شماره فصل: ۶	تعداد جلسات مورد نیاز: ۳ جلسه
اهداف: – آشنایی با تبدیلات شدت در تصویر (رسم هیستوگرام تصویر، اصلاح تصویر با تعدیل هیستوگرام، ...)	

رسم هیستوگرام تصویر سطح خاکستری:

```
## Display histogram of a grayscale image##
import cv2
from matplotlib import pyplot as plt # or use: import matplotlib.pyplot as
plt
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Convert BGR image to
grayscale
histogram = cv2.calcHist([gray_img], [0], None, [256], [0, 256]) #Compute a
grayscale histogram suing cv2.calcHist(image, image channel number, mask
for ROI, number of bins, Range of bins)
# matplotlib expects RGB images so convert and then display the image with
matplotlib
plt.figure()
plt.title("Grayscale image")
plt.axis("off")
plt.imshow(cv2.cvtColor(gray_img, cv2.COLOR_GRAY2RGB))
#Plot the histogram
plt.figure()
plt.title("Grayscale histogram")
plt.xlabel("Bins")
plt.ylabel("Number of pixels")
plt.plot(histogram, color='k')
plt.xlim([0, 256])
plt.show()
####
```

رسم هیستوگرام‌های تصویر رنگی:

```
## Display histograms of a color image##
import cv2
from matplotlib import pyplot as plt # or use: import matplotlib.pyplot as
plt
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
Blue_channel = img[:,:,0]
Green_channel = img[:,:,1]
Red_channel = img[:,:,2]
histogram_B = cv2.calcHist([Blue_channel], [0], None, [256], [0, 256])
#Histogram of blue channel
histogram_G = cv2.calcHist([Green_channel], [0], None, [256], [0, 256])
#Histogram of green channel
histogram_R = cv2.calcHist([Red_channel], [0], None, [256], [0, 256])
#Histogram of red channel
#Plot the color (RGB) image
plt.figure()
```

```
plt.title("Color (RGB) image")
plt.axis("off")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
#Plot the RGB histograms
plt.figure()
plt.title("Color (RGB) histogram")
plt.xlabel("Bins")
plt.ylabel("Number of pixels")
plt.plot(histogram_B, color='b')
plt.plot(histogram_G, color='g')
plt.plot(histogram_R, color='r')
plt.xlim([0, 256])
plt.show()
####
```

اصلاح تصویر توسط روش تعدیل هیستوگرام:

```
## Histogram equalization##
import cv2
import numpy as np
from matplotlib import pyplot as plt # or use: import matplotlib.pyplot as
plt
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
#Change image intensity by power law transformation [g(x,y)=f(x,y)^gamma]
gamma=0.3
modified_img=np.array(255*(img/255)**gamma,dtype='uint8')
cv2.imshow("Original image (left) and Modified image
(right)",np.hstack((img,modified_img)))
gray_img = cv2.cvtColor(modified_img, cv2.COLOR_BGR2GRAY)
#Equalize image using histogram equalization method
equlized_img = cv2.equalizeHist(gray_img)
cv2.imshow("Grayscale image (left) and Equalized image
(right)",np.hstack((gray_img,equlized_img)))
#Calculate images histogram
histogram_gray_img = cv2.calcHist([gray_img], [0], None, [256], [0, 256])
#Histogram of original image
histogram_equ = cv2.calcHist([equlized_img], [0], None, [256], [0, 256])
#Histogram of image after histogram equalization
#Plot the histogram
plt.figure()
plt.title("Grayscale histogram")
plt.xlabel("Bins")
plt.ylabel("Number of pixels")
plt.plot(histogram_gray_img, color='k')
plt.plot(histogram_equ, color='b')
plt.xlim([0, 256])
plt.show()
###
```

آستانه گذاری تصویر:

```
## Image thresholding ##
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray_img,127,255,cv2.THRESH_BINARY)
```

```
titles = ['Grayscale Image','Binary image (thresholding)']
images = [gray_img, thresh]
for i in range(2):
    plt.subplot(1,2,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
###
```

تبدیل سطح شدت با استفاده از توابع تبدیل:

```
## Gray level intensity transformation using basic transformation functions
##
import cv2
import numpy as np
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#Negative image
negative_img = abs(255-gray_img) #Alternatively you can use negative_img =
cv2.bitwise_not(gray_img)
cv2.imshow("Grayscale image (left) and Negative image
(right)",np.hstack((gray_img,negative_img)))
#Logarithmic transformation [g(x,y)=c*log(f(x,y)+1)]
c = 255 / np.log(1 + np.max(gray_img))
log_img = c*(np.log(gray_img + 1))
log_img = np.array(log_img, dtype = np.uint8) #Float numbers are converted
to uint8
cv2.imshow("Original image (left) and Logarithmic image
(right)",np.hstack((gray_img,log_img)))
#Inverse-log (exponential) transformation [g(x,y)=e^(f(x,y)^1/c)-1]
c = 255 / np.log(1 + np.max(gray_img))
invlog_img = np.exp(gray_img**1/c)-1
invlog_img = np.array(invlog_img, dtype = np.uint8)
cv2.imshow("Original image (left) and Inverse-logarithmic image
(right)",np.hstack((gray_img,invlog_img)))
#Power law transformation [g(x,y)=f(x,y)^gamma]
gamma=0.3
PoweLaw_img=np.array(255*(gray_img/255)**gamma,dtype='uint8')
cv2.imshow("Original image (left) and Power law image
(right)",np.hstack((gray_img,PoweLaw_img)))
cv2.waitKey(0)
###
```

تبدیل سطح شدت با تبدیل خطی تکه‌ای (Piecewise linear transformation):

```
## Piecewise linear transformation ##
import cv2
import numpy as np
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#The locations of (r1, s1) and (r2, s2) control the shape of transformation
function.
def Contrast_stretch(p, r1, s1, r2, s2):
    if (0 <= p and p <= r1):
        equation = (s1 / r1)*p
    elif (r1 < p and p <= r2):
        equation = ((s2 - s1)/(r2 - r1))*(p - r1)+s1
    else:
        equation = ((255 - s2)/(255 - r2))*(p - r2)+s2
```

```
    return equation
r1 = 55
s1 = 10
r2 = 140
s2 = 220
pixelVal_vec = np.vectorize(Contrast_stretch)
piecewise_img = pixelVal_vec(gray_img, r1, s1, r2, s2)
piecewise_img = np.array(piecewise_img, dtype = np.uint8)
cv2.imshow("Original image (left) and Piecewise linear transformation image
(right)",np.hstack((gray_img,piecewise_img)))
cv2.waitKey(0)
###
```

برش سطح خاکستری (سطح شدت):

```
## Gray level slicing ##
import cv2
import numpy as np
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
row, column= gray_img.shape
sliced_img = np.zeros((row,column),dtype = 'uint8') #Create a zeros array
to store the sliced image
# Specify the min and max range for slicing
min_range = 70
max_range = 190
# Loop over the gray scale image and if pixel value lies in desired range
set it to 255 otherwise set it to desired value
for i in range(row):
    for j in range(column):
        if gray_img[i,j]>min_range and gray_img[i,j]<max_range:
            sliced_img[i,j] = 255
        else:
            sliced_img[i,j] = gray_img[i,j]
cv2.imshow("Original image (left) and Gray level slicing image
(right)",np.hstack((gray_img,sliced_img)))
cv2.waitKey(0)
```

برش سطح بیت:

```
## Bit plane slicing ##
import cv2
import numpy as np
from matplotlib import pyplot as plt # or use: import matplotlib.pyplot as
plt
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
row, column= gray_img.shape
# Iterate over each pixel and change pixel value to binary using
np.binary_repr() and store it in a list.
binary_pixel_list = []
for i in range(row):
    for j in range(column):
        binary_pixel_list.append(np.binary_repr(gray_img[i][j], width=8))
# width = no. of bits
# We have a list of strings where each string represents binary pixel
value. To extract bit planes we need to iterate over the strings and store
the characters corresponding to bit planes into lists.
```

```
# Multiply with (2^bit number) and reshape to reconstruct the bit image.
bit0_img = (np.array([int(pixel[7]) for pixel in binary_pixel_list],
dtype=np.uint8) * 1).reshape(row, column)
bit1_img = (np.array([int(pixel[6]) for pixel in binary_pixel_list],
dtype=np.uint8) * 2).reshape(row, column)
bit2_img = (np.array([int(pixel[5]) for pixel in binary_pixel_list],
dtype=np.uint8) * 4).reshape(row, column)
bit3_img = (np.array([int(pixel[4]) for pixel in binary_pixel_list],
dtype=np.uint8) * 8).reshape(row, column)
bit4_img = (np.array([int(pixel[3]) for pixel in binary_pixel_list],
dtype=np.uint8) * 16).reshape(row, column)
bit5_img = (np.array([int(pixel[2]) for pixel in binary_pixel_list],
dtype=np.uint8) * 32).reshape(row, column)
bit6_img = (np.array([int(pixel[1]) for pixel in binary_pixel_list],
dtype=np.uint8) * 64).reshape(row, column)
bit7_img = (np.array([int(pixel[0]) for pixel in binary_pixel_list],
dtype=np.uint8) * 128).reshape(row, column)
titles = ['Bit 0','Bit 1','Bit 2','Bit 3','Bit 4','Bit 5','Bit 6','Bit 7']
images =
[bit0_img,bit1_img,bit2_img,bit3_img,bit4_img,bit5_img,bit6_img,bit7_img]
for i in range(8):
    plt.subplot(2,4,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
# Combining bit planes
Bit6_7=bit6_img+bit7_img
Bit5_6_7=bit5_img+bit6_img+bit7_img
Bit4_5_6_7=bit4_img+bit5_img+bit6_img+bit7_img
Bit3_4_5_6_7=bit3_img+bit4_img+bit5_img+bit6_img+bit7_img
titles = ['Grayscale image','Bit 7','Bits 6, and 7','Bits 5, 6, and
7','Bits 4, 5, 6, and 7','Bits 3, 4, 5, 6, and 7']
images = [gray_img,bit7_img,Bit6_7,Bit5_6_7,Bit4_5_6_7,Bit3_4_5_6_7]
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
##
```

صاف کردن (مات کردن) تصویر با استفاده از فیلترهای مکانی:

```
## Image smoothing (blurring) using spatial filters ##
import cv2
import numpy as np
from matplotlib import pyplot as plt # or use: import matplotlib.pyplot as
plt
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
# A function to generate different types of noises
def noisy(noise_typ,image):
   if noise_typ == "gauss":
      row,col,ch= image.shape
      mean = 0
      var = 0.1
      sigma = var**0.5
      gauss = np.random.normal(mean,sigma,(row,col,ch))
      gauss = gauss.reshape(row,col,ch)
      noisy = image + gauss
      return noisy
   elif noise_typ == "s&p":
```

```
        row,col,ch = image.shape
        s_vs_p = 0.5
        amount = 0.004
        out = np.copy(image)
        # Salt mode
        num_salt = np.ceil(amount * image.size * s_vs_p)
        coords = [np.random.randint(0, i - 1, int(num_salt))
                for i in image.shape]
        out[coords] = 1
        # Pepper mode
        num_pepper = np.ceil(amount* image.size * (1. - s_vs_p))
        coords = [np.random.randint(0, i - 1, int(num_pepper))
                for i in image.shape]
        out[coords] = 0
        return out
    elif noise_typ == "poisson":
        vals = len(np.unique(image))
        vals = 2 ** np.ceil(np.log2(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy
    elif noise_typ =="speckle":
        row,col,ch = image.shape
        gauss = np.random.randn(row,col,ch)
        gauss = gauss.reshape(row,col,ch)
        noisy = image + image * gauss
    return noisy
#Generate some noises on image using "noisy" function
noisy_img=noisy("s&p",img)
#Smoothing by simple averaging filters
kernel = np.ones((3,3),np.float32)/9
Blur_average = cv2.filter2D(noisy_img,-1,kernel) #Alternatively one can
use: Blur_simple = cv2.blur(img,(3,3))
#Smoothing by median filter
Blur_median = cv2.medianBlur(noisy_img,3)
#Smoothing by Gaussian function
Blur_gaussian =cv2.GaussianBlur(noisy_img, (3, 3), 0)
titles = ['Original image','Smoothed image by average filter','Smoothed
image by median filter','Smoothed image by Gaussian function']
images = [noisy_img,Blur_average,Blur_median,Blur_gaussian]
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(cv2.cvtColor(images[i],
cv2.COLOR_BGR2RGB))
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
###
```

لبه یابی و تیز کردن تصویر با استفاده از فیلترهای تفاضلی (مشتقات):

```
## Edge detection and image sharpening by spatial differentiation
(derivatives) filters ##
import cv2
import numpy as np
from matplotlib import pyplot as plt # or use: import matplotlib.pyplot as
plt
img=cv2.imread('E:/UOK/UOK_ Presentations/UOK_Digital Image
Processing/Python/Images/Fruits.jpeg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Implementation of Sobel on image in x direction
Sobel_x = cv2.Sobel(gray_img,cv2.CV_64F,1,0,ksize=3)
```

```
# Implementation of Sobel on image in y direction
Sobel_y = cv2.Sobel(gray_img,cv2.CV_64F,0,1,ksize=3)
# Sobel x and y direction
Sobel=Sobel_x+Sobel_y
# Calculation of Laplacian
Laplacian = cv2.Laplacian(gray_img,cv2.CV_64F)
# Sharpened image
Sharp_img=gray_img-Laplacian
titles = ['Grayscale image','Edges using Sobel in x direction','Edges using
Sobel in y direction','Edges using Sobel','Edges using
Laplacian','Sharpened image']
images = [gray_img,Sobel_x,Sobel_y,Sobel,Laplacian,Sharp_img]
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i], 'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
###
```