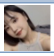
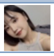
Homework # 1

姓名	學號
操之晴	R13922A04

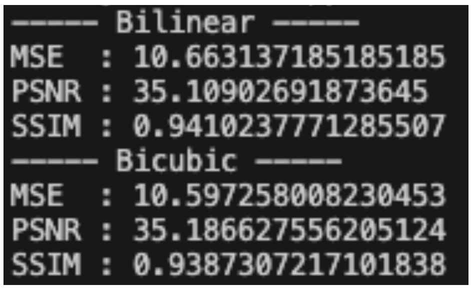
Exercise 1

1. Resulted images for comparison

	Original	Bilinear(*0.12*7)	Bicubic(*0.12*7)
Image			
size	1125*1125	945*945	945*945
MSE		10.663137	10.597258
PSNR		35.109027	35.186628
SSIM		0.941024	0.938731

	Bilinear	Bicubic	size
Shrink (*0.12)			135*135
Zoom (*7)			945*945

(The above images have all been scaled down proportionally.)



```
------ Bilinear ------
MSE  : 10.663137185185185
PSNR : 35.10902691873645
SSIM : 0.9410237771285507
------ Bicubic ------
MSE  : 10.597258008230453
PSNR : 35.186627556205124
SSIM : 0.9387307217101838
```

2. **Function I used or implemented**
   - I used the cv2.resize() function from the OpenCV library to perform image scaling. For bilinear interpolation, I used the cv2.INTER_LINEAR, and for bicubic interpolation, I used cv2.INTER_CUBIC.
   - I implemented the mse(), psnr(), and SSIM() functions to calculate the quality metrics of the scaled images. The MSE (Mean Squared Error) and PSNR (Peak Signal-to-Noise Ratio) were calculated using NumPy and OpenCV, while SSIM (Structural Similarity Index) was computed using the ssim() function from the skimage.metrics library.

3. **How does your program work**

   The program first reads the original origin image using OpenCV's cv2.imread() function.
   The scale_image() function is called twice: first to shrink the image by a factor of 0.12, and then to zoom the shrunken image by a factor of 7. The function is executed for both bilinear and bicubic interpolation methods.
   The quality metrics for both the bilinear and bicubic zoomed images are computed using the mse(), psnr(), and SSIM() functions.
   Finally, the processed images are saved using cv2.imwrite() and the quality results are printed to the console.

4. **How to use my program**
   a. Save an origin image with a size at least 512x512 pixels as 'origin.png' in the working directory.
   b. Run the program. It will shrink and zoom the image using both bilinear and bicubic interpolation methods.
   c. The processed images will be saved as 'ex1_bilinear_shrink.png' , 'ex1_bilinear_zoom.png', 'ex1_bicubic_shrink.png' and 'ex1_bicubic_zoom.png'.
   d. The program will also print the MSE, PSNR, and SSIM values of the zoomed images.

5. **Explanation for my resulted images**

   Bilinear interpolation works by doing linear interpolation along both the X and Y axes. It interpolates in one direction first, then in the other to get the final pixel value. Since it only uses the 4 nearest pixels to calculate a new on, it's pretty simple.

   However, **Bicubic** interpolation is more complex --- it uses cubic polynomials to get it done. Considering the brightness changes of neighboring pixels, which helps maintain more details and sharper edges.

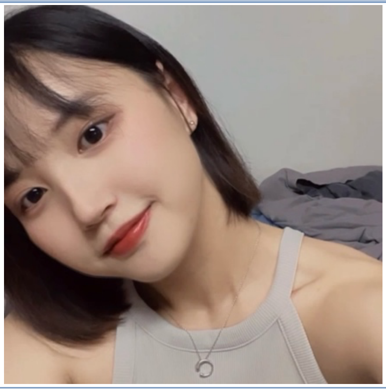
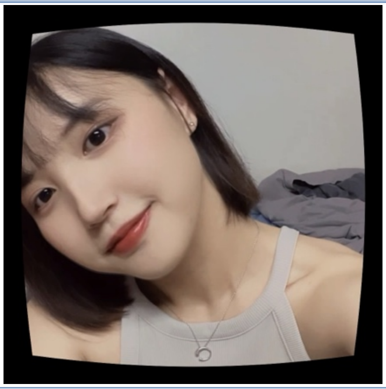
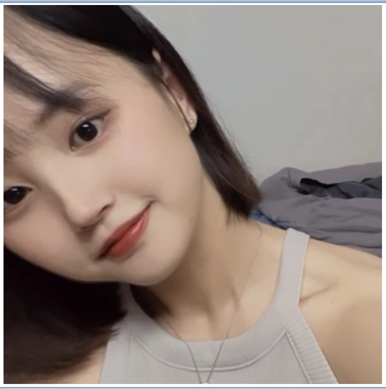
   In short, bilinear interpolation is simpler and faster, but it tends to lose detail and can make images blurry; On the other hand, bicubic interpolation is more complex but does a better job of preserving details and edges, giving a higher image quality overall. That's why we can see that no matter in MSE, PSNR or SSIM, bicubic interpolation has a better performance from the comparison above.

6. **A more detailed operation on Bicubic interpolation**
   (1) **Grid Selection**:
   It uses a 4x4 grid of pixels around the pixel being interpolated. This means it uses the 16 nearest pixel values to calculate the new pixel.
   (2) **Cubic Interpolation**:
   First, interpolation is done along one axis. The pixel values in the 4 nearest columns are interpolated using cubic polynomials. The process is repeated along the Y-axis. After calculating interpolated values along the X-axis, the same interpolation is done along the Y-axis, using the results from the X-interpolation.
   (3) **Cubic Polynomial**:
   The interpolation is based on a cubic function $f(x) = ax^3 + bx^2 + cx + d$,which a,b,c,d are determined using the four neighboring pixel values.
   (4) **Weighted Average:**
   The final interpolated pixel value is computed as a weighted average of the 16 surrounding pixels, with the weights determined by the cubic polynomial.

7. **Computational Complexity Comparison**
   Even though both Bilinear and Bicubic interpolation have the same complexity of O(n*m), where n*m represents the number of pixel in the image, Bicubic interpolation is slower. This is because it works with a larger constant factor (16 neighboring pixels and performs cubic interpolation).

Exercise 2

1. Resulted images for comparison

	Original	Barrel Distortion	Pincushion Distortion
Image			

2. Function I used or implemented

I used the cv2.undistort() function from the OpenCV library to apply lens distortion effects. The distortion coefficients k1 and k2 are used to control the radial distortion, where k1 mainly affects barrel or pincushion distortion.

3. How does your program work

The program reads an image using cv2.imread(). Then the warpping() function takes two radial distortion coefficients, k1 and k2, and creates a camera matrix based on the image size. It then applies lens distortion using cv2.undistort(), returning the warped image.
Two different distortions are simulated: barrel distortion with k1 = 0.5 and pincushion distortion with k1 = -0.5.
The processed images are saved using cv2.imwrite().

4. How to use my program

a. Save a origin image named 'origin.png' in the working directory.
b. Run the program. It will generate two distorted images: one with barrel distortion and one with pincushion distortion.
c. The distorted images will be saved as 'ex2_barrel.png' and 'ex2_pincushion.png'.

5. The difference between barrel and pincushion distortion

The main difference between them is how they affect the image.

- Barrel distortion: straight lines bow **outward** from the center, like a shape of barrel.
- Pincushion distortion: straight lines bow **inward** toward the center, like a shape of pincushion.

In the Brown-Conrady Model, these two distortions are caused by the radial distortion coefficients, which affect the image more when the pixels move away from the center.

6. Parameters in the code

Since we need some mathematic models to describe the distortion, there are 2 parameters I need to decide in this exercise -- camera_matrix and dist_coeffs.

- Camera_matrix

  It describes the camera's intrinsic parameters, which define how a 3D point in the real world is projected onto the 2D image plane. The matrix structure looks like this:

fx	0	cx
0	fy	cy
0	0	1

→

w	0	w/2
0	h	h/2
0	0	1

  Where fx and fy are the focal lengths, cx and cy are the coordinates of the center of projection I chose w/2 and h/2 as the center, which is typically the middle of the width and height. This indicates the projection center in the 2D image plane.

- Dist_coeffs

  Distortion coefficients contain about 4 values. Since we want to simulate the lens distortion, we are only concerned with the first two parameters, k1 and k2. Where k1 is the primary radial distortion parameter, which controls the main distortion effect. I selected k1 to be ± 0.5 in each distortion, as k1 greater than 0, it produces barrel distortion; while k1 less than 0, it produces pincushion distortion.

  k2 is a secondary radial distortion parameter, here I set it to 0 in simpler cases.

Version of packages I used in my program

Python	3.12.6
OpenCV	4.10.0
NumPy	2.1.1