PRÁCTICA 1

INTRODUCCIÓN AL PROCESADO DE IMAGEN CON PYTHON

TRATAMIENTO DIGITAL DE IMAGEN

GRADO EN INGENIERÍA DE SONIDO E IMAGEN

CURSO 2020/2021

1. INTRODUCCIÓN A PYTHON

Para llevar a cabo las prácticas de la asignatura, se aconseja realizar una introducción al lenguaje de programación Python, revisando los siguientes capítulos del manual "Scipy Lecture Notes" (*www.scipy-lectures.org*):

1. Scientific computing with tools and workflow
2. The Python language

En la primera parte del curso, relativa al procesado y a la extracción de descriptores de imágenes, se utilizará el entorno de trabajo Spyder y una consola IPython. Para su instalación, se recomienda hacer uso de la plataforma Anaconda, una distribución libre que se puede descargar en el siguiente enlace:

https://www.anaconda.com/distribution/

Además, se utilizarán fundamentalmente las siguientes librerías de Python:

- Numpy: import numpy as np
- Matplotlib: import matplotlib.pyplot as plt
- Scikit-Image: import skimage
- Scipy (ndimage): from scipy import ndimage
- Scikit-Learn: import sklearn

Las librerías citadas anteriormente se introducen en los siguientes capítulos del manual "Scipy Lecture Notes" (*www.scipy-lectures.org*):

3. NumPy: creating and manipulating numerical data
4. Matplotlib: plotting
5. Scipy: high-level scientific computing
12. Image manipulation and processing using Numpy and Scipy
17. Scikit-Image: Image processing
20. Scikit-Learn: Machine Learning in Python

Por otro lado, en la segunda parte del curso, en la cual se introducirán las redes neuronales convolucionales (CNNs), se utilizará la plataforma Google Colab, junto con dos librerías adicionales:

- Keras: import keras
- Tensorflow: import tensorflow as tf

1.1 FUNCIONES IMPORTANTES EN PYTHON

En este apartado se describen, de forma breve, los comandos, funciones básicas y módulos necesarios para utilizar Python en el entorno de trabajo Spyder, bien utilizando la consola IPython o mediante scripts:

- **help(nombre_funcion)**: Una vez importadas las librerías a utilizar en la consola IPython, permite obtener información sobre el uso y la sintaxis de

cada función. Se espera que, a lo largo del curso, el estudiante sea lo suficientemente autónomo en el uso de esta ayuda. La interfaz gráfica de Spyder también permite acceder a la ayuda.

- **Variable Explorer**: Pestaña en la interfaz gráfica de Spyder que permite ver las variables que tenemos en memoria.
- **%reset_selective var**: Borrar la variable 'var' de la memoria.
- **%reset**: Borrar todas las variables que hay en la memoria.
- **who/whos**: Listado de las variables que tenemos en memoria, con información sobre su tipo, dimensiones y tamaño en bytes.
- **np.load('nombre_fichero')**: Carga el contenido de un fichero de tipo .npy o .npz en memoria.
- **np.save('nombre_fichero', var)**: Guarda en el fichero 'nombre_fichero' la variable 'var'.
- **np.savez('nombre_fichero', var1=var1, var2=var2, ..., varn=varn)**: Guarda en el fichero 'nombre_fichero' las variables que le indiquemos (v1, v2, …, vn).

1.2 FUNCIONES ESPECÍFICAS DE PROCESADO DE IMAGEN

Para cargar una imagen, utilizaremos la función **io.imread** de la librería Scikit-Image como sigue:

```
from skimage import io
x = io.imread('peppers.png')
```

En la variable x tendremos almacenada una matriz de 2 o 3 dimensiones, según se trate de una imagen en escala de grises o en color, respectivamente.

En el primer caso, las dimensiones de la matriz son N x M, donde N indica la dimensión horizontal de la imagen (número de píxeles del eje x) y M la dimensión vertical (número de píxeles del eje y), siendo (0,0) las coordenadas del píxel superior izquierdo. Cada elemento contiene el valor de luminancia (intensidad) de cada píxel imagen.

En el segundo caso (imagen en color), las dimensiones de la matriz son N x M x 3, ya que en la tercera dimensión disponemos de las intensidades en los tres planos de color R, G y B (es decir, rojo, verde y azul).

Al leer los datos de una imagen con la función **io.imread** los valores de intensidad de cada píxel vendrán dados en formato entero de 8 bits sin signo (uint8). Así, una variable en este formato solo puede tomar valores enteros entre 0 y 255. Cuando tenga que utilizar ciertas funciones matemáticas (ej. logaritmo o exponencial)

puede encontrarse con el problema de que estas funciones operan sólo sobre número en coma flotante de tipo *double*. En estos casos, le serán de ayuda los comandos:

```
x_float = skimage.img_as_float64(x)
x_uint = skimage.img_as_uint(x_float)
```

que transforman un número, vector o matriz desde su formato original a formato *double* o *uint8*, respectivamente.

2. VISUALIZACIÓN DE UNA IMAGEN

Existen varias funciones útiles para visualizar una imagen; en particular, nosotros utilizaremos **plt.imshow** de la librería Matplotlib.

```
plt.imshow(x)
plt.show()
```

Si utiliza la función **plt.imshow**, el formato de la imagen debe ser uint8 para que la visualización sea correcta sin establecer parámetros adicionales. Si los datos están en formato *double*, Matplotlib "espera" que dichos datos estén comprendidos en el rango de 0 a 1. Si no es así, debemos optar por normalizar los datos a ese rango, o bien establecer el nuevo rango de visualización (distinto de [0,1], consultar la ayuda de la función).

Un mapa o paleta de color (parámetro **cmap** en **plt.imshow**) es una matriz mx3 que establece una correspondencia entre un índice y un color especificado mediante 3 componentes. En otras palabras, proporciona una lista de colores, cada uno identificado mediante su índice, de modo que podemos representar una imagen en color mediante una matriz bidimensional que almacena el índice (color) asociado a cada píxel. Esto resulta práctico, por ejemplo, en la visualización de imágenes en infrarrojo monocromas, donde podemos asignar colores cercanos al rojo (cálidos) a los valores de intensidad elevada y colores cercanos al azul (fríos) a los de intensidad baja.

En este enlace se incluye un listado de los mapas de color disponibles en Matplotlib:

https://matplotlib.org/tutorials/colors/colormaps.html

Además, también se pueden crear mapas de color personalizados, estableciendo un rango de visualización entre dos colores mínimo 'minimumColor' y máximo 'maximumColor', así como un número de colores 'numcolors':

```
from matplotlib.colors import LinearSegmentedColormap
cm = LinearSegmentedColormap.from_list('name',
                                       [minimumColor,maximumColor],
                                       N=numcolors)
```

```
plt.imshow(x,cmap=cm)
plt.show()
```

Consulte la ayuda de la función 'LinearSegmentedColormap' para definir los colores mínimo y máximo de manera adecuada.

Si no se especifica el mapa de color, Matplotlib mostrará:

- Para imágenes 2D, un mapa de calor. Si se quiere mostrar la imagen en escala de grises, es necesario indicar *cmap='gray'*.
- Si se tienen 3 componentes, asumirá que es una imagen RGB.

2.1 EJERCICIO 1

1. Lea la imagen *'dreamers.png'* (utilizando *io.imread*) y muéstrela utilizando *plt.imshow*.

2. Visualice la imagen utilizando una paleta de 8 niveles de gris (creando previamente un mapa de color personalizado).

3. Escriba un script de Python que permita representar el mapa de color anterior como se muestra en la siguiente figura.

Figura 2.1: Mapa de 8 niveles de gris.

NOTA: Puede hacer uso de la función *np.concatenate*.

4. Repita el ejercicio anterior, generando ahora una figura similar para un mapa de color que contenga las 8 esquinas del cubo RGB: [(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)].

5. De acuerdo con el listado de mapas de color disponibles, ¿cuál de los mapas de color sería adecuado para el ejemplo mencionado de las imágenes infrarrojas? Esto es, aquel que asigna colores cercanos al rojo (cálidos) a los valores de intensidad elevada y colores cercanos al azul (fríos) a los de intensidad baja. Repita el ejercicio anterior, generando una figura similar que represente la paleta de color elegida.

2.2 EJERCICIO 2

Cargue la imagen *'peppers.png'* en una matriz de tres dimensiones. Visualice la componente roja por los procedimientos siguientes:

1. Extraiga la componente roja en una matriz de dos dimensiones y muéstrela en escala de grises. ¿Cómo es posible que la componente R de algunos pimientos naranjas o incluso de la cabeza de ajo sea mayor que la de algunos pimientos rojos? Puede observar las componentes RGB de cada píxel situando el cursor sobre un píxel en particular.

2. Visualícela en pseudocolor, cambiando la paleta de colores (mediante el parámetro *cmap* de *plt.imshow*).

3. Visualícela en rojo, solamente con la componente R. Para ello, ponga a 0 las bandas G y B, y visualice la imagen en el color resultante mediante *plt.imshow*.

3. ESPACIOS DE COLOR: RGB Y HSV

Para pasar de un espacio de color a otro, podemos utilizar las funciones del módulo '*color*' de la librería Scikit-Image. Algunas de estas funciones son las que se mencionan a continuación:

- RGB a HSV: x_hsv = skimage.color.rgb2hsv(x_rgb)
- RGB a escala de grises: x_gray = skimage.color.rgb2gray(x_rgb)
- RGB a LAB: x_lab = skimage.color.rgb2lab(x_rgb)
- HSV a RGB: x_rgb = skimage.color.hsv2rgb(x_hsv)

3.1 EJERCICIO 3

Genere la siguiente imagen. En primer lugar, cree un mapa de 8 colores RGB. A continuación, genere las sub-imágenes cuadradas para cada color y concaténelas, utilizando la función *np.concatenate*.

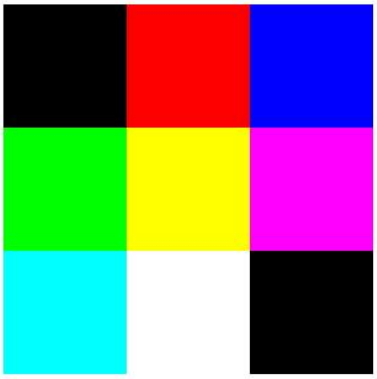

Figura 3.1: Mosaico de colores.

3.2 EJERCICIO 4

El objetivo de este ejercicio es estudiar el espacio de color HSV

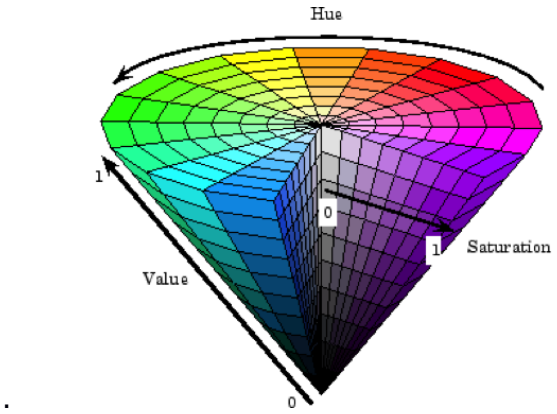

Figura 3.2: Espacio de color HSV.

- ¿Cuáles son las coordinadas del color rojo en una imagen HSV? Construya una imagen roja de tamaño 200x200 en el espacio de color HSV.
- Convierta la imagen a formato RGB. Muéstrela por pantalla utilizando *plt.imshow*. Verifique que obtiene efectivamente las componentes correspondientes al color rojo en RGB.
- Vuelva a la imagen HSV original, pruebe varios valores para S y V, y visualice los resultados.
- Observe siempre las correspondencias entre HSV y RGB. ¿Qué sistema le resulta más intuitivo para definir un color?

3.3 EJERCICIO 5

Cargue la imagen *'peppers.png'*. A continuación, visualice sus componentes R, G y B por separado en una misma figura, utilizando el comando *plt.subplot*.

Repita el ejercicio, visualizando ahora las componentes H, S y V.

4. PSEUDOCOLOR

En una imagen en pseudocolor, el color de cada píxel no es real, sino que se construye a partir de la información captada por uno o más sensores.

En los siguientes ejercicios se estudiará este concepto, utilizando una imagen dermatoscópica multiespectral tomada de [1]. La imagen se compone de 6 bandas espectrales: 3 componentes visibles RGB ('*SL-V.bmp*') y 3 infrarrojas ('SL-IR.bmp').

4.1 EJERCICIO 6

- Cargue y visualice las imágenes visible '*SL-V.bmp*' e infrarroja '*SL-IR.bmp*' en una misma figura. ¿Qué diferencias observa entre ellas?
- Convierta la imagen visible '*SL-V.bmp*' a escala de grises. A continuación, visualícela, en una misma figura, tanto en escala de grises como en pseudocolor, utilizando una paleta con 8 colores diferentes.
- Visualice ahora las componentes visibles e infrarrojas por separado en una misma figura.
- Finalmente, pruebe a sustituir una de las componentes de la imagen visible (G, por ejemplo) por cada una de las componentes infrarrojas. ¿Qué información queda resaltada en la imagen para cada una de ellas?

REFERENCIAS

[1] Lézoray, O., Revenu, M., & Desvignes, M. (2014, October). Graph-based skin lesion segmentation of multispectral dermoscopic images. In 2014 IEEE International Conference on Image Processing (ICIP) (pp. 897-901). IEEE.