HW1

Houlin Li

October 15, 2022

Exercise 1

We know High-definition television (HDTV) generates images with 1125 horizontal TV lines. And we know that the resolution of each TV (horizontal) line in their system is in proportion to vertical resolution, with the proportion being the width- to-height ratio of the images. So the number of vertical TV lines is :

$$V = 1125 \times \frac{16}{9} = 2000 \tag{1.1}$$

Then, we can conclude that the color image is a $1125 \times 2000 \times 24$ matrix. Because every other line is painted on the tube face in each of two fields, each field being 1>60th of a second in duration, so 30 complete color images can be generated every second. After 2 hours, bits are:

$$2 \times 3600 \times 30 \times H \times V \times 24 = 1.1664 \times 10^{13} \tag{1.2}$$

Exercise 2

from the definitions of 4- adjacent, 8-adjacent and m-adjacent. We easily know the table below:

adjacentimage	$S1$	$S2$
4-adjacent	✓	×
8-adjacent	✓	✓
m-adjacent	✓	✓

Exercise 3

We can easily found that the transform can be negative transformation which is:

$$P_r(r) = 2(1-r), \quad 0 \le r \le 1 \tag{3.1}$$

$$P_z(z) = 2z, \quad 0 \le z \le 1 \tag{3.2}$$

let:

$$
\begin{aligned}
S &= T(r) \\
&= \int_0^r P_r(w)dw \\
&= \int_0^r 2(1-w)dw \\
&= 2r - r^2
\end{aligned} \tag{3.3}
$$

$$
\begin{aligned}
G(z) &= \int_0^z P_z(w)dw \\
&= z^2 \\
&= S
\end{aligned} \tag{3.4}
$$

So, we have:

$$
z = \sqrt{r(2-r)} \tag{3.5}
$$

Exercise 4

1. No. After each image is blurred with a 3 * 3 averaging mask. Their histograms are different.
2. We assume that 0 presents white and 1 presents black. So, we simplify the two figures into the following two matrices as input:

$$
left = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad right = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}
$$

After each image is blurred with a 3 * 3 averaging mask. We have the following two matrices:

$$
left = \begin{bmatrix} 0 & 0 & \frac{1}{3} & \frac{4}{3} & 2 & \frac{1}{3} \\ 0 & 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & \frac{1}{3} & \frac{4}{3} & 2 & \frac{1}{3} \end{bmatrix} \quad right = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{4}{3} & \frac{1}{3} & \frac{4}{3} & \frac{1}{3} \\ 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 \\ \frac{1}{3} & \frac{1}{3} & \frac{4}{3} & \frac{1}{3} & \frac{4}{3} & \frac{1}{3} \end{bmatrix}
$$

Then the histograms are below:

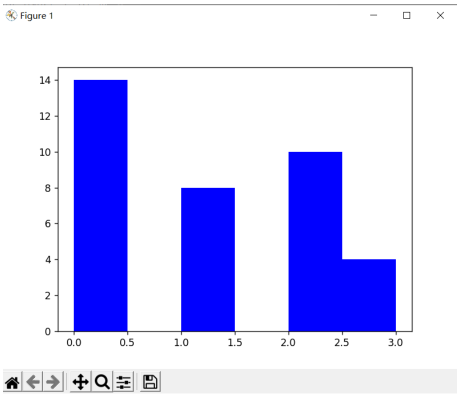


Figure 1: left

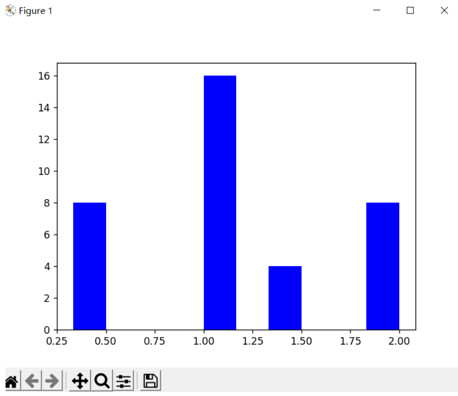


Figure 2: right

The python code is below:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import matplotlib.pyplot as plt
import numpy as np
input = np.array([[0,1,0,1,0,1],[0,1,0,1,0,1],[0,1,0,1,0,1],[0,1,0,1,0,1],[
                                    0,1,0,1,0,1],[0,1,0,1,0,1]])
input = input.reshape([1,6,6,1]) #
                                                                       conv2dreshape

kernel = np.array([[1,1,1],[1,1,1],[1,1,1]])
#kernel = kernel/3.0
kernel = kernel.reshape([3,3,1,1]) # k e r n e l reshape
print(input.shape,kernel.shape) #(1, 6, 6, 1) (3, 3, 1, 1)

x = tf.placeholder(tf.float32,[1,6,6,1])
k = tf.placeholder(tf.float32,[3,3,1,1])
output = tf.nn.conv2d(x,k,strides=[1,1,1,1],padding='SAME')


with tf.Session() as sess:
    y = sess.run(output,feed_dict={x:input,k:kernel})
    print(y.shape) #(1,6,6,1)
    print(y) #
                                                                                y6

                                             *    6
    #[[0,0,1/3,4/3,2,4/3],                  [[1/3,1/3,4/3,1/3,4/3,1/3],
    # [0,0,1,2,3,2],                         [1,1,2,1,2,1],
    # [0,0,1,2,3,2],                         [1,1,2,1,2,1],
    # [0,0,1,2,3,2],                         [1,1,2,1,2,1],
    # [0,0,1,2,3,2],                         [1,1,2,1,2,1],
    # [0,0,1/3,4/3,2,4/3]]                   [1/3,1/3,4/3,1/3,4/3,1/3]]

#a = np.array([0,0,1/3,4/3,2,4/3,0,0,1,2,3,2,0,0,1,2,3,2,0,0,1,2,3,2,0,0,1,
                                    2,3,2,0,0,1/3,4/3,2,4/3])
a = np.array([1/3,1/3,4/3,1/3,4/3,1/3,1/3,1/3,4/3,1/3,4/3,1/3,1,1,2,1,
2,1,1,1,2,1,2,1,1,1,2,1,2,1,1,1,2,1,2,1])
a1 = a*9
b = a.flatten()
plt.hist(b,color='b')
plt.show()
```

Exercise 5

The code is below: image1 is the origin image and image2 is the image after making equalization.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
def historgram(image):
 #print(cdf)
 plt.hist(image.flatten(), 256, [0, 256], color = 'r') # plot the historgram
 plt.xlim([0, 256]) # set x
 plt.show()


if __name__ == '__main__':
  img1 = cv2.imread('./cameraman.jpg', 0) # read the gray value of the image
  historgram(img1) # draw the gray value of the original image
  img2 = cv2.equalizeHist(img1) #make equalization
  historgram(img2)

  cv2.imwrite('./OriginImage.png',img1)
  cv2.imwrite('./NewImage.png',img2)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
```

The './cameraman.jpg' is the address of the input picture. The image is a "Standard" Gray Scale Image which is from ImageProcessingPlace.com

The input image, the histogram of the input image, the equalized image and its histogram are as follow:

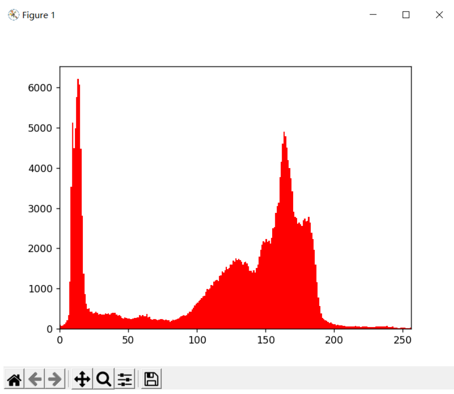


Figure 3: Origin image and its histogram

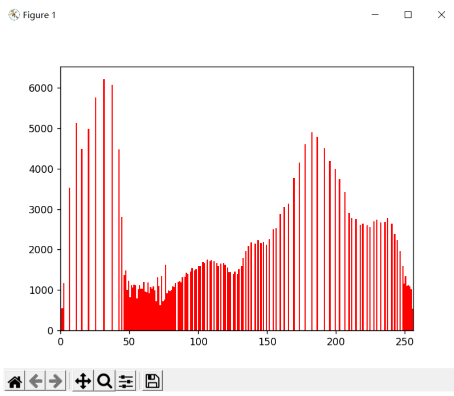


Figure 4: New image and its histogram

We can clearly see that the equalized image is brighter and more equalized than the original image.

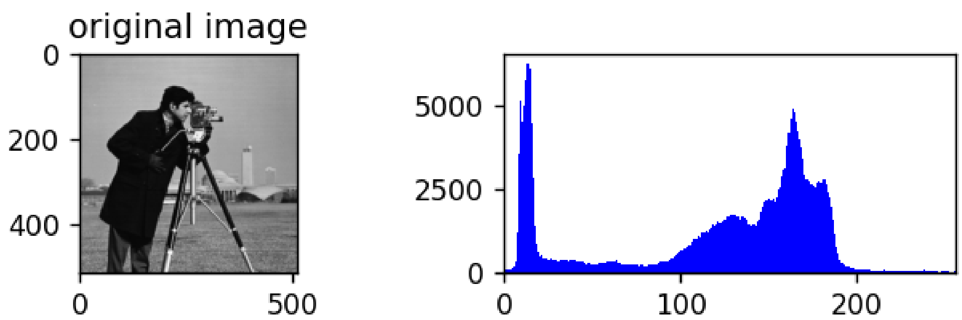
Exercise 6

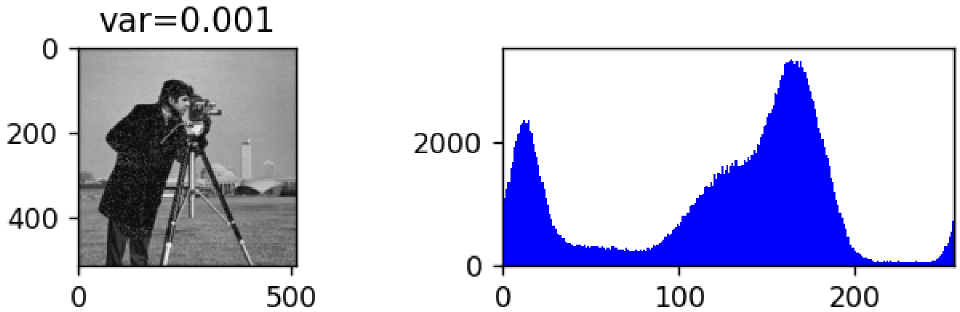
The code is below:

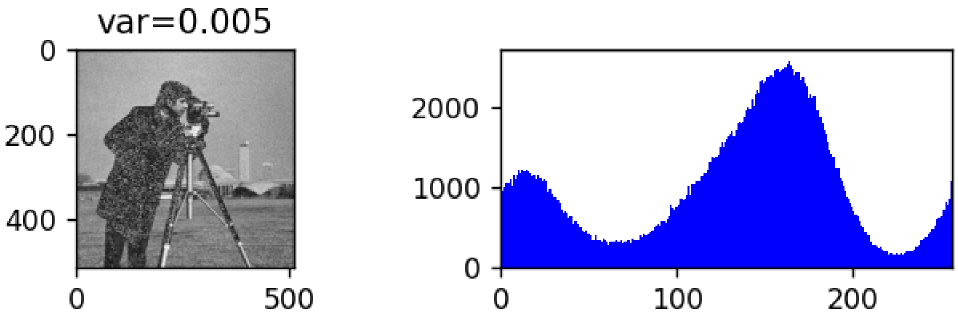
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def historgram(image):
 # Make a gray histogram of a given image, intensity [0,256]
 plt.hist(image.flatten(), 256, [0, 256], color = 'b') # plot the historgram
 plt.xlim([0, 256]) # set x
 plt.show()


def gaussian_noise(image, mean=0, var=0.001):
 # add gasuss noise
 image = np.array(image/255, dtype= float)
 noise = np.random.normal(mean, var ** 0.5, image.shape)
 out = image + noise
 if out. min() < 0:
  low_clip = -1.
 else:
  low_clip = 0.
 out = np.clip(out, low_clip, 1.0)
 out = np.uint8(out*255)
 return out

if __name__ == '__main__':
 img = cv2.imread('./cameraman.jpg', 0)

 # Add gaussian noise
 gauss1 = gaussian_noise(img,0,0.001)
 #historgram(gauss1)
 gauss2 = gaussian_noise(img,0,0.005)
 #historgram(gauss2)
```

```
""" plt.subplot(321)
plt.imshow(img, cmap='gray')
plt.title('original image')
plt.subplot(322)
historgram(img)
plt.subplot(323)
plt.imshow(gauss1, cmap='gray')
plt.title('var=0.001')
plt.subplot(324)
historgram(gauss1)
plt.subplot(325)
plt.imshow(gauss2, cmap='gray')
plt.title('var=0.005')
plt.subplot(326)
historgram(gauss2) """
#Remove gassuain noise by gaussian blur (for image gauss2)
dst1=cv2.GaussianBlur(gauss2,(7,7),3)
dst2=cv2.GaussianBlur(gauss2,(9,9),3)
dst3=cv2.GaussianBlur(gauss2,(11,11),3)
plt.subplot(131)
plt.imshow(dst1,cmap='gray')
plt.title('ksize = 7*7')
plt.subplot(132)
plt.imshow(dst2,cmap='gray')
plt.title('ksize = 9*9')
plt.subplot(133)
plt.imshow(dst3,cmap='gray')
plt.title('ksize = 11*11')
plt.show()
#cv2.imwrite('./dst.png',dst)
```

Below are images with different levels of gaussian noise and their histograms. We can see that as the noise level increases, the image gradually becomes blurred.







After implementing gaussian filter on these images, we have the result below: