Two Common Image Interpolation Algorithm

Nearest Neighbor Interpolation and Bilinear Interpolation

11712610 马思清

I. Nearest Neighbor Interpolation

1. Introduction

Nearest Neighbor Interpolation is a basic interpolation technique in image processing. When resizing an image(increase or decrease the number of pixels), we first create a new image which have different pixel density and normalized it with the old image in the same scale.The main problems is how to determine the intensity of each new pixel. In Nearest Neighbor Interpolation, for each new pixel, the intensity is determined by the nearest old pixel, so called nearest neighbor.

2. Algorithm description

In figure.1 The two image is normalized in the same scale(The new picture's size). We choose the Cartesian coordinate system whose origin is in the upper left corner and calculate the relative coordinate of each pixel. We label the pixel with red and blue each stands for the new image pixel and the origin ones. For each red pixel, we find the nearest blue dot directly and assign the intensity of the blue pixel to the red pixel.

Notice that some equidistance situations may occur(two circled pixel), if so, we choose one of the nearest pixels, in this algorithm, we always choose the top left corner.

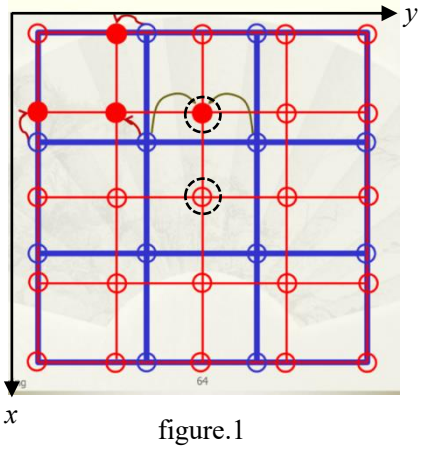


figure.1

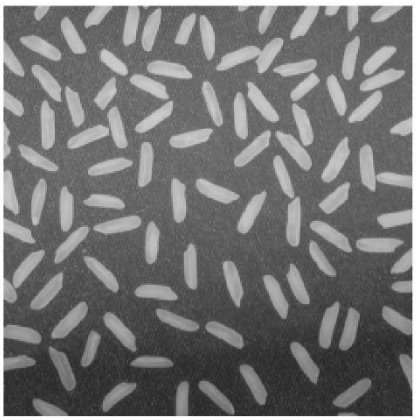
3. The pseudo code

- *Normalize the two image in the same scale(the new image size) and record the relative position of each pixel*
- *Mark the old pixel blue and new pixel red*
- *For each red column, find the nearest blue column*
- *For each red row, find the nearest blue row*
- *For each red pixel, according it's nearest blue column and blue row, get it's nearest blue pixel. Replace it's intensity by the blue pixel's intensity*

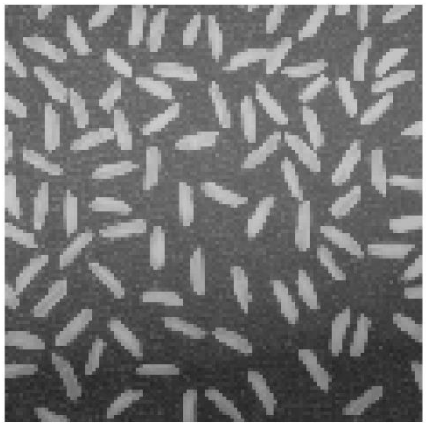
■ ***The MATLAB code is attached in Appendix***

Special case: Since the for loop is searched from the origin point, go down and right, when equidistance situations happens, the first searched nearest pixel is always the top left ones, and we select the first occur smallest ones.

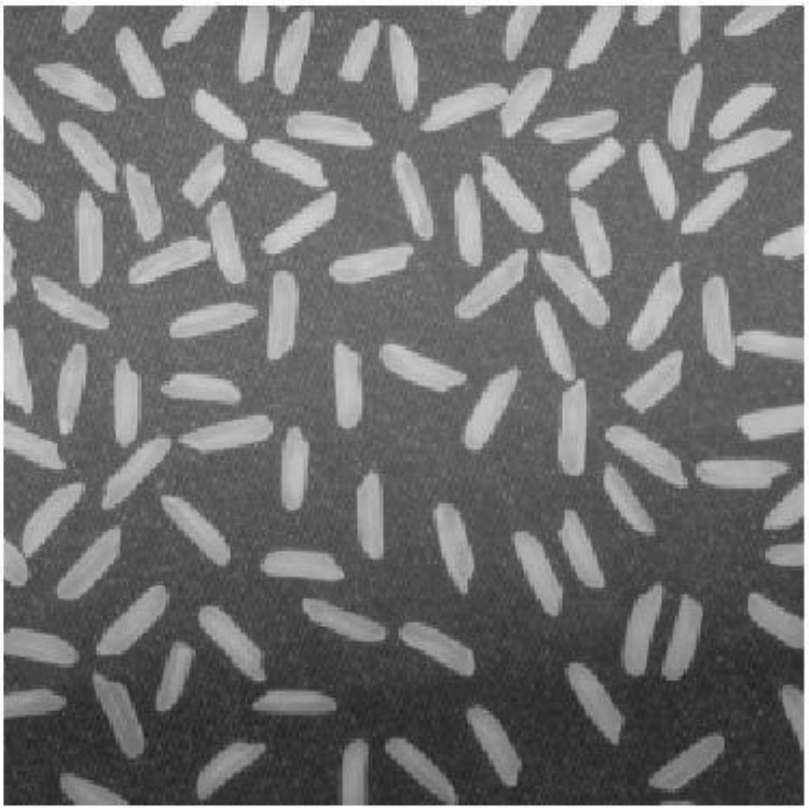
4. Results



(a) The origin 256x256 picture



(b) Resized 115x115 picture



(c) Resized 371x371 picture

5. Analysis and conclusions

In this algorithm, we first go through all the columns of the new image to find the nearest origin image's columns, then go through all the rows. After that, we go through all the new pixels and fill their intensity. The time complexity is $O(am + bn + mn)$, where a and b are the dimension of the origin image and n, m are the dimension of the new image, the running time of this algorithm is determined by both image's size.

Since the algorithm always choose the nearest one. So when we zooming the origin picture, the boundary intensity have a large difference, and the gray scale is not continuous. However, this algorithm is very simple and don't need a large amount of calculation, which is suitable for some applications which we don't need a high quality amplification.

II. Bilinear Interpolation

1. Introduction

Bilinear Interpolation is a modified interpolation approach. Comparing with the Nearest Neighbor Interpolation, Bilinear use a linear processing method instead of the nearest neighbor to determine the intensity of the new pixel. In this new approach, for each new pixel whose intensity is unknown, we first find four nearest old pixels, which form a rectangle around the new pixel. Then based on the intensity of these four points, use an similar linear fitting method to get the intensity of the new pixel.

2. Algorithm description

In figure.2 The two image is normalized in the same scale(The new picture's size). We choose the Cartesian coordinate system whose origin is in the upper left corner and calculate the relative coordinate of each pixel. We label the pixel with red and blue each stands for the new image pixel and the origin ones. For each red pixel, we find the nearest 4 blue dot and then do linear interpolation.

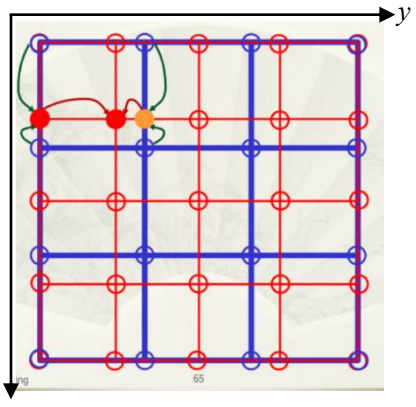
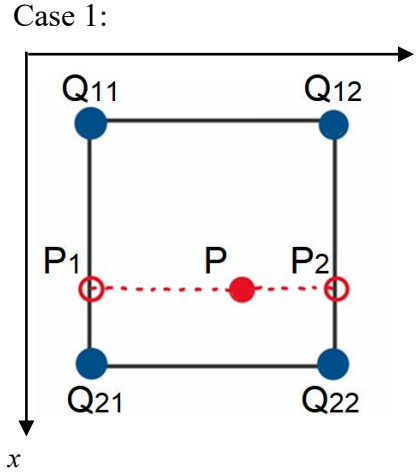


figure.2

There are main parts of the linear interpolation process. The first part is find the position ratio between known points and unknown points. Second is calculate the intensity using position ratio. In this algorithm, we first chose the y_scale to do linear fitting and get P1 and P2's intensity, then do linear fitting in the x_scale. Here are the computational formula.

Notice: Q stands for the origin pixels, and P is the new pixel whose intensity is unknown
The function I(Q) can return the intensity of the given pixel, x is the pixel's position

Case 1:

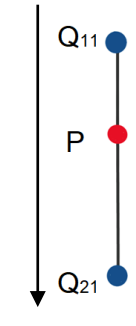


$$I(P_1) = \frac{x-x_1}{x_2-x_1}\big(I(Q_{21}) - I(Q_{11})\big) + I(Q_{11}) \quad ...(1)$$

$$I(P_2) = \frac{x-x_1}{x_2-x_1}\big(I(Q_{22}) - I(Q_{12})\big) + I(Q_{12}) \quad ...(2)$$

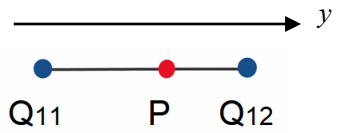
$$I(P) = \frac{y-y_2}{y_1-y_2}\big(I(P_1) - I(P_2)\big) + I(P_2) \quad ...(3)$$

Case 2:



$$I(P) = \frac{x-x_1}{x_2-x_1}\big(I(Q_{21}) - I(Q_{11})\big) + I(Q_{11}) \quad ...(4)$$

Case 3:



$$I(P) = \frac{y-y_2}{y_1-y_2}\left(I(Q_{11}) - I(Q_{12})\right) + I(Q_{12}) \quad \ldots(5)$$

Case 4:

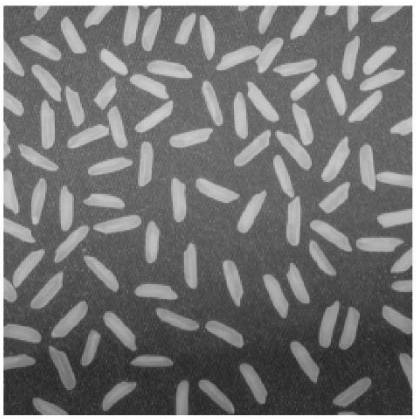
$$I(P) = I(Q) \quad \ldots(6)$$

3. The pseudo code:

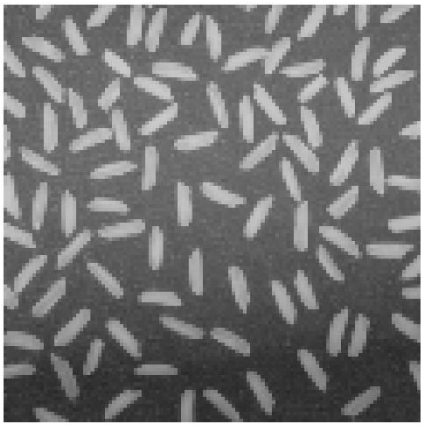
- *Normalize the two image in the same scale(the new image size) and record the relative position of each pixel*
- *Mark the old pixel blue and new pixel red*
- *For each red column, find the nearest **right blue column** and the nearest **left blue column***
- *For each red row, find the nearest **upper blue row** and the nearest **lower blue row***
- *For each red pixel, according to it's nearest column and row, find the neighbor points*
-
  - *If the two columns and two rows are different, use formula 1、2、3 to get the intensity of the red pixel*
  - *If the two columns are the same and two rows are different, use formula 4 to get the intensity of the red pixel*
  - *If the two rows are the same and two columns are different, use formula 5 to get the intensity of the red pixel*
  - *If rows and columns are all the same, use formula 6 to get the intensity of the red pixel*

■ ***The MATLAB code is attached in Appendix***

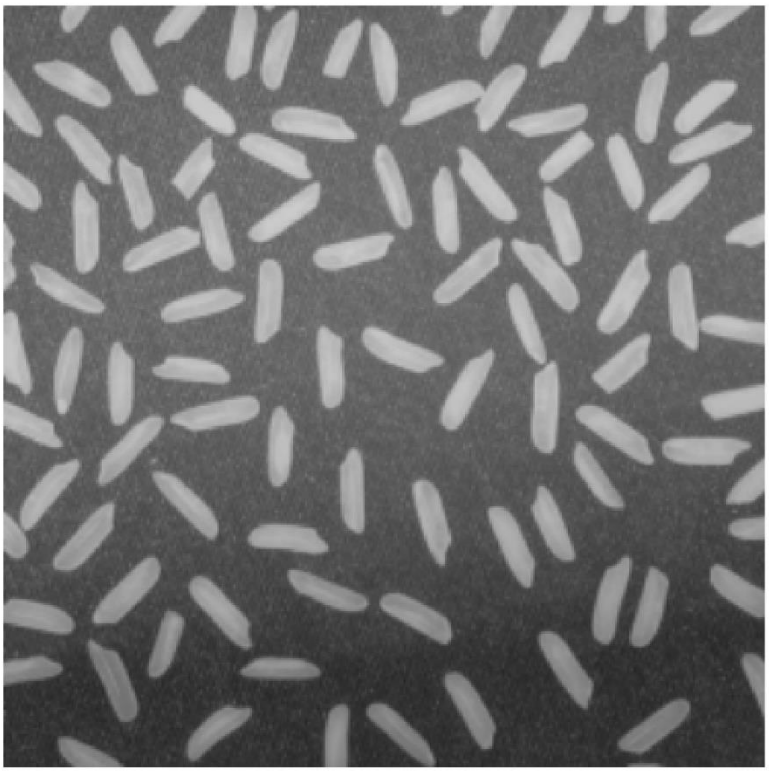
4. Results



(a) The origin 256x256 picture



(b) Resized 115x115 picture



(c) Resized 371x371 picture

5. Analysis and conclusions

In this algorithm, we first go through all the columns of the new image and find the nearest and next-nearest origin image's columns, then do the same things about the rows. After that, we go through all the new pixels and use the 4 neighbor points to calcuate their intensity. The time complexity is $O(am + bn + mn)$, where a and b are the dimension of the origin image and n, m are the dimension of the new image, which is the same as the Nearest Neighbor Interpolation. However, when calculating the single pixel's intensity, some special cases should be careful and the calculation complexity is much higher than the simple algorithm. The same, the running time is determined by both image's size, but this algorithm is a bit slower than the Nearest Neighbor when processing the same tasks.

We choose the nearest 4 origin pixel to determine a new pixel. So the intensity of the new pixel is not discrete, in contract, the transiation is very smooth. Given this property, the Bilinear Interpolation has the effect like a low-pass filter, the zoomed image's boundary is more indistinct, but comparing with the Nearest Neighbor approach, the quality of the zoomed picture is greatly improved.

III. Appendix

1. Matlab code of the Nearest Neighbor Interpolation

```
function Nearest_11712610(input_file, dim)
image = im2double(imread(input_file));
posi_i_x = [0 : size(image,1)-1]' ./ (size(image,1)-1) .*dim(1);
posi_i_y = [0 : size(image,2)-1]' ./ (size(image,2)-1) .*dim(2);
posi_ni_x = 0 : dim(1)-1;
posi_ni_y = 0 : dim(2)-1;

itp_x = zeros(dim(1), 1);
for i = 1:dim(1)
    var = 10000000000000000000000;
    for j = 1:size(image,1)
        if(var > abs(posi_ni_x(i) - posi_i_x(j)))
            var = abs(posi_ni_x(i) - posi_i_x(j));
            itp_x(i) = j;
        end
    end
end

itp_y = zeros(dim(2), 1);
for i = 1:dim(2)
    var = 10000000000000000000000;
    for j = 1:size(image,2)
        if(var > abs(posi_ni_y(i) - posi_i_y(j)))
            var = abs(posi_ni_y(i) - posi_i_y(j));
            itp_y(i) = j;
        end
    end
end

new_image = zeros(dim);
for i = 1:dim(1)
    for j = 1:dim(2)
        new_image(i,j) = image(itp_x(i), itp_y(j));
    end
end

figure(1)
imshow(image);
figure(2)
imshow(new_image);
```

2. Matlab code of the Bilinear Interpolation

```
function Bilinear_11712610(input_file, dim)
image = im2double(imread(input_file));
posi_i_x = [0 : size(image,1)-1]' ./ (size(image,1)-1) .*dim(1);
posi_i_y = [0 : size(image,2)-1]' ./ (size(image,2)-1) .*dim(2);
posi_ni_x = 0 : dim(1)-1;
posi_ni_y = 0 : dim(2)-1;

%for every new pixel, find nearest fourth
nearx = zeros(dim(1), 2);
for nx = 1:dim(1)
    [x1, x2] = deal(1);
    for ox = 1:size(image,1)
```

```
        if(posi_ni_x(nx) - posi_i_x(ox) < 0)
            x1 = ox-1;
            x2 = ox;
            break;
        end
        if(posi_ni_x(nx) == posi_i_x(ox))
            [x1, x2] = deal(ox);  break;
        end
    end
    nearx(nx, :) = [x1, x2];
end

neary = zeros(dim(2), 2);
for ny = 1:dim(2)
    [y1, y2] = deal(1);
    for oy = 1:size(image,2)
        if(posi_ni_y(ny) - posi_i_y(oy) < 0)
            y1 = oy-1;
            y2 = oy;
            break;
        end
        if(posi_ni_y(ny) == posi_i_y(oy))
            [y1, y2] = deal(oy);
            break;
        end
    end
    neary(ny, :) = [y1, y2];
end

new_image = zeros(dim);
for nx = 1:dim(1)
    for ny = 1:dim(2)
        x1 = nearx(nx, 1);
        x2 = nearx(nx, 2);
        y1 = neary(ny, 1);
        y2 = neary(ny, 2);
        x = posi_ni_x(nx);
        y = posi_ni_y(ny);
        ip = 0;
        if(x1 ~= x2 && y1 ~= y2)
            ip1 = (y-posi_i_y(y1))/(posi_i_y(y2)-posi_i_y(y1)) * (image(x2, y2) -
image(x2, y1)) + image(x2, y1);
            ip2 = (y-posi_i_y(y1))/(posi_i_y(y2)-posi_i_y(y1)) * (image(x1, y2) -
image(x1, y1)) + image(x1, y1);
            ip = (x-posi_i_x(x2))/(posi_i_x(x1)-posi_i_x(x2)) * (ip2 - ip1) + ip1;
        end
        if(x1 == x2 && y1 ~= y2)
            ip = (y-posi_i_y(y1))/(posi_i_y(y2)-posi_i_y(y1)) * (image(x2, y2) -
image(x2, y1)) + image(x2, y1);
        end
        if(x1~=x2 && y1==y2)
            ip = (x-posi_i_x(x2))/(posi_i_x(x1)-posi_i_x(x2)) * (image(x1, y1) -
image(x2, y1)) +  image(x2, y1);
        end
        if(x1==x2 && y1==y2)
            ip = image(x1, y1);
        end
        new_image(nx, ny) = ip;
    end
end
figure(1)
imshow(image);
figure(3)
imshow(new_image);
```