

Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Ing. Informática
VISION COMPUTACIONAL
Práctica N° 03

RUIDO Y SUAVIZADO

Iván C. Medrano Valencia

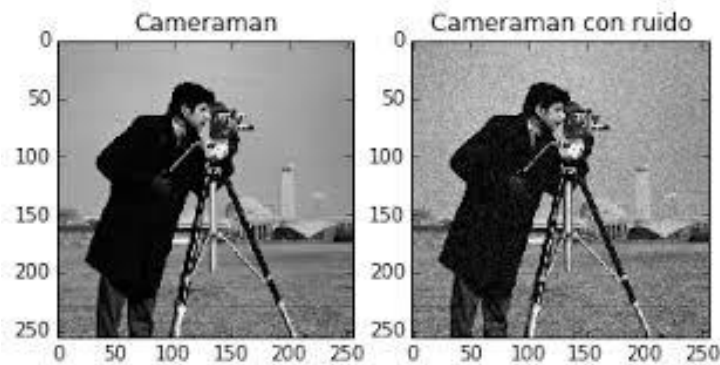
1. OBJETIVO.

- Conocer cómo agregar ruido de diferentes tipos a una imagen
- Conocer cómo disminuir el ruido en imágenes.
- Conocer el concepto de convolución de una imagen
- Realizar suavizado de imágenes.

2. ACTIVIDAD I.

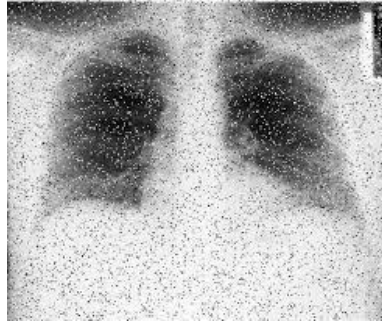
- **Ruido Gaussiano**

En el ruido de tipo gaussiano, todos y cada uno de los píxeles que componen la imagen cambian su valor, de acuerdo con una distribución normal o gaussiana. Se podrían aplicar otro tipo de distribuciones, pero la gaussiana o normal se toma como el modelo al que más se aproxima, debido al teorema central del límite, que dice que la suma de los diferentes ruidos tiende a aproximarse a una distribución normal o gaussiana.



- **Ruido Sal y Pimienta**

Está formado por puntos blancos y/o negros que se distribuyen de manera aleatoria por la imagen, este tipo de ruido es independiente de la señal. Es un ruido que aparece muchas veces producido por interferencias atmosféricas, o por acciones hechas por el humano.



Ejercicio 3.1

```
import cv2
import numpy as np
from skimage.util import random_noise

# cargar la imagen
img = cv2.imread("images\lena.jpg")

# agregar diferentes tipos de ruido a la imagen
noise_img = random_noise(img, mode='gaussian', clip=True)
#noise_img = random_noise(img, mode='salt', amount=0.005)
#noise_img = random_noise(img, mode='pepper', amount=0.005)
#noise_img = random_noise(img, mode='speckle', var=0.02, clip=True)
#noise_img = random_noise(img, mode='poisson', seed=1)
#noise_img = random_noise(img, mode='s&p', amount=0.3)

#Las funciones anteriores devuelven una imagen de punto flotante
#en el rango de [0,1], los cambiamos a 'uint8'
#y de [0,255]

noise_img = np.array(255*noise_img, dtype = 'uint8')

# mostrar el ruido en la imagen
cv2.imshow('ruidos', noise_img)
cv2.waitKey(0)
```

En este ejercicio se está usando la función *random_noise* de la librería *skimage*. Esta función tiene los siguientes parámetros:

img: es la imagen,

mode: es el modo de la adición de ruido. Puede tomar los siguientes valores:

- 'gaussian': Ruido aditivo en distribución gaussiana

- 'poisson': Ruido distribuido por Poisson generado a partir de los datos.

- 'salt': Reemplaza los píxeles aleatorios por 1.

- 'pepper': Reemplaza los píxeles aleatorios por 0

- 's&p': Reemplaza los píxeles aleatorios por 0 o 1.

- 'speckle': Ruido multiplicativo usando $out = image + n * image$, donde *n* es ruido uniforme con media y varianza especificadas.

- seed*: Si se proporciona, esto establecerá la semilla aleatoria antes de generar ruido, para comparaciones pseudoaleatorias válidas.

clip: bool

Si es Verdadero (predeterminado), la salida se recortará después de que se aplique el ruido para los modos 'moteado', 'poisson' y 'gaussiano'. Esto es necesario para mantener el rango de datos de imagen adecuado. Si es False, no se aplica el recorte y la salida puede extenderse más allá del rango [-1, 1].

mean: float

Media de distribución aleatoria. Usado en 'gaussiano' y 'moteado'. Predeterminado: 0.

var: float

Varianza de distribución aleatoria. Se utiliza en 'gaussiano' y 'moteado'.

Nota: $\text{varianza} = (\text{desviación estándar})^2$. Predeterminado: 0.01

local_vars: ndarray

Matriz de floats positivos, con la misma forma que la imagen, que define la varianza local en cada punto de la imagen. Usado en 'localvar'.

amount: float

Proporción de píxeles de la imagen para reemplazar con ruido en el rango [0, 1]. Se utiliza en 'sal', 'pimienta' y 'sal y pimienta'. Predeterminado: 0.05

salt_vs_pepper: float

Proporción de ruido de sal frente a pimienta para 's' & 'p' en el rango [0, 1]. Los valores más altos representan más sal. Predeterminado: 0.5 (cantidades iguales)

3. ACTIVIDAD 2

- **Eliminación de ruido en las imágenes**

OpenCV proporciona cuatro métodos para eliminar el ruido.

cv.fastNlMeansDenoising() - Procesar una sola imagen en escala de grises

cv.fastNlMeansDenoisingColored() - Procesar imágenes en color.

cv.fastNlMeansDenoisingMulti() - Procesar la secuencia de imágenes capturada en período corto de tiempo (imágenes en escala de grises)

cv.fastNlMeansDenoisingColoredMulti() -Igual que el anterior, pero para imágenes en color.

Los parámetros comunes son:

- h : parámetro que decide la fuerza del filtro. Un valor h más alto elimina mejor el ruido, pero también elimina los detalles de la imagen. (10 está bien)
- hForColorComponents : igual que h, pero solo para imágenes en color. (normalmente igual que h)
- templateWindowSize : debe ser impar. (recomendado 7)
- searchWindowSize : debe ser impar. (recomendado 21)

Ejercicio 3.2.

```

import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
#--leer la imagen
img = cv.imread('images/die.png')
#--eliminar el ruido
dst = cv.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
#--mostrar imágenes
plt.subplot(121), plt.imshow(img)
plt.subplot(122), plt.imshow(dst)
plt.show()

```

4. ACTIVIDAD 3

- **Filtrado de imágenes.**

El filtro es en realidad una pequeña matriz que usaremos para enfocar o desenfocar nuestra imagen original. Para hacer eso, necesitamos realizar una operación de **convolución**. Como siempre ocurre con los conceptos más importantes, refresquemos rápidamente nuestro conocimiento de la convolución.

Como puede ver en la animación GIF a continuación, tenemos una matriz de 6×6 píxeles que representa nuestra imagen. A continuación, realizamos una operación de **convolución** con un filtro de 3×3 . El producto final de este proceso de convolución será una matriz de 4×4 . Para calcular el primer elemento (esquina superior izquierda) de esta matriz de 4×4 , tomamos un filtro de 3×3 y lo colocamos en la parte superior de la región de 3×3 de la imagen de entrada. Luego, tomamos un producto de cada elemento correspondiente y los sumamos como puede ver en la fórmula a continuación.

$$\begin{array}{|c|c|c|c|c|c|} \hline 3 & 0 & 1 & 2 & 7 & 4 \\ \hline 1 & 5 & 8 & 9 & 3 & 1 \\ \hline 2 & 7 & 2 & 5 & 1 & 3 \\ \hline 0 & 1 & 3 & 1 & 7 & 8 \\ \hline 4 & 2 & 1 & 6 & 2 & 8 \\ \hline 2 & 4 & 5 & 2 & 3 & 9 \\ \hline \end{array} \quad 6 \times 6$$

$$* \quad \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad 3 \times 3$$

$$= \quad \begin{array}{|c|c|c|c|} \hline -5 & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \quad 4 \times 4$$

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

Entonces, después de evaluar la primera expresión obtuvimos el resultado de -5. Este será un valor de píxel en la esquina superior izquierda de la imagen de salida. Luego, movemos nuestro filtro a través de la imagen general y creamos una imagen de salida. Observe que nuestro filtro es una matriz con la misma altura y

ancho (3×3 , 5×5 , 9×9). Siempre usamos un número impar porque necesitamos un píxel en el centro de esta matriz.

En un proceso de convolución podemos utilizar una variedad de filtros. Al utilizar diferentes coeficientes de filtro, realizamos efectos de desenfoque, nitidez y otros efectos de procesamiento de imágenes.

- **Suavizado (Smoothing)**

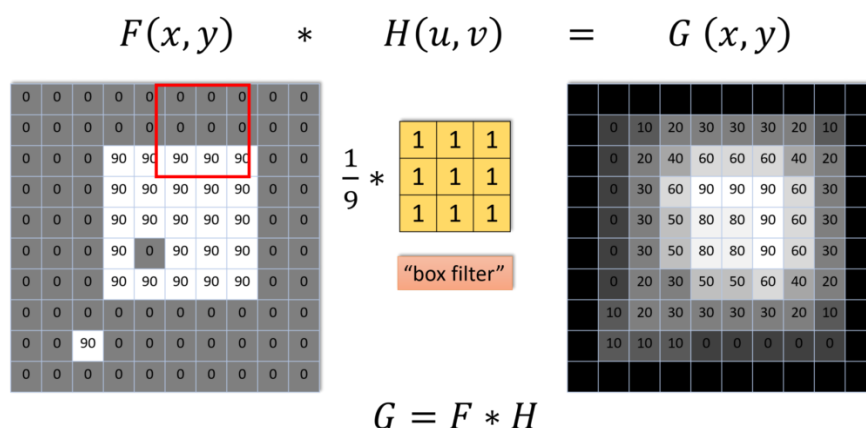
Veamos cómo podemos suavizar o difuminar una imagen. Ahora, puede preguntarse "¿Por qué tengo que difuminar mi imagen"? Bueno, aunque el desenfoque puede ser indeseable en las imágenes, será bastante útil más adelante cuando comencemos a trabajar con funciones OpenCV más avanzadas. Por ejemplo, se utiliza en la detección de bordes y umbrales de imágenes. Además, estas técnicas de desenfoque se utilizan habitualmente para reducir el ruido, y también podemos aplicarlas para reducir el efecto pixelado en imágenes de baja resolución.

OpenCV proporciona una gran cantidad de diferentes tipos de métodos de desenfoque. Entonces, exploremos algunos de ellos.

Filtro de promediado

En la imagen de abajo podemos ver que la imagen de entrada de la izquierda se procesa con el filtro de promedio (filtro de caja). Aquí, todos los valores de los coeficientes tienen el mismo valor de $1/9$. Después de haber aplicado el operador de convolución, hemos generado nuestra imagen de salida a la derecha. Es bueno saber que a medida que aumenta el tamaño del filtro, nuestra imagen se vuelve más borrosa.

Averaging filter

$$F(x, y) * H(u, v) = G(x, y)$$


The diagram illustrates the averaging filter process. It shows a 9x9 input image $F(x, y)$ with a 3x3 region highlighted in red. This region is convolved with a 3x3 kernel $H(u, v)$ where all elements are 1. The result is a 9x9 output image $G(x, y)$ where the central 3x3 region is blurred. The kernel is labeled "box filter".

$$G = F * H$$

Para realizar un promedio en OpenCV usamos las funciones **cv2.blur()** y **cv2.boxFilter()**. Solo se requieren dos argumentos: una imagen que queremos

desenfocar y el tamaño del filtro. Hemos elegido tres tamaños diferentes para el filtro para demostrar que la imagen de salida se volverá más borrosa a medida que aumenta el tamaño del filtro.

El resultado del filtrado se combina y se apila con una función **np.hstack()**.

Ejercicio 3.3.

```
import cv2
import numpy as np
# cargar las imágenes
img1 = cv2.imread("images/marsrover.png")
img2 = cv2.imread("images/lion.jpg")
cv2.imshow("ima1",img1)
cv2.imshow("ima2",img2)

# Realización de difuminado promedio de la primera imagen
# Filters - left (3,3), middle(5,5), right(9,9)
blurred_1 = np.hstack([
    cv2.blur(img1,(3,3)),
    cv2.blur(img1,(5,5)),
    cv2.blur(img1,(9,9))])
cv2.imshow("blurred",blurred_1)
# Realización de difuminado promedio de la segunda imagen
# Filters - left (3,3), middle(5,5), right(9,9)
blurred_2 = np.hstack([
    cv2.blur(img2,(3,3)),
    cv2.blur(img2,(5,5)),
    cv2.blur(img2,(9,9))])
cv2.imshow("blurred2",blurred_2)

cv2.waitKey(0)
```

Ejercicio 3.4

A continuación, revisamos el funcionamiento de la función `medianBlur()` en OpenCV de la siguiente manera:

- Para eliminar las variaciones aleatorias en los valores de píxel de la imagen dada o el ruido, utilizamos el filtro mediano en OpenCV.
- La función `medianBlur()` se usa para eliminar el ruido de la imagen dada.
- La imagen cuyo ruido debe eliminarse se lee mediante la función `imread()`.
- Luego, se aplica la función `medianBlur()` a la imagen junto con la especificación del tamaño del kernel para eliminar el ruido de la imagen.
- La función `medianBlur()` devuelve una imagen con el ruido eliminado de la imagen.

```
import cv2 as cv
#leyendo la imagen cuyo ruido se eliminará usando la función imread()
```

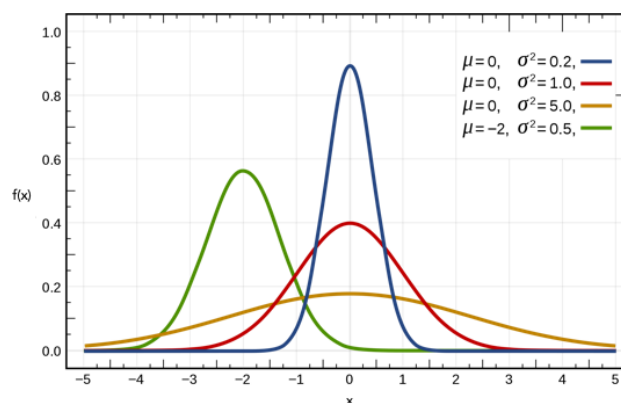
```

imageread = cv.imread('../images/die.png')
#usando la función medianBlur() para eliminar el ruido de la imagen
dada
imagenormal = cv.medianBlur(imageread, 5)
#mostrando la imagen sin ruido como la salida en la pantalla
cv.imshow('original:', imageread)
cv.imshow('Noiseless_image',imagenormal)
cv.waitKey(0)
cv.destroyAllWindows()

```

Filtro gaussiano

Además del filtro de promediado, podemos usar varios otros filtros comunes para realizar imágenes borrosas. Ahora vamos a explorar un filtro un poco más complicado. Es el núcleo más utilizado en el procesamiento de imágenes y se denomina filtro gaussiano. Para la creación de este filtro usamos la famosa función gaussiana. Esta función representa la probabilidad de que los eventos se centren alrededor del valor medio. Además, la desviación estándar (σ) de esta función controla qué tan amplia sería esta distribución. Al muestrear los valores de esta función, obtendremos coeficientes para una matriz de filtro gaussiano. El efecto de diferentes valores (σ) se puede observar en la siguiente imagen.



En nuestra práctica usaremos una función `cv2.GaussianBlur()`. De manera similar a un filtro de promediado, proporcionamos una tupla que representa nuestro tamaño de filtro. Así es como se ve nuestro filtro 3×3 :

Gaussian filter

$$\frac{1}{16} * \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

$H(u, v)$

Ejercicio 3.5.

```
import cv2
import numpy

# leer imagen
src = cv2.imread('../images/lena.jpg', cv2.IMREAD_UNCHANGED)

# aplicar el filtro gaussiano
dst = cv2.GaussianBlur(src, (5,5), cv2.BORDER_DEFAULT)

# mostrar la imagen original y la imagen filtrada
cv2.imshow("Gaussian Smoothing", numpy.hstack((src, dst)))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5. ACTIVIDAD 4

- **Enfocar una imagen**

El proceso de nitidez se usa generalmente para mejorar los bordes de una imagen. Hay muchos filtros que podemos usar, pero uno que puede enfocar nuestra imagen está representado en la matriz a continuación.

-1	-1	-1
-1	9	-1
-1	-1	-1

Como puede ver, este filtro tiene un 9 positivo en un centro, mientras que tiene -1 en todos los demás lugares. Para este filtro en particular, no tenemos una función OpenCV implementada. Por lo tanto, usaremos una función `cv2.filter2D ()` que procesará nuestra imagen con un filtro arbitrario. Este filtro se utiliza a menudo para mejorar la nitidez de las imágenes en color.

Otra versión popular de un filtro de enfocado es el llamado sombrero mexicano o filtro laplaciano. Aquí damos un ejemplo de un filtro 5×5 que usaremos para procesar nuestra imagen. El resultado de este procesamiento se muestra en el ejercicio 3.6. Puede notar que la imagen de salida ha conservado los bordes en nuestra imagen.

EJERCICIOS PRÁCTICOS

Ejercicio 3.6.

```
import cv2
import numpy as np
# cargar las imágenes
img1 = cv2.imread("images/cat.png",1)
# Creando un filtro de enfoque
filter = np.array([[ -1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
# Aplicando la función cv2.filter2D en la imagen
sharpen_img_1=cv2.filter2D(img1,-1,filter)
cv2.imshow("gato original",img1)
cv2.imshow("gato enfocado",sharpen_img_1)

img2 = cv2.imread("images/lena.jpg",1)
# Creating our sharpening filter
filter = np.array([[ -1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
# Applying cv2.filter2D function on our Logo image
sharpen_img_2=cv2.filter2D(img2,-1,filter)
cv2.imshow("lena original",img2)
cv2.imshow("lena enfocado",sharpen_img_2)

cv2.waitKey(0)
```

Ejercicio 3.7.

```
import cv2
import numpy as np
# cargar las imágenes
img1 = cv2.imread("images/cat.png",1)

# Creando el filtro de sombrero mexicano
filter = np.array([[0,0,-1,0,0],[0,-1,-2,-1,0],[-1,-2,16,-2,-1],[0,-1,-2,-1,0],[0,0,-1,0,0]])
# Aplicando la función cv2.filter2D
mexican_hat_img1=cv2.filter2D(img1,-1,filter)
cv2.imshow("cat con filtro",mexican_hat_img1)
```

```

img2 = cv2.imread("images/lena.jpg",1)
# Creando el filtro de sombrero mexicano
filter = np.array([[0,0,-1,0,0],[0,-1,-2,-1,0],[-1,-2,16,-2,-1],[0,-1,-2,-1,0],[0,0,-1,0,0]])
# Aplicando la función cv2.filter2D
mexican_hat_img2=cv2.filter2D(img2,-1,filter)
cv2.imshow("lena con filtro",mexican_hat_img2)

cv2.waitKey(0)

```

6. TAREA

- Aplicar un filtro de la media a la imagen lena_ruido.jpg. Usar una matriz de 31x31 elementos como filtro, del siguiente estilo:

$$\frac{1}{31^2} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}$$

- Cargar la imagen insta.jpg. Introducir ruido gaussiano y ruido sal y pimienta a la imagen. Probar con distintos tipos de filtros hasta obtener una imagen de calidad para los ruidos anteriormente introducidos. ¿Qué tipo de filtro funciona mejor en cada uno de los casos anteriores? Justifica tu respuesta-.

7. REFERENCIAS BIBLIOGRÁFICAS

- Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision With OpenCV, Wiley & Sons Ltd.
- Hafsa Asad, W. R., Nikhil Singh (2020). The Computer Vision Workshop. Birmingham U.K., Pack Publishing.
- Szeliski, R. (2011). Computer Vision Algorithms and Applications. London, Springer.