100296766 03/21/2022

CSE 107
Lab#2 Assignment

Image Resizing

Andre Martin

Abstract: In this lab, you will implement a function that resizes a grayscale image. It will perform either nearest neighbor or bilinear interpolation to generate the resized image. Resizing an image to a smaller size is called downsampling. Resizing an image to a larger size is called upsampling.

Technical Discussion: The execution for image resizing relies upon how much refining you need to apply. The initial step to compute the resizing we want to observe the ratio's between the old picture size and the new picture size. Figure 1 underneath shows how this can be determined by utilizing the basic function. M is the first image size and M' is the new image size. The input of this function is m' which is a point on the first image, the result is the determined point on the new image. This is the most direct method since pixels are careful which implies they are not linear, you need to find values of blocks. The primary strategy is called Nearest Interpolation which basically implies that the nearest pixel close to the target coordinate turns into that pixels value. The more confounded technique called Bilinear Interpolation includes taking the average of the closest pixels. To see the distances and afterward the values of the pixels I utilized the technique of triangle ratio's which included taking the 4 closest pixels. When comparing the aftereffects of resizing and looking at these pictures, it may be exceptionally hard to tell the difference in light of the fact that it essentially reuses a considerable lot of the pixels we utilized. Figure 2 beneath, it is known as the root mean square deviation. For the most part, it needs two contributions for this execution, we utilized pixel values from two related pictures. The result of this situation is a deviation which implies how unique the two inputs are from one another. This is exceptionally useful since it will let us know how unique the pictures turned out by utilizing RMSE. The higher the RMSE the less similar the two pictures are.

Figure 1

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

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: Reasonably you can truly tell that the images that were downsized and ended up being the most impacted in a negative way. The justification behind this defect may be that the downsizing needs to fit more pixel values to a more modest space thus making quality degradation.When looking at the bilinear and closest resized pictures, the bilinear pictures appear to have a higher contrast with the closest picture. This outcome could generally be caused since bilinear uses more info to choose every pixel value consequently making a more powerful picture. As for quantitative outcomes we resized each of the recently resized pictures back to unique size and contrasted them with the original picture using RMSE. The outcomes would appear close to the most dependable outcome when the picture was upsized. The worst outcome was bilinear when the picture was downsized. Furthermore, when seeing the outcomes from RMSE this lets us know that perhaps the closest result may be better for certain cases while bilinear is better for other people. In the end RMSE probably won't be the right method for making a decision about picture restoration since the pixel values differences is a restricted type in sorting and looking at complex pictures.

Results:

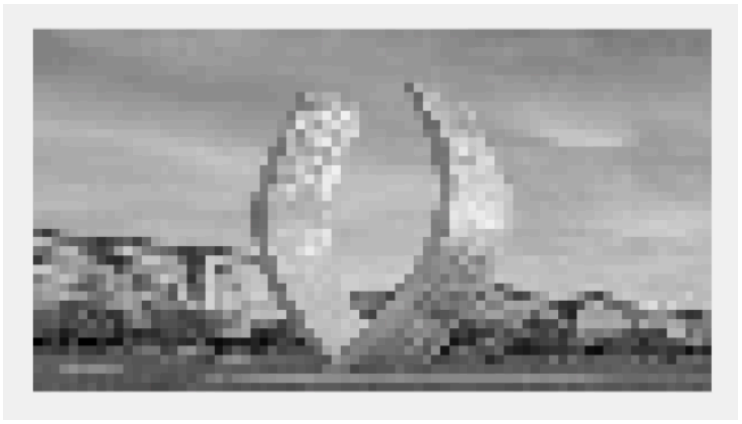
RMSE result 1 = nearest small (40x75) to big (300x300) =~ 21.4

RMSE result 2 = bilinear small (40x75) to big (300x300) =~ 26.8

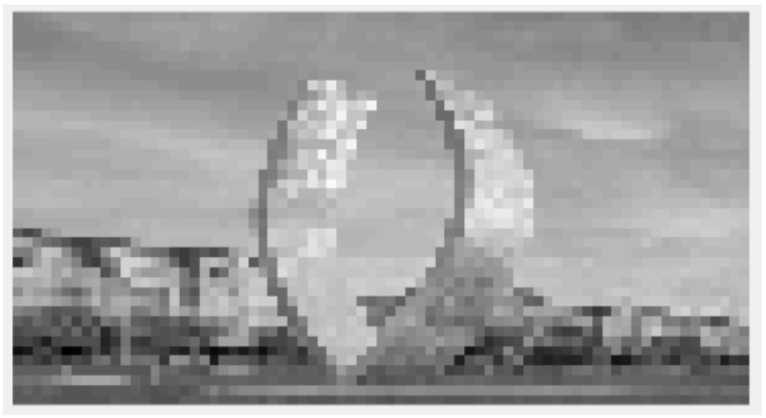
RMSE result 3 = nearest big (425x600) to small (300x300) = 0

RMSE result 4 = bilinear big (425x600) to small (300x300) =~ 15.5

Nearest [40x75]:



Bilinear [40x75]:



Nearest [425x600]:

Bilinear [425x600]:

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
         %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)
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
```