UNIVERSIDAD DE ANTIOQUIA

LABORATORIO 2 FILTRADO DE IMAGENES
PROCESAMIENTO DIGITAL DE IMÁGENES

WILLIAM ALEXANDER TORRES ZAMBRANO
ESTUDIANTE

SEBASTIÁN GUZMÁN OBANDO
PROFESOR

FACULTAD DE INGENIERÍA
INGENIERÍA DE SISTEMAS
SECCIONAL OCCIDENTE - ANTIOQUIA

INTRODUCCIÓN

El procesamiento digital de imágenes es una disciplina fundamental en el campo de la visión por computadora y el análisis de imágenes. Este campo de estudio se enfoca en desarrollar técnicas y algoritmos que permiten manipular y mejorar imágenes digitales para diversas aplicaciones. Una de las áreas más importantes dentro del procesamiento de imágenes es el uso de filtros.

Los filtros en el procesamiento digital de imágenes son herramientas esenciales que permiten realizar operaciones específicas sobre una imagen, como suavizar, realzar, resaltar bordes, eliminar ruido, entre otros. Estos filtros son capaces de mejorar la calidad visual de las imágenes, resaltar características relevantes y eliminar elementos no deseados, lo que resulta en una mejora significativa en el análisis y la interpretación de las imágenes.

En este laboratorio del curso de procesamiento digital de imágenes, nos adentraremos en el estudio y aplicación de diferentes filtros en el procesamiento de imágenes. Exploraremos una variedad de filtros ampliamente utilizados en el campo, como el filtro de promedio, el filtro de mediana, el filtro logarítmico, el filtro de cuadro normalizado, el filtro gaussiano, el filtro Laplace, el filtro Sobel y el filtro Canny.

Cada uno de estos filtros tiene características únicas y se aplica en situaciones específicas. A través de ejemplos prácticos, aprenderemos cómo implementar y utilizar estos filtros en el procesamiento de imágenes. Además, exploraremos las ventajas y desventajas de cada filtro y comprenderemos los efectos que tienen en las imágenes procesadas.

El conocimiento adquirido en este estudio nos permitirá aplicar técnicas de procesamiento de imágenes para mejorar la calidad visual, extraer información valiosa y preparar imágenes para su posterior análisis y aplicación en diversos campos, como la medicina, la robótica, la visión artificial, entre otros.

El desarrollo de este laboratorio 2 del curso de procesamiento digital de imágenes nos proporcionará las bases necesarias para comprender y aplicar diferentes filtros en el procesado de imágenes, lo que nos permitirá potenciar y enriquecer nuestras habilidades en este fascinante campo de estudio.

Desarrollo del laboratorio

PARTE 1: Filtros

1) Investigar y definir los siguientes filtros (la investigación debe contener la fórmula matemática y su explicación) “calcular a mano 2 o 3 pixeles a mano”:

- **Filtro de media:** El filtro de media, también conocido como filtro de promedio, es un filtro de suavizado utilizado en el procesamiento digital de imágenes. Su objetivo principal es reducir el ruido y suavizar los detalles de la imagen.

La fórmula matemática del filtro de media para un píxel dado es la siguiente:

$$Valor\ del\ píxel\ filtrado = \frac{Suma\ de\ valores\ de\ los\ píxeles\ vecinos}{Número\ de\ píxeles\ vecinos}$$

El filtro de media se aplica deslizando una ventana de tamaño fijo (por ejemplo, 3x3 o 5x5) sobre la imagen. Cada píxel en el centro de la ventana se reemplaza por el promedio de los valores de los píxeles vecinos que se encuentran dentro de la ventana.

Dada una imagen $f(x, y)$ de tamaño NxN, el valor del nivel de gris de la imagen suavizada $g(x, y)$ en el punto (x, y) se obtiene promediando los valores de nivel de gris de los puntos de f contenidos en una cierta vecindad de (x, y).

$$g(x, y) = \frac{1}{M} \sum_{(n,m)\in S} f(n, m)$$

donde $x, y = 0, 1, \ldots, N - 1$. S es el conjunto de coordenadas de los puntos vecinos a (x, y), incluyendo el propio (x, y), y M es el número de puntos de la vecindad. Por ejemplo, imaginemos la subimagen y la máscara siguientes:

		$\vdots$		
	$(x-1, y-1)$	$(x, y-1)$	$(x+1, y-1)$	
$\ldots$	$(x-1, y)$	(x, y)	$(x+1, y)$	$\ldots$
	$(x-1, y+1)$	$(x, y+1)$	$(x+1, y+1)$	
		$\vdots$		

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

y que queremos reemplazar el valor de $f(x, y)$ por el promedio de los puntos en una región de tamaño 3x3 centrada en (x, y), es decir, queremos asignar el valor promedio a $f(x, y)$:

$$g(x, y) = \tfrac{1}{9}\{f(x-1, y-1) + f(x-1, y) + f(x+1, y-1) + f(x-, y)+$$
$$+f(x, y) + f(x+1, y) + f(x-1, y-1) + f(x, y+1) + f(x+1, y+1)\}$$

Esta operación se puede realizar de forma general centrando la máscara en (x, y) y multiplicando cada punto debajo de la máscara por el correspondiente coeficiente de la máscara y sumando el resultado.

Ejemplo:

Supongamos que una imagen de 5×5 usa 3 bits para cada pixel, y que su representación matricial es la siguiente:

3	3	3	4	4
3	7	3	4	0
4	4	4	5	5
3	3	3	4	4
3	7	3	4	0

Imagen original

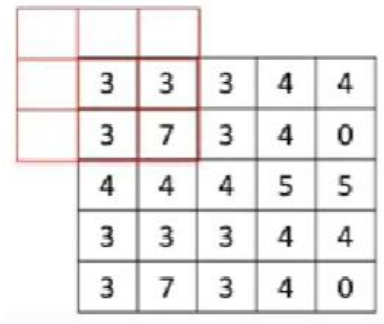
Apliquemos el filtro de la media representado mediante la máscara:

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

¿Cuáles serán los valores de la imagen suavizada?

Solución: para filtrar la imagen, superponemos la máscara teniendo en cuenta que la posición central de la máscara debe coincidir con la posición del píxel que se quiere cambiar:



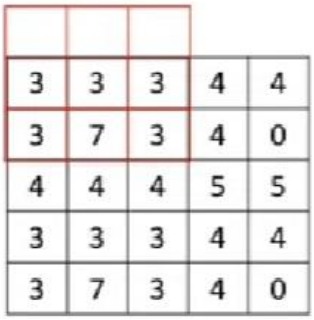
Vamos a calcular el pixel $(0,0)$ de la nueva imagen:

$f(0,0) = 3 \times \frac{1}{9} + 3 \times \frac{1}{9} + 3 \times \frac{1}{9} + 7 \times \frac{1}{9} = 1.77 \approx 2$ (se redondea al entero más cercano)

Por lo tanto, en la nueva imagen en la posición del pixel $(0,0)$ colocamos:

2				

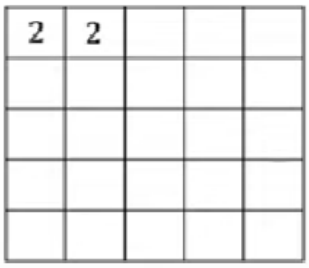
Ahora, calculamos el nuevo valor para el pixel $(1,0)$:



3	3	3	4	4
3	7	3	4	0
4	4	4	5	5
3	3	3	4	4
3	7	3	4	0

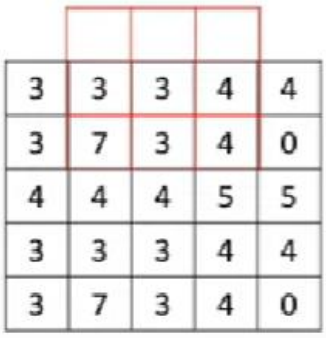
$f(1,0) = 3 \times \frac{1}{9} + 3 \times \frac{1}{9} + 3 \times \frac{1}{9} + 3 \times \frac{1}{9} + 7 \times \frac{1}{9} + 3 \times \frac{1}{9} = 2.44 \approx 2$ (se redondea al entero más cercano)

en la nueva imagen en la posición del pixel $(1,0)$ colocamos:



2	2			

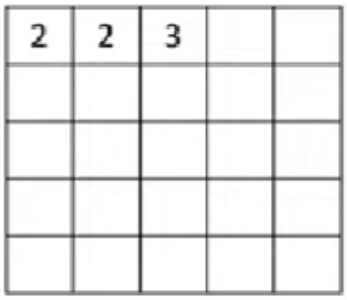
Ahora, calculemos el nuevo valor para el pixel $(2,0)$:



3	3	3	4	4
3	7	3	4	0
4	4	4	5	5
3	3	3	4	4
3	7	3	4	0

$f(2,0) = 3 \times \frac{1}{9} + 3 \times \frac{1}{9} + 4 \times \frac{1}{9} + 7 \times \frac{1}{9} + 3 \times \frac{1}{9} + 4 \times \frac{1}{9} = 2.66 \approx 3$ (se redondea al entero más cercano)

De este modo, en la nueva imagen en la posición del pixel $(2,0)$ colocamos:



2	2	3		

Siguiendo este mismo procedimiento para cada uno de los pixeles de la imagen original, obtenemos los valores para la nueva imagen. En este caso, luego de analizar todos los pixeles obtenemos los valores para la nueva imagen:

2	2	3	2	1
3	4	4	4	2
3	4	4	4	2
3	4	4	4	2
2	2	3	2	

Finalmente, calculemos el nuevo valor para el pixel $(4,4)$:

3	3	3	4	4
3	7	3	4	0
4	4	4	5	5
3	3	3	4	4
3	7	3	4	0

$f(4,4) = 4 \times \frac{1}{9} + 4 \times \frac{1}{9} + 4 \times \frac{1}{9} + 0 \times \frac{1}{9} = 1.33 \approx 1$ (se redondea al entero más cercano)

Finalmente, los valores de los pixeles en la nueva imagen quedan:

2	2	3	2	1
3	4	4	4	2
3	4	4	4	2
3	4	4	4	2
2	2	3	2	1

Imagen nueva

Notemos que en la imagen original, hay un cambio brusco por ejemplo de 4 a 0, veamos:

3	3	3	4	4
3	7	3	4	0
4	4	4	5	5
3	3	3	4	4
3	7	3	4	0

Mientras que, en esas mismas posiciones en la nueva imagen, hay un cambio suavizado continuo del valor 2 a 1:

2	2	3	2	1
3	4	4	4	2
3	4	4	4	2
3	4	4	4	2
2	2	3	2	1

- **Filtro de mediana:** Uno de los principales problemas del método anterior es que difumina los bordes y otros detalles de contraste. Un método alternativo es utilizar filtros de mediana, en los que remplazamos el valor de gris de un punto por la mediana de los niveles de gris de una cierta vecindad. Recordemos que la mediana m de un conjunto de valores es aquel tal que la mitad de los valores del conjunto son menores que m y la otra mitad son mayores que m. La función principal de los filtros de mediana es forzar a los puntos con valores de intesidad muy distintos a sus vecinos a tener valores más próximos a sus vecinos, de modo que se eliminan los picos de intensidad que aparecen en áreas aisladas. Un ejemplo de la utilidad de este tipo de filtro se muestra en el siguiente ejemplo:

Ejemplo: Supongamos que una imagen de 5×5 usa 3 bits para cada pixel, y que su representación matricial es la siguiente:

3	3	3	4	4
3	7	3	4	0
4	4	4	5	5
3	3	3	4	4
3	7	3	4	0

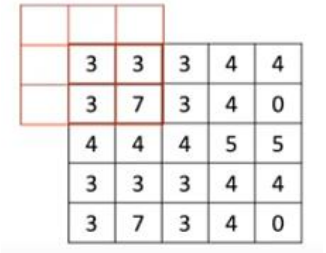
Imagen original

Apliquemos el filtro representado mediante la máscara:

1	1	1
1	1	1
1	1	1

¿Cuáles serán los valores de la imagen suavizada?

Solución:



Hallemos el nuevo valor para el pixel $(0,0)$, para esto vemos que debemos ordenar los números: $3,3,3,7$ de donde la **mediana** de ellos es el valor 3 (que corresponde al promedio de los datos centrales, por ser una cantidad par de datos), de este modo el nuevo valor del pixel en la posición $(0,0)$ es 3

3				

Calculemos ahora el nuevo pixel para la posición $(1,0)$,

3	3	3	4	4
3	7	3	4	0
4	4	4	5	5
3	3	3	4	4
3	7	3	4	0

Buscamos la mediana entre los valores: $3,3,3,3,7,3$. Ordenamos los datos: $3,3,3,3,3,7$, de nuevo la mediana es el promedio de los datos centrales, en este caso la mediana es 3. De este modo el nuevo valor del pixel $(1,0)$ es 3

3	3			

Y así sucesivamente hasta obtener el nuevo valor para cada pixel de la nueva imagen.
En este ejemplo esos nuevos valores son:

3	3	4	4	4
4	3	4	4	4
4	3	4	4	4
4	3	4	4	4
3	3	4	4	4

- **Filtro logarítmico:** es una técnica utilizada en el procesamiento digital de imágenes para mejorar el contraste en imágenes de baja iluminación o con un rango dinámico limitado. Este filtro aplica una transformación logarítmica a los valores de los píxeles de la imagen, lo que comprime el rango de valores oscuros y expande el rango de valores claros.

  La transformación logarítmica se aplica punto a punto a cada píxel de la imagen original. La fórmula general para la transformación logarítmica es:

$$V_{new} = c * \log(1 + V_{old})$$

Donde V_{new} es el nuevo valor del píxel después de la transformación, V_{old} es el valor original del píxel, log es la función logarítmica y c es una constante que controla la amplificación del contraste. La constante c se utiliza para ajustar el rango de salida de la transformación logarítmica y suele ser un valor positivo.

La transformación logarítmica tiene un efecto de compresión en los valores oscuros de la imagen, lo que significa que los detalles en las zonas de baja iluminación se vuelven más visibles. Además, expande los valores claros, mejorando la diferenciación y el contraste en áreas de alta iluminación.

El filtro logarítmico es útil para imágenes que contienen una amplia gama de niveles de iluminación, como imágenes astronómicas o fotografías de escenas con luces brillantes y sombras profundas. Al aplicar la transformación logarítmica, se puede realzar la información en las regiones de baja intensidad sin saturar las áreas más brillantes.

Es importante destacar que después de aplicar el filtro logarítmico, es posible que sea necesario realizar una normalización o ajuste adicional para que los valores de los píxeles estén dentro del rango de visualización deseado.

Ejemplo: Supongamos que tenemos una imagen cuyos valores de pixeles son los que se presentan a continuación:

20	40	60
80	100	120
140	160	180

A continuación, usando el filtro logarítmico calculo el valor de cada nuevo pixel usando la expresión:

$$V_{new} = c * \log(1 + V_{old})$$

Donde:

V_{new} es el nuevo valor del pixel después de la transformación

V_{old} es el valor original del pixel

log es la función logarítmica

c es una constante de amplificación del contraste

Supongamos que $c = 1$. Ahora calculemos

el primer pixel o posición $(0,0)$:

$V_{new} = 1 * \ln(1 + 20) = \log(21) \approx 3.04 \approx 3$

Pixel de la posición $(1,0)$:

$V_{new} = 1 * \ln(1 + 40) = \log(41) \approx 3.71 \approx 4$

Pixel de la posición $(2,0)$:

$V_{new} = 1 * \ln(1 + 60) = \log(61) \approx 4.11 \approx 4$

Pixel de la posición $(0,1)$:

$V_{new} = 1 * \ln(1 + 80) = \log(81) \approx 4.39 \approx 4$

Pixel de la posición $(1,1)$:

$V_{new} = 1 * \ln(1 + 100) = \log(101) \approx 4.6 \approx 5$

Pixel de la posición $(2,1)$:

$V_{new} = 1 * \ln(1 + 120) = \log(121) \approx 4.79 \approx 5$

Pixel de la posición $(0,2)$:

$V_{new} = 1 * \ln(1 + 140) = \log(141) \approx 4.95 \approx 5$

Pixel de la posición $(1,2)$:

$V_{new} = 1 * \ln(1 + 160) = \log(161) \approx 5.08 \approx 5$

Pixel de la posición $(2,2)$:

$V_{new} = 1 * \ln(1 + 180) = \log(181) \approx 5.19 \approx 5$

Entonces, la nueva imagen queda:

3	4	4
4	5	5
5	5	5

- **Filtro de cuadro normalizado:** también conocido como filtro de promedio, es otra técnica común en el procesamiento digital de imágenes utilizada para suavizar una imagen y reducir el ruido. A diferencia del filtro gaussiano, el filtro cuadrado normalizado utiliza un kernel que tiene todos sus elementos con un valor igual y constante.

  El kernel del filtro cuadrado normalizado es una matriz cuadrada con dimensiones predefinidas, por ejemplo, 3x3 o 5x5. Cada elemento del kernel tiene el mismo valor, que es el inverso del número total de elementos en el kernel. Por lo tanto, si el kernel es 3x3, cada elemento tendrá un valor de 1/9, y si el kernel es 5x5, cada elemento tendrá un valor de 1/25.

  Al aplicar el filtro cuadrado normalizado, se coloca el kernel sobre cada píxel de la imagen original y se realiza una multiplicación punto a punto entre los valores del kernel y los valores de los píxeles correspondientes. Luego, se suman todos los productos resultantes y se divide por el número total de elementos en el kernel. El resultado se

asigna al píxel central. Este proceso se repite para todos los píxeles de la imagen, generando una nueva imagen filtrada.

El filtro cuadrado normalizado suaviza la imagen al calcular el promedio de los valores de intensidad de los píxeles en el área definida por el tamaño del kernel. Dado que todos los elementos del kernel tienen el mismo valor, el promedio se calcula de manera equitativa y se aplica a todos los píxeles. Esto ayuda a reducir el ruido y las imperfecciones al eliminar las variaciones abruptas en los valores de los píxeles.

A diferencia del filtro gaussiano, el filtro cuadrado normalizado tiende a producir un suavizado más uniforme en la imagen, pero puede generar una pérdida de detalles finos debido a su naturaleza de promedio. La elección del tamaño del kernel determina el grado de suavizado aplicado, siendo un kernel más grande igual a un suavizado más pronunciado.

Ejemplo: Supongamos que tenemos una imagen cuyos valores de pixeles son los que se presentan a continuación:

50	100	150
200	125	75
25	175	225

Calculemos el valor de los pixeles de la nueva imagen si aplicamos el filtro cuadro normalizado. La ecuación para calcular los nuevos valores de los píxeles utilizando el filtro de cuadro normalizado es:

$$V_{new} = (V_{old} - V_{mín}) * \frac{255}{V_{máx} - V_{mín}}$$

Donde V_{new} es el nuevo valor del píxel, V_{old} es el valor original del píxel, $V_{mín}$ es el valor mínimo de píxel en la imagen y $V_{máx}$ es el valor máximo de píxel en la imagen.

En este caso, $V_{mín} = 25$ y $V_{máx} = 225$, ya que esos son los valores mínimos y máximos en la matriz de píxeles.

Luego, el nuevo valor del pixel $(0,0)$ es:

$$V_{new} = (50 - 25) * \frac{255}{225 - 25} = 25 * \frac{255}{200} \approx 31.87 \approx 32$$

Calculamos el valor del pixel $(1,0)$ es:

$$V_{new} = (100 - 25) * \frac{255}{225 - 25} = 75 * \frac{255}{200} \approx 95.62 \approx 96$$

Calculamos el valor del pixel $(2,0)$ es:

$$V_{new} = (150 - 25) * \frac{255}{225 - 25} = 75 * \frac{255}{200} \approx 159.38 \approx 159$$

Con un procedimiento similar se continúa con los demás pixeles.

- **Filtro gaussiano:** El filtro gaussiano se basa en la convolución de la imagen original con un kernel gaussiano. Un kernel gaussiano es una matriz que contiene valores calculados a partir de la distribución Gaussiana. Cada valor en el kernel representa el peso que se asigna a los píxeles de la imagen original durante la convolución.

  Cuando se aplica el filtro gaussiano, se coloca el kernel sobre cada píxel de la imagen original y se realiza una multiplicación punto a punto entre los valores del kernel y los valores de los píxeles correspondientes. Luego, se suman todos los productos resultantes y se asigna el resultado al píxel central. Este proceso se repite para todos los píxeles de la imagen, generando una nueva imagen filtrada.

  El filtro gaussiano tiene la propiedad de difuminar suavemente los detalles de alta frecuencia en una imagen, lo que ayuda a eliminar el ruido y las imperfecciones. Esto se debe a que la distribución Gaussiana tiene una respuesta suave y decaimiento gradual desde el centro hacia los extremos, lo que resulta en una mezcla suave de valores vecinos.

  El nivel de suavizado o difuminado aplicado por el filtro gaussiano depende del tamaño del kernel utilizado. Un kernel más grande produce un suavizado más pronunciado, mientras que un kernel más pequeño produce un suavizado más suave. La elección del tamaño del kernel se basa en la cantidad de ruido presente en la imagen y en el grado de suavizado deseado.

  La fórmula general para aplicar el filtro gaussiano es la siguiente:

$$V_{new} = \frac{1}{2\pi\sigma^2} * e^{\frac{-(x^2+y^2)}{2\sigma^2}} * V_{original}$$

Donde:

V_{new} es el nuevo valor del píxel después de aplicar el filtro gaussiano.
$V_{original}$ es el valor original del píxel en la matriz.
x y y representan las coordenadas de desplazamiento del píxel en la matriz.
σ es el parámetro que controla la desviación estándar de la distribución gaussiana.

Ejemplo: Suponga que se tiene una imagen con los siguientes pixeles:

10	20
30	40

Con sigma = 1. Hallar el valor de los nuevos pixeles si se aplica filtro gaussiano.

Solución: el valor del kernel gaussiano es:

0.075	0.124	0.075
0.124	0.204	0.124
0.075	0.124	0.075

Ahora, vamos a realizar la convolución de la matriz original con el kernel gaussiano. Para cada píxel en la matriz original, multiplicamos los valores correspondientes del kernel gaussiano y sumamos los resultados.

Para el píxel en la posición (0, 0) de la matriz original:

Nuevo valor del píxel (0, 0):
(10 * 0.204) + (20 * 0.124) + (30 * 0.124) + (40 * 0.075) = 19.22

Para el píxel en la posición (0, 1) de la matriz original:
Nuevo valor del píxel (0, 1):
(10 * 0.124) + (20 * 0.075) + (30 * 0.075) + (40 * 0.124) = 16.28

Para el píxel en la posición (1, 0) de la matriz original:
Nuevo valor del píxel (1, 0):
(10 * 0.124) + (20 * 0.204) + (30 * 0.204) + (40 * 0.124) = 31.56

Para el píxel en la posición (1, 1) de la matriz original:
Nuevo valor del píxel (1, 1):
(10 * 0.075) + (20 * 0.124) + (30 * 0.124) + (40 * 0.075) = 23.72

Finalmente, creamos la matriz resultante con los nuevos valores de píxeles obtenidos:

19.22	16.28
31.56	23.72

- **Filtro Laplace:** es una técnica utilizada en el procesamiento de imágenes para realzar los bordes y detalles de una imagen. Este filtro se basa en el cálculo de la segunda derivada de la imagen, lo que resalta los cambios abruptos de intensidad en la imagen, como los bordes.

  El filtro Laplaciano se puede aplicar utilizando una máscara o kernel, que es una matriz de números que se utiliza para realizar operaciones de convolución en la imagen. La máscara típica para el filtro Laplaciano es la siguiente:

0	1	0
1	−4	1
0	1	0

Para aplicar el filtro Laplaciano, se realiza la convolución de la máscara con la imagen. Esto implica deslizar la máscara sobre cada píxel de la imagen y realizar la multiplicación de los valores de la máscara con los valores correspondientes de los píxeles de la imagen. Luego, se suma el resultado de todas las multiplicaciones y se coloca en el píxel central de la máscara en la posición correspondiente en la imagen de salida.

El resultado de aplicar el filtro Laplaciano es una imagen en la que los bordes y detalles se realzan, ya que los cambios abruptos de intensidad se acentúan. Sin embargo, el filtro Laplaciano también resalta el ruido de alta frecuencia en la imagen, por lo que a menudo se utiliza en combinación con técnicas de suavizado para reducir el ruido antes de aplicar el filtro.

Ejemplo:

Supongamos que una imagen de 4×4 usa 3 bits para cada pixel, y que su representación matricial es la siguiente:

3	0	5	0
3	0	1	0
4	0	0	0
7	0	7	0

Hallemos la nueva imagen usando el filtro Laplaciano.

Solución:

Comenzamos por hallar el pixel $(0,0)$,

Máscara

0	1	0
1	−4	1
0	1	0

	3	0	5	0
	3	0	1	0
	4	0	0	0
	7	0	7	0

$f(0,0) = 3 \times (-4) + 0 \times 1 + 3 \times 1 + 0 \times 0 = -9$, este valor lo colocamos en la nueva imagen en la posición $(0,0)$, así

-9			

Supongamos que ahora vamos a buscar el pixel de la posición $(1,0)$:

Máscara

0	1	0
1	−4	1
0	1	0

3	0	5	0
3	0	1	0
4	0	0	0
7	0	7	0

$$\boldsymbol{f(1,0) = 3 \times 1 + 0 \times (-4) + 5 \times 1 + 3 \times 0 + 0 \times 1 + 1 \times 0 = 8}$$

De esta manera, ubicamos el valor del nuevo pixel en la posición $(1,0)$:

-9	8		

Con este mismo procedimiento se continúa con los demás pixeles hasta obtener la nueva imagen:

-9	8	-19	5
-5	4	1	1
-6	4	8	0
-24	14	-28	7

- **Filtro Sobel:** Supongamos que una imagen de 4×4 usa 3 bits para cada pixel, y que su representación matricial es la siguiente:

3	0	5	0
3	0	1	0
4	0	0	0
7	0	7	0

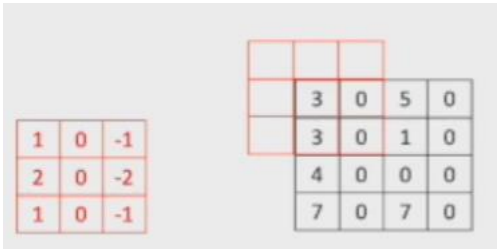
Aplique la máscara del gradiente de Sobel, dada por:

1	0	-1
2	0	-2
1	0	-1

Y use el método de re-escalamiento para calcular los valores de las intensidades de la imagen filtrada.

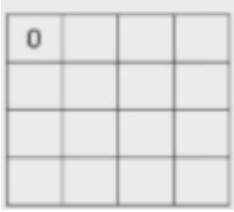
Solución:

Calculemos el valor del pixel $(0,0)$:

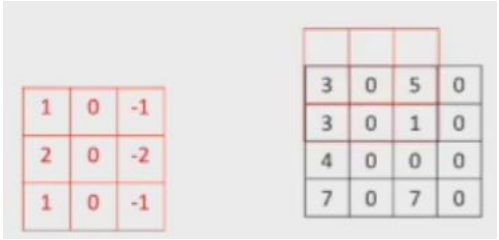


$f(0,0) = 3 \times 0 + 0 \times (-2) + 3 \times 0 + 0 \times (-1) = 0$

Por tanto, en la nueva imagen obtenemos:

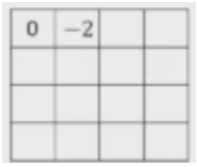


Calculemos el valor del pixel $(1,0)$:



$f(1,0) = 3 \times 2 + 0 \times 0 + 5 \times (-2) + 3 \times 1 + 0 \times 0 + 1 \times (-1) = -2$

Por tanto, en la nueva imagen obtenemos:



De manera sucesiva, se continúa con los demás pixeles y se obtiene:

0	-2	0	11
0	6	0	7
0	10	0	8
0	4	0	14

- **Filtro Canny:**

```
import cv2


# Cargar la imagen
imagen1 = cv2.imread('imagen1.jpg', 0)  # Asegúrate de proporcionar la ruta correcta de la imagen


# Aplicar el filtro Canny
canny = cv2.Canny(imagen1, 100, 200)  # Ajusta los valores de los umbrales según sea necesario
# Mostrar la imagen original y el resultado
cv2.imshow('Imagen original', imagen1)
cv2.imshow('Resultado de Canny', canny)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2) De los filtros definidos en el numeral describir ventajas y desventajas

Filtro de media

Ventajas:

- Reducción del ruido: El filtro de media es efectivo para reducir el ruido aleatorio presente en una imagen, como el ruido de fondo o el ruido causado por condiciones de iluminación.
- Preservación de bordes: A diferencia de otros filtros de suavizado, como el filtro gaussiano, el filtro de media tiende a preservar los bordes en una imagen. Esto significa que los detalles importantes de la imagen se conservan mejor.
- Fácil implementación: El filtro de media es simple de implementar y calcular, lo que lo hace computacionalmente eficiente. No requiere operaciones matemáticas complejas ni una gran cantidad de recursos computacionales.

Desventajas:

- Pérdida de detalles finos: El filtro de media tiende a suavizar una imagen y puede provocar una pérdida de detalles finos, especialmente en áreas donde los cambios de intensidad son bruscos. Esto puede resultar en una imagen con una apariencia más borrosa o menos nítida.
- Borrosidad global: El filtro de media aplica una operación promedio en una ventana local, lo que puede provocar una apariencia borrosa en toda la imagen, incluso en áreas donde no hay ruido. Esto puede ser problemático si se desea preservar la nitidez en toda la imagen.
- Tamaño de la ventana: El tamaño de la ventana utilizada para el filtro de media afecta su desempeño. Si se usa una ventana demasiado pequeña, es posible que no se

reduzca el ruido de manera efectiva. Por otro lado, si se usa una ventana demasiado grande, se puede perder más información y los bordes pueden ser suavizados en exceso.

Filtro de mediana

Ventajas:

- Reducción del ruido de manera efectiva: El filtro de mediana es muy eficaz para reducir el ruido impulsivo, también conocido como ruido de sal y pimienta. Este tipo de ruido aparece como píxeles individuales extremadamente brillantes o extremadamente oscuros y puede afectar significativamente la calidad de la imagen. El filtro de mediana reemplaza el valor de un píxel por el valor mediano de los píxeles en su vecindario, lo que permite eliminar eficientemente el ruido impulsivo.
- Preservación de bordes y detalles finos: A diferencia de otros filtros de suavizado, el filtro de mediana preserva mejor los bordes y detalles finos en una imagen. Esto se debe a que el valor mediano se calcula tomando en cuenta los valores reales de los píxeles vecinos, sin verse afectado por valores atípicos causados por el ruido.
- Aplicación localizada: El filtro de mediana se aplica en una ventana local, lo que permite ajustar el tamaño de la ventana según las necesidades específicas de la imagen. Esto proporciona flexibilidad y control sobre el proceso de filtrado.

Desventajas:

- Pérdida de detalles sutiles: Al igual que otros filtros de suavizado, el filtro de mediana puede provocar una pérdida parcial de detalles sutiles en una imagen. Esto se debe a que el proceso de suavizado está diseñado para eliminar el ruido, lo que puede hacer que los detalles de baja frecuencia también se vean afectados.
- Pérdida de nitidez: En comparación con el filtro de media, el filtro de mediana puede introducir un ligero efecto borroso en la imagen, especialmente si se utilizan tamaños de ventana grandes. Esto puede afectar la nitidez general de la imagen, aunque en menor medida que otros filtros de suavizado más agresivos.

Filtro logarítmico

Ventajas:

- Mejora del rango dinámico: El filtro logarítmico es útil para expandir el rango dinámico de una imagen. Permite resaltar detalles en áreas de baja intensidad y al mismo tiempo comprimir los detalles en áreas de alta intensidad. Esto ayuda a obtener una imagen con un mayor contraste y una apariencia más atractiva visualmente.
- Mejora de detalles en sombras: El filtro logarítmico puede realzar los detalles presentes en las regiones de sombras de una imagen. A menudo, las áreas oscuras contienen información valiosa, pero pueden perderse o ser difíciles de apreciar debido a la falta de contraste. Al aplicar el filtro logarítmico, se pueden revelar detalles ocultos en estas áreas, mejorando así la calidad y la interpretación de la imagen.
- Mayor flexibilidad en el ajuste de contraste: El filtro logarítmico permite un ajuste más flexible del contraste en comparación con otros métodos. Al ajustar los parámetros de ganancia y offset en la función logarítmica, es posible adaptar la mejora del contraste a las necesidades específicas de la imagen.

Desventajas:

- Posible pérdida de información: Si se aplica un filtro logarítmico de manera excesiva o inapropiada, existe el riesgo de perder información valiosa en la imagen. Un ajuste incorrecto de los parámetros puede llevar a una amplificación excesiva del ruido o a la pérdida de detalles finos. Por lo tanto, es necesario tener cuidado al aplicar este filtro y evaluar los resultados de manera adecuada.
- Requiere una calibración adecuada: Para obtener los mejores resultados con el filtro logarítmico, es importante realizar una calibración adecuada de los parámetros de ganancia y offset. Esto implica conocer las características específicas de la imagen y ajustar los parámetros en consecuencia. Si la calibración no se realiza correctamente, los resultados pueden no ser satisfactorios y la imagen podría aparecer distorsionada o poco natural.

Filtro de cuadro normalizado

Ventajas:

- Reducción del ruido: El filtro de cuadro normalizado es efectivo para reducir el ruido aleatorio presente en una imagen. Al calcular el promedio de los valores de los píxeles vecinos, se suavizan las variaciones de intensidad abruptas y se reducen los efectos del ruido.
- Preservación de bordes: A diferencia de otros filtros de suavizado, el filtro de cuadro normalizado tiende a preservar los bordes en una imagen. Esto significa que los detalles importantes, como los contornos y las transiciones de intensidad, se conservan mejor.
- Fácil implementación: El filtro de cuadro normalizado es fácil de implementar y computacionalmente eficiente. Solo requiere el cálculo del promedio de los valores de los píxeles vecinos dentro de la ventana, lo que lo hace rápido y accesible en términos de recursos computacionales.

Desventajas:

- Pérdida de detalles finos: El filtro de cuadro normalizado puede suavizar demasiado la imagen y provocar una pérdida de detalles finos. Al calcular el promedio de los valores de los píxeles vecinos, se puede perder información de alta frecuencia, lo que resulta en una imagen más borrosa o menos nítida.
- Efecto borroso en áreas texturizadas: Si una imagen contiene regiones con texturas o patrones detallados, el filtro de cuadro normalizado puede producir un efecto borroso o suavizado excesivo en esas áreas. Esto se debe a que el promedio de los píxeles vecinos puede eliminar parte de la textura y los detalles de alta frecuencia presentes en esas regiones.
- Sensibilidad al tamaño de la ventana: El tamaño de la ventana utilizada en el filtro de cuadro normalizado afecta su desempeño. Si se utiliza una ventana pequeña, es posible que no se reduzca el ruido de manera efectiva. Sin embargo, si se utiliza una ventana demasiado grande, se puede perder más información y los bordes pueden ser suavizados en exceso.

Filtro gaussiano

Ventajas:

- Reducción del ruido: El filtro gaussiano es efectivo para reducir el ruido aleatorio en una imagen. Su aplicación suaviza las variaciones de intensidad abruptas y reduce los efectos del ruido, lo que resulta en una imagen más limpia y con menos perturbaciones.
- Preservación de detalles estructurales: A diferencia de otros filtros de suavizado, el filtro gaussiano tiene la capacidad de preservar los detalles estructurales de una imagen. Esto significa que los bordes y las características importantes se conservan mejor después de aplicar el filtro, lo que permite una mejor interpretación visual de la imagen.
- Control del nivel de suavizado: El filtro gaussiano permite ajustar el nivel de suavizado mediante la elección del tamaño del *kernel* o el valor de la desviación estándar. Esto brinda flexibilidad para adaptar el filtro a las necesidades específicas de la imagen y lograr el equilibrio deseado entre la reducción de ruido y la preservación de detalles.

Desventajas:

- Pérdida de detalles finos: El filtro gaussiano puede suavizar demasiado la imagen y provocar una pérdida de detalles finos. A medida que se aplica el desenfoque, las texturas y los detalles de alta frecuencia se vuelven menos pronunciados, lo que puede resultar en una imagen más borrosa o menos nítida.
- Difuminado global: El filtro gaussiano aplica un efecto de desenfoque en toda la imagen, lo que puede llevar a un difuminado global. Esto significa que incluso en áreas sin ruido, se produce un ligero efecto de desenfoque que puede afectar la nitidez general de la imagen.
- Mayor costo computacional: En comparación con otros filtros más simples, el filtro gaussiano puede tener un mayor costo computacional, especialmente cuando se utilizan tamaños de kernel más grandes. El cálculo de la convolución con el kernel gaussiano puede ser más exigente en términos de recursos computacionales.

Filtro Laplace

Ventajas:

- Realce de bordes: El filtro Laplace es efectivo para realzar los bordes presentes en una imagen. Resalta los cambios bruscos en la intensidad de los píxeles, lo que permite una mejor visualización y comprensión de las estructuras y detalles presentes en la imagen.
- Detección de características: El filtro Laplace también puede utilizarse para detectar características específicas en una imagen, como líneas, esquinas o puntos de interés. Al aplicar el filtro, las características importantes se destacan y pueden ser utilizadas posteriormente en tareas de segmentación, reconocimiento o análisis de imágenes.
- Fácil implementación: El filtro Laplace es relativamente fácil de implementar y computacionalmente eficiente. Puede ser aplicado mediante una convolución simple utilizando un kernel predefinido, lo que lo hace accesible y rápido en términos de recursos computacionales.

Desventajas:

- Sensible al ruido: El filtro Laplace tiende a amplificar el ruido presente en una imagen. Debido a su naturaleza de resaltar cambios bruscos, cualquier ruido presente en la

imagen también puede ser resaltado y aumentar su visibilidad. Por lo tanto, es necesario tener cuidado al aplicar este filtro en imágenes ruidosas o aplicar técnicas de reducción de ruido previas al filtro Laplace.

- Respuesta a bordes diagonales: El filtro Laplace puede producir una respuesta débil o inconsistente en bordes diagonales. Debido a la discretización espacial de los píxeles, los bordes diagonales pueden no ser detectados con la misma intensidad que los bordes horizontales o verticales.
- Posible sobrerealce: En algunos casos, el filtro Laplace puede producir un sobrerealce en los bordes, lo que resulta en una apariencia de "efecto halo" alrededor de los contornos. Esto puede ser indeseable, especialmente si se desea una representación precisa de los bordes sin ninguna distorsión adicional.

Filtro Sobel

Ventajas:

- Detección precisa de bordes: El filtro Sobel es eficaz para detectar bordes en una imagen con alta precisión. Al calcular las derivadas de primer orden en dirección horizontal y vertical, resalta los cambios bruscos de intensidad en los píxeles y permite identificar con precisión los contornos de los objetos presentes en la imagen.
- Robustez ante el ruido: El filtro Sobel es relativamente robusto frente al ruido en comparación con otros filtros de detección de bordes. Esto se debe a que utiliza una técnica de diferenciación local, lo que reduce la influencia del ruido en la detección de bordes.
- Fácil implementación: El filtro Sobel es fácil de implementar y computacionalmente eficiente. Puede aplicarse mediante una convolución simple utilizando dos kernels predefinidos (uno para la dirección horizontal y otro para la dirección vertical), lo que facilita su aplicación y reduce la carga computacional.

Desventajas:

- Respuesta limitada a bordes diagonales: El filtro Sobel tiene una respuesta limitada a bordes diagonales. Debido a la discretización espacial de los píxeles, los bordes diagonales pueden no ser detectados con la misma intensidad que los bordes horizontales o verticales. Esto puede resultar en una menor precisión en la detección de bordes diagonales.
- Pérdida de detalles finos: El filtro Sobel puede suavizar la imagen y provocar una pérdida de detalles finos en las regiones cercanas a los bordes detectados. Al aplicar la derivada espacial, los detalles de alta frecuencia pueden atenuarse, lo que puede resultar en una imagen ligeramente borrosa o menos nítida.
- Dependencia del tamaño del kernel: El tamaño del kernel utilizado en el filtro Sobel afecta su desempeño. Si el tamaño del kernel es demasiado pequeño, es posible que no se detecten todos los bordes presentes en la imagen. Sin embargo, si el tamaño del kernel es demasiado grande, se puede perder la precisión en la detección de bordes finos.

Filtro Canny

Ventajas:

- Detección precisa de bordes: El filtro Canny es conocido por su capacidad para detectar bordes con alta precisión. Utiliza una combinación de técnicas, incluyendo suavizado, cálculo de gradiente y umbralización, para identificar los cambios bruscos en la intensidad de los píxeles y determinar los contornos de los objetos en la imagen.
- Supresión de ruido: El filtro Canny incluye un paso de suavizado utilizando un filtro gaussiano para reducir el ruido presente en la imagen. Esto ayuda a eliminar el ruido de fondo y garantiza que solo se detecten los bordes más significativos y relevantes.
- Respuesta a bordes finos: El filtro Canny es capaz de detectar bordes finos en una imagen. Puede resaltar con precisión los detalles más sutiles, lo que es especialmente útil en aplicaciones donde se requiere una alta resolución y detalles nítidos en los bordes.

Desventajas:

- Mayor complejidad computacional: El filtro Canny implica varios pasos, incluyendo el suavizado gaussiano, el cálculo del gradiente y la umbralización. Esto implica un mayor costo computacional en comparación con otros métodos de detección de bordes más simples. Sin embargo, con el avance de la tecnología, esta desventaja es cada vez menos relevante.
- Sensibilidad a la selección de parámetros: El filtro Canny requiere la elección adecuada de parámetros, como el tamaño del kernel gaussiano y los umbrales de umbralización, para obtener los resultados deseados. La selección incorrecta de estos parámetros puede afectar la calidad de los bordes detectados y generar resultados subóptimos.
- Respuesta a condiciones de iluminación variables: El filtro Canny puede verse afectado por cambios en las condiciones de iluminación de la imagen. En áreas con cambios abruptos en la iluminación, el filtro puede producir resultados inconsistentes o detectar bordes falsos. Es importante tener esto en cuenta al aplicar el filtro en imágenes con iluminación variable.

3) De los filtros definidos en el numeral 1 programar un ejemplo usando opencv ó Matlab.

Ver el desarrollo de este ejercicio en
https://github.com/watorres/PDI/blob/main/Filtros_OpenCv.ipynb

Conclusiones

- Los filtros son herramientas fundamentales en el procesamiento digital de imágenes, ya que permiten mejorar la calidad visual, realzar características importantes y eliminar ruido o imperfecciones presentes en las imágenes.

- Existen diferentes tipos de filtros que se pueden aplicar en el procesamiento de imágenes, como el filtro de promedio, filtro mediana, filtro logarítmico, filtro de cuadro normalizado, filtro gaussiano, filtro Laplace, filtro Sobel y filtro Canny, entre otros.

Cada uno de ellos tiene sus propias ventajas y desventajas, y su selección depende del objetivo específico del procesamiento.

- La elección adecuada del filtro depende de la naturaleza de la imagen y del resultado deseado. Es importante comprender las propiedades y características de cada filtro para utilizarlos de manera efectiva y obtener los resultados deseados.

- El procesamiento digital de imágenes con filtros puede aplicarse en una amplia gama de aplicaciones, como mejoramiento de imágenes médicas, reconocimiento de objetos, detección de bordes, filtrado de ruido en fotografías, entre otros. Esto demuestra la relevancia y versatilidad de los filtros en el campo del procesamiento de imágenes.

- Es importante tener en cuenta que el uso de filtros en el procesamiento digital de imágenes no debe ser excesivo, ya que puede generar efectos no deseados o distorsionar la información original de la imagen. Se recomienda realizar pruebas y ajustes adecuados para obtener los mejores resultados.

Referencias bibliográficas

- Contreras, D., & Sossa, H. (2016). Procesamiento Digital de Imágenes: Fundamentos y Aplicaciones. México: Alfaomega.
- González, R. C., & Woods, R. E. (2008). Técnicas de Procesamiento Digital de Imágenes. México: Pearson Educación.
- Ramos, S. L., & Muñoz, A. (2012). Procesamiento Digital de Imágenes: Fundamentos y Aplicaciones. España: Marcombo.

PARTE 2: Descriptores

1) Generar un banco de imágenes con frutas

El banco de imágenes se encuentra en el enlace: https://udeaeduco-my.sharepoint.com/:f:/r/personal/walexander_torres_udea_edu_co/Documents/BDFRUTAS?csf=1&web=1&e=ZtKiTU

2) Hacer preprocesamiento a la imagen:

A continuación, se realiza un preprocesamiento de redimensionamiento (resize) en cada imagen y calcular otro aspecto de preprocesamiento, en este caso, el histograma de cada imagen

```
import os

import cv2

import matplotlib.pyplot as plt


# Ruta del banco de imágenes

ruta_banco = r'C:\BD'


# Obtener la lista de carpetas
```

```
carpetas = ['FRESA', 'MANGO', 'MANZANA', 'PERA']

# Tamaño deseado para redimensionar las imágenes
nuevo_tamaño = (256, 256)

# Función para redimensionar una imagen y calcular su histograma
def preprocesar_imagen(ruta_imagen):
    # Leer la imagen
    imagen = cv2.imread(ruta_imagen)

    # Redimensionar la imagen
    imagen_redimensionada = cv2.resize(imagen, nuevo_tamaño)

    # Calcular el histograma de la imagen
    histograma = cv2.calcHist([imagen_redimensionada], [0], None, [256], [0, 256])

    return imagen_redimensionada, histograma

# Procesar cada imagen en las carpetas
for carpeta in carpetas:
    # Ruta completa de la carpeta
    ruta_carpeta = os.path.join(ruta_banco, carpeta)

    # Obtener la lista de imágenes en la carpeta
    imagenes = os.listdir(ruta_carpeta)

    # Procesar cada imagen
    for imagen_nombre in imagenes:
        # Ruta completa de la imagen
        ruta_imagen = os.path.join(ruta_carpeta, imagen_nombre)

        # Preprocesar la imagen
        imagen_redimensionada, histograma = preprocesar_imagen(ruta_imagen)

        # Mostrar la imagen redimensionada
        cv2.imshow('Imagen redimensionada', imagen_redimensionada)
        cv2.waitKey(0)

        # Mostrar el histograma
```

```
    plt.plot(histograma)
    plt.title('Histograma')
    plt.show()

# Cerrar las ventanas abiertas
cv2.destroyAllWindows()
```

3) Definir características HOG, SIFT y extraer dichas características a las imágenes de las frutas (SIFT opcionales en implementación para el entrenamiento de la red, se escribe el algoritmo en qué consiste y un ejemplo de cómo se implementa en una imagen)

Características HOG

```
import os
import cv2
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import exposure

# Ruta del banco de imágenes
ruta_banco = r'C:\BD'

# Obtener la lista de carpetas
carpetas = ['FRESA', 'MANGO', 'MANZANA', 'PERA']

# Tamaño deseado para redimensionar las imágenes
nuevo_tamaño = (256, 256)

# Parámetros para HOG
orientaciones = 9
pixeles_por_celda = (8, 8)

# Función para redimensionar una imagen, calcular su histograma y extraer características HOG
def preprocesar_imagen(ruta_imagen):
    # Leer la imagen
    imagen = cv2.imread(ruta_imagen)

    # Redimensionar la imagen
    imagen_redimensionada = cv2.resize(imagen, nuevo_tamaño)

    # Calcular el histograma de la imagen
```

```
    histograma = cv2.calcHist([imagen_redimensionada], [0], None, [256], [0, 256])

    # Calcular las características HOG
    hog_features, hog_image = hog(imagen_redimensionada, orientations=orientaciones,
pixels_per_cell=pixeles_por_celda,
                        visualize=True, channel_axis=-1)  # Utiliza `channel_axis` en lugar de `multichannel` en versiones futuras

    return imagen_redimensionada, histograma, hog_image

# Procesar cada imagen en las carpetas
for carpeta in carpetas:
    # Ruta completa de la carpeta
    ruta_carpeta = os.path.join(ruta_banco, carpeta)

    # Obtener la lista de imágenes en la carpeta
    imagenes = os.listdir(ruta_carpeta)

    # Procesar cada imagen
    for imagen_nombre in imagenes:
        # Ruta completa de la imagen
        ruta_imagen = os.path.join(ruta_carpeta, imagen_nombre)

        # Preprocesar la imagen
        imagen_redimensionada, histograma, hog_image = preprocesar_imagen(ruta_imagen)

        # Mostrar la imagen redimensionada
        plt.imshow(imagen_redimensionada)
        plt.title('Imagen redimensionada')
        plt.axis('off')
        plt.show()

        # Mostrar el histograma
        plt.plot(histograma)
        plt.title('Histograma')
        plt.show()

        # Mostrar la representación HOG
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

        ax1.imshow(imagen_redimensionada)
```

```
        ax1.axis('off')
        ax1.set_title('Imagen redimensionada')

        hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
        ax2.imshow(hog_image_rescaled, cmap='gray')
        ax2.axis('off')
        ax2.set_title('Representación HOG')

        plt.show()

        # Pausa para ver la imagen antes de mostrar la siguiente
        input("Presiona Enter para continuar...")

# Cerrar las ventanas abiertas
cv2.destroyAllWindows()
```

Descriptores ORB:

```
import os
import cv2
import matplotlib.pyplot as plt
from skimage.feature import hog

# Ruta del banco de imágenes
ruta_banco = r'C:\2023 10\UdeA\PDI\Lab02'

# Obtener la lista de carpetas
carpetas = ['FRESA', 'MANGO', 'MANZANA', 'PERA']

# Tamaño deseado para redimensionar las imágenes
nuevo_tamaño = (256, 256)

# Parámetros para HOG
orientaciones = 9
pixeles_por_celda = (8, 8)
celdas_por_bloque = (2, 2)

# Función para redimensionar una imagen y calcular sus características HOG y ORB
def preprocesar_imagen(ruta_imagen):
    # Leer la imagen
```

```
    imagen = cv2.imread(ruta_imagen)

    # Redimensionar la imagen
    imagen_redimensionada = cv2.resize(imagen, nuevo_tamaño)

    # Calcular el histograma de la imagen
    histograma = cv2.calcHist([imagen_redimensionada], [0], None, [256], [0, 256])

    # Calcular las características HOG de la imagen
    hog_features = hog(imagen_redimensionada, orientations=orientaciones, pixels_per_cell=pixeles_por_celda,
                       cells_per_block=celdas_por_bloque, channel_axis=-1)

    # Crear el detector ORB
    orb = cv2.ORB_create()

    # Encontrar los keypoints y descriptores ORB
    keypoints, descriptors = orb.detectAndCompute(imagen_redimensionada, None)

    return imagen_redimensionada, histograma, hog_features, keypoints, descriptors

# Procesar cada imagen en las carpetas
for carpeta in carpetas:
    # Ruta completa de la carpeta
    ruta_carpeta = os.path.join(ruta_banco, carpeta)

    # Obtener la lista de imágenes en la carpeta
    imagenes = os.listdir(ruta_carpeta)

    # Procesar cada imagen
    for imagen_nombre in imagenes:
        # Ruta completa de la imagen
        ruta_imagen = os.path.join(ruta_carpeta, imagen_nombre)

        # Preprocesar la imagen
        imagen_redimensionada, histograma, hog_features, keypoints, descriptors = preprocesar_imagen(ruta_imagen)

        # Mostrar la imagen redimensionada
        cv2.imshow('Imagen redimensionada', imagen_redimensionada)
        cv2.waitKey(0)
```

```
    # Mostrar el histograma
    plt.plot(histograma)
    plt.title('Histograma')
    plt.show()

    # Imprimir características HOG
    print('Características HOG:', hog_features)
    print()

    # Imprimir keypoints y descriptores ORB
    for i, keypoint in enumerate(keypoints):
        print('Keypoint', i+1, ':', keypoint)
        print('Descriptor', i+1, ':', descriptors[i])
        print()

# Cerrar las ventanas abiertas
cv2.destroyAllWindows()
```

4) Entrenar una red neuronal con las características obtenidas (HOG)

Código para entrenar la red neuronal:

```
import os
import cv2
import numpy as np
from skimage.feature import hog
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, confusion_matrix, f1_score

# Ruta del banco de imágenes
ruta_banco = r'C:\BD'

# Obtener la lista de carpetas
carpetas = ['FRESA', 'MANGO', 'MANZANA', 'PERA']

# Tamaño deseado para redimensionar las imágenes
nuevo_tamaño = (256, 256)

# Parámetros para HOG
```

```
orientaciones = 9
pixeles_por_celda = (8, 8)

# Lista para almacenar las características HOG y las etiquetas
hog_features_list = []
etiquetas = []

# Procesar cada imagen en las carpetas
for carpeta in carpetas:
    # Ruta completa de la carpeta
    ruta_carpeta = os.path.join(ruta_banco, carpeta)

    # Obtener la lista de imágenes en la carpeta
    imagenes = os.listdir(ruta_carpeta)

    # Procesar cada imagen
    for imagen_nombre in imagenes:
        # Ruta completa de la imagen
        ruta_imagen = os.path.join(ruta_carpeta, imagen_nombre)

        # Leer la imagen
        imagen = cv2.imread(ruta_imagen)

        # Redimensionar la imagen
        imagen_redimensionada = cv2.resize(imagen, nuevo_tamaño)

        # Calcular las características HOG
        hog_features = hog(imagen_redimensionada, orientations=orientaciones,
pixels_per_cell=pixeles_por_celda,
                           channel_axis=-1)

        # Agregar las características HOG y la etiqueta a las listas
        hog_features_list.append(hog_features)
        etiquetas.append(carpeta)

# Convertir las listas a matrices numpy
X = np.array(hog_features_list)
```

```
y = np.array(etiquetas)

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear y entrenar el clasificador de red neuronal
clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500)
clf.fit(X_train, y_train)

# Realizar predicciones en el conjunto de prueba
y_pred = clf.predict(X_test)

# Calcular las métricas de evaluación
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
confusion = confusion_matrix(y_test, y_pred)
f_score = f1_score(y_test, y_pred, average='weighted')

# Imprimir los resultados
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Confusion Matrix:")
print(confusion)
print
```

5) Investigar métricas (por ejemplo: Accuracy, precisión, matriz de confusión, falsos positivos, falsos negativos, medidas F-Score)

- **Exactitud (Accuracy):** Es una métrica que indica la proporción de muestras clasificadas correctamente sobre el total de muestras. Se calcula dividiendo el número de muestras clasificadas correctamente entre el total de muestras.
- **Precisión (Precision):** Es la proporción de muestras positivas correctamente clasificadas sobre el total de muestras clasificadas como positivas. Mide la precisión del clasificador al predecir las muestras positivas. Se calcula dividiendo el número de verdaderos positivos entre la suma de verdaderos positivos y falsos positivos.
- **Matriz de confusión (Confusion Matrix):** Es una tabla que muestra el número de muestras clasificadas correctamente e incorrectamente para cada clase. En una matriz de confusión típica de una tarea de clasificación binaria, los verdaderos positivos (TP) se encuentran en la posición superior izquierda, los verdaderos negativos (TN) en la

posición inferior derecha, los falsos positivos (FP) en la posición superior derecha y los falsos negativos (FN) en la posición inferior izquierda.

- **Falsos positivos (False Positives) y Falsos negativos (False Negatives):** Los falsos positivos son las muestras que son clasificadas incorrectamente como positivas, es decir, el clasificador las identifica como pertenecientes a una clase cuando en realidad no lo son. Los falsos negativos son las muestras que son clasificadas incorrectamente como negativas, es decir, el clasificador las identifica como no pertenecientes a una clase cuando en realidad sí lo son.
- **Puntuación F (F-Score):** Es una medida que combina la precisión y el recall (también conocido como sensibilidad o exhaustividad) en un solo valor. Es útil cuando se desea tener en cuenta tanto los falsos positivos como los falsos negativos. La puntuación F se calcula como la media armónica entre la precisión y el recall, y proporciona una medida equilibrada de ambas métricas.

Al entrenar la red obtuve:

```
In [6]: runfile('C:/2023 10/UdeA/PDI/Lab02/RedNeuronal.py', wdir='C:/2023 10/UdeA/
PDI/Lab02')
Accuracy: 0.8656716417910447
Precision: 0.8843799435999724
Confusion Matrix:
[[ 6  0  4  0]
 [ 0 18  1  0]
 [ 0  1 25  0]
 [ 0  2  1  9]]
```

Interpretación:

- **Accuracy (exactitud**): Indica la proporción de predicciones correctas en relación con el total de predicciones. En este caso, el valor de exactitud es 0.8657, lo que significa que el modelo tiene una precisión del 86.57% en la clasificación de las muestras.
- **Precision (precisión):** Mide la proporción de instancias clasificadas correctamente de una clase específica en relación con el total de instancias clasificadas en esa clase. En tu caso, la precisión promedio es de 0.8844, lo que indica que el modelo tiene una precisión del 88.44% en la clasificación de las muestras.
- **Confusion Matrix (matriz de confusión):** Es una tabla que muestra la relación entre las etiquetas reales y las etiquetas predichas por el modelo. En este caso, la matriz de confusión indica lo siguiente:

    a. La primera fila indica que hay 6 instancias de la clase 'FRESA' que se predijeron correctamente, 0 instancias de 'FRESA' se predijeron como 'MANGO', 4 instancias de 'FRESA' se predijeron como 'MANZANA' y 0 instancias de 'FRESA' se predijeron como 'PERA'.
    b. La segunda fila indica que hay 0 instancias de la clase 'MANGO' que se predijeron incorrectamente como 'FRESA', 18 instancias de 'MANGO' se predijeron correctamente, 1 instancia de 'MANGO' se predijo como 'MANZANA' y 0 instancias de 'MANGO' se predijeron como 'PERA'.

    c. La tercera fila indica que no hay instancias de la clase 'MANZANA' que se predijeron incorrectamente, 1 instancia de 'MANZANA' se predijo incorrectamente

como 'MANGO', 25 instancias de 'MANZANA' se predijeron correctamente y 0 instancias de 'MANZANA' se predijeron como 'PERA'.

d. La cuarta fila indica que no hay instancias de la clase 'PERA' que se predijeron incorrectamente, 2 instancias de 'PERA' se predijeron incorrectamente como 'MANGO', 1 instancia de 'PERA' se predijo correctamente y 9 instancias de 'PERA' se predijeron como 'MANZANA'.
e. La matriz de confusión te proporciona una visión detallada del rendimiento del modelo al mostrar las predicciones correctas e incorrectas para cada clase. Puedes analizar los valores fuera de la diagonal principal para identificar las instancias que se clasificaron incorrectamente.