수치해석

HW02

담당교수 : 심동규 교수님

학 번 : 2021202058

성 명 : 송채영

1. Introduction

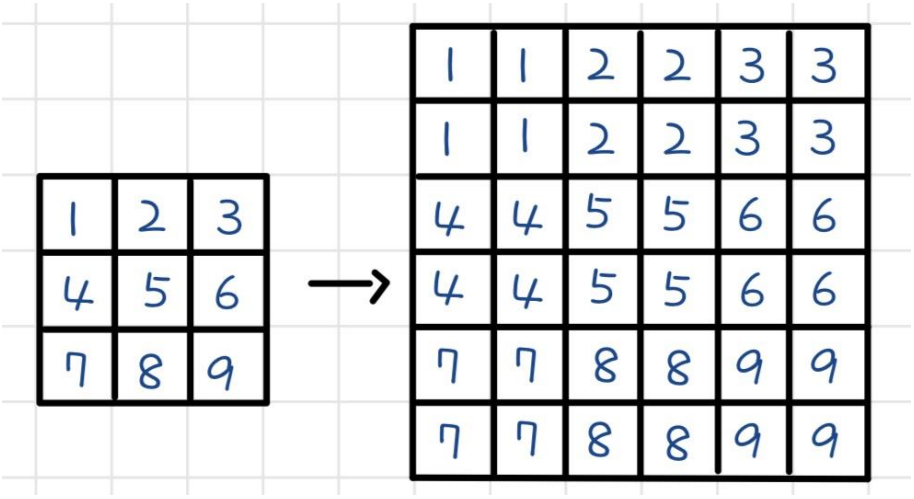
이번 과제는 128x128 크기의 이미지를 512x512 의 크기로 upscaling 하는 것이다. 이를 위해 4 가지의 interpolation method 를 사용하는데 다음과 같다. Nearest Neighbor Interpolation, Bilinear Interpolation, Bicubic Interpolation, Six-tab Linear Interpolation 를 통해 Interpolation 한 결과와 Ground Truth 폴더에 있는 512x512 크기의 이미지를 PSNR(Peak Signal-to-Noise Ratio)을 통해 평가해본다.

2. Interpolation Method

- Nearest Neighbor Interpolation

Nearest Interpolation 은 출력 이미지의 각 픽셀에 대해 가장 가까운 원시(원본) 픽셀을 찾아 해당 값을 할당하는 방법이다. 각 출력 픽셀은 가장 가까운 원시 픽셀의 값을 갖게 된다. 이때, 분수 주소(예: 1.5, 2.3)의 경우는 가장 가까운 유효한 화소(정수 주소)로 반올림한다.

이 방법의 단점은 blockness 또는 pixelation 현상이 발생할 수 있다는 점이다. 확대 작업을 수행할 때, 출력 이미지는 보다 큰 픽셀로 구성되므로, 각 픽셀은 원본 이미지의 픽셀을 단순히 복사하게 되는데, 결과적으로 세부 정보가 손실될 수 있다.

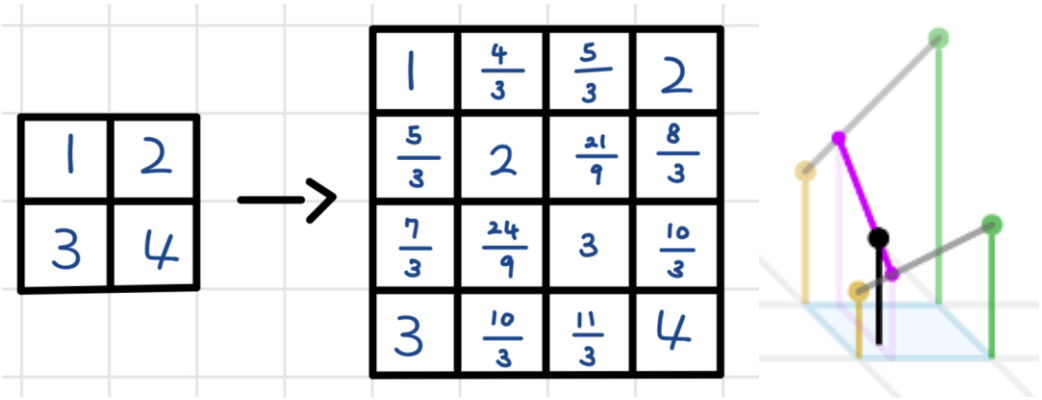


- Bilinear Interpolation

Bilinear Interpolation 은 이미지 처리 분야에서 매우 일반적으로 사용되는 보간 방법 중 하나이다. 새롭게 생성될 픽셀 값을 계산하기 위해 해당 위치 주변의 가장 가까운 픽셀 값 4 개를 사용하며, 이 네 개의 픽셀 값에 가중치를 곱한 후 합산하여 새로운 픽셀 값을 만든다. 또한 각 픽셀 위치에서 가로 방향과 세로 방향으로 각각

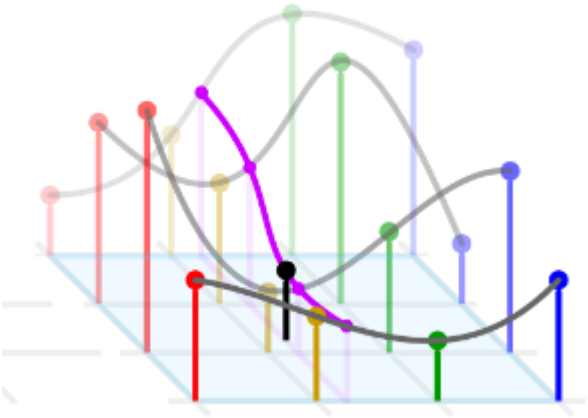
선형적이며 각 픽셀 주변의 가중치는 해당 위치와 가까운 픽셀까지의 거리에 정비례하다는 특징을 가지고 있다. 즉 더 가까운 픽셀일수록 더 높은 가중치를 가지며 먼 픽셀은 낮은 가중치를 가진다.

이 방법은 각 새로운 픽셀 값을 계산하기 위해 4 개의 가중치와 해당 픽셀 값을 사용하기 때문에 Nearest Neighbor Interpolation 에 비해 더 많은 계산을 필요로 한다.



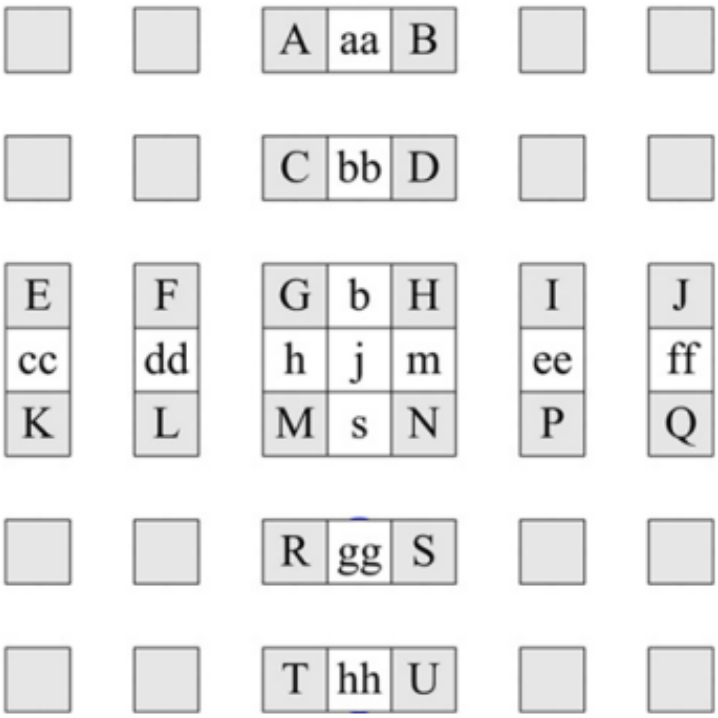
- Bicubic Interpolation

Bicubic Interpolation 은 주어진 데이터 포인트들 사이의 값을 추정하기 위해 2 차원 평면에서 사용되는 보간 기법 중 하나이다. 주변 픽셀들의 값을 이용하여 부드러운 곡선을 형성하고 누락된 데이터를 추정하는 데 사용한다. 4x4 개의 픽셀 그리드를 사용하여 주어진 픽셀 값 주위의 구간을 추정하는데, 이때 각 픽셀에는 가중치가 할당되며, 이 가중치는 보간을 수행하는데 사용한다. 일반적으로 이 가중치는 주변 픽셀 간의 거리나 차이에 따라 결정됩니다. 또한 Bicubic Interpolation 은 주로 다항식 보간 방법을 사용하는데, 일반적으로 3 차원 다항식이 사용되며, 이를 통해 보다 부드럽고 정확한 곡선을 형성할 수 있다.



- Six-tab Linear Interpolation

Six-tab linear interpolation 은 우선 가로 방향으로 6 개의 연속된 픽셀 값을 가지고 있으며, 이 픽셀 값들은 필터와 곱한다. 필터는 일반적으로 가중치를 가진 값을 나타내며, 이를 사용하여 가로 방향에서 새로운 값을 생성한다. 이어서 가로 6 개의 픽셀 값에서 수직 방향으로 6 개의 새로운 값을 생성한다. 수직 방향으로 생성된 6 개의 값을 이용해 중앙값을 계산하며 중앙과 픽셀 사이의 값의 평균을 내어 계산하는 방식을 의미하며 아래 사진처럼 동작한다.



3. Experiments

- Nearest Neighbor Interpolation

I 와 j 는 이미지에서의 픽셀 좌표이고, ORIGIN_HEIGHT 와 ORIGIN_WIDTH 는 input 이미지의 높이와 너비를, RESULT_HEIGHT 와 RESULT_WIDTH 는 결과 이미지의 높이와 너비를 말한다. 알고리즘에서 각 이미지의 픽셀 resultImage[i][j]를 구하는 수식에 대해 설명하겠다.

I * ORIGIN_HEIGHT / RESULT_HEIGH 와 J * ORIGIN_WIDTH / RESULT_WIDTH 는 이미지의 픽셀 resultImage[i][j]가 원본 이미지에서 어떤 위치의 값을 가져와야 하는지를 계산하는 부분으로 각각 세로 방향과 가로방향으로 원본 이미지에서의 위치를 계산한다. 즉 이미지의 각 픽셀은 원본 이미지에서 가장 가까운 위치에 있는 원본 이미지의 픽셀 값을 복사한다.

- Bilinear Interpolation

  originX 와 originY 는 이미지의 각 픽셀 위치(I,j)에 해당하는 원본이미지의 좌표를 나타내며 이미지의 픽셀 위치가 원본 이미지에서 어떤 위치에 대응되는지를 먼저 계산하였다. 이후 가장 가까운 4 개의 원본 이미지 픽셀 x1, x2, y1, y2 를 찾았다. 이때 x1 과 y1 은 왼쪽 상단 픽셀을, x2 과 y2 는 오른쪽 하단 픽셀에 해당한다.

  주변 네 개의 픽셀 값과 각 픽셀에 대한 가중치를 사용하여 대상 이미지의 각 픽셀 값을 계산하였다. 이때 dx 와 dy 는 originX 와 originY 의 정수 부분과 소수 부분의 차이를 말한다.

  value = (1.0 - dx) * (1.0 - dy) * originImage[y1][x1] + dx * (1.0 - dy) * originImage[y1][x2] + (1.0 - dx) * dy * originImage[y2][x1] + dx * dy * originImage[y2][x2]

- Bicubic Interpolation

  Cubic Polynomial Function 은 가중치를 계산하는 데 사용되며 t 값을 -1 에서 2 사이의 범위로 정규화하여 사용하였다. 만약 t 가 0 보다 작은 경우 양수로 만들어주며, t 가 1 보다 작은 경우 첫 번째 세그먼트에 대한 Cubic Polynomial 계산한다. t 가 2 보다 작은 경우 두 번째 세그먼트에 대한 Cubic Polynomial 계산한다. t 가 2 이상인 경우 가중치를 0 으로 설정한다.

  Interpolation 함수는 입력 이미지를 확대 또는 축소하여 결과 이미지를 생성한다. 우선 입력 이미지의 픽셀 위치와 결과 이미지의 픽셀 위치를 정규화(nolmalized) 하였다. 이후 정규화된 픽셀 위치를 기반으로 가장 가까운 원본 이미지 상의 픽셀 위치를 찾았으며 x0, y0 으로 나타냈다. Subpixel 위치를 계산하였는데, dx 와 dy 로 나타내었다. 마지막으로 가중치를 계산해주었다. 4x4 픽셀 그리드를 반복하여 결과 이미지의 각 픽셀을 계산하였고, 부분 픽셀 위치와 가장 가까운 픽셀 주변의 16 개의 픽셀을 사용하여 계산하였다. cubicPolynomial 함수를 사용하여 계산한 가중치를 이용해 결과 이미지의 각 픽셀 값을 보간하였다.

  위 구현에서 이용한 3 차 다항식은 다음과 같다.

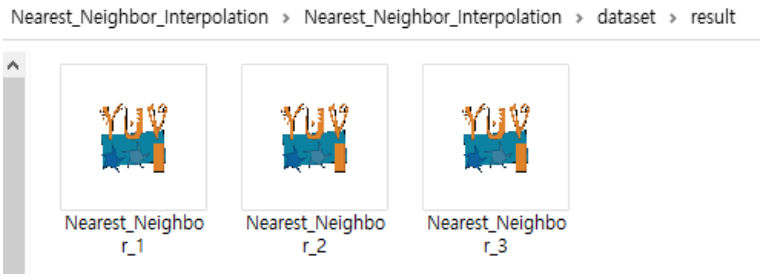
$$K(x) = \begin{cases} \frac{1}{2}(3|x|^3 - 5|x|^2 + 2), & |x| < 1 \\ \frac{1}{2}(-|x|^3 + 5|x|^2 - 8|x| + 4), & 1 \leq |x| < 2 \\ 0, & le|x| \geq 2 \end{cases},$$

- Six-tab Linear Interpolation

우선 Six-Tap Linear Interpolation 에 필요한 계수를 정의하였으며, 계수는 {1.0, -5.0, 20.0, 20.0, -5.0, 1.0}로 설정하였다. 이후 결과 이미지의 모든 픽셀을 순회하며 결과 이미지와 원본 이미지 간의 좌표 매핑하였다. 이때 입력 이미지와 결과 이미지의 크기 차이를 보정하기 위해 수행하였다. 다음으로 interpolation 에 사용할 데이터를 결정하고 난 후 six-tap interpolation 을 수행하였다. 주변 픽셀 값에 interpolation 계수를 적용해 결과를 계산해주었다. 계산한 interpolation 을 32.0 으로 나누어 normalize 해주어 픽셀 값의 범위를 0 에서 255 사이로 맞춰주었다.

4. Conclusion

- Nearest Neighbor Interpolation

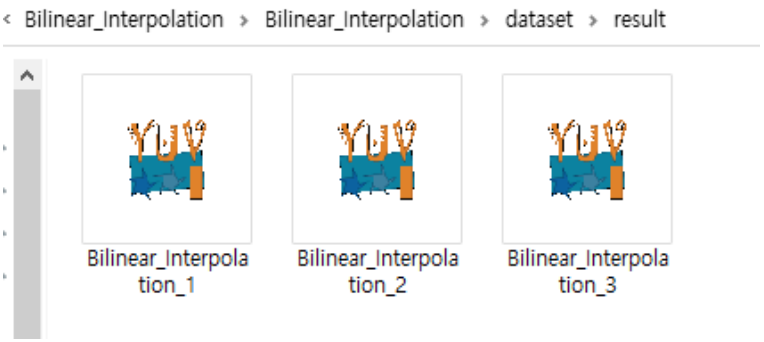


코드를 실행하면 result 폴더에 사진이 3 개 저장되는 것을 확인할 수 있다.

512x512 의 크기, color space 는 RGB, subsampling 은 4:0:0 으로 했을 때의 결과이다. Image1 은 Barbara, image2 는 Couple, image3 은 Lena 에 해당한다. 사진을 자세히 보면 모자이크 된 것처럼 화질이 좋지 않은 것을 볼 수 있었다.

Ground Truth dataset 과 result 를 PSNR 을 통해 평가한 결과이다. 3 번 이미지의 평가가 가장 좋게 나왔다.

- Bilinear Interpolation



코드를 실행하면 result 폴더에 사진이 3 개 저장되는 것을 확인할 수 있다.

512x512 의 크기, color space 는 RGB, subsampling 은 4:0:0 으로 했을 때의 결과이다. Image1 은 Barbara, image2 는 Couple, image3 은 Lena 에 해당한다. 1, 2 번 이미지에 비해 3 번 이미지가 가장 잘 나오는 것을 시각적으로 확인할 수 있지만 화질이 그렇게 좋다고 느껴지진 않았다.

```
Bilinear Interpolation PSNR for image 1: 22.71 dB
Bilinear Interpolation PSNR for image 2: 23.95 dB
Bilinear Interpolation PSNR for image 3: 27.17 dB
```

Ground Truth dataset 과 result 를 PSNR 을 통해 평가한 결과이다. 3 번 이미지의 평가가 가장 좋게 나온 것을 확인할 수 있다.

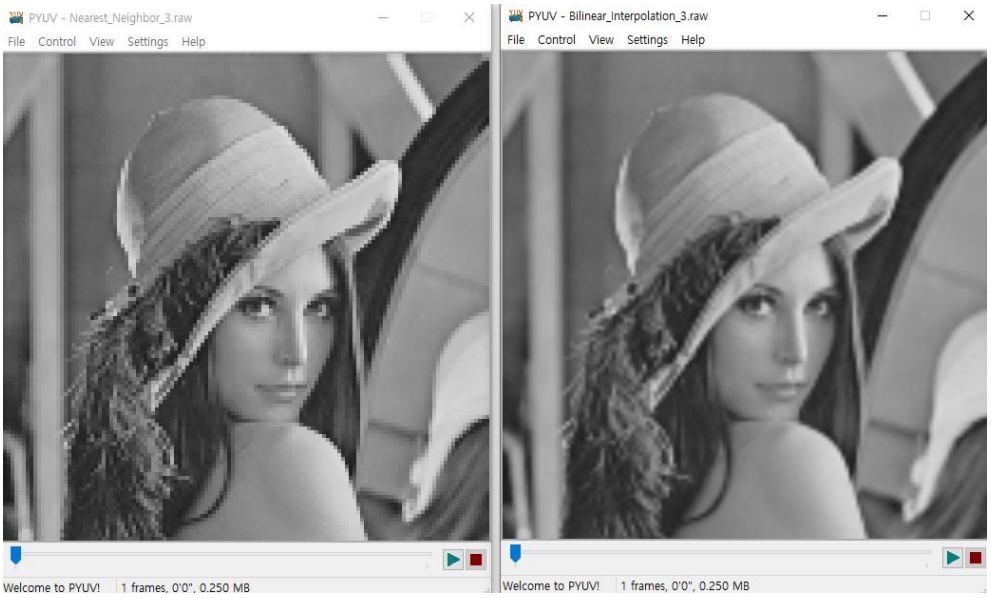
- Nearest Neighbor Interpolation 과 Bilinear Interpolation 의 결과를 비교해보았다. 왼쪽은 Nearest Neighbor Interpolation 에 해당하며 오른쪽은 Bilinear Interpolation 에 해당한다.


PYUV - Nearest_Neighbor_1.raw
File Control View Settings Help
Welcome to PYUV!
1 frames, 0'0", 0.250 MB


PYUV - Bilinear_Interpolation_1.raw
File Control View Settings Help
Welcome to PYUV!
1 frames, 0'0", 0.250 MB


PYUV - Nearest_Neighbor_2.raw
File Control View Settings Help
Welcome to PYUV!
1 frames, 0'0", 0.250 MB

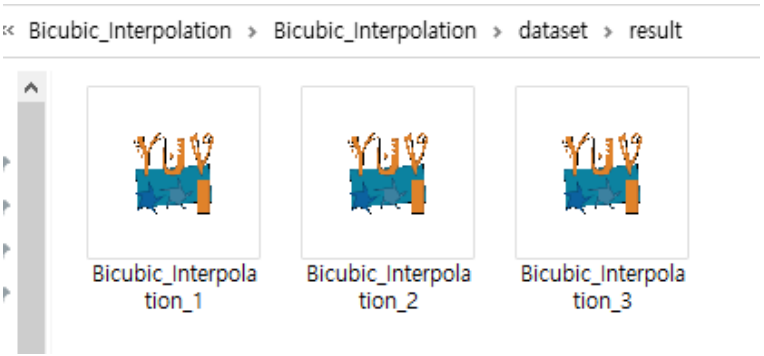

PYUV - Bilinear_Interpolation_2.raw
File Control View Settings Help
Welcome to PYUV!
1 frames, 0'0", 0.250 MB




```
Nearest Neighbor PSNR for image 1: 22.54 dB
Bilinear Interpolation PSNR for image 1: 22.71 dB
Nearest Neighbor PSNR for image 2: 24.03 dB
Bilinear Interpolation PSNR for image 2: 23.95 dB
Nearest Neighbor PSNR for image 3: 26.77 dB
Bilinear Interpolation PSNR for image 3: 27.17 dB
```

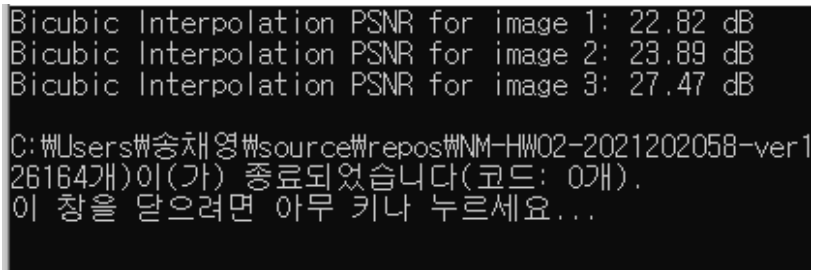
PSNR 을 확인해보면 Bilinear Interpolation 의 평가 결과가 더 좋게 나올 것이라고 예상했지만 2 번 사진은 Nearest Neighbor Interpolation 의 평가 결과가 더 좋은 것을 확인할 수 있다. 그러나 시각적으로 확인해보았을 때 Bilinear Interpolation 의 사진이 화질이 더 좋게 나온 것을 확인할 수 있었다.

- Bicubic Interpolation



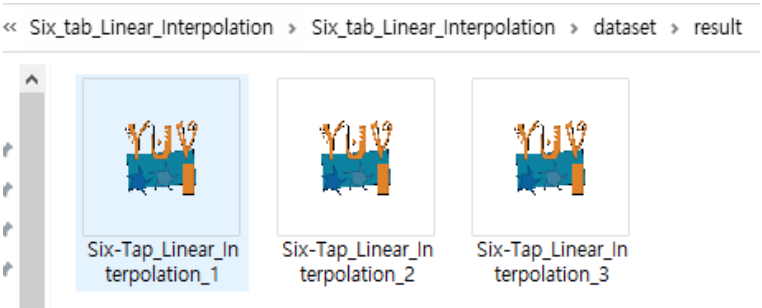
코드를 실행하면 result 폴더에 사진이 3 개 저장되는 것을 확인할 수 있다.

512x512 의 크기, color space 는 RGB, subsampling 은 4:0:0 으로 했을 때의 결과이다. Image1 은 Barbara, image2 는 Couple, image3 은 Lena 에 해당한다. 시각적으로 봤을 때는 Bilinear Interpolation 의 결과와 크게 다른 점이 없었다.

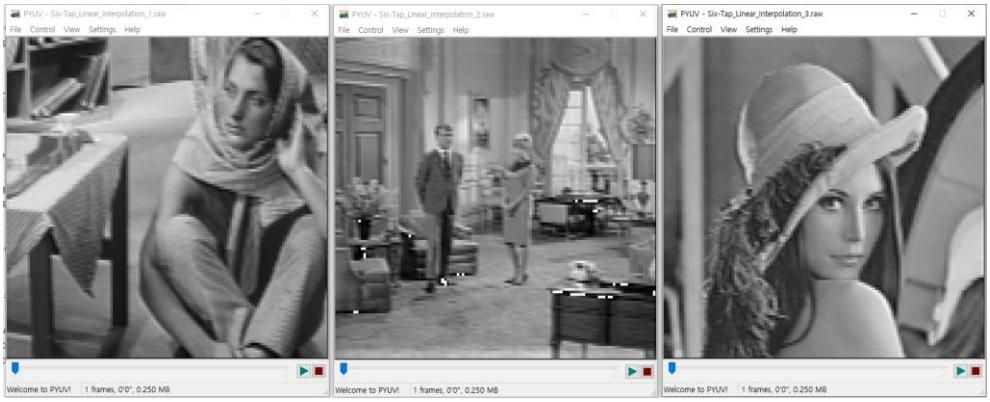


PSNR 을 확인해보면 1 번 사진과 3 번 사진은 Nearest Neighbor Interpolation 과 Bilinear Interpolation 의 PSNR 보다 결과가 좋게 나온 것을 확인할 수 있지만, 2 번 사진의 경우는 Nearest Neighbor Interpolation 과 Bilinear Interpolation 의 PSNR 보다 더 작게 나왔다. 2 번 사진이 다른 사진에 비해 특징도 많고 디테일 적인 부분이 많기 때문에 뭉개져서 나오는 듯한 느낌을 보이는 것 같았다.

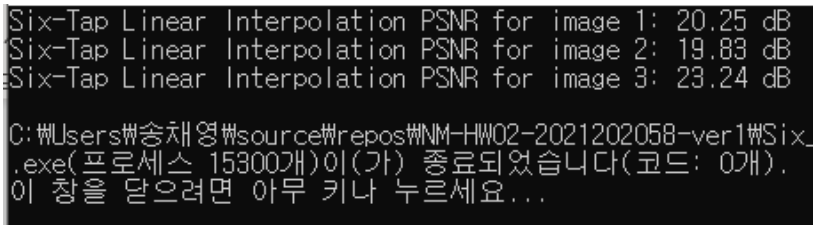
- Six-tab Linear Interpolation



코드를 실행하면 result 폴더에 사진이 3 개 저장되는 것을 확인할 수 있다.



512x512 의 크기, color space 는 RGB, subsampling 은 4:0:0 으로 했을 때의 결과이다. Image1 은 Barbara, image2 는 Couple, image3 은 Lena 에 해당한다. 시각적으로 봤을 때는 2 번의 사진에서 흰색 값들이 몇 개 나오는 것을 제외하고는 크게 다른 점은 없었다.



PSNR 을 확인해보면 전체적으로 다른 interpolation 의 결과보다 다 작게 나오는 것을 확인할 수 있었다. 눈으로 봤을 때는 큰 차이를 못 느꼈다.

전체적인 결과를 살펴보면 bicubic interpolation 의 PSNR 이 가장 높게 나왔다. Six-tap 은 구현에 있어 강의자료대로 진행해보려고 하였으나, 패딩하는 과정에서 무한 루프를 돌며 탈출하지 못하여 성능 측정을 제대로 해보지 못하였다. 따라서 코드를 다른 방향으로 구현해보았는데, 이 역시 성능이 좋게 나오지 않아 아쉬웠다. 잘 구현되었다고 가정하고 결과를 예상해보면 Nearest Neighbor Interpolation 의 결과가 가장 적게 나오고 그 다음으로 Bilinear Interpolation, 그 다음은 Six-tab linear Interpolation 그리고 마지막으로 Bicubic Interpolation 의 결과가 가장 좋게 나왔을 것 같다. Bicubic 과 Six-tab linear Interpolation 에 대한 개념도 부족했고, 그에 대한 검색 결과도 많지 않아 구현 하는 부분에 있어 아쉬움이 남는다.