

# Práctica 0

Antonio Álvarez Caballero

October 7, 2015

## Cuestiones

- 1 ¿Qué relación hay en OpenCV entre imágenes y matrices? Justificar la respuesta.**

OpenCV (y la mayoría de software por no decir todo) utilizan matrices como estructura de datos para almacenar imágenes. Esto lo consiguen asignando cada píxel a una posición de la matriz, y en ella un vector de 1, 3 o 4 valores dependiendo del número de canales de la imagen.

- 2 Diga el significado de los siguientes tipos OpenCV: 8UC1, 8UC2, 8UC3, 32SC1, 32SC2, 32SC3, 32FC1, 32FC2, 32FC3. ¿Cuáles de ellos están asociados a imágenes? Justificar la respuesta.**

El primer dígito indica los bits que ocupa cada píxel. La primera letra indica el tipo. *U* de unsigned char, *S* de signed int y *F* de float (podría ser un double). Y por último *Cx* indica el número de canales de la imagen.

Asociados a imágenes son todos excepto los que tienen 2 canales. No hay tipos de imagen con 2 canales y por tanto OpenCV no puede mostrarlos. Los demás los muestra OpenCV sin problema.

- 3 ¿Qué relación existe entre cada tipo visual de una imagen: (color, grises, blanco y negro) y los tipos de almacenamiento de OpenCV ? Justificar la respuesta.**

El primer número, el de bits que contiene cada píxel, se utiliza para el número de colores que puede albergar. A más cantidad de píxels mayor variedad de información podremos albergar en nuestra imagen.

El número de canales nos permite guardar más información para cada píxel: si usamos un solo canal, sólo podremos albergar imágenes en blanco y negro o en escala de grises. En cambio, utilizando tres canales podremos albergar imágenes RGB o HSV entre otras. Si utilizáramos 4 podríamos usar el canal *alpha* para la transparencia.

**4 ¿Es posible realizar operaciones entre imágenes de distinto tipo visual? Justificar la respuesta.**

Sí, ya que el tipo visual HSV (Tono, Saturación, Valor) tiene el mismo tipo de almacenamiento que el RGB (Rojo, Verde, Azul), y por tanto las matrices que almacenan imágenes de estos tipos tienen las mismas dimensiones. Eso sí, no tendría sentido realizar estas operaciones, ya que HSV no tiene nada que ver con RGB.

**5 ¿Cuál es la orden OpenCV que permite transformar el tipo de almacenamiento de una matriz en otro distinto?**

La orden buscada es *void convertTo(OutputArray img, int rtype, double alpha = 1, double beta = 0)*. Es un método de la clase Mat para convertir al tipo *rtype* y guardar en *img*. *alpha* es un parámetro de escala opcional y *beta* de traslación.

**6 ¿Cuál es la orden OpenCV que permite transformar el tipo visual de una imagen en otro distinto? ¿Por qué es distinta de la que transforma un tipo de almacenamiento en otro?**

La función buscada es *void cvtColor(inputArray src, OutputArray Dst, int code, int dstCn = 0)*. Transforma la imagen *src* en *dst* al tipo visual *code*. El parámetro *dstCn* es el número de canales de la nueva imagen, que si es 0 se calculará automáticamente.

Estas funciones son distintas ya que, aún teniendo la misma cantidad de información, dos imágenes con tipo visual RGB y HSV no son iguales. Almacenan la misma cantidad de información pero su codificación interna es distinta: en una son valores de Rojo, Azul y Verde y en la otra Tono, Saturación y Valor.

# Ejercicios

- 1 Escribir una función que lea una imagen en niveles de gris o en color ( `im=leeimagen(filename, flagColor)` )

```
1  #!/usr/bin/env python
2  # -*- coding: UTF-8 -*-
3
4  import cv2
5  import numpy as np
6
7  def leeImagen(nombre, flagColor):
8      if flagColor == True:
9          flag = cv2.IMREAD_COLOR
10     else:
11         flag = cv2.IMREAD_GRAYSCALE
12     return cv2.imread(nombre, flag)
13
14 # Aquí cargamos imágenes con varios flags
15 img = leeImagen("lena.jpg", True)
16
17 cv2.imshow("Lena", img)
18 cv2.waitKey(0)
19 cv2.destroyAllWindows()
```

- 2 Escribir una función que visualice una imagen (`pintaI(im)`)

```
1  #!/usr/bin/env python
2  # -*- coding: UTF-8 -*-
3
4  import cv2
5  import numpy as np
6
7  def leeImagen(nombre, flagColor):
8      if flagColor == True:
9          flag = cv2.IMREAD_COLOR
10     else:
11         flag = cv2.IMREAD_GRAYSCALE
12     return cv2.imread(nombre, flag)
13
14 def pintaImagen(nombre, imagen):
15     cv2.imshow(nombre, imagen)
16     cv2.waitKey(0)
17     cv2.destroyAllWindows()
18
19 # Aquí cargamos imágenes con varios flags
20 # img = leeImagen("lena.jpg", cv2.IMREAD_GRAYSCALE)
21 # img = leeImagen("lena.jpg", cv2.IMREAD_COLOR)
22 img = leeImagen("lena.jpg", True)
23 pintaImagen("Lena", img)
```

### 3 Escribir una función que visualice varias imágenes a la vez: `pintaMI(vim)`. (vim será una secuencia de imágenes) ¿Qué pasa si las imágenes no son todas del mismo tipo: (nivel de gris, color, blanco-negro)?

```
1  #!/usr/bin/env python
2  # -*- coding: UTF-8 -*-
3
4  import cv2
5  import numpy as np
6
7  def leeImagen(nombre, flagColor):
8      if flagColor == True:
9          flag = cv2.IMREAD_COLOR
10     else:
11         flag = cv2.IMREAD_GRAYSCALE
12     return cv2.imread(nombre, flag)
13
14 def pintaSecuenciaImágenes(imágenes):
15     # El ancho total será la suma de los anchos de todas las imágenes
16     # El alto total será el máximo de los altos de todas las imágenes
17     ancho_total = sum([img.shape[1] for img in imágenes])
18     alto_total = max([img.shape[0] for img in imágenes])
19
20     # Creamos una imagen vacía para ir rellenando
21     tira_imágenes = np.zeros((alto_total, ancho_total, 3), np.uint8)
22
23     # Rellenamos la tira de imágenes
24     ancho_anterior = 0
25     for indice_imagen in range(len(imágenes)):
26         imagen_actual = imágenes[indice_imagen]
27         # Si no está en a color, las convertimos
28         if len(imagen_actual.shape) < 3:
29             imagen_actual = cv2.cvtColor(imagen_actual, cv2.COLOR_GRAY2RGB)
30
31         for indice_fila in range(imagen_actual.shape[0]):
32             for indice_columna in range(imagen_actual.shape[1]):
33                 tira_imágenes[indice_fila][indice_columna + ancho_anterior] =
34                     imagen_actual[indice_fila][indice_columna]
35                 ancho_anterior = ancho_anterior + imagen_actual.shape[1]
36
37     # Visualizamos
38     cv2.imshow("Tira imágenes", tira_imágenes)
39     cv2.waitKey(0)
40     cv2.destroyAllWindows()
41
42 # Aquí cargamos imágenes con varios flags
43 imágenes = [leeImagen("lena.jpg", True) for i in range(3)]
44 imágenes.append(leeImagen("lena.jpg", False))
45 pintaSecuenciaImágenes(imágenes)
```

#### 4 Escribir una función que modifique el valor en una imagen de una lista de coordenadas de píxeles.

```
1  #!/usr/bin/env python
2  # -*- coding: UTF-8 -*-
3
4  import cv2
5  import numpy as np
6
7  def leeImagen(nombre, flagColor):
8      if flagColor == True:
9          flag = cv2.IMREAD_COLOR
10     else:
11         flag = cv2.IMREAD_GRAYSCALE
12     return cv2.imread(nombre, flag)
13
14 def pintaImagen(nombre, imagen):
15     cv2.imshow(nombre, imagen)
16     cv2.waitKey(0)
17     cv2.destroyAllWindows()
18
19 # Supongamos que pixels es una listas de tuplas (y,x) y valores una lista de valores
20 def modificaPixels(imagen, pixels, valores):
21     if len(pixels) != len(valores):
22         return "No hay el mismo número de pÃxels que de valores"
23     for i in range(len(pixels)):
24         imagen[pixels[i][0], pixels[i][1]] = valores[i]
25
26 # Cargar imagen
27 img = leeImagen("lena.jpg", True)
28 height, width = img.shape[:2]
29 pixels = [(y,x) for x in range(width) for y in range(height) if y % 2 != 0]
30 valores = [[255, 255, 255] for i in range(len(pixels))]
31
32 modificaPixels(img, pixels, valores)
33 pintaImagen("Lena", img)
```