CMPEN/EE455: Digital Image Processing I

Computer Project # 1:

Lab Introduction and Digital Image Quantization

Ritvik Muralidharan, Georges Junior Naddaf, Zian Wang

Date: 09/08/2017

A. Objectives

This project's main purpose is to

- Introduce the team to standard project requirements
- Introduce the team to basic image processing concepts
- Introduce the team to MATLAB's capabilities for digital image processing.

B. Methods

First and foremost, the code is run by running the main Proj1RE.m file. We have a separate file called bilinear_interpolation that performs the task required in task 2 and is plugged in the main file. We also had to change the image format from 512*512*2 to 512*512 and this was done using `f = f(:,:,1);.` We will be going over the algorithms implemented for every task.

For task 1:

We performed downscaling of images by simply skipping a set amount of pixels depending on the image that we had to produce. For example, to downscale a 512x512 image to 256x256, we skip 1 pixel jumping 2 pixels at a time and rely on two nested loops.

Then we had to perform nearest-neighbor interpolation:

```
scale = [2 2];
oldSize = size(f1);
newSize = max(floor(scale.*oldSize(1:2)),1);
```

The first part of code simply sets the present and desired sizes of the images through scale.

```
rowIndex = min(round(((1:newSize(1))-0.5)./scale(1)+0.5),oldSize(1));
colIndex = min(round(((1:newSize(2))-0.5)./scale(2)+0.5),oldSize(2));
```

Each of these lines of code goes pixel by pixel. Each pixel has a range (0-1,1-2,...) and that is why we perform -0.5 to be at the center of that range. Then for each value, we divide by the scale (similar to creating cells of the same value) and then add 0.5 again to return to its original location and perform rounding in case of decimals. Then we perform a minimum operation on the pixel values and compare them to oldSize() to make sure that we don't have any pixel that is smaller than that of oldSize(). This gives us the indexes that we can use to iterate and fill up the new and desired matrix/image.

<u>For task 2:</u>

We utilized the bilinear interpolation algorithm (similar to linear interpolation) to process and upscale the 32X32 image to a 512x512 image.

This was done by taking the input image's pixels and inserting them into the 512x512 image with scaled spacing and with a scaling factor of 16.

```
[height,width,depth]=size(im);
factor = 16;
for i=1:height
    for j=1:width
       im1(1+(i-1)*factor,1+(j-1)*factor,:)=im(i,j,:);
    end
end
```

The equation was taken from Proj1-Interpolation.pdf. It takes 2 corner pixels based on the X coordinate and creates a line between these 2 points. Any X coordinate that is covered by the line could have its corresponding value predicted. Then, the other 2 corner pixels are processed with the same steps as above. Next, the Y coordinate is taken into consideration to estimate the value of the points that are within the line coverage. Thus, it can estimate all the pixel values based on the pixel X,Y coordinates.

```
i=1;
j=1;
while i<1+(height-1)*factor
    j=1;
    while j<1+(width-1)*factor

       if (~((rem(i-1,factor)==0) && (rem(j-1,factor)==0)))
           h0=im1(
ceil(i/factor)*factor-factor+1,ceil(j/factor)*factor-factor+1,:);
           h1=im1(
ceil(i/factor)*factor-factor+1+factor,ceil(j/factor)*factor-factor+1,:);
           h2=im1(
ceil(i/factor)*factor-factor+1,ceil(j/factor)*factor-factor+1+factor,:);
           h3=im1(
ceil(i/factor)*factor-factor+1+factor,ceil(j/factor)*factor-factor+1+facto
r,:);

           x=rem(i-1,factor);
           y=rem(j-1,factor);

           dx=x/factor;
           dy=y/factor;

           b1=h0;
           b2=h1-h0;
           b3=h2-h0;
           b4=h0-h1-h2+h3;
           im1(i,j,:)=b1+b2*dx+b3*dy+b4*dx*dy;
       end
       j = j+1;
    end
    i = i+1;
  %imshow(cast(im1,'uint8'));
end
```

By using this bilinear interpolation method, we were able to scale the 32x32 image to 512x512 image with limited image processing steps applied.

For Task 3:

We had to decrease the grey-level quantization of the original image by decreasing the number of bits per pixel (from 8 bits all the way to 1bit/pixel) and then output the corresponding images. As specified in the question, we had to make sure that the grey levels span the 8 bit range in the new images. The code for each bit/pixel is similar but with different conditions for each case. For example, for the 6-bit grey case, we simply iterated over the array and compared the remainder of f(x,y)/4 to the midpoint (2) so that we know if we need to round the grey-scale up or down if the pixel value falls within that range. One thing to note is for each example, as the bit/pixel decreases by 1, the number that we compare our pixels to changes by a factor of 2.

```
%6bit grey
for x = 1:1:M
    for y = 1:1:N
        if(rem(f(x,y),4)==0)
            g6(x,y) = f(x,y);
        elseif (rem(f(x,y),4) >=2 && (f(x,y) <= 192))
            g6(x,y) = f(x,y) + 4 - rem(f(x,y),4);
        else
            g6(x,y) = f(x,y) - rem(f(x,y),4);
        end
    end
end


imwrite(g6,'g6.tif');
imtool(g6);
```

The same reasoning applied to the 5 bit grey function. Each if statement in the function covers a case in which the pixel of the image may fall into.

```
%5bit grey
for x = 1:1:M
    for y = 1:1:N
        if(rem(f(x,y),8)==0)
            g5(x,y) = f(x,y);
        elseif (rem(f(x,y),8) >=4 && (f(x,y) <= 192))
            g5(x,y) = f(x,y) + 8 - rem(f(x,y),8);
        else
            g5(x,y) = f(x,y) - rem(f(x,y),8);
        end
    end
end


imwrite(g5,'g5.tif');
imtool(g5);
```

The 1 bit grey is the last function in the series that simply determines white/black pixels.

```
%1bit grey
for x = 1:1:M
    for y = 1:1:N
        if(f(x,y)>= 128)
            g1(x,y) = 255;
        else
            g1(x,y) = 0;
        end
    end
end


imwrite(g1,'g1.tif');
imtool(g1);
```

For Task 4:

We took the already down scaled 256x256 image from earlier and decreased the grey-level quantization by using the same method used in task 3.

```
[M, N] = size(f1);

for x = 1:1:M
    for y = 1:1:N
        if(rem(f1(x,y),4)==0)
            g6_256(x,y) = f1(x,y);
        elseif (rem(f1(x,y),4) >=2 && (f1(x,y) <= 192))
            g6_256(x,y) = f1(x,y) + 4 - rem(f1(x,y),4);
        else
            g6_256(x,y) = f1(x,y) - rem(f1(x,y),4);
        end
    end
end

imwrite(g6_256,'g6_256.tif');
imtool(g6_256);
```

Proj1RE.m is our actual project code.

bilinear_interpolation.m is our bilinear interpolation function.

C. Results

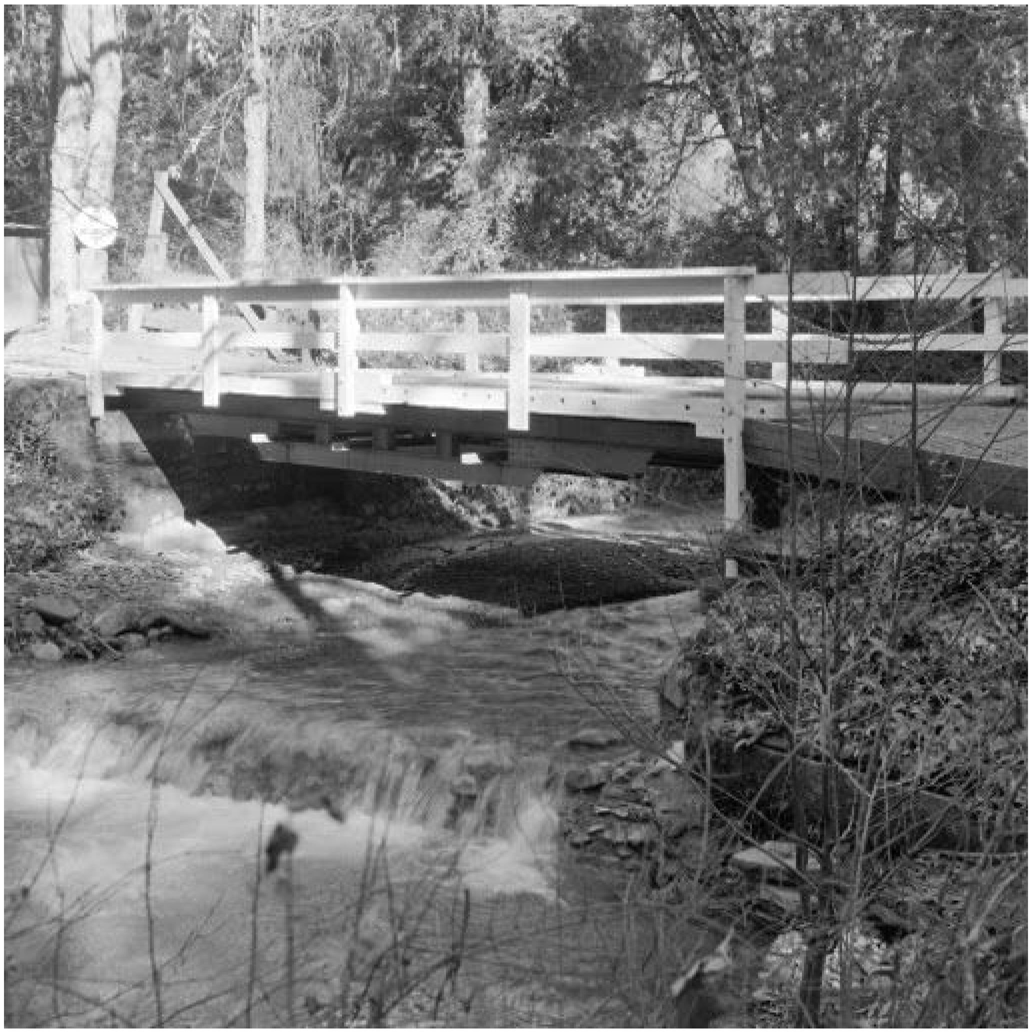
Figure 1. Original “walkbridge.tif” image.

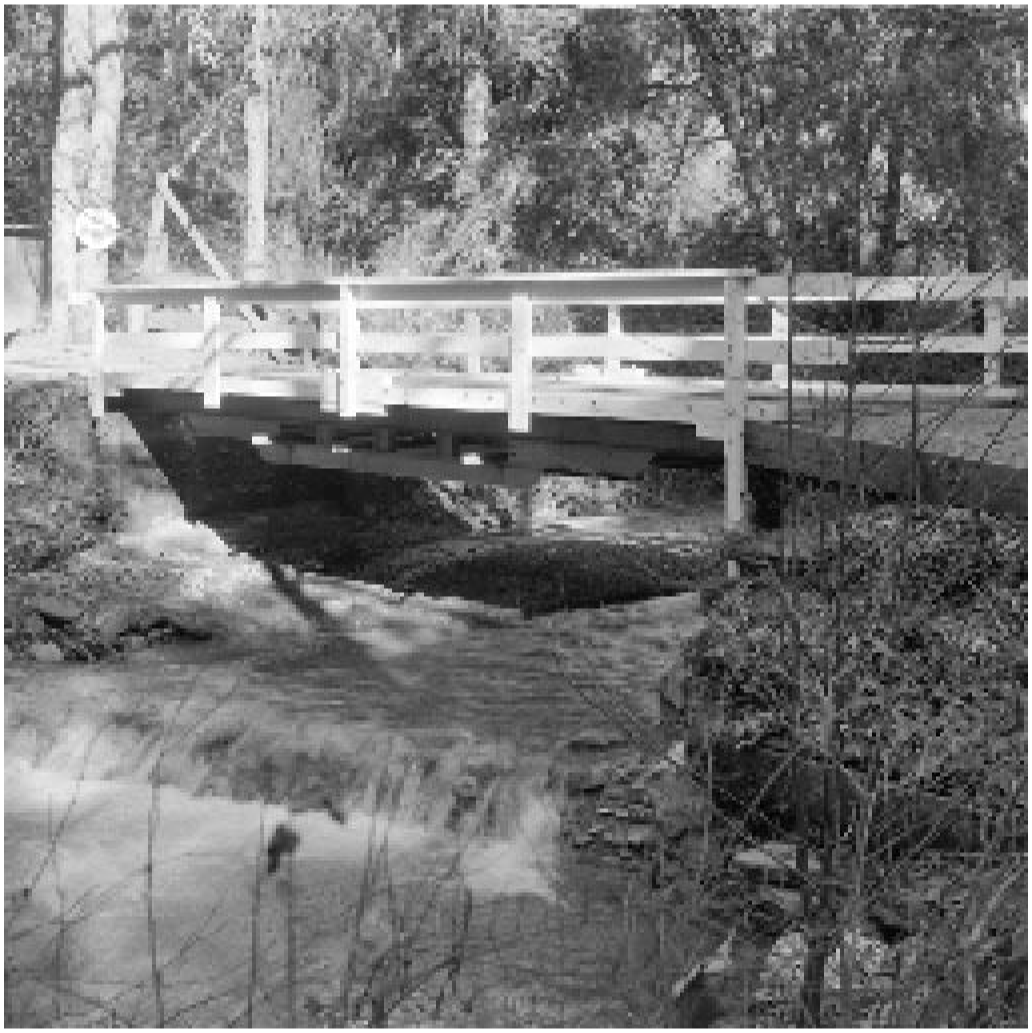

Figure 2. Results after changing the spatial resolution of the original “walkbridge.tif” image from 512x512 to 256x256, then saved as a 512x512 image.

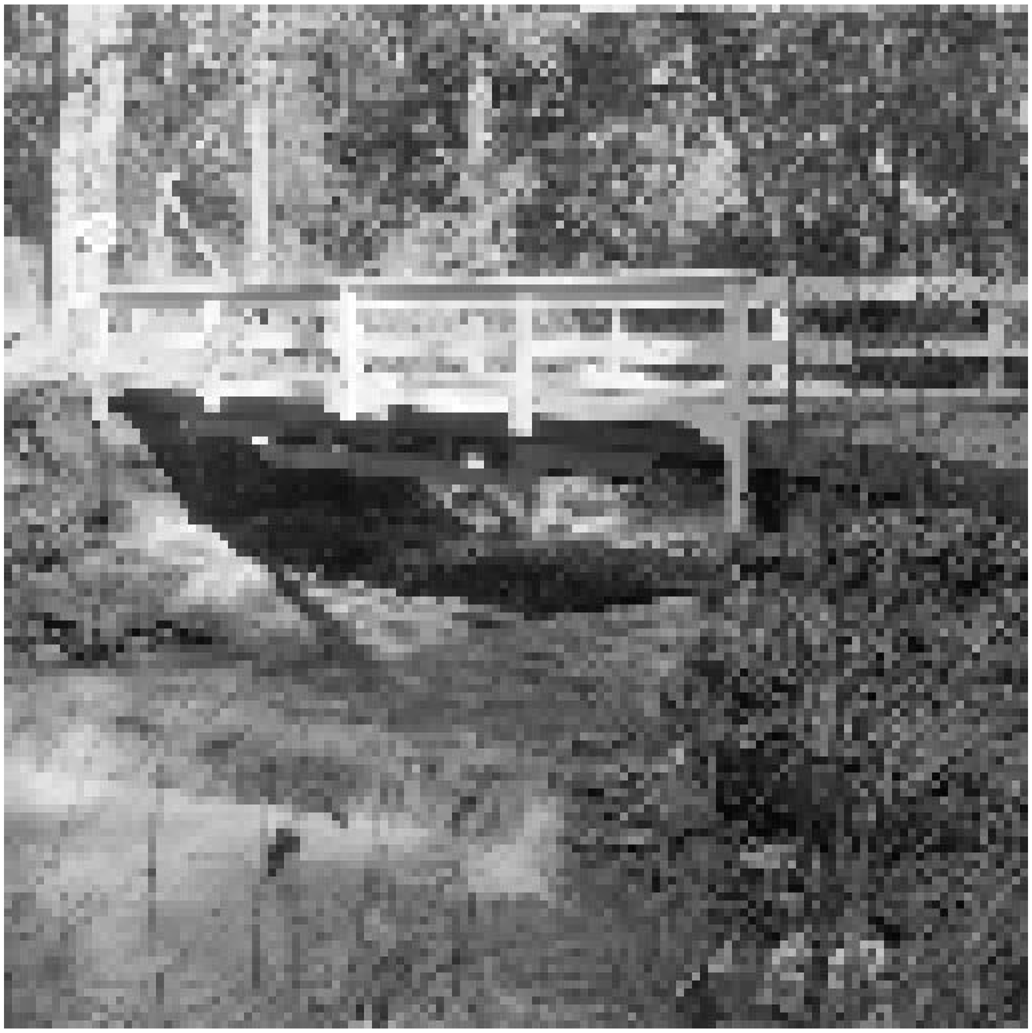

Figure 3. Results after changing the spatial resolution of the original “walkbridge.tif” image from 512x512 to 128x128, then saved as a 512x512 image.

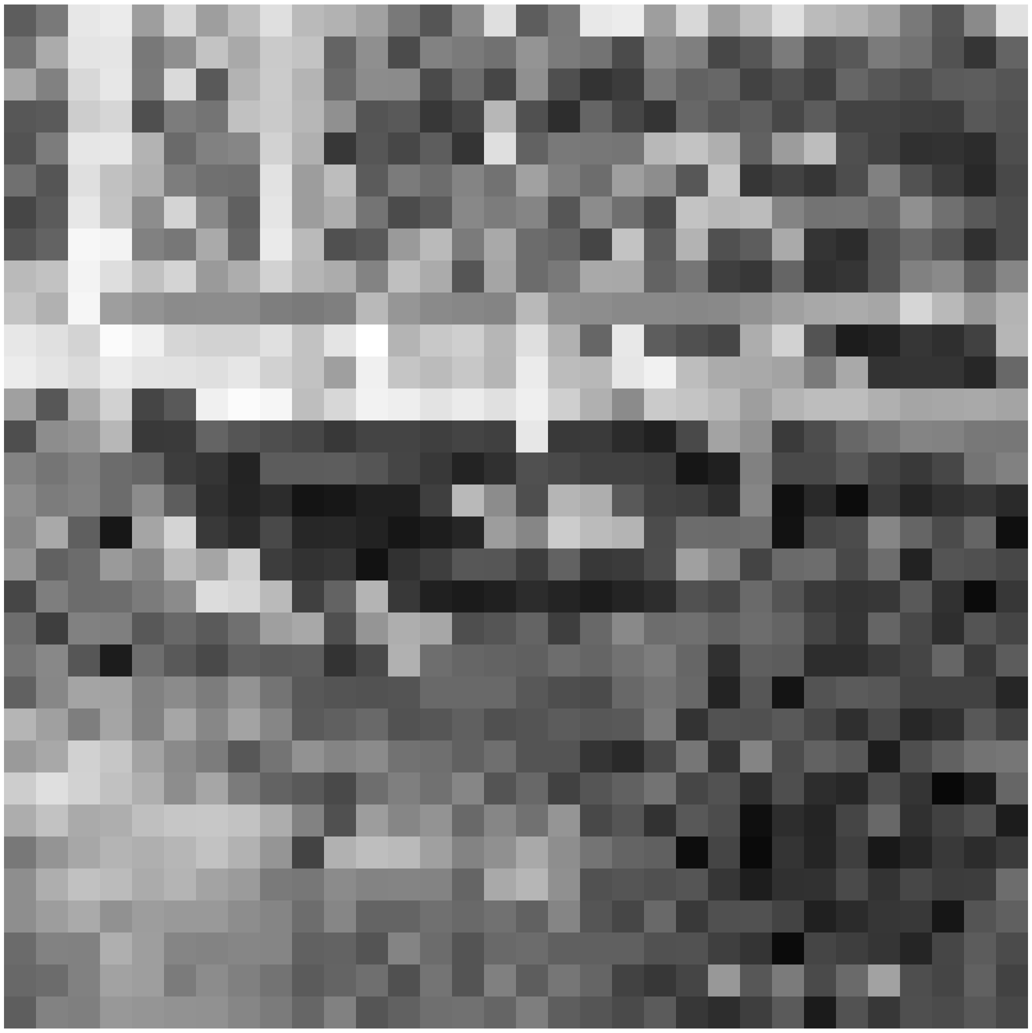
Figure 4. Results after changing the spatial resolution of the original “walkbridge.tif” image from 512x512 to 32x32, then saved as a 512x512 image.

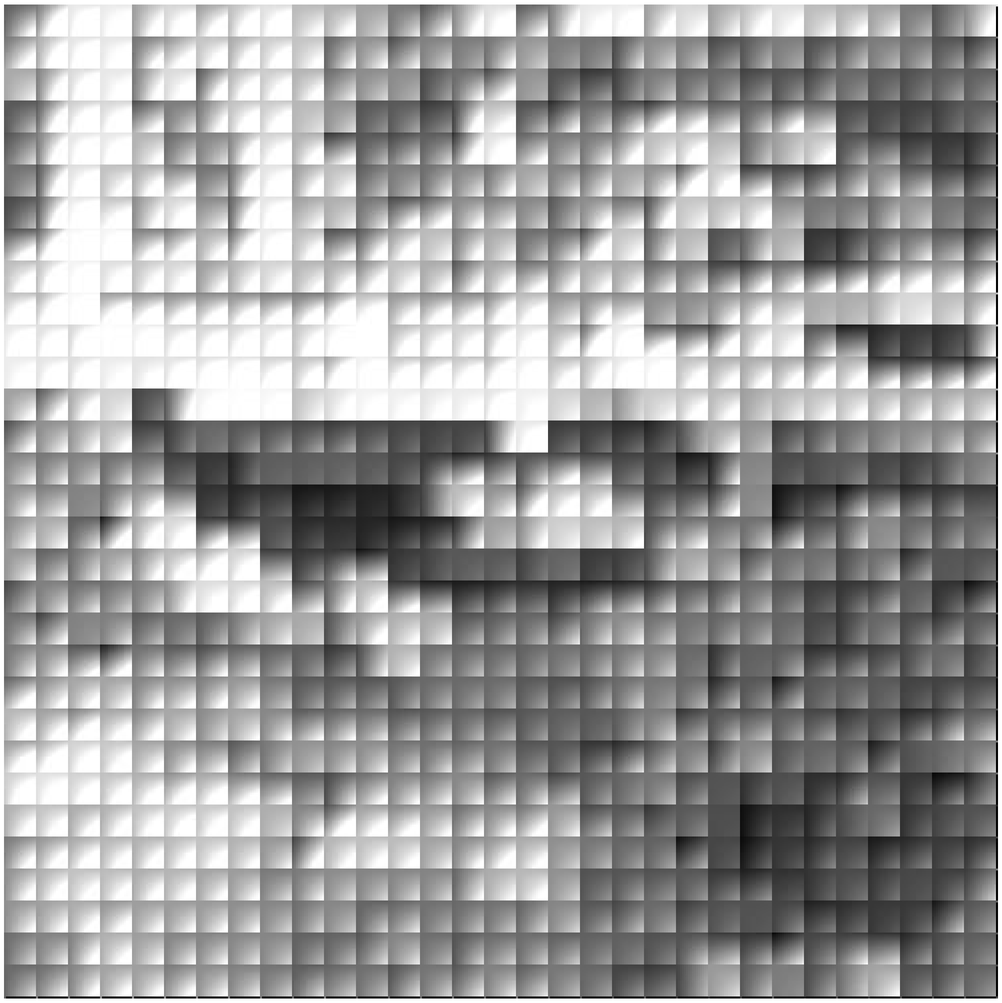

Figure 5. Results after creating an interpolated 512x512 image from the 32x32 image from the previous part using bilinear interpolation.

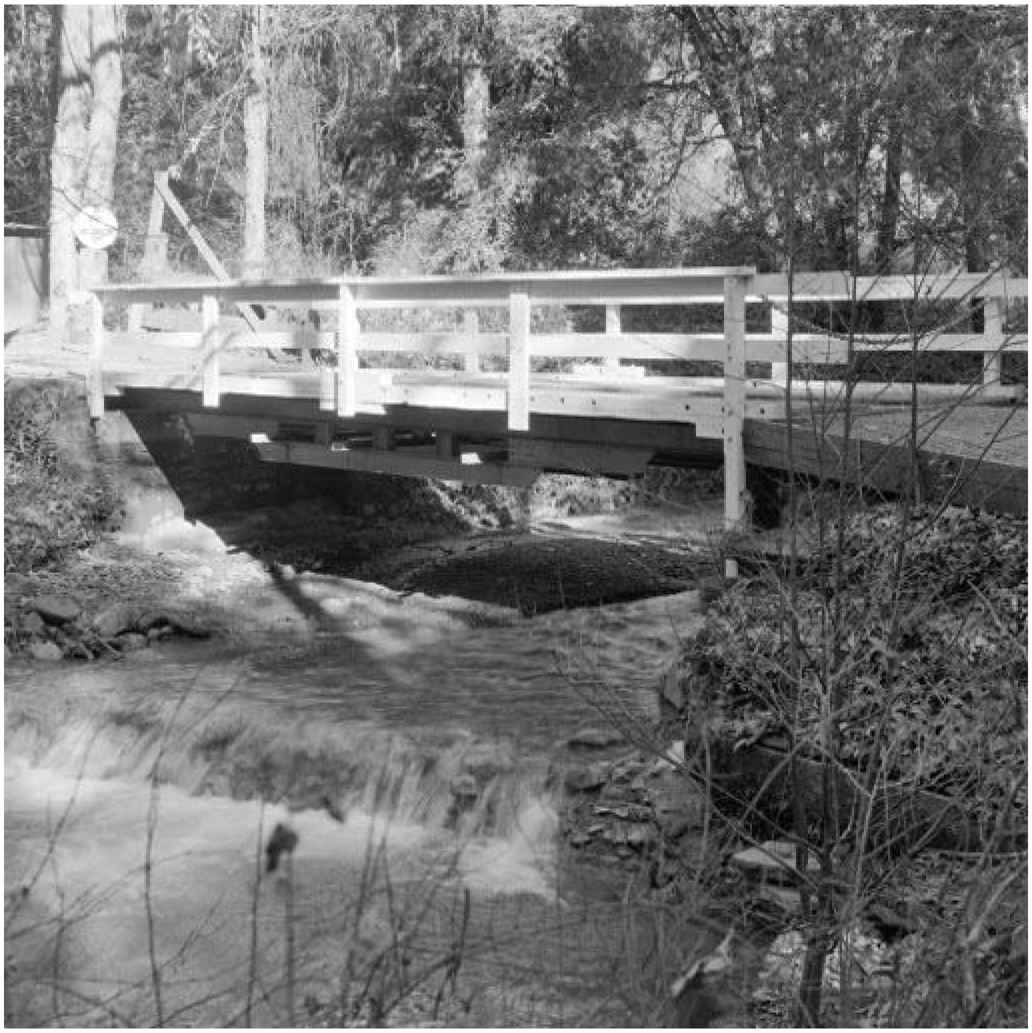

Figure 6. Results after reducing the gray-level quantization of the original “walkbridge.tif” image from 8 bits/pixel to 7 bits/pixel.

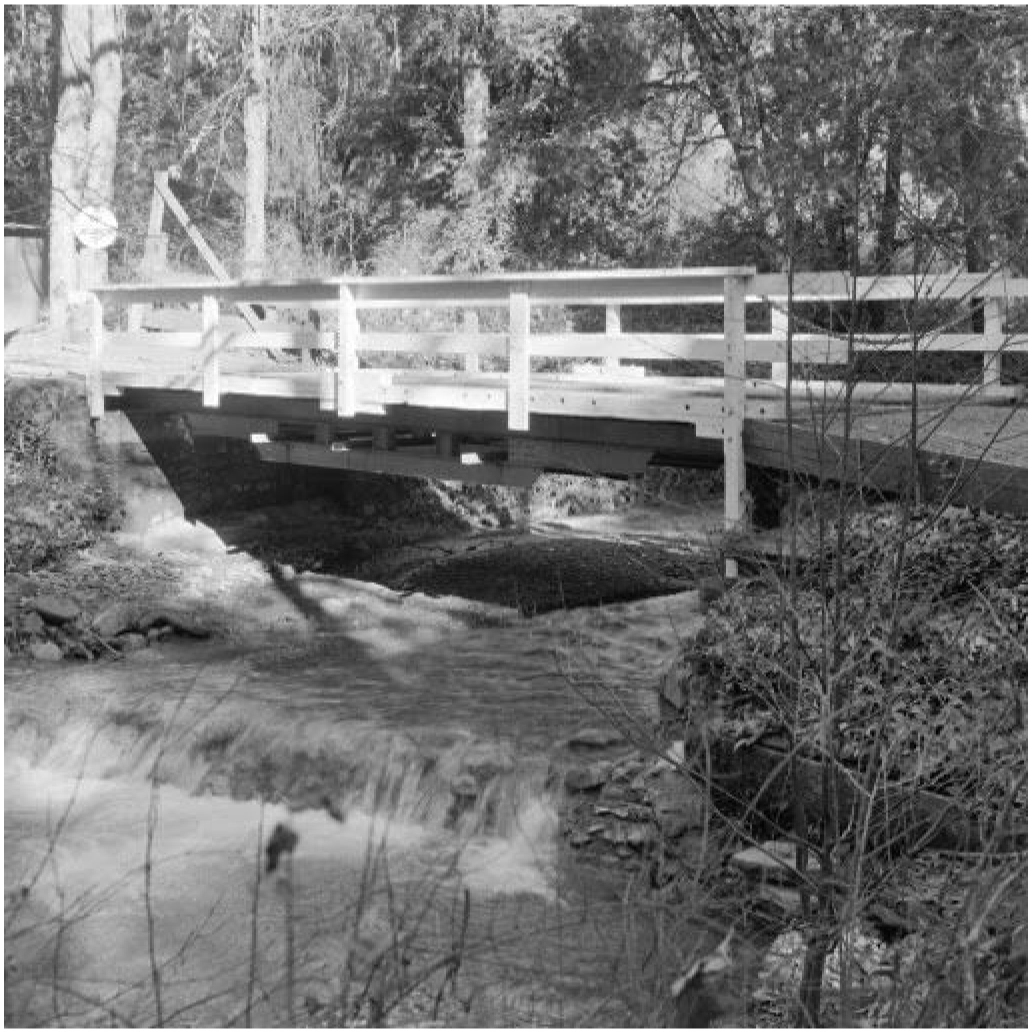

Figure 7. Results after reducing the gray-level quantization of the original “walkbridge.tif” image from 8 bits/pixel to 6 bits/pixel.

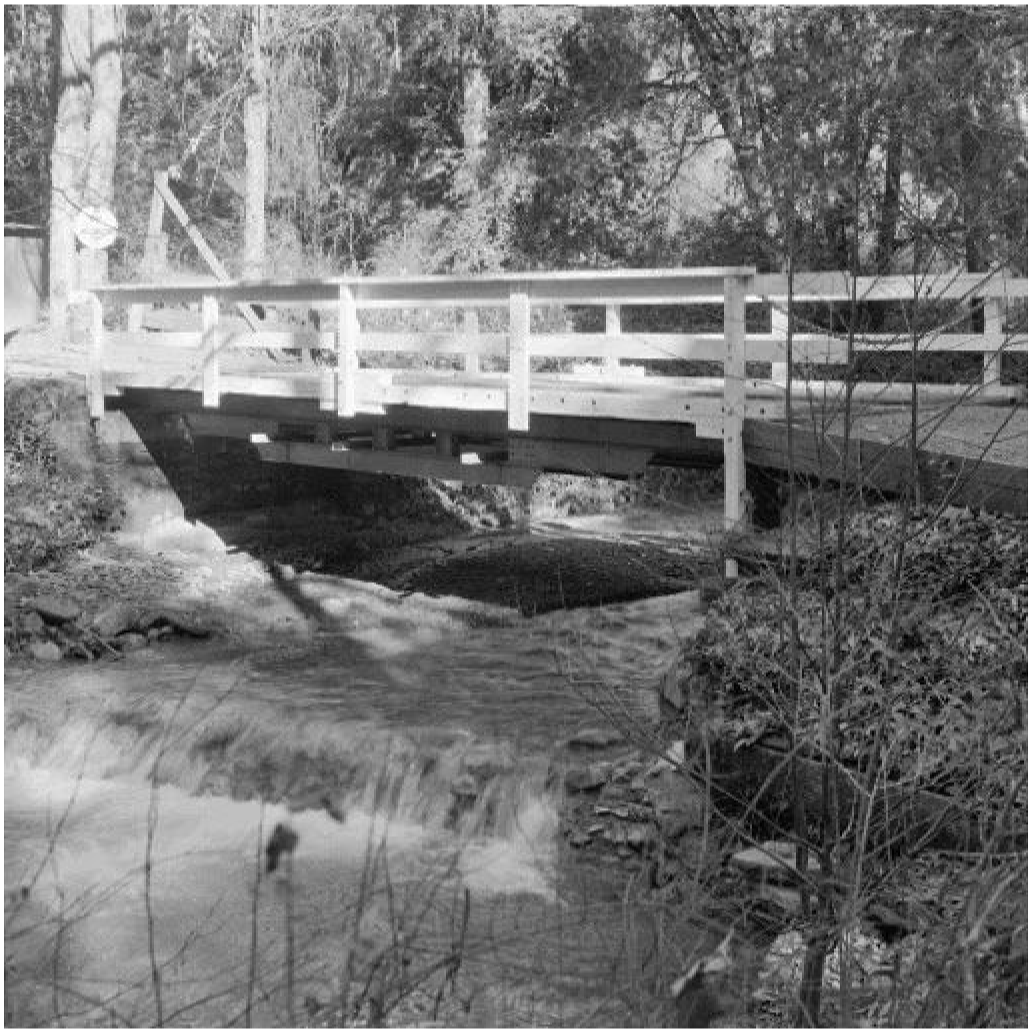

Figure 8. Results after reducing the gray-level quantization of the original “walkbridge.tif” image from 8 bits/pixel to 5 bits/pixel.

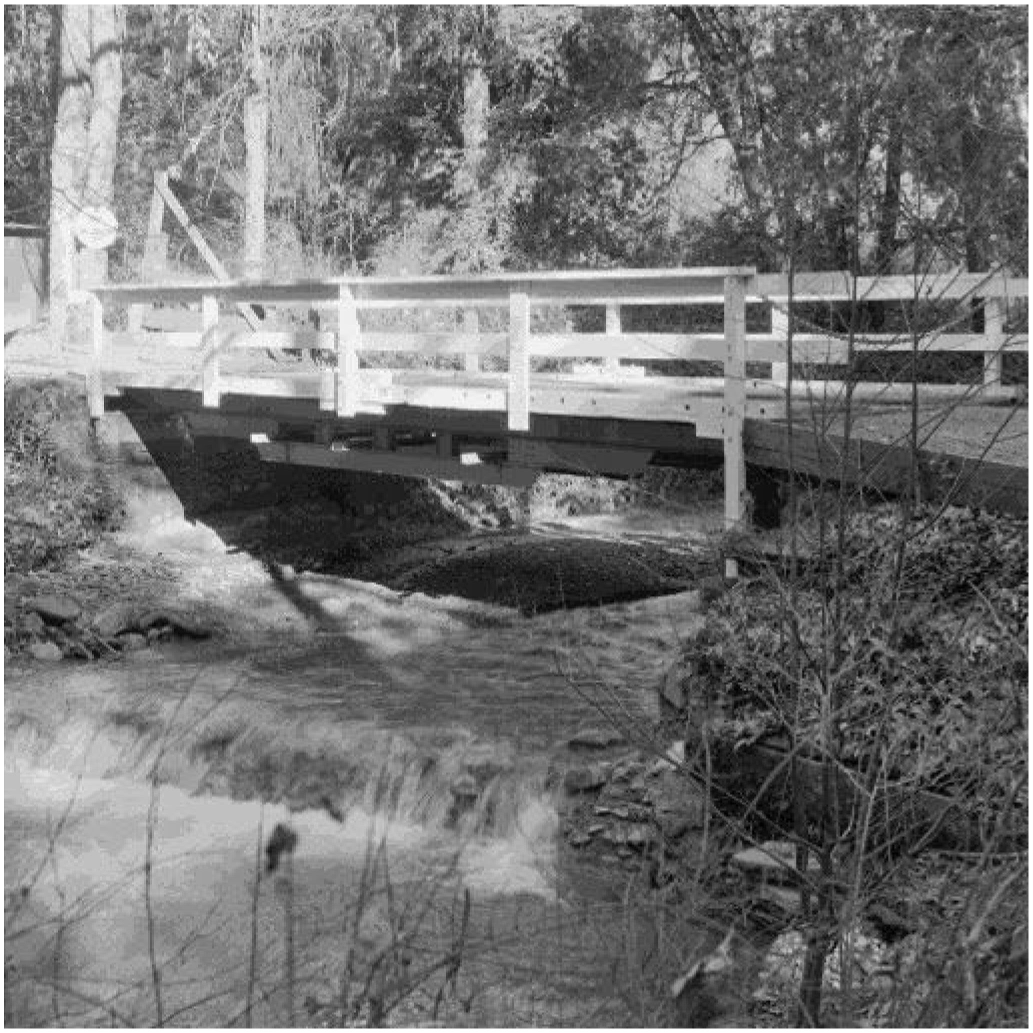

Figure 9. Results after reducing the gray-level quantization of the original “walkbridge.tif” image from 8 bits/pixel to 4 bits/pixel.

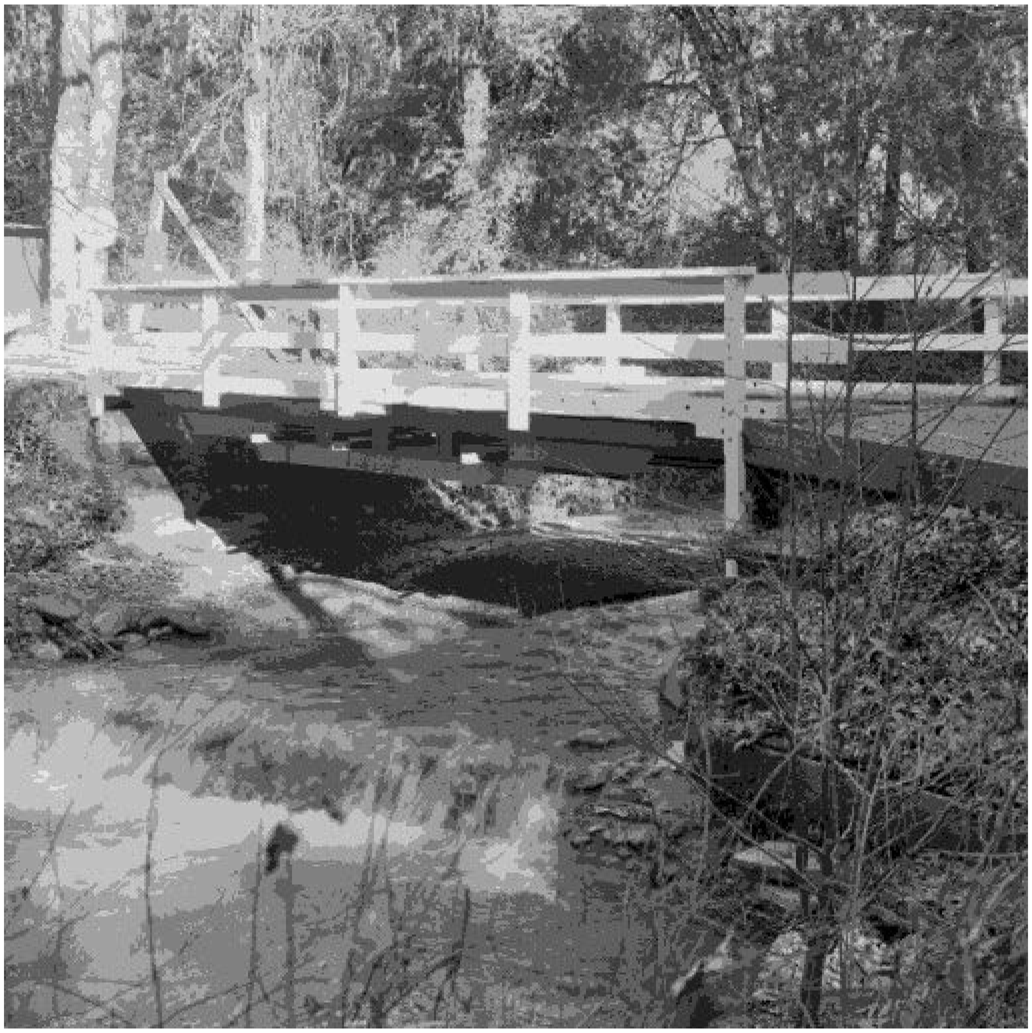

Figure 10. Results after reducing the gray-level quantization of the original “walkbridge.tif” image from 8 bits/pixel to 3 bits/pixel.

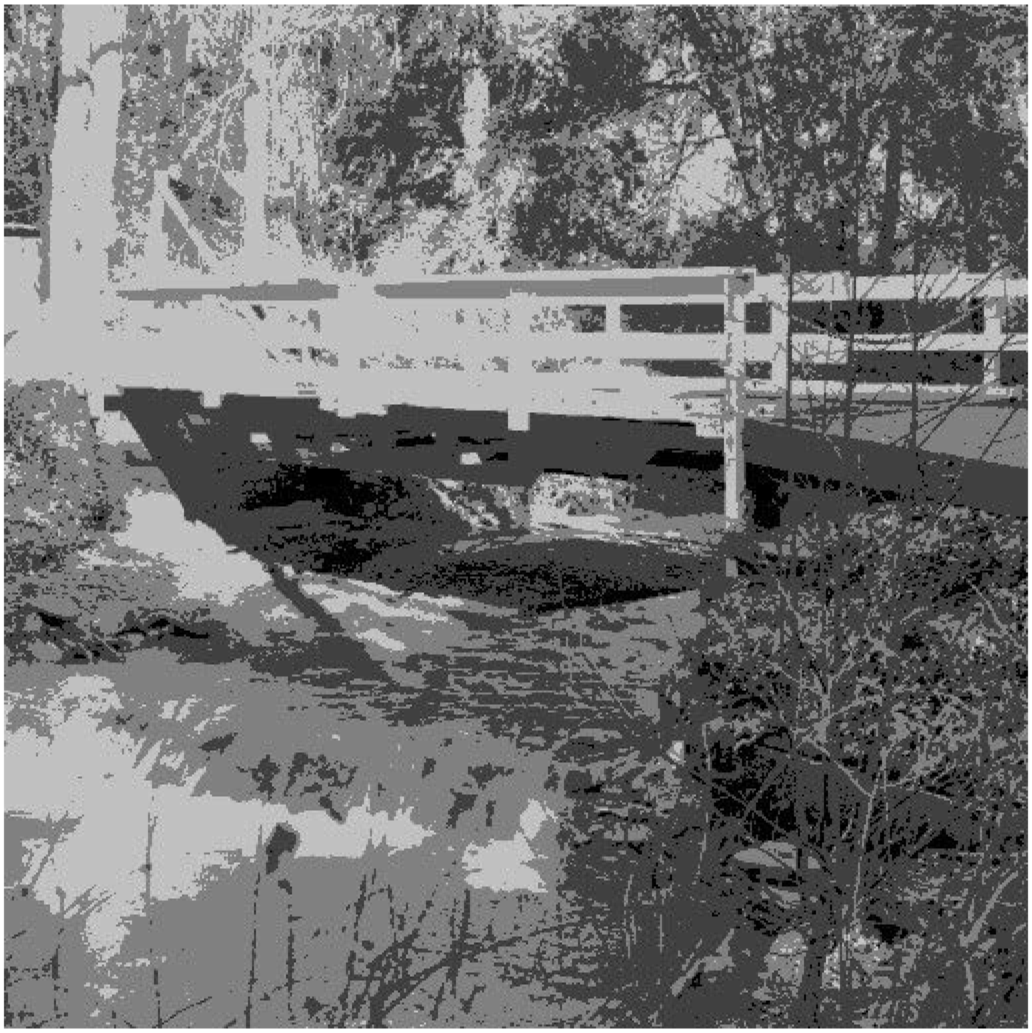

Figure 11. Results after reducing the gray-level quantization of the original “walkbridge.tif” image from 8 bits/pixel to 2 bits/pixel.

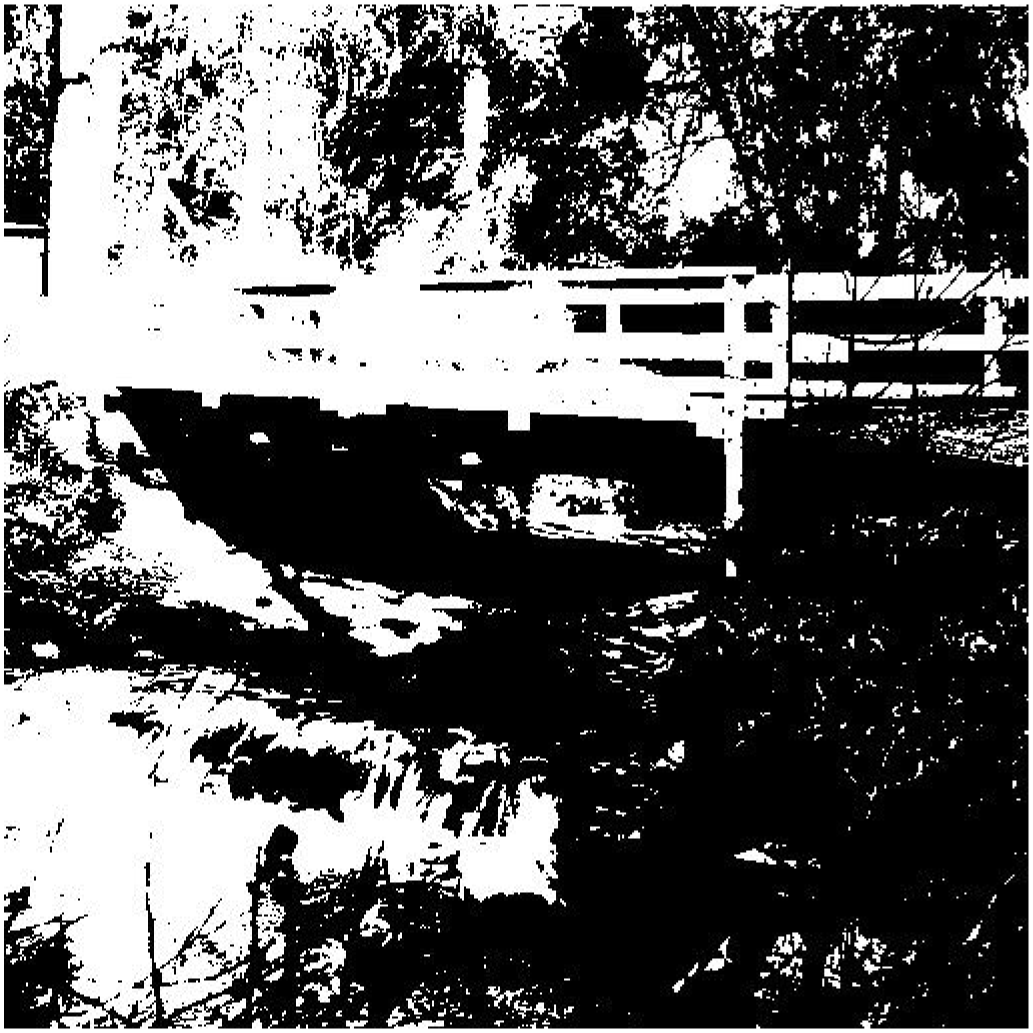

Figure 12. Results after reducing the gray-level quantization of the original “walkbridge.tif” image from 8 bits/pixel to 1 bits/pixel.

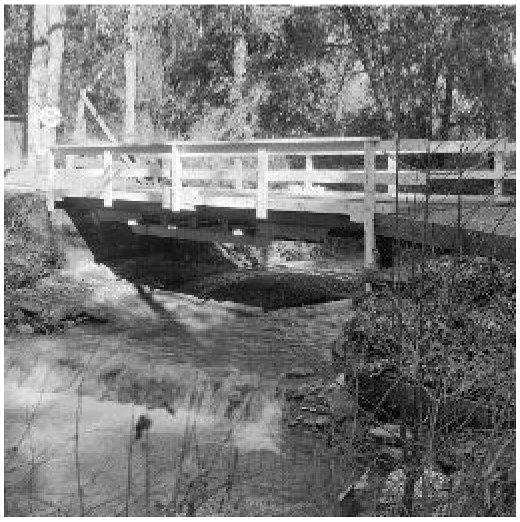

Figure 15. Results after changing the spatial resolution to 256x256 pixels and gray-scale resolution to 6 bits/pixel.

The results were as we expected. Decreasing the spatial resolution of the 512x512 image to 256x256, 128x128, and 32x32 and then saving it as a 512x512 image again showed a clear decrease in clarity. Reducing the gray-scale resolution from 8 bits/pixel did not show a significant decrease in clarity and quality of the image until we reduced it to 3 bits/pixel, 2 bits/pixel and 1 bit/pixel.

D. Conclusions

The project addresses peculiar questions concerning upscaling, downscaling and image gray-levels. The key points to take out are:

- Bilinear interpolation is a better alternative to nearest-neighbor interpolation. The latter is a rudimentary method that produces bad quality images. Bilinear interpolation on the other hand, while more complex, preserves a good fraction of the fidelity of the original image and looks better to the eye.
- Manipulating the gray levels of pixels in an image can cause a drastic change in how an image looks. Reducing gray levels to 7, 6, 5 or even 4 bits does not impact the image greatly. When reduce the levels to 3,2 and 1 however, the image changes become apparent.

MATLAB is an excellent tool for image processing and allows anyone to modify images at pinpoint accuracy. The project was extremely enlightening in regards to how much can be done on images with very little code.