Image Resizing

Lab Assignment #2

Ralphilou Tatoy

March 9, 2022

Abstract:

This lab is about image processing of grayscale images into a resized image. Either nearest neighbor or bilinear interpolation methods to perform the resizing of the images. A process called downsampling is resizing into smaller images and resizing images into larger images, a process called upsampling. The output resized image should resemble the input image. Using the output images, we can determine the Root-mean-square deviation (RMSE) to calculate the difference between the original image and the transformed image.

Technical Directions:

Nearest neighbor or bilinear interpolation are the two methods we used for this lab to perform image resizing. Each method is different in terms of how to calculate and some can be performed in different ways. For both methods, I used an equation (Eq.1 and Eq. 2, myimresize code) that can proportionately direct me to the corresponding image from the transformed image to the original image. This equation helps the algorithm to synthesize the correct pixel value that the transformed pixel location should be. Thus, getting a near-replica of the original image in a resized image.

The nearest neighbor algorithm uses the output coordinates to pick the closest coordinates to get the pixel value from the original image. Need to account for a decimal value that is less than one as that MATLAB doesn't start the iteration of the matrix by zero. For example, if the output coordinate is 0.5, we don't want the coordinate to be zero as that will output an error. The cases are less than 1, more than the original max length, and between. Depending on the cases, I used either round, ceil, or floor to get a positive integer number. The function should be able to collect the coordinates from both axes. Once the cases are settled, the pixel value are transmitted from the original image to the resize image appropriately.

The bilinear interpolation algorithm is slightly longer process. First, I used the same method of finding the cases such as with the nearest neighbor algorithm to appropriately be able to access the pixel value of the original image. The difference is that to take account of not just one coordinate but three more coordinates to the process. You can use the same process as the nearest neighbor to get the first out of four coordinates. You can also use the coordinate to get the other by adding one to either the x, y, or both. Once the coordinates are collected, the pixel value of each coordinate pairs must be taken into a data structure. Both the coordinates and pixel value of each collected coordinates are used for the bilinear algorithm.

The bilinear algorithm can be structured differently by either using the geometry of each point or using the polynomial relationship to find the pixel value. I used the polynomial method. In the polynomial method, there is the same equation used for each coordinate pair. All coordinates can be represented via matrices (Eq.3, mybilinear code). The equation uses the pixel value, coordinates, and the desired variables (a,b,c,d). Once the desired variable is found (Eq.4, mybilinear code), the algorithm can use it with the same equation format to find the pixel value that is designated to the transformed image (Eq.5, mybilinear code).

The root-mean-squared deviation is an equation that can determine the difference between two images. The input for this is two images that are the original image and the transformed image. First, calculate the difference between each pixel value with the same coordinate locations and square the value (Eq. 5, myRMSE code). Add each of that value throughout both images. The equation is adding the difference for all pixel locations in an image, there must be a division of the size of both images (Eq. 6, myRMSE code). The square root nullifies the effects of the squared factor. The squared is there for negative cases because we don't want the addition to be skewed by a negative integer or number.

Both methods will give the pixel value for the transformed image and output a resize image.

Results

Downsampling first using nearest neighbor:

Figures 2 and 3 are images are outputted using this process. The output image that we want to compare to the original is blurry. So, the RMSE is not as low as it would be.

Downsampling first using bilinear interpolation:

Figures 6 and 7 are outputted during this process of downsampling and then upsampling to regain the original image. The output image seems to be blurry just like downsampling using the nearest neighbor. It seems as when the image is shrunk, the information that gave the image clarity is lost. The RMSE is higher than the RMSE of the image using the nearest neighbor both are recognizable but blurry.

Upsampling first using nearest neighbor:

Figures 4 and 5 are the images that are outputted using this process. The nearest neighbor seems to output a better image to compare to the original image. The output 300x300 during this process output a similar image to the eye.

Up sampling first using bilinear interpolation:

Figures 8 and 9 are the output images using this process. The image is clear, but the RMSE is higher than using the nearest neighbor by doing upsampling first.

Conclusion: Nearest neighbor seem to be the better procedure for this process of resizing the image. Doing upsampling first and then retrieving the original size seem to not lose information of the original image. Doing downsampling and then retrieving the original size for both processes seem to output not as good of detail as the original image.

Figure 1: Original Image

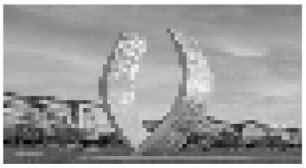

Figure 2: Nearest Neighbor 300x300 (Original Image) to 40x75

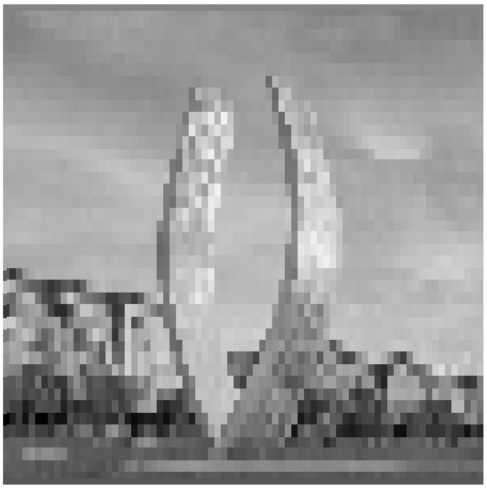

Figure 3: Nearest Neighbor 40x75 to 300x300

Figure 4: Original Image 300x300 to 425x600 using nearest neighbor

Figure 5: 425x600 image to 300x300 using nearest neighbor

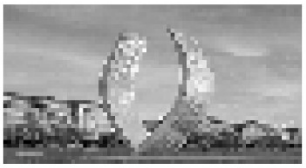

Figure 6: Original image 300x300 to 40x75 using Bilinear Interpolation

Figure 7:40x75 to 300x300 using Bilinear Interpolation

Figure 8: Original 300x300 to 425x600 using Bilinear Interpolation

Figure 9: 425x600 to 300x300 using Bilinear Interpolation

```
%Image input
A = imread("Lab_02_image1.tif");

%downsample using nearest neighbor
B = myimresize(A,[40,75],'nearest');
B1 = myimresize(B,[300,300],'nearest');

% %upsample using nearest neighbor
C = myimresize(A,[425,600],'nearest');
C1 = myimresize(C,[300,300],'nearest');

imshow(A);
figure;
%Down Sampling process (Original Image 300x300 -> 40x75 -> 300x300)
imshow(B);
figure;
imshow(B1);
figure;

%Upsampling process (Original Image 300x300 0> 425x600 0> 300x300)
imshow(C);
figure;
imshow(C1);
figure;

%downsampling using bilinear interpolation

D = myimresize(A,[40,75],'bilinear');
D1 = myimresize(D,[300,300],'bilinear');

%upsampling using bilinear interpolation
E = myimresize(A,[425,600],'bilinear');
E1 = myimresize(E,[300,300],'bilinear');

%Downsampling process (Original Image 300x300 -> 40x75 -> 300x300)
imshow(D);
figure;
imshow(D1);
figure;

%Upsampling process (Original Image 300x300 0> 425x600 0> 300x300)
imshow(E);
figure;
imshow(E1);

%Finding error of transforming image
Bs = myRMSE(A,B1);
fprintf('RMSE of downsampling using nearest neighbor %f\n', Bs);
Ds = myRMSE(A,D1);
```

```
fprintf('RMSE of downsampling using bilinear interpolation %f\n', Ds);
Cu = myRMSE(A,C1);
fprintf('RMSE of upsampling using nearest neighbor %f\n', Cu);
Eu = myRMSE(A,E1);
fprintf('RMSE of upsampling using bilinear interpolation %f\n', Eu);
```

```
function B = myimresize(A, insize,method)
% Functionality - The input image is transformed into a resize function and outputs ↙
desired
% size of the input image.
%
% Syntax:
% out = myimresize(A, insize, method)
%
% Input:
% A - input image
% insize - size of desired output image
% method - type of interpolation
%
% Output: resize image
% B -  resized image

%output image size
M = insize(1);
N = insize(2);

%input image size
[Minput, Ninput] = size(A);

%Create output image
B = uint8(zeros(M,N));

%Interpolation process
for m=1:M
    for n=1:N

        %Interpolation method chosen
        %Choose the location for the original image to compare.
        xCH = (Minput/M)*(m-0.5)+0.5; %Eq. 1
        yCH = (Ninput/N)*(n-0.5)+0.5; %Eq. 2


        if strcmp(method,'nearest')
            %Nearest Neighbor interpolation
            %Determine value for the pixel location
            %Choosing which coordinate is viable
            if(xCH < 1)
                nearestx = ceil(xCH);
            end
            if(yCH < 1)
                nearesty = ceil(yCH);
            end
            if(xCH>Minput)

                nearestx = floor(xCH);
```

```
        end
        if(yCH > Ninput)
            nearesty = floor(yCH);

        end
        if(xCH>=1 && yCH>=1&& xCH<=Minput && yCH<=Ninput)
            nearestx = round(xCH);
            nearesty = round(yCH);

        end



        %Assigned pixel value to the location chosen
        B(m,n) = A(nearestx,nearesty);
    elseif strcmp(method,'bilinear')
        %Bilinear Interpolation
        %Determine value for the pixel location
        %Choosing which coordinate is viable
        if(xCH < 1)
            billx = ceil(xCH);
        end
        if(yCH < 1)
            billy = ceil(yCH);
        end
        if(xCH>Minput)

            billx = floor(xCH);
        end
        if(yCH > Ninput)
            billy = floor(yCH);

        end
        if(xCH>=1 && yCH>=1&& xCH<=Minput && yCH<=Ninput)
            billx = round(xCH);
            billy = round(yCH);

        end
        %First coordinate to use for bilienar interpolation
        x(1) = billx;
        y(1) = billy;
        p(1)= A(x(1),y(1));

        %Second coordinate
        if (billx+1 <= Minput)
            x(2) = billx+1;
            y(2) = billy;
            p(2) = A(x(2),y(2));
        else
```

```
            x(2) = billx+1;
            y(2) = billy;
            p(2) = A(x(1),y(1));
        end

        %Third coordinate
        if (billy+1 <= Ninput)
            x(3) = billx;
            y(3) = billy+1;
            p(3) = A(x(3),y(3));
        else
            x(3) = billx;
            y(3) = billy+1;
            p(3) = A(x(1),y(1));
        end
        %Fourth coordinate
        if(billx+1 <= Minput && billy+1 <= Ninput)
            x(4) = billx+1;
            y(4) = billy+1;
            p(4) = A(x(4),y(4));
        else
            x(4) = billx+1;
            y(4) = billy+1;
            p(4) = A(x(1),y(1));
        end

        %The coordinate of the pixel location of the transformed image
        x(5) = xCH;
        y(5) = yCH;

        %Bilinear function call.
        pixelValue = mybilinear(x,y,p);

        %Pixel value assignment
        B(m,n) = uint8(pixelValue);


    end
  end
end
```

```
function v = mybilinear(x,y,p)
% Using bilinear interpolation to calculate the value of pixel depending on
% the four points sorrounding the chosen points
%
%
% Syntax:
% v = mybilinear(x,y,p)
%
% Input:
% x - x coordinate of the output pixel
% y - y coordinate of the output pixel
%
% Output:
% p - pixel value of each coordinate from the original image using
% appropriate x and y coordinates

x1y1 = x(1)*y(1);
x2y2 = x(2)*y(2);
x3y3 = x(3)*y(3);
x4y4 = x(4)*y(4);

% A matrix
a1 = [x(1),y(1),x1y1,1;x(2),y(2),x2y2,1;x(3),y(3),x3y3,1;x(4),y(4),x4y4,1]; %Eq. 3

% uint8 to double for matrix inverse and multiplication purposes
pixel1 = double(p(1));
pixel2= double(p(2));
pixel3 = double(p(3));
pixel4 = double(p(4));

%Pixel value from original image
h = [pixel1;pixel2;pixel3;pixel4];

%Getting a,b,c,d
g = a1\h; %Eq.4

%Equation to get pixel value
v = round(g(1)*x(5) + g(2)*y(5) + g(3)*x(5)*y(5)+g(4)); %Eq.5

% Another method:
% Interpolation between 1 and 3
% p51 = ((pixel3-pixel1)*((x(5)-x(1))/(x(3)-x(1))))+pixel1;

% Interpolation between 4 and 2
% p52 = ((pixel4-pixel2)*((x(5)-x(2))/(x(4)-x(2))))+pixel2;

% Interpolation between 1 and 2
% v = ((p52-p51)*((y(5)-y(1))/(y(2)-y(1)))) + p51;
```

```
function out = myRMSE(image1,image2)
% Functionality: Implementing calculation of error to determine difference  between two images
%
% Syntax:
% out = myRMSE(image1,image2)
%
% Input:
% image1 - original image
% image2 - transformed image
%
% Output:
% out - the average error or difference of each pixel

[Minput1, Ninput1] = size(image1);

%Initialize
RMSE = 0;
for m=1:Minput1
    for n=1:Ninput1
         %Add difference and squared
        RMSE = RMSE + (double(image1(m,n)) - double(image2(m,n)))^2;  % Eq. 5
    end
end
%Proportionate to the size
RMSE1 = RMSE/(Minput1*Ninput1); %Eq. 6
%Square root
RMSE2 = sqrt(RMSE1); %Eq. 7
%output
out = RMSE2;
```