Práctica 1: Boletín de Ejercicios

Principios de Visión por Computador

Grado en Inteligencia Artificial

La práctica consiste en la realización de una serie de ejercicios de procesado de imágenes, haciendo uso de técnicas vistas en la asignatura. Los ejercicios del curso se codificarán en **Python 3.x** con las bibliotecas de procesamiento **Numpy**, **SciPy** y **Scikit-image**.

- La práctica deberá realizarse de forma **individual**
- Se deberá **responder a los ejercicios en un Notebook**, mezclando bloques de código y visualización con respuestas a las preguntas en formato Markdown. Se pueden usar ficheros `.py` externos si se considera conveniente.
- No está permitido el uso de versiones de Python 2.x.

Entrega y evaluación

- El alumnado deberá entregar en el Campus Virtual (Moodle de la UDC), en un repositorio habilitado para tal fin:
  - El código de las prácticas y las respuestas a los ejercicios en formato **Notebook** (fichero `.ipynb`), junto con el resto de ficheros `.py` de la práctica.
  - Un fichero de texto (`python_version.txt`) indicando la versión de Python utilizada
  - Un fichero (`requirements.txt`), en formato compatible con **pip**, indicando las librerías utilizadas y sus versiones.
  - El contenido del notebook, con todos los resultados ejecutados y mostrados, guardado en formato PDF (preferiblemente) o HTML.
- El material entregado debe estar subido a la plataforma Moodle. No se aceptarán entregas a través de enlaces a otros servicios de almacenamiento en la nube.
- La fecha límite de entrega será especificada y notificada en el Campus Virtual.
- El alumnado que presente códigos y/o respuestas con indicios de plagio (en cualquiera de las respuestas y con respecto a otras prácticas presentadas este curso o en cursos anteriores, o recursos online) obtendrá una calificación de Suspenso en la parte práctica, (calificado con 0.0), independientemente de la nota que pudiera merecer la calidad de la práctica, e independientemente de si las personas implicadas resultan ser plagiadas o plagiadoras.

Consideraciones previas

1. Se recomienda instalar las **últimas versiones estables** de las bibliotecas aplicables a la versión de Python seleccionada, utilizando un gestor de paquetes (pip o conda).

2. Se recomienda usar un entorno de **Virtual Environment** (e.g. `venv`) que permita gestionar adecuadamente las dependencias de paquetes y reproducir el entorno de ejecución. No usar un Virtual Environment no exime de llevar un control de las versiones de Python y librerías usadas, así como facilitarlas en la entrega.

3. Se recomienda seguir las **guias de estilo estándar PEP 8** (`PEP 8 Style Guide for Python Code`[1]) de codificación en Python que sean aplicables.

Librerías mínimas recomendadas

Para la realización de la práctica se necesitará, como mínimo, instalar y consultar parte de la documentación de las siguientes librerías:

- `Scikit-image`[2]. Librería de procesado de imágenes, basada en **Numpy** y **SciPy**, en la que centraremos las prácticas. Nombre del paquete: `scikit-image`.
- `NumPy`[3]. Librería de operaciones matriciales. Se instalará automáticamente como dependencia de **Scikit-image**.
- `SciPy`[4]. Librería extensible de algoritmos de computación científica basados en **NumPy**. Es de especial relevancia para esta práctica el módulo `scipy.ndimage`[5]. **SciPy** Se instalará automáticamente como dependencia de **Scikit-image**.
- `Matplotlib`[6]. Librería de visualización científica compatible con **NumPy**. Será necesaria para visualizar imágenes y gráficas. También la usaremos para lectura y escritura de imágenes. Nombre del paquete: `matplotlib`.

Virtual Environments y Jupyter Notebooks

El uso de Virtual Environments es habitual en el desarrollo de software usando Python, pues permite gestionar de forma sencilla los siguientes retos:

- Mantener múltiples instalaciones de Python en el sistema con diferentes versiones del lenguaje y diferentes versiones de librerías.
- Poder controlar la versión concreta de Python y de las librerías utilizadas, independientemente de las disponibles a nivel global en el sistema.
- Reproducir el entorno de ejecución exacto en cualquier otro sistema.

Esto es util para nuestro caso concreto, pues permitirá mantener estable el entorno de desarrollo durante todo el curso (independientemente de posibles actualizaciones del sistema), así como establecer mecanismos adecuados para que los profesores puedan ejecutar las prácticas en condiciones reproducibles con respecto a las instalaciones del alumnado.

Para reproducir un entorno Python, típicamente necesitaremos conocer dos cosas:

[1] `PEP 8 Style Guide for Python Code`: https://www.python.org/dev/peps/pep-0008/
[2] `Scikit-image`: http://scikit-image.org/
[3] `NumPy`: https://numpy.org/doc/
[4] `SciPy`: https://scipy.org/
[5] `scipy.ndimage`: https://docs.scipy.org/doc/scipy/reference/ndimage.html
[6] `Matplotlib`: https://matplotlib.org/stable/index.html

Versión de Python. Para la usar una versión concreta de Python necesitaremos disponer de una instalación de Python con esa versión en el sistema. Por lo general, dependiendo del sistema, podremos tener disponibles diferentes versiones con varios nombres (e.g. `python`, `python3`, `python3.6`. `python3.7`. `python3.8`, etc.). Podremos consultar la versión de cualquier intérprete python usando la opción `--version` por linea de comandos. Por ejemplo, para almacenar la versión en un fichero:

```
$ python --version > python_version.txt
```

Lista de librerías y sus versiones. Típicamente gestionaremos las librerías en una instalación de Python usando un gestor de paquetes tipo `pip`[7]. Estos gestores permiten instalar paquetes de uno en uno (`pip install ...`), especificando la lista de nombres. Esto sería lo habitual, a medida que vamos desarrollando y requiriendo funcionalidades. En cualquier caso, siempre podremos obtener una foto fija de la instalación actual usando `pip freeze` y guardarla en disco en un fichero, normalmente llamado `requirements.txt`:

```
$ pip freeze > requirements.txt
```

Y para instalar todos los paquetes especificados en un fichero previamente almacenado (se entiende que en otro sistema), podremos hacerlo mediante:

```
$ pip install -r requirements.txt
```

Configuración y uso de Virtual Environments con `venv`

Python dispone de un entorno estándar de Virtual Environments (`venv`[8]) desde su versión 3.3, que explicaremos a continuación. Si bien, existen opciones equivalentes con librerías alternativas, o en distribuciones Python propietarias como Anaconda. El alumnado puede seguir estas indicaciones o consultar la documentación de los sistemas alternativos no estándar que desee utilizar, aunque muchas de las indicaciones son equivalentes y con pocos cambios.

Crear `venv`. Asumiendo que la versión concreta de python que queremos usar se encuentra disponible en el sistema bajo el nombre `python3.7`, podemos crear un nuevo Virtual Environment vacío, en el directorio actual (e.g. `/GIA/PVC/P1/`), llamado `PVC`.

```
$ cd ~/GIA/PVC/P1
$ python3.7 -m venv pvc
```

Activar y desactivar `venv`. Para entrar, o salir, del Virtual Environment, podemos usar los scripts `activate` y `deactivate`, respectivamente, disponibles en la carpeta local (llamada `pvc`, en este caso). Dentro del Virtual Environment, tanto el comando `python`, como otros asociados a la instalación (e.g. `pip`), y las librerías disponibles, serán aquellas propias del Virtual Environment, y no las del sistema. Por ejemplo, podemos probar `pip freeze` para verificar que no hay paquetes instalados.

[7] `pip`: https://pip.pypa.io/en/stable/
[8] `venv`: https://docs.python.org/3/library/venv.html

```
# Entramos en 'pvc'
$ source pvc/bin/activate
(pvc) $ _

# Ahora estamos dentro de 'pvc'. Veamos, por ejemplo, dónde
↪  está 'python', cual es su versión, dónde está 'pip' y qué paquetes
↪  hay instalados:
(pvc) $ which python
(pvc) $ python --version
(pvc) $ which pip
(pvc) $ pip freeze

# Salimos de 'pvc'
(pvc) $ deactivate
$ _

# Ahora podríamos volver a probar lo anterior, para verificar que ya
↪  estamos, otra vez, en la instalación del sistema
```

Instalación de paquetes en `venv`. Dentro del Virtual Environment, tras haber usado el script `activate`, los comandos de gestión de paquetes tendrán efecto sobre la instalación del `venv`. Es **altamente recomendable**, antes de nada, sobre una instalación limpia, actualizar `pip`, así como instalar `setuptools` y `wheel`. Este último permite acelerar la instalación de paquetes evitando la compilación. A continuación, podemos instalar la lista de paquetes, a mano, por ejemplo `scikit-image` y `matplotlib` (alternativamente mediante un fichero de `requirements.txt`).

```
(pvc) $ pip install -U pip wheel setuptools
# Instalamos las librerías (instalará sus dependencias):
(pvc) $ pip install scikit-image matplotlib
# Alternativametne, si disponemos de un fichero requirements.txt para
↪  reproducir un entorno previo:
(pvc) $ pip install -r requirements.txt
```

Uso de Virtual Environments en Jupyter Notebooks

En general, si tenemos un script de Python que queremos ejecutar en un Virtual Environment, bastará con activar (con `activate`) el entorno, e invocar el comando `python` local.

No obstante, en Jupyter Notebook, el sistema de ejecución es diferente, ya que el entorno Notebook hace uso de "kernels"(IPython Kernel, para ser precisos), sobre los que se delega cada ejecución del programa. Por lo general, los entornos Python del sistema estarán disponibles por defecto en nuestra instalación de Notebook, pero esto no ocurrirá con nuestros Virtual Environments.

Para instalar un Virtual Environment como un Kernel de IPython o Jupyter, debemos activar el entorno, e invocar el comando indicado a continuación.

```
$ cd ~/GIA/PVC/P1/
$ source pvc/bin/activate
(pvc) $ python -m ipykernel install --user --name=Pvc
(pvc) $ deactivate
$ _
```

Tras la instalación es posible salir del entorno, y tendremos un nuevo Kernel disponible en Notebook, en este caso con el nombre "Pvc". Encontraremos el Kernel en el menú **Kernel > Change kernel > ...** de la aplicación.

El paso de instalación del kernel solamente será necesario hacerlo una vez, para que Jupyter tenga constancia de este Virtual Environment y lo pueda usar. Los cambios efectuados en el Virtual Environment, como pueden ser instalación de paquetes nuevos, tendrán efecto directo sobre este kernel. Aunque será necesario reiniciar el kernel para que surtan efecto.

Podemos ver y administrar la lista de kernels disponibles en Jupyter usando la utilidad `jupyter kernelspec` (seguido por e.g. `--help`, `list`, `uninstall <name>`, etc.). Por ejemplo:

```
$ jupyter kernelspec list
```

1. Entrada, salida y representación de imágenes

Entrada/salida recomendada

La librería **Scikit-image**, cuenta con un módulo de entrada, salida y visualización de datos de imagen `skimage.io`[9].

Es relevante tener en cuenta que el módulo **skimage.io** utiliza diferentes implementaciones de cada una de sus funciones, basándose en librerías externas que sirven de *plugins*, cuya documentación debemos consultar para comprender las funcionalidades avanzadas. Podemos ver las opciones disponibles, así como la preferencia para cada función, mediante:

```
import skimage.io as io
# Lista de plugins disponibles (no necesariamente instalados):
io.find_available_plugins()
# Lista de plugins instalados para cada función:
io.plugin_order()
```

En esta práctica, recomendamos el uso de `skimage.io` para leer (`io.imread`[10]) y escribir (`io.imsave`[11]) ficheros de imagen, que por defecto estarán soportados por la librería `imageio`[12]. Para la visualización de imágenes usaremos `io.imshow`[13], que por defecto estará implementada por `pyplot.imshow`[14]. En ocasiones será conveniente usar directamente esta última, o usar sus parámetros específicos, para tener mayor control sobre la visualización.

Representación numérica de imágenes

En **Scikit-image**, así como en otras librerías de procesado de imagen en Python, las imágenes vienen representadas por matrices NumPy (`np.array`[15]). Existen convenciones sobre la forma e índices de las matrices (`np.array.shape`, típicamente `[filas, columnas, canales]` para imágenes 2D), así como sobre el tipo de datos (`np.array.dtype`) y su rango de valores (típicamente con valores restringidos al rango `[0,1]` para tipos flotantes, y al rango completo de enteros positivos para tipos enteros, e.g. `[0,255]` para `np.uint8`). Esto tiene implicaciones relevantes en la forma en la que las funciones de la librería interpretan los datos. Revise la `Guía de Usuario`[16] para obtener información detallada.

Se recomienda el uso de respresentaciones flotantes en el rango `[0, 1]` para hacer operaciones con imágenes, salvo causas debidamente justificadas.

Imágenes de prueba

A lo largo de la práctica, usaremos imágenes de prueba de tres fuentes distinas:

a) Leyendo ficheros de imagen, típcamente en formato `.png`, no comprimido.

b) Construyendo imágenes artificiales mediante manipulación directa de matrices.

c) Invocando las funciones del repositorio de pruebas de **Sciki-image** (`skimage.data`[17]).

[9] `skimage.io`: https://scikit-image.org/docs/dev/api/skimage.io.html
[10] `io.imread`: https://scikit-image.org/docs/dev/api/skimage.io.html#skimage.io.imread
[11] `io.imsave`: https://scikit-image.org/docs/dev/api/skimage.io.html#skimage.io.imsave
[12] `imageio`: https://imageio.readthedocs.io/en/stable/reference/userapi.html
[13] `io.imshow`: https://scikit-image.org/docs/dev/api/skimage.io.html#skimage.io.imshow
[14] `pyplot.imshow`: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html
[15] `np.array`: https://numpy.org/doc/stable/reference/generated/numpy.array.html
[16] `Guía de Usuario`: https://scikit-image.org/docs/dev/user_guide.html
[17] `skimage.data`: https://scikit-image.org/docs/stable/api/skimage.data.html

Figura 1: lena.png.

Ejercicio 1.1.- Leer y visualizar imágenes

a) Lea la imagen 'lena.png' (Figura 1); llamémosle `lena1`. Escriba por pantalla su tamaño (`shape`), sus valores mínimo y máximo, y su tipo. Visualice la imagen.

b) Lea la imagen 'lena.png', usando el parámetro '`as_gray=True`'; llamémosle `lena2`. Escriba por pantalla su tamaño, sus valores mínimo y máximo, y su tipo. Visualice la imagen, consiguiendo que se vea en escala de grises.

c) Divida el valor de la imagen `lena2` por 2 y súmele 0,25; llamémosle `lena3` al resultado. Escriba por pantalla sus valores mínimo y máximo. Visualice en subfiguras anexas `lena2` y `lena3`, usando escala de grises, y consiguiendo que las diferencias de brillo sean apreciables visualmente.

Ejercicio 1.2.- Escribir imágenes en disco

a) Guarde las imágenes `lena2` y `lena3` del Ejercicio 1.1, en los ficheros `lena2.png` y `lena3.png`, respectivamente, de la carpeta `resultados`. Asegúrese de que la diferencia de intesidades es apreciable en las imágenes guardadas (con un visor externo, o volviendolas a leer).

b) Repita la operación anterior, pero multiplicando antes las imágenes por 10, y almacenando los resultados en `lena2b.png` y `lena3b.png`, respectivamente. ¿Qué ocurre y por qué? ¿Cómo debemos hacer para que las diferencias en las imágenes se puedan almacenar?

Ejercicio 1.3.- Enteros, Flotantes y Booleanos

Usando la siguiente función, que crea imágenes de degradado en escala de grises, con un tamaño, tipo y rango dados:

```
import numpy as np

def intensity_gradient(size=(100, 100), dtype=np.uint8,
        values=(0, 255)):
    rows, cols = size
    mn, mx = values
    result = np.linspace(mn, mx, rows*cols).reshape(rows, cols)

    return result.astype(dtype)
```

a) Cree tres imágenes de degradado, de tamaño 100×100, y:

`A:` tipo `np.float64`, y valores de `0.1` a `0.9`.

`B:` tipo `np.uint8`, y valores de `25` a `230`.

`C:` tipo `np.int32`, y valores de `-230` a `230`.

b) Visualice las tres imágenes, en escala de grises y sin normalizar, y reporte su valor mínimo y máximo.

c) Explore el resultado de las funciones `img_as_float`, `img_as_uint`, `img_as_int` e `img_as_bool` de `skimage` sobre las tres imágenes anteriores, así como la salida de la función `skimage.dtype_limits` aplicada sobre las imágenes de entrada y salida. Explique lo que ocurre.

Ejercicio 1.4.- Indexación

Considerando las imágenes `data.brick()` y `data.astronaut()`, convertidas a números flotantes en un rango apropiado:

a) Sobre una copia de `brick`, asigne un valor de intensidad "blanco puro", a la ventana definida por las filas 100 a 150, y las columnas 20 a 120. Visualice el resultado, su valor máximo, su valor mínimo y el tipo.

b) Cree una máscara binaria (llamémosle `brickmsk`) que valga `1` en las posiciones donde `brick` sea mayor que el 50 % del rango dinámico y `0` en el resto. Visualice el resultado, así como su valor máximo, su valor mínimo y el tipo.

c) Modifique los valores de una copia de `astronaut` para tomen un valor `[r,g,b]` aleatorio en las posiciones indicadas por la máscara `brickmsk`[a]

d) Usando una copia de `astronaut`, visualice, en cuatro subfiguras, cada uno de sus canales RGB, en escala de grises, junto con la imagen en color.

e) Usando la función `skimage.draw.disk`[b], obtenga las coordinadas de un círculo en una posición aleatoria del espacio 512×512, y radio 20. Modifique la imagen resultante del apartado (c) para que los valores del canal verde en las posiciones del círculo sean 100 % del rango dinámico, dejando el resto de valores sin cambiar. Visualize el resultado.

[a]Note que puede usar una máscara para indexar una imagen, si coinciden los tamaños de las dimensiones indexadas

[b]`skimage.draw.disk`: https://scikit-image.org/docs/stable/api/skimage.draw.html#skimage.draw.disk

2. Transformación del espacio de color

El módulo `skimage.color`[18] proporciona funciones para la tranformación del espacio de color, como pueden ser: `rgb2gray`, `rgb2lab`, `rgb2hsv`, entre otros.

Ejercicio 2.1.- Espacios de color

Supongamos que queremos segmentar la vasculatura de la imagen `data.retina()`, y segmentar la mariposa de la imagen `butterfly.jpg`. Considerando cada una de estas imágenes:

a) Muestre en una cuadrícula de 3×3 de subfiguras: los tres canales de RGB en escala de grises, los tres canales de CIE-L*a*b* en escala de grises, y los tres canales de HSV en escala de grises. Use los títulos de las figuras para indicar el nombre del canal y su rango de valores.

b) Observando los resultados: ¿Qué canal es más apropiado para la segmentación objetivo? Razone la respuesta.

c) Compare diferentes canales de gris calculado a partir de las siguientes opciones: media de RGB, V de HSV, L* de CIE-L*a*b*.

d) Realizar la misma comparación pero teniendo en cuenta la saturación de HSV y la saturación calculada a partir de L*a*b*.

e) Calcule los canales oponentes Red-Green y Blue-Yellow, calculados a partir de RGB o usando las componentes a* y b* de CIE-L*a*b*.

3. Histogramas y ajustes de luminosidad y contraste

En **Scikit-image** disponemos del módulo `skimage.exposure`[19], que proporciona funciones para el ajuste de intensidad y contraste de imagen.

La representación del histograma de una imagen es de especial relevancia para analizar la distribución de valores de la misma. Existen diversas utilidades en las librerías utilizadas que nos permitirán obtener dicha representación, como pueden ser: `exposure.histogram`[20],

[18]`skimage.color`: https://scikit-image.org/docs/dev/api/skimage.color.html

[19]`skimage.exposure`: https://scikit-image.org/docs/stable/api/skimage.exposure.html

[20]`exposure.histogram`: https://scikit-image.org/docs/stable/api/skimage.exposure.html#skimage.exposure.histogram

`np.histogram`[21], `ndi.histogram`[22], etc. o `plt.hist`[23] para visualización.

Ejercicio 3.1.- Representación de histogramas

Considerando la imagen `data.camera()`, represéntela en escala de gris junto con su histograma.

a) Calcule el histograma de brillo de las imágenes en escala de grises.

b) Represente el histograma con diferentes números de bins: 10 y 100.

c) Haga lo mismo para diferentes ventanas de la imagen. Represente visualmente la imagen general, cuadrados de subventanas y sus histogramas asociados. Seleccione regiones de alto contraste, de bajo contraste, y zonas irregular de más o menos brillo.

d) Repita el ejercicio anterior sumando una constante positiva y negativa: 0,25 y -0,25. Analice de nuevo los histogramas resultantes y sus cambios.

e) Repita el ejercicio anterior multiplicando una constante mayor y menos que 1: 0,5 y 1.5. Analice de nuevo los histogramas resultantes y sus cambios.

4. Representación en frecuencia

La Transformada de Fourier es una herramienta fundamental para analizar la información en frecuencia de una imagen. Permite descomponer una imagen en sus componentes de frecuencia, facilitando el procesamiento en el dominio frecuencial, como el filtrado o la eliminación de ruido. En la librería **Scikit-image**, el módulo `skimage.filters`[24] y las funciones de Fourier de `numpy.fft`[25] proporcionan herramientas para realizar estos análisis.

A continuación, trabajaremos con la transformada de Fourier en una serie de ejercicios que involucran la manipulación y análisis de imágenes en el dominio de la frecuencia.

Ejercicio 4.1.- Transformada de Fourier

Para la imagen A = `data.camera()`:

a) Utilice la función `numpy.fft.fft2`[a] para calcular la Transformada de Fourier de la imagen. Represente la magnitud de la transformada de Fourier de la imagen en escala logarítmica y aplique `numpy.fft.fftshift`[b] para centrar la componente continua de frecuencia.

b) Repita el ejercicio anterior, tras multiplicar la imagen A por una ventana Gaussiana de una anchura σ adecuada, usando `skimage.filters.window`[c], para que se eliminen los componentes de frecuencia asociadas al borde de la imagen (imagen B).

c) Realice una convolución de la imagen B con una Gaussiana de $\sigma = 10$ (imagen C) y represente el resultado, en dominio espacial y en frecuencia.

[21] `np.histogram`: https://numpy.org/doc/stable/reference/generated/numpy.histogram.html
[22] `ndi.histogram`: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.histogram.html
[23] `plt.hist`: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
[24] `skimage.filters`: https://scikit-image.org/docs/stable/api/skimage.filters.html
[25] `numpy.fft`: https://numpy.org/doc/stable/reference/routines.fft.html

d) Calcule la imagen resta D = B - C, y represente el resultado en espacial y frecuencia.

[a] `numpy.fft.fft2`: https://numpy.org/doc/stable/reference/generated/numpy.fft.fft2.html

[b] `numpy.fft.fftshift`: https://numpy.org/doc/stable/reference/generated/numpy.fft.fftshift.html

[c] `skimage.filters.window`: https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.gaussian