Laboratory of Image Processing

Histograms: stretching Equalization and CLAHE

Pier Luigi Mazzeo
pierluigi.mazzeo@cnr.it

Institute of Applied Sciences and Intelligent Systems
ScienceApp
National Research Council of Italy

Histograms

- Given a grayscale image, its histogram consists of the histogram of its gray levels; that is, a graph indicating the number of times each gray level occurs in the image.
- We can infer a great deal about the appearance of an image from its histogram.
- In a **dark** image, the gray levels would be clustered at the lower end
- In a **uniformly bright** image, the gray levels would be clustered at the upper end.
- In a **well contrasted** image, the gray levels would be well spread out over much of the range.
- **Problem**: Given a poorly contrasted image, we would like to enhance its contrast, by spreading out its histogram. There are **two** ways of doing this.

Histogram Calculation

Let k be an integer from 0 to 255 and represent a shade for a pixel at some location (i,j) in an mxn image matrix, u. Define a histogram of the image u to be the function from integers 0:1:255 to the integers 0:1:mn given by

$h(k)$ = number of pixels whose shade equals k ($u(i,j) = k$).

The normalized histogram is $H(k) = h(k)/(mn)$.

Histograms Stretching

Poorly contrasted image of range [a,b]

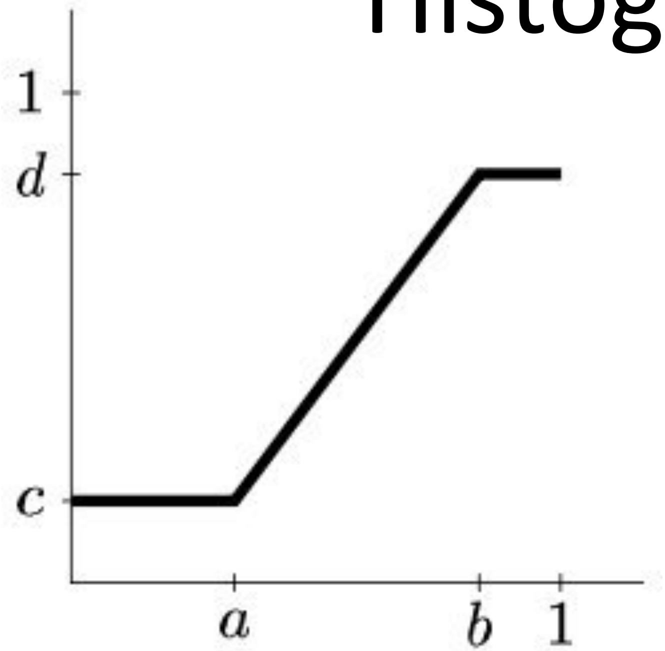
- We can stretch the gray levels in the center of the range out by applying a piecewise linear function
- This function has the effect of stretching the gray levels [a,b] to gray levels [c,d], where a<c and d>b according

$$j = \frac{(c-d)}{(b-a)} \cdot (i-a) + c$$

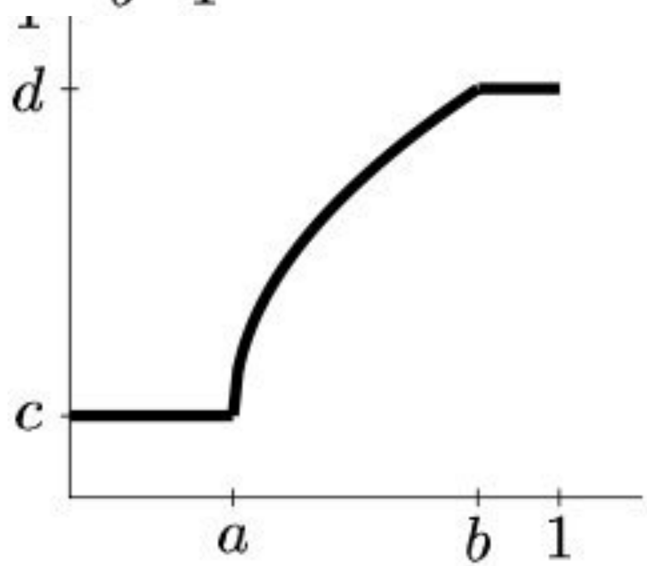
imadjust(I,[a,b],[c,d])

- Pixel values less than c are all converted to c, and pixel values greater than d are all converted to d.

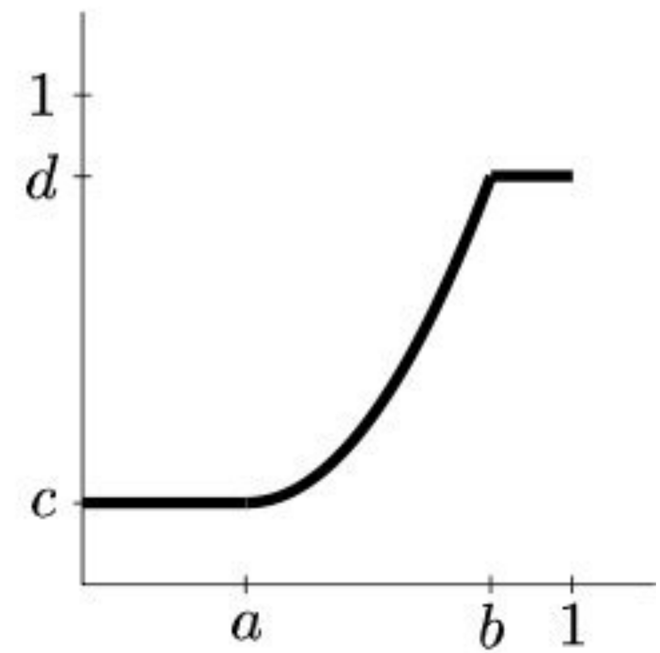
Histograms Stretching



$$y = \left(\frac{x-a}{b-a}\right)^{\gamma}(d-c)+c.$$



gamma < 1



gamma > 1

Imadjust and imhist

Try to use imadjust and imhist

```
I = imread('pout.tif');
J = imadjust(I);
figure, imshow(I), figure, imshow(J)

K = imadjust(I,[0.3 0.7],[]);
figure, imshow(K)

RGB1 = imread('football.jpg');
RGB2 = imadjust(RGB1,[.2 .3 0; .6 .7 1],[]);
figure, imshow(RGB1), figure, show(RGB2)
```

Try to use imhist

```
I = imread('pout.tif');
imhist(I)
```

Histograms Equalization

The trouble with the previous method of histogram stretching is that they require user input.

• Histogram equalization, is an entirely automatic procedure.

• Suppose an image has ***L*** different gray levels ***0,1,2,...,1-L*** and that gray level ***i*** occurs ***ni*** times in the image. Suppose also that the total number of pixels in the image is ***n*** so that ***n0+n1+n2+...nL=n***. To transform the gray levels to obtain a better contrasted image, we change gray level ***i*** to:

$$\left(\frac{n_0 + n_1 + \cdots + n_i}{n}\right)(L-1).$$

and this number is rounded to the nearest integer.

• A roughly equal number of pixels is mapped to each of the ***L*** levels, so that the histogram of the output image isapproximately flat.

Histograms Equalization with matlab

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

Practice: Create a darker image with imdivide, displayed it, and apply histogram Equalization. SHOW the Results.

Contrast-limited Adaptive Histogram Equalization (CLAHE)

adapthisteq enhances the contrast of images by transforming the values in the intensity image I. Unlike HISTEQ, it operates on small data regions (tiles), rather than the entire image. Each tile's contrast is enhanced, so that the histogram of the output region approximately matches the specified histogram. The neighboring tiles are then combined using bilinear interpolation in order to eliminate artificially induced boundaries.
The contrast, especially in homogeneous areas, can be limited in order to avoid amplifying the noise which might be present in the image.

Contrast-limited Adaptive Histogram Equalization (CLAHE)

```
 I = imread('tire.tif');
 A = adapthisteq(I,'clipLimit',
0.02,'Distribution','rayleigh');
       figure, imshow(I);
       figure, imshow(A);
```

HSV and RGB Color Space

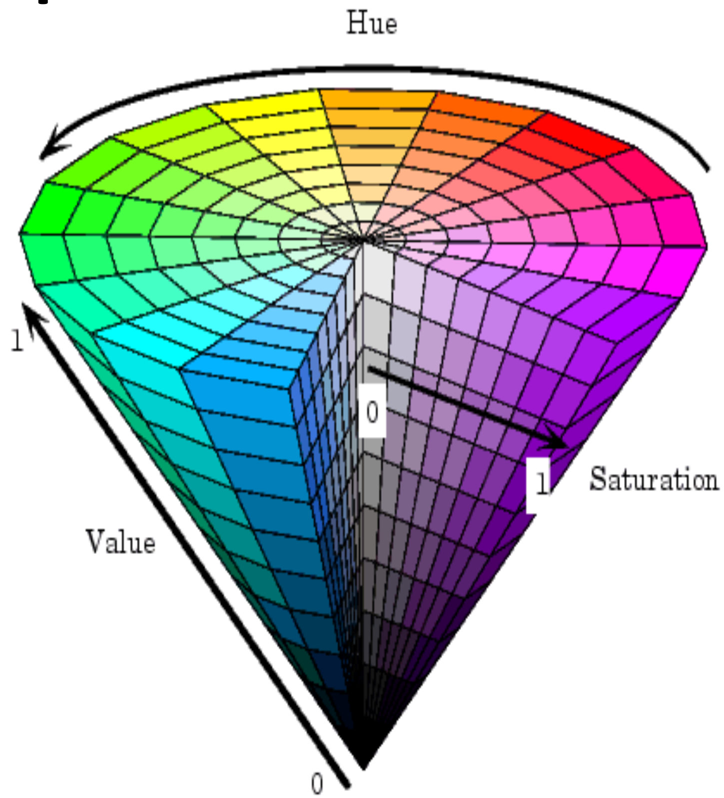
The HSV color space (hue, saturation, value) is often used by people who are selecting colors (e.g., of paints or inks) from a color wheel or palette, because it corresponds better to how people experience color than the RGB color space does. The functions rgb2hsv and hsv2rgb convert images between the RGB and HSV color spaces.

The function rgb2hsv converts colormaps or RGB images to the HSV color space. hsv2rgb performs the reverse operation. These commands convert an RGB image to HSV color space.

```
RGB = imread('flowers.tif');
HSV = rgb2hsv(RGB);
```

HSV Color Space

As hue varies from 0 to 1.0, the corresponding colors vary from red, through yellow, green, cyan, blue, and magenta, back to red, so that there are actually red values both at 0 and 1.0. As saturation varies from 0 to 1.0, the corresponding colors (hues) vary from unsaturated (shades of gray) to fully saturated (no white component). As value, or brightness, varies from 0 to 1.0, the corresponding colors become increasingly brighter.

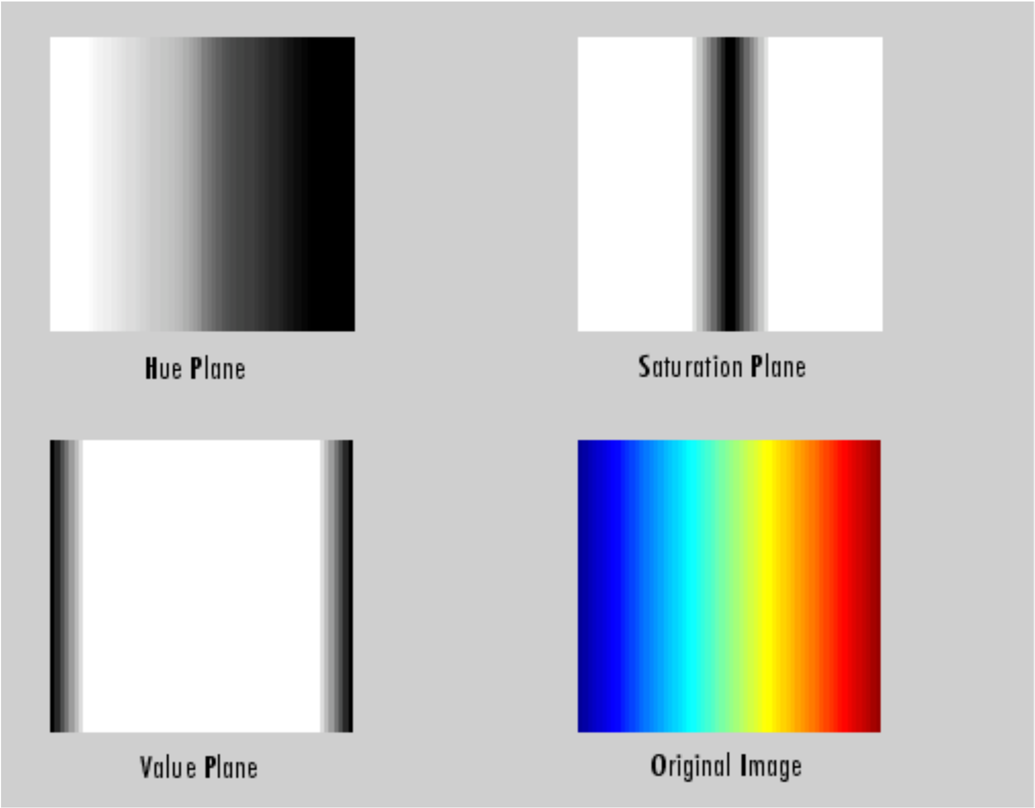


HSV and RGB Color Space

For closer inspection of the HSV color space, the next block of code displays the separate color planes (hue, saturation, and value) of an HSV image.

```
RGB=reshape(ones(64,1)*reshape(jet(64),1,192),[64,64,3]);
HSV=rgb2hsv(RGB);
H=HSV(:,:,1);
S=HSV(:,:,2);
V=HSV(:,:,3);
imshow(H)
figure, imshow(S);
figure, imshow(V);
figure, imshow(RGB);
```

Results of transformation



Other color space transformation

C = makecform(type)

creates the color transformation structure C that defines the color space conversion specified by type. To perform the transformation, pass the color transformation structure as an argument to the applycform function.

Type	Description
'cmyk2srgb'	Convert from the *CMYK* color space to the *sRGB* color space.
'lab2lch'	Convert from the $L^*a^*b^*$ to the L^*ch color space.
'lab2srgb'	Use `lab2rgb` instead.
'lab2xyz'	Use `lab2xyz` instead.
'lch2lab'	Convert from the L^*ch to the $L^*a^*b^*$ color space.
'srgb2cmyk'	Convert from the *sRGB* to the *CMYK* color space.
'srgb2lab'	Use `rgb2lab` instead.
'srgb2xyz'	Use `rgb2xyz` instead.
'upvpl2xyz'	Convert from the *u′v′L* to the *XYZ* color space.
'uvl2xyz'	Convert from the *uvL* to the *XYZ* color space.
'xyl2xyz'	Convert from the *xyY* to the *XYZ* color space.
'xyz2lab'	Use `xyz2lab` instead.
'xyz2srgb'	Use `xyz2rgb` instead.
'xyz2upvpl'	Convert from the *XYZ* to the *u′v′L* color space.
'xyz2uvl'	Convert from the *XYZ* to the *uvL* color space.
'xyz2xyl'	Convert from the *XYZ* to the *xyY* color space.

Apply CLAHE to Color Image

```
[X MAP] = imread('shadow.tif');

RGB = ind2rgb(X,MAP); % convert indexed image to truecolor
format
cform2lab = makecform('srgb2lab');
LAB = applycform(RGB, cform2lab); %convert image to L*a*b
color space
L = LAB(:,:,1)/100; % scale the values to range from 0 to 1
LAB(:,:,1) = adapthisteq(L,'NumTiles',[8 8],'ClipLimit',
0.005)*100;
cform2srgb = makecform('lab2srgb');
J = applycform(LAB, cform2srgb); %convert back to RGB
figure, imshow(RGB); %display the results
figure, imshow(J);
```

Query Image By Histogram Similarity

- Content-based image retrieval is the task of searching images in databases by analyzing the image contents. In this demo, a simple image retrieval method is presented, based on the color distribution of the images. The user simply provides an "example" image and the search is based upon that example (query by image example).

- Almost 1000 images have been used for populating the database. For each image a 3-D histogram of it's HSV values is computed. At the end of the training stage, all 3D HSV histograms are stored in the same .mat file.

- Download Image ImQuery.zip from my website.

Query Image By Histogram Similarity

In order to retrieve M (user-defined) query results, the following steps are executed:

1. The 3D (HSV) histogram of the query image is computed. Then, the number of bins in each direction (i.e., HSV space) is calculated.
2. For each image i in the database: Load its histogram Hist(i).
3. For each 3-D hist bin, compute the distance (D) between the hist of the query image and the i-th database image.
4. The similarity measure is defined as Euclidean distance among histograms.
5. Sort the similarity vector and prompt the user with the images that have the M smaller S values.

Query Image By Histogram Similarity

```
function [Hist, RGBt] = getImageHists(imageName, PLOT)

% read RGB data:
RGB = imread(imageName);
RGBt = RGB;
RGB = rgb2hsv(RGB);

% get image size:
[M,N,ttt] = size(RGB);

range = 0.0:0.1:1.0;

Hist = zeros(length(range),length(range),length(range));

for (i=1:M)
    for (j=1:N)

        nn1 = round(RGB(i,j,1) * 10)+1;
        nn2 = round(RGB(i,j,2) * 10)+1;
        nn3 = round(RGB(i,j,3) * 10)+1;

        Hist(nn1, nn2, nn3) = Hist(nn1, nn2, nn3) + 1;

    end
end

Hist = Hist / (M*N);
```

Query Image By Histogram Similarity

```
function searchImageHist(imageName, modelName, nResults)

    % load train model:
    load(modelName);

    % compute 3-D image histograms (HSV color space):
    fprintf('Computing 3-D (HSV) histogram for query
    image...\n');
    [Hist, RGBQ] = getImageHists(imageName);

    % number of training samples:
    Nfiles = length(Hists);

    % decision thresholds:
    t=0.01;
```

Query Image By Histogram Similarity

```
function searchImageHist(imageName, modelName, nResults)

for (i=1:Nfiles) % for each file in database:
    files{i}(8)='/'; %FOR MAC
    files{i}(9)='/';
    % compute (normalized) euclidean distance for all hist
bins:

    DIFF = abs(Hist-Hists{i}) ./ Hist;
    % keep distance values for which the corresponding
query image's values
    % are larger than the predefined threshold:
    DIFF = DIFF(Hist>t)

    % compute the similarity meaasure:

    Similarity(i) = mean(DIFF)';
```

Query Image By Histogram Similarity

```
function searchImageHist(imageName, modelName, nResults)

% find the nResult "closest" images:
[Sorted, ISorted] = sort(Similarity);

NRows = ceil((nResults+1) / 3);

% plot query image:
subplot(NRows,3,1); imshow(RGBQ); title('Query
Image');

% ... plot similar images:
for (i=1:nResults)
```

Skin Detection by thresholding

This file helps identify the presence of a human face, hand or any other body part by identifying and marking skin-like pixels within a given image. With further image processing techniques, the output produced by this script can be refined and processed to be fed into larger face detection and tracking, gesture recognition, and other HCI applications.

Skin Detection by thresholding

Gray world is among the simplest estimation methods. The main premise behind it is that in a normal well color balanced photo, the average of all the colors is a neutral gray. Therefore, we can estimate the illuminant color cast by looking at the average color and comparing it to gray.

Download generate_skinmap.zip from my website

```
function out = grayworld(I)
    out = uint8(zeros(size(I,1), size(I,2), size(I,
3)));
    %R,G,B components of the input image
    R = I(:,:,1);
    G = I(:,:,2);
    B = I(:,:,3);
    %Inverse of the Avg values of the R,G,B
    mR = 1/(mean(mean(R)));
    mG = 1/(mean(mean(G)));
    mB = 1/(mean(mean(B)));
```

Skin Detection by thresholding

```
%Smallest Avg Value (MAX because we are dealing with the
inverses)
    maxRGB = max(max(mR, mG), mB);

    %Calculate the scaling factors
    mR = mR/maxRGB;
    mG = mG/maxRGB;
    mB = mB/maxRGB;

    %Scale the values
     out(:,:,1) = R*mR;
     out(:,:,2) = G*mG;
     out(:,:,3) = B*mB;
end
```

Skin Detection by thresholding

```
function [out bin] = generate_skinmap(filename)


%   The function reads an image file given by the
input parameter string
%   filename, read by the MATLAB function 'imread'.
%   out - contains the skinmap overlayed onto the
image with skin pixels
%   marked in blue color.
%   bin - contains the binary skinmap, with skin
pixels as '1'.
%
%   Example usage:
%       [out bin] = generate_skinmap('nadal.jpg');
%       generate_skinmap('nadal.jpg');
```

Skin Detection by thresholding

```
 function [out bin] = generate_skinmap(filename)
if nargin > 1 | nargin < 1
        error('usage: generate_skinmap(filename)');
    end;

    %Read the image, and capture the dimensions
    img_orig = imread(filename);
    height = size(img_orig,1);
    width = size(img_orig,2);

    %Initialize the output images
    out = img_orig;
    bin = zeros(height,width);

    %Apply Grayworld Algorithm for illumination compensation
    img = grayworld(img_orig);
```

Skin Detection by thresholding

```
function [out bin] = generate_skinmap(filename)
    %Convert the image from RGB to YCbCr
        img_ycbcr = rgb2ycbcr(img);
        Cb = img_ycbcr(:,:,2);
        Cr = img_ycbcr(:,:,3);

  %Detect Skin
    [r,c,v] = find(Cb>=77 & Cb<=127 & Cr>=133 & Cr<=173);
    numind = size(r,1);

    %Mark Skin Pixels
    for i=1:numind
        out(r(i),c(i),:) = [0 0 255];
        bin(r(i),c(i)) = 1;
    end
    imshow(img_orig);
    figure; imshow(out);
    figure; imshow(bin);
end
```

Exercises

The transformation matrix to convert a truecolor image from RGB to YIQ (also note as NTSC) is the inverse of the following matrix:

1	0.956	0.621
1	-0.272	-0.647
1	-1.106	1.703

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595716 & -0.274453 & -0.321263 \\ 0.211456 & -0.522591 & 0.311135 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Calculate the inverse of this matrix and transform the RGB to YIQ. Are there differences with rgb2gray?