Práctica 0

Antonio Álvarez Caballero

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 distintos tipos visuales pueden utilizarl el mismo tipo de almacenamiento. Por ejemplo, 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.

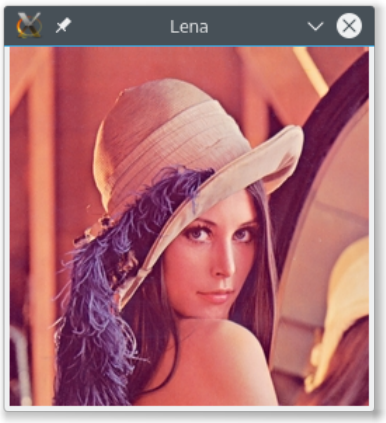
Estas funciones son distintas ya que, aún teniendo la misma cantidad de información, dos imágenes con tipo visual RBG y HSV no son iguales. Almacenan la misma cantidad de información pero su codificación interna es distinta: en una son valores de Rojo, Azul y Verde y en la otra Tono, Saturación y Valor.

Ejercicios

Estos ejercicios están implementados en Python (cuidando que sea compatible con Python2 y Python3 sin problema), pero a partir de la práctica 1 todo el código estará en C++.

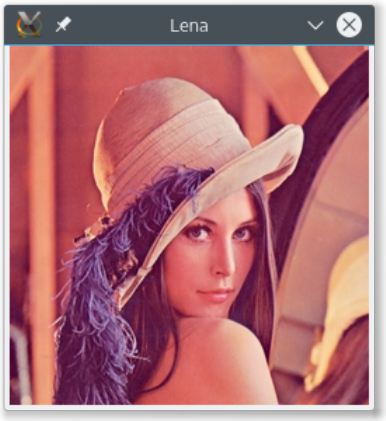
1 Escribir una función que lea una imagen en niveles de gris o en color (im=leeimagen(filename, flagColor))

```python
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from __future__ import print_function

import cv2
import numpy as np

def leeImagen(nombre, flagColor):
    if flagColor == True:
        flag = cv2.IMREAD_COLOR
    else:
        flag = cv2.IMREAD_GRAYSCALE
    return cv2.imread(nombre,flag)

# AquÃ cargamos imÃ¡genes con varios flags
img = leeImagen("lena.jpg",True)

cv2.imshow("Lena",img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



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

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from __future__ import print_function

import cv2
import numpy as np

def leeImagen(nombre, flagColor):
    if flagColor == True:
        flag = cv2.IMREAD_COLOR
    else:
        flag = cv2.IMREAD_GRAYSCALE
    return cv2.imread(nombre,flag)

def pintaImagen(nombre, imagen):
    cv2.imshow(nombre, imagen)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# AquÃ cargamos imÃ¡genes con varios flags
# img = leeImagen("lena.jpg",cv2.IMREAD_GRAYSCALE)
# img = leeImagen("lena.jpg",cv2.IMREAD_COLOR)
img = leeImagen("lena.jpg",True)
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)?

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from __future__ import print_function

import cv2
import numpy as np

def leeImagen(nombre, flagColor):
    if flagColor == True:
        flag = cv2.IMREAD_COLOR
    else:
        flag = cv2.IMREAD_GRAYSCALE
    return cv2.imread(nombre,flag)

def pintaSecuenciaImagenes(imagenes):
    # El ancho total serÃ¡ la suma de los anchos de todas las imÃ¡genes
    # El alto total serÃ¡ el mÃ¡ximo de los altos de todas las imÃ¡genes
    ancho_total = sum([img.shape[1] for img in imagenes])
    alto_total = max([img.shape[0] for img in imagenes])

    # Creamos una imagen vacia para ir rellenando
    tira_imagenes = np.zeros((alto_total,ancho_total,3), np.uint8)

    # Rellenamos la tira de imagenes
    ancho_anterior = 0
    for indice_imagen in range(len(imagenes)):
        imagen_actual = imagenes[indice_imagen]
        # Si no estÃn a color, las convertimos
        if len(imagen_actual.shape) < 3:
            imagen_actual = cv2.cvtColor(imagen_actual,cv2.COLOR_GRAY2RGB)

        for indice_fila in range(imagen_actual.shape[0]):
            for indice_columna in range(imagen_actual.shape[1]):
                tira_imagenes[indice_fila][indice_columna + ancho_anterior] =
                    imagen_actual[indice_fila][indice_columna]
        ancho_anterior = ancho_anterior + imagen_actual.shape[1]

    # Visualizamos
    cv2.imshow("Tira imÃ¡genes",tira_imagenes)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# AquÃ cargamos imÃ¡genes con varios flags
imagenes = [leeImagen("lena.jpg",True) for i in range(3)]
imagenes.append(leeImagen("lena.jpg",False))
pintaSecuenciaImagenes(imagenes)
```


Tira imágenes

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

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

from __future__ import print_function

import cv2
import numpy as np

def leeImagen(nombre, flagColor):
    if flagColor == True:
        flag = cv2.IMREAD_COLOR
    else:
        flag = cv2.IMREAD_GRAYSCALE
    return cv2.imread(nombre,flag)

def pintaImagen(nombre, imagen):
    cv2.imshow(nombre, imagen)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Supongamos que pixels es una listas de tuplas (y,x) y valores una lista de valores
def modificaPixels(imagen,pixels,valores):
    if len(pixels) != len(valores):
        return "No hay el mismo nÃºmero de pÃxels que de valores"
    for i in range(len(pixels)):
        imagen[pixels[i][0],pixels[i][1]] = valores[i]

# Cargar imagen
img = leeImagen("lena.jpg",True)
height, width = img.shape[:2]
pixels = [(y,x) for x in range(width) for y in range(height) if y % 2 != 0]
valores = [[255,255,255] for i in range(len(pixels))]

modificaPixels(img,pixels,valores)
pintaImagen("Lena",img)
```