# CSC317 Digital Image Processing

# Lecture 4 Point Process

## 4.1 Introduction

A screenshot of a cell phone

Description automatically generated

Fig. 4.1 Schema for Transform Processing

There are *three* classes of image processing:

1. Point Processes – A pixel’s grayscale value is changed without the knowledge of the surrounding. A histogram stretching or equalization is a typical example.
2. Local Processes – Neighborhood processing using a kernel/operator/filter. Linear or nonlinear kernels are used.
3. Global Transformation – A “transform” represents the pixel values in some other, but equivalent, form. Examples include Geometric Transformations, Fourier Transforms.

## 4.2 Arithmetic Operations

We will begin from the simplest, i.e., a Point Process to adjust the contrast of images. Before that, we are going to take a look at some arithmetic operations with the grayscale value, e.g., adding and subtraction some values.

k

255-n

255-k+1 n+1

Fig. 4.2 Adding Fig. 4.3. Subtracting

These operations can be represented by the following mapping:

y= ∈[0 .. 255] and y ∈[0 .. 255]

Examples:

Y = f(x) = x + C or x – C # C is a constant

Y = C x # y <- 255 if y > 255; y<- 0 if y < 0

In Fig. 4.2,

new value = (the original value + k) mod 256, (0 <= k <= 255)

In Fig. 4.3,

new value = (the original value + 255 – n) mod 256 ( 0<= n <=255)

Try the following in your python environment:

1. Display the original image blocks.jpg

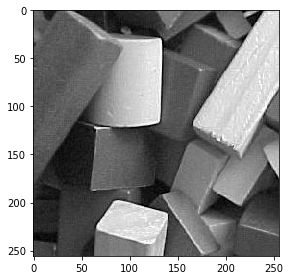


Fig. 4.4 The Original Image

﻿>>>import skimage.io as io

>>>b = io.imread('../../OriginalImages/blocks.jpg')

>>>io.imshow(b)

1. Display the image of dividing the grayscale level by 2 (b/2)

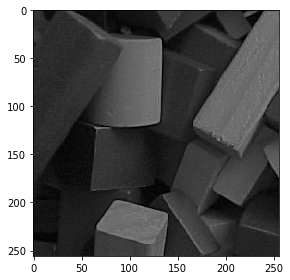


Fig. 4.5 Arithmetic Operations: Divided the Gray Level by 2

>>>﻿import skimage.io as io

>>>b = io.imread('../../OriginalImages/blocks.jpg')

>>>io.imshow(b/2)

1. Display the image of multiplying the gray level by 2 (b\*2)

A picture containing photo, indoor

Description automatically generated

Figure 4.6: Arithmetic Operations: Multiply the Gray Level by 2

﻿>>> ﻿import skimage.io as io

>>> import numpy as np

>>> b = io.imread('../../OriginalImages/blocks.jpg')

>>> io.imshow(b)

>>> bf = np.float64(b)

>>> b1 = np.uint8(np.clip(bf\*2, 0, 255))

>>> io.imshow(b1)

1. Display the image of b/2+128

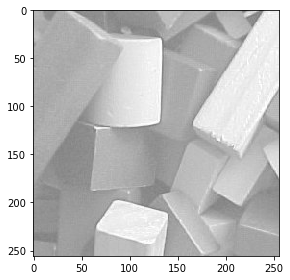


Figure 4.7: Arithmetic Operations: Divide the Gray Level by 2 and add 128

>>>import skimage.io as io

>>>import numpy as np

>>>b = io.imread('../../OriginalImages/blocks.jpg')

>>>io.imshow(b)

>>>bf = np.float64(b)

>>>b1 = np.uint8(np.clip(bf/2+128, 0, 255))

>>>io.imshow(b1)



Figure 4.8: Adding the Gray Level by 128

In Figure 4.8, the grayscale level of each pixel is added by 128. The effect is to make the whole image brighter. In Figure 4.9, the image becomes it’s negative by subtracting 255 from the original gray level.

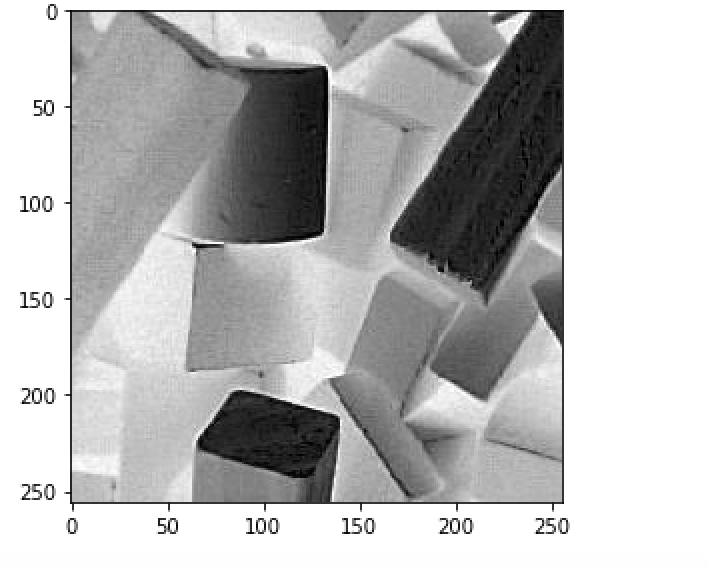


Figure 4.9: Complement: Image Negative

In summary, with these simple operations we have applied, the contrast was not improved; only the brightness or darkness was changed. What we may want is to *increase* the contrast as the objects in Figure 4.8; or *decrease* the contrast of the objects inside an image, e.g., improve the contrast, of an image in Figure 4.9. We just call it “*contrast enhancement*”.

There are two ways to perform contrast enhancement: histogram stretching and histogram equalization. In the following sections, we will explain the principles and demonstrate with programs.

## 4.3 Histogram Stretching (Contrast Stretching) (P. 74 – P. 77)

The histogram



Figure 4.10: Image of wo chickens and the histogram

In Figure 4.10, there are two chickens: one is extremely white, and another is extremely dark. We can show the total number of pixels with various gray levels using a histogram. In this

The *histogram* of an image is *a graph indicating the number of pixels with specific gray levels in a given image*. We can apply *Histogram Stretching* to an image if a stretching function is given. Otherwise, we need to apply *Histogram Equalization* to change the contrast of an image.

## Histogram Stretching (P. 73 – P. 74)

In this section, we use simpler examples to illustrate how Histogram Stretching is completed with a given shtretching range of gray levels from one range to another.

The original gray levels and the number of pixels with each gray level is shown in the table below:

i ni

|  |  |
| --- | --- |
| 0 | 15 |
| 5 | 70 |
| 6 | 110 |
| 7 | 45 |
| 8 | 70 |
| 9 | 35 |
| 15 | 15 |



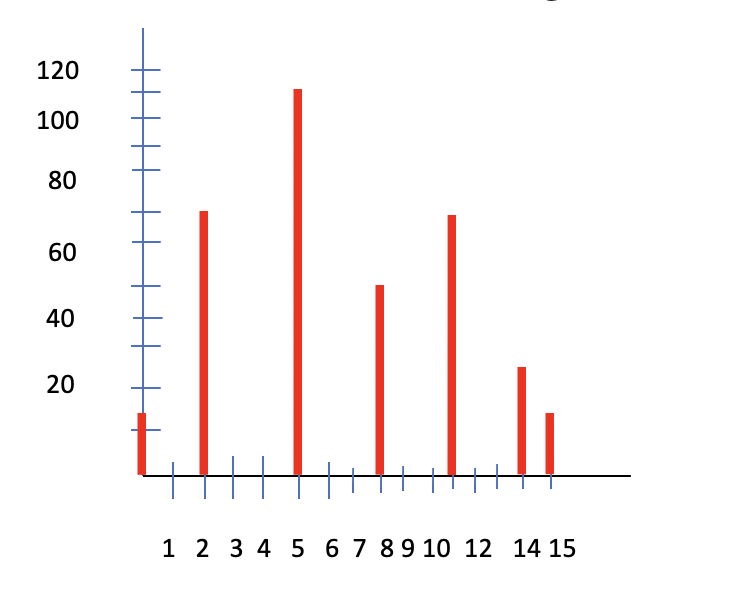
Fig. 4.11 Before Fig. 4.12 Stretching Function

Fig. 4.13 After Stretching

The problem of Histogram Stretching requires the background about finding the equation of a straight line. Usually, the stretching is given as ranges. For example, the example stretches the range of gray levels falling in a – b to c – d as indicated in Fig. 4.12 where the range of 5 – 9 is stretched to the range of 2 - 14.

### Procedure to Find the Mapping Function

The procedure to find the equation for the straight line from the point (a, c) to (b, d) is illustrated below:

**Step #1** Find the slope m of the straight line.

*m* = (d-c)/(b-a) = 3

**Step#2** Use the point (a, c) and the slope m to represent the equation of the straight line as

*m* = (y-c)/(x-a)

**Step #3** Simplify the above equation:

y - c = *m* (x – a) or y = *m* (x – a) + c

y = 3 (x – 5) + 2

x = 5, 6, 7, 8, 9 => y = 2, 5, 8, 11, 14

(This is the solution before drawing the final Histogram as in Fig. 4.13)

*(There is an error in Fig. 4.13 of our textbook: y = 5 instead of 4 when x = 6)*

Now, you may take a look at the effects of the Histogram Stretching. A range of gray levels of 5 to 9 is changed to the range of 2 to 14. The gray level values are spread wider than the original gray levels. That simply “stretches out” the “contrast” of pixel gray values.

### Adjustment with Gamma

There is a function exposure.adjust\_gamma() in the Skimage module that provides the function of non-linear stretching function based on the value of in addition to the ranges:

Diagram, text

Description automatically generated

Using python, we can write the program with the value of 0.5 for in the following example:

>>> import skimage.exposure as ex

>>> c = io.imread(‘chickens.png’)

>>> ca1 = ex.adjust\_gamma(c, 0.5)

As illustrated in our textbook, the adjusted images look different from the original image using different adjustment values.



Figure 4.14: Stretching with different adjustment values for

## Histogram Equalization (P. 78 – P. 82)

Histogram Stretching requires a user to provide a mapping function which may not be available.

Applying Histogram Equalization can change Fig. 4.15 to Fig. 4.16.



Fig. 4.15 Fig. 4.16

The image and the histogram of the previous image in Fig. 4.10 are shown below after equalization.

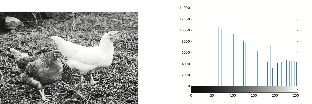


Figure 4.17: After equalization

Please refer to P.78 – P.79 of the textbook for the detailed steps to complete the Histogram Equalization. At the end of this section, I used another example to illustrate the Histogram Equalization.

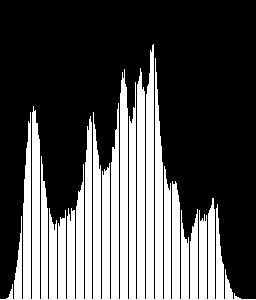
## In summary

If we provide a function to map one image to another with the goal of enlarging the range of gray levels (defined by Graylevelmax – Graylevelmin), this operation is called a *Histogram Stretching*. If the operation of image stretching is based on the function of a uniform distribution, the operation is called a *Histogram Equalization*.

Example: Histogram Equalization

Below is an example of before and after applying the histogram equalization to the image

Lena.png with the histograms. This is an example result of one of the homework exercises.



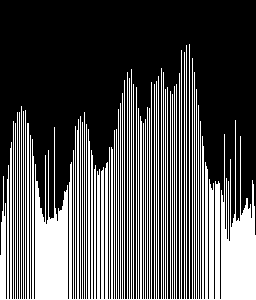


Figure 4.18(a)-(d): Histogram Equalization

### EXAMPLES

**Example #1** Shift the grayscale value ranges from 1-8 to 1-20

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 4 | 5 |
| 7 | 7 | 8 | 2 |
| 3 | 1 | 2 | 3 |
| 5 | 4 | 6 | 7 |

Method #1 Histogram Stretching

Derive a mapping function

Slope (m) = (20-1)/(8-1) = 19/7 = 2.7

Equation for the mapping function: m = (y-1)/(x-1)

y-1 = m (x – 1)

y = m (x -1) + 1

|  |  |  |
| --- | --- | --- |
| Old value | New value | rounded |
| 1 | 1 | 1 |
| 2 | 3.7 | 4 |
| 3 | 6.4 | 6 |
| 4 | 9.1 | 9 |
| 5 | 11.8 | 12 |
| 6 | 14.5 | 15 |
| 7 | 17.2 | 17 |
| 8 | 19.9 | 20 |

The grayscale vaues of the original image will become:

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 4 | 9 | 12 |
| 17 | 17 | 20 | 4 |
| 6 | 1 | 4 | 6 |
| 12 | 9 | 15 | 17 |

Method #2 Applying Histogram Equalization

When the stretch function is not given, we use Histogram Equalization.

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 4 | 5 |
| 6 | 6 | 8 | 2 |
| 3 | 1 | 2 | 3 |
| 5 | 4 | 6 | 6 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Intensity | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| # Pixels | 1 | 3 | 3 | 2 | 2 | 1 | 3 | 1 |
| prob | 0.0625 | 0.1875 | 0.1875 | 0.125 | 0.125 | 0.0625 | 0.1875 | 0.0625 |

Total # of pixels = 4 x 4 = 16

Cumulative Probability (C.P.)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.0625 | 0.25 | 0.4375 | 0.5625 | 0.6875 | 0.75 | 0.9375 | 1 |
| C.P.\*20 | 1.25 | 5 | 8.75 | 11.25 | 13.75 | 15 | 18.75 | 20 |
| New Intensity | 1 | 5 | 8 | 11 | 13 | 15 | 18 | 20 |

The grayscale values become:

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 5 | 11 | 13 |
| 18 | 18 | 20 | 5 |
| 8 | 1 | 5 | 8 |
| 13 | 11 | 15 | 18 |

**Example #2 *Histogram Equalization***

Given an image with the following ***grayscale levels (i)*** and ***the number of pixels*** with the respective grayscale levels ***(ni*),** apply a Histogram Equalization

i = 0 1 2 3 4 5 6 7 8 9 10

ni= 10 0 0 0 0 20 20 30 10 10 0

ni 10 10 10 10 10 30 50 80 90 100 100

ni xRatio 1 1 1 1 1 3 5 8 9 10 10

Note: Ratio = grayscale levels / total number of pixels = 10/100 = 0.1



1 2 3 4 5 6 7 8 9 10

40

30

20

10



1 2 3 4 5 6 7 8 9

40

30

20

10

## CV Programming - Histograms (Taken from the OpenCV Tutorial)

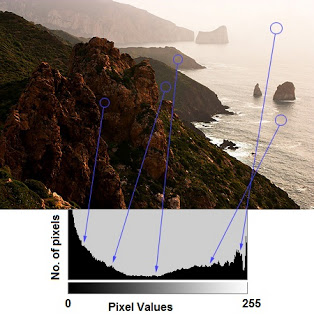
### Goals

1. Find histograms, using both OpenCV and Numpy functions
2. Plot histograms, using OpenCV and Matplotlib functions
3. You will see these functions : **cv2.calcHist()**, **np.histogram()** etc.

### Theory

So, what is a histogram? You can consider a histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

It is just another way of understanding the image. By looking at the histogram of an image, you get intuition about contrast, brightness, intensity distribution etc. of that image. Almost all image processing tools today, provides features on histogram. Below is an image from [Cambridge in Color website](http://www.cambridgeincolour.com/tutorials/histograms1.htm), and I recommend you to visit the site for more details.



You can see the image and its histogram. (Remember, this histogram is drawn for grayscale image, not color image). Left region of histogram shows the number of darker pixels in image and right region shows the number of brighter pixels. From the histogram, you can see dark region is more than brighter region, and the amount of mistunes (pixel values in mid-range, say around 127) is very few.

### Find Histogram

Now we have an idea on what is histogram, we can look into how to find this. Both OpenCV and Numpy come with in-built function for this. Before using those functions, we need to understand some terminologies related with histograms.

**BINS**: The above histogram shows the number of pixels for every pixel value, i.e. from 0 to 255, i.e. you need 256 values to show the above histogram. But consider, what if you need not find the number of pixels for all pixel values separately, but number of pixels in an interval of pixel values? say for example, you need to find the number of pixels lying between 0 to 15, then 16 to 31, ..., 240 to 255. You will need only 16 values to represent the histogram. And that is what is shown in example given in [OpenCV Tutorials on histograms](http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_calculation/histogram_calculation.html#histogram-calculation).

So, what you do is simply split the whole histogram to 16 sub-parts and value of each sub-part is the sum of all pixel count in it. This each sub-part is called “BIN”. In first case, number of bins where 256 (one for each pixel) while in second case, it is only 16. BINS is represented by the term **histSize** in OpenCV docs.

**DIMS**: It is the number of parameters for which we collect the data. In this case, we collect data regarding only one thing, intensity value. So here it is 1.

**RANGE**: It is the range of intensity values you want to measure. Normally, it is [0,256], ie all intensity values.

### Histogram Calculation in OpenCV

So now we use **cv2.calcHist()** function to find the histogram. Let’s familiarize with the function and its parameters:

**cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])**

1. images: it is the source image of type uint8 or float32. it should be given in square brackets, i.e., “[img]”.
2. channels: it is also given in square brackets. It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.
3. mask: mask image. To find histogram of full image, it is given as “None”. But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
4. histSize: this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
5. ranges: this is our RANGE. Normally, it is [0,256].

So, let’s start with a sample image. Simply load an image in grayscale mode and find its full histogram.

img = cv2.imread('home.jpg',0)

hist = cv2.calcHist([img],[0],None,[256],[0,256])

hist is a 256x1 array, each value corresponds to number of pixels in that image with its corresponding pixel value.

### Histogram Calculation in Numpy

Numpy also provides you a function, **np.histogram()**. So instead of calcHist() function, you can try below line :

hist,bins = np.histogram(img.ravel(),256,[0,256])

hist is same as we calculated before. But bins will have 257 elements, because Numpy calculates bins as 0-0.99, 1-1.99, 2-2.99 etc. So final range would be 255-255.99. To represent that, they also add 256 at end of bins. But we don’t need that 256. Up to 255 is sufficient.

**See also**

Numpy has another function, **np.bincount()** which is much faster than (around 10X) np.histogram(). So, for one-dimensional histograms, you can better try that. Don’t forget to set minlength = 256 in np.bincount. For example, hist = np.bincount(img.ravel(),minlength=256)

**Note**

OpenCV function is faster than (around 40X) than np.histogram(). So, stick with OpenCV function.

Now we should plot histograms, but how ?

### Plotting Histograms

There are two ways for this. One, the short way: use Matplotlib plotting functions. Second, the long way: use OpenCV drawing functions. We are showing both ways below.

#### 1. Using Matplotlib

Matplotlib comes with a histogram plotting function: matplotlib.pyplot.hist()

It directly finds the histogram and plot it. You need not use calcHist() or np.histogram() function to find the histogram. See the code below:

**import** **cv2**

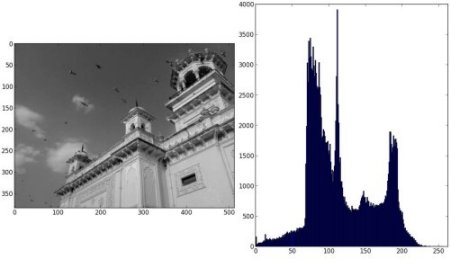
**import** **numpy** **as** **np**

**from** **matplotlib** **import** pyplot **as** plt

img = cv2.imread('home.jpg',0)

plt.hist(img.ravel(),256,[0,256]); plt.show()

You will get a plot as below:



Or you can use normal plot of matplotlib, which would be good for BGR plot. For that, you need to find the histogram data first. Try below code:

**import** **cv2**

**import** **numpy** **as** **np**

**from** **matplotlib** **import** pyplot **as** plt

img = cv2.imread('home.jpg')

color = ('b','g','r')

**for** i,col **in** enumerate(color):

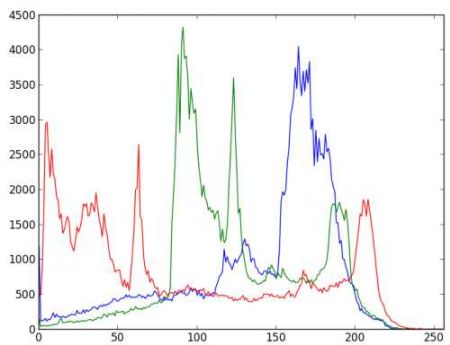
histr = cv2.calcHist([img],[i],None,[256],[0,256])

plt.plot(histr,color = col)

plt.xlim([0,256])

plt.show()

Result:



You can deduct from the above graph that blue has some high value areas in the image (obviously it should be due to the sky)

#### 2. Using OpenCV

Well, here you adjust the values of histograms along with its bin values to look like x,y coordinates so that you can draw it using cv2.line() or cv2.polyline() function to generate same image as above. This is already available with OpenCV-Python2 official samples. [Check the Code](https://github.com/Itseez/opencv/raw/master/samples/python2/hist.py)

### Application of Mask

We used cv2.calcHist() to find the histogram of the full image. What if you want to find histograms of some regions of an image? Just create a mask image with white color on the region you want to find histogram and black otherwise. Then pass this as the mask.

img = cv2.imread('home.jpg',0)

*# create a mask*

mask = np.zeros(img.shape[:2], np.uint8)

mask[100:300, 100:400] = 255

masked\_img = cv2.bitwise\_and(img,img,mask = mask)

*# Calculate histogram with mask and without mask*

*# Check third argument for mask*

hist\_full = cv2.calcHist([img],[0],None,[256],[0,256])

hist\_mask = cv2.calcHist([img],[0],mask,[256],[0,256])

plt.subplot(221), plt.imshow(img, 'gray')

plt.subplot(222), plt.imshow(mask,'gray')

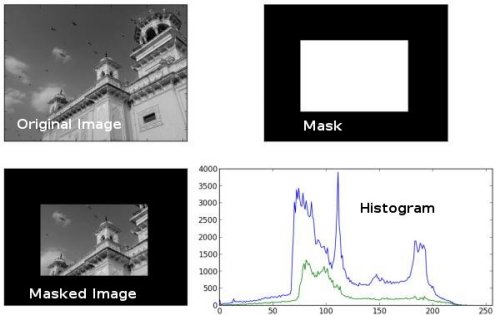
plt.subplot(223), plt.imshow(masked\_img, 'gray')

plt.subplot(224), plt.plot(hist\_full), plt.plot(hist\_mask)

plt.xlim([0,256])

plt.show()

See the result. In the histogram plot, blue line shows histogram of full image while green line shows histogram of masked region.



### Histogram Equalization

We use the following example taken from our textbook to demonstrate how to do histogram equalization.

import skimage.io as io

import skimage.exposure as ex

import pylab

from matplotlib import pyplot as plt

c = io.imread('chickens.png')

#io.imshow(c)

ch = ex.equalize\_hist(c)

io.imshow(ch)

f = pylab.figure(); f.show(plt.hist(ch.flatten(), bins=256))

﻿

The histogram after equalizetion is completed.



Note: Running the program using Spyder version 3.3.0, Python 3.7, and OpenCV version 3.4.2, we obtain the image on an iPython console. To find your OpenCV version, you may use the following python command after entering “import cv2”:

>>>import cv2

>>>cv2.\_\_version\_\_

The versions of Python, iPython, and OpenCV are critical. You may use the python statement to find the version of matplotlib:

>>> import matplotlib

>>> matplotlib.\_\_varsion\_\_

>>> 3.2.2