Tarea 2

Filtrado lineal y no lineal de imágenes

Profesor: Claudio Pérez F.
Auxiliar: Diego Maureira
Ayudantes: Juan Pablo Pérez, Jhon Pilataxi, Jorge Zambrano

1. Objetivo

El objetivo de esta tarea es comprender y aplicar diversos filtros de procesamiento digital de imágenes, tanto lineales como no lineales, así como también detectores de bordes. Usted deberá implementar estos filtros, evaluar su desempeño y comparar los resultados obtenidos en imágenes con distintos niveles de ruido y complejidad. Al finalizar la tarea, deberia ser capaz de generalizar el conocimiento adquirido, pudiendo seleccionar el filtro o detector más adecuado para una aplicación específica.

2. Descripción

En el procesamiento digital de imágenes, los filtros juegan un rol crucial tanto en la mejora de la calidad como en la extracción de información relevante desde las imágenes. Entre los filtros más utilizados se encuentran los lineales y no lineales, los cuales permiten suavizar, reducir el ruido y mejorar el contraste de imágenes. Los detectores de bordes, por otro lado, son herramientas clave para identificar cambios abruptos en la intensidad de la imagen, permitiendo destacar contornos y estructuras importantes.

En esta tarea, se explorará el uso práctico de estos filtros y detectores, aplicándolos a imágenes reales para mejorar su interpretación y extraer características clave. Además, aprenderán a implementar y analizar el impacto de diferentes tipos de filtros y detectores, entendiendo sus aplicaciones y limitaciones.

3. Filtros lineales

Los filtros lineales son aquellos cuya salida es una combinación lineal de los valores de entrada. La mayoría de los filtros de este tipo se implementan mediante una operación de convolución entre la imagen y un *kernel*, lo que define la manipulación de los valores de los pixeles.

Ejemplos de este tipo de filtro son el filtro de la media (promedio), filtro Gaussiano y filtro de Sobel (para detección de bordes). El primero de ellos se basa en calcular el valor promedio (media aritmética) de los pixeles en un vecindario determinado alrededor de cada pixel de la imagen y, luego, reemplazar el valor del pixel central con el calculado. Por otro lado, los filtros Gaussianos suavizan las imágenes mediante una distribución Gaussiana de los valores de los pixeles de su vecindario, es decir, es una suma con ponderaciones diferentes para cada pixel en el vecindario considerado.

Por otra parte, es necesario introducir el concepto de operaciones morfológicas. Estas son técnicas basadas en la forma o estructura de los objetos presentes en una imagen, que utilizan elementos estructurantes (*kernels* con una forma predefinida como cuadrado, círculo, cruz, etc.) que se mueven a lo largo de la imágenes para determinar cómo se deben modificar los pixeles en función de las formas encontradas en ellas. Se utilizan principalmente en imágenes binarias y son esenciales para manipular la forma de los objetos en tareas como la segmentación de objetos, el relleno de agujeros, la extracción de contornos, y la eliminación de ruido específico.

En esta primera actividad se hará una comparación entre el efecto de la aplicación del filtro de la media y el Gaussiano sobre dos imágenes. Para esto deberá seguir los siguientes pasos:

1. Extienda la función de convolución implementada en la Tarea anterior a 3 dimensiones, es decir, ahora su función debe aceptar como entrada una imagen de dimensiones $H \times W \times C$, donde $C = 3$, ya que las imágenes que recibirá serán RGB.

2. Genere una nueva función que reciba de entrada un número entero llamado *size* y entregue un *kernel* de tamaño $size \times size$ con todos sus valores iguales a $\frac{1}{size^2}$. ¿Qué efecto cree que tendrá la convolución entre una imagen y este *kernel*?

3. Cargue las imágenes *Abyssinian_83* y *Abyssinian_94* desde la carpeta *cats*. Asegúrese de que las imágenes esten en formato RGB, ya que normalmente cuando se utiliza **OpenCV**, las imágenes se cargan en orden BGR. Para convertir la imagen a RGB la siguiente línea de código podría serle útil:

```
# Convert to RGB
rgb_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB)

```

4. Cargue las máscaras asociadas a las imágenes, estas tienen el mismo nombre pero son de tipo *.png*. Note que las máscaras no son binarias, de hecho, presentan 3 clases. Visualice las máscaras y deduzca qué representa cada clase.

5. Convierta las máscaras a binarias. La idea es que las zonas asociadas a clases relevantes (en este caso, sectores relacionados con *gatos*) tengan valor igual a 1 y el resto 0.

6. Investigue sobre el efecto de la operación morfológica de erosión. Aplique esta función sobre la máscara binaria, con un *kernel* de 3×3 y 11 iteraciones. La siguiente línea de código puede ayudarlo:

```
# Erode mask
eroded_binary_mask = cv2.erode(binary_mask, np.ones((3,3), np.uint8), iterations
 ↪ =11)

```

7. Aplique su función de convolución 3D entre las imágenes y un *kernel* de tamaño 15×15 desde su función generadora. ¿Qué observa en el resultado?

8. Aplique la función *cv2.GaussianBlur* sobre la imagen con un filtro del mismo tamaño del punto anterior.

9. Reemplace en ambos resultados anteriores los pixeles de la imagen original en las posiciones donde la máscara binaria erosionada posee un valor igual a 1. La siguiente línea podría ayudarle.

```
result[eroded_binary_mask==1] = original_image[eroded_binary_mask==1]

```

10. Visualice la imagen original, el resultado con el filtro de media y el resultado con el filtro Gaussiano en una misma figura. Comente respecto al resultado.

La Figura 1 presenta un ejemplo de los resultados visuales que deberia obtener en esta actividad.

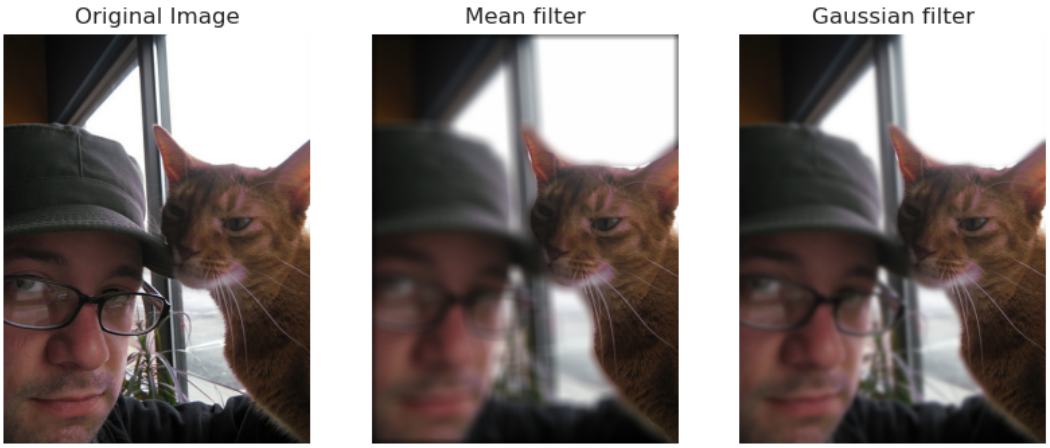

Figura 1: Ejemplo de filtrado lineal con enmascaramiento posterior.

4. Filtros no lineales

Este tipo de filtros, a diferencia de los filtros lineales, utilizan operaciones que no son combinaciones lineales de los pixeles, como la selección del valor máximo, mínimo o mediano

de un vecindario.

Un filtro no lineal muy utilizado es el filtro de mediana, el cual ordena los valores en la vecindad de pixeles seleccionada y reemplaza el valor del pixel central por la mediana de ese conjunto. Este filtro es especialmente eficaz para eliminar el ruido sal y pimienta, que es un tipo de ruido impulsivo que se manifiesta en las imágenes digitales como pixeles aleatoriamente dispersos con valores extremos, es decir, **pixeles que se vuelven completamente blancos (sal) o completamente negros (pimienta)**.

En esta segunda actividad usted deberá implementar un filtro de mediana y aplicarlo sobre las imágenes contenidas en la carpeta *sap_noise* de la tarea, para así ver la utilidad de este tipo de filtro. Para esto deberá seguir los siguientes pasos:

1. Cargue, convierta a RGB y visualice las imágenes contenidas en la carpeta *sap_noise*, estos ejemplos presentan ruido de tipo sal y pimienta. ¿Las características de estas imágenes se condicen con la definición de este tipo de ruido? Argumente.

2. Separe las imágenes en sus 3 canales y observe cada uno por separado. Responda la misma pregunta del punto anterior y concluya.

3. Cree una función que reciba como entrada una imagen de cualquier tamaño y un parámetro que defina el tamaño de vecindad a considerar. La función debe entregar una nueva imagen donde cada elemento de salida corresponde a la mediana de la vecindad del elemento en la imagen original. Debe tratar las condiciones de borde con la función *np.pad*, con una estrategia *edge*.

4. Extienda la función anterior a imágenes de tipo RGB. Se sugiere operar su función de filtro de mediana en cada canal y luego unir los mismos.

5. **Desde aqui en adelante utilizarán *kernels* con un tamaño de** 5×5

6. Aplique un filtro Gaussiano sobre la imagen RGB. Observe el resultado y comente acerca del efecto de este tipo de filtro sobre el ruido sal y pimienta.

7. Repita el punto anterior pero con su implementación del filtro de mediana.

8. Repita el punto 6 pero con la función *cv2.medianBlur*.

9. Obtenga el porcentaje de pixeles resultantes que son iguales en su implementación del filtro de mediana y la implementación de **OpenCV**. Si el porcentaje es distinto a 100 % argumente qué factores pueden influir en esto.

   Hint: El porcentaje sí puede ser 100 %. El no llegar a este porcentaje de similitud no representa un descuento de puntaje si es que la argumentación es completa.

La Figura 2 presenta un ejemplo de los resultados visuales que deberia obtener en esta actividad.

Figura 2: Ejemplo de filtrado de ruido sal y pimienta.

5. Actividad Final: Detección de bordes

La detección de bordes es una técnica fundamental en el procesamiento de imágenes, utilizada para identificar áreas de cambio abrupto en la intensidad de pixeles, que generalmente corresponden a los contornos o límites de objetos. Existen tanto filtros lineales como no lineales que se emplean para este propósito. Dentro de los filtros lineales utilizados están: filtro de Prewitt, Sobel, Laplace y Roberts (que se concentra más en variaciones diagonales). Por otro lado, dentro de los filtros no lineales para la detección de bordes, el que más destaca por su robustez y equilibrio entre suavizado, precisión y detección de bordes es Canny, el cual se compone de un filtro Gaussiano + NMS + umbral de histéresis.

En esta actividad usted utilizará el filtro Canny para la detección de bordes en imágenes que presentan grietas en concreto, para luego, mediante la aplicación de operaciones morfológicas, ser capaz de segmentar la zona agrietada. Para esto deberá seguir los siguientes pasos:

- Cargue cada una de las 4 imágenes de la carpeta *crack_segmentation* y su respectiva máscara binaria.
- Convierta las imágenes a escala de grises.
- Aplique un filtro Gaussiano de 3×3 a la imagen en escala de grises. Investigue la utilidad de la aplicación de un suavizado previo a la detección de bordes y comente.
- Genere una detección de bordes tanto en la imagen en escala de grises suavizada como en la original. Para esto utilice la función *cv2.Canny*, la cual recibe como entrada una imagen y un umbral inferior y superior de histéresis. Utilice los valores 50 y 200 para estos umbrales, respectivamente.
- Visualice la detección de bordes en ambas imágenes. Comente al respecto.
- Aplique sobre el resultado de detección de bordes en las imágenes suavizadas la operación morfológica de cierre. La siguiente línea de código lo puede ayudar:

```
# Closing mask
closed_crack_img = cv2.morphologyEx(canny_crack_img, cv2.MORPH_CLOSE,
 ↪ np.ones((3,3), np.uint8), iterations=2)

```

- Investigue sobre las métricas de *Accurary* e *Intersection over Union (IoU)* para evaluar la segmentación de imágenes. Implemente y compare con estas métricas la máscara original de cada imagen con la segmentación generada por su método. Reporte en una tabla los resultados para cada una de las 4 imágenes. Elija la métrica que a su juicio mejor expresa el resultado que usted puede visualizar en las imágenes. Argumente.
- Concluya ventajas y desventajas de este método de detección de bordes. Proponga alguna mejora del método para obtener una mejor segmentación, no es necesario implementarla.

La Figura 3 presenta un ejemplo de los resultados visuales que deberia obtener en esta actividad.

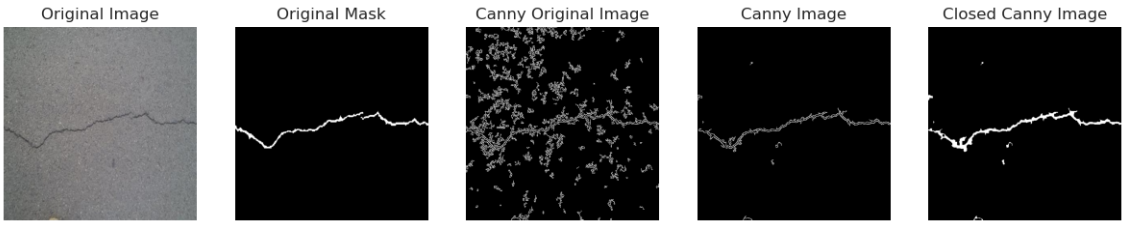

Figura 3: Ejemplo de detección de bordes para segmentación de grietas.

6. Entregables

- Presente un reporte del trabajo. Incluya en el informe los análisis y visualizaciones solicitadas.
- El código también debe ser entregado, asegúrese de que pueda ser ejecutado por los revisores para comprobar el algoritmo.

7. Recordatorio

- El reporte es individual.
- Recuerde que, para realizar la convolución, el filtro debe ser reflejado horizontal y verticalmente.
- Hay un plazo de entrega con atraso de 4 días máximo, pero cada día representa un punto de descuento en su nota final.

La fecha de entrega de trabajo será el día 03/10/2024 a las 23:59 hrs por medio de U-cursos.