

Práctica 1 de Visión Artificial

Grao en Enxeñaría Informática – Computación

Curso 24/25

La práctica consiste en implementar varias operaciones de procesado de imagen vistas en clase con el objetivo de comprender su funcionamiento y sus efectos sobre las imágenes de entrada.

- Las prácticas deberán realizarse de forma **individual**.
- Se utilizará uno de los siguientes lenguajes, o entornos de desarrollo, a elección del alumnado:
 1. **Python 3 + NumPy/SciPy + Scikit-image y/o OpenCV**. Recomendado.
 2. **Matlab, Octave** o herramientas similares.
- **No está permitido** el uso de funciones de tratamiento de imágenes existentes en los entornos de desarrollo en la implementación de las operaciones. En especial se prohíbe usar las funciones que implementan las mismas operaciones o cualquier parte sustancial de ellas.
- **Está permitido** el uso de cualquier librería para leer, escribir y visualizar imágenes, para transformar las imágenes al formato de entrada de las operaciones, o para cubrir cualquier otra necesidad en el código de pruebas. También está permitido usar cualquier función no relacionada con el procesado de imagen (estadística, ordenación, indexación de matrices, etc.) tanto en el código de prueba como en las operaciones implementadas.

1. Consideraciones previas

Los métodos implementados tendrán imágenes como entrada y salida. Asumimos que estas imágenes son matrices que cumplen:

- Tienen un único valor de intensidad por cada punto (escala de grises).
- El tipo de datos es de punto flotante de doble o simple precisión.
- El valor de intensidad se encuentra, como norma general¹, en el rango $[0, 1]$.
- El tamaño de la imagen $M \times N$ es arbitrario y en general $M \neq N$.

Cualquiera de los dos entornos de desarrollo propuestos es capaz de leer, escribir y visualizar imágenes de este tipo, así como transformar otras representaciones de imagen a esta. El alumnado debe ser capaz de crear código de prueba que permita leer y convertir imágenes de entrada y salida a este formato, para poder usarlas en las llamadas a las operaciones implementadas, así como visualizar y guardar en disco los resultados. Es conveniente, verificar que las imágenes se guardan en disco correctamente.

¹Algunas operaciones podrían dar como salida valores fuera del rango $[0, 1]$, por ejemplo negativos, como consecuencia de parámetros de entrada concretos. En dicho caso, la responsabilidad de la renormalización al rango $[0, 1]$, para su correcta visualización y almacenamiento, recaería sobre el usuario de la operación. Es decir, salvo que se especifique lo contrario, las operaciones no deben renormalizar la salida internamente para ajustarla a $[0, 1]$.

2. Entrega y evaluación

- El alumnado deberá **entregar el código en UDCOnline** con fecha límite por especificar.
 - El alumnado que presente códigos con indicios de plagio (en cualquiera de las operaciones y con respecto a otras prácticas presentadas este curso o en cursos anteriores) obtendrá una calificación de Suspenso (calificado con 0.0) en la práctica, independientemente de la nota que pudiera merecer la calidad de la práctica.
- La **evaluación se realizará mediante defensa ante el profesorado de prácticas**.
 - En la defensa se facilitarán imágenes de prueba para varias de las funciones implementadas. El alumnado debe ser capaz de leer las imágenes de entrada, adaptarlas a la especificación de las funciones, usarlas como entrada y visualizar correctamente la salida obtenida. El resultado obtenido debe ser correcto, así como tener capacidad para analizar y comprender por qué es así o no.

3. Operaciones a implementar

3.1. Histogramas: mejora de contraste

- Implementar un algoritmo de **alteración del rango dinámico** de la imagen que permita hacer una compresión (o estiramiento) lineal de histograma mediante la introducción de nuevos límites inferior y superior.

```
outImage = adjustIntensity (inImage, inRange=[], outRange=[0 1])
```

inImage: Matriz MxN con la imagen de entrada.

outImage: Matriz MxN con la imagen de salida.

inRange: Vector 1x2 con el rango de niveles de intensidad [**imin**, **imax**] de entrada. Si el vector está vacío (por defecto), el mínimo y máximo de la imagen de entrada se usan como **imin** e **imax**.

outRange: Vector 1x2 con el rango de niveles de intensidad [**omin**, **omax**] de salida. El valor por defecto es [0 1].

- Implementar un algoritmo de **ecualización de histograma**.

```
outImage = equalizeIntensity (inImage, nBins=256)
```

inImage, outImage: ...

nBins: Número de bins utilizados en el procesamiento. Se asume que el intervalo de entrada [0 1] se divide en **nBins** intervalos iguales para hacer el procesamiento, y que la imagen de salida vuelve a quedar en el intervalo [0 1]. Por defecto 256.

3.2. Filtrado espacial: suavizado

- Implementar una función que permita realizar un **filtrado espacial mediante convolución** sobre una imagen con un kernel arbitrario que se le pasa como parametro.

```
outImage = filterImage (inImage, kernel)
```

inImage, outImage: ...

kernel: Matriz $P \times Q$ con el kernel del filtro de entrada. Se asume que la posición central del filtro está en $(\lfloor P/2 \rfloor + 1, \lfloor Q/2 \rfloor + 1)$.

- Implementar una función que calcule un **kernel Gaussiano unidimensional** con σ dado.

```
kernel = gaussKernel1D (sigma)
```

sigma: Parámetro σ de entrada.

kernel: Vector $1 \times N$ con el kernel de salida, teniendo en cuenta que:

- El centro $x = 0$ de la Gaussiana está en la posición $\lfloor N/2 \rfloor + 1$.
- N se calcula a partir de σ como $N = 2\lceil 3\sigma \rceil + 1$.

- Implementar una función que permita realizar un **suavizado Gaussiano bidimensional** usando un filtro $N \times N$ de parámetro σ , donde N se calcula igual que en la función anterior.

```
outImage = gaussianFilter (inImage, sigma)
```

inImage, outImage, sigma: ...

NOTA. Como el filtro Gaussiano es lineal y separable podemos implementar este suavizado simplemente convolucionando la imagen, primero, con un kernel Gaussiano unidimensional $1 \times N$ y, luego, convolucionando el resultado con el kernel transpuesto $N \times 1$.

- Implementar el **filtro de medianas bidimensional**, especificando el tamaño del filtro.

```
outImage = medianFilter (inImage, filterSize)
```

inImage, outImage: ...

filterSize: Valor entero N indicando que el tamaño de ventana es de $N \times N$. La posición central de la ventana es $(\lfloor N/2 \rfloor + 1, \lfloor N/2 \rfloor + 1)$.

3.3. Operadores morfológicos

- Implementar los **operadores morfológicos de erosión, dilatación, apertura y cierre para imágenes binarias** y elemento estructurante arbitrario.

```
outImage = erode (inImage, SE, center=[])
outImage = dilate (inImage, SE, center=[])
outImage = opening (inImage, SE, center=[])
outImage = closing (inImage, SE, center=[])

inImage, outImage: ...
SE: Matriz PxQ de zeros y unos definiendo el elemento estructurante.
center: Vector 1x2 con las coordenadas del centro de SE. Se asume que el [0 0] es
la esquina superior izquierda. Si es un vector vacío (valor por defecto), el centro
se calcula como ( $\lfloor P/2 \rfloor + 1$ ,  $\lfloor Q/2 \rfloor + 1$ ).
```

- Implementar el algoritmo de **llenado morfológico de regiones** de una imagen, dado un elemento estructurante de conectividad, y una lista de puntos semilla

```
outImage = fill (inImage, seeds, SE=[], center=[])

inImage, outImage, center: ...
seeds: Matriz Nx2 con N coordenadas (fila,columna) de los puntos semilla.
SE: Matriz PxQ de zeros y unos definiendo el elemento estructurante de conectividad.
Si es un vector vacío se asume conectividad 4 (cruz  $3 \times 3$ ).
```

3.4. Detección de bordes

- Implementar una función que permita obtener las componentes G_x y G_y del **gradiente de una imagen**, pudiendo elegir entre los operadores de **Roberts**, **CentralDiff** (Diferencias centrales de Prewitt/Sobel sin promedio: i.e. $[-1, 0, 1]$ y transpuesto), **Prewitt** y **Sobel**.

```
[gx, gy] = gradientImage (inImage, operator)

inImage: ...
gx, gy: Componentes  $G_x$  y  $G_y$  del gradiente.
operator: Permite seleccionar el operador utilizado mediante los valores: 'Roberts',
'CentralDiff', 'Prewitt' o 'Sobel'.
```

- Implementar el **filtro Laplaciano de Gaussiano** que permita especificar el parámetro σ de la Gaussiana utilizada.

```
outImage = LoG (inImage, sigma)

inImage, outImage: ...
sigma: Parámetro  $\sigma$  de la Gaussiana
```

- Implementar el **detector de bordes de Canny**.

```
outImage = edgeCanny (inImage, sigma, tlow, thigh)
```

`inImage`, `outImage`: ...

`sigma`: Parámetro σ del filtro Gaussiano.

`tlow`, `thigh`: Umbrales de histéresis bajo y alto, respectivamente.