CSL461: Digital Image Analysis

Semester I, 2016 – 2017

Lab 1: Intensity Transformations

Aim:

- To get students comfortable with using MATLAB by doing some simple exercises in Intensity Transformations
- *This lab assumes that you are familiar and comfortable using the Linux operating system!!*

- **Introduction**

  As mentioned in class, the simplest form of point-to-point transformation is when a single pixel is considered and altered individually. We call such transforms intensity or gray-level transforms. In this first exercise we investigate the use of an intensity transform on a gray-scale images.

- **Let's get started!**
  - Create a directory structure to hold your work for this course and all the subsequent labs:
    - Suggestion: `CSL461/Lab1`

- **Intensity Transformation Functions**

When you are working with gray-scale images, sometimes you want to modify the intensity values. For instance, you may want to reverse black and the white intensities or you may want to make the darks darker and the lights lighter. An application of intensity transformations is to increase the contrast between certain intensity values so that you can pick out things in an image. For instance, the following two images show an image before and after an intensity transformation. Originally, the camera man's jacket looked black, but with an intensity transformation, the difference between the black intensity values, which were too close before, was increased so that the buttons and pockets became viewable.

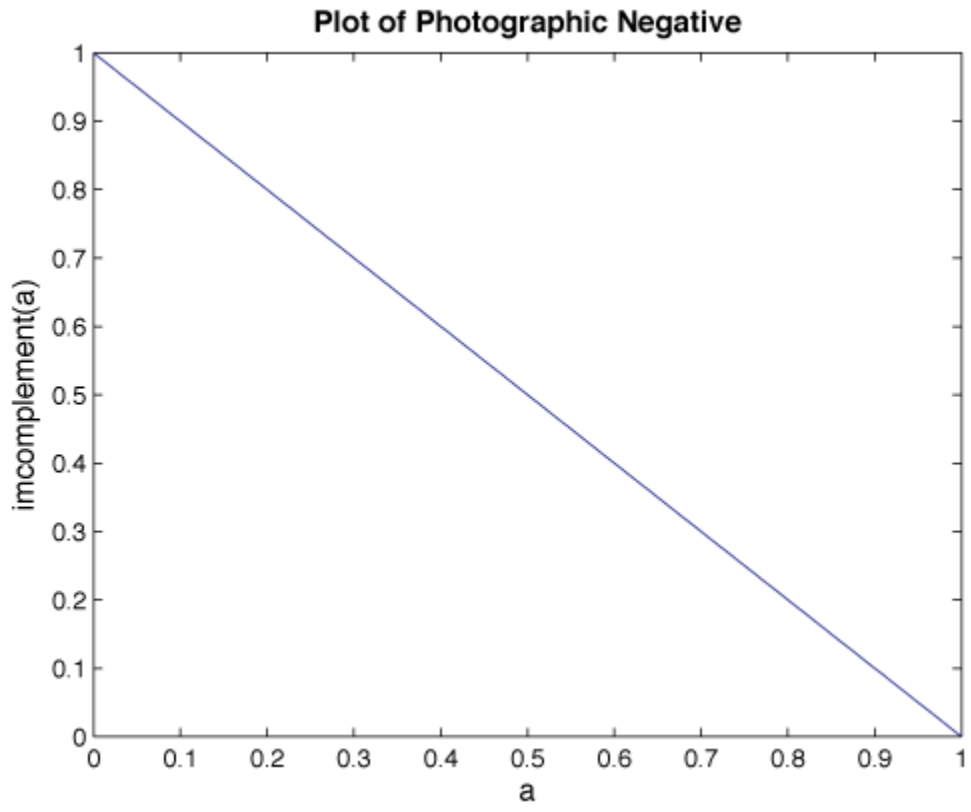
Original	After Intensity Transformation
	

Generally, making changes in the intensity is done through **Intensity Transformation Functions**. The next sections talk about four main intensity transformation functions:

1. photographic negative (using `imcomplement`)
2. gamma transformation (using `imadjust`)
3. logarithmic transformations (using `c*log(1+f)`)
4. contrast-stretching transformations (using `1./(1+(m./(double(f)+eps)).^E`)

- **Photographic Negative**

The Photographic Negative is probably the easiest of the intensity transformations to describe. Assume that we are working with grayscale double arrays where black is 0 and white is 1. The idea is that 0's become 1's, 1's become 0's, and any gradients in between are also reversed. In intensity, this means that the true black becomes true white and vice versa. MATLAB has a function to create photographic negatives: `imcomplement(f)`. Given `a=0:.01:1`, the below shows a graph of the mapping between the original values (x-axis) and the `imcomplement` function.



The following is an example of a photographic negative. Notice how you can now see the writing in the middle of the tire better than before:

Original	Photographic Negative
	

The MATLAB code that created these two images is:

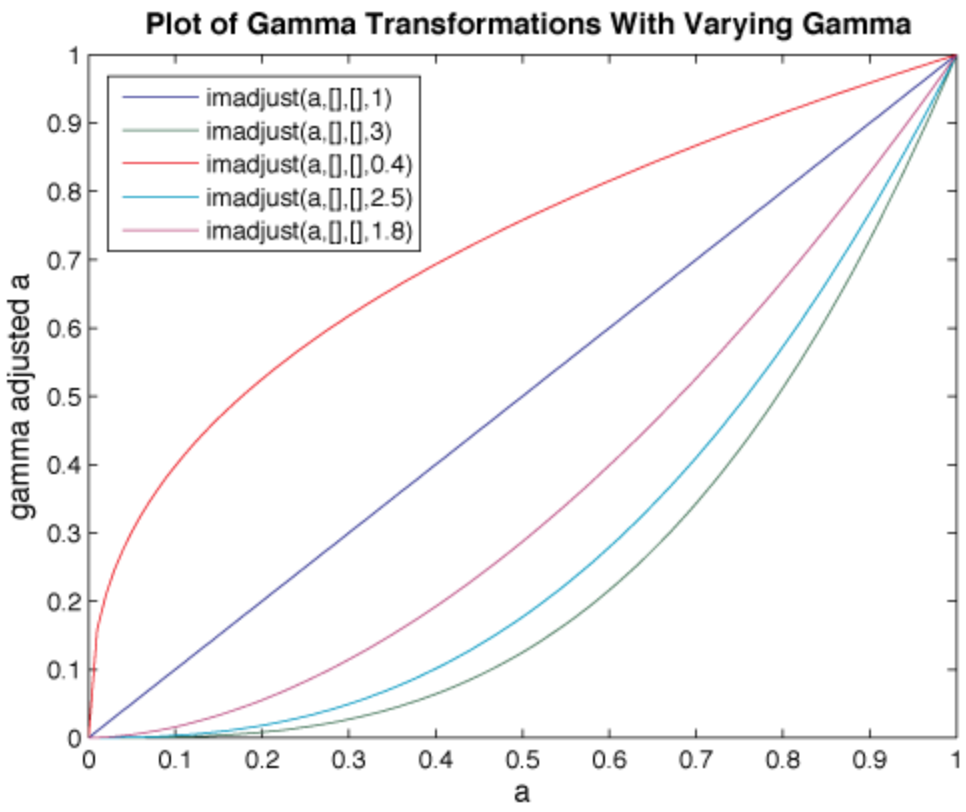
```
I= imread('tire.tif');
imshow(I)
J = imcomplement(I);
figure, imshow(J)
```

- **Gamma Transformations**

With Gamma Transformations, you can curve the grayscale components either to brighten the intensity (when `gamma` is less than one) or darken the intensity (when `gamma` is greater than one). The MATLAB function that creates these `gamma` transformations is (use command '`help imadjust`' to get more information):

```
imadjust(f, [low_in high_in], [low_out high_out], gamma)
```

`f` is the input image, gamma controls the curve, and `[low_in high_in]` and `[low_out high_out]` are used for clipping. Values below `low_in` are clipped to `low_out` and values above `high_in` are clipped to `high_out`. For the purposes of this lab, we use [] for both `[low_in high_in]` and `[low_out high_out]`. This means that the full range of the input is mapped to the full range of the output. Given `a=0:.01:1`, the following plots show the effect of the gamma transformation with varying gamma. Notice that the red line has `gamma=0.4`, which creates an upward curve and will brighten the image.



The following shows the results of three of the gamma transformations shown in the plot above. Notice how the values greater than 1 one create a darker image, whereas values between 0 and 1 create a brighter image with more contrast in dark areas so that you can see the details of the tire.

Original (and gamma=1)	gamma=3	gamma=0.4
		

The MATLAB code that created these three images is:

```
I = imread('tire.tif');
J1 = imadjust(I,[],[],1);
```

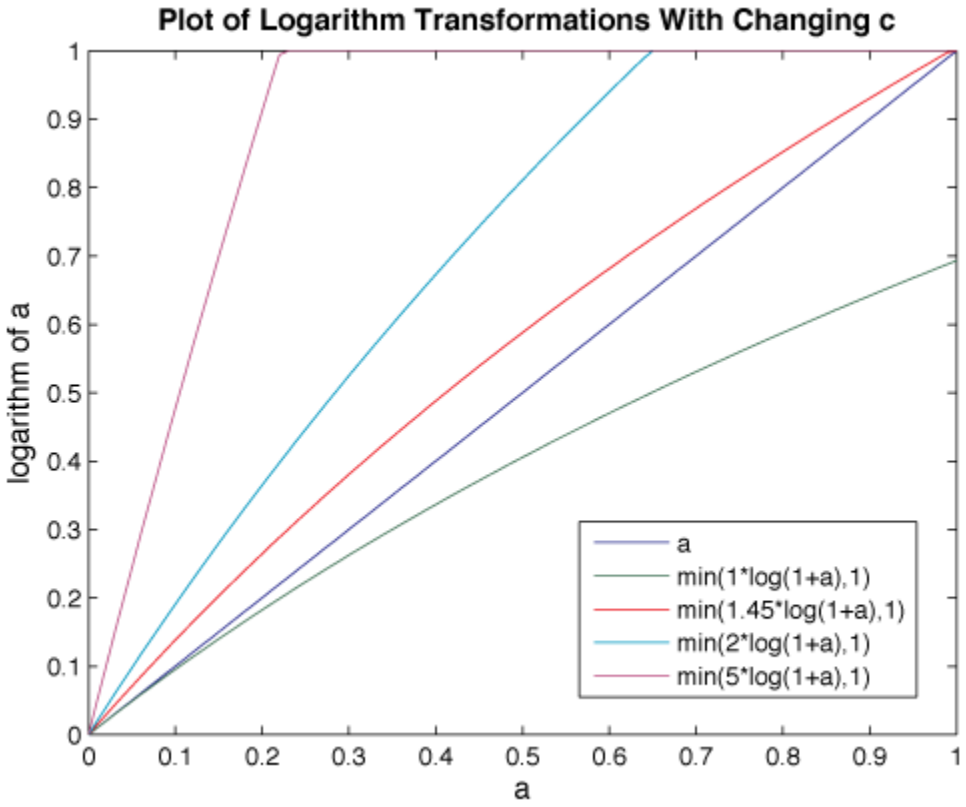
```
J2 = imadjust(I,[],[],3);
J3 = imadjust(I,[],[],0.4);
imshow(J1);
figure, imshow(J2);
figure, imshow(J3);
```

- **Logarithmic Transformations**

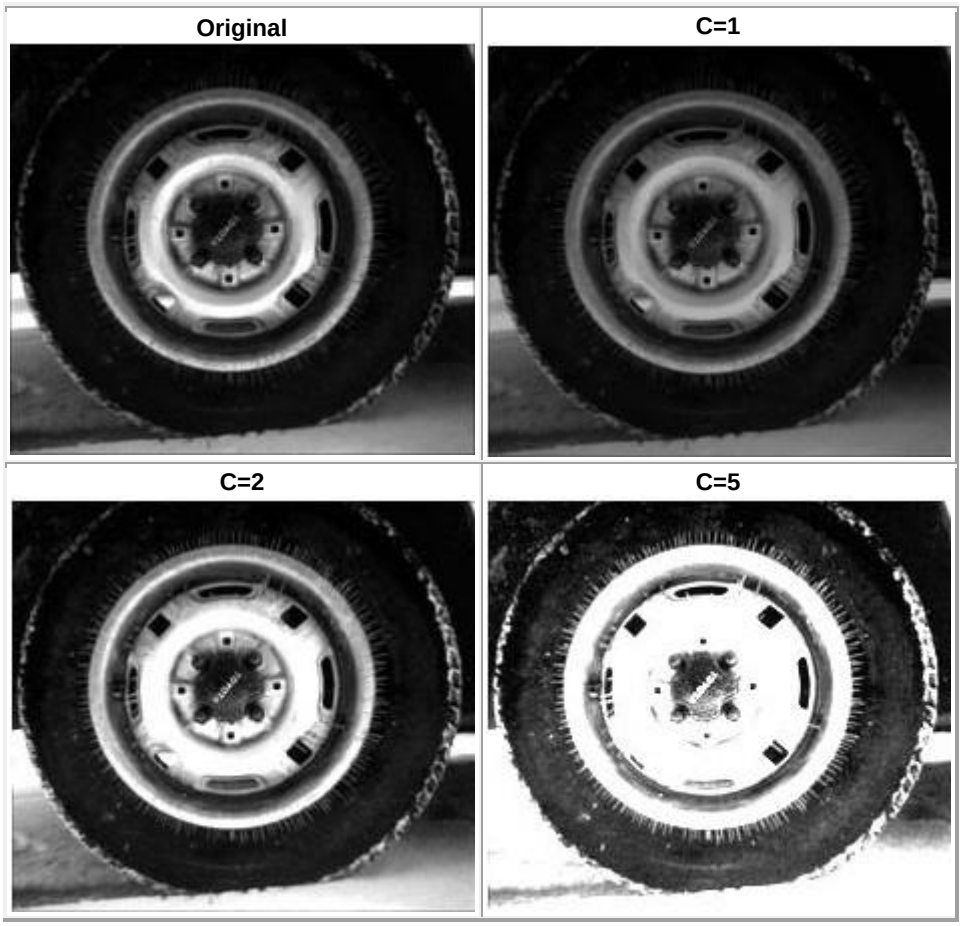
Logarithmic Transformations can be used to brighten the intensities of an image (like the Gamma Transformation, where gamma < 1). More often, it is used to increase the detail (or contrast) of lower intensity values. They are especially useful for bringing out detail in Fourier transforms (covered in a later lab). In MATLAB, the equation used to get the Logarithmic transform of image `f` is:

```
g = c*log(1 + double(f))
```

The constant `c` is usually used to scale the range of the log function to match the input domain. In this case `c=255/log(1+255)` for a uint8 image, or `c=1/log(1+1)` (~1.45) for a double image. It can also be used to further increase contrast—the higher the c, the brighter the image will appear. Used this way, the log function can produce values too bright to be displayed. Given `a=0:.01:1`, the plot below shows the result for various values of `c`. The y-values are clamped at 1 by the min function for the plot of `c=2` and `c=5` (teal and purple lines, respectively).



The following shows the original image and the results of applying three of the transformations from above. Notice that when `c=5`, the image is the brightest and you can see the radial lines on the inside of the tire (these lines are barely viewable in the original because there is not enough contrast in the lower intensities).

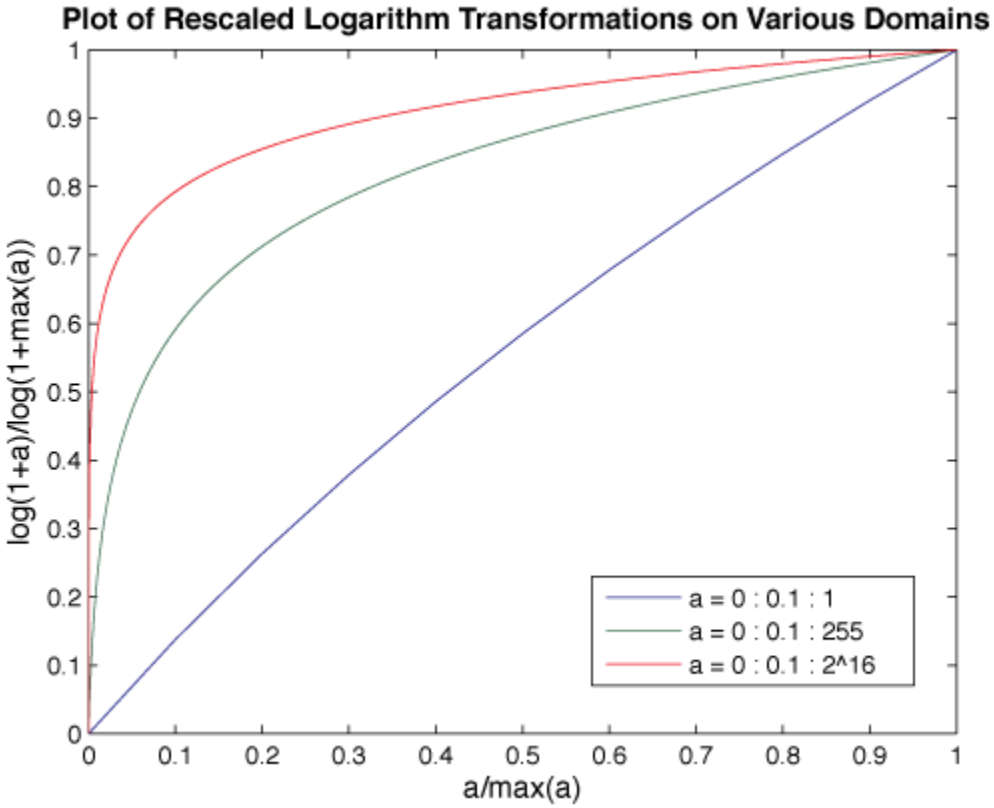


The MATLAB code that created these images is:

```
I = imread('tire.tif');
imshow(I)
I2 = im2double(I);
J1 = 1*log(1+I2);
J2 = 2*log(1+I2);
J3 = 5*log(1+I2);
figure, imshow(J1)
figure, imshow(J2)
figure, imshow(J3)
```

Notice the loss of detail in the bright regions where intensity values are clamped. Any values greater than one, produced from the scaling, are displayed as having a value of 1 (full intensity) and should be clamped. Clamping in MATLAB can be performed by the `min(matrix, upper_bound)`, and `max(matrix, lower_bound)` functions as shown in the legend for the plot above.

Although logarithms may be calculated in different bases such as MATLAB's builtin log10, log2 and log (natural log), the resulting curve, when the range is scaled to match the domain, is the same for all bases. The shape of the curve is dependent instead on the range of values it is applied to. Here are examples of the log curve for multiple ranges of input values:



It is important to be aware of this effect if you plan to use logarithm transformations successfully, so here is the result of scaling an image's values to those ranges before applying the logarithm transform:

Original Picture	log on domain [0, 1]
	
log on domain [0, 255]	**log on domain [0, 65535]**
	

The MATLAB code that produced these images is:

```
tire = imread('tire.tif');
d = im2double(tire);
figure, imshow(d);

%log on domain [0,1]
f = d;
c = 1/log(1+1);
j1 = c*log(1+f);
figure, imshow(j1);
```

```
%log on domain [0, 255]
f = d*255;
c = 1/log(1+255);
j2 = c*log(1+f);
figure, imshow(j2);

%log on domain [0, 2^16]
f = d*2^16;
c = 1/log(1+2^16);
j3 = c*log(1+f);
figure, imshow(j3);
```

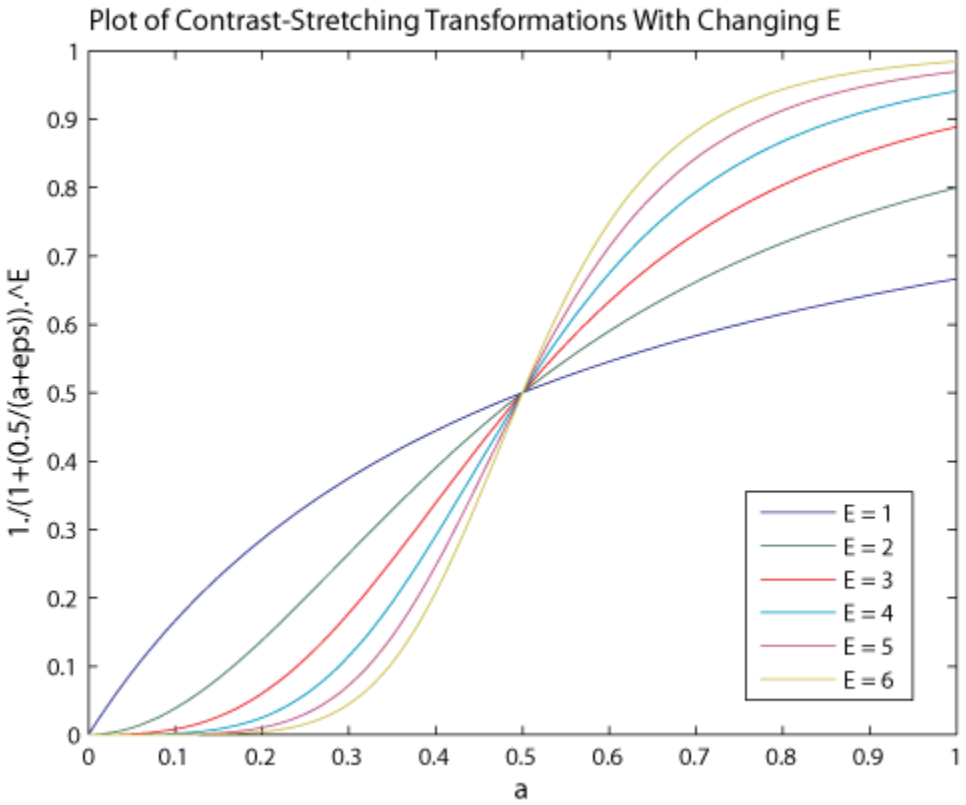
Note that for domain [0, 1] the effects of the logarithm transform are barely noticeable, while for domain [0, 65535] the effect is extremely exaggerated. Also note that, unlike with linear scaling and clamping, gross detail is still visible in light areas.

- **Contrast-Stretching Transformations**

Contrast-stretching transformations increase the contrast between the darks and the lights. Sometimes you want to stretch the intensity around a certain level. You end up with everything dark being a lot darker and everything light being a lot lighter, with only a few levels of gray around the level of interest. To create such a contrast-stretching transformation in MATLAB, you can use the following function:

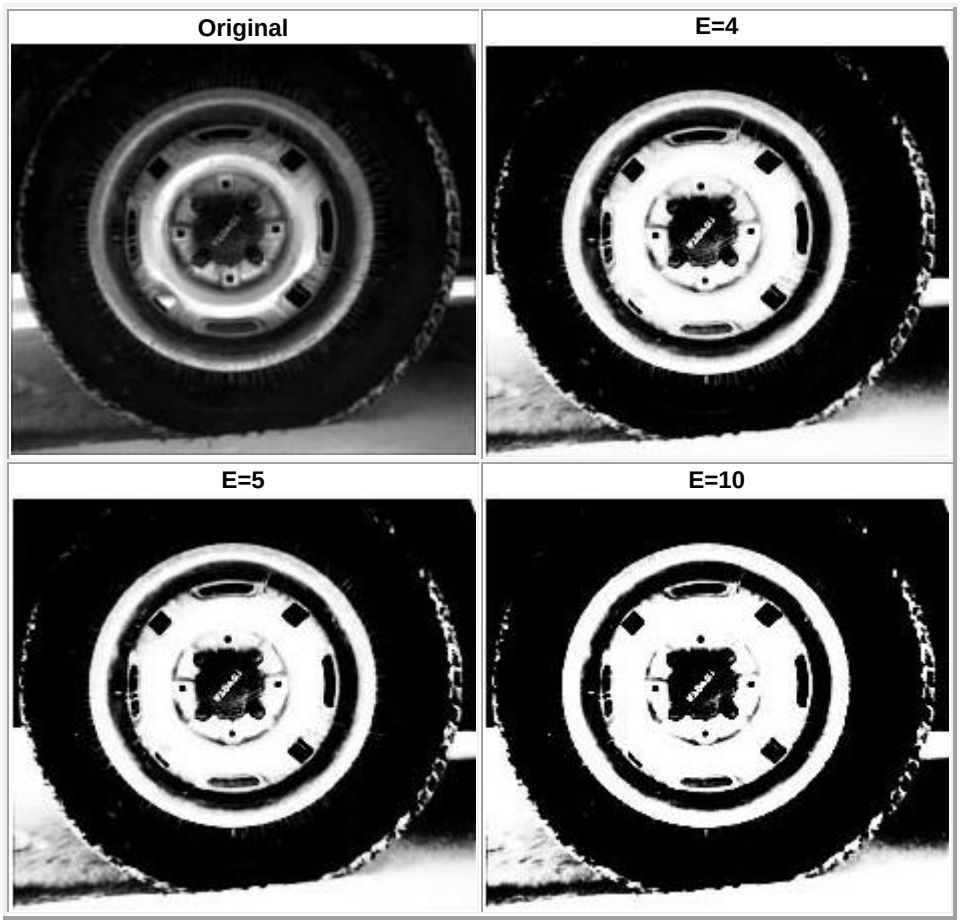
```
g=1./(1 + (m./(double(f) + eps)).^E)
```

`E` controls the slope of the function and m is the mid-line where you want to switch from dark values to light values. `eps` is a MATLAB constant that is the distance between 1.0 and the next largest number that can be represented in double-precision floating point. In this equation it is used to prevent division by zero in the event that the image has any zero valued pixels. There are two plot/diagram sets below to represent the results of changing both `m` and `E`. The below plot shows the results for several different values of `E` given `a=0:.01:1` and `m=0.5`.



The following shows the original image and the results of applying the three transformations from above. The `m` value used below is the mean of the image intensities (0.2104). At very high `E` values, such

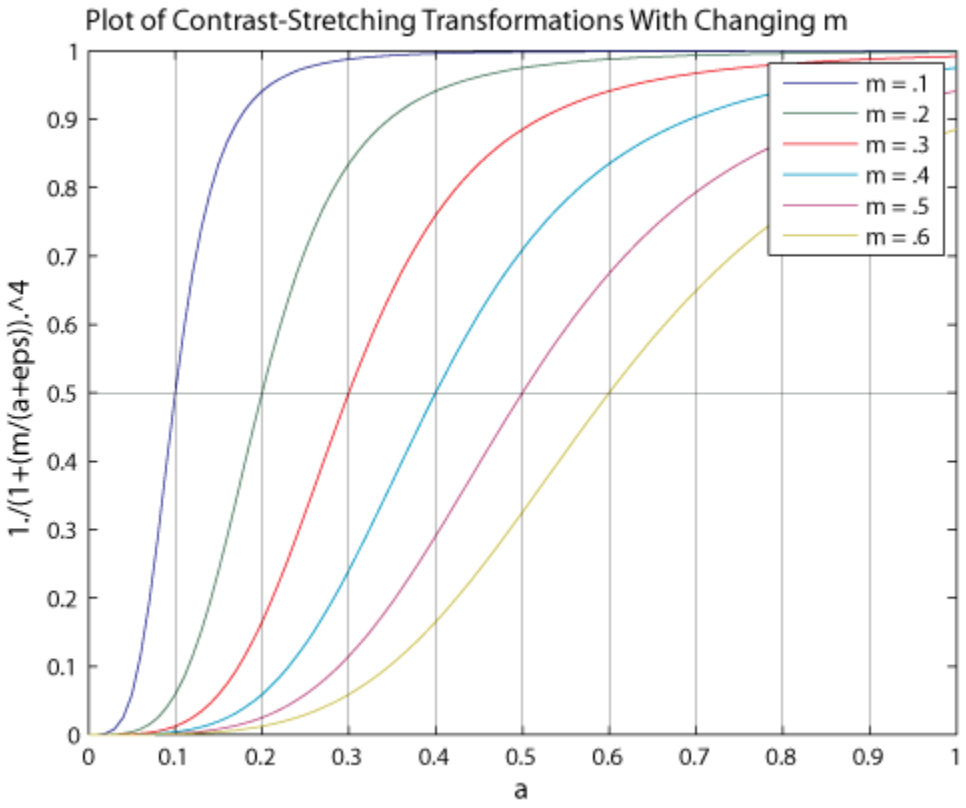
as 10, the function becomes more like a thresholding function with threshold `m`—the resulting image is more black and white than grayscale.



The MATLAB code that created these images is:

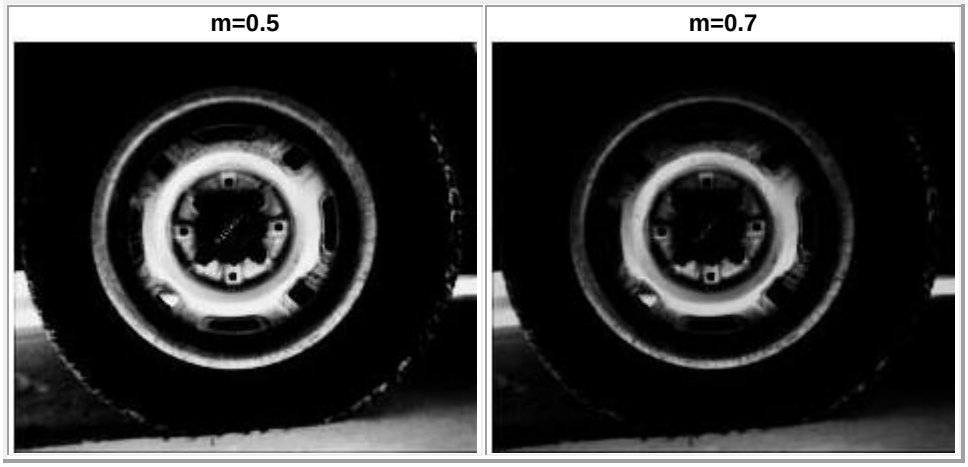
```
I=imread('tire.tif');
I2=im2double(I);
m=mean2(I2)
contrast1=1./(1+(m./(I2+eps)).^4);
contrast2=1./(1+(m./(I2+eps)).^5);
contrast3=1./(1+(m./(I2+eps)).^10);
imshow(I2)
figure,imshow(contrast1)
figure,imshow(contrast2)
figure,imshow(contrast3)
```

This second plot shows how changes to `m` (using `E=4`) affect the contrast curve:



The following shows the original image and the results of applying the three transformations from above. The `m` value used below is 0.2, 0.5, and 0.7. Notice that 0.7 produces a darker image with fewer details for this tire image.

Original	m=0.2
	



The MATLAB code that created these images is:

```
I = imread('tire.tif');
I2 = im2double(I);
contrast1 = 1./(1+(0.2./(I2+eps)).^4)
contrast2 = 1./(1+(0.5./(I2+eps)).^4);
contrast3 = 1./(1+(0.7./(I2+eps)).^4);
imshow(I2)
figure,imshow(contrast1)
figure,imshow(contrast2)
figure,imshow(contrast3)
```

- **Exercises – Histograms!!**
    - By now you should be comfortable with MATLAB and with using inbuilt functions from the image processing toolbox.
    - We have just started talking about 'Histogram Processing' in class. What better way to learn than to implement some of these utility functions from scratch!!
    - Although MATLAB will have in-built functions for most of the following, you will be implementing your own versions of the following functions in this lab.
    - Each of the functions will be implemented in a separate `.m` file and should work as an independent function with specified inputs and outputs
    - MATLAB: Creating functions in Files: https://in.mathworks.com/help/matlab/matlab_prog/create-functions-in-files.html

- **Histogram:** Given an image and number of bins, it should plot the corresponding histogram and also return frequency counts:

```
[f] = myImHist( inImg, nBins );
```

- **PDF:** Given an image and number of bins, it should plot the corresponding normalized histogram / PDF and also return the list of probabilities:

```
[p] = myImPDF ( inImg, nBins );
```

- **CDF:** Given an image and number of bins, it should plot the corresponding cumulative distribution (CDF) and return the list of corresponding values:

```
[c] = myImCDF ( inImg, nBins);
[c] = myCDF ( p )  # Second version that takes PDF as
input
```

- **Intensity Transformation Functions:** After the above exercises, creating your own versions of the transformation functions should be cake walk!

  - **Image Negative :** `myImgNeg( inImg )`
  - **Log Transformation :** `myImgLog( inImg, c )`
  - **Gamma Transformation:** `myImgGamma( inImg, m, E)`

  - As output, each of these functions should *display and return* the transformed image

- **Test Script**
  - Create a test script to showcase your functions (`TestLab1.m`)
  - The test script should run the above functions on each of the images provided with the lab (*in the order: rice, lena, mri, pout*)
  - For each input image, the script should display two figures:
    - Figure 1: Use `subplot` to display the original image, it's histogram, pdf and cdf
    - Figure 2: Use `subplot` to display the original image along with its transformations (Neg, Log, and Gamma)
    - After the figure 1 is displayed, there should be a pause for user to evaluate the results
    - When resumed, figure 1 should be closed and figure two should be displayed
    - This process then should be repeated for other sample images

- **Submitting your work:**
    - All source files and class files as one tar-gzipped archive.
        - When unzipped, it should create a directory with your ID. Example: **P2008CS1001–L1** (NO OTHER FORMAT IS ACCEPTABLE!!! Case sensitive!!!)
        - ***Negative marks if the TA has to manually change this to run his/her scripts!!***
    - Source / class files should include the following: (Case-Sensitive file names!!)
        - myImgHist.m
        - myImPDF.m
        - myImCDF.m
        - myCDF.m
        - myImgNeg.m
        - myImgLog.m
        - myImgGamma.m
        - TestLab1.m
    - ***Negative marks for any problems/errors in running your programs***
    - Submit/Upload it to Moodle
        - **Due Date/Time: 23rd Aug 2017, 5pm**