Practice Adjusting Image Contrast

In this reading provides, you'll review and practice the image contrast adjustment techniques presented in the videos. You'll also expand these techniques to work with color images.

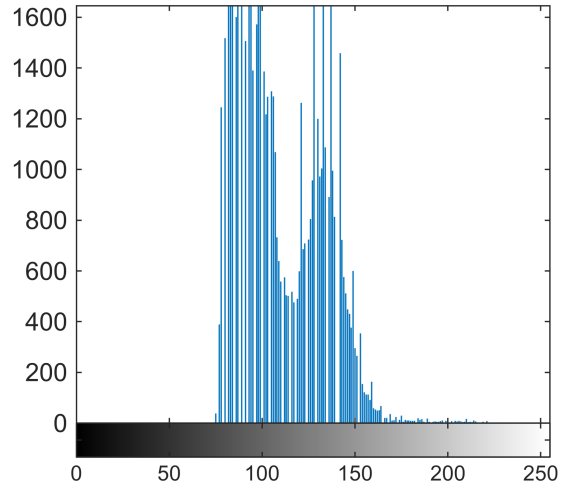
Table of Contents

Image Histograms

An image histogram shows the number of pixels in an image that take on each of the available brightness levels. For most grayscale images, the brightness values range from 0 to 255. The `imhist` function creates an image's histogram.

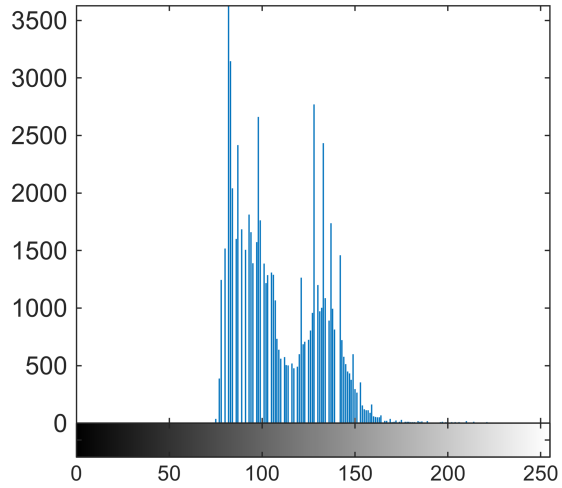
```
img = imread("pout.tif");
imshow(img)
```

```
imhist(img)
```



The histogram is scaled to allow brightness levels with fewer pixels to be seen when they have far fewer pixels than the highest peaks. The y-axis limits can be changed to show the entire histogram. This histogram has some very high peaks and only uses the middle of the available brightness values

```
imhist(img)
ylim([0 Inf])
```



Adjusting Image Histograms to Increase Image Contrast

MATLAB contains several methods for changing in images contrast:

- `imadjust` linearly stretches the histogram to use the full range of available values.
- `histeq` changes intensity values to try to utilize the full range of values equally.
- `adapthisteq` breaks the image into tiles and equalizes the histogram in each tile, then blends the results together to produce the final image.

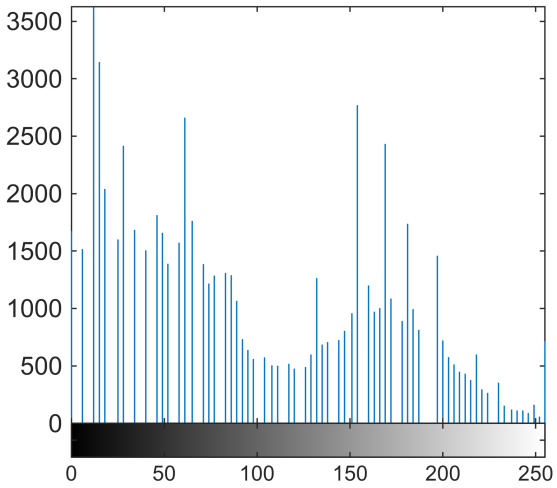
The choice of which of these methods to use will depend on the image. Here is an example of all three methods being applied to the image above and the resulting histograms. If the output is too small, you can open it in a figure window with the arrow that appears in the top right corner.

```
imgAdjust = imadjust(img);
imgEqual = histeq(img);
imgAdapt = adapthisteq(img);
montage({img, imgAdjust, imgEqual, imgAdapt}, "Size", [1 4])
```

imadjust

The `imadjust` function scales the brightness values to use the entire available range.

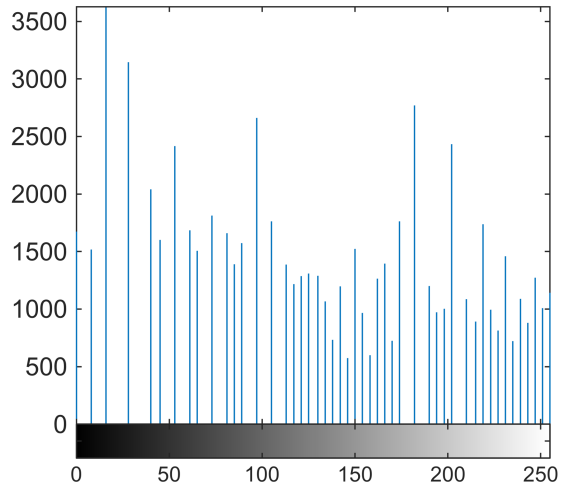
```
imhist(imgAdjust)
ylim([0 Inf])
```



histeq

The `histeq` function attempts to use all of the available brightness levels equally. While there are still high peaks, the histogram is overall flatter.

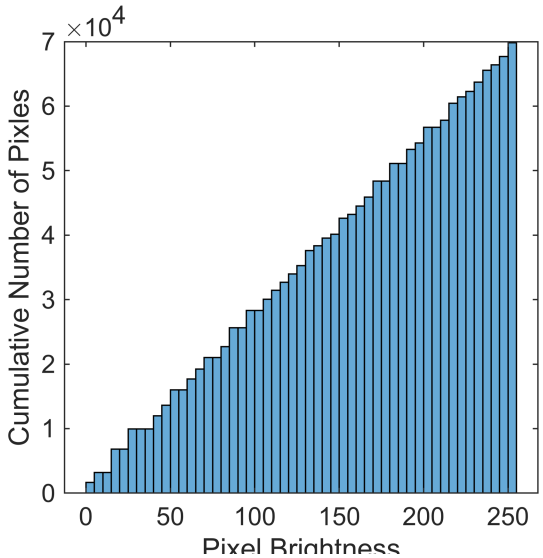
```
imhist(imgEqual)
ylim([0 Inf])
```



A Note on Histogram Equalization

The histogram produced by `histeq` doesn't usually look like a uniform distribution. The histogram shown above has a number of large peaks. A *cumulative histogram* shows the equalization more clearly. A cumulative histogram plots the number of pixels at or below a given intensity. The `histogram` function can generate a cumulative histogram using the `"Normalization"`, `"cumcount"` name-value pair. The peaks in the image histogram above correspond to the larger "stair steps" in the cumulative histogram.

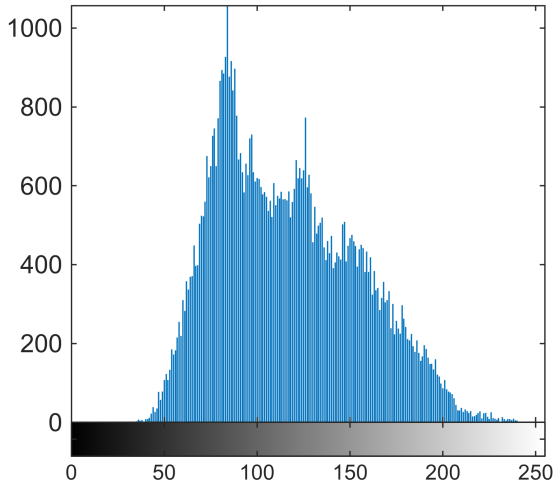
```
histogram(imgEqual, "Normalization", "cumcount")
xlabel("Pixel Brightness")
ylabel("Cumulative Number of Pixles")
```



adapthisteq

The `adapthisteq` function's behavior is more complex than the previous adjustment methods. Since the image is not adjusted as a single unit, the effect on the histogram is not as uniform.

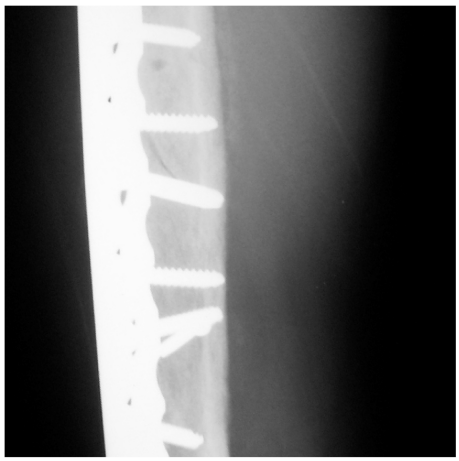
```
imhist(imgAdapt)
ylim([0 Inf])
```



Focusing on a Specific Brightness Range

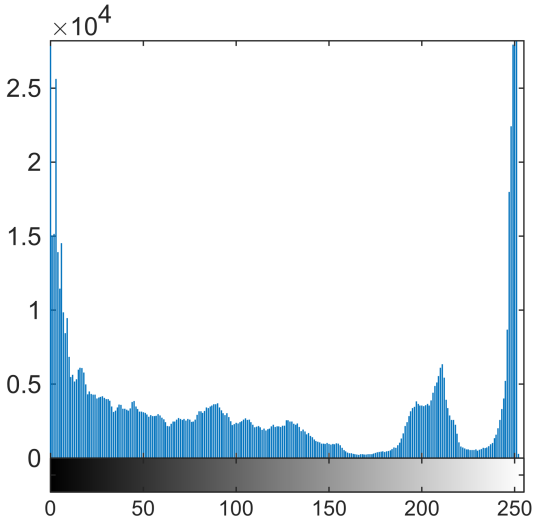
The imadjust function can focus on a specific range of intensity values. In the x-ray image below, different intensities are associated with different materials. The brightest areas are the metal pins, followed by the bone, tissue and background. Note that this image is being converted to grayscale when it is imported.

```
xrayImg = im2gray(imread("armxray.png"));
imshow(xrayImg)
```



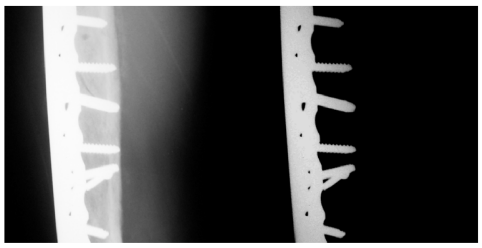
The histogram shows large peaks near certain intensity values.

```
imhist(xrayImg)
```



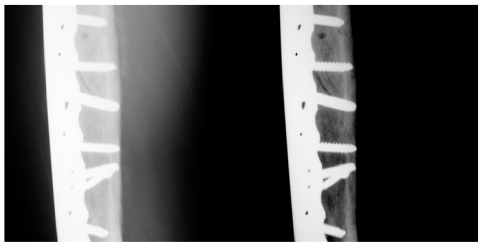
Selecting only the pixels with brightness values above 225 highlights the pins while removing the bone and tissue.

```
xrayPins = imadjust(xrayImg, [225/255, 255/255]);
montage({xrayImg, xrayPins})
```



You can adjust the slider in the code below to include different portions of the image.

```
myXray = imadjust(xrayImg, [175/255, 255/255]);
montage({xrayImg, myXray})
```



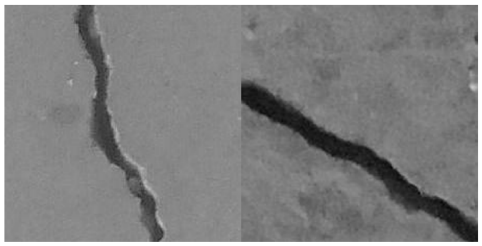
Histogram Matching

Histogram matching adjusts the brightness values in an image to match a reference image. This can be used as a preprocessing step for thresholding, allowing for a common threshold value to be applied to multiple images. Crack images 00001 and 00115 have similar brightness distributions. Both have large peaks related to the lighter foreground section and small peaks associated with the crack. The small peak in the crack image (00115) is broader and taller than the small peak in the reference image. This indicates a large crack, since more pixels will be classified as being part of the crack.

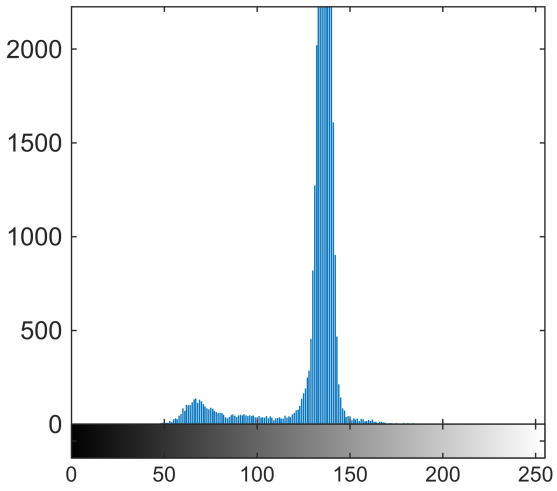
Again, the images are converted to grayscale on import.

```
refImg = im2gray(imread("00001.jpg"));
```

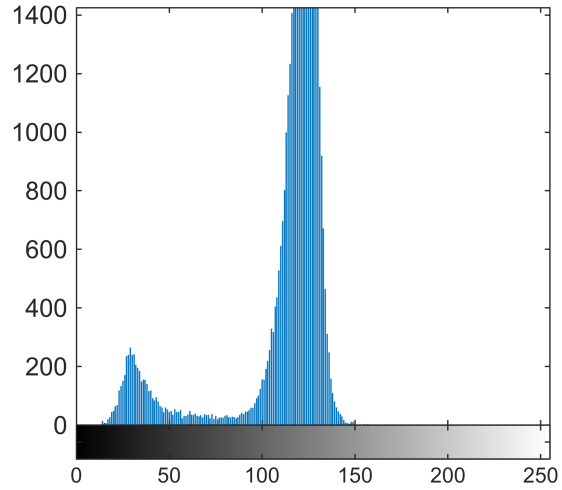
```
crackImg = im2gray(imread("00115.jpg"));
montage({refImg, crackImg})
```



```
imhist(refImg)
```




```
imhist(crackImg)
```



The threshold value calculated for the reference image (00001) is much higher than for the crack image (00115).

```
refThresh = graythresh(refImg) * 255
```

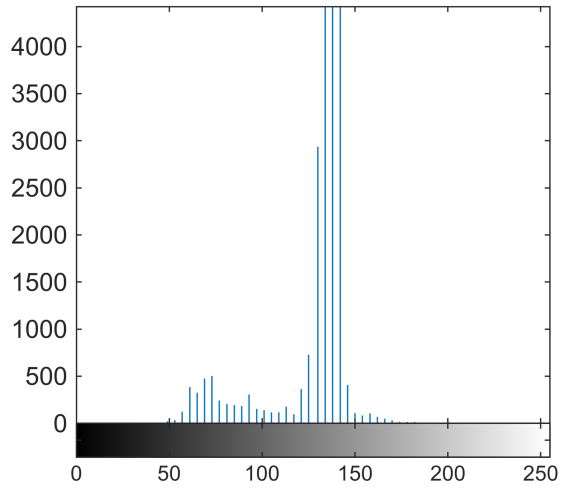
```
refThresh = 105
```

```
crackThresh = graythresh(crackImg) * 255
```

```
crackThresh = 79
```

The two cracks shown cover roughly the same fraction of the image, and the image histograms have similar shapes. The `imhistmatch` function can match the crack image (00115) to the reference image (00001) to allow a common threshold value to be applied to both images.

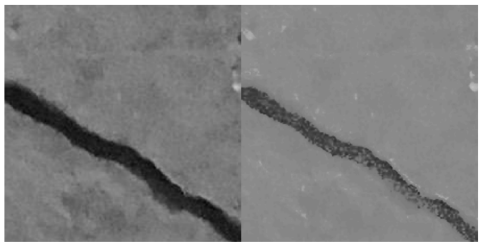
```
matchImg = imhistmatch(crackImg, refImg);
imhist(matchImg)
```




```
matchThresh = graythresh(matchImg) * 255
```

```
matchThresh = 106.5000
```

```
montage({crackImg, matchImg})
```



Since the reference image had a slightly smaller crack that the image being matched to it, some of the crack pixels have been brightened and will no longer be classified as being in the crack.

A Note on Histogram Matching

For histogram matching to be effective for image segmentation, the images need to be fairly similar. An ideal case would be for correcting the illumination of multiple images of the same object. In the case above, some of the crack pixels would be misclassified. This may be an acceptable tradeoff for not having to calculate a new threshold for each image. Your use case will determine whether computational speed or accuracy has a higher priority.

Adjusting HSV Color Planes

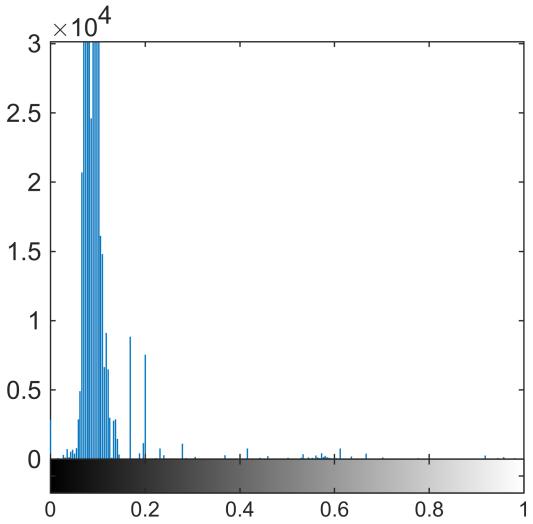
Contrast adjustment can be performed on color images as well. Working with RGB images is difficult, since changes to one color plane can change the image in unexpected ways. Adjusting the saturation and value planes of the HSV color space can often increase the clarity of the color image.

```
rgbImg = imread("sherlock.jpg");
hsvImg = rgb2hsv(rgbImg);
```

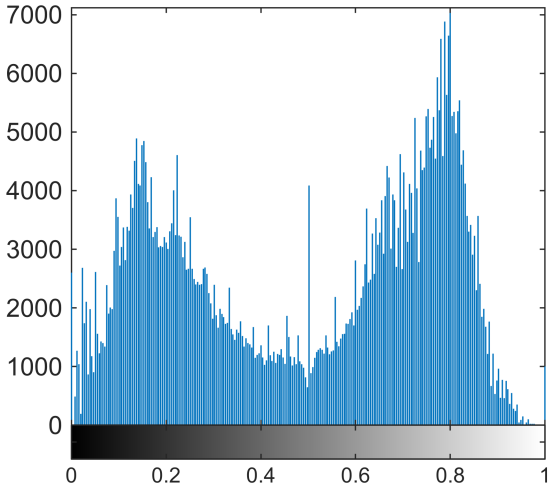
Color Plane Histograms

The histograms of the individual channels can reveal areas to be altered to increase the image clarity. Note that adjusting the hue channel can have strange effects on the final color image.

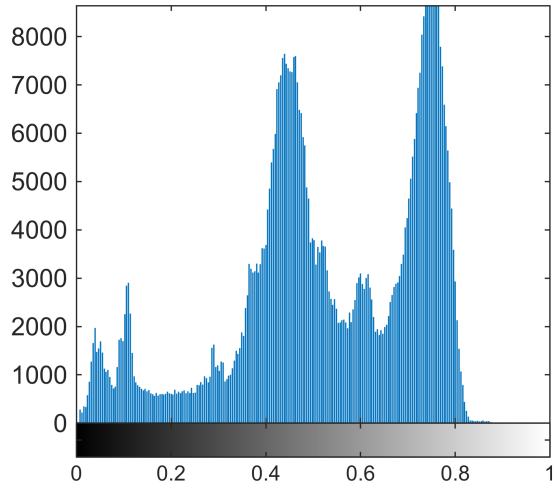
```
imhist(hsvImg(:,:,1)) % hue, direction on the top surface of the color cone (i.e., the color)
```




```
imhist(hsvImg(:,:,2)) % saturation, distance from the axis of the color cone
```



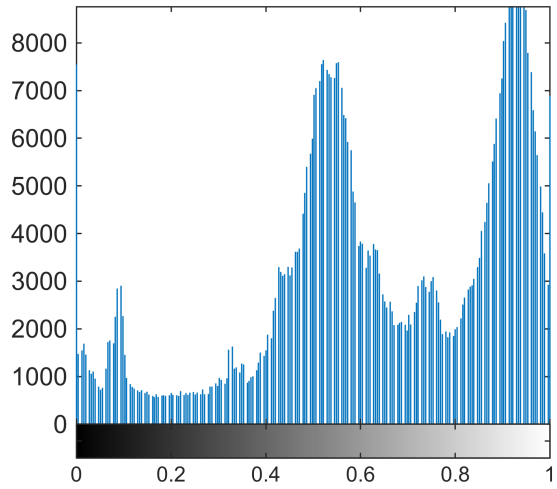

```
imhist(hsvImg(:,:,3)) % value, distance along the axis on the color cone, from the bottom
```



Color Plane Adjustment

The value channel is not using the highest 20% of the possible range. imadjust can stretch the histogram to make use of this part of the range.

```
hsvImg(:,:,3) = imadjust(hsvImg(:,:,3));
imhist(hsvImg(:,:,3))
```




```
newImg = hsv2rgb(hsvImg);
montage({rgbImg, newImg})
```

More Examples of Color Plane Manipulation

The histeq function can be used similarly to change the distribution of a single channel. The code block below contains selectable several color images. Try adjusting the saturation and value planes to achieve the clearest images.

```
rgbImg = imread("car_2.jpg");
hsvImg = rgb2hsv(rgbImg);
n = 3; % plane to adjust
hsvImg(:,:,n) = histeq(hsvImg(:,:,n));
newImg = hsv2rgb(hsvImg);
montage({rgbImg, newImg})
```