

# Assignment 5

2024017201 최선웅

12034 수치해석

## 필요한 Package

```
pip install numpy  
pip install pillow
```

- Pillow 패키지: 이미지 입출력과 기본 핸들링
- NumPy 패키지: 픽셀별 계산과 보간법 같은 수치 연산을 수행하는 역할

## Main.py 코드

```
import numpy as np  
from PIL import Image  
  
def bilinear_interpolation(img, new_height, new_width):  
    """  
    이미지 배열 img를 (new_height, new_width)로 리샘플링 (양선형 보간)  
    """  
    orig_height, orig_width = img.shape[:2]  
  
    # 컬러/흑백 지원  
    if len(img.shape) == 3:  
        channels = img.shape[2]  
        resized = np.zeros((new_height, new_width, channels), dtype=np.uint8)  
    else:  
        channels = 1  
        resized = np.zeros((new_height, new_width), dtype=np.uint8)  
  
    # 스케일 비율
```

```

x_ratio = (orig_width - 1) / (new_width - 1) if new_width > 1 else 0
y_ratio = (orig_height - 1) / (new_height - 1) if new_height > 1 else 0

for i in range(new_height):
    for j in range(new_width):
        x = j * x_ratio
        y = i * y_ratio
        x1 = int(np.floor(x))
        y1 = int(np.floor(y))
        x2 = min(x1 + 1, orig_width - 1)
        y2 = min(y1 + 1, orig_height - 1)
        dx = x - x1
        dy = y - y1

        if channels == 1:
            top = (1 - dx) * img[y1, x1] + dx * img[y1, x2]
            bottom = (1 - dx) * img[y2, x1] + dx * img[y2, x2]
            value = (1 - dy) * top + dy * bottom
            resized[i, j] = int(value)
        else:
            for c in range(channels):
                top = (1 - dx) * img[y1, x1, c] + dx * img[y1, x2, c]
                bottom = (1 - dx) * img[y2, x1, c] + dx * img[y2, x2, c]
                value = (1 - dy) * top + dy * bottom
                resized[i, j, c] = int(value)

return resized

```

```

def main():
    # 1. 파일 읽기
    input_path = "image.jpg" # 입력 파일명
    img = Image.open(input_path)
    img_array = np.array(img)
    print(f"원본 해상도: {img_array.shape[0]} x {img_array.shape[1]}")

    # 2. 목표 해상도 입력
    new_height = int(input("변환할 세로 크기(M'): "))
    new_width = int(input("변환할 가로 크기(N'): "))

```

```

# 3. 리샘플링 실행
print("리샘플링 중...")
resized_img = bilinear_interpolation(img_array, new_height, new_width)

# 4. 결과 저장
result = Image.fromarray(resized_img)
output_path = "resampled_image.jpg"
result.save(output_path)
print(f"완료! 저장 경로: {output_path}")

if __name__ == "__main__":
    main()

```

해당 `main.py` 코드는 이미지 파일을 읽고, 사용자가 입력한 목표 해상도로 양선형 보간을 통해 리샘플링한 뒤, 결과 이미지를 저장하는 과정을 담당한다.

주요 흐름은 다음과 같다.

#### 1. 이미지 파일 읽기

`image.jpg` 파일을 불러와서 배열로 변환한다.

#### 2. 원본 해상도 출력

읽은 이미지의 세로, 가로 픽셀 수를 출력한다.

#### 3. 사용자 입력

새롭게 바꿀 세로( $M'$ )와 가로( $N'$ ) 해상도를 입력받는다.

#### 4. 리샘플링 실행:

`bilinear_interpolation` 함수를 사용해서 원하는 크기로 이미지를 리샘플링한다.

#### 5. 결과 저장

리샘플링된 이미지를 `resampled_image.jpg` 파일로 저장한다.

#### 6. 완료 메시지 출력

작업이 끝나면 결과 파일 경로를 출력한다.

## Resampled Image



기존 이미지는 다음과 같다. (1080 X 1920)



3배 Upscaling 한 이미지이다. (3240 X 3240)



3배 Downscaling 한 이미지이다. (360 X 640)

## 결과 분석





순서대로 기본, 3배 업스케일링, 3배 다운스케일링한 이미지의 확대 모습이다.

세 가지 변환된 이미지의 시각적 특성을 비교하면 다음과 같다.

### 1. 기본 이미지

- 특징: 선명하고 깔끔한 경계선, 명확한 텍스처 및 세부사항
- 해상도: 고해상도의 모든 디테일이 뚜렷하게 표현됨

### 2. 3배 업스케일링 이미지

- 특징: 블러 효과 관측, 경계선이 부드럽지만 뭉개진 느낌

- 관찰 포인트: 기존 이미지의 날카로운 엣지가 둥글고 부드러워짐
- 원인: 양선형 보간법이 주변 4개 픽셀의 가중치 평균을 취하기 때문에 고주파 성분(명확한 경계)이 자연스럽게 필터링되고 날카로움이 감소한다.

### 3. 3배 다운스케일링 이미지

- 특징: 계단 현상(Aliasing), 거칠고 뭉개진 모습, 모자이크 같은 효과
- 관찰 포인트: 매우 낮은 해상도로 축소되어, 세부사항이 거의 손실, 생상 블록이 크게 둑여 보임. 선명한 경계가 계단 형태로 표현됨
- 원인: 원본의 많은 픽셀 정보를 소수의 픽셀로 압축해야 하므로, 정보 손실이 필연적이고 보간된 값들이 대표성을 잃으면서 계단 패턴이 생김.

## 결론

- 업스케일링: 양선형 보간이 부드럽게 작동하지만, 원본에 없는 정보는 만들 수 없으므로, 블러 효과가 발생한다.
- 다운스케일링: 많은 정보를 버려야 하기 때문에 계단 현상과 모자이크 효과가 나타난다.
- 양선형 보간의 한계: 원본 해상도보다 훨씬 크거나 작은 변환에서는 품질 손실이 불가피하다.