

Digital Image Processing

Term Project



학과 : 전자공학과

학번 : 21611648

이름 : 유준상

담당교수 : 김성호

Table of Contents

Introduction

Main – Survey related technologies

Results – Matlab-based implementation

Conclusion

Reference

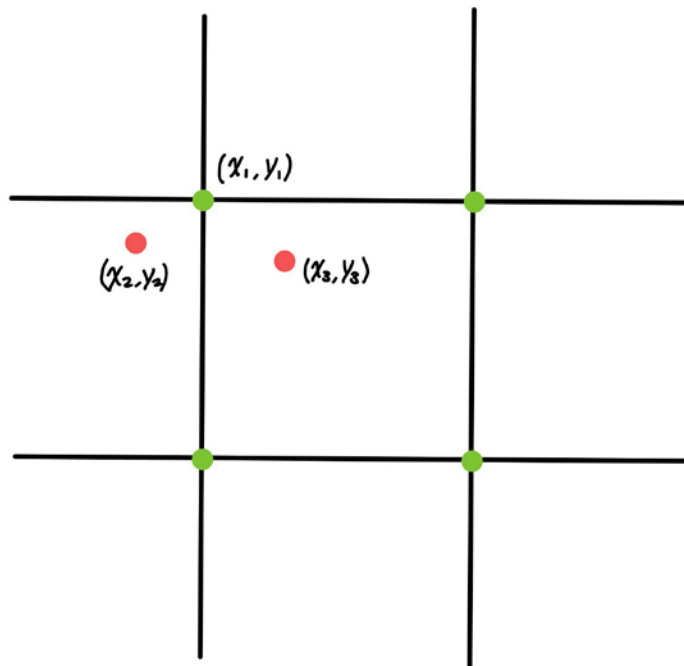
Introduction

본 레포트는 Image Super-resolution에 대한 Term project를 진행한 후 작성한 것이다. Image super-resolution은 저해상도 이미지를 고해상도 이미지로 변환시키는 것을 의미한다. Image Super-resolution이 필요한 이유는 다음과 같다. 예전에는 640x480 크기의 SD 급 이미지들의 크기를 대부분 사용해왔다. 하지만 최근에는 하드웨어의 발전으로 Full HD 급의 디스플레이가 가능해지고 많이 보급 되었기 때문에 과거의 이미지들의 크기를 변환해야 하는 중요한 문제가 있기 때문이다.

따라서 본 레포트에서는 본론에서 고전 기법인 Nearest Neighbor, Bilinear interpolation, Bicubic interpolation 세 기법을 먼저 설명한다. 그리고 SRMDNF라는 기법에 대해 설명한다. 이후 결과 파트에서 네 기법의 구현 코드와 결과 이미지, 평가 지표인 PSNR 값을 보인다. 마지막 결론에서는 네 기법을 비교하고, 디스커션, 향후 계획에 대해 말한다.

Survey related technologies

(1) NN(Nearest Neighbor)



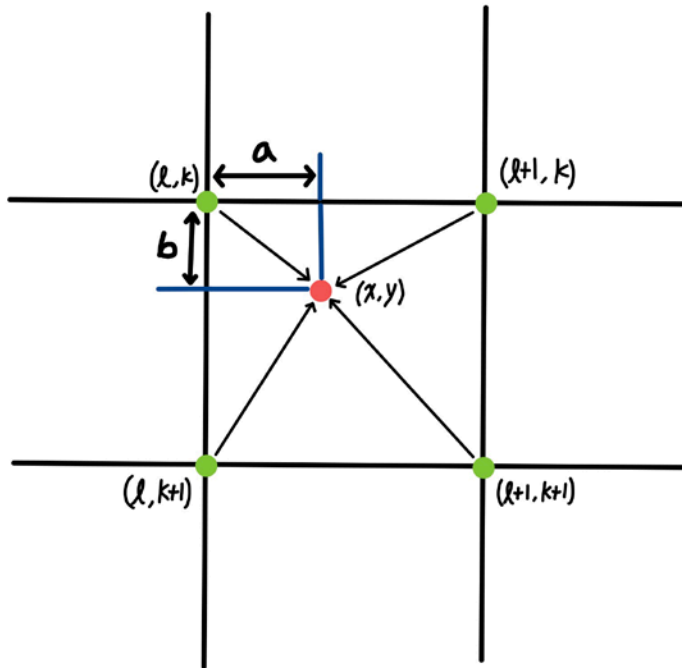
NN은 Nearest Neighbor(최근접 이웃)의 약자로, 특정 좌표(붉은 점)의 값을 구하고 싶을 때, 해당 좌표와 가장 가까운 거리의 좌표 값으로 가져오는 방법이다. 가까운 좌표로 가져오는 방법은 반올림을 사용한다.

위의 이미지를 예시로 수식을 사용하면 아래와 같다. f 는 해당 픽셀에서의 Intensity를 의미한다.

$$f_1(x_2, y_2) = f(\text{round}(x_2), \text{round}(y_2)) = f(x_1, y_1) \quad , \quad f_1(x_3, y_3) = f(\text{round}(x_3), \text{round}(y_3)) = f(x_1, y_1)$$

(x_2, y_2) 와 (x_3, y_3) 두 붉은 점 모두 좌표 위의 값 (x_1, y_1) 이 가장 가깝기 때문에 $f(x_1, y_1)$ 의 값을 가지게 된다.

(2) Bilinear



Nearest Neighbor 방법에서 가장 가까운 한 점을 고려했다면, Bilinear interpolation 방법은 인접

네 점을 고려하는 방법이다. 미지의 픽셀(붉은 점)의 값을 구하기 위해 인접한 네 개의 픽셀(초록

점)에 가깝고 먼 정도에 따라 가중치(weight)를 부여하여 곱하고 더하여 계산한다. 붉은 점과 가까운 점일 수록 큰 가중치를 받는다고 생각하면 된다.

수식을 세우기 전, 다음의 사항을 먼저 알아야 한다. $l = \text{floor}(x)$, $k = \text{floor}(y)$, $a = x - l$, $b = y - k$

을 의미한다. floor는 내림을 의미한다. 위의 이미지를 예시로 수식을 사용하면 아래와 같다.

$$\begin{aligned} f(x, y) = & (1 - a) * (1 - b) * f(l, k) + a * (1 - b) * f(l + 1, k) \\ & + (1 - a) * b * f(l, k + 1) + a * b * f(l + 1, k + 1) \end{aligned}$$

미지의 점 (x, y) 과 가까운 네 점 (l, k) , $(l+1, k)$, $(l, k+1)$, $(l+1, k+1)$ 에 가중치를 곱하여 intensity를 계산한다.

(3) Bicubic

위의 Bilinear interpolation 방법이 인접 4개의 픽셀을 사용했다면, Bicubic interpolation 방법은 16개의 픽셀을 고려하는 방법이다. 인접한 16개 픽셀의 값과 거리에 따른 가중치의 곱으로 결정한다. 더 많은 픽셀을 고려하기 때문에, Bicubic interpolation을 사용한 이미지는 Bilinear interpolation 방법을 사용한 이미지보다 더 매끄럽고 불필요한 아티팩트가 적게 보인다. Bicubic interpolation은 cubic interpolation을 x축 y축 각각 실행하여 구한다. 각 cubic interpolation은 인접한 4개의 픽셀 값을 사용하여 파라미터가 4개인 3차 방정식을 만든다. a는 16개($a_{00} \sim a_{33}$)의 계수를 의미한다.

cubic interpolation

1. $f(0, 0) = p(0, 0) = a_{00},$
2. $f(1, 0) = p(1, 0) = a_{00} + a_{10} + a_{20} + a_{30},$
3. $f(0, 1) = p(0, 1) = a_{00} + a_{01} + a_{02} + a_{03},$
4. $f(1, 1) = p(1, 1) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij}.$

x축 cubic interpolation(1~4) / y축 cubic interpolation(5~8) /

1. $f_x(0, 0) = p_x(0, 0) = a_{10},$
2. $f_x(1, 0) = p_x(1, 0) = a_{10} + 2a_{20} + 3a_{30},$
3. $f_x(0, 1) = p_x(0, 1) = a_{10} + a_{11} + a_{12} + a_{13},$
4. $f_x(1, 1) = p_x(1, 1) = \sum_{i=1}^3 \sum_{j=0}^3 a_{ij}i,$
5. $f_y(0, 0) = p_y(0, 0) = a_{01},$
6. $f_y(1, 0) = p_y(1, 0) = a_{01} + a_{11} + a_{21} + a_{31},$
7. $f_y(0, 1) = p_y(0, 1) = a_{01} + 2a_{02} + 3a_{03},$
8. $f_y(1, 1) = p_y(1, 1) = \sum_{i=0}^3 \sum_{j=1}^3 a_{ij}j.$

xy 부분 미분

1. $f_{xy}(0, 0) = p_{xy}(0, 0) = a_{11},$
2. $f_{xy}(1, 0) = p_{xy}(1, 0) = a_{11} + 2a_{21} + 3a_{31},$
3. $f_{xy}(0, 1) = p_{xy}(0, 1) = a_{11} + 2a_{12} + 3a_{13},$
4. $f_{xy}(1, 1) = p_{xy}(1, 1) = \sum_{i=1}^3 \sum_{j=1}^3 a_{ij} ij.$

위의 16개의 미분 식을 아래의 4개의 식으로 정리한다. x축에 대한 3차 방정식과 y축에 대한 3차 방정식을 곱해서 2D로 만들 수 있다.

$$p(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j$$

$$p_x(x, y) = \sum_{i=1}^3 \sum_{j=0}^3 a_{ij} i x^{i-1} y^j,$$

$$p_y(x, y) = \sum_{i=0}^3 \sum_{j=1}^3 a_{ij} x^i j y^{j-1},$$

$$p_{xy}(x, y) = \sum_{i=1}^3 \sum_{j=1}^3 a_{ij} i x^{i-1} j y^{j-1}$$

16개의 nearest neighbors를 사용해서 16개의 계수를 정하는 것이 포인트이다.

위의 식들을 행렬로 만들면 계산이 쉬워진다. $A^{-1}x = a$ 의 형태로 나타내기 위해서 아래의 행렬을 사용한다. x는 각 픽셀 값을 의미한다.

$$A^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{bmatrix}$$

16개의 계수 a를 구하면 아래의 식을 만들 수 있고, 아래 식을 완성해서 interpolation한 결과를 얻을 수 있다.

$$p(x, y) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}$$

(4) 평가 지표

① PSNR

PSNR은 Peak to Noise Ratio의 약자로 직역하면 최대 신호 대 잡음 비이다. 이미지, 비디오 압축 프로세스에서 화질을 평가할 때 사용되는 대표적인 성능 지표이다. 단위는 dB이고, 손실이 없는 이미지의 경우 MSE가 0이기 때문에 PSNR은 정의되지 않는다. 계산하는 식은 아래와 같다.

$$\begin{aligned} PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

여기서 MAX_I 는 해당 이미지의 최대값으로, 이미지의 채널에서 최대값에서 최소값을 빼서 구한다.

ex) 8bit image : $MAX_I = 255 - 0$.

MSE는 Mean Square Error의 약자로 평균 제곱 오차이다. 오차제곱의 합을 자유도로 나눈 것이다.

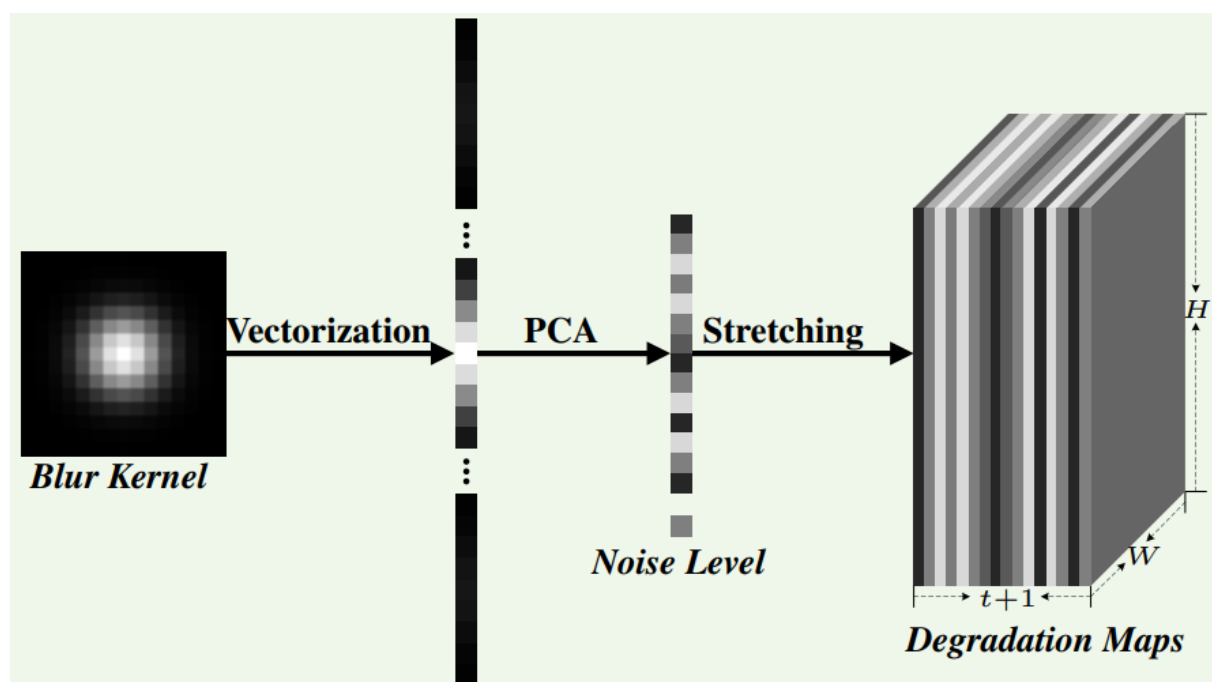
② 함수 처리 속도

프로그래밍을 할 때 성능과 함께 중요한 것이 함수의 처리 속도이다. 따라서 이미지 크기 변환 함수의 처리 속도를 계산하여 Results 파트에서 내장 함수와 구현 함수의 결과를 비교한다.

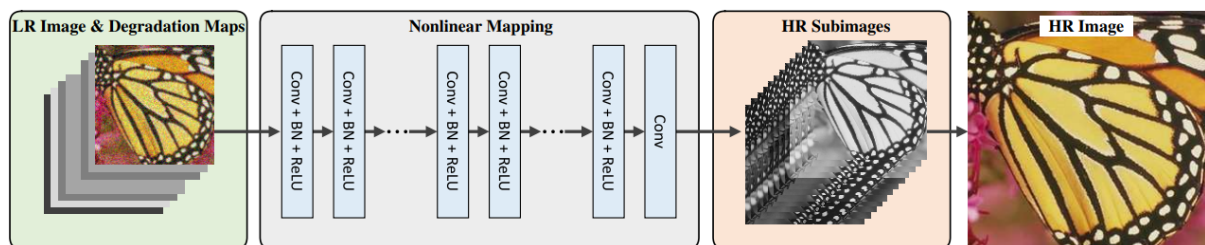
(5) 최신기술 - SRMDNF

SRMDNF는 저명한 컴퓨터 비전 국제 저널인 CVPR(Conference on Computer Vision and Pattern Recognition)에 2018년에 발표된 논문 "Learning a Single Convolutional Super-Resolution Network for Multiple Degradation – Kai Zhang(하얼빈 공업대학, 홍콩 이공대학, 알리바바 그룹) 등" 에서 제안된 기법이다. 여기서 NF는 Noise Free를 의미한다.

일반적인 CNN 기반의 SISR(Single Image Super-Resolution) 기법들은 대부분 저해상도 이미지가 고해상도 이미지에서 bicubicly down sampling 된 것으로 가정하므로 실제 저해상도 이미지가 이 가정을 따르지 않을 경우 성능이 저하되는 문제가 있다. 또한 여러 개의 저하된 이미지를 다루기 위해 단일 모델을 학습할 때 확장성이 부족하다. 이를 해결하기 위해서 저자는 blur kernel 과 noise level을 입력으로 사용하는 차원 스트레칭을 가진 SRMD를 제안했다. 결과적으로 공간적으로 저하된 region도 처리할 수 있다.



차원 스트레칭을 간략히 설명하자면, Blur Kernel을 사용하여 $(\cdot, 1)$ 의 크기로 Vectorization을 시키고 PCA를 한다. PCA는 위의 vector처럼 여러 데이터들이 분포되어 있을 때 이 분포의 주성분을 분석하여 가장 높은 성분부터 연관성이 없는 저 차원 축으로 만들어 주는 방법이다. 이렇게 얻은 것과 Noise level을 연산하여 Degradation Maps로 맵핑하는 것이다. 이렇게 하여 LR 이미지와 연결하여 CNN의 입력으로 사용하여 해결한다. 아래는 SRMD의 Architecture이며 그리 복잡하지 않으므로 자세한 설명은 하지 않는다.



마지막으로, SRMD보다 SRMDNF를 사용한 이유는 더 좋은 결과를 보이기 때문이다. 그 이유는 Bicubic interpolation된 Low Resolution 이미지는 노이즈의 복잡성을 악화시키기 때문에 학습할 때 어려움을 증가시키기 때문이다. SRMD와의 차이는 첫 번째 컨볼루션 필터에서 Noise level map과의 연결을 제거하고 새로운 학습으로 Fine-tuning하여 Noise Free인 모델을 학습시킨다.

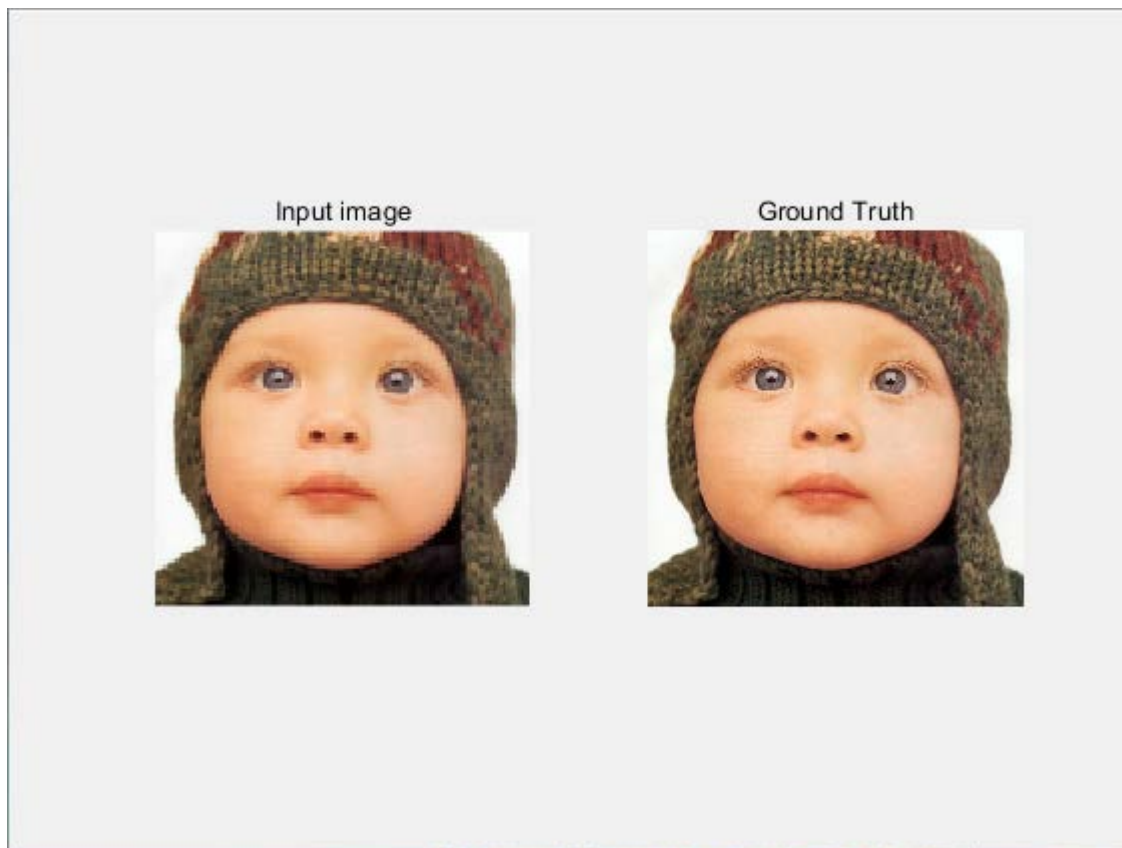
Table 1. Average PSNR and SSIM results for bicubic degradation on datasets Set5 [3], Set14 [54], BSD100 [33] and Urban100 [19]. The best two results are highlighted in red and blue colors, respectively.

Dataset	Scale Factor	Bicubic	SRCNN [9]	VDSR [24]	SRResNet [29]	DRRN [44]	LapSRN [27]	SRMD	SRMDNF
		PSNR / SSIM							
Set5	$\times 2$	33.64 / 0.929	36.62 / 0.953	37.56 / 0.959	—	37.66 / 0.959	37.52 / 0.959	37.53 / 0.959	37.79 / 0.960
	$\times 3$	30.39 / 0.868	32.74 / 0.908	33.67 / 0.922	—	33.93 / 0.923	33.82 / 0.922	33.86 / 0.923	34.12 / 0.925
	$\times 4$	28.42 / 0.810	30.48 / 0.863	31.35 / 0.885	32.05 / 0.891	31.58 / 0.886	31.54 / 0.885	31.59 / 0.887	31.96 / 0.893
Set14	$\times 2$	30.22 / 0.868	32.42 / 0.906	33.02 / 0.913	—	33.19 / 0.913	33.08 / 0.913	33.12 / 0.914	33.32 / 0.915
	$\times 3$	27.53 / 0.774	29.27 / 0.821	29.77 / 0.832	—	29.94 / 0.834	29.89 / 0.834	29.84 / 0.833	30.04 / 0.837
	$\times 4$	25.99 / 0.702	27.48 / 0.751	27.99 / 0.766	28.49 / 0.780	28.18 / 0.770	28.19 / 0.772	28.15 / 0.772	28.35 / 0.777
BSD100	$\times 2$	29.55 / 0.843	31.34 / 0.887	31.89 / 0.896	—	32.01 / 0.897	31.80 / 0.895	31.90 / 0.896	32.05 / 0.898
	$\times 3$	27.20 / 0.738	28.40 / 0.786	28.82 / 0.798	—	28.91 / 0.799	28.82 / 0.798	28.87 / 0.799	28.97 / 0.803
	$\times 4$	25.96 / 0.667	26.90 / 0.710	27.28 / 0.726	27.58 / 0.735	27.35 / 0.726	27.32 / 0.727	27.34 / 0.728	27.49 / 0.734
Urban100	$\times 2$	26.66 / 0.841	29.53 / 0.897	30.76 / 0.914	—	31.02 / 0.916	30.82 / 0.915	30.89 / 0.916	31.33 / 0.920
	$\times 3$	24.46 / 0.737	26.25 / 0.801	27.13 / 0.828	—	27.38 / 0.833	27.07 / 0.828	27.27 / 0.833	27.57 / 0.840
	$\times 4$	23.14 / 0.657	24.52 / 0.722	25.17 / 0.753	—	25.35 / 0.758	25.21 / 0.756	25.34 / 0.761	25.68 / 0.773

Results : Matlab-based implementation

본 프로젝트의 Results에서는 Nearest Neighbor, Bilinear interpolation, Bicubic interpolation 세 방법에 대해서 ① Code, ② Result image, ③ PSNR 및 처리속도 세 가지로 본 프로젝트의 결과를 나타낸다. 먼저, ① Code 에서는 사용한 구현 함수를 보인다. 간단한 설명은 주석으로 대체한다. ② Result image 에서는 원본 이미지와 매트랩 내장 함수인 imresize를 사용하여 얻은 결과 이미지와 구현한 함수로 얻은 결과 이미지를 비교한다. ③ PSNR 및 처리속도 에서는 평가 지표인 PSNR 값과 처리 속도 면에서 내장 함수와 구현 함수간의 차이를 비교한다.

1. Small(input) / Ground Truth



(1) Nearest Neighbor

① Code

[NN.m]

```
%% jsyoo
function out = NN(image, scale)
    % 입력 이미지로부터 R, G, B 각 채널 추출
    R = image(:,:,1); % 이미지의 Red 채널 추출
    G = image(:,:,2); % 이미지의 Green 채널 추출
    B = image(:,:,3); % 이미지의 Blue 채널 추출

    % 입력 이미지의 크기 구하기
    [H,W] = size(R);

    % Resize한 새로운 이미지의 크기를 변수에 할당
    Hn = ceil(H * scale);
    Wn = ceil(W * scale);

    Rn = zeros(Hn,Wn,'uint8');
    Gn = zeros(Hn,Wn,'uint8');
    Bn = zeros(Hn,Wn,'uint8');

    % 결과 이미지의 각 픽셀 별로 NN 실행
    if scale > 1
        for i = 1:Hn
            for j = 1:Wn
                % 입력 이미지의 좌표로 가서 가까운 값 할당
                r = ceil(i/scale); c = ceil(j/scale);
                % R, G, B 각 채널 별로 값 할당
                Rn(i,j) = R(r,c);
                Gn(i,j) = G(r,c);
                Bn(i,j) = B(r,c);
            end
        end
    else
        for i = 1:Hn
            for j = 1:Wn
                % 입력 이미지의 좌표로 가서 가까운 값 할당
                r = floor(i/scale); c = floor(j/scale);
                % R, G, B 각 채널 별로 값 할당
                Rn(i,j) = R(r,c);
                Gn(i,j) = G(r,c);
                Bn(i,j) = B(r,c);
            end
        end
    end

    % R, G, B 컬러 3채널을 결과 이미지로 결합시키기
    out = cat(3, Rn, Gn, Bn);
```

end

② Result image

2. 내장 함수



3. 구현 함수



③ PSNR 및 처리속도

	내장 NN	내장 처리속도	구현 NN	구현 처리속도
Baby	27.9239	0.008630	27.9239	0.003810
Bird	25.4156	0.020225	25.4156	0.004557
Butterfly	19.0549	0.002694	19.0549	0.001555
Head	27.9197	0.001956	27.9197	0.001584
Woman	23.0855	0.002498	23.0855	0.002253
Average	24.6799	0.0072006	24.6799	0.0027518

➔ 내장 NN과 구현 NN은 PSNR값은 같지만, 처리속도가 짧을수록 좋기 때문에 구현 NN 함수가 더 좋다고 볼 수 있다.

(2) Bilinear

① Code

[Bilinear.m]

```
%% jsyoo
function out = Bilinear(image, scale)
    % 입력 이미지로부터 R, G, B 각 채널 추출

    R = image(:,:,1); % 이미지의 Red 채널 추출
    G = image(:,:,2); % 이미지의 Green 채널 추출
    B = image(:,:,3); % 이미지의 Blue 채널 추출

    % 입력 이미지의 크기 구하기
    [H,W] = size(R);

    % Resize한 새로운 이미지의 크기를 변수에 할당
    Hn = ceil(H * scale);
    Wn = ceil(W * scale);

    Rn = zeros(Hn,Wn,'uint8');
    Gn = zeros(Hn,Wn,'uint8');
    Bn = zeros(Hn,Wn,'uint8');

    % 결과 이미지의 각 픽셀 별로 Bilinear 실행
    for i = 1:Hn
        x = (i/scale) + (0.5 * (1 - 1/scale));
        for j = 1:Wn
            y = (j/scale) + (0.5 * (1 - 1/scale));

            y(y < 1) = 1;
            if y >= W
                y = W;
                k = floor(y) - 1;
            else
                k = floor(y);
            end

            x(x < 1) = 1;
            if x >= H
                x = H;
                l = floor(x) - 1;
            else
                l = floor(x);
            end

            R1 = (k + 1 - y)*R(l,k) + (y - k)*R(l,k + 1);
            R2 = (k + 1 - y)*R(l + 1,k) + (y - k)*R(l + 1,k + 1);
            Rn(i,j) = (1 + 1 - x)*R1 + (x - 1)*R2;

            G1 = (k + 1 - y)*G(l,k) + (y - k)*G(l,k + 1);
            G2 = (k + 1 - y)*G(l + 1,k) + (y - k)*G(l + 1,k + 1);
```



```

        Gn(i,j) = (1 + 1 - x)*G1 + (x - 1)*G2;

        B1 = (k + 1 - y)*B(1,k) + (y - k)*B(1,k + 1);
        B2 = (k + 1 - y)*B(1 + 1,k) + (y - k)*B(1 + 1,k + 1);
        Bn(i,j) = (1 + 1 - x)*B1 + (x - 1)*B2;
    end
end
% R, G, B 컬러 3채널을 결과 이미지로 결합시키기
out = cat(3, Rn, Gn, Bn);
end

```

② Result image

2. 내장 함수



3. 구현 함수



③ PSNR 및 처리속도

	내장 Bilinear	내장 처리속도	구현 Bilinear	구현 처리속도
Baby	29.4078	0.029585	29.4084	0.387044
Bird	26.8473	0.009707	26.8459	0.128964
Butterfly	19.9601	0.003496	19.9595	0.126831
Head	28.5458	0.002319	28.5450	0.101501
Woman	24.2150	0.001804	24.2144	0.111633
Average	25.7952	0.0093822	25.7946	0.1711946

➔ 구현 Bilinear보다 내장 Bilinear가 PSNR값이 더 높다. 그리고, 처리속도 면에서는 내장 Bilinear가 훨씬 짧다.

(3) Bicubic

① Code

[cubic.m]

```
%% jsyoo
function f = cubic(x)
    % 절댓값 구하기

    abs_x = abs(x); % x의 절댓값을 abs_x에 저장
    abs_x2 = abs_x.^2; % abs_x의 제곱을 abs_x2에 저장
    abs_x3 = abs_x.^3; % abs_x의 세제곱을 abs_x3에 저장
    % cubic convolution 진행
    f = (1.5*abs_x3 - 2.5*abs_x2 + 1) .* (abs_x <= 1) + ...
        (-0.5*abs_x3 + 2.5*abs_x2 - 4*abs_x + 2) .* ((1 < abs_x) & (abs_x <=
2));
end
```

[Bicubic.m]

```
%% jsyoo
function out = Bicubic(image, scale)
    % 입력 이미지로부터 R, G, B 각 채널 추출
    R = image(:,:,1); % 이미지의 Red 채널 추출
    G = image(:,:,2); % 이미지의 Green 채널 추출
    B = image(:,:,3); % 이미지의 Blue 채널 추출

    % 입력 이미지의 크기 구하기
    [H,W] = size(R);

    % Resize한 새로운 이미지의 크기를 변수에 할당
    Hn = ceil(H * scale);
    Wn = ceil(W * scale);

    Rn = zeros(Hn,Wn,'uint8');
    Gn = zeros(Hn,Wn,'uint8');
    Bn = zeros(Hn,Wn,'uint8');

    % 결과 이미지의 각 픽셀 별로 Bicubic 실행
    for y_out = 1:Hn
        dy = (y_out/scale) + (0.5 * (1 - 1/scale));

        y1 = floor(dy) - 1;
        y1(y1 < 1) = 1;

        y2 = y1 + 1;
        y3 = y2 + 1;
        y4 = y3 + 1;
```

```

% Height 경계 내에서 커널 유지하기
if y4 >= H
    y4 = H;
    y3 = y4 - 1;
    y2 = y3 - 1;
    y1 = y2 - 1;
end

for x_out = 1:Wn
    dx = (x_out/scale) + (0.5 * (1 - 1/scale));

    x1 = floor(dx) - 1;
    x1(x1 < 1) = 1;

    x2 = x1 + 1;
    x3 = x2 + 1;
    x4 = x3 + 1;

    % Width 경계 내에서 커널 유지하기
    if x4 >= W
        x4 = W;
        x3 = x4 - 1;
        x2 = x3 - 1;
        x1 = x2 - 1;
    end

    % cubic interpolatio을 진행
    % x-axis
    cc_x1 = double(cubic(dx - x1));
    cc_x2 = double(cubic(dx - x2));
    cc_x3 = double(cubic(dx - x3));
    cc_x4 = double(cubic(dx - x4));

    % x축 계산한 것 다 더하기
    cc_X = double(cc_x1 + cc_x2 + cc_x3 + cc_x4);
    % y-axis
    cc_y1 = double(cubic(dy - y1));
    cc_y2 = double(cubic(dy - y2));
    cc_y3 = double(cubic(dy - y3));
    cc_y4 = double(cubic(dy - y4));

    % y축 계산한 것 다 더하기
    cc_Y = double(cc_y1 + cc_y2 + cc_y3 + cc_y4);

    % R, G, B 각 채널 별로 구한 cubic 값을 곱하여서
    % Bicubic interpolation 진행하기
    % Red Channel
    R1 = cc_x1*double(R(y1,x1))/cc_X + cc_x2*double(R(y1,x2))/cc_X +
    cc_x3*double(R(y1,x3))/cc_X + cc_x4*double(R(y1,x4))/cc_X;
    R2 = cc_x1*double(R(y2,x1))/cc_X + cc_x2*double(R(y2,x2))/cc_X +
    cc_x3*double(R(y2,x3))/cc_X + cc_x4*double(R(y2,x4))/cc_X;
    R3 = cc_x1*double(R(y3,x1))/cc_X + cc_x2*double(R(y3,x2))/cc_X +
    cc_x3*double(R(y3,x3))/cc_X + cc_x4*double(R(y3,x4))/cc_X;
    R4 = cc_x1*double(R(y4,x1))/cc_X + cc_x2*double(R(y4,x2))/cc_X +
    cc_x3*double(R(y4,x3))/cc_X + cc_x4*double(R(y4,x4))/cc_X;
    Rn(y_out,x_out) = cc_y1*R1/cc_Y + cc_y2*R2/cc_Y + cc_y3*R3/cc_Y +

```

```

cc_y4*R4/cc_Y;
    % Green Channel
    G1 = cc_x1*double(G(y1,x1))/cc_X + cc_x2*double(G(y1,x2))/cc_X +
cc_x3*double(G(y1,x3))/cc_X + cc_x4*double(G(y1,x4))/cc_X;
    G2 = cc_x1*double(G(y2,x1))/cc_X + cc_x2*double(G(y2,x2))/cc_X +
cc_x3*double(G(y2,x3))/cc_X + cc_x4*double(G(y2,x4))/cc_X;
    G3 = cc_x1*double(G(y3,x1))/cc_X + cc_x2*double(G(y3,x2))/cc_X +
cc_x3*double(G(y3,x3))/cc_X + cc_x4*double(G(y3,x4))/cc_X;
    G4 = cc_x1*double(G(y4,x1))/cc_X + cc_x2*double(G(y4,x2))/cc_X +
cc_x3*double(G(y4,x3))/cc_X + cc_x4*double(G(y4,x4))/cc_X;
    Gn(y_out,x_out) = cc_y1*G1/cc_Y + cc_y2*G2/cc_Y + cc_y3*G3/cc_Y +
cc_y4*G4/cc_Y;
    % Blue Channel
    B1 = cc_x1*double(B(y1,x1))/cc_X + cc_x2*double(B(y1,x2))/cc_X +
cc_x3*double(B(y1,x3))/cc_X + cc_x4*double(B(y1,x4))/cc_X;
    B2 = cc_x1*double(B(y2,x1))/cc_X + cc_x2*double(B(y2,x2))/cc_X +
cc_x3*double(B(y2,x3))/cc_X + cc_x4*double(B(y2,x4))/cc_X;
    B3 = cc_x1*double(B(y3,x1))/cc_X + cc_x2*double(B(y3,x2))/cc_X +
cc_x3*double(B(y3,x3))/cc_X + cc_x4*double(B(y3,x4))/cc_X;
    B4 = cc_x1*double(B(y4,x1))/cc_X + cc_x2*double(B(y4,x2))/cc_X +
cc_x3*double(B(y4,x3))/cc_X + cc_x4*double(B(y4,x4))/cc_X;
    Bn(y_out,x_out) = cc_y1*B1/cc_Y + cc_y2*B2/cc_Y + cc_y3*B3/cc_Y +
cc_y4*B4/cc_Y;
    end
end
    % R, G, B 컬러 3채널을 결과 이미지로 결합시키기
    out = cat(3, Rn, Gn, Bn);
end

```

② Result image

2. 내장 함수



3. 구현 함수



③ PSNR 및 처리속도

	내장 Bicubic	내장 처리속도	구현 Bicubic	구현 처리속도
Baby	30.3700	0.010138	30.3738	0.650132
Bird	28.0513	0.006767	28.0525	0.239635
Butterfly	20.8895	0.001826	20.8928	0.166594
Head	28.9396	0.001708	28.9421	0.183340
Woman	25.1118	0.001897	25.1185	0.241769
Average	26.67244	0.0044672	26.67594	0.296294

➔ 내장 Bicubic보다 구현 Bicubic이 PSNR값이 더 높다. 하지만, 처리속도 면에서는 내장 Bicubic이 더 짧은 시간이 걸리기 때문에 더 좋다고 볼 수 있다.

★ 위의 세 기법(Nearest Neighbor, Bilinear interpolation, Bicubic interpolation)

을 비교하기 위해 수행한 코드

[compare.m]

```
%% jsyoo
%% Load input image & ground truth image
clear; close all; clc;
% 확대할 입력 이미지 불러오기
image='woman';
I=imread(['./small/[크기변환]',image,'.png']);
figure(1); subplot(1,2,1); imshow(I); title('Input image');
% 이미지 저장할 경로 지정
folderResult = 'result_figure';
if ~exist(folderResult,'file') % results 폴더가 있는지 확인하고 없으면
    mkdir(folderResult); % 폴더 만들기
end
% Ground Truth 이미지 불러오기
Igt=imread(['./gt/',image,'.png']);
subplot(1,2,2); imshow(Igt); title('Ground Truth');
disp('=====');
%% Nearest Neighbor
disp('NN 구현 함수');
tic % Resizing 시간 측정 - 시작
INN=NN(I,4);
toc % Resizing 시간 측정 - 정지
figure(2); imshow(INN); title('NN 구현 함수');
imwrite(INN,fullfile(folderResult,[image,'_NN_구현.png']));% save results
% calculate psnr
i=imread(['./result_figure/',image,'_NN_구현.png']);
my_psnr_value=my_psnr(i,Igt);
psnr_value=psnr(i,Igt);
fprintf('Implementation PSNR %0.4f.\n',my_psnr_value);
fprintf('Built-in PSNR %0.4f.\n',psnr_value);
disp('NN 내장 함수');
tic
Inn=imresize(I,4,'nearest');
toc
figure(3); imshow(Inn); title('NN 내장 함수');
imwrite(Inn,fullfile(folderResult,[image,'_NN_내장.png']));% save results
% calculate psnr
i=imread(['./result_figure/',image,'_NN_내장.png']);
my_psnr_value=my_psnr(i,Igt);
psnr_value=psnr(i,Igt);
fprintf('Implementation PSNR %0.4f.\n',my_psnr_value);
fprintf('Built-in PSNR %0.4f.\n',psnr_value);
disp('=====');
%% Bilinear
```



```

disp('Bilinear 구현 함수');
tic
IBB=Bilinear(I,4);
toc
figure(4); imshow(IBB); title('Bilinear 구현 함수');
imwrite(IBB,fullfile(folderResult,[image,'_Bil_구현.png']));% save results
% calculate psnr
i=imread(['./result_figure/',image,'_Bil_구현.png']);
my_psnr_value=my_psnr(i,Igt);
psnr_value=psnr(i,Igt);
fprintf('Implementation PSNR %0.4f.\n',my_psnr_value);
fprintf('Built-in PSNR %0.4f.\n',psnr_value);
disp('Bilinear 내장 함수');
tic
Ibb=imresize(I,4,'bilinear');
toc
figure(5); imshow(Ibb); title('Bilinear 내장 함수');
imwrite(Ibb,fullfile(folderResult,[image,'_Bil_내장.png']));% save results
% calculate psnr
i=imread(['./result_figure/',image,'_Bil_내장.png']);
my_psnr_value=my_psnr(i,Igt);
psnr_value=psnr(i,Igt);
fprintf('Implementation PSNR %0.4f.\n',my_psnr_value);
fprintf('Built-in PSNR %0.4f.\n',psnr_value);
disp('=====');
%% Bicubic
disp('Bicubic 구현 함수');
tic
ICC=Bicubic(I,4);
toc
figure(6); imshow(ICC); title('Bicubic 구현 함수');
imwrite(ICC,fullfile(folderResult,[image,'_Bic_구현.png']));% save results
% calculate psnr
i=imread(['./result_figure/',image,'_Bic_구현.png']);
my_psnr_value=my_psnr(i,Igt);
psnr_value=psnr(i,Igt);
fprintf('Implementation PSNR %0.4f.\n',my_psnr_value);
fprintf('Built-in PSNR %0.4f.\n',psnr_value);
disp('Bicubic 내장');
tic
Icc=imresize(I,4);
toc
figure(7); imshow(Icc); title('Bicubic 내장 함수');
imwrite(Icc,fullfile(folderResult,[image,'_Bic_내장.png']));% save results
% calculate psnr
i=imread(['./result_figure/',image,'_Bic_내장.png']);
my_psnr_value=my_psnr(i,Igt);
psnr_value=psnr(i,Igt);
fprintf('Implementation PSNR %0.4f.\n',my_psnr_value);
fprintf('Built-in PSNR %0.4f.\n',psnr_value);
disp('=====');

```

★ PSNR 계산 – 구현한 함수와 매트랩 내장 함수 비교

PSNR은 매트랩 내장함수 `psnr`로도 구할 수 있지만, 코드로 구현해 보았다.

[my_psnr.m]

```
%% jsyoo
function out = my_psnr(I,ref)
% I : Interpolation 이미지, ref : 기준(Ground Truth) 이미지
[H,W,D] = size(I);

% MSE 계산
R_diff = (double(I(:,:,1))-double(ref(:,:,1))).^2;
G_diff = (double(I(:,:,2))-double(ref(:,:,2))).^2;
B_diff = (double(I(:,:,3))-double(ref(:,:,3))).^2;

% RGB 각각의 MSE
R_mse = sum(sum(R_diff)) / (H * W); % R의 mse
G_mse = sum(sum(G_diff)) / (H * W); % G의 mse
B_mse = sum(sum(B_diff)) / (H * W); % B의 mse

% R, G, B 에 대한 MSE 평균값
MSE = (R_mse + G_mse + B_mse) / 3;
% PSNR 계산식 // MAX_I = 255(8bit image)
out = 10*log10(255^2/MSE);
```

구현한 PSNR 함수와 내장 함수간의 차이가 있는지 `compare.m`을 만들어 실험해 보았다. 실험 기

법은 NN, Bilinear, Bicubic 세 가지이고, 실험 이미지는 woman이다.

=====	=====	=====
NN 구현 함수	Bilinear 구현 함수	Bicubic 구현 함수
경과 시간은 0.011536초입니다.	경과 시간은 0.115220초입니다.	경과 시간은 0.245806초입니다.
Implementation PSNR 23.0855.	Implementation PSNR 24.2144.	Implementation PSNR 25.1185.
Built-in PSNR 23.0855.	Built-in PSNR 24.2144.	Built-in PSNR 25.1185.
NN 내장 함수	Bilinear 내장 함수	Bicubic 내장 함수
경과 시간은 0.140516초입니다.	경과 시간은 0.047671초입니다.	경과 시간은 0.022557초입니다.
Implementation PSNR 23.0855.	Implementation PSNR 24.2150.	Implementation PSNR 25.1118.
Built-in PSNR 23.0855.	Built-in PSNR 24.2150.	Built-in PSNR 25.1118.
=====	=====	=====

위의 결과를 보다시피 같은 값을 계산해 내는 것을 확인할 수 있다.

(4) SRMD

① Code

[demo_SRMDNF.m]

```
%% jsyoo
format compact;
addpath('utilities');
imageSets = {'Set5','Set14','BSD100','Urban100'}; % testing dataset
%% Select test dataset and set folder
setTest = imageSets([1]);
method = 'SRMDNF';
test_folder = 'testsets';
result_folder = 'results';
if ~exist(result_folder,'file') % results 폴더가 있는지 확인하고 없으면
    mkdir(result_folder); % 폴더 만들기
end
sf = 4; % scale factor = 4
%% Load model
model_folder = 'models';
load(fullfile(model_folder,['SRMDNFx4.mat']));
% set network
net = vl_simplenn_tidy(net);
%% degradation parameter (kernel) setting
global degpar;
% kernel : isotropic Gaussian---although it is a special case of
anisotropic Gaussian.
kernelwidth = 2.6; % from a range of [0.2, 4] for sf = 4.
kernel = fspecial('gaussian',15, kernelwidth); % Note: the kernel size is
fixed to 15X15.
tag = ['_',method,'_x',num2str(sf),'_itrG_',int2str(kernelwidth*10)];

figure(6); surf(kernel) % show kernel
view(45,55);
title('Assumed kernel');
xlim([1 15]);
ylim([1 15]);
%% for degradation maps
degpar = single(net.meta.P*kernel(:)); % save single(4byte)
for n_set = 1 : numel(setTest)
    %% search images
    setTestCur = cell2mat(setTest(n_set));
    testCur_folder = fullfile(test_folder,setTestCur);
    ext = {'*.jpg','*.png','*.bmp'};
    filepaths = [];
    for i = 1 : length(ext)
        filepaths = cat(1,filepaths,dir(fullfile(testCur_folder, ext{i})));
    end
    %% prepare results
    eval(['PSNR_',setTestCur,'_x',num2str(sf),' =
zeros(length(filepaths),1);']);
    resultCur_folder = fullfile(result_folder, [setTestCur,tag]);
    if ~exist(resultCur_folder,'file')
        mkdir(resultCur_folder);
    end
end
```

```

%% perform SISR(Single Image Super Resolution)
for i = 1 : length(filepaths)
    HR = imread(fullfile(testCur_folder,filepaths(i).name));
    [~,imageName,ext] = fileparts(filepaths(i).name);
    HR = modcrop(HR, sf);
    label_RGB = HR;
    % blur using kernel
    blurry_HR = imfilter(im2double(HR),double(kernel),'replicate'); %
blur
    % make degradation with direct downsampler by approximating it
    LR = imresize(blurry_HR,1/sf,'bicubic'); % bicubic downsampling
    input = single(LR); % save 32bit
    % run srmd network
    res = vl_srmd_matlab(net, input);

    output_RGB = gather(res(end).x);

    label = rgb2ycbcr(im2double(HR));
    output = rgb2ycbcr(double(output_RGB));
    label = label(:, :, 1);
    output = output(:, :, 1);
    %% calculate PSNR and SSIM
    [PSNR_Cur,SSIM_Cur] = Cal_PSNRSSIM(label*255,output*255,sf,sf); %%%
single
    figure(i);
    disp([setTestCur,' ',int2str(i),'
',num2str(PSNR_Cur,'%2.2f'),'dB',' ',filepaths(i).name]);
    eval(['PSNR_',setTestCur,'_x',num2str(sf),'(',num2str(i),') =
PSNR_Cur;']);

    imshow(cat(2,label_RGB,imresize(im2uint8(LR),sf),im2uint8(output_RGB)));

    title(['SISR ',filepaths(i).name,'
',num2str(PSNR_Cur,'%2.2f'),'dB'],'FontSize',12)
    pause(1)

    imwrite(output_RGB,fullfile(resultCur_folder,[imageName,'_x',int2str(sf),'_
',int2str(PSNR_Cur*100),'.png'])); % save results
end
    disp(['Average PSNR is
',num2str(mean(eval(['PSNR_',setTestCur,'_x',num2str(sf)])),'%2.2f'),'dB'])
;
    %% Save PSNR and SSIM results

    save(fullfile(resultCur_folder,['PSNR_',setTestCur,'_x',num2str(sf),'.mat']
),['PSNR_',setTestCur,'_x',num2str(sf)]);
end

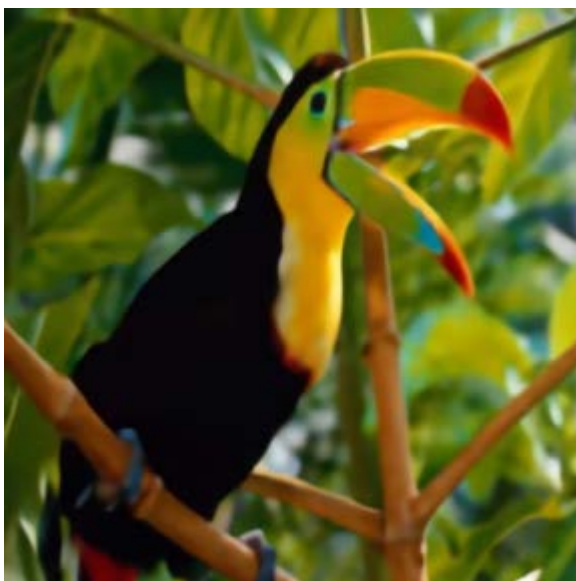
```

② Result image

1. baby



2. bird



3.butterfly



4. head



5. woman




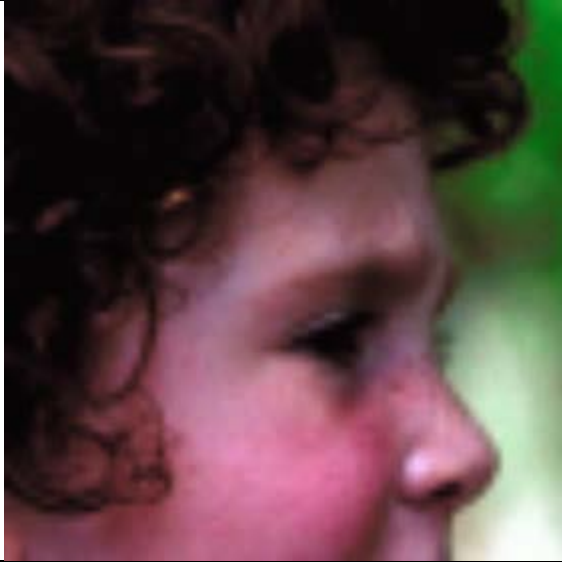


③ PSNR

	PSNR
Baby	33.6593
Bird	34.3348
Butterfly	27.6652
Head	32.7867
Woman	30.3936
Average	31.7679

Conclusion

결론에서는 본론에서 설명한 네 기법(Nearest Neighbor, Bilinear interpolation, Bicubic interpolation, SRMDNF)의 결과 이미지와 PSNR 값을 먼저 비교한 것을 보이고 설명하겠다.

★ Nearest Neighbor, Bilinear interpolation, Bicubic interpolation(구현), SRMDNF 비교

Nearest Neighbor	Bilinear interpolation
	
27.9197 db	28.5450
Bicubic interpolation	SRMDNF
	
28.9421 db	32.7867 db

➔ 주어진 head 이미지로 사람의 눈과 PSNR 값으로 Nearest Neighbor, Bilinear interpolation, Bicubic interpolation, SRMDNF 네 기법을 비교해 보았다. 결과 이미지를 보면, Nearest Neighbor가 확연히 많이 깨져 보인다. Bilinear interpolation과 Bicubic interpolation은 사람의 눈으로 어느 기법이 더 좋은지 판단하기에는 어렵다. 실제로 PSNR 값도 약 0.4db 차이로 그리 크지는 않지만 Bicubic interpolation이 조금 더 우수하다고 볼 수 있다. 네 기법 중 최신 기법인 SRMDNF는 눈으로 보기에 확연히 좋은 결과를 보이는 것을 확인할 수 있다. 구레나룻 쪽 곱슬머리나, 눈, 코 쪽이 다른 기법들에 비해 선명함을 볼 수 있다. 그리고 PSNR 값도 다른 기법들에 비해 훨씬 높은 것으로 가장 우수하다고 할 수 있다.

과제를 통해서도 실습해보았지만, gray image로 변환 후 수행해본 것이었다. 하지만 본 프로젝트를 수행하며 Red, Green, Blue 컬러 3채널 이미지를 입력으로 사용하여 결과 이미지도 컬러 이미지로 출력해봄으로써 영상과 영상처리에 대한 이해도를 많이 늘 수 있는 좋은 기회가 되었다. 또, 세 기법(Nearest Neighbor, Bilinear interpolation, Bicubic interpolation)은 고전 기법이고 함수 출력 결과 이미지도 다소 아쉬운 품질이었다. SRMDNF라는 최신 기법에 대해 공부하고 코드 분석하며, 수행 해봄으로써 논문 분석 및 코드 분석과 영상 처리에 대한 이해도와 동향을 공부해볼 수 있는 좋은 기회가 된 것 같고 실력도 조금 는 것 같아서 좋았다. 이번 강의 수강으로 끝내지 않고 향후 컴퓨터 비전도 공부하여 실력을 늘릴 계획이다.

Reference

https://en.wikipedia.org/wiki/Bicubic_interpolation

https://ko.wikipedia.org/wiki/%EC%B5%9C%EB%8C%80_%EC%8B%A0%ED%98%B8_%EB%8C%80_%EC%9E%A1%EC%9D%8C%EB%B9%84

<https://github.com/thoste/matlab-scaler>

https://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_Learning_a_Single_CVPR_2018_paper.html

<https://github.com/cszn/SRMD>