CSC 481

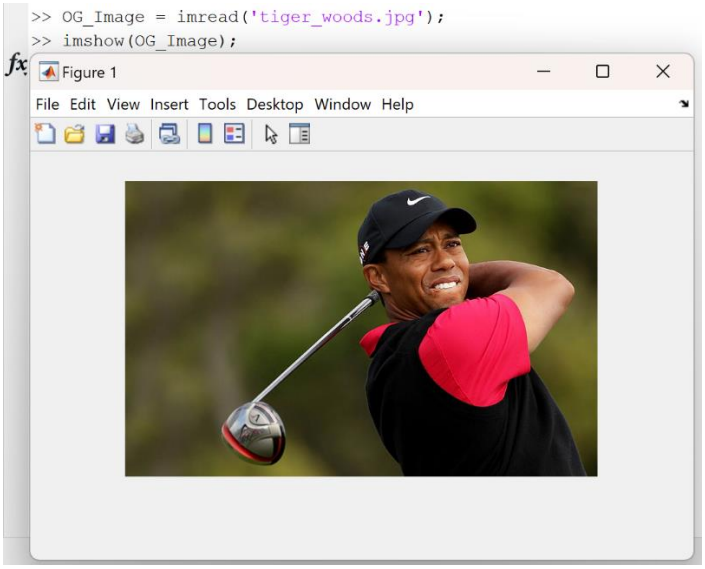
Assignment 1 (30/23 points)

1. **Getting familiar with image manipulation in Matlab (or your favorite language) (*12/10*)**

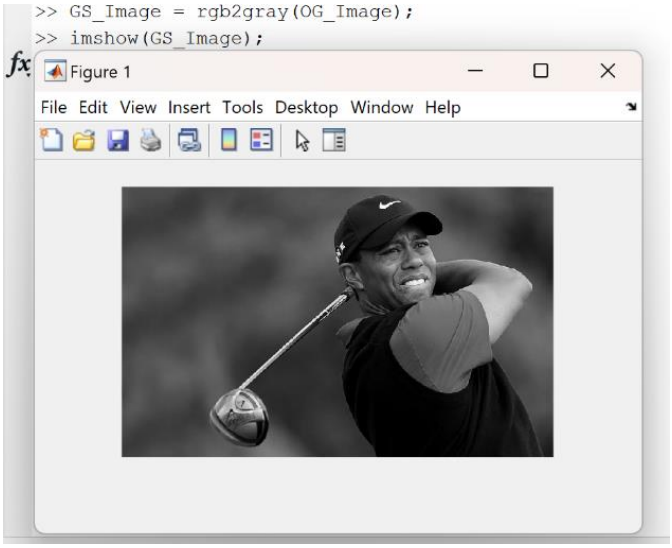
Write a program/function that will:

(a) Read an image, convert it to grayscale if it isn't already, and display the converted image

```
>> OG_Image = imread('tiger_woods.jpg');
>> imshow(OG_Image);
```




```
>> GS_Image = rgb2gray(OG_Image);
>> imshow(GS_Image);
```



(b) Calculate the size (total number of pixels) of the image

```
>> whos
  Name            Size                Bytes  Class    Attributes

  GS_Image      455x728              331240  uint8
  OG_Image      455x728x3            993720  uint8

>> p_size = numel(GS_Image);
>> disp(['size of the image: ' num2str(p_size) ' pixels']);
size of the image: 331240 pixels
fx >>
```

(c) Calculate the maximum pixel value

```
>> max_p_size = max(GS_Image(:));
>> disp(['Max pixel size of the image: ' num2str(max_p_size)]);
```

o/p:

```
Command Window
  >> IP_Assign_1
  Max pixel size of the image: 255
fx >>
```

(d) Calculate the mean pixel value

```
>> mean_p_size = mean(GS_Image(:));
>> disp(['Mean pixel value: ' num2str(mean_p_size)]);
```

o/p:

```
  Max pixel size of the image: 255
  Mean pixel value: 67.9146
fx >>
```

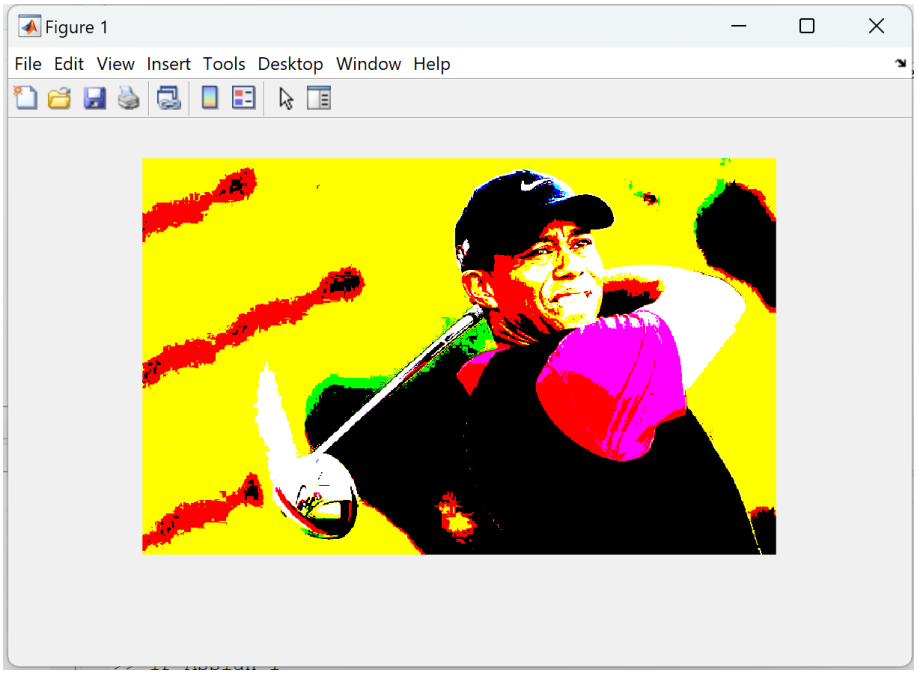
(e) Change the pixel values of the image in the following way: all pixels' values less than the average calculated at (d) will be equal to 0 and all the others will be equal to 1. Make sure your displayed image is black and white, not black and near black. **481 students (2/0):** perform your thresholding on a color image, thresholding separately in each color channel. Combine the results for display.

CODE:

```
mean_p_size = mean(OG_Image(:));
threshold_value = mean_p_size;

if size(OG_Image, 3) == 3
   red_Channel = OG_Image(:, :, 1);
   green_Channel = OG_Image(:, :, 2);
```

```
    blue_Channel = OG_Image(:, :, 3);

    % set index values to 0
    threshold_Red = zeros(size(red_Channel));
    threshold_Green = zeros(size(green_Channel));
    threshold_Blue = zeros(size(blue_Channel));

    threshold_Red(red_Channel < thresholdValue) = 0;
    threshold_Green(green_Channel < thresholdValue) = 0;
    threshold_Blue(blue_Channel < thresholdValue) = 0;

    threshold_Red(red_Channel >= thresholdValue) = 1;
    threshold_Green(green_Channel >= thresholdValue) = 1;
    threshold_Blue(blue_Channel >= thresholdValue) = 1;

    % Concatenate the results
    combined_Image = cat(3, threshold_Red, threshold_Green, threshold_Blue);
    figure;
    imshow(combined_Image);

end
```

O/P:



2. Image Interpolation (*7/5*)

Write a computer program that will, given an input image, reduce its spatial resolution (use a reduction of 1/10 to 1/20 in each dimension), and then return it to its original resolution. Use all of nearest neighbor, bilinear and bicubic interpolation to do this. Your display should include at least 7 images: the original, three reduced images and three restored images. Comment on the differences you see among the three restored images. **481 Students (*2/0*):** Design your program such that the desired change in spatial resolution (e.g. 0.5, which will halve the image in each dimension, or 2.0, which will double the image in each dimension) is a variable input to your program. Show an example run of your code using a non-integer scaling factor.

Code:

```
SF = 0.5;

red_Nearest = imresize(GS_Image, SF, 'nearest');
red_Bilinear = imresize(GS_Image, SF, 'bilinear');
red_Bicubic = imresize(GS_Image, SF, 'bicubic');

restored_Nearest = imresize(red_Nearest, 1/SF, 'nearest');
restored_Bilinear = imresize(red_Bilinear, 1/SF, 'bilinear');
restored_Bicubic = imresize(red_Bicubic, 1/SF, 'bicubic');

% Display
figure;
subplot(2, 4, 1);
imshow(GS_Image);
title('Original Image');

subplot(2, 4, 2);
imshow(red_Nearest);
title('Reduced Nearest');

subplot(2, 4, 3);
imshow(restored_Nearest);
title('Restored Nearest');

subplot(2, 4, 4);
imshow(red_Bilinear);
title('Reduced Bilinear');

subplot(2, 4, 5);
imshow(restored_Bilinear);
title('Restored Bilinear');

subplot(2, 4, 6);
imshow(red_Bicubic);
title('Reduced Bicubic');
```

```
subplot(2, 4, 7);
imshow(restored_Bicubic);
title('Restored Bicubic');
```

O/P:

The picture produced by Bilinear was the smoothest of all. Whereas the image restored by nearest-neighbor method showed a little blurring and blocky image. The bilinear also shows some blurriness but is smoother compared to Nearest-neighbor.

3. Reducing the Number of Gray Levels in an Image (*11/8*)

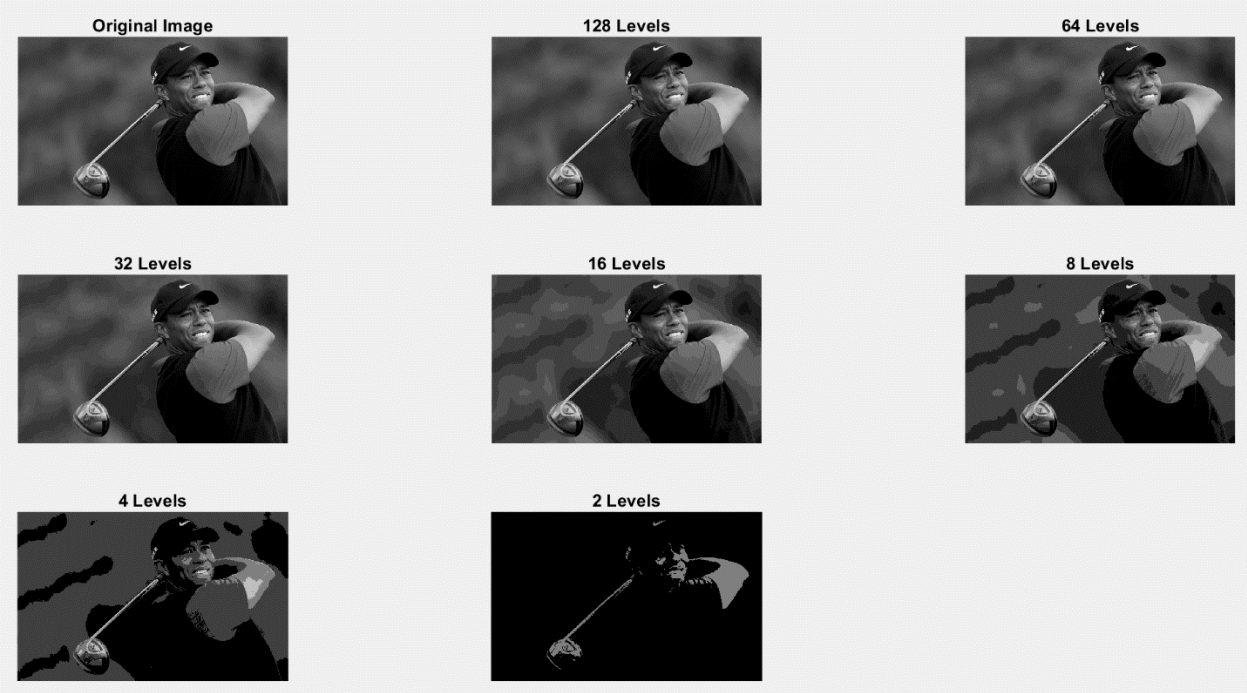
Write a computer program capable of reducing the number of gray levels in an image from 256 to 2, in integer powers of 2. Do not use the matlab function quantize or any library equivalent. Your display should include 8 images (as we saw in class), the original and the 7 reduced intensity images. Strive to have your images not to get darker and darker as you reduce the number of intensities. **481 Students (*3/0*):** In addition, write a program such that the desired number of gray levels does not have to be a power of 2. Show an example run of your code using a non-integer power of 2 number of gray levels.

CODE part A:

```
figure;
subplot(3, 3, 1);
imshow(GS_Image);
title('Original Image');

Gray_Levels = 256;
reduced_Images = cell(1, 7);
```

```
%Looping to reduce the levels "/2" each time
for i = 1:7
    Gray_Levels = Gray_Levels / 2;
    reduced_Image = floor((double(GS_Image) / 256) * Gray_Levels) * (256 / Gray_Levels);
    reduced_Images{i} = reduced_Image;

    subplot(3, 3, i + 1);
    imshow(uint8(reduced_Image));
    title([num2str(Gray_Levels) ' Levels']);
end
```

O/P:

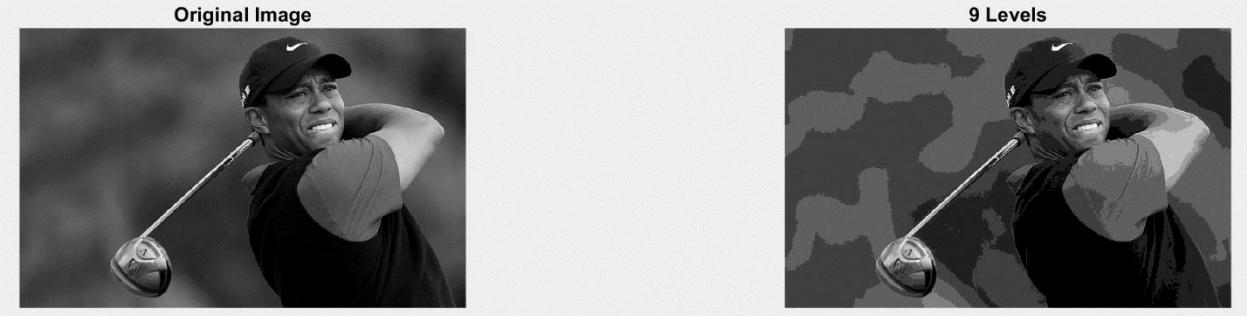


CODE Part B:

```
figure;
subplot(2, 2, 1);
imshow(GS_Image);
title('Original Image');

desired_Levels = 9;
SF = (desired_Levels - 1) / 255;

reduced_Image = round(double(GS_Image) * SF) / SF;

subplot(2, 2, 2);
```

```
imshow(uint8(reduced_Image));
title([num2str(desiredGrayLevels) ' Levels']);
```

O/P:



General submission instructions:

(a) Be kind to your aging, over-worked professor and submit only a single document. This can be pdf, MS Word, OpenOffice, etc. Do not submit a zip file.

(b) Your single document should include the input image for your problem, if required, and answers to each of the sub-problems (text, image or both, as appropriate). For example, 1(e) will require you to show an output image and a text answer, 1(a) only an image, and 1(b) only text. Your document should also include code that you wrote to generate your answers.

(c) You may use any images you like for the programming; I encourage you to use images that might be useful/interesting for your final project.

(d) Feel free to use whatever functions MatLab supplies. Also feel free to write your own, if you are so inclined; it will take more time, but you will gain a deeper understanding of the material. It is one thing, for example, to implement bicubic interpolation using the matlab `resize`, quite another to write a bicubic interpolator yourself.

(e) Feel free to use whatever programming language you like. Much of the class will use MatLab, but python, Java, C++ (and probably others) all have good support for image processing.

(f) Point values for each question are indicated as ***x/y*** in which ***x*** is the point value for 481 students and ***y*** is the point value for 381 students.