Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Ing. Informática
VISION COMPUTACIONAL
Práctica Nº 02

MUESTREO, CUANTIZACION Y ESPACIOS DE COLORES DE LAS IMAGENES

Iván C. Medrano Valencia

1. OBJETIVO.

- Consolidar los conceptos de muestreo, cuantización y espacio de colores de las imágenes

2. ACTIVIDAD 1.

- **MUESTREO.**

  El muestreo de imágenes consiste en dividir una imagen continua espacialmente en cuadrículas M × N, y cada cuadrícula se representa mediante un valor de brillo o un valor de gris. Cuanto mayor sea el intervalo de muestreo de la imagen, menor será el número de píxeles de la imagen, menor será la resolución espacial y peor será la calidad de la imagen; por el contrario, cuanto menor sea el intervalo de muestreo de la imagen, mayor será el número de píxeles de la imagen y mayor será la resolución espacial. Mayor y mejor será la calidad de la imagen, pero la cantidad de datos aumentará en consecuencia. La Fig. 1 muestra la gráfica "Lena" con diferentes intervalos de muestreo.

Fig. 1. Diferentes intervalos de muestreo

- **EJERCICIOS PRÁCTICOS**

  Ejercicio 2.1

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
# Leer la imagen original
img = cv2.imread("images\lena.jpg")

# Obtener altura y ancho
height = img.shape[0]
width = img.shape[1]
print('height : ',img.shape[0])
print('width : ', img.shape[1])
# Conversión de muestra en área de 16 * 16
numHeight = int(height/16)
numwidth = int(width/16)

# crea una nueva imagen
new_img = np.zeros((height, width, 3), np.uint8)

# bucle de muestreo de imagen en area de 16*16
for i in range(16):
    # Obtener coordena Y
    y = i*numHeight
    for j in range(16):
        # Obtener coordenada X
        x = j*numwidth
        # Obtener los píxeles de color de relleno en la esq. sup. izq.
        b = img[y, x][0]
        g = img[y, x][1]
        r = img[y, x][2]
        # Configuración de ciclo de muestreo de área pequeña
        for n in range(numHeight):
            for m in range(numwidth):
                new_img[y+n, x+m][0] = np.uint8(b)
                new_img[y+n, x+m][1] = np.uint8(g)
                new_img[y+n, x+m][2] = np.uint8(r)

# Mostrar imagenes
cv2.imshow("src", img)
cv2.imshow("", new_img)

# Esperar
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. ACTIVIDAD 2

- **Cuantización**

La denominada cuantificación es el proceso de convertir el intervalo de cambio continuo del brillo correspondiente del píxel de la imagen en un único valor específico, es decir, discretizar el valor de amplitud de las coordenadas espaciales de la imagen gris original. Cuanto mayor sea el nivel de cuantificación, más rica será la imagen, mayor será la resolución de la escala de grises y mejor será la calidad de la imagen; cuanto menor sea el nivel de cuantificación, menos rico será el nivel de la imagen y cuanto menor sea la resolución de la escala de grises, se producirá el fenómeno de estratificación del contorno de la imagen. Reduce la calidad de la imagen. La Figura 2 es el proceso de convertir el valor de gris continuo de una imagen a un nivel de gris de 0 a 255.

Si el nivel de cuantificación es 2, se utilizarán dos niveles de gris para representar los píxeles de la imagen original (0-255), 0 para valores de escala de grises inferiores a 128, 128 para aquellos superiores o iguales a 128; si el nivel de cuantificación es 4, utilizará Los cuatro niveles de gris representan los píxeles de la imagen original. La nueva imagen se superpondrá en cuatro colores, el intervalo 0-64 es 0, el intervalo 64-128 es 64, el intervalo 128-192 es 128 y el intervalo 192-255 es 192; analogía.

La figura 6-2 es un gráfico "Lena" que compara diferentes niveles de cuantificación. El nivel de cuantificación de (a) es 256, el nivel de cuantificación de (b) es 64, el nivel de cuantificación de (c) es 16, el nivel de cuantificación de (d) es 8, el nivel de cuantificación de (e) es 4, y el nivel de cuantificación de (f) es 2.

- **EJERCICIOS PRÁCTICOS**

Ejercicio 2.2.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Leer la imagen original
img = cv2.imread("images/lena.jpg")

# Obtener alto y ancho de la imagen
height = img.shape[0]
width = img.shape[1]

#Crear una imagen
new_img = np.zeros((height, width, 3), np.uint8)
```

```
# Operación de cuantización de la imagen.
# el nivel de cuantización es 2
for i in range(height):
    for j in range(width):
        for k in range(3): #Corresponding to BGR three components
            if img[i, j][k] < 128:
                gray = 0
            else:
                gray = 128
            new_img[i, j][k] = np.uint8(gray)

# Mostrar la imagen
cv2.imshow("src", img)
cv2.imshow("", new_img)

# Esperar
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Ejercicio 2.3.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Leer la imagen original
img = cv2.imread("images/lena.jpg")

#Get image height and width
# Obtener alto y ancho de la imagen
height = img.shape[0]
width = img.shape[1]

# Crear imagenes nuevas
new_img1 = np.zeros((height, width, 3), np.uint8)
new_img2 = np.zeros((height, width, 3), np.uint8)
new_img3 = np.zeros((height, width, 3), np.uint8)

# Cuantización de la imagen nivel 2
for i in range(height):
    for j in range(width):
        for k in range(3): # Corresponmdiente a los comp. BGR
            if img[i, j][k] < 128:
                gray = 0
            else:
                gray = 128
            new_img1[i, j][k] = np.uint8(gray)

# Cuantización de la imagen nivel 4
for i in range(height):
```

```
        for j in range(width):
            for k in range(3): # Correspondente a los comp. BGR
                if img[i, j][k] < 64:
                    gray = 0
                elif img[i, j][k] < 128:
                    gray = 64
                elif img[i, j][k] < 192:
                    gray = 128
                else:
                    gray = 192
                new_img2[i, j][k] = np.uint8(gray)

    # Cuantización de la imagen nivel 4
    for i in range(height):
        for j in range(width):
            for k in range(3): # Correspondente a los comp. BGR
                if img[i, j][k] < 32:
                    gray = 0
                elif img[i, j][k] < 64:
                    gray = 32
                elif img[i, j][k] < 96:
                    gray = 64
                elif img[i, j][k] < 128:
                    gray = 96
                elif img[i, j][k] < 160:
                    gray = 128
                elif img[i, j][k] < 192:
                    gray = 160
                elif img[i, j][k] < 224:
                    gray = 192
                else:
                    gray = 224
                new_img3[i, j][k] = np.uint8(gray)

    # Mostrar las imagenes
    titles = [u'(a) Imagen Original', u'(b) Cuantización-
    L2', u'(c) Cuantización-L4', u'(d) Cuantización-L8']
    images = [img, new_img1, new_img2, new_img3]
    for i in range(4):
       plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray'),
       plt.title(titles[i])
       plt.xticks([]),plt.yticks([])
    plt.show()
```

4. ACTIVIDAD 3

- **Cambio del Espacio de Colores**

  Ejercicio 2.4. Dividir los canales RGB

```
import cv2 as cv
import numpy as np

imagen = cv.imread("images/lion.jpg")
cv.namedWindow("input image", cv.WINDOW_AUTOSIZE)
cv.imshow("input image", imagen)

B, G, R = cv.split (imagen) # separación de canales
cv.imshow("blue", B)
cv.imshow("green", G)
cv.imshow("red", R)

zeros = np.zeros(imagen.shape[:2], dtype = "uint8")
cv.imshow("Red", cv.merge([zeros, zeros, R]))
cv.imshow("Green", cv.merge([zeros, G, zeros]))
cv.imshow("Blue", cv.merge([B, zeros, zeros]))
cv.waitKey(0)

# mezclar nuevamente los canales
imagen = cv.merge ([B, G, R]) # Combinación de canales
cv.imshow("imagen mezclada", imagen)

cv.waitKey(0)
cv.destroyAllWindows()
```

Ejercicio 2.5.

```
import cv2
img_BGR = cv2.imread("images/cat.png")

cv2.imshow("img_BGR",img_BGR)

# Convertir al formato RGB
img_RGB = cv2.cvtColor(img_BGR,cv2.COLOR_BGR2RGB)
cv2.imshow("img_RGB",img_RGB)

# Convertir de RGB a HSV
img_RGB2HSV = cv2.cvtColor(img_BGR,cv2.COLOR_RGB2HSV)
cv2.imshow("img_RGB2HSV",img_RGB2HSV)

# Convertir de BGR
img_BGR2HSV = cv2.cvtColor(img_BGR,cv2.COLOR_BGR2HSV)
cv2.imshow("img_BGR2HSV",img_BGR2HSV)

img_BGR2GRAY = cv2.cvtColor(img_BGR, cv2.COLOR_BGR2GRAY)
cv2.imshow("img_BGR2GRAY",img_BGR2GRAY)
# Esperar
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Ejercicio 2.6.

```
import cv2 as cv
import numpy as np
def color_space_demo (imagen): # conversión de color
    #Convertir a escala de grises
    gray = cv.cvtColor (imagen, cv.COLOR_BGR2GRAY)
    cv.imshow("gray", gray)
    #Convertir imagen en formato HSV
    hsv = cv.cvtColor (imagen, cv.COLOR_BGR2HSV)
    cv.imshow("hsv", hsv)
    #Convertir imagen en formato YUV (más común)
    yuv = cv.cvtColor (imagen, cv.COLOR_BGR2YUV)
    cv.imshow("yuv", yuv)
    #Convertir imagen en formato Ycrcb
    Ycrcb = cv.cvtColor (imagen, cv.COLOR_BGR2YCrCb)
    cv.imshow("ycrcb", Ycrcb)
    #Convertir imagen en formato Ycrcb
    XYZ= cv.cvtColor(imagen, cv.COLOR_BGR2XYZ)
    cv.imshow("xyz", XYZ)

src = cv.imread("images/lena.jpg")
cv.namedWindow("input image", cv.WINDOW_AUTOSIZE)
cv.imshow("input image", src)
color_space_demo(src)

cv.waitKey(0)
cv.destroyAllWindows()
```

5. REFERENCIAS BIBLIOGRÁFICAS

- Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision With OpenCV, Wiley & Sons Ltd.
- Hafsa Asad, W. R., Nikhil Singh (2020). The Computer Vision Workshop. Birmingham U.K., Pack Publishing.
- Szeliski, R. (2011). Computer Vision Algorithms and Applications. London, Springer.