

# DIP Homework 1

---

## 1. Program

---

### 1.1. Function used

- cv2 (opencv-python)
  - cv2.imread() - read image
  - cv2.imwrite() - write image
  - cv2.cvtColor() - convert image color space
  - cv2.resize() - scale image

### 1.2. Program introduction

- Implements cv2.resize() function to scale image with different methods (bilinear, bicubic).
- Embeds demo option for grading purpose.
- Environments
  - macos 10.14.6
  - python 3.7.4
    - opencv-python == 4.1.1.26

### 1.3. Program usage

- Scale a image given scaling factor and method

```
python3 resize.py [image_path] --scale [scaling_factor] --method [scaling_method]
```







- Demo homework 1 with given input image path

```
python3 resize.py [image_path] --demo
```



- Print help message

```
python3 resize.py --help
```

## 2. Demo

Scale	Bilinear	Bicubic
1.0		
0.2		
3.0		
10.0		

### 3. Comparison

	Bilinear	Bicubic
Pros	fast	smoother curves when scaling up
Cons	artifacts on curves when scaling up, blurred	slow
Example		

## 4. Bicubic Interpolation

### 4.1. Introduction

- Bicubic interpolation uses 16 points for each target pixel to compute its value. During process, cubic functions are implemented to generate smoother result than linear functions. The following are the details.
- Our target is to modeling a function  $f(x, y)$

$$\begin{aligned} f(x, y) &= \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} x^i y^j \\ &= \begin{bmatrix} x^3 & x^2 & x^1 & 1 \end{bmatrix} \begin{bmatrix} a_{33} & a_{32} & a_{31} & a_{30} \\ a_{23} & a_{22} & a_{21} & a_{20} \\ a_{13} & a_{12} & a_{11} & a_{10} \\ a_{03} & a_{02} & a_{01} & a_{00} \end{bmatrix} \begin{bmatrix} y^3 \\ y^2 \\ y^1 \\ 1 \end{bmatrix} \\ &= X^T A Y \end{aligned}$$

- Given 4x4 known data points

$$F = \begin{bmatrix} f(-1, -1) & f(-1, 0) & f(-1, 1) & f(-1, 2) \\ f(0, -1) & f(0, 0) & f(0, 1) & f(0, 2) \\ f(1, -1) & f(1, 0) & f(1, 1) & f(1, 2) \\ f(2, -1) & f(2, 0) & f(2, 1) & f(2, 2) \end{bmatrix}$$

- We could obtain the equation

$$F = X^T A Y$$
$$\begin{bmatrix} f(-1, -1) & f(-1, 0) & f(-1, 1) & f(-1, 2) \\ f(0, -1) & f(0, 0) & f(0, 1) & f(0, 2) \\ f(1, -1) & f(1, 0) & f(1, 1) & f(1, 2) \\ f(2, -1) & f(2, 0) & f(2, 1) & f(2, 2) \end{bmatrix} = \begin{bmatrix} (-1)^3 & (-1)^2 & (-1)^1 & 1 \\ 0 & 0 & 0 & 1 \\ 1^3 & 1^2 & 1^1 & 1 \\ 2^3 & 2^2 & 2^1 & 1 \end{bmatrix} \begin{bmatrix} a_{33} & a_{32} & a_{31} & a_{30} \\ a_{23} & a_{22} & a_{21} & a_{20} \\ a_{13} & a_{12} & a_{11} & a_{10} \\ a_{03} & a_{02} & a_{01} & a_{00} \end{bmatrix} \begin{bmatrix} (-1)^3 & 0 & 1^3 & 2^3 \\ (-1)^2 & 0 & 1^2 & 2^2 \\ (-1)^1 & 0 & 1^1 & 2^1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Therefore, the coefficient  $A$  of the model  $f(x, y)$  could be obtained by calculating the inverse of  $X^T$  and  $Y$ .

$$A = \begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ -\frac{1}{3} & -\frac{1}{2} & 1 & -\frac{1}{6} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} f(-1, -1) & f(-1, 0) & f(-1, 1) & f(-1, 2) \\ f(0, -1) & f(0, 0) & f(0, 1) & f(0, 2) \\ f(1, -1) & f(1, 0) & f(1, 1) & f(1, 2) \\ f(2, -1) & f(2, 0) & f(2, 1) & f(2, 2) \end{bmatrix} \begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & -\frac{1}{3} & 0 \\ \frac{1}{2} & 1 & -\frac{1}{2} & 1 \\ -\frac{1}{2} & \frac{1}{2} & 1 & 0 \\ \frac{1}{6} & 0 & -\frac{1}{6} & 0 \end{bmatrix}$$

- With the model  $f(x, y)$ , we are able to compute the data point (coordinate within  $0 \leq x \leq 1$  and  $0 \leq y \leq 1$ ) referenced from these 4x4 known data points.

## 4.2. Complexity

- Without any optimization, we could summarize the multiplication and addition steps for each pixel to compare the computational complexity between bilinear and bicubic interpolation.

	Bilinear	Bicubic
Addition	$2 \times 2 - 1 = 3$	$4 \times 4 - 1 = 15$
Multiplication	$2 \times 2 \times 2 = 8$	$4 \times 4 \times 2 = 32$
Overall	11	47

- From the table above, we could conclude that bicubic interpolation is about 4 times slower than bilinear interpolation.