Universidad Nacional Autónoma de México

Facultad de Ciencias

Procesamiento Digital de Imágenes

Práctica 2
Resolución espacial y de intensidad

29 de septiembre de 2023

PROFESORA:
Dra. María Elena Martíiez Pérez
AYUDANTE:
Miguel Angel Veloz Lucas

ALUMNA:
Janet Illescas Coria

Objetivo

- Observar de manera experimental los efectos de la manipulación de la resolución y cuantización de una imagen.
- Familiarizarse con la manipulación de las imágenes.
- Comprender la noción de adyacencia de pixeles y distancia entre pixeles de una imagen.

Introducción

Hacer el muestreo de una señal significa elegir un conjunto de valores de la señal con el fin de reconstruirla posteriormente. Típicamente, dichos valores se obtienen por medio de la digitalización de los valores coordenados. De modo que la información se pierde excepto en los puntos elegidos.

Matemáticamente esto se representa como la multiplicación de la función continua, la señal, y otra función que es cero en toda la imagen excepto en los puntos correspondientes a la malla.

Un ejemplo gráfico de las implicaciones que tiene el manejo del muestreo es el del cambio de la resolución espacial de una imagen, ya sea reduciéndola o aumentándola. En el primer caso de hace un submuestreo, esto se logra eliminando renglones o columnas de la matriz que representa la imagen. El número y frecuencia con el que esto se haga dependerá de la razón de la reducción. En el segundo caso se hace un sobremuestreo de los valores de la imagen. Uno de los métodos más sencillos para lograrlo es usando la interpolación del vecino más cercano. Sean $A \in \mathbf{M}_{nm}(\mathbf{N})$ una imagen cuyas entradas son $a(i,j)$ y $B \in \mathbf{M}_{2n2m}$ cuyas entradas son $b(k,l)$. La interpolación del vecino más cercano $I_A : B \to B$ está dada por la siguiente expresión:

$$I(b(k,l)) = \begin{cases} a(k/2, l/2) & \text{si } k \bmod(2) = 0 \text{ y } l \bmod(2) = 0, \\ a((k+1)/2, l/2) & \text{si } k \bmod(2) = 1 \text{ y } l \bmod(2) = 0, \\ a(k/2, (l+1)/2) & \text{si } k \bmod(2) = 0 \text{ y } l \bmod(2) = 1, \\ a((k+1)/2, (l+1)/2) & \text{si } k \bmod(2) = 1 \text{ y } l \bmod(2) = 1. \end{cases}$$

Cabe mencionar que la interpolación del vecino más cercano puede resultar en un error bastante notorio para el ojo humano. De modo que este método no es muy usado en aplicaciones que requieran de una resolución fina.

Cuando se usa una computadora, la intensidad de los elementos en una imagen debe ser mapeada en una cantidad discreta de niveles de gris. A esto se le llama cuantización. El número de niveles de gris requerido dependerá, en buena medida, de la aplicación para la que se haga. Típicamente, las imágenes se cuantizan con 256 niveles de gris, en donde cada pixel ocupa un byte de memoria (8 bits). Esta cantidad de niveles de gris produce en el ojo humano la sensación de un cambio gradual en los grises de una imagen. Alternativamente, aplicaciones tales como umbralización sólo requieren dos niveles de gris, y aplicaciones en imagenología médica, tales como los rayos-x, requieren de 256 niveles de gris.

Desarrollo

Resuelve los problemas de la lista siguiente y describe tu solución en cada inciso.

1. Busca una imagen de 1024x1024 pixeles en 256 niveles de gris. Reduce su resolución espacial a 512x512, 256x256, 128x128 y 64x64 pixeles.

   Se creó una función llamada `reduccion_espacial` que recibiera una imagen en formato png o jpg de 1024x1024 píxeles.

   Se crean cuatro nuevas imágenes sin información (negras) de los diferentes tamaños a los que se quiere reducir la resolución de la imagen original. Notamos que entre ellas la resolución se debe reducir a la mitad, por lo que decidí procesarlas en orden.

   Por lo tanto procesamos cada imagen pixel a pixel, copiando los valores que le corresponden de la imagen anterior (o en el caso de la primera imagen a reducir con los valores de la imagen original), estos valores se encuentran multiplicando las coordenadas de la imagen que está siendo procesada por dos.

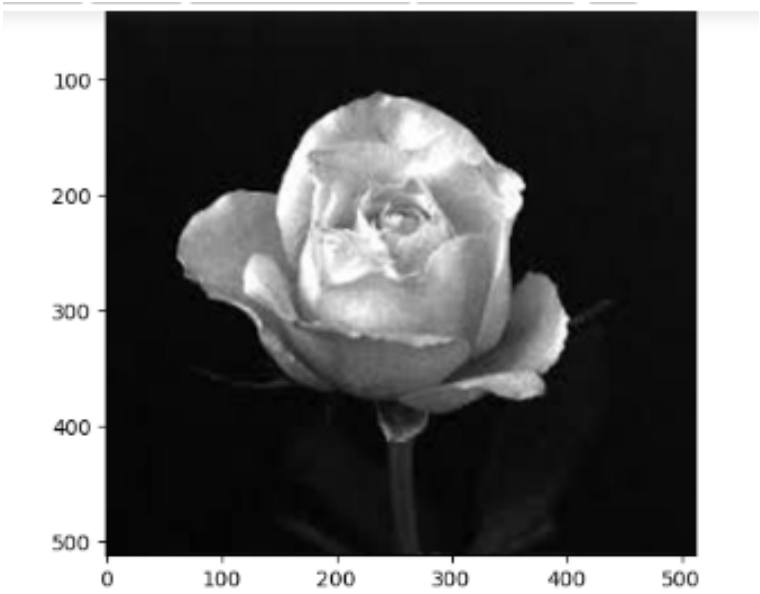
   Al terminar este proceso podemos regresar todas las imágenes ya procesadas y reducidas con sus respectivas resoluciones.

2. Despliega las cuatro imágenes anteriores en tamaño real.

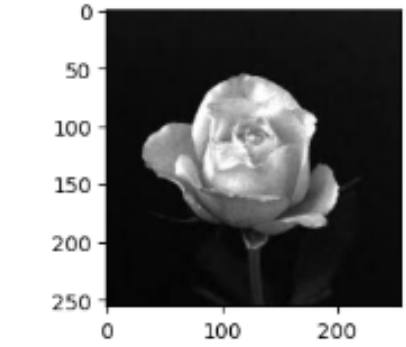
   Una vez ejecutada la función descrita anteriormente con la imagen deseada (en nuestro caso la imagen de la rosa), obtendremos una lista con todas las imágenes reducidas. Recorremos dicha lista y desplegamos una por una.

   Para desplegar las imágenes se utilizó la función imshow() de la biblioteca matplotlib.pyplot, en la cual podemos ver la resolución de la imagen en los ejes y y x. Aclaro esto ya que dicha librería, por defecto, no respeta el tamaño de la imagen al mostrarla y por lo tanto tuve que hacer un cambio en mi código para poder apreciar el cambio de resolución.

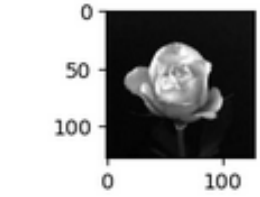
   Línea de código agregada: `plt.figure(figsize=(img.shape[1]/100, img.shape[0]/100))`



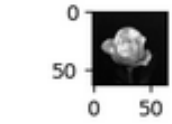

```
<Figure size 640x480 with 0 Axes>
```




```
<Figure size 640x480 with 0 Axes>
```




```
<Figure size 640x480 with 0 Axes>
```



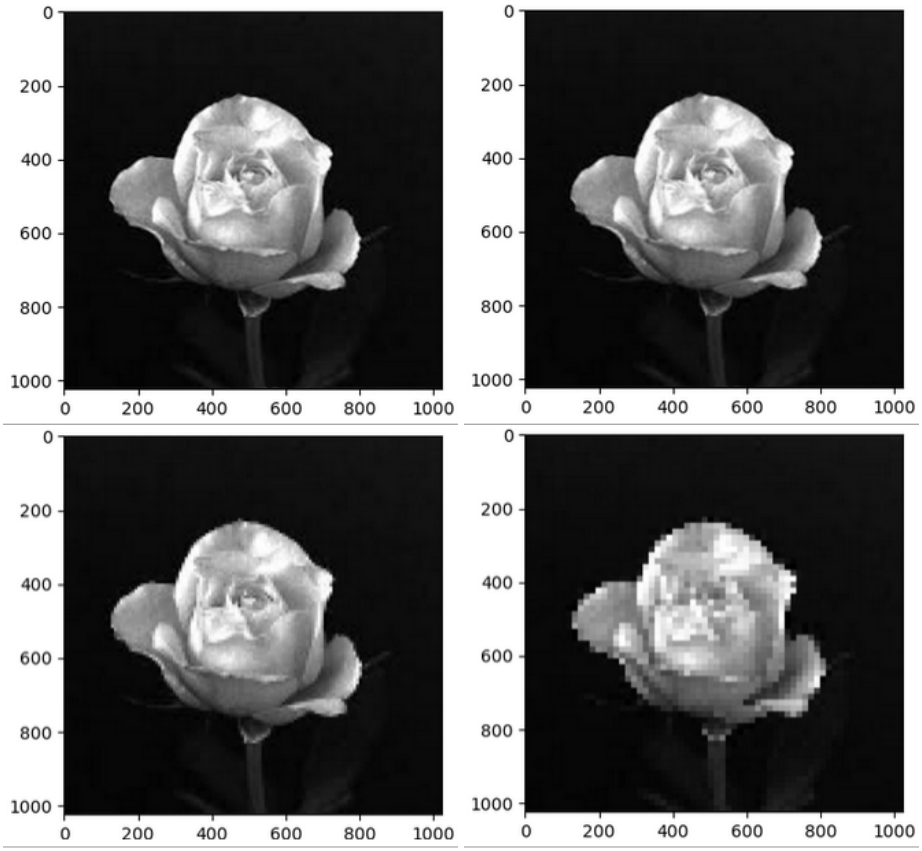
3. Haz un zoom a 1024x1024 pixeles de las cuatro imágenes obtenidas en el inciso 1 usando el método del vecino más cercano.

   Se creó una función llamada `zoom_vecino_mas_cercano` que recibiera una imagen ya representada como una matriz.

   Se crea una nueva imagen sin información (negra) de tamaño 1024x1024. Antes de iniciar el procesamiento obtenemos el factor necesario para escalar la imagen, es decir, dividimos 1024 entre el tamaño de la imagen para obtener el factor que determina cuántas veces se debe aumentar su tamaño para tener una resolución de 1024x1024 pixeles.

   Recorremos pixel por pixel de esta imagen y le asignamos su valor correspondiente en la imagen original, este valor se encuentra haciendo una división entera de la coordenada entre el factor. Al ser una división entera los pixeles que no tienen una relación exacta con los pixeles de la imagen original, tomarán el valor del vecino más cercano.

   Al terminar este proceso se regresa la imagen ya procesada y ampliada. Para hacer zoom a las 4 imágenes, simplemente se aplica la función a cada una de ellas.



4. Reducir la resolución de intensidad del inciso anterior.

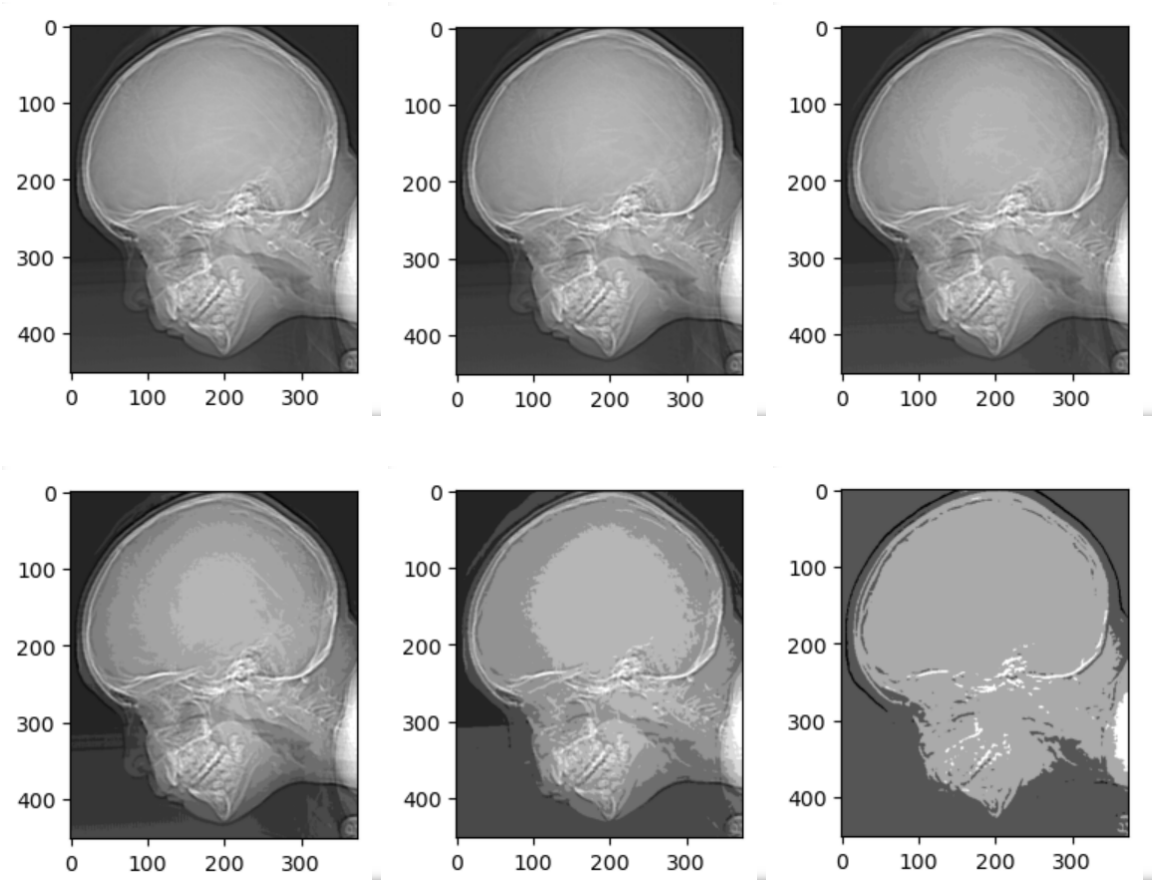
   Se creó una función llamada `reduccion_intensidad` que recibiera una imagen en formato png o jpg.

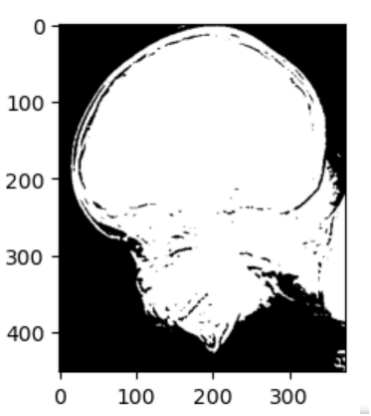
   Por cada intensidad a la que se va a reducir la imagen (128, 64, 32, 16, 8, 4, 2), creamos una nueva imagen sin información (negra) del tamaño de la imagen recibida y la procesamos pixel por pixel. Para obtener el nuevo valor del pixel primero dividiremos su valor en la imagen original entre 255 (para generar un número entre el 0 y 1) y luego es multiplicado por la intensidad a la que se debe escalar menos uno (ya que el 0 se considera parte del rango de intensidades), y por último, como se pueden obtener números decimales hacemos un redondeo para que el valor sea un entero.

   Al terminar este proceso podemos regresar todas las imágenes generadas y reducidas en intensidad.

5. Desplegar las imágenes del inciso anterior.

   Una vez ejecutada la función descrita anteriormente con la imagen deseada (en nuestro caso la imagen de los rayos x), obtendremos una lista con todas las imágenes reducidas en intensidad. Recorremos dicha lista y desplegamos una por una.





6. Busca una imagen de 64x64 pixeles. Escoge 5 pixeles aleatoriamente dentro de la imagen y determina los pixeles 4-adyacentes y 8-adyacentes.

   Se creó una función llamada `crea_ady` que recibiera una imagen en formato png o jpg.

   Primero crea un par de coordenadas aleatorias, entre 0 y el tamaño de la imagen en su respectivo eje menos uno, por cada uno de los 5 pixeles que se deben generar.

   Por cada uno de estos nuevos pixeles calculamos sus cuatro adyacencias, simplemente sumando o restando 1 a la coordenada en el eje y o la coordenada en el eje x.

   De forma similar, calculamos las ocho adyacencias para cada pixel, simplemente sumando o restando 1 a la coordenada en el eje y y/o la coordenada en el eje x (todos los pixeles alrededor de él, incluidos los diagonales).

   Al terminar este proceso regresamos los pixeles generados aleatoriamente, sus respectivas 4 adyacencias y 8 adyacencias.

7. Desplegar las vecindades obtenidas de los 5 pixeles escogidos en el inciso anterior.

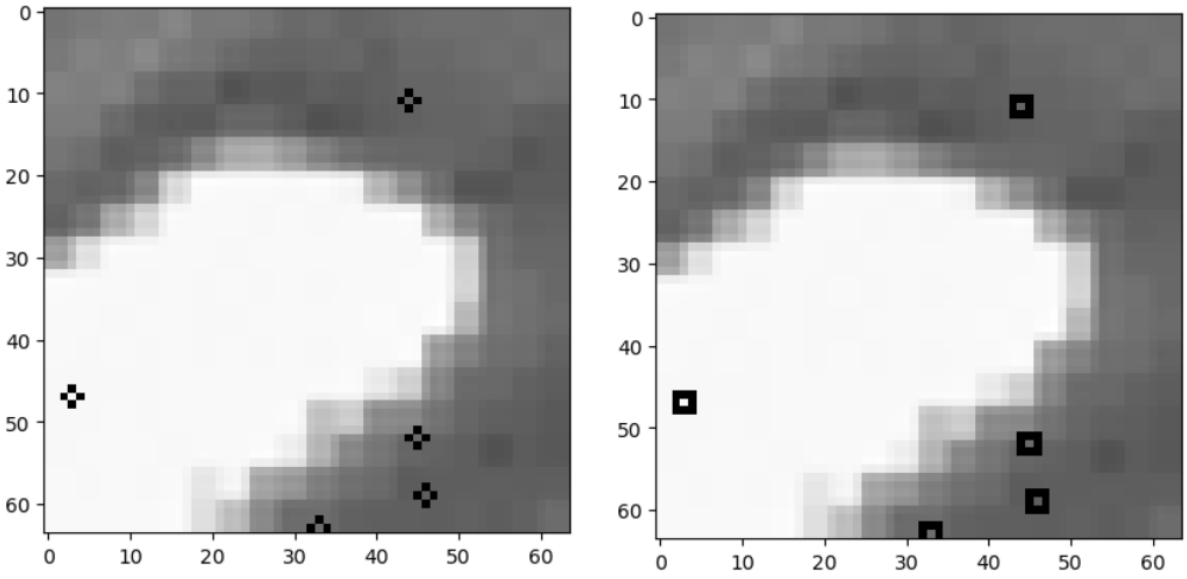
   Una vez ejecutada la función descrita anteriormente con la imagen deseada (en nuestro caso la imagen cuadro), obtendremos los pixeles 5 aleatorios, sus 4 adyacencas y 8 adyacencias.

   Para poder desplegar las adyacencias, antes debemos agregar dichas adyacencias a la imagen por lo que se creó otra función llamada `agregar_vecindades` que recibe la imagen y las vecindades obtenidas (ya sea las 4 adyacencias o las 8 adyacencias).

   Recorremos cada pixel de cada vecindad, marcamos el pixel en la imagen asignándole el valor de 0 para poder apreciarlo cuando se despliegue. Aquí es importante resaltar que en cada caso verificamos que el pixel a marcar se encuentre dentro del rango espacial de la imagen.

   Una vez terminado el proceso, regresamos la imagen con la vecindades marcadas.

Finalmente podemos ejecutar esta función dos veces para obtener la imagen que contenga las 4 adyacencias y otra que contenga las 8 adyacencias de los cinco pixeles. Deplegamos ambas imágenes con la función imshow.



8. Toma uno de los pixeles antes escogidos como referencia, y calcula la distancia de ese pixel a los otros 4. Has lo anterior utilizando a) la métrica City Block y b) la Chessbord.

Creamos dos funciones:

- `city_block` para calcular la distancia City Block entre dos pixeles dados con la siguiente operación:

$$|y_1 - y_2| + |x_1 - x_2|$$

- `chessbord` para calcular la distancia Chessbord entre dos pixeles dados con la siguiente operación:

$$max(|y_1 - y_2|, |x_1 - x_2|)$$

Donde y_1 y x_1 representan las coordenadas en el eje y y x del primer pixel, y y_2 y x_2 las coordenadas en el eje y y x del segundo pixel.

Primero desplegamos la información de las distancias usando la métrica City Block, aclaramos a partir de que pixel (el primer pixel de la lista de pixeles) se calcularán las distancias a los demás pixeles. Recorremos el resto de los pixeles mostrando sus coordenadas, seguido de su distancia al primer pixel (ejecutando `city_block` con ambos pixeles para obtenerla).

Similar para las distancias con la métrica Chessbord, pero en este caso utilizando la función `chessbord`.

```
Distancias City Block desde (63, 33):
  (52, 45): 23
  (11, 44): 63
  (59, 46): 17
  (47, 3): 46
Distancias Chessbord desde (63, 33):
  (52, 45): 12
  (11, 44): 52
  (59, 46): 13
  (47, 3): 30
```

Conclusiones

Durante el procesamiento de las diversas imágenes proporcionadas, se llevaron a cabo cambios pixel a pixel en relación con una imagen de referencia. En algunas situaciones, como la reducción de la resolución espacial o el aumento de zoom en una imagen, fue necesario considerar un factor que determinara cuántas veces se debía aumentar o reducir su tamaño. En general, se requirió un análisis matemático de las operaciones a realizar y sus objetivos.

Esta práctica me permitió una comprensión más sólida de los conceptos abordados en el segundo capítulo del curso. Por ejemplo, se exploraron conceptos como el muestreo (realizado en los ejercicios 1 y 3), que se centra en las coordenadas que le corresponden a los pixeles, y la cuantización (realizada en el ejercicio 4), que se enfoca en los valores de intensidad de los pixeles.

Los ejercicios finales se centraron en la relación espacial entre los pixeles, abordando conceptos como los tipos de adyacencia y las distancias entre pixeles mediante métodos que hasta antes de este curso desconocía.

Esta práctica me ayudó a familiarizarme con el manejo de imágenes en Python y, al mismo tiempo, me permitió visualizar las modificaciones visuales de los procesos que estábamos llevando a cabo. Esto contribuyó a una mejor comprensión del impacto de dichos procesos en la imagen y a consolidar mis conocimientos en ciertas definiciones clave.

Referencias

1. Gonzalez, R., Woods, R., Digital Image Processing, Prentice Hall, 2008.
2. Pratt, W. k., Digital Image Processing, John Wiley and Sons Inc, 2001.
3. Jahne, B., Digital Image Processing, Springer,2005.