HW 01 – REPORT

소속 :

학번 :

이름 :

1. 서론

실습 목표 및 이론적 배경 기술 (1~2페이지)

이 실습을 통해 파이썬에서 이미지를 처리하는 방법을 알아본다.

이 실습에서 쓰일 파이썬 라이브러리는 pillow와 numpy가 있다.

Pillow 라이브러리는 파이썬에서 이미지를 분석하고 처리해주는 라이브러리다. 이 라이브러리를 이용하여 이미지를 불러오고 크기를 조절하거나 크롭핑을 할 수 있다. Pillow 라이브러리의 Image 모듈의 주요 메소드는 표1과 같다.

표 1 파이썬 pillow 라이브러리의 Image 모듈의 주요 메소드

메소드	설명	사용 예
open()	이미지 파일을 로드한다	im=Image.open("test.jpg")
show()	이미지 파일을 보여준다 (이미지를 임시 저장한 후, 이미지를 보여줄 유틸리티를 호출한다.)	im.show()
save()	이미지를 원하는 포멧으로 저장한다.	im.save("test.jpg","JPEG")
crop()	이미지를 입력받은 (left, upper, right, lower) 좌표 튜플에 맞게 자른다. 왼쪽 위 코너의 좌표가 (0,0)이다.	im2=im.crop((100,100,400,400))
transpose()	이미지를 회전시킨다	im2=im.transpose (Image.Transpose.ROTATE_180)
convert()	이미지의 모드를 바꾼다(L, RGB)	im=im.convert("L")
filter()	이미지를 필터링한다	im=im.filter(ImageFilter.BLUR)

pillow 라이브러리의 주요 어트리뷰트는 표2와 같다.

표 2 pillow 라이브러리의 Image 모듈의 주요 어트리뷰트

어트리뷰트	설명	사용 예
format	해당 이미지 파일의 포멧(jpg,png 등)	im.format

size	해당 이미지의(폭, 높이)로 이루어진 튜플 (단위 : 픽셀)	im.size
mode	픽셀의 유형과 깊이를 정의 (L : 8비트 픽셀, 흑백, RGB : 3*8비트 픽셀, 트루 컬러 HSV : 3*8비트 픽셀, 색조 채도 값 색 공간 등)	im.mode

Numpy는 행렬이나 대규모 다차원 배열을 쉽게 처리할 수 있도록 해주는 파이썬 라이브러리이다. Numpy의 주요 메소드 및 어트리뷰트는 표3과 같다.

표 3 Numpy 라이브러리의 주요 기능

기능	설명	사용 예
np.array()	array를 만든다	x=np.array([1,2,3],[4,5,6])
x.size	전체 데이터의 개수	x.size
x.reshape()	array를 재배열한다	y=x.reshape(2,3,1)
np.dot()	행렬곱을 수행한다.	np.dot(x,y)
+,-,*,/(행렬 간)	행렬끼리 사칙연산을 수행한다.	x=np.array([1,2,3]) y=np.array([4,5,6]) z=x+y
+,-,*,/(행렬과 스칼라 간)	행렬의 각 값에 스칼라 값을 연산한다(broadcasting)	x=np.array([1,2,3]) y=3 z=x+y
np.argmax()	그 행렬의 최대값, 최소값이 있는 인덱스 번호를 반환한다. (axis=0 : 열 기준, axis=1 : 행 기준)	x=np.array([1,2,3],[4,5,6]) np.argmax(x,axis=1)

2. 본론

실습 내용 및 결과 기술 (2페이지 이상)

```
# import the packages we need for this assignment
from PIL import Image
import numpy as np
```

그림 1 import packages

그림1과 같이 코드 실행에 필요한 라이브러리(pillow, numpy)를 import 한다.

```
# open the test image
# Note: If you didn't launch Pyt
#       the file, chipmunk.png,
#       the argument to Image.op
im = Image.open('chipmunk.png')
```

그림 2 image load

open() 메소드를 사용하여 이 파이썬 파일과 동일한 디렉토리에 저장되어있는 chipmunk.png 사진 파일을 로드한다.

```
# display relevant Image class attrib
# pixel format and file format
print (im.size, im.mode, im.format)
```

그림 3 display information of the image

로드한 이미지의 사이즈(폭, 높이), 모드(L, RGB 등), 포멧(PNG, JPEG 등)를 출력한다(표2 참조) 그림3의 코드를 실행한 결과는 그림4와 같다.

```
PS D:\강의자료\4학년 1학기\컴퓨터비전개론\과제1> & C:/Users/0508t/miniconda3/envs/cv2024/python.exe "d:/강
hw1.py"
(750, 599) RGB JPEG
PS D:\강의자료\4학년 1학기\컴퓨터비전개론\과제1>
```

그림 4 그림3 코드 실행 결과. 이미지의 크기는 (750,599) 이미지는 컬러 이미지이며, JPEG 포맷임을 알 수 있다.

```
# display the image
im.show()
```

그림 5 code of displaying image

그림5의 코드를 이용하여 이미지를 화면에 띄운다. 이때 이미지를 임시 저장한 후 외부 유틸리티를 이용하여 화면에 띄운다. 결과는 그림6과 같다.

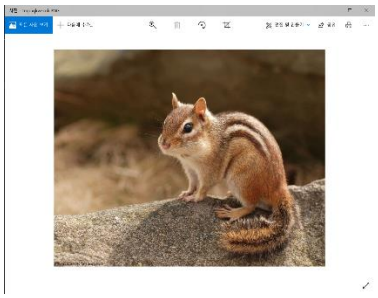

그림 6 displaying image

```
# convert the image to a black and white "luminance" greyscale image
im = im.convert('L')

# select a 100x100 sub region (containing the chipmunk's head)
im2 = im.crop((280,150,430,300))

# save the selected region
im2.save('chipmunk_head.png','PNG')

# PIL and numpy use different internal representations
# convert the image to a numpy array (for subsequent processing)
im2_array = np.asarray(im2)

# compute the average intensity
average = np.mean(im2_array)
```

그림 7 chipmunk 사진의 머리 부분만 잘라 저장하고, 이미지를 행렬로 바꾸어 처리하기

그림7의 코드를 이용하여 chipmunk 사진의 머리 부분만 잘라 저장하고, 이미지를 행렬로 바꾸어 처리한다. 먼저 convert('L') 부분을 통해 컬러이미지를 8비트 흑백 이미지로 바꾼다. 그리고 crop() 메소드를 이용하여 이미지를 자른다(crop 안에는 (left, upper, right, lower) 튜플이 들어감, 표1 참조). 그리고 자른 이미지를 PNG 형식으로 저장한다(im2.save()). 결과는 그림8과 같다.

이미지도 결국 픽셀 값이 들어있는 행렬이기 때문에 자른 이미지 im2를 np.asarray()를 이용하여 넘파이 행렬로 바꿀 수 있다. 그리고 이 넘파이 행렬을 이용하면 각 픽셀 값의 평균을 구할 수 있다(np.mean).

그림 8 그림7의 코드 실행 결과. chipmunk 머리 부분만 흑백으로 저장되었다.

```
# Note: we need to make a copy to change the values of ar
# np.asarray
im3_array = im2_array.copy()

# add 50 to each pixel value (clipping above at 255, the
# Note: indentation matters
for x in range(0,150):
    for y in range(0,150):
        im3_array[y,x] = min(im3_array[y,x] + 50, 255)

# convert the result back to a PIL image and save
im3 = Image.fromarray(im3_array)
im3.save('chipmunk_head_bright.png','PNG')
```

그림 9 chipmunk 머리 부분 흑백 사진 행렬을 copy하고, copy한 사진 행렬의 값을 조정하여 밝기를

올린다.

그림9의 코드는 그림7에서 처리하여 얻은 chipmunk 머리 부분 흑백 사진의 밝기를 올린 사진을 저장하는 코드이다. numpy의 copy() 메소드를 이용하여 그림7의 코드를 이용하여 얻은 넘파이 행렬(chipmunk 머리 부분 흑백 사진을 넘파이 배열로 바꾼 것이다)을 복사하고, 이 복사한 행렬 각 값에 50씩 더한다. (이때 이미지 모드가 L, 즉 8비트 흑백 이미지이므로, 255 이상 값은 255로 표시한다). 이 행렬을 다시 이미지로 바꾸고(Image.fromarray()) 저장한다(im3.save()). 결과는 그림10과 같다.

그림 10 그림9의 코드를 실행한 결과. 그림 8의 이미지보다 밝아졌다.

```
# again make a copy of the (original) 100x100 sub-region
im4_array = im2_array.copy()

# this time, reduce the intensity of each pixel by half
# Note: this converts the array to a float array
im4_array = im4_array * 0.5

# convert the array back to a unit8 array so we can write to a file
im4_array = im4_array.astype('uint8')

# convert the numpy array back to a PIL image and save
im4 = Image.fromarray(im4_array)
im4.save('chipmunk_head_dark.png','PNG')
```

그림 11 그림8 사진을 어둡게 하여 저장하는 코드

그림11의 코드는 그림8의 사진을 어둡게 하여 저장하는 코드. 그림8의 사진을 넘파이 배열로 바꾼 im2_array를 복사하여 각 배열의 값에 0.5를 곱한다. 이때 0.5를 곱하면 float array가 되므로, 8비트 배열로 바꿔주기 위해(즉, 이미지 모드 L로 바꾸기 위해) im4_array.astype('uint8') 코드를 이용하여 float array를 8비트 배열로 바꾼다. 그리고 fromarray 메소드를 이용하여 0.5를 곱한 배열을 이미지로 바꾸고 save 메소드를 이용하여 이미지를 저장한다.

그림 12 그림11 코드 실행결과. 그림8의 이미지보다 더 어두워졌다.

```
# let's generate our own image, a simple gradient test pattern
# make a 1-D array of length 256 with the values 0 - 255
grad = np.arange(0,256)

# repeat this 1-D array 256 times to create a 256x256 2-D array
grad = np.tile(grad,[256,1])

# convert to uint8 and then to a PIL image and save
im5 = Image.fromarray(grad.astype('uint8'))
im5.save('gradient.png','PNG')
```

그림 13 간단한 이미지 만들기

그림13의 코드는 넘파이로 행렬을 만들고, 그것을 Image의 fromarray 메소드를 사용하여 이미지를 만드는 코드이다. grad=np.arange(0,256)은 [0,1,2,3,...,256]의 배열을 생성한다.이 배열을 tile 메소드를 이용해 [0,1,2,3...,256] 배열을 256번 반복해 256*256 2-D array를 만든다. 이를 이미지로 변환하여 저장하면 오른쪽으로 갈수록 점점 밝아지는 이미지가 생성된다.

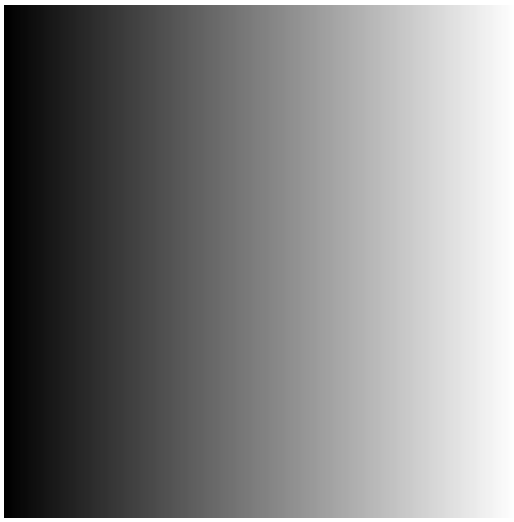

그림 14 그림13의 코드를 이용하여 생성한 이미지

3. 결론

토의 및 결론 (1페이지)

위 실습에서 chipmunk 이미지의 모드(L, RGB 등)를 바꾸고, 자르고, 밝게(또는 어둡게)해보았으며 간단한 이미지를 만들어보았다. 이를 통해, 파이썬에서 간단한 이미지 처리 방법을 알게 되었으며, 이미지도 결국, 픽셀 값의 배열임을 알게 되었다. 또한 crop 시 좌표가 왼쪽 상단이 (0,0) 임을 알게 되었다. 이미지도 픽셀 값의 배열이므로, 픽셀 값의 배열만 있으면 이를 이용하여 이미지를 만들 수 있음도 알게 되었다.

그리고 numpy를 이용하면 배열을 일반 파이썬 배열보다 편리하게 처리할 수 있음을 알았다. 특히 넘파이 배열에 스칼라 값을 곱하면, 해당 넘파이 배열의 모든 원소에 해당 스칼라 값이 곱해짐을 알게 되었다.

결론적으로, numpy와 pillow 라이브러리가 있으면, 파이썬에서 간단하게 이미지를 조작할 수 있음을 알게 되었다.

참고한 자료는 아래와 같다.

https://cord-ai.tistory.com/143

https://pillow.readthedocs.io/en/latest/handbook/tutorial.html#using-the-image-class

https://blockchainstudy.tistory.com/39

https://numpy.org/devdocs/user/basics.html