UIN: 01183457

PROJECT 1

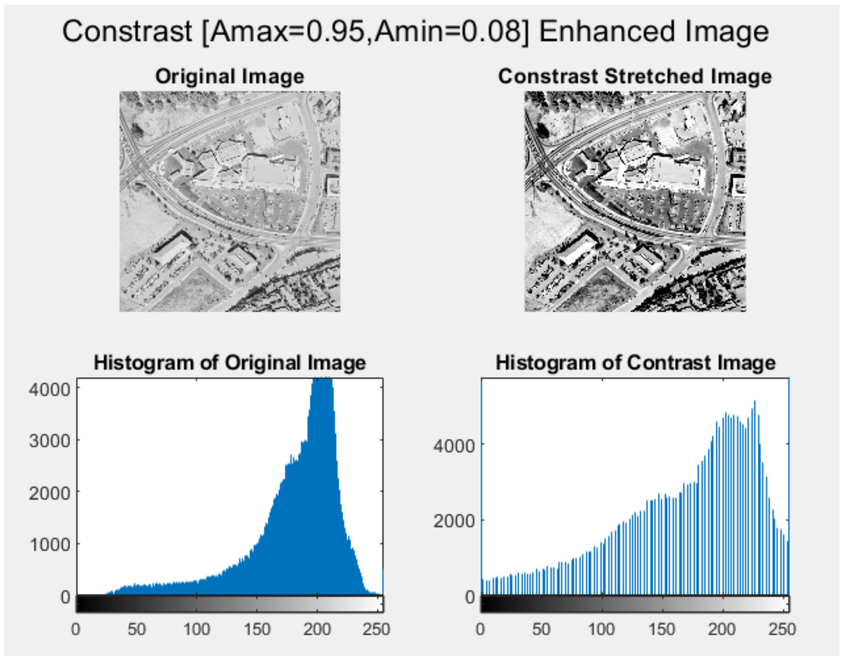
Task 1:

Write a program to contrast adjust the same image. The contrast adjustment operation involves rescaling the input image pixels with values in the range [Amin, Amax] at different pre-defined ranges. Amin and Amax are the minimum and maximum gray scale in your input image. Compare and contrast at least two different methods to change contrast for this image.

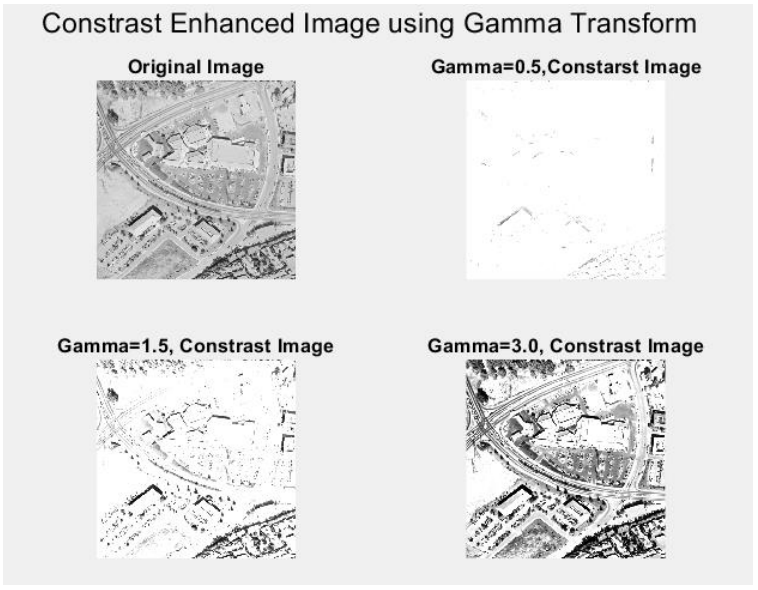
MATLAB Code:

```
I=imread('5.2.09.tiff'); %Read in image
Ics=imadjust(I,stretchlim(I, [0.08 0.95]),[]); %Stretch contrast using method
1
figure;
%sgtitle('Constrast [Amax=0.95,Amin=0.05] Enhanced Image')
subplot(2,2,1), imshow(I); title ('Original Image')%Display image
subplot(2,2,2), imshow(Ics);title('Constrast Stretched Image')% Contrast
Enhanced Image
subplot(2,2,3), imhist(I); %Display input histogram
subplot(2,2,4), imhist(Ics); %Display output histogram

sgtitle('Constrast [Amax=0.95,Amin=0.08] Enhanced Image')


%%%%%%     Appling Gamma Transform  %%%%%%%%%

figure;
subplot(2,2,1), imshow(I); title('Original Image') %Display image
Id= im2double(I);
Output1=2*(Id.^0.5);
Output2=2*(Id.^1.5);
Output3=2*(Id.^3.0);
subplot(2,2,2), imshow(Output1);  title('Gamma=0.5,Constarst Image')
%Display result images
subplot(2,2,3), imshow(Output2);  title('Gamma=1.5, Constrast Image')
subplot(2,2,4), imshow(Output3);  title('Gamma=3.0, Constrast Image')

sgtitle(' Constrast Enhanced Image using Gamma Transform')
```

Output:



Here stretchlim() function is used to identify the c and d pixel values at the 5th and 95th percentile points of the (normalized) histogram distribution of the image. The imadjust() function is then used to map the this range to the (default) maximum quantization range of the image.


Constrast Enhanced Image using Gamma Transform
Original Image
Gamma=0.5,Constarst Image
Gamma=1.5, Constrast Image
Gamma=3.0, Constrast Image

Here, power law(gamma) transform is used where the image each pixel value is raised to fixed value of gamma (0.5, 1.5, 3.0) and the output is observed for each gamma value respectively.

Task 2:

Write a program capable of zooming and shrinking an image with and without bilinear interpolation methods using four nearest neighbors of a point (see the section in your text). The input to your function is the desired size of the resulting image in the horizontal and vertical direction. Use your program to shrink this image from 512x512 to 16 x 16 pixels and then zoom the image back to 512 x 512. You must take care of the boundary image pixels appropriately (this may be done in different ways).

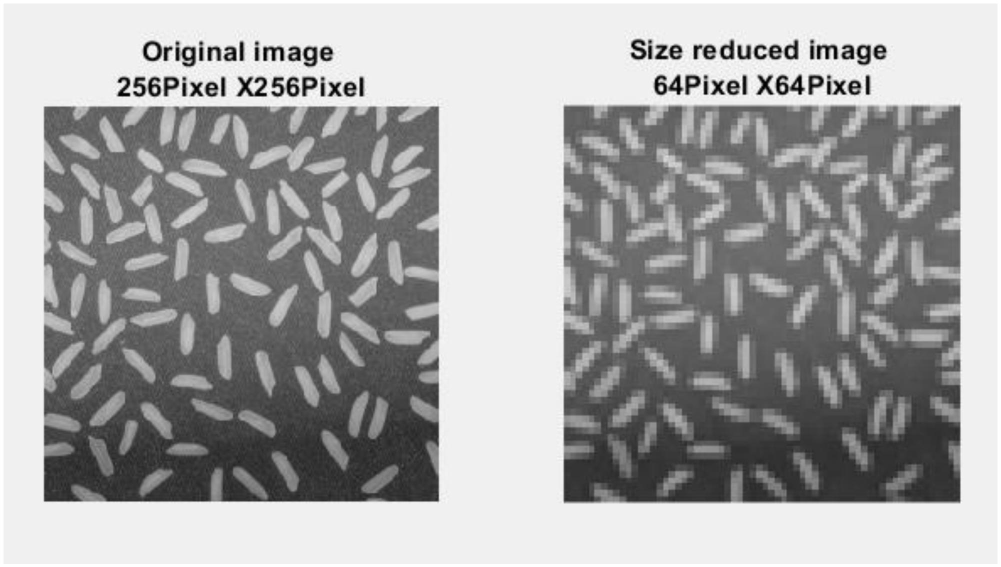
MATLAB Code:

Method 1:

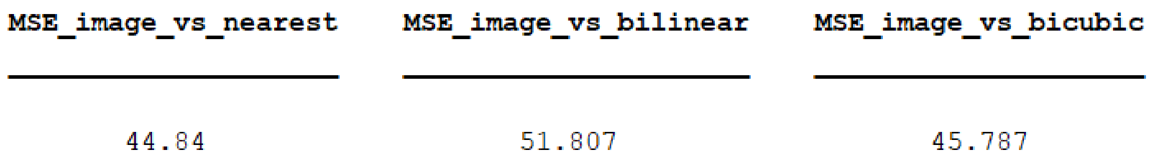
```
imdata=imread('rice.png');
scale=1/4;
[W,H,D]=size(imdata);
w=W*scale;h=H*scale;
shrink_imdata=imresize(imdata,[w,h]);
subplot (1,2,1);
imshow(imdata);
str=[{'Original image ' [num2str(W) , 'Pixel X' num2str(H) 'Pixel']}];
title(str);
subplot (1,2,2);
imshow(shrink_imdata);
str=[{'Size reduced image ' [num2str(w) 'Pixel X' num2str(h) 'Pixel']}];
title(str);
figure;
zoom_nearest = imresize(shrink_imdata, [W,H], 'nearest');
[W1,H1]=size(zoom_nearest);
subplot (2,2,1);
imshow(zoom_nearest);
%str=[{'Zoomed image with nearest neighbour interpolation'}];
title('Zoomed image with nearest neighbour interpolation');

zoom_bilinear = imresize(shrink_imdata,[W,H],'bilinear');
[W2,H2]=size(zoom_bilinear);
subplot (2,2,2);
imshow(zoom_bilinear);
%str=[{'Zoomed image with bilinear interpolation'}];
title('Zoomed image with bilinear interpolation');


zoom_bicubic = imresize(shrink_imdata,[W,H],'bicubic');
[W3,H3]=size(zoom_bicubic);
subplot (2,2,3);
imshow(zoom_bilinear);
%str=[{'Zoomed image with bicubic interpolation'}];
title('Zoomed image with bicubic interpolation');
```

```
switch D
   case 3
MSE_image_vs_nearest=(sum(sum((imdata(:,:,1)-zoom_nearest(:,:,1)).^2+(imdata(:,:,2)-
zoom_nearest(:,:,2)).^2+(imdata(:,:,3)-zoom_nearest(:,:,3)).^2))./(W*H);
MSE_image_vs_bilinear=sum(sum((imdata(:,:,1)-zoom_bilinear(:,:,1)).^2+(imdata(:,:,2)-
zoom_bilinear(:,:,2)).^2+(imdata(:,:,3)-zoom_bilinear(:,:,3)).^2))./(W*H);
MSE_image_vs_bicubic=sum(sum((imdata(:,:,1)-zoom_bicubic(:,:,1)).^2+(imdata(:,:,3)-
zoom_bicubic(:,:,3)).^2+(imdata(:,:,3)-zoom_bicubic(:,:,3)).^2))./(W*H);
   case 1
MSE_image_vs_nearest=sum(sum((imdata(:,:,1)-zoom_nearest(:,:,1)).^2))./(W*H);
MSE_image_vs_bilinear=sum(sum((imdata(:,:,1)-zoom_bilinear(:,:,1)).^2))./(W*H);
MSE_image_vs_bicubic=sum(sum((imdata(:,:,1)-zoom_bicubic(:,:,1)).^2))./(W*H);
end
T = table(MSE_image_vs_nearest,MSE_image_vs_bilinear,MSE_image_vs_bicubic)
```

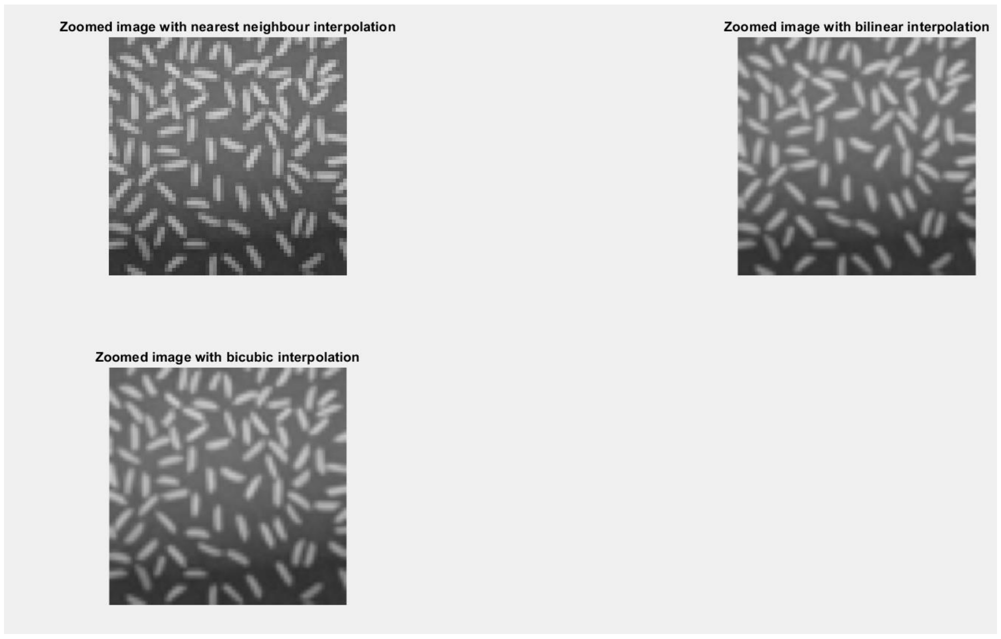
Output:



Here, a scale value of 1/4 is used to scale down the image to 64x64 using imresize function of MATLAB. Afterwards, this reduced image is used to zoom it back to original size of 256x256 using nearest, bilinear and bicubic interpolation method. Then Mean square is calculated for each interpolation method.



```
MSE_image_vs_nearest    MSE_image_vs_bilinear    MSE_image_vs_bicubic
____________________    _____________________    ____________________

       44.84                   51.807                   45.787
```


Zoomed image with nearest neighbour interpolation
Zoomed image with bilinear interpolation
Zoomed image with bicubic interpolation

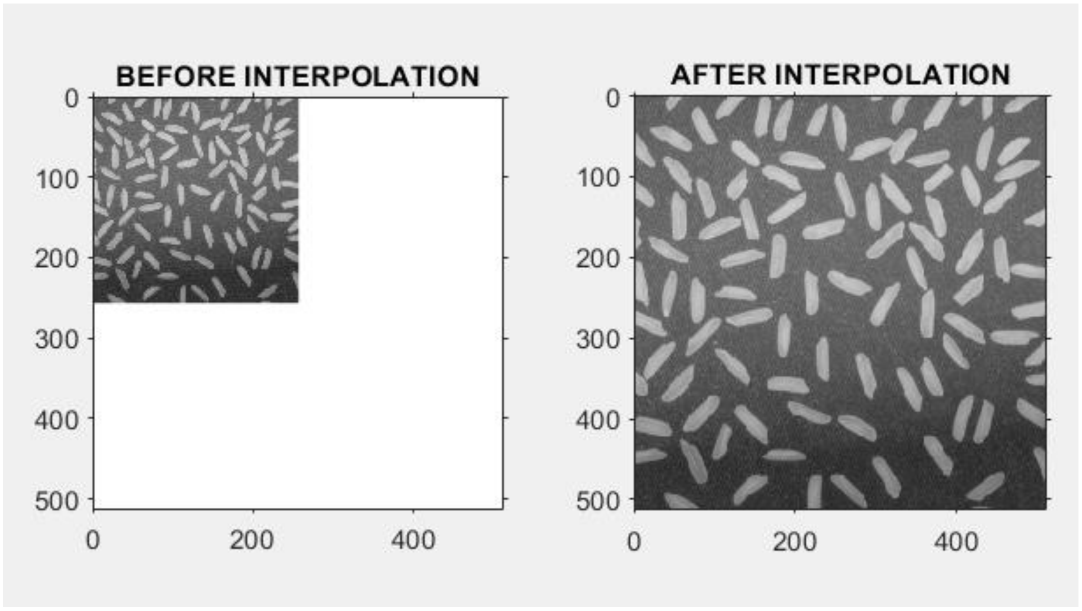
From mean square error from the aforementioned table, it is shown nearest and bicubic is better than bilinear method and nearest method mean square is the least.

Method 2:

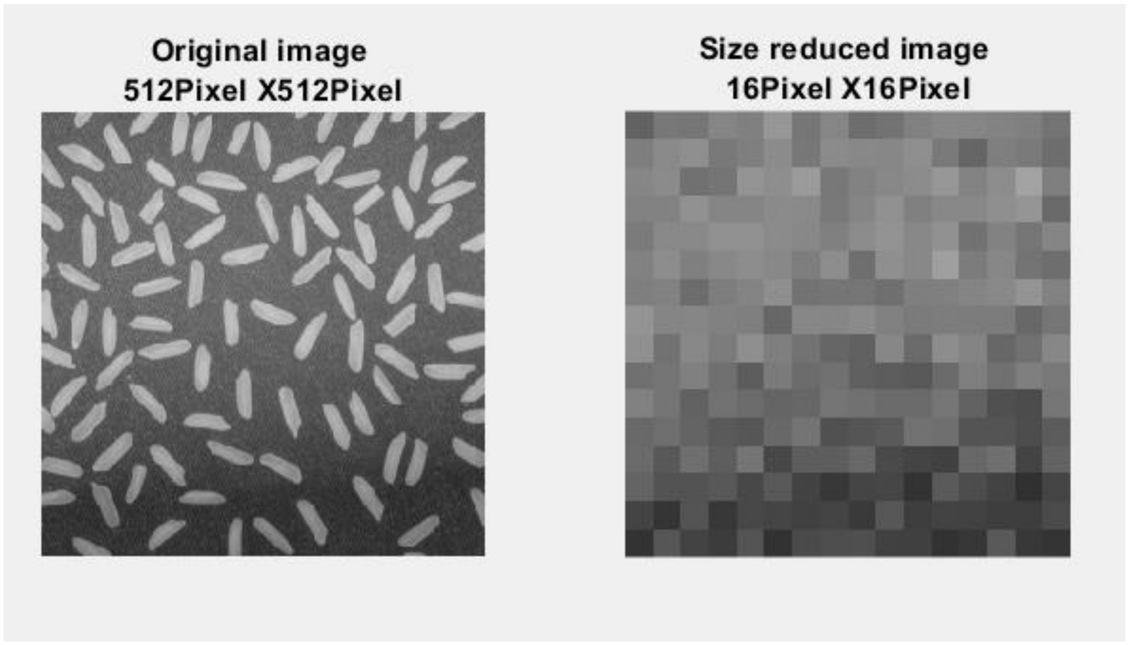
```
%%%%Image Interpolation without using imresize function%%%%%%
A=imread('rice.png');
% DEFINE THE RESAMPLE SIZE
Col = 512;
Row = 512;
%FIND THE RATIO OF THE NEW SIZE BY OLD SIZE
rtR = Row/size(A,1);
rtC = Col/size(A,2);

%OBTAIN THE INTERPOLATED POSITIONS
IR = ceil([1:(size(A,1)*rtR)]./(rtR));
IC = ceil([1:(size(A,2)*rtC)]./(rtC));

%ROW_WISE INTERPOLATION
B = A(:,IR);

%COLUMN-WISE INTERPOLATION
B = B(IC,:);


figure,subplot(121),imshow(A);title('BEFORE INTERPOLATION'); axis([0,512,0,512]);axis on;

subplot(122),imshow(B);title('AFTER INTERPOLATION');  axis([0,512,0,512]);axis on;

figure;
```

```
scale=1/32;
[W,H,D]=size(B);
w=W*scale;h=H*scale;
shrink_imdata=imresize(B,[w,h]);
subplot (1,2,1);
imshow(B);
str=[{'Original image ' [num2str(W) , 'Pixel X' num2str(H) 'Pixel']}];
title(str);
subplot (1,2,2);
imshow(shrink_imdata);
str=[{'Size reduced image ' [num2str(w) 'Pixel X' num2str(h) 'Pixel']}];
title(str);

figure;
zoom_nearest = imresize(shrink_imdata, [W,H], 'nearest');
[W1,H1]=size(zoom_nearest);
subplot (2,2,1);
imshow(zoom_nearest);
%str=[{'Zoomed image with nearest neighbour interpolation'}];
title('Zoomed image with nearest neighbour interpolation');

zoom_bilinear = imresize(shrink_imdata,[W,H],'bilinear');
[W2,H2]=size(zoom_bilinear);
subplot (2,2,2);
imshow(zoom_bilinear);
%str=[{'Zoomed image with bilinear interpolation'}];
title('Zoomed image with bilinear interpolation');


zoom_bicubic = imresize(shrink_imdata,[W,H],'bicubic');
[W3,H3]=size(zoom_bicubic);
subplot (2,2,3);
imshow(zoom_bilinear);
%str=[{'Zoomed image with bicubic interpolation'}];
title('Zoomed image with bicubic interpolation');

switch D
   case 3
MSE_image_vs_nearest=(sum(sum((B(:,:,1)-zoom_nearest(:,:,1)).^2+(B(:,:,2)-zoom_nearest(:,:,2)).^2+(B(:,:,3)-
zoom_nearest(:,:,3)).^2)))./(W*H);
MSE_image_vs_bilinear=sum(sum((B(:,:,1)-zoom_bilinear(:,:,1)).^2+(B(:,:,2)-zoom_bilinear(:,:,2)).^2+(B(:,:,3)-
zoom_bilinear(:,:,3)).^2))./(W*H);
MSE_image_vs_bicubic=sum(sum((B(:,:,1)-zoom_bicubic(:,:,1)).^2+(B(:,:,3)-zoom_bicubic(:,:,3)).^2+(B(:,:,3)-
zoom_bicubic(:,:,3)).^2))./(W*H);
   case 1
MSE_image_vs_nearest=sum(sum((B(:,:,1)-zoom_nearest(:,:,1)).^2))./(W*H);
MSE_image_vs_bilinear=sum(sum((B(:,:,1)-zoom_bilinear(:,:,1)).^2))./(W*H);
MSE_image_vs_bicubic=sum(sum((B(:,:,1)-zoom_bicubic(:,:,1)).^2))./(W*H);
end
T = table(MSE_image_vs_nearest,MSE_image_vs_bilinear,MSE_image_vs_bicubic)
```

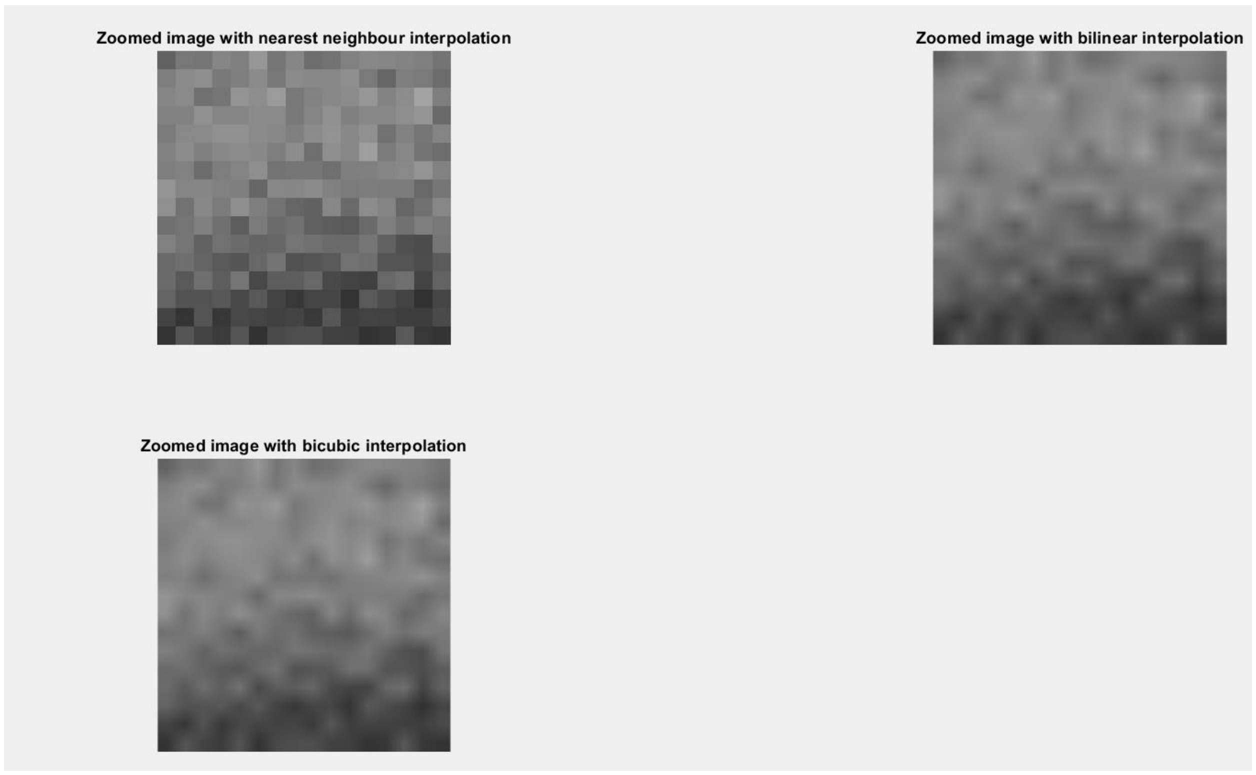
Output:



The above output depicts the image interpolation method without using imresize function in MATLAB. The image is interpolated from size 256X256 to size 512X512 size.



The interpolated image is then reduced to size 16X 16 size, and it is observed that the image quality degraded and it is also observed when the reduced size image is zoomed back and mean square error is calculated for nearest, bilinear and bicubic method. The resulting image and calculation are the following:


Zoomed image with nearest neighbour interpolation
Zoomed image with bilinear interpolation
Zoomed image with bicubic interpolation

```
MSE_image_vs_nearest    MSE_image_vs_bilinear    MSE_image_vs_bicubic
____________________    _____________________    ____________________

       72.602                   70.59                   71.328
```

The difference between Method 1 and Method 2 is the following:

1. Method 1 used imresize function, Method 2 did manual calculation for image resampling.
2. Method 1 the reduced image size is 64X64 and mean square error is less than Method 2 mean square error in all three-interpolation method after zooming the image.
3. Moreover, one of the observations is with reduced image (16X16), the mean square error for bilinear method is the least whereas with image size 64X64 the bilinear method has the highest mean square error.

Task 3:

Download the Clown image from image processing course web page. Select a subimage of size 32x32 from this image. Convert this image to binary by appropriate thresholding. Now, apply chain code technique to extract edges of this binary image. Experiment with different methods and comment on the code performance (i.e., time taken on a type of processor).

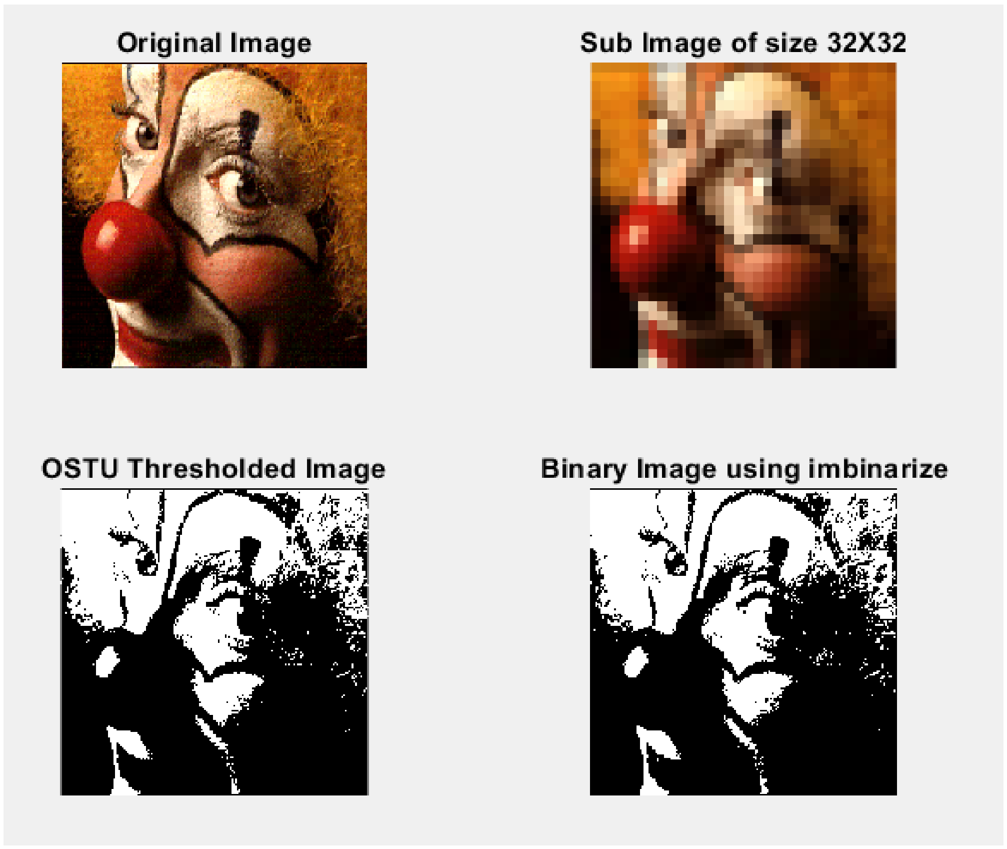
MATLAB Code:

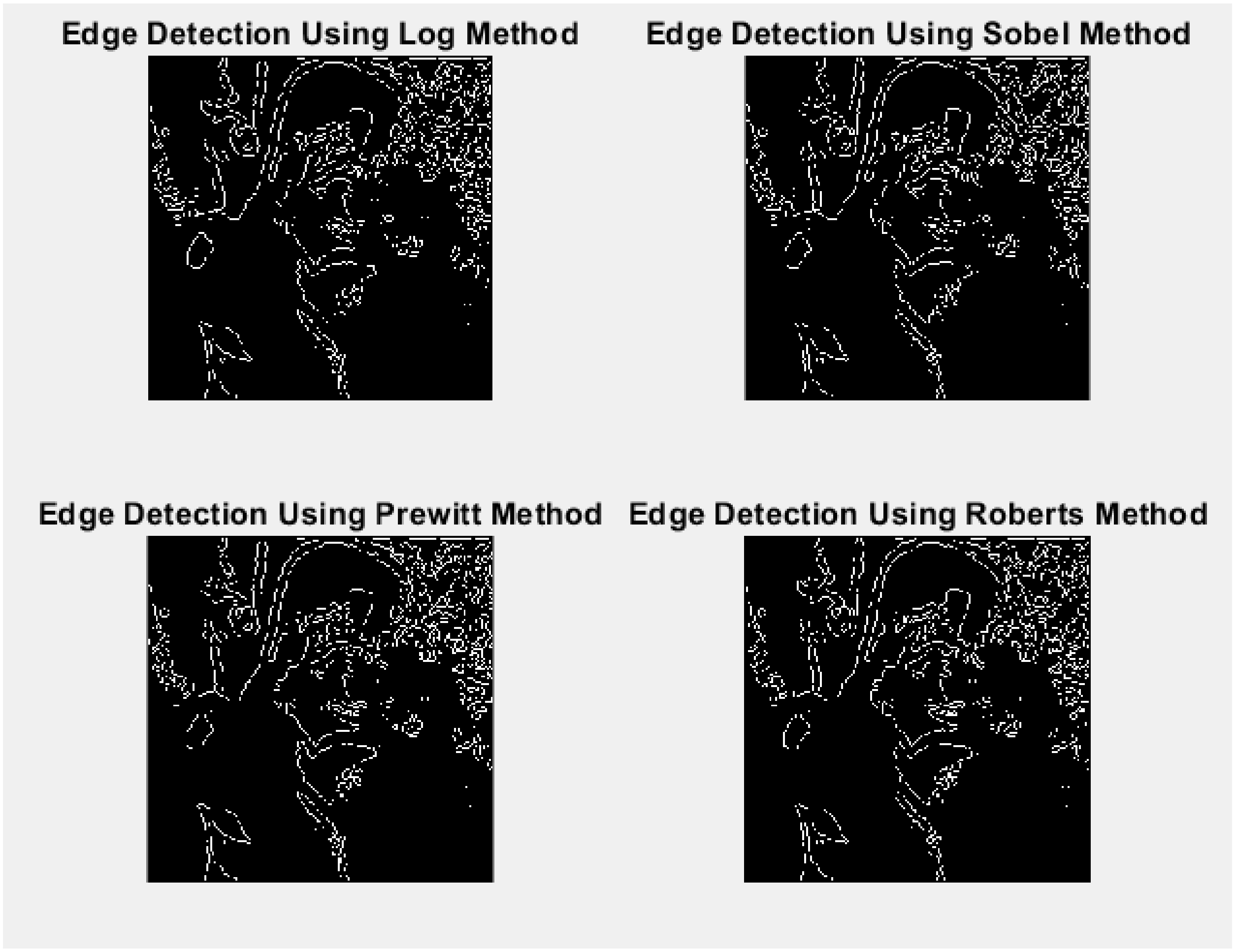
Method 1: Using 'edge' function of MATLAB to find the edge of the image

```
% Load clown image data.
s = load('clown.mat')
% Display it as an indexed, pseudocolored image.
%imshow(s.X, s.map);
% Create and display the RGB image.
rgbImage = ind2rgb(s.X, s.map);
rgbImage=imresize(rgbImage,[256,256]); %%% Resizing the original Image to 256X256
figure;
subplot(2,2,1),imshow(rgbImage);
title('Original Image')

%%%Selecting Sub-Image from the original Image%%%%%

fun = @(block_struct) imresize(block_struct.data,0.125);%% To take a block of 32X32 , multiply the image with
(1/8=0.125)
I2 = blockproc(rgbImage,[32 32],fun); %% reference :https://www.mathworks.com/help/images/ref/blockproc.html

subplot(2,2,2), imshow(I2);
title('Sub Image of size 32X32')

%%%%Image Thresholding %%%%%%%%%%%%%

I= rgb2gray(rgbImage)
level=graythresh(I); %Get OTSU threshold
It=im2bw(I, level); %Threshold image
subplot(2,2,3),imshow(It);
title('OSTU Thresholded Image')

BW = imbinarize(I);
subplot(2,2,4),imshow(BW);
title('Binary Image using imbinarize')
%figure
%imshowpair(I,BW,'montage')

BW1 = edge(BW,'log'); % reference:https://www.mathworks.com/help/images/ref/edge.html
BW2 = edge(BW,'Sobel');
BW3= edge(BW,'Prewitt');
bw4= edge(BW,'Roberts');
figure;
subplot (2,2,1),imshow(BW1); title('Edge Detection Using Log Method')
subplot (2,2,2),imshow(BW1); title('Edge Detection Using Sobel Method')
subplot (2,2,3),imshow(BW1); title('Edge Detection Using Prewitt Method')
subplot (2,2,4),imshow(BW1); title('Edge Detection Using Roberts Method')
```

```
fcn_log = @() edge(BW,'log');
fcn_sobel = @() edge(BW,'sobel');
fcn_prewitt = @() edge(BW,'Prewitt');
fcn_Roberts = @() edge(BW,'Roberts');
t_log=timeit(fcn_log)
t_sobel=timeit(fcn_sobel)
t_prewitt=timeit(fcn_prewitt)
t_roberts=timeit(fcn_Roberts)
fprintf('Time taken by log method:%0.5f sec\n',t_log)
fprintf('Time taken by Sobel method:%0.5f sec\n',t_sobel)
fprintf('Time taken by Prewitt method:%0.5f sec\n',t_prewitt)
fprintf('Time taken by Roberts method:%0.5f sec\n',t_roberts)
```

Output:




Edge Detection Using Log Method
Edge Detection Using Sobel Method
Edge Detection Using Prewitt Method
Edge Detection Using Roberts Method

Time Taken By Different Edge Method in MATLAB:

```
Time taken by log method:0.00485 sec
Time taken by Sobel method:0.00050 sec
Time taken by Prewitt method:0.00048 sec
Time taken by Roberts method:0.00124 sec
```

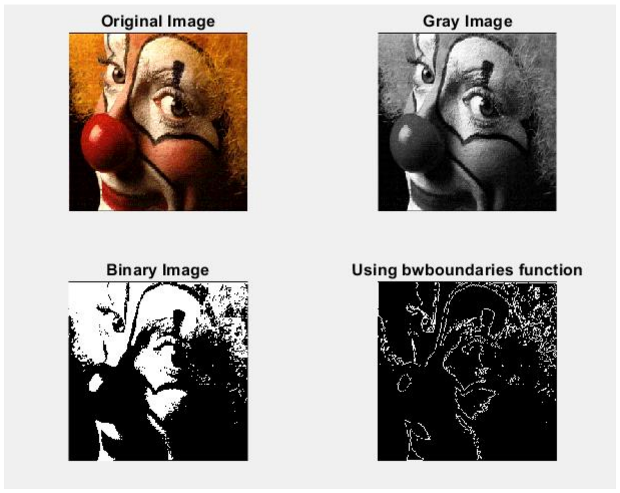
Method 2: Edge Detection of Image using ‘bwboundaries’ function (reference provided within the code)

Code:

```
s = load('clown.mat')
% Display it as an indexed, pseudocolored image.
%imshow(s.X, s.map);
```

```
% Create and display the RGB image.
rgbImage = ind2rgb(s.X, s.map);
rgbImage=imresize(rgbImage,[256,256]); %%% Resizing the original Image to 256X256
figure;
subplot(2,2,1),imshow(rgbImage);
title('Original Image')

Ib= rgb2gray(rgbImage);
subplot(2,2,2),imshow(Ib);
title('Gray Image')

I=imbinarize(Ib);
subplot(2,2,3),imshow(I);
title('Binary Image')

[B,L] = bwboundaries(I, 8);  %reference:https://blogs.mathworks.com/pick/2021/02/09/tracing-the-
boundary-of-a-binary-region-in-an-image/
Contour2 = false(size(L));
for ii = 1:numel(B)
   thisB = B{ii};
   inds = sub2ind(size(L), thisB(:, 1), thisB(:, 2));
   Contour2(inds) = 1;
end

subplot(2,2,4), imshow(Contour2)
title('Using bwboundaries function')

fcn = @() bwboundaries(I, 8);
t_IPT = timeit(fcn);
fprintf('Time Required Using Boundary Function: %0.5f sec\n', t_IPT)
```

Output:



```
Time Required Using Boundary Function: 0.00326 sec
```

Method 3: Using Chain Code to find the chain code (8 bit) for image boundary

Code:

```
% Load clown image data.
s = load('clown.mat')
% Display it as an indexed, pseudocolored image.
%imshow(s.X, s.map);
% Create and display the RGB image.
rgbImage = ind2rgb(s.X, s.map);
%rgbImage=imresize(rgbImage,[256,256]);
subplot (2,2,1),imshow(rgbImage);title('Original Image')
%imwrite(rgbImage,'myclown.png')

I= rgb2gray(rgbImage);
I = imbinarize(I);
subplot (2,2,2),imshow(I);title('Binary Image')

%C = imcontour(I, 1);
C=~bwperim(I);
subplot(2,2,3), imshow(C); title('Image Contour using bwperim function')
%[y,x] = find(C== 0);


[cc] = chaincode(C,true) %reference:https://www.mathworks.com/matlabcentral/fileexchange/29518-
freeman-chain-code

fcn = @()chaincode(C,true);
t_IPT = timeit(fcn);
fprintf('Time Required Using Chain Function: %0.5f sec\n', t_IPT)
T = struct2table(cc,'AsArray',true)
k=cell2table(T.code)
writecell(k.Var1,'C_tab.txt','Delimiter','tab')
type 'C_tab.txt'
```

Output:

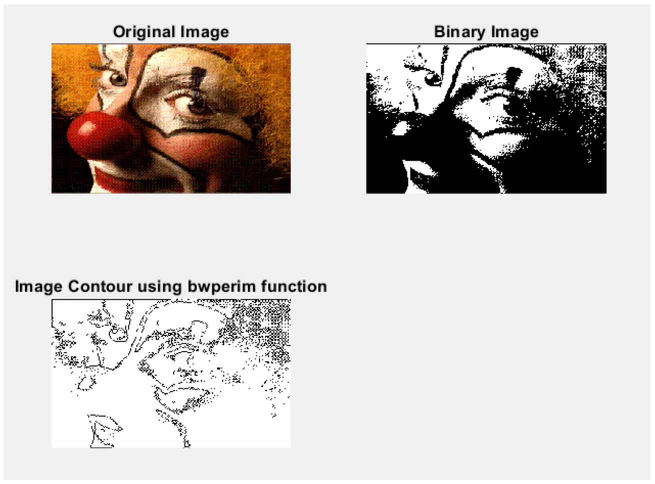
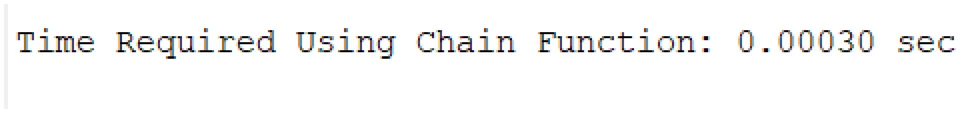


Image Contour found by beperium function is utilized as a input to the chain code and the following change code is saved as text file output.

The chain code:

```
5    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    3
     7    4    0    4    1    5    0    0    0    0    0    4
     0    2    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    4    6    0
     1    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0
```



```
Time Required Using Chain Function: 0.00030 sec
```

Comparing the timing for edge detection, chain function require less time than defined bwboundaries function in Method 2. In Method 1, perwitt method take 0.00048 second compare to sobel, Roberts and log method.