ET3112 Image Processing and Machine Vision: Point Operations

Ranga Rodrigo
ranga@uom.lk

The University of Moratuwa, Sri Lanka

January 2, 2020

Contents

Section 1

Basics

What is a Digital Image?

- A grayscale digital image is a rectangular array of numbers which represent pixels.
- Each pixel can take an integer value in the range $[0, 255]$, for an 8-bit image.
- If the image is a color image, then there would be three such arrays.
- The size of this array is actually the resolution of the image, e.g., example, 3048×2248.
- We take the top-left pixel as the $(0, 0)$ pixel and vertical axis as the i axis.

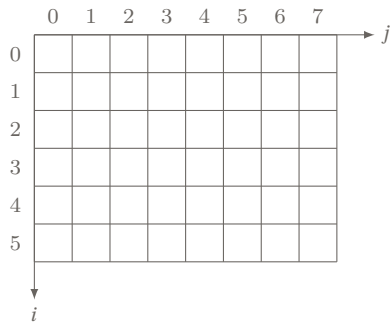


Figure: A 6×8 image

What is a Digital Image?

- A grayscale digital image is a rectangular array of numbers which represent pixels.
- Each pixel can take an integer value in the range $[0, 255]$, for an 8-bit image.
- If the image is a color image, then there would be three such arrays.
- The size of this array is actually the resolution of the image, e.g., example, 3048×2248.
- We take the top-left pixel as the $(0, 0)$ pixel and vertical axis as the i axis.

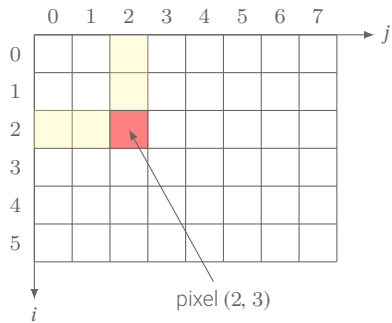


Figure: A 6×8 image

What is a Color Digital Image?

- The grayscale image that we considered s a two-dimensional array, or a single plane.
- A color image has three such planes, one for blue, one for green and one for red. We call such an image an BGR image.
- If we access a pixel location such as $(2, 3)$, we will get three values, B, G, and R.
- Each value is in $[0, 255]$. In this context, $2^8 \times 2^8 \times 2^8$ different colors are possible.

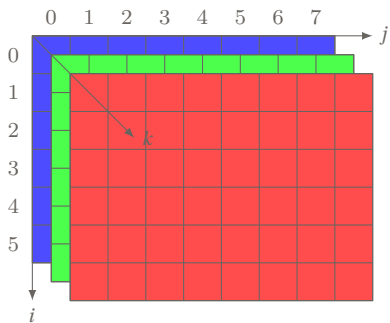


Figure: A 6×8 BGR image

Code for Generating a 6×8 Images

Listing 1: A Grayscale Image

```
%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
im = np.zeros((6,8),dtype=np.uint8)
im[2,3] = 255
fig, ax = plt.subplots()
ax.imshow(im, cmap = 'gray')
ax.xaxis.tick_top()
plt.show()
```

Listing 2: A Color Image

```
%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
im = np.zeros((6,8, 3),dtype=np.uint8)
im[2,3] = [255, 255, 100]
print(im[2,3])
fig, ax = plt.subplots()
ax.imshow(im, cmap = 'gray')
ax.xaxis.tick_top()
plt.show()
```

Opening and Displaying Images

1. We can use OpenCV to open and display images. We can examine the size (resolution) of the image as well.

Listing 3: Displaying Using Matplotlib

```
%matplotlib inline
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread('../images/gal.jpg',
      cv.IMREAD_COLOR)
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
fig, ax = plt.subplots()
ax.imshow(img)
ax.set_title('Image')
plt.show()
```

Listing 4: Displaying Using OpenCV

```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread('../images/katniss.
      jpg', cv.IMREAD_COLOR)
cv.namedWindow('Image', cv.WINDOW_NORMAL)
cv.imshow('Image', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Q:Why do we need to `cv.cvtColor` only when displaying using Matplotlib?

Displaying Image Properties

Listing 5: Displaying Image Properties

```
%matplotlib inline
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread('../images/hugh.jpg', cv.IMREAD_COLOR)
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
fig, ax = plt.subplots()
ax.imshow(img)
ax.set_title('Image')
plt.show()
print(img.shape)
print(img.size)
print(img.dtype)
```

Increasing the Brightness Using OpenCV

Listing 6: Increasing the Brightness Using OpenCV

```
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread('../images/keira.jpg', cv.IMREAD_GRAYSCALE)
imgb = img + 100
imgc = cv.add(img, 100)
f, ax = plt.subplots(1,3)
ax[0].imshow(img, cmap="gray")
ax[0].set_title('Original')
ax[1].imshow(imgb, cmap="gray")
ax[1].set_title('img + 100')
ax[2].imshow(imgc, cmap="gray")
ax[2].set_title('cv.add')
```

- What is the reason for the `img + 100` operation to be incorrect?

Increasing the Brightness Using Loops I

Listing 7: Increasing the Brightness Using Loops

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
def image_brighten(image, shift):
        h = image.shape[0]
        w = image.shape[1]
        result = np.zeros(image.shape, image.dtype)
        for i in range(0,h):
                for j in range(0,w):
                        no_overflow = True if image[i,j] + shift < 255 else False
                        result[i,j] = min(image[i,j] + shift, 255) if no_overflow else 255
        return result

img = cv.imread('../images/keira.jpg', cv.IMREAD_GRAYSCALE)
%timeit imgb = image_brighten(img, 200)
f, ax = plt.subplots(1,2)
ax[0].imshow(img, cmap = 'gray')
ax[0].set_title('Original')
```

Increasing the Brightness Using Loops II

```
19 ax[1].imshow(imgb, cmap = 'gray')
20 ax[1].set_title('image_brighten')
```

- Do not use this, due to inefficiencies. Instead, use OpenCV filters or Cython (?).
- In the convolution operations—which we would cover later—to produce each output pixel, we need to multiply and add, say, a 3×3 neighborhood of pixels. How many loops would this require? What is the computational complexity?

Zeroing Out Green and Blue Planes

Listing 8: Zeroing Out Green and Blue Planes

```
%matplotlib inline
import matplotlib.pyplot as plt
import cv2 as cv
img = cv.imread('images/tom.jpg', cv.IMREAD_ANYCOLOR)
if img is None:
    print('Image could not be read.')
    assert False
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
plt.title('Image')
plt.xticks([]), plt.yticks([])
plt.show()
img[:,:,1:3] = 0
plt.imshow(img)
plt.title('After Zeroing Planes')
plt.xticks([]), plt.yticks([])
plt.show()
```

Summary

1. A grayscale digital image is an array of 8-bit unsigned integers in $[0, 255]$. We call each integer a pixel.
2. Color images have three such arrays (planes), one for red, one for green, and one for blue.
3. We can manipulate an image using OpenCV function or a pair of for-loops to access each pixel (slower).

Section 2

Intensity Transformations

Intensity Transformations (Point Operations): Introduction

- In intensity transformations, the output value of the pixel depends only on the input values of that pixel, not it neighbors.
- Input image: $f(\mathrm{x})$
  Output image: $g(\mathrm{x})$
  Intensity transform $g(\mathrm{x}) = T(f(\mathrm{x}))$
- E.g., identity transform $T(.) = 1$

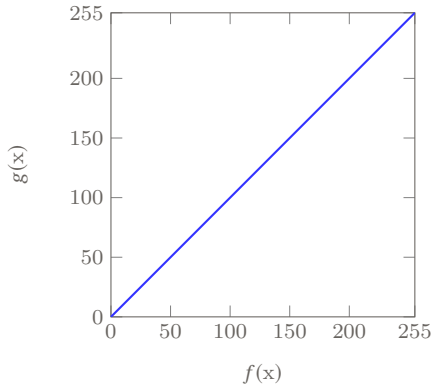


Figure: Identity transform

Intensity Transforms: Identity Transformation Code I

Listing 9: Identity Transformation

```
%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
transform = np.arange(0, 256).astype('uint8')
fig , ax = plt.subplots()
ax.plot(transform)
ax.set_xlabel(r'Input , $f(\mathbf{x})$')
ax.set_ylabel('Output , $\mathrm{T}[f(\mathbf{x})]$')
ax.set_xlim(0,255)
ax.set_ylim(0,255)
ax.set_aspect('equal')
plt.savefig('transform.png')
plt.show()
img_orig = cv.imread('../images/katrina.jpg', cv.IMREAD_GRAYSCALE)
print(img_orig.shape)
cv.namedWindow('Image', cv.WINDOW_AUTOSIZE)
cv.imshow('Image', img_orig)
```

Intensity Transforms: Identity Transformation Code II

```
19 cv.waitKey(0)
20 image_transformed = cv.LUT(img_orig, transform)
21 cv.imshow('Image', image_transformed)
22 cv.waitKey(0)
23 cv.destroyAllWindows()
```

- Explain the line image_transformed = cv.LUT(img_orig, transform).
- This can be done using NumPy only as image_transformed = transform[img_orig]. Explain this operation.
- What is $T()$ here?

What Is This Transformation?

- $g(x) = -f(x)$

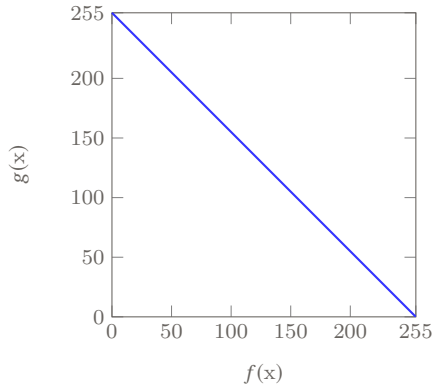


Figure: Negative transform

What Is This Transformation?

- $g(x) = -f(x)$
- Implementation:
  transform = np.arange(255,–1, –1).astype('uint8').

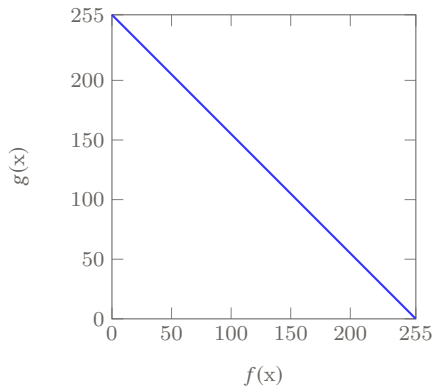


Figure: Negative transform

Intensity Transforms: Intensity Windowing I

Listing 10: Intensity Windowing

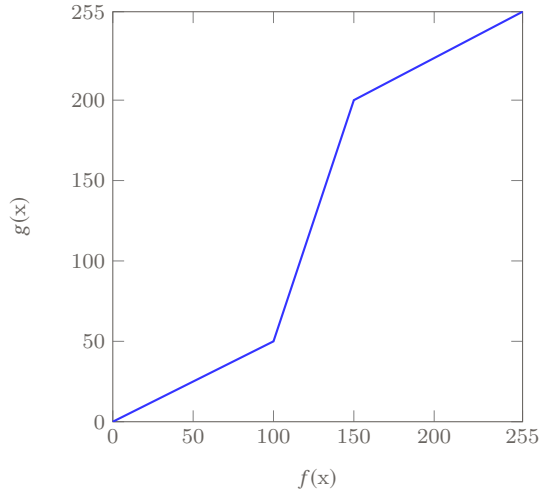
```
%matplotlib inline
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
c = np.array([(100, 50), (150, 200)])
t1 = np.linspace(0, c[0,1], c[0,0] + 1 - 0).astype('uint8')
print(len(t1))
t2 = np.linspace(c[0,1] + 1, c[1,1], c[1,0] - c[0,0]).astype('uint8')
print(len(t2))
t3 = np.linspace(c[1,1] + 1, 255, 255 - c[1,0]).astype('uint8')
print(len(t3))
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
transform = np.concatenate((transform, t3), axis=0).astype('uint8')
print(len(transform))
fig, ax = plt.subplots()
ax.plot(transform)
ax.set_xlabel(r'Input, $f(\mathbf{x})$')
ax.set_ylabel('Output, $\mathrm{T}[f(\mathbf{x})]$')
```

Intensity Transforms: Intensity Windowing II

```
19 ax.set_xlim(0,255)
20 ax.set_ylim(0,255)
21 ax.set_aspect('equal')
22 plt.savefig('transform.png')
23 plt.show()
24 img_orig = cv.imread('../images/katrina.jpg', cv.IMREAD_GRAYSCALE)
25 cv.namedWindow("Image", cv.WINDOW_AUTOSIZE)
26 cv.imshow("Image", img_orig)
27 cv.waitKey(0)
28 image_transformed = cv.LUT(img_orig, transform)
29 cv.imshow("Image", image_transformed)
30 cv.waitKey(0)
31 cv.destroyAllWindows()
```

- What is $T()$ here?

Intensity Transforms: Intensity Windowing III



Intensity Transforms: Intensity Windowing IV

Figure: Intensity windowing.

Histograms

1. We can represent the intensity distribution over the range of intensities $[0, 255]$, using a histogram.
2. If h is the histogram of a particular image, $h(r_k)$ gives us how many pixels have the intensity r_k. The histogram of a digital image with gray values in the range $[0, L-1]$ is a discrete function $h(r_k) = n_k$ where r_k is the kth gray level and n_k is the number of pixels having gray level r_k.
3. We can normalize the histogram by dividing by the total number of pixels n. Then we have an estimate of the probability of occurrence of level r_k, i.e., $p(r_k) = n_k/n$.
4. The histogram that we described above has L bins. We can construct a coarser histogram by selecting a smaller number of bins than L. Then several adjacent values of k will be counted for a bin.

Activity

1. The figure shows a 3×4 image. The range of intensities that this image has is $[0, 7]$. Compute its histogram.

6	5	5	3
7	6	6	4
2	3	5	4

Figure: Image for histogram computation.

Histograms Using OpenCV

Listing 11: Histogram of a Grayscale Image

```
1 import cv2 as cv
2 import numpy as np
3 import matplotlib.pyplot as plt
4 img = cv.imread('../images/gal.jpg',
      cv.IMREAD_GRAYSCALE)
5 hist = cv.calcHist([img], [0], None, [256], [0,256])
6 plt.plot(hist)
7 plt.xlim([0,256])
8 plt.show()
```

Listing 12: Histogram of a Color Image

```
1 import cv2 as cv
2 import numpy as np
3 import matplotlib.pyplot as plt
4 img = cv.imread('../images/gal.jpg',
      cv.IMREAD_COLOR)
5 color = ('b', 'g', 'r')
6 for i, c in enumerate(color):
7  hist = cv.calcHist([img], [i], None, [256], [0,256])
8       plt.plot(hist, color = c)
9       plt.xlim([0,256])
10 plt.show()
```

Image Properties through Histograms

1. If the image is dark histogram will have many values in the left region, that correspond to dark pixels.
2. If the image is bright histogram will have many values in the right region, that correspond to bright pixels.
3. If a significant number of pixels are dark and a significant number of pixels a bright, the histogram will have two modes, one in the left region and the other in the right region.
4. A flat histogram signifies that the image has a uniform distribution of all intensities. Then, the contrast is high and image will look vibrant.

Histogram Equalization

1. Photographers like to shoot pictures with a flat histogram, as such pictures are vibrant.
2. Histogram equalization is a gray-level transformation that results in an image with a more or less flat histogram.
3. We can take and image and make its histogram flat by using the operation called histogram equalization.

Consider, for now, that continuous intensity values of an image are to be processed. We assume that $r \in [0, L-1]$. Lets consider the intensity transformation

$$s = T(r) \quad 0 \leq r \leq L-1 \tag{1}$$

that produces an output intensity level s for every pixel in the input image having intensity r. We assume that

- $T(r)$ is monotonically increasing in the interval $0 \leq r \leq L-1$, and
- $0 \leq T(r) \leq L-1$ for $0 \leq r \leq L-1$.

The intensity levels in the image may be viewed as random variables in the interval $[0, L-1]$. Let $p_r(r)$ and $p_s(s)$ denote the probability density functions (PDFs) of r and s. A fundamental result from basic probability theory is that if $p_r(r)$ and $T(r)$ are known, and $T(r)$ is continuous and differentiable over the range of values of interest, then the PDF of the transformed variable s can be obtained using the simple formula

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right|. \tag{2}$$

Now let's consider the following transform function:

$$s = T(r) = (L-1)\int_0^r p_r(w)dw, \tag{3}$$

where w is the dummy variable of integration. The right-hand side of this equation is the cumulative distribution function (CDF) of random variable r. This function satisfies the two conditions that we mentioned.

- Because PDFs are always positive and the integral of a function is the area under the curve, Equation 3 satisfies the first condition. This is because the area under the curve cannot decrease as r increases.
- When the upper limit in this equation is $r = L-1$, the integral evaluates to 1, because the area under a PDF curve is always 1. So the maximum value of s is $L-1$, and the second condition is also satisfied.

To find $p_s(s)$ corresponding to this transformation we use Equation 1.

$$\begin{aligned} \frac{ds}{dr} &= \frac{dT(r)}{dr}, \\ &= (L-1)\frac{d}{dr}\left[\int_0^r p_r(w)dw\right], \\ &= (L-1)p_r(r). \end{aligned} \tag{4}$$

Substituting this result in Equation 1, and keeping in mind that all probability values are positive, yields

$$\begin{aligned} p_s(s) &= p_r(r)\left|\frac{dr}{ds}\right|, \\ &= p_r(r)\left|\frac{1}{(L-1)p_r(r)}\right|, \\ &= \frac{1}{L-1} \quad 0 \le s \le L-1. \end{aligned} \tag{5}$$

We recognize the form of $p_s(s)$ in the last line of this equation as a uniform probability density function.

For discrete values, we deal with probabilities (histogram values) and the summation instead of probability density functions and integrals. The probability of occurrence of intensity level r_k in a digital image is approximated by

$$p_r(r_k) = \frac{n_k}{MN} \quad k = 0, 1, \ldots, L-1, \tag{6}$$

where MN is the total number of pixels in the image, n_k is the number of pixels that have intensity r_k, and L is the number of possible intensity levels in the image (e.g., 256). Recall that a plot of $p_r(r_k)$ versus r_k is commonly referred to as a histogram.
The discrete form of the Equation 2 is

$$\begin{aligned} s_k &= T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j), \\ &= \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \quad k = 0, 1, \ldots, L-1. \end{aligned} \tag{7}$$

Thus the output image is obtained by mapping each pixel in the input image with intensity r_k into a corresponding pixel level s_k in the output image using Equation 7.

Example

Suppose that a 3-bit image ($L = 8$) of size 64×64 pixels ($MN = 4096$) has the intensity distribution shown in Table 1, where the intensity levels are in the range $[0, L-1] = [0, 7]$. carry out histogram equalization.

r_k	n_k	$p_r(r_k) = n_k/MN$	$\sum_{j=0}^{k} n_j$	$\frac{(L-1)}{MN}\sum_{j=0}^{k} n_j$	Rounded
$r_0 = 0$	790	0.19			
$r_1 = 1$	1023	0.25			
$r_2 = 2$	850	0.21			
$r_3 = 3$	656	0.16			
$r_4 = 4$	329	0.08			
$r_5 = 5$	245	0.06			
$r_6 = 6$	122	0.03			
$r_7 = 7$	81	0.02			

Table: Table for Example 1.

Histogram Equalization I

Listing 13: Histogram Equalization

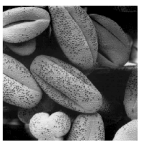
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('../images/shells.tif',cv.IMREAD_GRAYSCALE)

hist,bins = np.histogram(img.ravel(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * hist.max()/ cdf.max()
plt.plot(cdf_normalized, color = 'b')
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.title('Histogram of the Original Image')
plt.show()
equ = cv.equalizeHist(img)
hist,bins = np.histogram(equ.ravel(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * hist.max()/ cdf.max()
```

Histogram Equalization II

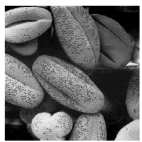
```
19 plt.plot(cdf_normalized, color = 'b')
20 plt.hist(equ.flatten(),256,[0,256], color = 'r')
21 plt.xlim([0,256])
22 plt.legend(('cdf','histogram'), loc = 'upper left')
23 plt.title('Histogram of the Equalized Image')
24 plt.show()
25 res = np.hstack((img,equ))
26 plt.axis('off')
27 plt.imshow(res, cmap='gray')
```

Histogram Equalization



(a) Original image

Histogram Equalization



(a) Original image

(b) Histogram-equalized image

Figure: Histogram Equalization

Power-Law Transformation (Gamma Correction)

$$g = f^{\gamma}, \quad f \in [0, 1].$$

Values of γ such that $0 < \gamma < 1$ map a narrow range of dark pixels to a wider range of dark pixels.
$\gamma > 0$ has the opposite effect.
$\gamma = 1$ gives the identity transform.

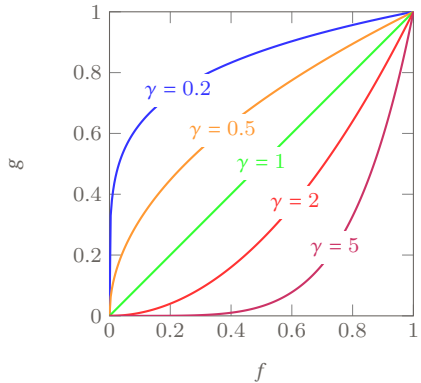


Figure: Plots of f^{γ}

Gamma Correction I

Listing 14: Gamma Correction

```
%matplotlib inline
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
img_orig = cv.imread('../images/gal.jpg', cv.IMREAD_COLOR)
gamma = 2
table = np.array([(i/255.0)**(1/gamma)*255.0 for i in np.arange(0,256)]).astype('uint8')
img_gamma = cv.LUT(img_orig, table)
img_orig = cv.cvtColor(img_orig, cv.COLOR_BGR2RGB)
img_gamma = cv.cvtColor(img_gamma, cv.COLOR_BGR2RGB)
f, axarr = plt.subplots(3,2)
axarr[0,0].imshow(img_orig)
axarr[0,1].imshow(img_gamma)
color = ('b', 'g', 'r')
for i, c in enumerate(color):
    hist_orig = cv.calcHist([img_orig], [i], None, [256], [0,256])
    axarr[1,0].plot(hist_orig, color = c)
    hist_gamma = cv.calcHist([img_gamma], [i], None, [256], [0,256])
```

Gamma Correction II

```
19        axarr[1,1].plot(hist_gamma, color = c)
20    axarr[2,0].plot(table)
21    axarr[2,0].set_xlim(0,255)
22    axarr[2,0].set_ylim(0,255)
23    axarr[2,0].set_aspect('equal')
```

Summary

1. Basics: digital image, color images, creating an image in Python, displaying images, increasing brightness, image planes
2. Intensity transformations:
   - 2.1 Identity, negative, intensity windowing
   - 2.2 Histograms, histogram equalization
   - 2.3 Gamma correction