Práctica 0
Introducción a OpenCV

Alejandro García Montoro
agarciamontoro@correo.ugr.es

4 de octubre de 2015

1. Cuestiones

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

Solución. Las imágenes en OpenCV se codifican como matrices; en particular, como arrays de dos o tres dimensiones, dependiendo de si la imagen tiene un canal —usualmente para imágenes en blanco y negro— o varios —normalmente tres: rojo, verde y azul; y en algún caso un cuarto: alpha—.

En la documentación así se especifica y, además, es directo comprobar el tipo de dato que devuelve, por ejemplo, en Python, la orden de lectura de imágenes:

```
>>> img = cv2.imread("../IMG/lena.jpg")
>>> type(img)
<class 'numpy.ndarray'>
>>> img.shape
(256, 256, 3)
```

El tipo de *img* es *numpy.ndarray*; como decíamos, un array de dimensiones variables. En este caso, al ser una imagen cuadrada de 256×256 px con tres canales: B, G, R, tenemos un array tridimensional, de $256 \times 256 \times 3$.

Cuestión 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.

Solución. Todos ellos se refieren a tipos de datos. Atendiendo a la documentación[1], los tipos siguen la siguiente notación:

CV_<bit-depth>{U|S|F}C(<number_of_channels>)

[1] http://docs.opencv.org/modules/core/doc/basic_structures.html

donde U significa *unsigned*, sin signo; S significa *signed*, con signo; y F, *float*, flotante. Por tanto, la lista de tipos y su significado queda como sigue:

- **8UC1**: dato entero de 8 bits sin signo, un canal.
- **8UC2**: dato entero de 8 bits sin signo, dos canales.
- **8UC3**: dato entero de 8 bits sin signo, tres canales.
- **32SC1**: dato entero de 32 bits con signo, un canal.
- **32SC2**: dato entero de 32 bits con signo, dos canales.
- **32SC3**: dato entero de 32 bits con signo, tres canales.
- **32FC1**: dato flotante de 32 bits, un canal.
- **32FC2**: dato flotante de 32 bits, dos canales.
- **32FC3**: dato flotante de 32 bits, tres canales.

Cuestión 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.

Solución. Cada tipo visual se codifica de una manera en los tipos de almacenamiento. Así, el número de canales de la imagen —esto es, si es una imagen en grises, en color, o incluso con canales de transparencia—, determina la posible tercera dimensión del array donde se almacenan los valores de los píxeles. Por ejemplo: si la imagen es a color —con los canales B, G, R—, el tipo de almacenamiento tiene que ser un array de tres dimensiones, donde las dos primeras se corresponden con las coordenadas de cada píxel y la tercera con el valor de los canales que, unidos, reproducen un color. Por otro lado, en el caso de imágenes en grises sólo se necesita almacenar el nivel de luminosidad de cada píxel, que se codifica con un solo entero de 0 a 255; por tanto, sólo se necesita un array de dos dimensiones que almacene, para cada píxel, su nivel de luminosidad.

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

Solución. Los distintos tipos visuales de las imágenes se traducen en muchas ocasiones, como acabamos de ver, en matrices de distinta dimensión. Por la naturaleza matemática de las operaciones sobre las imágenes, que al fin y al cabo son operaciones sobre matrices, sabemos que es imposible definir todas ellas para matrices de dimensión general y distinta.

En algunos casos, y de forma particular, sí que se podrían definir operaciones para imágenes de distinto tipo visual; pero en ningún caso podríamos esperar una librería que proporcionara estas funciones de forma general.

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

Solución. En la API de C++, la orden es la siguiente:

```
void Mat::convertTo(OutputArray m, int rtype, double
    alpha=1, double beta=0) const
```

La API de Python no tiene una llamada equivalente propia de OpenCV. En este caso, la conversión se hace directamente con las funciones del módulo NumPy. Hay algunas conversiones, como la de pasar de un canal a varios, que sí se hacen con funciones propias. Sirva el siguiente trozo de código como ejemplo:

```
#Leemos una imagen con tipo de dato 8UC1
>>> img_8UC1 = cv2.imread("../IMG/lena.jpg",
    cv2.IMREAD_GRAYSCALE)
>>> print(img_8UC1.shape, img_8UC1.dtype)
(256, 256) uint8

#Convertimos a tipo de dato 32FC1
>>> img_32FC1 = np.float32(img_8UC1)
>>> print(img_32FC1.shape, img_32FC1.dtype)
(256, 256) float32

#Convertimos a tipo de dato 32FC3
>>> img_32FC3 = cv2.merge([img_32FC1,img_32FC1,img_32FC1])
>>> print(img_32FC3.shape, img_32FC3.dtype)
(256, 256, 3) float32
```

Cuestión 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?

Solución. En este caso sí que existe una función en la API de Python que hace exactamente esto:

```
cv2.cvtColor(src, code[, dst[, dstCn]])
```

Como se describe en la documentación[2], *src* es la imagen que se quiere cambiar, *dst* es la imagen donde se guardará el cambio —si no se especifica, la función devuelve esta imagen—, *dstCn* es el número de canales que tendrá la imagen final —si no se especifica o se deja el valor por defecto: 0, el número de canales se infiere de *code*— y *code* es el código de conversión entre espacios, que puede ser uno de los siguientes[3]:

[2] `http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor`

[3] En la API de Python, el prefijo *CV_* de los códigos hay que cambiarlo por *COLOR_*.

- RGB ↔ GRAY
  - CV_BGR2GRAY, CV_RGB2GRAY, CV_GRAY2BGR, CV_GRAY2RGB
- RGB ↔ CIE XYZ.Rec 709 con punto blanco D65
  - CV_BGR2XYZ, CV_RGB2XYZ, CV_XYZ2BGR, CV_XYZ2RGB
- RGB ↔ YCrCb JPEG (o YCC)
  - CV_BGR2YCrCb, CV_RGB2YCrCb, CV_YCrCb2BGR, CV_YCrCb2RGB
- RGB ↔ HSV
  - CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB
- RGB ↔ HLS
  - CV_BGR2HLS, CV_RGB2HLS, CV_HLS2BGR, CV_HLS2RGB
- RGB ↔ CIE L*a*b*
  - CV_BGR2Lab, CV_RGB2Lab, CV_Lab2BGR, CV_Lab2RGB
- RGB ↔ CIE L*u*v*
  - CV_BGR2Luv, CV_RGB2Luv, CV_Luv2BGR, CV_Luv2RGB
- Bayer → RGB
  - CV_BayerBG2BGR, CV_BayerGB2BGR, CV_BayerRG2BGR, CV_BayerGR2BGR, CV_BayerBG2RGB, CV_BayerGB2RGB, CV_BayerRG2RGB, CV_BayerGR2RGB

El tipo visual de una imagen no sólo comprende el número de canales que esta tiene, sino la forma en la que los datos están estructurados dentro de la matriz. Así, por ejemplo, una imagen en RGB tiene una codificación diferente a como la tiene una imagen en CIE L*a*b*, que no guarda los datos en los valores separados de rojo, verde y azul sino en luminosidad y valor en los ejes rojo-verde y amarillo-azul. No tiene sentido, por tanto, juntar las funciones que modifican los tipos de estructura y los que modifican los espacios de color.

Como ejemplo de esta discusión, veamos que las imágenes en RGB y en CIE L*a*b*, dos espacios de color diferentes, tienen el mismo tipo de estructura de datos:

```
1  >>> img = cv2.imread("../IMG/lena.jpg")
2  >>> img_Lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
3  >>> print(type(img), img.shape); print(type(img_Lab),
       img_Lab.shape)
4  <class 'numpy.ndarray'> (256, 256, 3)
5  <class 'numpy.ndarray'> (256, 256, 3)
```

2. Ejercicios

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

Solución.

```
def leeimagen(file_name, flag_color=true):
    #Valor por defecto
    cv_flag = cv2.IMREAD_COLOR

    if flag_color == false:
        cv_flag =cv2.IMREAD_GRAYSCALE

    return cv2.imread(file_name, cv_flag)
```

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

Solución.

```
def pintaI(img):
    cv2.namedWindow("Imagen")
    cv2.imshow("Imagen", img)
```

Ejercicio 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)?

Solución.

```
def pintaMI(vim):
    width = 0
    height = 0

    #Calculamos el ancho de la imagen final sumando
    #los anchos de todas las imagenes. Ademas, calculamos
    #el alto de la imagen final como el alto maximo de
    #todas las imagenes
    for img in vim:
        width += img.shape[0]
        if img.shape[1] > height: height = img.shape[1]

    big_img = np.zeros((height, width, 3), np.uint8)

    width = 0

    for img in vim:
        #Si es ByN, trasladamos esa imagen a tres canales
        #copiando el mismo valor en cada uno de ellos
```

```
            if len(img.shape) < 3:
                img = cv2.merge([img,img,img])

            #Copiamos la imagen actual en el ROI de la
            #imagen destino
            big_img[:, width:width+img.shape[0],:] = img

            #Actualizamos el valor del ancho para el proximo ROI
            width += img.shape[0]

        pintaI(big_img)
```

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

Solución.

```
    def setPXvalue(img, px_coord, value=[0,0,255]):
        for px in px_coord:
            img[px[0],px[1],:] = value
```