 특정한 축 기준의 대칭 변환으로 수행
- OpenCV에서는 cv2.flip() 함수를 사용하여 대칭 변환을 수행

영상 대칭 변환 함수 : flip() 함수

- 입력된 영상 데이터에 대하여 대칭 변환을 수행
- 함수명 : cv2.flip(src, flipCode)
- 매개변수
 - src(numpy.ndarray) : 변환시킬 입력영상 데이터
 - flipCode : 플립 방법을 지정하는 코드
 - x축 대칭 = 0
 - y축 대칭 = 양수
 - x축 및 y축 대칭 = 음수
- 리턴값 : 출력 영상 데이터 (numpy.ndarray)

```
In [9]: ori_img = cv2.imread("images/netero.jpg", cv2.IMREAD_UNCHANGED)
ori_img = cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB)

f_image0 = cv2.flip(ori_img, 0)
f_imagep1 = cv2.flip(ori_img, 1)
f_imagem1 = cv2.flip(ori_img, -1)

plt.show('ori img', ori_img)
plt.show('flip img (up/down, 0)', f_image0)
plt.show('flip img (left/right, 1)', f_imagem1)
plt.show('flip img (y=x, -1)', f_imagep1)
```

ori img



flip img (up/down, 0)



flip img (left/right, 1)



flip img (y=x, -1)



week5

기하학적 변환

어파인 변환(Affine)

어파인 변환 (Affine) 변환

- 평행선이 유지되는 영상 변환
- 어파인 변환에서는 이동, 확대, 크기 변환, 반전이 포함
- OpenCV에서는 cv2.warpAffine() 함수를 사용하여 어파인 변환

다양한 어파인 변환을 수행하는 코드

- 어파인 변환에 의한 영상 변환을 쉽게 확인하기 위하여 RGB 색상으로 각 변환 기준 점의 위치를 표시

```
In [10]: ori_img = cv2.imread('images/netero.jpg', cv2.IMREAD_UNCHANGED)
c_img = cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB)
plt.show('Origin Image', c_img)

rows, cols = ori_img.shape[:2] # channel 여부 무시

# pts1 좌표 표시
pts1 = np.float32([[200, 200], [300, 200], [200, 300]])

cv2.circle(c_img, (200, 200), 9, (255, 0, 0), -1)
```

```

cv2.circle(c_img, (300, 200), 9, (0, 255, 0), -1)
cv2.circle(c_img, (200, 300), 9, (0, 0, 255), -1)

pts2 = np.float32([[200, 200], [350, 200], [200, 250]])
Mat1 = cv2.getAffineTransform(pts1, pts2)
r_image1 = cv2.warpAffine(c_img, Mat1, (cols, rows))

pts2 = np.float32([[200, 200], [300, 230], [260, 300]])
Mat2 = cv2.getAffineTransform(pts1, pts2)
r_image2 = cv2.warpAffine(c_img, Mat2, (cols, rows))

pts2 = np.float32([[200, 200], [100, 170], [170, 100]])
Mat3 = cv2.getAffineTransform(pts1, pts2)
r_image3 = cv2.warpAffine(c_img, Mat3, (cols, rows))

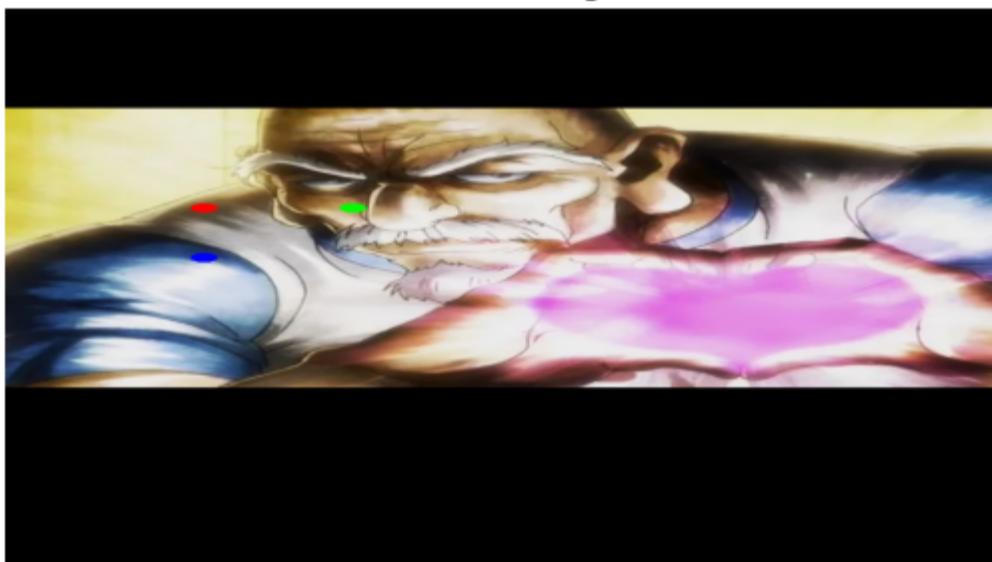
plt.show('Affine 1 image', r_image1)
plt.show('Affine 2 image', r_image2)
plt.show('Affine 3 image', r_image3)

```

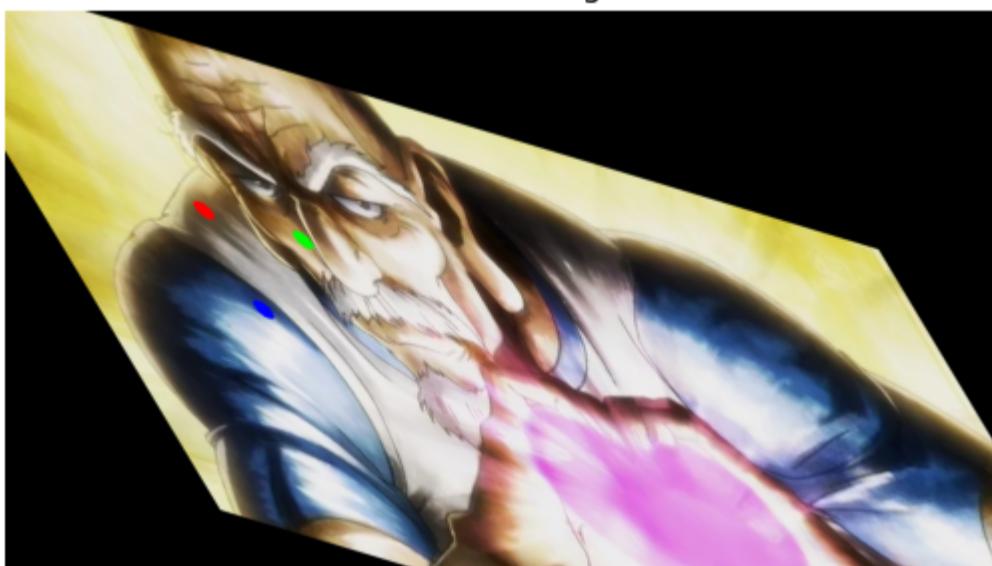
Origin Image



Affine 1 image



Affine 2 image



Affine 3 image



원근 변환(Perspective)

원근 변환

- 직선의 성질만 유지되고 평행선이 유지되지 않는 영상 변환
- OpenCV에서는 cv2.warpPerspective() 함수를 사용하여 원근 변환을 변환

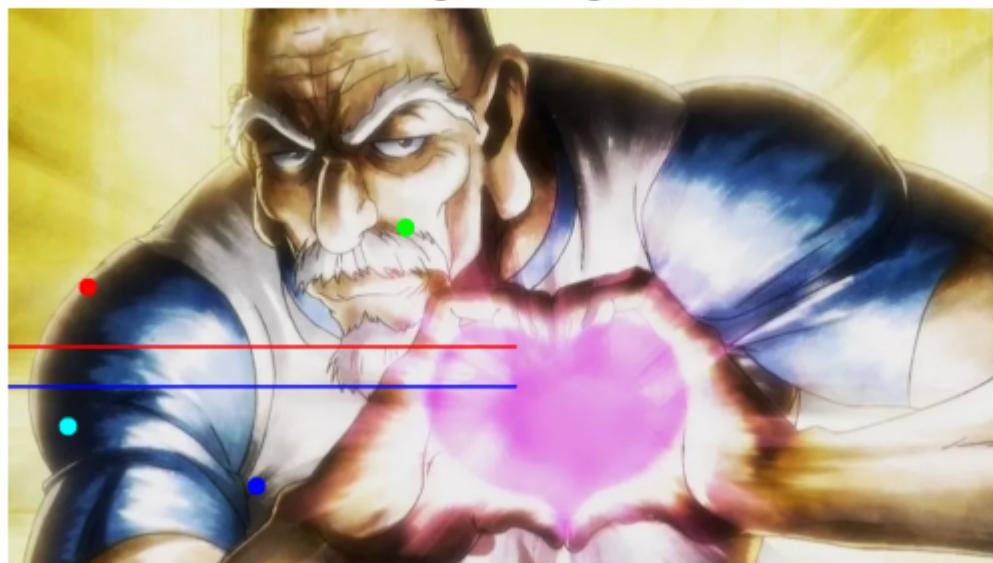
영상 원근(Perspective) 변환 함수 : cv2.warpPerspective() 함수

- 입력된 영상 데이터에 대하여 지정한 원근 변환 행렬에 따른 변환을 수행
- 함수명 : cv2.warpPerspective(src, M, dsize, dst, flags, borderMode, borderValue)
- 매개변수
 - src(numpy.ndarray) : 변환시킬 입력영상 데이터
 - M (3x3) 변환 행렬
 - dsize : 출력 영상 데이터 크기
 - dst : 출력 영상 데이터 (dsize 크기로 조정됨)
 - flags : 보간 방법과 역변환 방법 설정
 - borderMode : 픽셀 외삽(extrapolation) 방법
 - borderValue : BORDER_CONSTANT 경우 사용할 상수값 (기본값은 0)
- 리턴값 : 출력 영상 데이터 (numpy.ndarray)

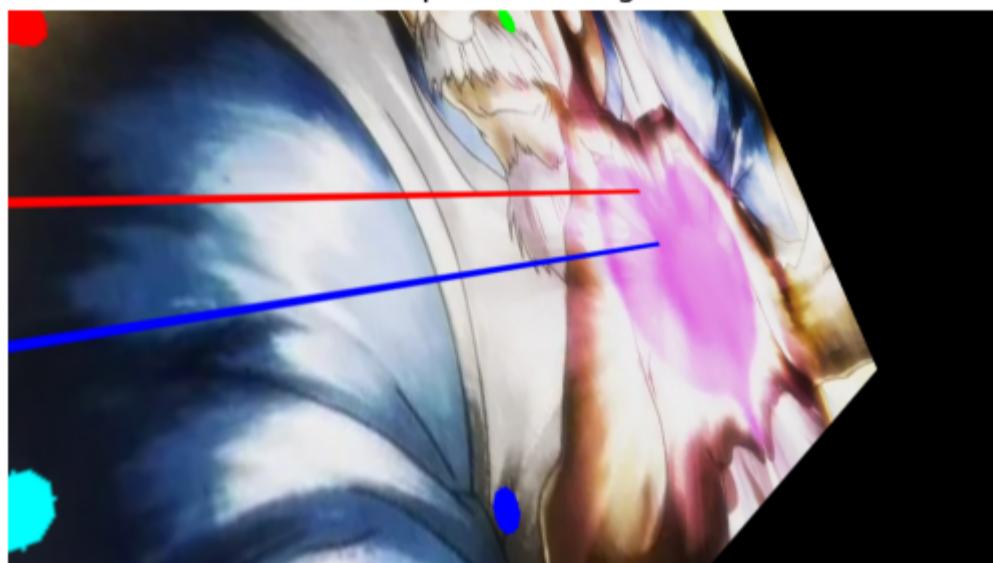
```
In [11]: ori_img = cv2.imread("images/netero.jpg", cv2.IMREAD_COLOR)
ori_img = cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB)
rows, cols = ori_img.shape[:2] # channel 여부 무시
# pts1 좌표 표시
pts1 = np.float32([[80, 280], [400, 220], [250, 480], [60, 420]])
cv2.circle(ori_img, (80, 280), 9, (255, 0, 0), -1)
cv2.circle(ori_img, (400, 220), 9, (0, 255, 0), -1)
cv2.circle(ori_img, (250, 480), 9, (0, 0, 255), -1)
cv2.circle(ori_img, (60, 420), 9, (0, 255, 255), -1)
cv2.line(ori_img, (0, 340), (511, 340), (255, 0, 0), 2)
cv2.line(ori_img, (0, 380), (511, 380), (0, 0, 255), 2)
pts2 = np.float32([[10, 10], [502, 10], [502, 502], [10, 502]])
Mat1 = cv2.getPerspectiveTransform(pts1, pts2)
print('Perspective matrix')
print(Mat1)
r_image = cv2.warpPerspective(ori_img, Mat1, (cols, rows))
plt.show('Original image', ori_img)
plt.show('Perspective image', r_image)
```

```
Perspective matrix
[[ 3.23187758e+00  4.47028347e-01 -3.74074102e+02]
 [ 5.45182861e-01  2.73071886e+00 -7.98571868e+02]
 [ 3.12600637e-03 -1.02027266e-03  1.00000000e+00]]
```

Original image



Perspective image



Week 6

에지와 특징

Roberts & Prewitt 에지

로버츠(Roberts)와 프리위(Prewitt) 필터

- 에지 검출을 위한 1차 미분 마스크 형태를 정의
- 미분은 공간 기반 처리의 대표적인 컨볼루션을 통하여 수행

```
In [12]: img = cv2.imread('images/netero.jpg',cv2.IMREAD_REDUCED_GRAYSCALE_2)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
roberts_x = np.array([[0, 0,-1], [0, 1, 0], [0, 0, 0]]) #|
roberts_y = np.array([[ -1, 0, 0], [0, 1, 0], [0, 0, 0]]) #\

prewitt_x = np.array([[1, 0,-1], [1, 0,-1], [1, 0,-1]]) #| 수직엣지 검출 시험출제
prewitt_y = np.array([[ -1,-1,-1], [0, 0, 0], [1, 1, 1]]) #- 수평엣지 검출 시험출제

r_imageX = cv2.convertScaleAbs(cv2.filter2D(img,-1, roberts_x))
r_imageY = cv2.convertScaleAbs(cv2.filter2D(img,-1, roberts_y))

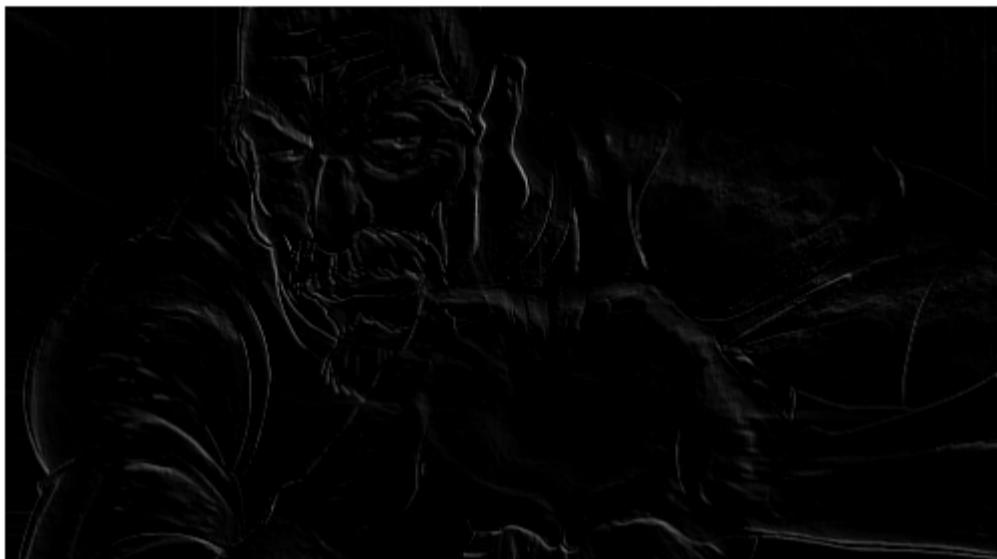
p_imageX = cv2.convertScaleAbs(cv2.filter2D(img,-1, prewitt_x))
p_imageY = cv2.convertScaleAbs(cv2.filter2D(img,-1, prewitt_y))

plt_show('Original image', img, True)
plt_show('Roberts X direction image', r_imageX, True)
plt_show('Roberts Y direction image', r_imageY, True)
plt_show('Prewitt X direction image', p_imageX, True)
plt_show('Prewitt Y direction image', p_imageY, True)
```

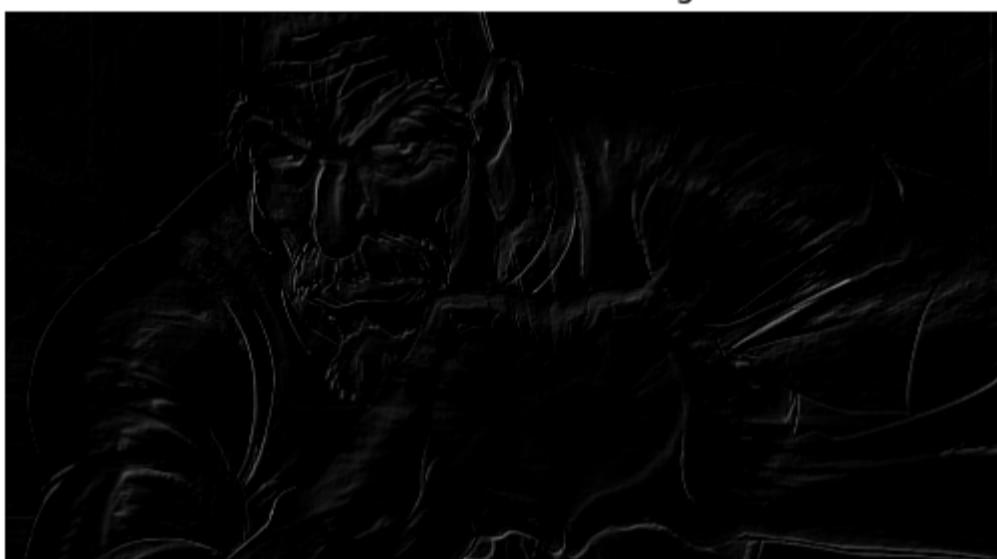
Original image



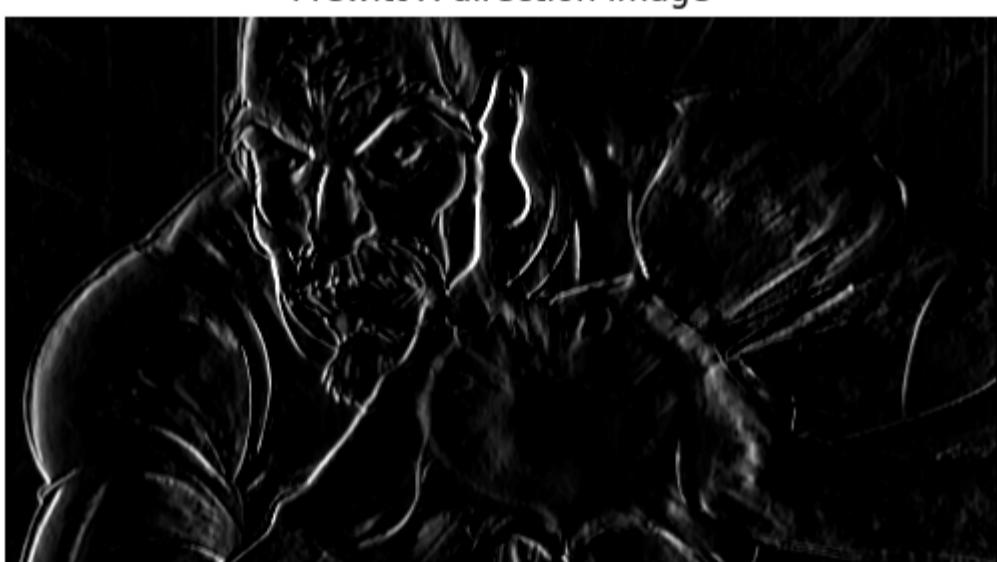
Roberts X direction image



Roberts Y direction image



Prewitt X direction image



Prewitt Y direction image



소벨 필터

소벨(Sobel filter) 필터

- 영상 처리에 있어서 가장 많이 사용되는 1차 미분 필터
- 프리윗 필터와 유사하지만, 중심 픽셀에 가까운 점에서 영향력을 더 높게 고려하도록 설계
- 중앙 부분의 기울기 변화를 더 크게 반영하면서, 주변 픽셀의 잡음(노이즈) 영향은 덜 받도록 설계

소벨 필터링 함수 : Sobel()함수

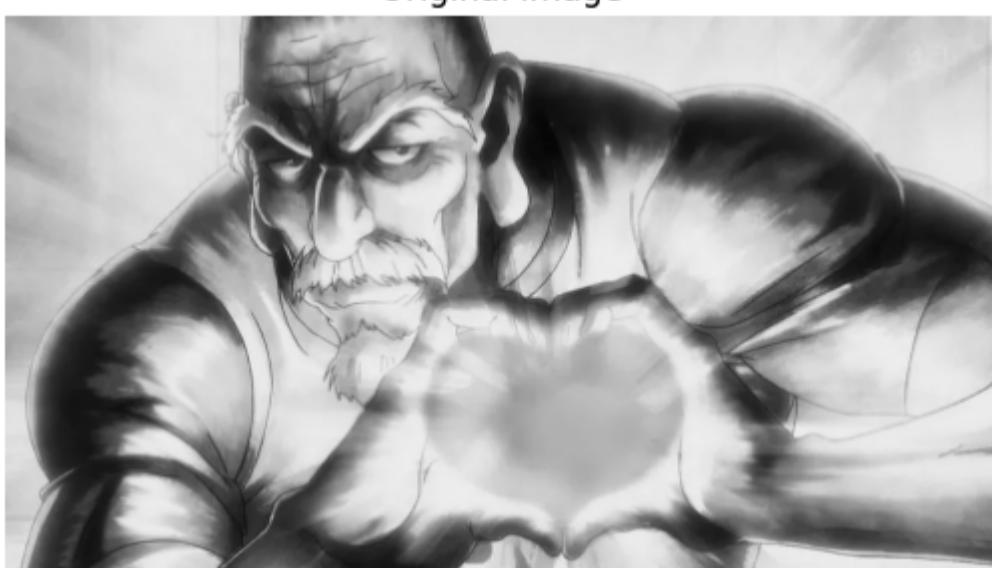
- 입력된 영상 데이터에 대하여 소벨 필터를 적용
- 함수명 : `cv2.Sobel(src, Ddepth, dx, dy, dst, ksize, scale, delta, borderType)`
- 매개변수
 - `src` : 입력 영상 데이터
 - `Ddepth` : 출력 영상의 깊이 정밀도 (-1이면 입력 영상과 동일)
 - `dx` : x 방향 미분 차수
 - `dy` : y 방향 미분 차수
 - `dst` : 출력 영상 데이터
 - `ksize` : 소벨 마스크 커널 크기 (홀수 1,3,5,7)
 - `scale` : 계산된 미분 값에 적용할 스케일 요소 (비율)
 - `delta` : 오프셋
 - `borderType` : 픽셀 외삽(extrapolation) 방법 BORDER_CONSTANT 또는 BORDER_REPLICATE
- 리턴값 : 출력 영상 데이터 (numpy.ndarray)

```
In [13]: img = cv2.imread('./images/netero.jpg', cv2.IMREAD_REDUCED_GRAYSCALE_2)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

s_imageX = cv2.Sobel(img, cv2.CV_8U, 1, 0, ksize=3)
s_imageY = cv2.Sobel(img, cv2.CV_8U, 0, 1, ksize=3)
s_imageXY = cv2.Sobel(img, cv2.CV_8U, 1, 1, ksize=3)

plt.show('Original image', img)
plt.show('Sobel X direction image', s_imageX, True)
plt.show('Sobel Y direction image', s_imageY, True)
plt.show('Sobel X-Y direction image', s_imageXY, True)
```

Original image



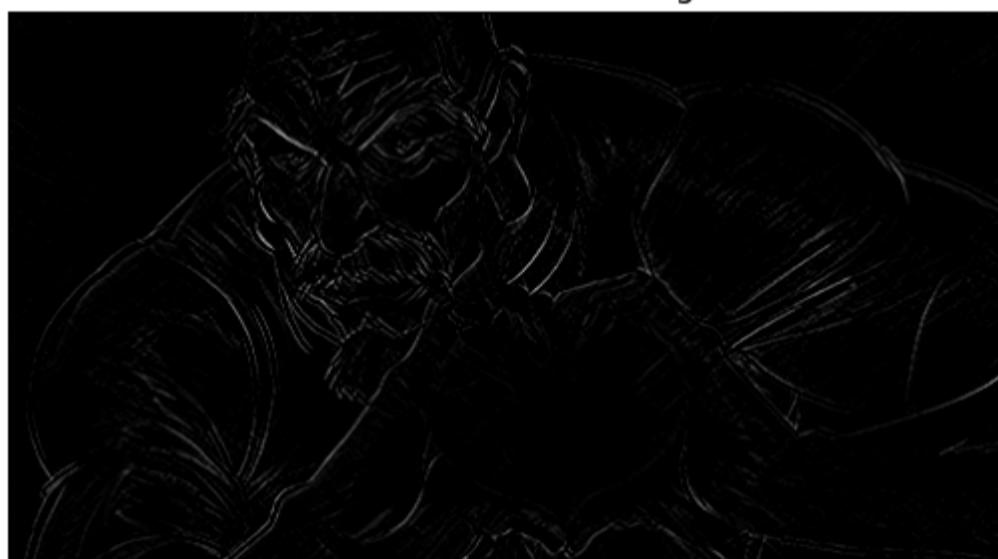
Sobel X direction image



Sobel Y direction image



Sobel X-Y direction image



라플라시안 필터

라플라시안 필터링 함수: `Laplacian()` 함수

- 입력된 영상 데이터에 대하여 라플라시안 필터를 적용
- 함수명 : `cv2.Laplacian(src, ddepth, dst, ksize, scale, delta, borderType)`
- 매개변수
 - `src` : 입력 영상 데이터
 - `ddepth` : 출력 영상의 깊이 정밀도 (-1이면 입력 영상과 동일)
 - `dst` : 출력 영상 데이터
 - `ksize` : 2차 미분 필터 계산에 사용할 커널 크기 (양수, 홀수)
 - `scale` : 계산된 미분 값에 적용할 스케일 요소 (비율)
 - `delta` : 오프셋
 - `borderType` : 픽셀 외삽(extrapolation) 방법 BORDER_CONSTANT 또는 BORDER_REPLICATE
- 리턴값 : 출력 영상 데이터 (`numpy.ndarray`)

```
In [14]: img = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_GRAYSCALE_2)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
l_image = cv2.Laplacian(img, cv2.CV_8U, ksize=3)
plt.show('Original image', img, True)
plt.show('Laplacian image', l_image, True)
```

Original image



Laplacian image



샤르 필터

Scharr 필터링 함수: Scharr() 함수

- 입력된 영상 데이터에 대하여 Scharr 필터를 적용
- 함수명 : cv2.Scharr(src, ddepth, dx, dy, dst, scale, delta, borderType)
- 매개변수
 - src : 입력 영상 데이터
 - ddepth : 출력 영상의 깊이 정밀도 (-1이면 입력 영상과 동일)
 - dx : x 방향 미분 차수
 - dy : y 방향 미분 차수
 - dst : 출력 영상 데이터
 - scale : 계산된 미분 값에 적용할 스케일 요소 (비율)
 - delta : 오프셋
 - borderType : 픽셀 외삽(extrapolation) 방법 BORDER_CONSTANT 또는 BORDER_REPLICATE
- 리턴값 : 출력 영상 데이터 (numpy.ndarray)

```
In [15]: img = cv2.imread('images/netero.jpg',cv2.IMREAD_REDUCED_GRAYSCALE_2)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
s_imageX = cv2.Scharr(img, cv2.CV_8U, 1,0)
s_imageY = cv2.Scharr(img, cv2.CV_8U, 0,1)
plt.show('Original image', img)
plt.show('Scharr X direction image', s_imageX, True)
plt.show('Scharr Y direction image', s_imageY, True)
```

Original image



Scharr X direction image



Scharr Y direction image



케니 에지

Canny 경계선 검출 함수 : Canny() 함수

- 함수명 : `cv2.Canny(src, threshold1, threshold2, edges, apertureSize, L2gradient)`
- 매개변수
 - `src` : 입력 영상 데이터
 - `threshold1` : hysteresis 절차에 사용되는 1번째 임계값 (최소 임계값)
 - `threshold2` : hysteresis 절차에 사용되는 2번째 임계값 (최대 임계값)
 - `edges` : 출력 영상 데이터 (경계선 데이터)
 - `apertureSize` : Sobel 필터에 사용되는 커널 크기
 - `L2gradient` : 영상 Gradient 계산에 L1 norm을 사용할지 L2 norm을 사용할지 설정
- 리턴값 : 출력 영상 데이터 (`numpy.ndarray`)

```
In [16]: img = cv2.imread('images/netero.jpg', cv2.IMREAD_REDUCED_GRAYSCALE_2)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
c_image1 = cv2.Canny(img, 10, 50)
c_image2 = cv2.Canny(img, 200, 250)
plt.show('Original image', img)
plt.show('Canny image 1', c_image1, True)
plt.show('Canny image 2', c_image2, True)
```

Original image



Canny image 1



Canny image 2

