Image Resizing

Sigi Lopez

10/08/18

Image Resizing is a technique that manipulates an image, so it can be transformed into a smaller or bigger size. There's multiple methods to resize an image, each one being more complex than the other, such methods are called Nearest, Bilinear, and Bicubic interpolation. All of them follow the basic rule which is to calculate each resized pixel using location of its surrounding pixels. The fastest method is called Nearest interpolation the biggest benefit is that it uses very little processing power compare to bilinear. This is mostly due to the need for bilinear to calculate four times the values compared to nearest neighbor which only estimates based on one value. All of the image manipulation in this lab was done with 256-bit grayscale Images.

Technical Discussion

The implementation for image resizing varies depending on the amount of refining you want to apply. The first step to calculate resizing we need to find the ratios between the old image size and the new image size. Figure 1 below shows how this can be calculated by using this simple function. M is the original image size and M' is the new image size. The input of this function is m' which is a point on the original image, the output is the calculated point on the new image. This is the most straightforward approach but since pixels are discreet which means they are not linear, you need to find the values of blocks. The first method is called Nearest Interpolation which essentially means that the closest pixel near the target coordinate becomes that pixels value. The more complicated method called Bilinear Interpolation involves taking the average of the nearest pixels. To fist the distances and then the values of the pixels I used the method of triangle ratios which involved taking 4 nearest pixels. Since the visual results of resizing and comparing these images might be very minimal because it basically reuses many of the pixels we used an equation to analyze the results. This equation is figure 2 below, it is called the root mean square deviation. It generally needs two inputs for this implementation we used pixel values from two related images. The result of this equation is a deviation which means how different the two inputs are from each other. This is very helpful since it will tell us how different the images turned out by using RMSE. The higher the RMSE the less similar the two images are.

Figure 1

$$x = t(m') = \frac{M}{M'}(m' - 0.5) + 0.5$$

Figure 2

$$RMSE = \sqrt{\frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(I1(m,n) - I2(m,n)\right)^2}\ .$$

Discussion of results

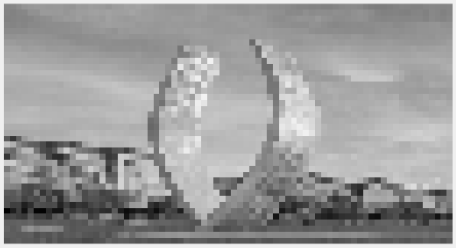
Qualitatively you can really tell that the images that were downsized turned out to be the most affected negatively. The main reason for this defect might be that downsizing needs to fit more pixel values to a smaller space therefore this created quality degradation. Now comparing the bilinear and nearest resized images, the bilinear images seem to have a higher contrast compared to the nearest image. This result could mostly be caused since bilinear uses more data to select each pixel value therefore creating a more dynamic image. Now for quantitative results we resized all of the previously resized images back to original size and compared them to the original image using RMSE. The results were very impressive it turns out

that nearest was the most accurate result when the image was upsized. The worst result unfortunately was bilinear when the image was downsized. Overall the results from RMSE tell us that maybe nearest might be better for some cases while bilinear is better for others. Another conclusion is that RMSE might not be the correct way to judge image restoration since pixel values differences is a very limited type of comparing for complex images.

Results

RMSE result 1 = nearest small (40x75) to big (300x300) = 21.36
RMSE result 2 = bilinear small (40x75) to big (300x300) = 26.82
RMSE result 3 = nearest big (425x600) to small (300x300) = 0
RMSE result 4 = bilinear big (425x600) to small (300x300) = 15.48

Nearest [40,75]



Bilinear [40,75]

Nearest [425,600]

Bilinear [425,600]

Code

```
function [resizedImage] = myimresize(image, resize, interpolation)
%myimresize
%   This function takes in an image and resizes it to either nearest or
%   bilinear interpolation
%
%Input: Image as a matrix, new size of two dimensions, type of Interpolation
```

```
%
%Output: New interpolated image as a matrix
%
%Example: myimresize(imread('test.tif'), [300,300], "bilinear")

    %find matrix size for rows and columns
    [xSize,ySize] = size(image);

    %create an empty matrix for resized image
    resizedImage = zeros([resize(1),resize(2)]);

    if interpolation == "nearest"

        disp('Computating nearest')

        %loop for going through all of the values needed for the new image
        for pixelX = 1:resize(1)
            for pixelY = 1:resize(2)

                %to find the closest x and y just round to nearest interger
                locatedXPixel = round((xSize)/(resize(1))*(pixelX - 0.5) +
0.5);
                locatedYPixel = round((ySize)/(resize(2))*(pixelY - 0.5) +
0.5);

                %place selected value to new resized matrix
                resizedImage(pixelX,pixelY) =
image(locatedXPixel,locatedYPixel);

            end
        end

        %changed format of matrix to 256
        resizedImage = uint8(resizedImage)







    elseif interpolation == "bilinear"
        disp('computating bilinear')


        %loop for going through all of the values needed for the new image
        for pixelX = 1:resize(1)
            for pixelY = 1:resize(2)

                %find pixel location using ratios
                x = (xSize)/(resize(1))*(pixelX - 0.5) + 0.5;
                y = (ySize)/(resize(2))*(pixelY - 0.5) + 0.5;


                %set of rules to find surounding pixel values
```

```
                % X

                %If its an integer
                if mod(x,1) == 0
                    m1 = x
                    m2 = x
                else
                    if x < 1
                        m1 = 1
                        m2 = 2
                    elseif x > xSize
                        m1= xSize-1
                        m2= xSize
                    else
                        m1 = floor(x)
                        m2 = ceil(x)
                    end
                end

                %Y

                %If its an integer
                if mod(y,1) == 0
                    n1=y
                    n2=y
                else
                    if y < 1
                        n1=1
                        n2=2
                    elseif y > ySize
                        n1=ySize-1
                        n2=ySize
                    else
                        n1 = floor(y)
                        n2 = ceil(y)
                    end
                end

                pixelLocs1 = [m1,n1,m1,n2,m2,n1,m2,n2]

                pixelVals1 =
[image(m1,n1),image(m1,n2),image(m2,n1),image(m2,n2)]

                %Call function to calculate bilinear value
                bilinearVal = mybilinear(pixelLocs1, pixelVals1, [x,y])

                %place selected value to new resized matrix
                resizedImage(pixelX,pixelY) = bilinearVal;

            end
        end

        %changed format of matrix to 256
        resizedImage = uint8(resizedImage)
```

```
    else

        %Prints if the input string for type of interpolation is unknown
        disp('could not understand interpolation')

    end

end
```

```
function [bilinearValue] = mybilinear(pixelLocs,pixelVals,interpoLoc)
%mybilinear
%
%   This function takes in an all 4 nearest pixel locations and values it
%   also takes in the new pixel location and outputs the calculation of
%   new pixel value.
%
%Input: 4 nearest pixel locations, 4 nearest pixel values, new pixel location
%
%Output: New pixel value


P51 = ((pixelVals(3) - pixelVals(1))* ((interpoLoc(1) -
pixelLocs(1))/(pixelLocs(5)-pixelLocs(1)))) + pixelVals(1)

P52 = ((pixelVals(3) - pixelVals(1))* ((interpoLoc(1) -
pixelLocs(3))/(pixelLocs(7)-pixelLocs(3)))) + pixelVals(2)

P5 = ((P52 - P51)* ((interpoLoc(2)-pixelLocs(2))/(pixelLocs(4)-
pixelLocs(2)))) + P51

bilinearValue = P5;

end

function [RMSE] = myRMSE(img1,img2)
    [r,c] = size(img1)

    pixelDiff = 0

    for x = 1:r
        for y = 1:c
            pixelDiff = pixelDiff + (double(img1(x,y))-double(img2(x,y)))^2
        end
    end

    RMSE = sqrt(pixelDiff/(r*c))

end
```