



Day68; 20221213

날짜	@2022년 12월 13일
유형	@2022년 12월 13일
태그	

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/u8yes/ai>

u8yes/AI



Contributor 1 Issues 0 Stars 0 Forks 0

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/80a6a5f2-80b1-474c-b5ac-15a639ec9e8a/06_%EA%B8%B0%ED%95%98%ED%95%99%EC%B2%98%EB%A6%AC.pdf

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/8045ce4c-c2ec-4296-98ea-fc5cffb19b48/07_%EC%98%81%EC%83%81_%EB%B6%84%ED%95%A0_%EC%B2%98%EB%A6%AC.pdf

프로젝트는 2개를 하는 것으로 7명이 한 팀이 되어 진행됨.

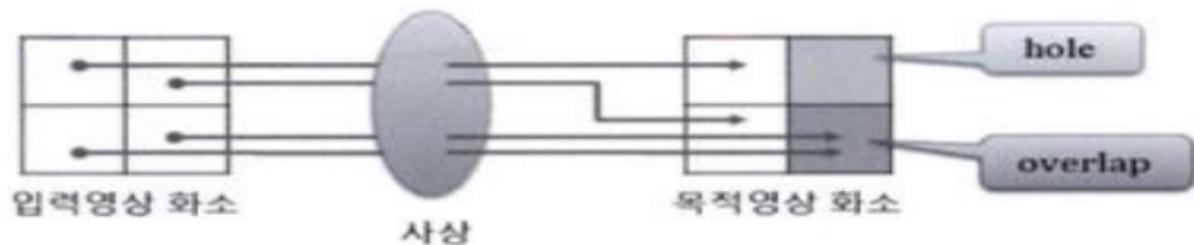
기하학 처리

변환에는 크게 회전, 크기 변경, 평행이동 등이 있으며 영상처리 관련 논문에서는 이 세 가지 변환을 일컬어 RST 변환이라고 하는데 R은 Rotation, S는 Scaling, T는 Translation의 첫 글자

R은 Rotation 회전, S는 Scaling 크기 변경, T는 Translation 평행이동

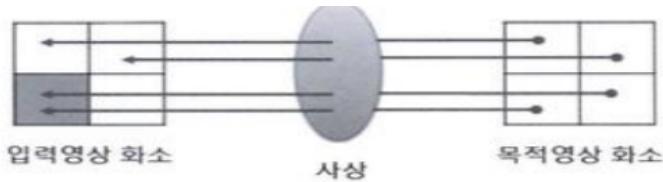
Mapping(사상)

Mapping되지 않는 Pixel을 hole이라고 한다. 영상 축소 시 오버랩이 발생할 수 있다.



홀은 입력 영상의 좌표들로 목적 영상의 좌표를 만드는 과정에서 사상되지 않은 픽셀을 의미하는데 보통 영상을 확대하거나 회전할 때에 발생하는 반면 오버랩은 영상을 축소할 때 주로 발생하는데 이것은 입력 영상의 여러 픽셀들이 목적 영상의 한 픽셀로 사상되는 것을 의미

역방향 사상



- 입력 영상에서 하단 왼쪽 한 개의 픽셀이 목적 영상의 두 개 픽셀로 각각 사상
 - 역방향 사상의 방식은 홀이나 오버랩은 발생하지 않지만, 입력 영상의 한 픽셀을 목적 영상의 여러 픽셀에서 사용하게 되면 결과 영상의 품질이 떨어질 수 있음

항상 **반대**로 생각

```
import numpy as np, cv2

def scaling(img, size): # 크기 변경 함수
    # size - 축소, 확대하고자 하는 폭과 높이를 받기
    # 250x300, 24bpp
    # 크기: 102.97 kB
    # 타입: 이미지
    # 수정됨: 2020-01-23 오후 6:53
    # 생성됨: 2022-12-13 오전 9:33

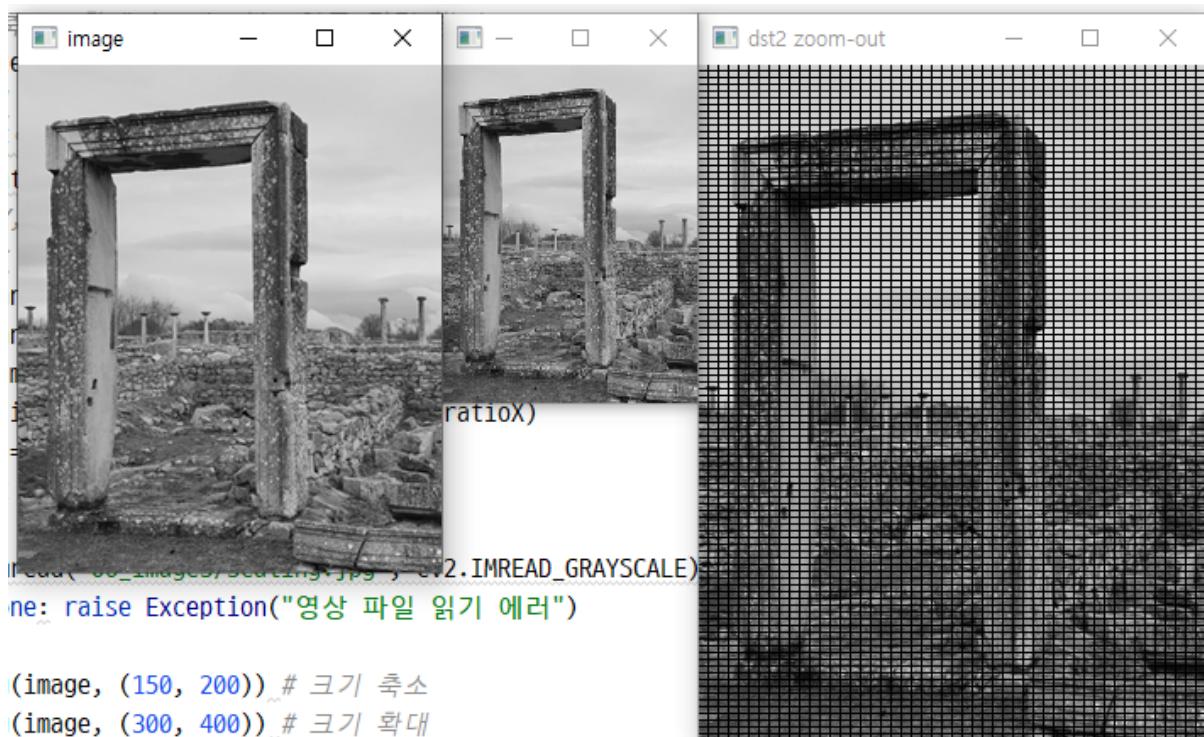
image = cv2.imread("06_1images/scaling.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: r
```

Y
300
행
h

250x300, 24bpp
크기: 102.97 kB
타입: 이미지
수정됨: 2020-01-23 오후 6:53
생성됨: 2022-12-13 오전 9:33

01_scaling.py

```
scaling.py ^  
  # img와 동일하게 dataType을 잡아줌.  
  ratioY, ratioX = np.divide(size[::-1], img.shape[:2])  
  # 행ratioY, 열ratioX을 나누어주면 비율이 나누어진다.  
  # 원본이랑 나누기 해서 비율을 계산, 그러면 1이 나오게 됨.  
  y = np.arange(0, img.shape[0], 1) # 0 ~ 299까지 step 1씩 증가시킴.  
  x = np.arange(0, img.shape[1], 1)  
  y, x = np.meshgrid(y, x) # 직사각형의 격자가 만들어져서 반환됨  
  i, j = np.int32(y * ratioY), np.int32(x * ratioX)  
  dst[i, j] = img[y, x]  
  return dst  
  
image = cv2.imread("06_images/scaling.jpg", cv2.IMREAD_GRAYSCALE)  
if image is None: raise Exception("영상 파일 읽기 에러")  
  
dst1 = scaling(image, (150, 200)) # 크기 축소  
dst2 = scaling(image, (300, 400)) # 크기 확대  
  
cv2.imshow("image", image)  
cv2.imshow("dst1 zoom-in", dst1)  
cv2.imshow("dst2 zoom-out", dst2)  
  
cv2.waitKey(0)
```



빈 hole이 생기는 현상을 볼 수가 있음.

보간법

interpolation

미국·영국 [ɪntə:rpeleɪʃən] ◻ 영국식 ◻

- 1 [U] 써넣음, [C] 써넣은 어구[사항]
- 2 보간법(補間法), 내삽법(內插法)

~-----~-----~-----~-----~

02_scaling_nearest.py

```
import numpy as np, cv2

def scaling(img, size): # 크기 변경 함수
    # size - 축소, 확대하고자 하는 인품 전달 받기
    dst = np.zeros(size[:-1], img.dtype)
    # size값이 역순으로 해당 크기를 만들어줌.
    # img와 동일하게 dataType을 잡아줌.
    ratioY, ratioX = np.divide(size[:-1], img.shape[:2])
    # 행ratioY, 열ratioX을 나누어주면 비율이 나누어진다.
    # 원본이랑 나누기 해서 비율을 계산, 그러면 1이 나오게 됨.
    y = np.arange(0, img.shape[0], 1) # 0 ~ 299까지 step 1씩 증가시킴.
    x = np.arange(0, img.shape[1], 1)
    y, x = np.meshgrid(y, x) # 직사각형의 격자가 만들어져서 반활됨
    i, j = np.int32(y * ratioY), np.int32(x * ratioX)
    dst[i, j] = img[y, x]
    return dst
```

```

def scaling_nearest(img, size):
    dst = np.zeros(size[::-1], img.dtype)
    ratioY, ratioX = np.divide(size[::-1], img.shape[:2])
    i = np.arange(0, size[1], 1)
    j = np.arange(0, size[0], 1)
    i, j = np.meshgrid(i, j)
    y, x = np.int32(i / ratioY), np.int32(j / ratioX)
    # 반올림 해주면 이웃픽셀까지 해당범위에 들어오게 되는 효과
    dst[i, j] = img[y, x]

    return dst

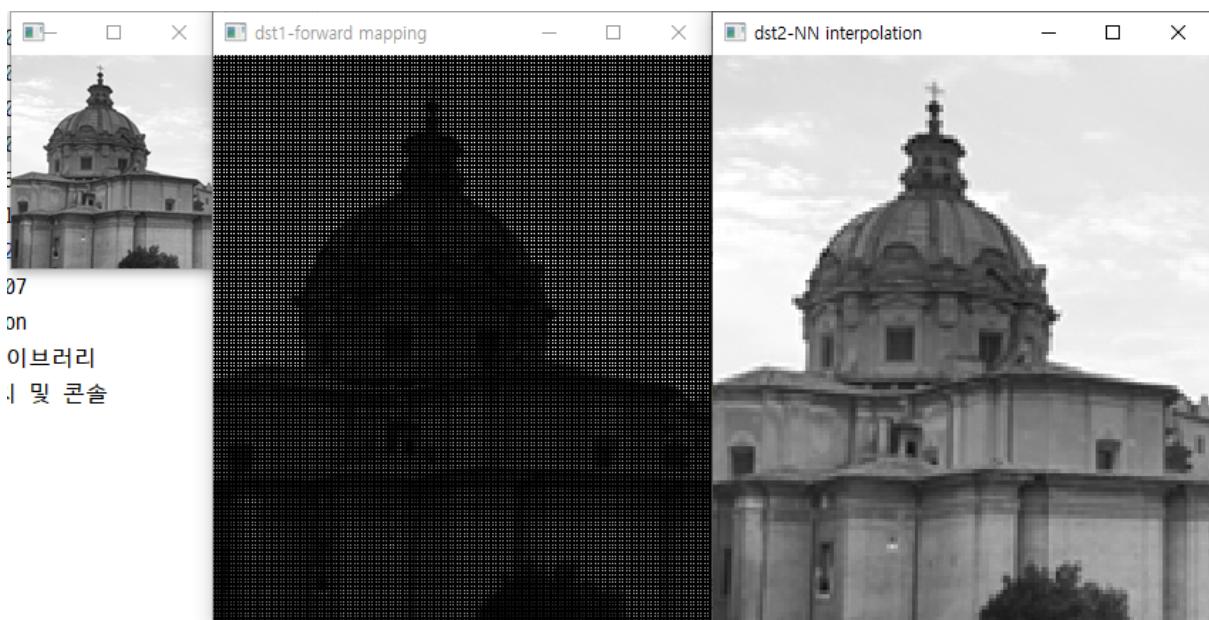
image = cv2.imread("06_images/interpolation.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")

dst1 = scaling(image, (350, 400)) # 확대시킴
dst2 = scaling_nearest(image, (350, 400))

cv2.imshow("image", image)
cv2.imshow("dst1-forward mapping", dst1)
cv2.imshow("dst2-NN interpolation", dst2)

cv2.waitKey(0)

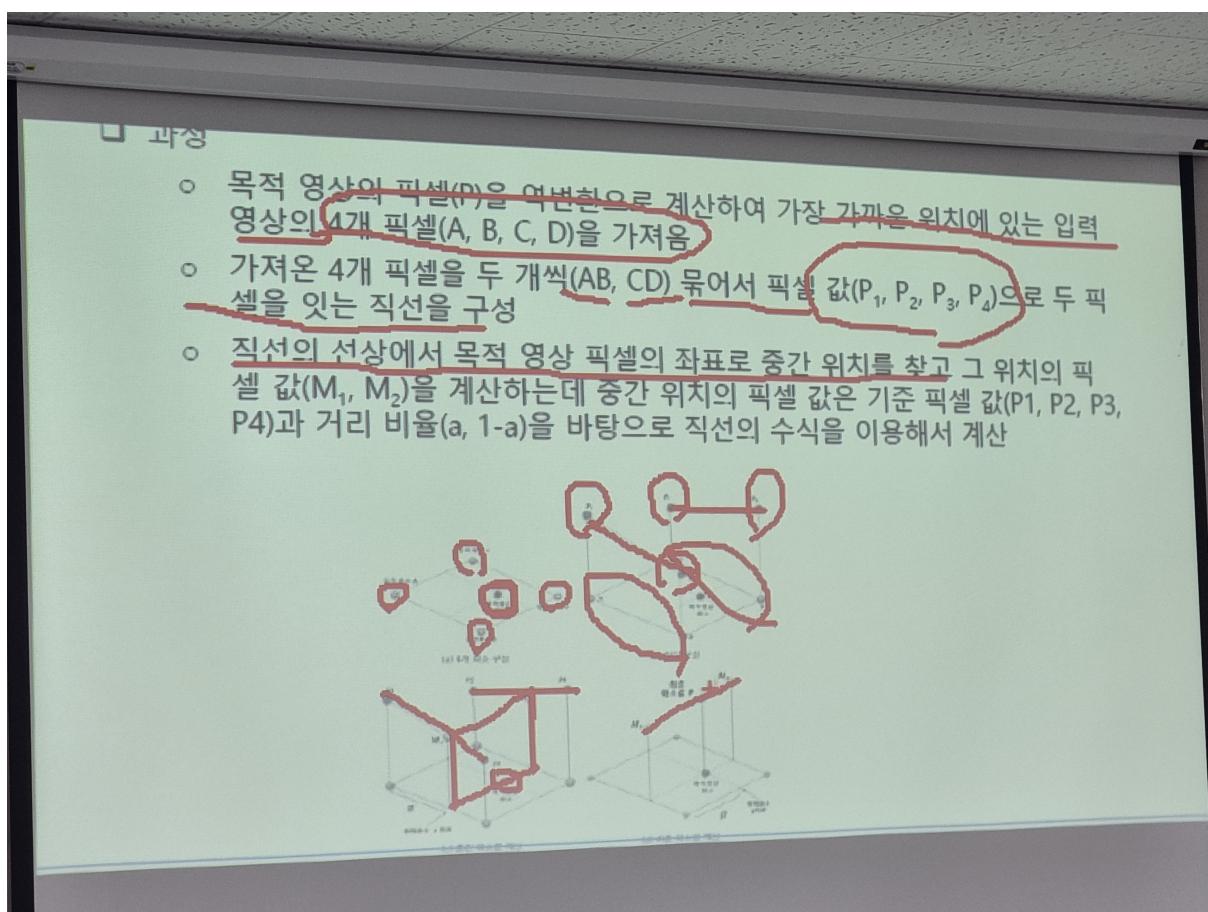
```



양선형 보간법(bilinear interpolation)

보간

- ✓ 양선형 보간법(bilinear interpolation)
 - 선형 보간을 두 번에 걸쳐서 수행하는 것이 양선형 보간
 - 과정



두 번 선형을 계산한다고 해서 양선형(bilinear interpolation)

03_scaling_interpolation.py

```

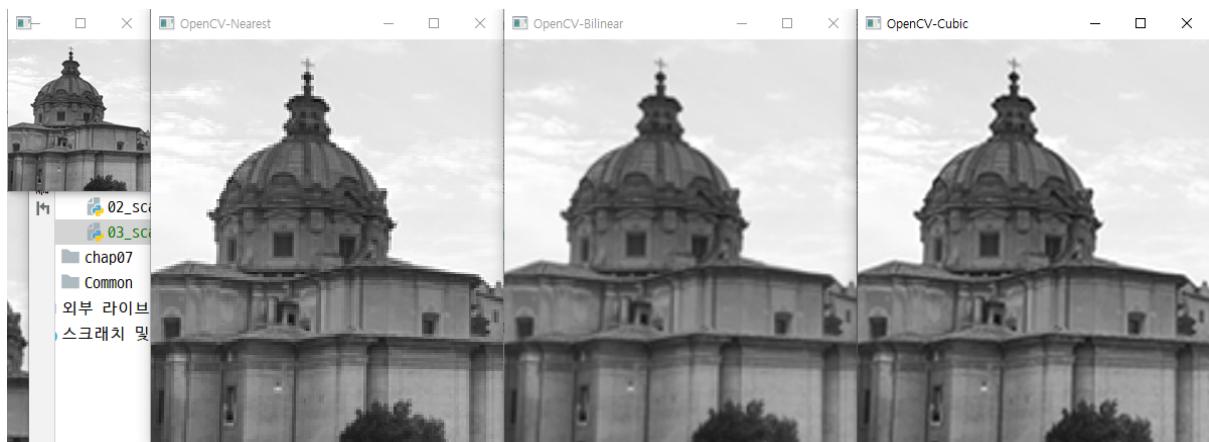
import numpy as np, cv2

image = cv2.imread("06_images/interpolation.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")
# 141 by 150
size = (350, 400)

dst1 = cv2.resize(image, size, interpolation=cv2.INTER_NEAREST) # 최근접 보간
dst2 = cv2.resize(image, size, interpolation=cv2.INTER_LINEAR) # 양선형 보간
dst3 = cv2.resize(image, size, interpolation=cv2.INTER_CUBIC) # 3차 회선 보간

cv2.imshow("image", image)
cv2.imshow("OpenCV-Nearest", dst1)
cv2.imshow("OpenCV-Bilinear", dst2)
cv2.imshow("OpenCV-Cubic", dst3)
cv2.waitKey(0)

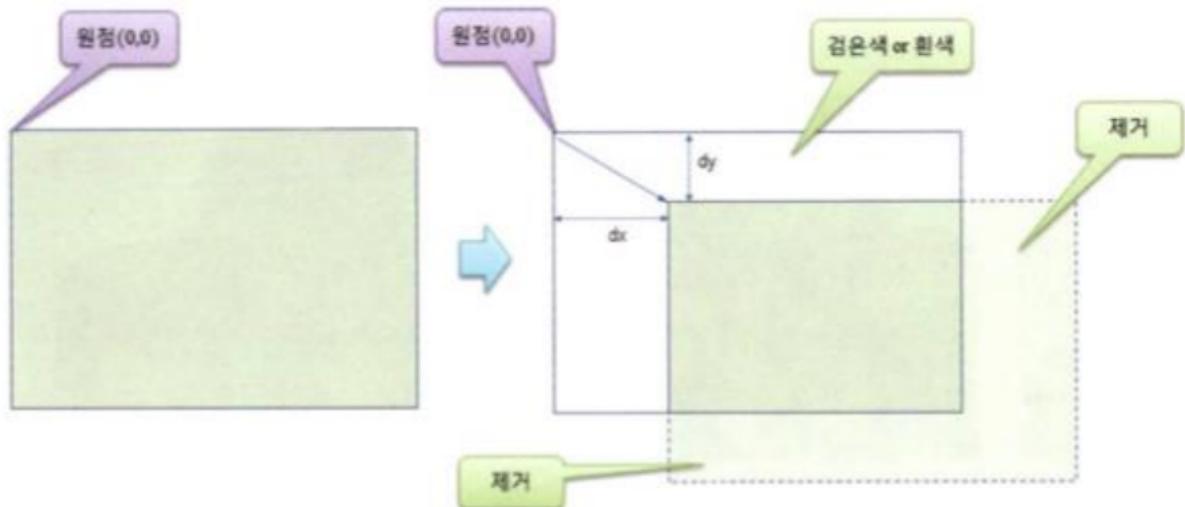
```



양선형 보간은 계단형이 보정되어짐

평행 이동(translation)

이동한 만큼 dx, dy라고 함.



04_translation.py

```
import numpy as np, cv2

def contain(p, shape):
    return 0 <= p[0] < shape[0] and 0 <= p[1] < shape[1]

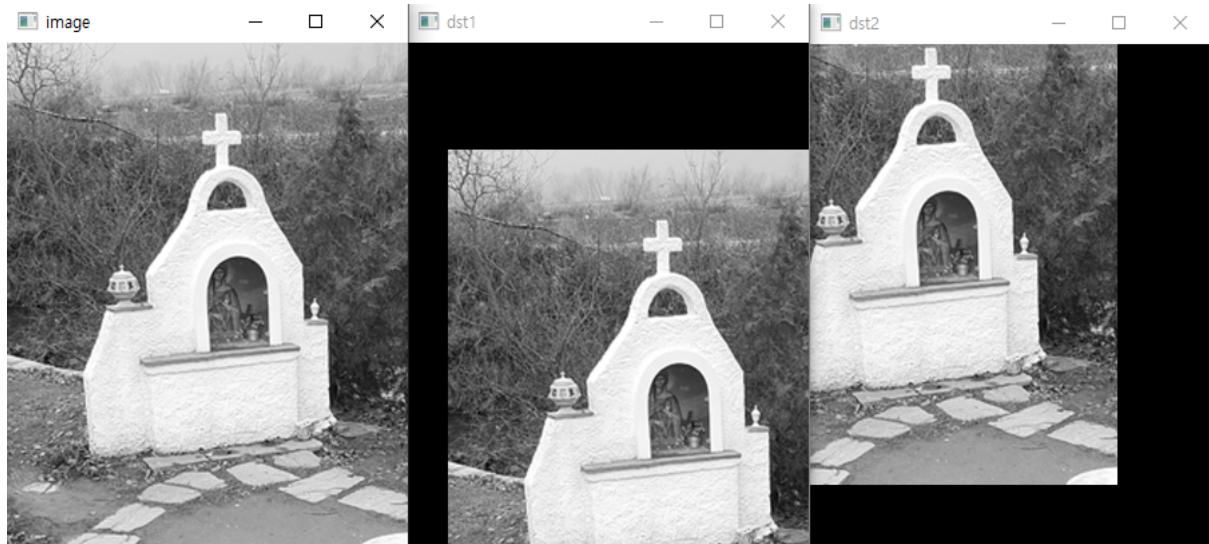
def translate(img, pt):
    dst = np.zeros(image.shape, img.dtype)
    for i in range(img.shape[0]): # 목적 영상 순회·역방향 mapping
        for j in range(img.shape[1]):
            x, y = np.subtract((j, i), pt) # subtract은 뺄셈
            if contain((y, x), img.shape):
                dst[i, j] = img[y, x]

    return dst

image = cv2.imread("06_images/translate.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")

dst1 = translate(image, (30, 80))
dst2 = translate(image, (-70, -50))

cv2.imshow("image", image)
cv2.imshow("dst1", dst1)
cv2.imshow("dst2", dst2)
cv2.waitKey(0)
```



국가공간정보포털

국가는 다양한 방법으로 공간정보 서비스를 제공하기 위해 노력해왔으나, 산재된 서비스 체계로 인해 공간정보 활용에 어려움이 있었습니다. 그래서 국가·공공·민간에서 생산한 공간정보를 한 곳에서, 한 번에,

<http://www.nsdi.go.kr/lxportal/?menuno=2679#none>



05_rotation.py

```
import numpy as np, cv2
from Common.interpolation import bilinear_value

def contain(p, shape): # 좌표(y,x)가 범위내 인지 검사
    return 0 <= p[0] < shape[0] and 0 <= p[1] < shape[1]

def rotate(img, degree):
    dst = np.zeros(img.shape[:2], img.dtype)
    radian = (degree/180) * np.pi
    sin, cos = np.sin(radian), np.cos(radian)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            y = -j * sin + i * cos
            x = j * cos + i * sin

            if contain((y, x), img.shape):
                dst[i, j] = bilinear_value(img, [x, y])

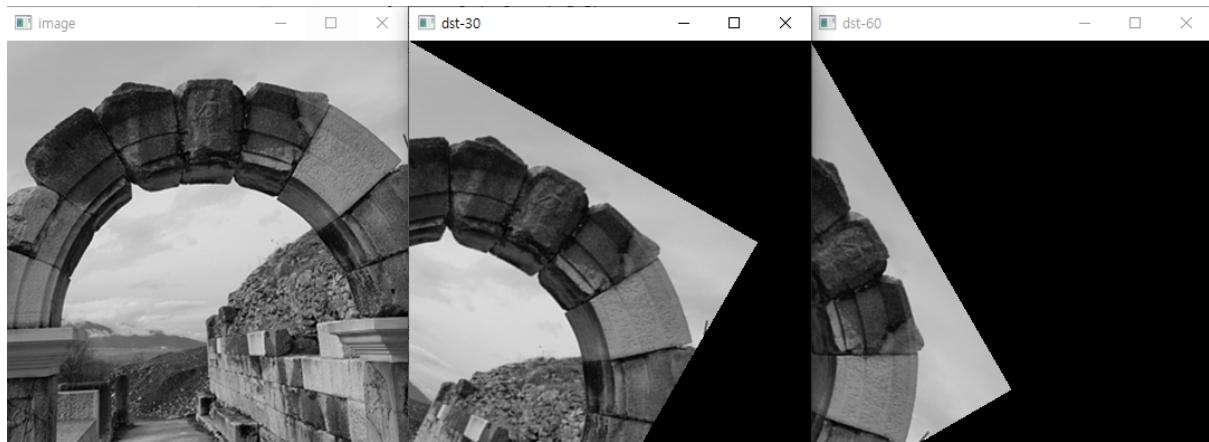
    return dst
```

```
image = cv2.imread("06_images/rotate.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")
# 이미지 사이즈 360x360

dst1 = rotate(image, 30)
dst2 = rotate(image, 60)

cv2.imshow("image", image)
cv2.imshow("dst-30", dst1)
cv2.imshow("dst-60", dst2)

cv2.waitKey(0)
```



Affine 계층

- 행렬의 내적에서는 대응하는 차원의 원소 수 일치 시켜야 함.

$$\begin{matrix} \mathbf{X} & \bullet & \mathbf{W} & = & \mathbf{O} \\ (2,) & & (2, 3) & & (3,) \\ \text{일치} & & & & \end{matrix}$$

↑

- 어파인 변환(Affine transformation) : 신경망의 순전파 때 수행하는 행렬의 내적을 기하학에서 일컫는 말.
- Affine 계층 : affine 변환을 수행하는 처리.

어파인 변환

- OpenCV에서도 어파인 변환을 수행할 수 있는 cv2.warpAffine() 함수를 제공하는데 이 함수는 지정된 어파인 변환 행렬을 적용하면 입력 영상에 어파인 변환을 수행한 목적 영상을 리턴
- cv2.getAffineTransform()은 변환 전의 좌표 3개와 변환 후의 좌표 3개를 지정하면 해당 변환을 수행해 줄 수 있는 어파인 행렬을 반환
- cv2.getRotationMatrix2D()는 회전 변환과 크기 변경을 수행하는 어파인 행렬을 리턴하며 여기서 회전의 방향은 양수일 때 반시계 방향으로 회전하는 행렬을 반환하는데 이것은 영상 좌표에서 직교 좌표계에서의 회전과 같은 방향으로 표현하기 위함

cv2.getRotationMatrix2D() - 시계 방향의 각도로 처리된다. (보통 일반적으로는 반시계 방향인데 이건 시계 방향)

06_affine_transform.py

```
import numpy as np, cv2

image = cv2.imread("06_images/affine.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")
# image size 320x330

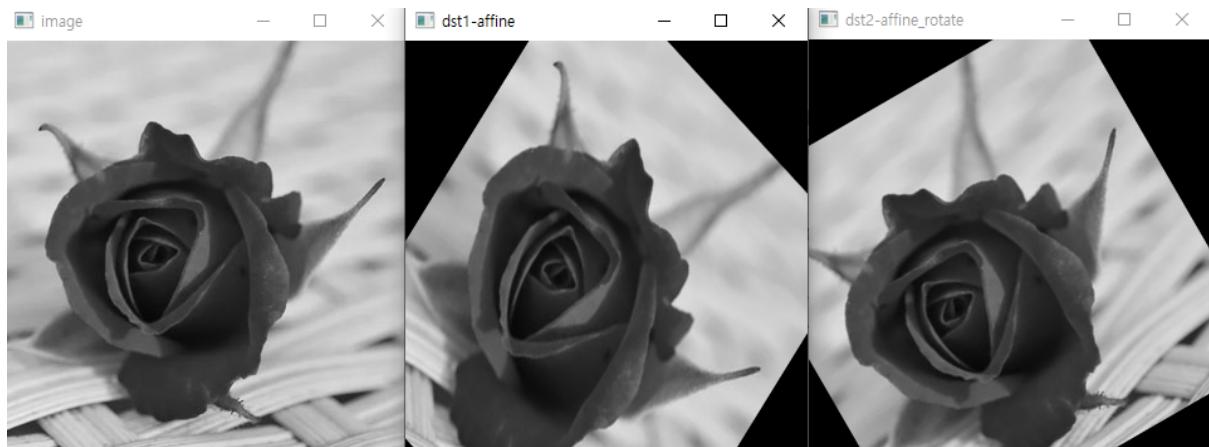
center = (200, 200) # 중심점
angle, scale = 30, 1
size = image.shape[::-1]

pt1 = np.array([(30, 70), (20, 240), (300, 110)], np.float32) # float32는 .001 붙게 만들.
pt2 = np.array([(120, 20), (10, 180), (280, 260)], np.float32) # 1차원

aff_mat = cv2.getAffineTransform(pt1, pt2)
rot_mat = cv2.getRotationMatrix2D(center, angle, scale) # affine 행렬 반환

dst1 = cv2.warpAffine(image, aff_mat, size, cv2.INTER_LINEAR)
dst2 = cv2.warpAffine(image, rot_mat, size, cv2.INTER_LINEAR)

cv2.imshow("image", image)
cv2.imshow("dst1-affine", dst1)
cv2.imshow("dst2-affine_rotate", dst2)
cv2.waitKey(0)
```



허프(Hough) 변환

Hough 변환에 의한 직선

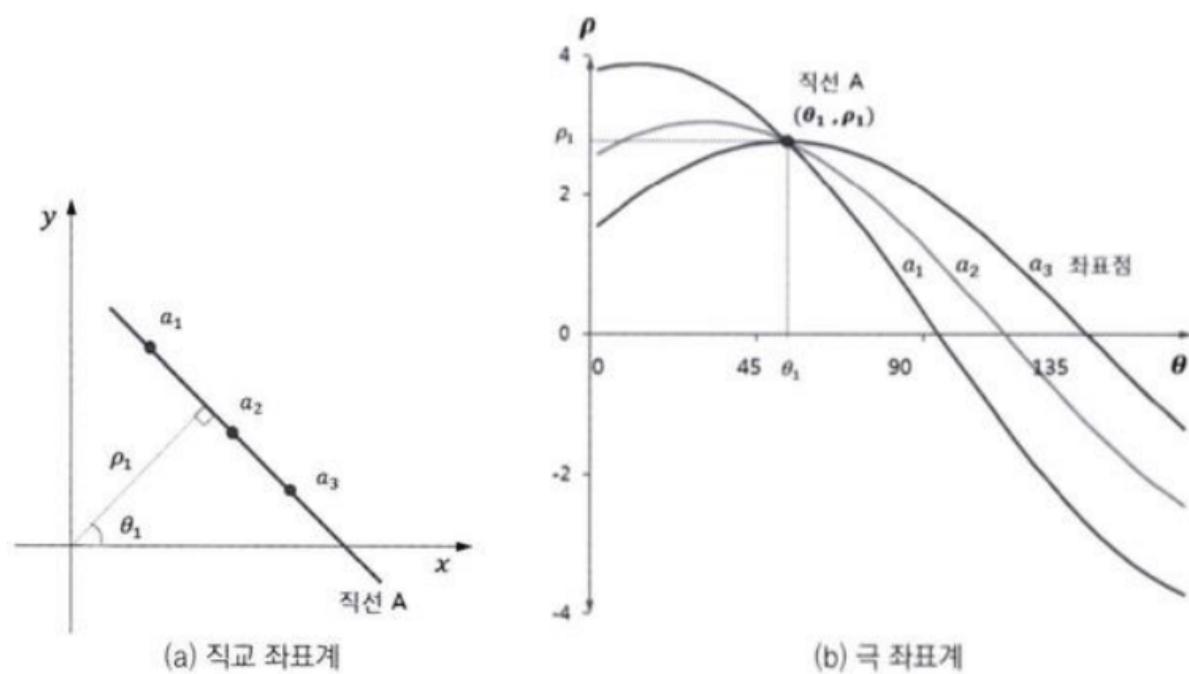
- ✓ 영상 내의 선뿐 아니라 임의의 형태를 지닌 물체를 감지해 내는 대표적인 기술
- ✓ 데이터 손실 및 왜곡이 포함된 영상에서도 직선을 잘 검출



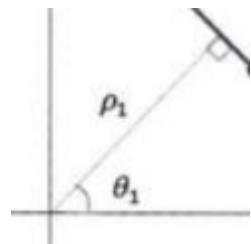
(a) 차선 및 장애물 인식 시스템



(b) Tiny Scanner 앱



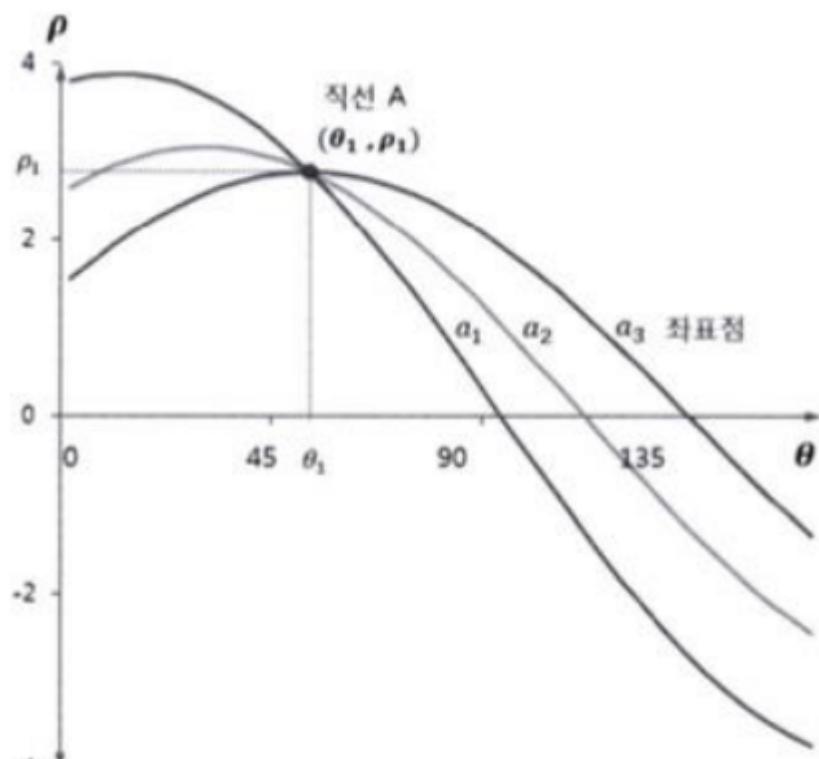
이 값들(각도, 직교하는 선의 거리)을 알면



식을 구할 수 있음.

$$y = ax + b \Leftrightarrow p = x \cos(\theta) + y \sin(\theta)$$

사인의 형태나 코사인의 형태로 변환



(b) 극 좌표계

[01_hough_lines.py](#)

```

import numpy as np, cv2, math
from Common.hough import draw_houghLines, accumulate, masking, select_lines

image = cv2.imread("07_images/hough.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")

blur = cv2.GaussianBlur(image, (5, 5), 2, 2) # 5 by 5
canny = cv2.Canny(blur, 100, 200, 5) # mask는 5로 지정

rho, theta = 1, np.pi / 180

lines2 = cv2.HoughLines(canny, rho, theta, 80)
dst = draw_houghLines(canny, lines2, 7) # lines의 갯수는 7

cv2.imshow("image",image)
cv2.imshow("canny",canny)
cv2.imshow("detected lines_OpenCV", dst)

cv2.waitKey(0)

```



Common/hough.py

```

import numpy as np, math , cv2
from Common.utils import ck_time

```

```

def accumulate(image, rho, theta):
    h, w = image.shape[:2]
    rows, cols = (h + w) * 2 // rho, int(np.pi/theta)# 누적행렬 너비, 높이
    accumulate = np.zeros((rows, cols), np.int32)# 직선 누적행렬

    sin_cos = [(np.sin(t * theta), np.cos(t * theta)) for t in range(cols)]# 삼각 함수값 미리
    저장
    # pts = [(y, x) for x in range(w) for y in range(h) if image[y, x] > 0 ]
    pts = np.where(image > 0)

    polars = np.dot(sin_cos, pts).T# 행렬곱으로 허프 변환 수식 계산
    polars = (polars/ rho + rows / 2)# 해상도 변경 및 위치 조정

    for row in polars.astype(int):
        for t, r in enumerate(row):
            accumulate[r, t] += 1# 좌표 누적

    return accumulate

# 허프 누적 행렬의 지역 최대값 선정
def masking(accumulate, h, w, thresh):
    rows, cols = accumulate.shape[:2]
    dst = np.zeros(accumulate.shape, np.uint32)

    for y in range(0, rows, h):# 누적행렬 조회
        for x in range(0, cols, w):
            roi = accumulate[y:y+h, x:x+w]
            _, max, _, (x0, y0) = cv2.minMaxLoc(roi)
            dst[y+y0, x+x0] = max
    return dst

# 임계값 이상인 누적값(직선) 선별
def select_lines(acc_dst, rho, theta, thresh):
    rows = acc_dst.shape[0]
    r, t = np.where(acc_dst>thresh)# 임계값 이상인 좌표(세로, 가로)

    rhos = ((r - (rows / 2)) * rho)# rho 계산
    radians = t * theta# theta 계산
    value = acc_dst[r, t]# 임계값 이상인 누적값

    idx = np.argsort(value)[::-1]# 누적값 기준 세로 방향 내림차순 정렬
    lines = np.transpose([rhos, radians])# ndarray 객체 생성후 전치
    lines = lines[idx, : ]# 누적값에 따른 정렬

    return np.expand_dims(lines, axis=1)

# 허프 변환
def houghLines(src, rho, theta, thresh):
    acc_mat = accumulate(src, rho, theta)# 허프 누적 행렬 계산
    acc_dst = masking(acc_mat, 7, 3, thresh)# 마스킹 처리 7행,3열
    lines = select_lines(acc_dst, rho, theta, thresh)# 직선 가져오기
    return lines

# 검출한 직선을 원 영상에 그리기
def draw_houghLines(src, lines, nline):

```

```

dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR) # 컬러 영상 변환
min_length = min(len(lines), nline)

for i in range(min_length):
    rho, radian = lines[i, 0, 0:2] # 수직거리 , 각도 - 3차원 행렬임
    a, b = math.cos(radian), math.sin(radian)
    pt = (a * rho, b * rho) # 검출 직선상의 한 좌표 계산
    delta = (-1000 * b, 1000 * a) # 직선상의 이동 위치
    pt1 = np.add(pt, delta).astype('int')
    pt2 = np.subtract(pt, delta).astype('int')
    cv2.line(dst, tuple(pt1), tuple(pt2), (0, 255, 0), 2, cv2.LINE_AA)
return dst

```

chap07 01_hough_lines.py

```

import numpy as np, cv2, math
from Common.hough import accumulate, masking, select_lines
|_
def houghLines(src, rho, theta, thresh):
    acc_mat = accumulate(src, rho, theta) # 허프 누적 행렬 계산
    acc_dst = masking(acc_mat, 7, 3, thresh) # 마스킹 처리 7행,3열
    lines = select_lines(acc_dst, rho, theta, thresh) # 직선 가져오기
    return lines

def draw_houghLines(src, lines, nline):
    dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR) # 컬러 영상 변환
    min_length = min(len(lines), nline)

    for i in range(min_length):
        rho, radian = lines[i, 0, 0:2] # 수직거리 , 각도 - 3차원 행렬임
        a, b = math.cos(radian), math.sin(radian)
        pt = (a * rho, b * rho) # 검출 직선상의 한 좌표 계산
        delta = (-1000 * b, 1000 * a) # 직선상의 이동 위치
        pt1 = np.add(pt, delta).astype('int')
        pt2 = np.subtract(pt, delta).astype('int')
        cv2.line(dst, tuple(pt1), tuple(pt2), (0, 255, 0), 2, cv2.LINE_AA)

    return dst

```

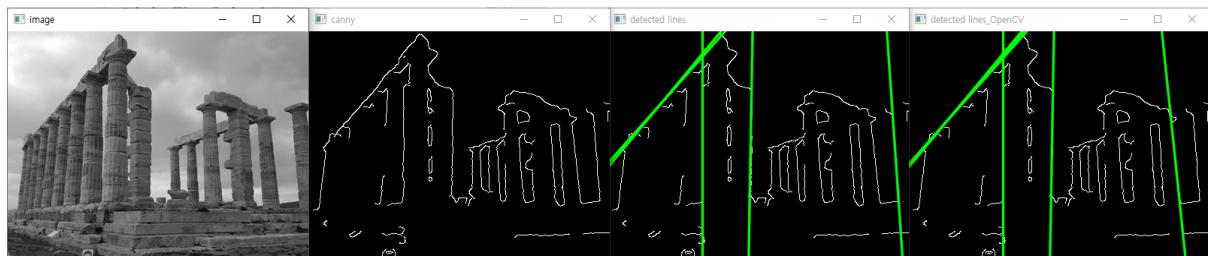
```

image = cv2.imread('07_images/hough.jpg', cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")
blur = cv2.GaussianBlur(image, (5, 5), 2, 2)
canny = cv2.Canny(blur, 100, 200, 5)

rho, theta = 1, np.pi / 180
lines1 = houghLines(canny, rho, theta, 80)
lines2 = cv2.HoughLines(canny, rho, theta, 80)
dst1 = draw_houghLines(canny, lines1, 7)
dst2 = draw_houghLines(canny, lines2, 7)

cv2.imshow("image", image)
cv2.imshow("canny", canny)
cv2.imshow("detected lines", dst1)
cv2.imshow("detected lines_OpenCV", dst2)
cv2.waitKey(0)

```



K-NN(k-Nearest Neighbors) Classification

02_mnist_kNN.py

```
import cv2, numpy as np
import os, gzip, pickle
from urllib.request import urlretrieve
# url을 통해 다운로드 가능하게 해주는 라이브러리
import matplotlib.pyplot as plt

def load_mnist(filename):
    if not os.path.exists(filename):
        print("Downloading...")
        link = "http://figshare.com/ndownloader/files/25635053"
        urlretrieve(link, filename)

    with gzip.open(filename, 'rb') as f:
        return pickle.load(f, encoding='latin1')

def graph_image(data, lable, title, nsample):
    plt.figure(num=title, figsize=(6, 9))
    rand_idx = np.random.choice(range(data.shape[0]), nsample)
    for i, id in enumerate(rand_idx):
        img = data[id].reshape(28, 28)
        plt.subplot(6, 4, i + 1), plt.axis('off'), plt.imshow(img, cmap='gray')
        plt.title('%s: %d' % (title, lable[id]))
    plt.tight_layout()
```

```

train_set, valid_set, test_set = load_mnist('mnist.pk1.gz')
# 압축 이미지 파일을 제공
train_data, train_label = train_set
test_data, test_label = test_set

# MNIST로드 데이터 크기 확인
print("train_set", train_set[0].shape)
print("valid_set", valid_set[0].shape)
print("test_set", test_set[0].shape)

print("training...")
knn = cv2.ml.KNearest_create() # KNearest_create는 학습을 시켜주는 기능.
knn.train(train_data, cv2.ml.ROW_SAMPLE, train_label)

nsample = 100 # 100개의 샘플 데이터
print("%d개 prediction..." % nsample)

_, resp, _, _ = knn.findNearest(test_data[:nsample], k=5)
accur = sum(test_label[:nsample] == resp.flatten()) / len(resp)

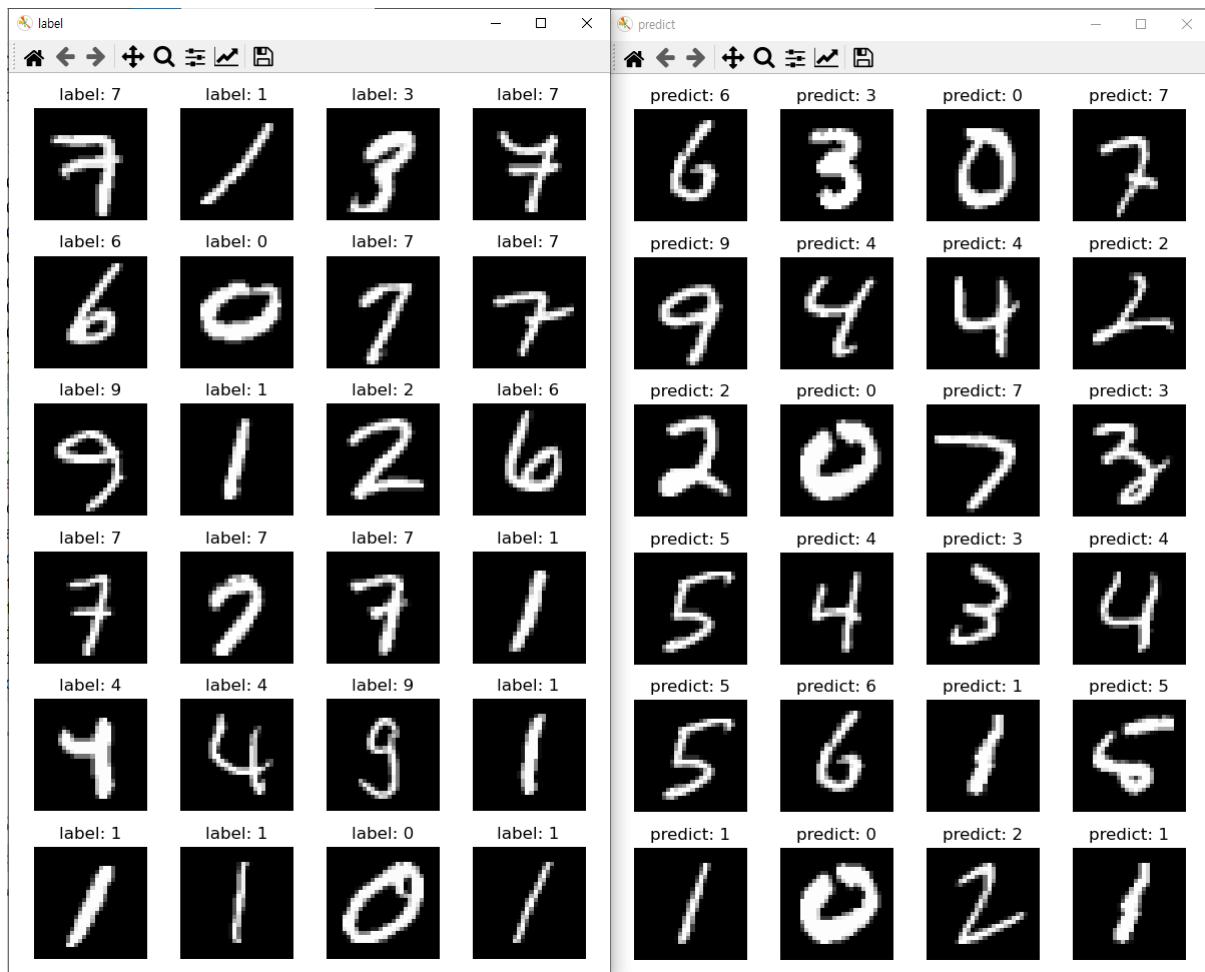
print("정확도=", accur * 100, "%")
graph_image(train_data, train_label, 'label', 24)
graph_image(test_data[:nsample], resp, 'predict', 24)
plt.show()

```

```

C:\ProgramData\Anaconda3\envs\tensorflow\lib\site-packages\cv2\__init__.py:13: UserWarning: numpy.ufunc size changed, may indicate binary incompatibility. Expected 32, got 48 bytes.
  warnings.warn(warning_msg)
Downloaded...
train_set (50000, 784)
valid_set (10000, 784)
test_set (10000, 784)
training...
100개 prediction...
정확도= 98.0 %

```



영상 워핑과 영상 모핑

워핑(warping)과 모핑(morphing)

아래는 워핑 기법을 사용한 영상



03_warping.py

```
import numpy as np, cv2

def morphing():
    h, w = image.shape[:2]
    dst = np.zeros((h, w), image.dtype)
    ys = np.arange(0, image.shape[0], 1)
    xs = np.arange(0, image.shape[1], 0.1) # 0.1 단위로 저장

    x1, x10 = pt1[0], pt1[0] * 10
    ratios = xs / x1
    ratios[x10:] = (w - xs[x10:]) / (w-x1)

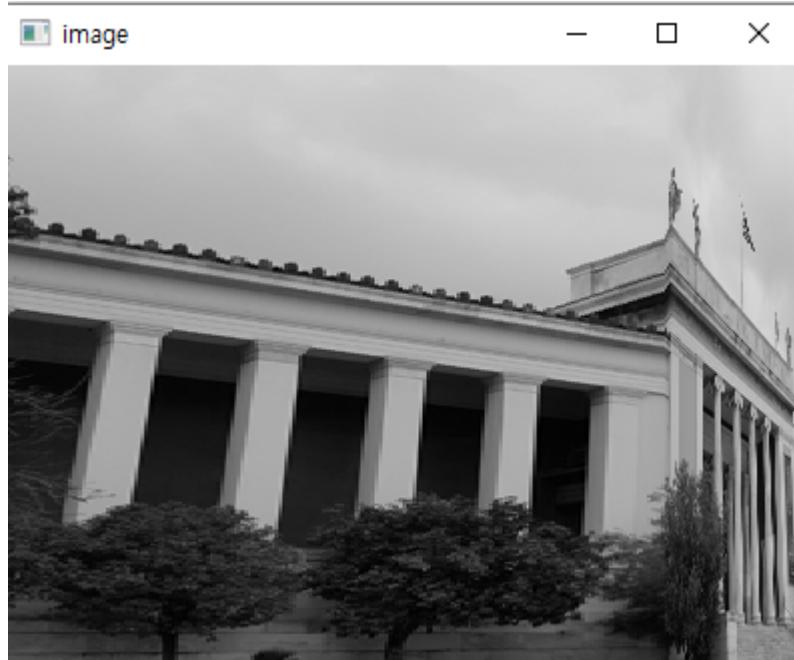
    dxs = xs + ratios * (pt2[0] - pt1[0])
    xs, dxs = xs.astype(int), dxs.astype(int)

    ym, xm = np.meshgrid(ys, xs)
    _, dxm = np.meshgrid(ys, dxs)
    dst[ym, dxm] = image[ym, xm]
cv2.imshow("image", dst)
```

```
def onMouse(event, x, y, flags, param):
    global pt1, pt2
    if event == cv2.EVENT_LBUTTONDOWN:
        pt1 = (x, y)
    elif event == cv2.EVENT_LBUTTONUP:
        pt2 = (x, y)
        morphing()
    elif event == cv2.EVENT_LBUTTONDBLCLK:
        pt1 = pt2 = (-1, -1)
        cv2.imshow("image", image)

image = cv2.imread("07_images/warp.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 에러")

pt1 = pt2 = (-1, -1)
cv2.imshow("image", image)
cv2.setMouseCallback("image", onMouse, 0)
# 마우스 클릭할 때마다 onMouse의 함수를 호출해주게 됨.
cv2.waitKey(0)
```



Left더블클릭을 하면 돌아옴

