Ejercicio 1

Elige una de las dos imágenes oscuras que te proporcionamos: *calle.png* o *templo.png*. Como puedes observar, se trata de imágenes en color. También puedes utilizar cualquier otra imagen oscura que tengas. Consulta a tu profesor si vas a utilizar una imagen distinta de las proporcionadas.

Intenta mejorar el contraste de la imagen elegida de las siguientes formas:

1. Usando las dos funciones de aclarado que hemos estudiado en la asignatura.
2. Mediante la ecualización del histograma.
3. Mediante la ecualización adaptativa del histograma. En este caso, prueba los valores 2, 4 y 8 para el parámetro *kernel_size*.

Ten en cuenta que las funciones de ecualización del histograma sólo admiten imágenes en niveles de gris. Para aplicarlas sobre imágenes en color tienes dos opciones:

- Programar tú explícitamente que cada función se aplique sobre cada canal de color R, G, B independientemente.
- Crear una nueva función utilizando el decorador *adapt_rgb* con el parámetro *each_channel*. Puedes leer la documentación del ejemplo Adapting gray-scale filters to RGB images. No es necesario leer el ejemplo completo, basta con la parte inicial.

Como resultado final debes visualizar las 7 imágenes en color (original, aclarada 1, aclarada 2, ecualizada, ecualizada adaptativa 2, ecualizada adaptativa 4 y ecualizada adaptativa 8) y, junto a cada una de ellas, los histogramas de sus correspondientes canales R, G, B. No olvides comentar en la memoria los resultados obtenidos, tanto desde el punto de vista de la mejora del contraste en cada caso, como de la influencia del parámetro *kernel_size* en la ecualización adaptativa.

¿Consideras que tiene sentido hacer una ecualización de histograma en una imagen en color aplicando el proceso a cada banda por separado?

```
@adapt_rgb(each_channel)
def equalization_each(image):
    return ski.exposure.equalize_hist(image)


3 usages
@adapt_rgb(each_channel)
def equalization_adaptativa(image, value):
    return ski.exposure.equalize_adapthist(image, kernel_size=value)


1 usage
def rgb(image, axis, fila):
    banda = ["Roja", "Verde", "Azul"]
    for nBanda, color in enumerate(banda):
        h_orig, c_orig = ski.exposure.histogram(image[:, :, nBanda])

        axis[fila, nBanda + 1].bar(c_orig, h_orig, 1.1)
```

equalization_each: Esta función realiza la ecualización del histograma en cada canal de la imagen de entrada. Esta mejora el contraste de las imágenes al expandir el rango de intensidad.

equalization_adaptativa: Esta función realiza la ecualización adaptativa del histograma en cada canal de la imagen de entrada. Opera de manera similar a la ecualización regular, pero se calcula sobre regiones más pequeñas de la imagen.

rgb: Esta función traza histogramas para cada canal (rojo, verde y azul) de la imagen de entrada. Utiliza ski.exposure.histogram para calcular el histograma y luego lo traza usando axis.bar.

```
img_original = ski.io.imread("images/calle.png")
h_orig, c_orig = ski.exposure.histogram(img_original)

img_real = ski.util.img_as_float(img_original) * 255

img_clara = np.sqrt(255 * img_real)
img_clara = ski.util.img_as_ubyte(img_clara / 255)
h_clara, c_clara = ski.exposure.histogram(img_clara)

img_clara_2 = np.cbrt(255 ** 2 * img_real)
img_clara_2 = ski.util.img_as_ubyte(img_clara_2 / 255)
h_clara_2, c_clara_2 = ski.exposure.histogram(img_clara_2)

img_eq = equalization_each(img_real / 255)
img_eq = ski.util.img_as_ubyte(img_eq)
h_eq, c_eq = ski.exposure.histogram(img_eq)

img_eq_adapt_2 = equalization_adaptativa(img_real / 255, value: 2)
img_eq_adapt_2 = ski.util.img_as_ubyte(img_eq_adapt_2)
h_eq_adapt_2, c_eq_adapt_2 = ski.exposure.histogram(img_eq_adapt_2)

img_eq_adapt_4 = equalization_adaptativa(img_real / 255, value: 4)
img_eq_adapt_4 = ski.util.img_as_ubyte(img_eq_adapt_4)
h_eq_adapt_4, c_eq_adapt_4 = ski.exposure.histogram(img_eq_adapt_4)

img_eq_adapt_8 = equalization_adaptativa(img_real / 255, value: 8)
img_eq_adapt_8 = ski.util.img_as_ubyte(img_eq_adapt_8)
h_eq_adapt_8, c_eq_adapt_8 = ski.exposure.histogram(img_eq_adapt_8)
```

En este tramo de código, se obtienen las diferentes imágenes que nos piden: la aclarada 1 y 2, la ecualizada y las ecualizadas adaptadas.

```python
fig, axs = plt.subplots( nrows: 7,  ncols: 4, layout="constrained")
imagenes = [img_original, img_clara, img_clara_2, img_eq, img_eq_adapt_2, img_eq_adapt_4, img_eq_adapt_8]
nombres = ["img_orig", "img_clara", "img_clara_2", "img_eq", "img_eq_adapt_2", "img_eq_adapt_4",
           "img_eq_adapt_8"]

for n, imagen in enumerate(imagenes):
    axs[n, 0].imshow(imagen, cmap=plt.cm.gray)
    rgb(imagen, axs, n)
    img = f'images/{nombres[n]}.png'
    ski.io.imsave(fname=img, arr=imagen)

axs_lineal = axs.ravel()
for i in range(0, axs_lineal.size, 4):
    axs_lineal[i].set_axis_off()
    axs_lineal[i + 1].set_xticks([0, 64, 128, 192, 255])
    axs_lineal[i + 2].set_xticks([0, 64, 128, 192, 255])
    axs_lineal[i + 3].set_xticks([0, 64, 128, 192, 255])

plt.show()
```

Aquí podemos ver cómo se realiza un bucle sobre las imágenes, donde se itera sobre cada imagen en una lista llamada imágenes. Se muestra cada imagen en la posición correspondiente de un conjunto de subgráficos (axs) utilizando imshow. Y se llama a la función rgb para trazar los histogramas de color de la imagen actual en los subgráficos. Luego se guarda cada imagen en el disco en formato PNG en la carpeta images, utilizando los nombres de archivo proporcionados en una lista llamada nombres.

Por último, se ajustan los ejes de los subgráficos para ocultar algunos de ellos y configurar los valores de las marcas en los subgráficos restantes. Y se muestra el gráfico completo que contiene las imágenes y sus histogramas.

Con respecto a la pregunta, sí tiene sentido realizar la ecualización de histograma en una imagen en color aplicando el proceso a cada banda por separado. La razón principal es que cada canal de color representa diferentes aspectos de la imagen, y ecualizar cada canal por separado puede ayudar a mejorar el contraste y la calidad de la imagen de manera más específica en cada componente de color, especialmente cuando hay problemas de iluminación desigual o cuando se busca mejorar el contraste en áreas específicas de la imagen.

Sin embargo, es posible que se necesite un ajuste adicional para mantener la consistencia de color si es necesario.

Ejercicio 2

Queremos visualizar la misma imagen que has elegido en el ejercicio anterior, pero ahora en niveles de gris. Para ello, utiliza la función*rgb2gray* para obtener una imagen en niveles de gris para cada una de las 7 imágenes en color visualizadas en el ejercicio anterior. Muestra las 7 imágenes en niveles de gris junto con sus correspondientes histogramas y comenta los resultados obtenidos.

```python
for n, imagen in enumerate(nombres):
    # Cargar la imagen original
    img_original = ski.io.imread(f'images/{imagen}.png')

    # Convertir la imagen a niveles de gris
    img_gray = rgb2gray(img_original)
    img_gray = ski.util.img_as_ubyte(img_gray)

    # Calcular el histograma de la imagen en niveles de gris
    h_gray, c_gray = ski.exposure.histogram(img_gray)

    axs[n, 0].imshow(img_gray, cmap=plt.cm.gray)

    # Mostrar el histograma en la segunda columna de la fila actual
    axs[n, 1].bar(c_gray, h_gray, 1.1)
```

Algo a destacar en el código siguiente, sería que se lleva a cabo un bucle que carga imágenes desde el disco utilizando los nombres de archivo proporcionados en la lista nombres. Luego realiza las siguientes operaciones para cada imagen:

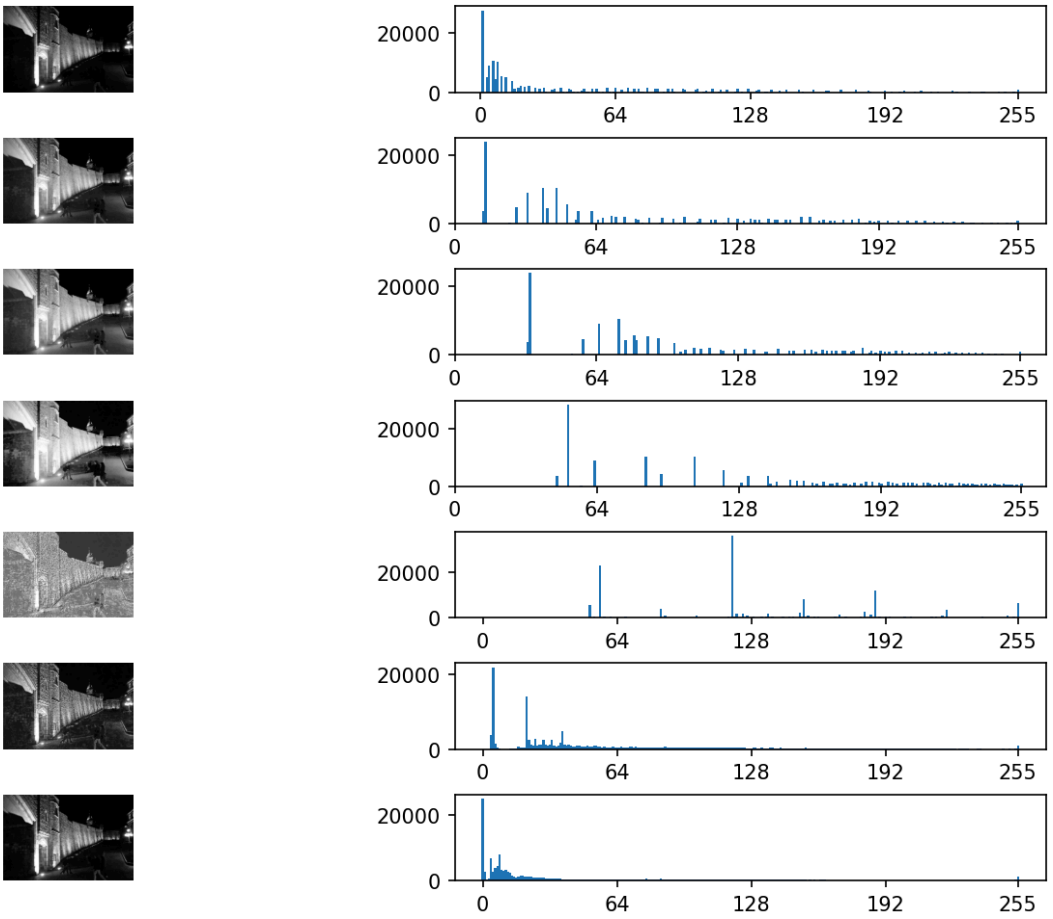
***Cargar la imagen original**: Lee la imagen con skimage.io.imread().

***Convertir la imagen a niveles de gris**: Utiliza la función rgb2gray() para convertir la imagen a una escala de grises. Después, normaliza los valores de píxeles utilizando ski.util.img_as_ubyte() para asegurarse de que estén en el rango.

***Calcular el histograma de la imagen en niveles de gris**: Utiliza ski.exposure.histogram() para calcular el histograma de la imagen

***Mostrar la imagen y el histograma**: Utiliza imshow() para mostrar la imagen en niveles de gris en la primera columna de la fila actual de subgráficos (axs[n, 0]). Luego, utiliza bar() para trazar el histograma en la segunda columna de la misma fila (axs[n, 1]).

A continuación, tenemos los resultados, de los cuales podemos decir que los resultados obtenidos son bastantes similares al del ejercicio anterior, pero hay que tener en cuenta



Ejercicio 3

Vamos a repetir el ejercicio anterior, pero con un enfoque distinto. A partir de la imagen original en color, obtén una versión en niveles de gris mediante la función *rgb2gray*. Trata de mejorar el contraste de esta imagen en niveles de gris empleando las mismas funciones de aclarado y ecualización que has usado en el ejercicio 1. Muestra las 7 imágenes obtenidas junto con sus correspondientes histogramas y comenta los resultados obtenidos.

Los resultados obtenidos en este caso son parecidos a los obtenidos en el ejercicio 2, pero ¿son exactamente iguales? Razona tu respuesta.

¿Qué proceso te parece más acertado para obtener una imagen en niveles de gris en la que hayamos mejorado el contraste? ¿El utilizado en los ejercicios 1 y 2 o el utilizado en el ejercicio 3? Justifica tu respuesta.

```
for i, image in enumerate(images):
    axs[i, 0].imshow(image, cmap=plt.cm.gray)
    h_gray, c_gray = ski.exposure.histogram(image)
    axs[i, 1].bar(c_gray, h_gray, 1.1)
```

Algo a destacar de este código es que para poder tratar las imágenes, cada una de estas se ha añadido a una lista, de modo que luego se cree un bucle para poder visualizar cada imagen en escala de grises, calcular su histograma correspondiente y posteriormente, mostrarlo.

A la hora de analizar los resultados podemos decir que a simple vista no podemos apreciar mucho cambio entre ambos resultados, por lo que hay que fijarse en los detalles de la imagen que buscamos, puesto que ambos enfoques son adecuados. En el caso de aclarar y mejorar el contraste antes de la conversión a grises, podemos decir que afirmar que si deseamos mejorar el contraste de la imagen en general este es el método adecuado. Pero existe un riesgo de que la mejora del contraste afecte a la relación del tono entre los diferentes canales de color, lo que podría resultar en una conversión a escala de grises que no refleje fielmente la percepción del color original.

En el caso de la conversión a escala de grises primero, luego edición de la imagen, sabemos que puede ser beneficioso si se desea conservar la relación de los tonos entre los diferentes canales de color, lo que podría resultar en una mejor representación del color en la imagen final en escala de grises. Sin embargo, puede haber casos en los que se pierda información del color durante la conversión a escala de grises, especialmente si la imagen original tiene un rango de colores muy amplio.

Con esto a su vez, podemos concluir que los resultados entre los ejercicios 2 y 3 aunque se parecen no son iguales, pues se realiza el proceso de una forma diferente.

Ejercicio 4

A partir de la imagen en color del ejercicio 1, mejora su contraste de las siguientes formas:

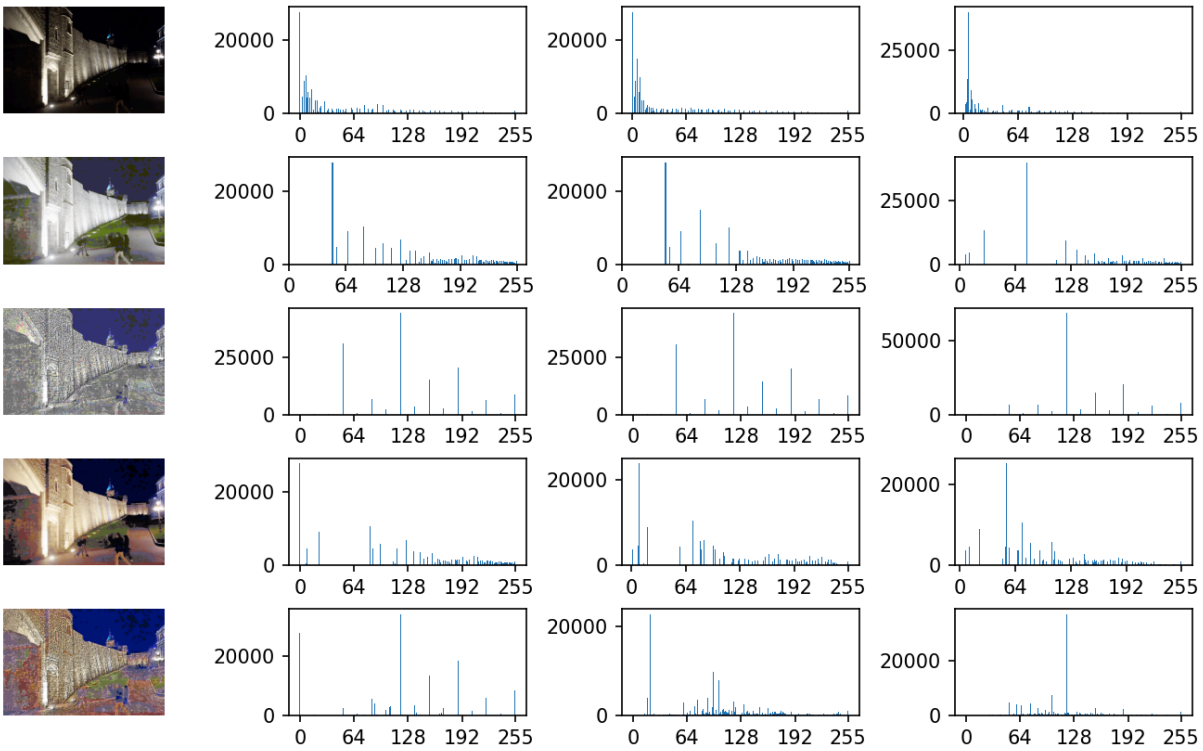
- Aplicando el proceso de ecualización a cada banda de color. En este caso, debes crear una nueva función utilizando el decorador *adapt_rgb* con el parámetro *each_channel*, como se propuso en el ejercicio 1. Si ya lo hiciste así en ese ejercicio, solo tienes que reutilizar tu código.
- Aplicando también el proceso de ecualización y creando una nueva función utilizando el decorador *adapt_rgb*, pero ahora con el parámetro *hsv_value*.

Muestra la imagen original y las dos imágenes ecualizadas. ¿Qué resultado te parece más adecuado? Para entender lo que hace el parámetro *hsv_value*, acaba de leer el ejemplo Adapting gray-scale filters to RGB images o consulta a tu profesor.

```
@adapt_rgb(hsv_value)
def equalization_hsv(image):
    return ski.exposure.equalize_hist(image)


1 usage
@adapt_rgb(hsv_value)
def equalization_adaptativa_hsv(image, value):
    return ski.exposure.equalize_adapthist(image, kernel_size=value)
```

Lo más destacable de este código es el uso de hsv_value, puesto que el código es prácticamente el mismo que en el ejercicio 1.



En mi caso, además de la ecualización normal, también realice la adaptativa, de modo que muestro 5 imágenes en vez de 3. Con la ecualización de histograma por canal (equalization_each), se mejora el contraste de cada canal de color individualmente, lo que puede ser útil si hay desequilibrios de contraste entre los canales de color en la imagen. Visualmente tiene mejor aspecto esta imagen, aunque se ve muy clara, por lo que pierde el color un poco.

Con la ecualización mediante HSV (hsv_value), se ajusta el contraste en función de la distribución de intensidades locales en la imagen, lo que puede ser beneficioso para mejorar el contraste en áreas con diferentes niveles de iluminación. Visualmente con este efecto la imagen se ve realmente saturada, aunque los colores se ven más similares a los originales.

Estudiante: Irene Mejía Hurtado
Profesor: Pedro García Sevilla