CSE 107 Lab 2

Image Resizing

Darren Correa

3/21/2022

Abstract

This lab aims to teach us how to scale images using two methods as well as testing how well our solutions were. Nearest Neighbor and Bilinear Interpolation. Nearest Neighbor interpolation is the process of assigning the pixel intensity value closest to the pixel whose value we want to find. Bilinear interpolation takes the 4 adjacent points surrounding the pixel in question and takes an average intensity value from the surrounding pixels. The methods differ in complexity and implementation strategies. I was successful in implementing the Nearest Neighbor method but had a hard time with Bilinear. The most challenging point was scaling the resized images back to their original size. For both methods, I was able to collect good, rescaled images. Logically, if I was able to down sample a large image, and I was able to up sample a smaller image, I should have been able to take either the up sampled, or down sampled images and down/up sample them again. This was not my case. When trying to up sample the down sampled images for both methods, I was given bound errors. This was odd because I those errors were not occurring when up sampling the original image. I was able to fix the errors for the Nearest Neighbor approach but not for Bilinear.

Technical Discussion

The Nearest Neighbor approach can be explained quickly. Our instructions told us the functions input should be the grayscale image, the size of the rows and columns for our scaled image and a string value to determine the interpolation method to be used (Nearest or Bilinear). The Nearest Neighbor interpolation function comes down to one line of code.

```
output(i,j) = img_matrix(round(i/resize_factor_rows),
round(j/resize_factor_cols));
```

`output(i,j)` is the pixel for which we are trying to find the intensity value for. The resize factors are the scale value between the original and resized image. For instance, if we want to transform a 300 x 300 image to a 600 x 600 image, our scale value is 2. For this lab, we separate the scale values separately by rows and columns since our scaled images are not squares. The right side of the code is saying assign the pixel intensity value from pixel closest to the location of the pixel of the scaled image. This is relative since the two matrix sizes are not equal. The scale factors account for the difference by scaling the location in the original image by the scaling factor of the resized image. After iterating through the matrix for the output image, we have collected enough pixel intensities to put together an image,

The Bilinear approach is a bit more complex. My implementation follows the Method 1 approach shown in our notes from lecture 9. Following the instructions of the lab, we needed to collect the location and pixel weights of 4 neighboring pixels. In this method, the pixel locations were used to get the pixel intensities. The locations were found using the same method as my nearest neighbor approach by using the scale factor to scale the location of original image to the relative location of the resized image. From that, lines 66 and 67 of imresize.m, we can gather the pixel intensities of the four surrounding pixels. I thought of it like a grid. If we are in the center of a grid and want to reach a corner using two axes', we have to travel up (+1) and left (-1), up (+1) and right (+1), down (-1) and left (-1), or down (-1) and right (+1). Having collected the pixel weights and the location of the pixel we want to have interpolated; we pass the information to the bilinear interpolation function. The pixel locations used inside the function as the notes made more sense to me in terms of having the x and y values of surrounding pixels inside. It follows the same principle as collecting the pixel weights. Rather than collecting the intensity values, we collect locations. Calculate the intensity of the interpolated pixel (IP) we did this in two steps. First compute the average location in the x dimension. Then we do the same in the y dimension. The pixel weights combined with average locations compute a pixel intensity value that average all four neighboring pixels for the pixel in our resized image.

Lastly, we were asked to test the similarity of our scaled images to our original image. To do this we had to scale our up or downscaled images back to their original 300x300 size. Then we had to implement the Root Mean Squared Error function.

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

This equation is used to estimate how well a reconstructed image compares to the original. It does this by taking the difference between all the pixel intensities from the resized image and the original. My implementation followed the equation very easily by taking in two images as inputs and using the size of the original image as our M and N. To find the sum of pixel intensities, we iterate through the size of the photos and take pixel values from each image at each iteration and take the difference between them. Squaring and adding that value to a total lets us keep track of the total difference between all pixels. The square root then gives us the actual value we are looking for that tells us how well our resized image matches the original.

Discussion of Results

Overall, I am satisfied with the image results from the different interpolation methods. I think Bilinear interpolation produced better results. To the naked eye, the down sampled image looks softer around the curves of the statue while shadows on the statue look finer. The up sampled images look very similar between both methods, so it is a lot harder to make a subjective decision. The RMSE results were troubling because there were issues getting the resized images back to the original size. As we can see with Images 2 through 5, the rescaling from the original image went fine. Images 6 and 7 show the issue with going back to original size. For Nearest Neighbor interpolation, I was getting out of bound errors when calculating the nearest pixel location. I was able to fix the case error for the down sampled image but as shown for the up sampled image, that wasn't the case. The same solution failed for both Bilinear interpolation images. Due to time constraints, I was unable to collect any results for those. From the values I was able to obtain, it is obvious that a smaller value is better. Since I was having trouble with scaling my images to the original images, for the bilinear image I used the built in resize function, just to get some results for my myRMSE function. Using the built-in function Bilinear interpolation produces better RMSE results, however, with how close my real result is to the built-in function result for Nearest Neighbor interpolation, I do think we need to keep objective evaluations to determine which results are better. As vision clarity varies person to person, we can't rely on our eyes when comparisons are this close.

Results

Image 1: Original 300x300 image

Image 2: 40x75 Down sampled Image using NN Interpolation

Image 3: 425x600 Image using NN Interpolation

Image 4: 40x75 Down sampled Image using BN Interpolation

Image 5: 425x600 Up Sampled Image using BN

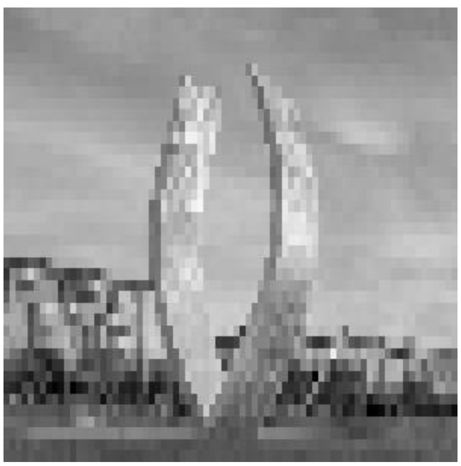

Image 6: 300x300 Resized Down Sample Image by NN

Image 7: 300x300 Resized Up Sample Image by NN

Table 1: RMSE Results

NN(40x75) = 18.9617	NN(425x600) = 120.521	*BN(40x75)	*BN(425x600) = 3.624

*Values obtained using built in resize function using Bilinear method.

Code

myResize.m

```
function [output] = myimresize(img_matrix, rows, cols, resize_type)
% myresize      The function should take in an image matrix, the number of
%               rows and colums the image should be resized too and a
%               string of what method of interpolation to be used, Nearest
%               or Bilinear.
%
%Syntax:
%   output = myimresize(img_matrix, rows, cols, resize_type)
%
%Input:
%   img_matrix = Matrix gathered from input image
%   rows = # of rows for new image size
%   cols = # of columns for new image size
%   resize_type = Method of interpolation (Select either Nearest or
%                   Bilinear)
%
%History:
%   D. Correa       3/14/2022   Created
%   D. Correa       3/18/2022   Completed Nearest Neighbor Function
%   D. Correa       3/21/2022   Completed Bilinear Function
%   D. Correa       3/21/2022   Tested and fixed out of bound erros
%

    %Nearest Neighbor Section

    if resize_type == "Nearest"
    %Create empty output matrix
    % and find scaling factor for rows and colums
    output = zeros([rows,cols],'uint8');
    resize_factor_rows = rows/size(img_matrix,1);
    resize_factor_cols = cols/size(img_matrix,2);

    %Ensure interpolation method is correct
        for i = 1:rows
            for j = 1:cols
                %for each pixel in output, assign intensity from nearest
                %neigbor
                if (i/resize_factor_rows) < rows  && (j/resize_factor_cols) <
cols
                output(i,j) = img_matrix(round(i/resize_factor_rows),
round(j/resize_factor_cols));
                end
            end
        end

    end


    %Bilinear Interpolation Section
```

```
    if resize_type == "Bilinear"
    %Create empty matrix for output image
    B = zeros([rows,cols], 'uint8');

    %Get factors of how the rows and columns are scaled from original image
    scale_rows = rows/size(img_matrix,1);
    scale_cols = cols/size(img_matrix,2);

    %To keep from out of bound cases, begin iteration at 3 and end at max
    %num rows/cols - 3.
    %Values < 3 result in out of bound errors.
    for i = 3:rows-3
        for j = 3:cols-3

            %Get equivalent positions from original image
            inputImgRowPos = round(i/scale_rows);
            inputImgColPos = round(j/scale_cols);


            %Get pixel intensities from 4 surrounding pixels
            PW_topL = img_matrix(inputImgRowPos+1,inputImgColPos-1);
            PW_topR = img_matrix(inputImgRowPos+1,inputImgColPos+1);
            PW_botL = img_matrix(inputImgRowPos-1,inputImgColPos-1);
            PW_botR = img_matrix(inputImgRowPos-1,inputImgColPos+1);

            %Call mybilinear function to get intensity value of
            %interpolated pixel
            IP = mybilinear(PW_topL, PW_topR, PW_botL, PW_botR,
inputImgRowPos, inputImgColPos, scale_rows, scale_cols);

            %Assign interpolated pixel intensity to location scaled image
            B(i,j) = IP;

        end
    end

    %Show and save returned image matrix from bilinear interpolation
    imshow(B, []);
    %imwrite(output, '40x75_Downsampled_BN.png');
    %imwrite(output, '425x600_UpSampled_BN.png');


    %imwrite(B, '300x300_Resized_Downsampled_BN.png');
    %imwrite(B, '300x300_Resized_Upsampled_BN.png');

    end

end
```

myBilinear.m

```
function IP = mybilinear(PW_topL, PW_topR, PW_botL, PW_botR, inputImgRowPos, 
inputImgColPos, scale_rows, scale_cols)
% mybilinear    Computes interpolated pixel value using 4 adjacent pixel
%               weights, location of interpolated pixel, and scale factors.
%
%Syntax:
%   IP = mybilinear(PW_topL, PW_topR, PW_botL, PW_botR, inputImgRowPos, 
inputImgColPos, scale_rows, scale_cols)
%
%Input:
%   PW_topL = Pixel wieght of pixel above and to the right of interpolated
%               pixel.
%   PW_topR = Pixel wieight of pixel above and to the right of interpolated
%               pixel.
%   PW_botL = Pixel wieight of pixel below and to the left of interpolated
%               pixel.
%   PW_botR = Pixel wieight of pixel below and to the right of interpolated
%               pixel.
%   inputImgRowPos = Relative row position of interpolated pixel from 
original
%                       image.
%   inputImgColPos = Relative column position of interpolated pixel from 
original
%                       image.
%   scale_rows = Scale factor between original row size and new row size.
%   scale_cols = Scale factor between original column size and new column
%                   size.
%
%Output:
%   IP = Pixel weight of interpolated pixel.
%
%History:
%   D. Correa   3/18/2022   Created
%   D. Correa   3/21/2022   Function completed
%   D. Correa   3/21/2022   Tested and fixed issues with location variables
%                           in p5_1 and p5_2.

            %Calculate position to scaled image
            y = round(inputImgRowPos*scale_rows);
            y1 = round((inputImgRowPos+1)*scale_rows);
            y2 = round((inputImgRowPos-1)*scale_rows);
            
            x = round(inputImgColPos*scale_cols);
            x1 = round((inputImgColPos-1)*scale_cols);
            x2 = round((inputImgColPos+1)*scale_cols);
            
            
            
            %Interpolate using intesities of 4 neighboring points
            p5_1 = ((x2-x) / (x2-x1)) * PW_botL + ((x-x1) / (x2-x1)) * 
PW_botR;
            p5_2 = ((x2-x) / (x2-x1)) * PW_topL + ((x-x1) / (x2-x1)) * 
PW_topR;
```

```
            IP = round(((y2-y) / (y2-y1)) * p5_1 + ((y-y1) / (y2-y1)) *
p5_2);

end
```

myRMSE.m

```
function RMSE = myRMSE(rows, cols, img1, img2)
% RMSE  Compares image reconstruction accuracy between original image and
%       scaled images.
%
%Syntax:
%   RMSE = myRMSE(rows,cols,img1,img2)
%
%Input:
%   rows = # of rows in original image matrix.
%   cols = # of colums in original image matrix.
%   img1 = Resized scaled image.
%   img2 = Orignial image
%
%Output:
%   RMSE = float value that shows accuracy or resized image to original
%           image/
%
%History:
% D. Correa     3/21/2022   Created, completed and tested.

    total = double(0);
    for i = 1:rows
        for j = 1:cols
            %Get sum of difference between image pixel intensities
            SQ = double((img1(i,j)-img2(i,j)));
            total = total + SQ^(2);
        end
    end
    %Calculate accuracy value.
    RMSE = sqrt((1/(rows*cols)*total));

end
```

```
%Script for running myimresise, mybilinear and myRMSE
```

Script.m

```
%
%For testing, ensure rows and images to be used are correct and inputs are
%valid.
%
% D. Correa     3/14/2022   Created
% D. Correa     3/18/2022   Added more commands for images to be read for
%                           testing.
% D. Correa     3/21/2022   Added more commands for testing RMSE function


%Get image matrix from input image
img_matrix = imread('Lab_02_image1.tif');
%img_matrix = imread('40x75_Downsampled_NN.png');
%img_matrix = imread('40x75_Downsampled_BN.png');
%img_matrix = imread('425x600_UpSampled_NN.png');
%img_matrix = imread('425x600_UpSampled_BN.png');


%New size for resized image matrix
%rows = 40;
%cols = 75;

%rows = 425;
%cols = 600;

rows = 300;
cols = 300;

%Declare interpolation method to be used
%resize_type = "Nearest";
resize_type = "Bilinear";

%Generate output image using myimresize function
%output = myimresize(img_matrix, rows, cols, resize_type);


%Show and save returned image matrix
%imshow(output, []);
%imwrite(output, '40x75_Downsampled_NN.png');
%imwrite(output, '425x600_UpSampled_NN.png');

%imwrite(output, '300x300_Resized_Downsampled_NN.png');
%imwrite(output, '300x300_Resized_Upsampled_NN.png');

%img2 = imread('300x300_Resized_Downsampled_NN.png');
img2 = imread('300x300_Resized_UpSampled_NN.png');
%img2 = imread('300x300_Resized_Downsampled_BN.png');
%img2 = imread('300x300_Resized_UpSampled_BN.png');


RMSE = myRMSE(rows, cols, img_matrix, img2);
```

Script2.m

```
%Script for testing RMSE values of Bilinear interpolation
%
%History:
% D. Correa     3/21/2022   Created
%

orig = imread('Lab_02_image1.tif');

%img1 = imread('40x75_Downsampled_BN.png');
%img2 = imread('425x600_UpSampled_BN.png');

%A = imresize(img1, [300,300], 'Method','bilinear');
%imwrite(A, '300x300_Resized_Downsampled_BN.png');

%B = imresize(img2, [300,300], 'Method','bilinear');
%imwrite(B, '300x300_Resized_UpSampled_BN.png');

%img = imread('300x300_Resized_Downsampled_BN.png');
img = imread('300x300_Resized_UpSampled_BN.png');

RMSE = myRMSE(300,300,img,orig);
```