**영상처리**

**3장 과제 #3**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **학과/분반** | 컴퓨터공학과 / 02 | **학번** | 20210262 |
| **이름** | 김우혁 | **제출일** | 2025. 05. 08 (목) |
| **주제** | 관심 영역의 모자이크 영상 생성하기 | | |

**■ 목차**

**• 실습 목표**

**• 해결 과정 및 주요 코드 설명**

**• 실행 결과**

**• 느낀점**

**■ 실습 목표**

- 기본 설정:

기존 영상에 크기가 150x100이고 블록의 평균을 사용한 모자이크를 (0, 0) 위치에 출력한다.

- 사용자 지정 설정:

기존 영상에 사용자가 입력한 크기와 모자이크 종류, 위치를 반영한 모자이크를 출력한다.

**■ 해결 과정 및 주요 코드 설명**

0. 원하는 값 추출

|  |
| --- |
| def mosaic\_image(img, rect, size=(5,5), mtype=1):  dst = img.copy() # 원본 영상 복사해서 사용  start\_x, start\_y, end\_x, end\_y = rect  block\_w, block\_h = size  roi = dst[start\_y:end\_y, start\_x:end\_x]  h, w = roi.shape[:2]  img\_len = len(img.shape) # shape 속성이 길이   # 영역 사이즈를 블록 단위로 자를 수 있도록 크기 맞추기  h\_trim = h - (h % block\_h)  w\_trim = w - (w % block\_w)  roi = roi[:h\_trim, :w\_trim]   # 블록 개수  n\_blocks\_y = h\_trim // block\_h  n\_blocks\_x = w\_trim // block\_w  # 블록화  roi\_blocks = roi.reshape(n\_blocks\_y, block\_h, n\_blocks\_x, block\_w, 3) |

1. 블록의 평균값을 사용한 모자이크 처리 방법

|  |
| --- |
| block\_means = roi\_blocks.mean(axis=(1, 3), keepdims=True) roi\_mosaic = np.broadcast\_to(block\_means, roi\_blocks.shape).reshape(h\_trim, w\_trim) |

2. 블록의 최대값을 사용한 모자이크 처리 방법

|  |
| --- |
| block\_maxs = roi\_blocks.max(axis=(1, 3), keepdims=True) roi\_mosaic = np.broadcast\_to(block\_maxs, roi\_blocks.shape).reshape(h\_trim, w\_trim) |

3. 블록의 최솟값을 사용한 모자이크 처리 방법

|  |
| --- |
| block\_mins = roi\_blocks.min(axis=(1, 3), keepdims=True) roi\_mosaic = np.broadcast\_to(block\_mins, roi\_blocks.shape).reshape(h\_trim, w\_trim) |

4. 블록의 임의 위치 값을 사용한 모자이크 처리 방법

|  |
| --- |
| # 블록 개수  nby, nbx = h\_trim // block\_h, w\_trim // block\_w  # 블록 단위로 reshape roi\_blocks = roi.reshape(nby, block\_h, nbx, block\_w)  # (nby, nbx, block\_h, block\_w, 3) 로 transpose → 블록 단위 쉽게 다루기 blocks = roi\_blocks.transpose(0, 2, 1, 3, 4)  # 각 블록마다 랜덤한 y, x 좌표 생성 rand\_y = np.random.randint(0, block\_h, size=(nby, nbx)) rand\_x = np.random.randint(0, block\_w, size=(nby, nbx))  # 인덱스 메쉬 생성 y\_idx, x\_idx = np.meshgrid(np.arange(nby), np.arange(nbx), indexing='ij')  # 각 블록에서 무작위로 뽑은 픽셀 random\_pixels = blocks[y\_idx, x\_idx, rand\_y, rand\_x]  # 이 픽셀들을 각 블록에 반복해서 채움 filled\_blocks = np.broadcast\_to(random\_pixels[:, :, None, None, :], blocks.shape)  # 다시 원래 차원으로 reshape roi\_mosaic = filled\_blocks.transpose(0, 2, 1, 3, 4).reshape(h\_trim, w\_trim) |

컬러 영상의 경우 3차원이니 코드를 조금만 수정해주면 된다.

|  |
| --- |
| # 컬러 영상의 경우 코드 수정  roi\_blocks = roi.reshape(n\_blocks\_y, block\_h, n\_blocks\_x, block\_w, 3)  # ------------------------------------  roi\_mosaic=np.broadcast\_to(block\_means, roi\_blocks.shape).reshape(h\_trim, w\_trim, 3)  # ------------------------------------  roi\_mosaic = filled\_blocks.transpose(0, 2, 1, 3, 4).reshape(h\_trim, w\_trim, 3) |

**■ 실행 결과**

|  |
| --- |
| python mosaic\_image.py -i nature.jpg |
|  |

|  |
| --- |
| python mosaic\_image.py -i nature.jpg -s 200 300 -e 400 500 -z 20 50 -t 2 |
|  |

|  |
| --- |
| python mosaic\_image.py -i nature.jpg -s 100 200 -e 400 400 -z 10 10 -t 3 |
|  |

|  |
| --- |
| python mosaic\_image.py -i nature.jpg -s 100 200 -e 300 300 -z 20 5 -t 4 |
|  |

**■ 느낀점**

각 블록의 평균값, 최대, 최소값으로 모자이크 영상을 만드는 것 구현은 할 만했는데,  
랜덤 픽셀 값 하나를 고르는 과정이 좀 어려웠다.  
평소 같았으면 random.randrange()나 random.choice() 함수를 사용했으면 되는 건데,  
구한 roi\_blocks 값이 일반적인 리스트 값이 아니기 때문에, 이를 바로 사용하기 어려웠다.  
roi\_blocks는 NumPy 배열이기 때문에 단순히 random.choice()를 적용하는 것만으로는

해결할 수 없었다.  
이 문제를 해결하기 위해 블록 내에서 임의의 위치를 계산하고,  
np.random.randint()를 사용하여 각 블록마다 랜덤 픽셀 위치를 선택하는 방식으로

구현할 수 있었다.  
이 과정에서 배열을 다루는 방법에 대한 깊은 이해가 필요하다는 점을 느꼈다.  
이번 과제를 통해 NumPy 배열의 다차원 인덱싱을 적극적으로 활용할 수 있게 되어

유익했다.