Universidad Nacional Autónoma de México

Facultad de Ciencias

Proceso Digital de Imágenes

Práctica 02
Resolución espacial y de intensidad

Autor:

Delgado Díaz Hermes Alberto
319258613

23 de Septiembre de 2025

Objetivo

- Observar de manera experimental los efectos de la manipulación de la resolución y cuantización de una imagen.
- Familiarizarse con la manipulación de 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. Típicamene, dichos valores se obtienen por medio de la digitalización de los valores de 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 se 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 $A \in \mathbb{M}_{n\times m}(\mathbb{N})$ una imagen cuyas entradas son $a(i,j)$ y $B \in \mathbb{M}_{2n\times 2m}$ cuyas entradas son $b(k,l)$. La interpolación del vecino más cercano $I_A : B \to B$ está dada por la siguiente expresión:

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

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 ($8bits$). 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 1024×1024 píxeles en 256 niveles de gris. Reduce su resolución espacial a 512×512, 256×256, 128×128 y 64×64 píxeles.

   Se hace un *submuestreo* por 2 repetidamente: de cada bloque 2×2 de la imagen se toma un solo píxel representativo(el de la esquina superior izquierda en la implementación) y así reduce a la mitan en ambos ejes en cada paso, encadenándolo cuatro veces para llegar a una escala de 64×64 píxeles.

2. Despliega las cuatro imágenes anteriores en tamaño real.

   Tras generar cada versión reducida, se muestran en figuras separadas con `matplotlib` sin reescalar, de modo que cada imagen se despliega "a su tamaño" (mismas dimensiones de su matriz).

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

   Para volver 1024×1024 desde cada versión reducida, se usa replicación de píxeles: cada píxel de la imagen fuerte se copia en un bloque 2×2 (o el factor que toque) de la imagen destino. Eso implementa el **vecino más cercano**.

4. Reducir la resolución de intensidad, cuantización, de la imagen original a $128, 64, 32, 16, 8, 4$ y 2.

   Se aplica cuantización de niveles de gris con la regla

   $$nuevo = px - px \bmod nivel$$

   Al aumentar el parámetro `nivel`$(2, 4, 8, 16, \cdot)$ se eliminan detalles finos y se conservan menos niveles efectivos(hasta llegar a 2).

5. Desplegar las imágenes del inciso anterior.

   Igual que en el inciso 2, cada versión cuantizada se muestra en su propia figura, indicando en el título el nivel/escala correspondiente para facilitar la comparación visual de la pérdida de detalle al subir el paso de cuantización

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

   Se cargan los datos de la imagen 64×64. Se eligen 5 píxeles al azar(sin reemplazo). Para cada píxel (y, x), los **4-adyacentes** son arriba/abajo/izquierda/derecha (dentro de límite), y los **8-adyacentes** agregan las diagonales.

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

   Se genera una copia **RGB** donde: vecinos 4 se marcan rojo, diagonales (8 pero no 4) azul, y el píxel central amarillo. Luego se muestran imagen original y la marcada para visualizar las vecindades.

8. Toma uno de los pixeles antes elegidos como referencia, y calcula la distancia de ese pixel a los otros 4.

   Tomando el primero de los 5 como referencia, se calcula hacia cada uno de los otros 4:

   - `City-Block:` $|x_1 - x_2| + |y_1 y_2|$
   - `Chessboard:` $max(|x_1 - x_2|, |y_1 - y_2|)$

Código

Estructura

- `Main.py`: punto de entrada, muestra menú, despacha a cada ejercicio y controla la interacción.
- `EscalarMitad.py`: reducción espacial(mitad) y visualización (Ej 1 y 2).
- `EscalarDoble.py`: escalado al doble y visualización (Ej 3).
- `EscalarGrises.py`: cuantización (reducción de niveles de gris) y visualización (Ej. 4 y 5).
- `Adyacencias.py`: cálculo de vecindades 4 y 8, pintado de vecinos y distancias City-Block / Chessboard (Ej. 6, 7 y 8).

Módulos y funciones clave

Main

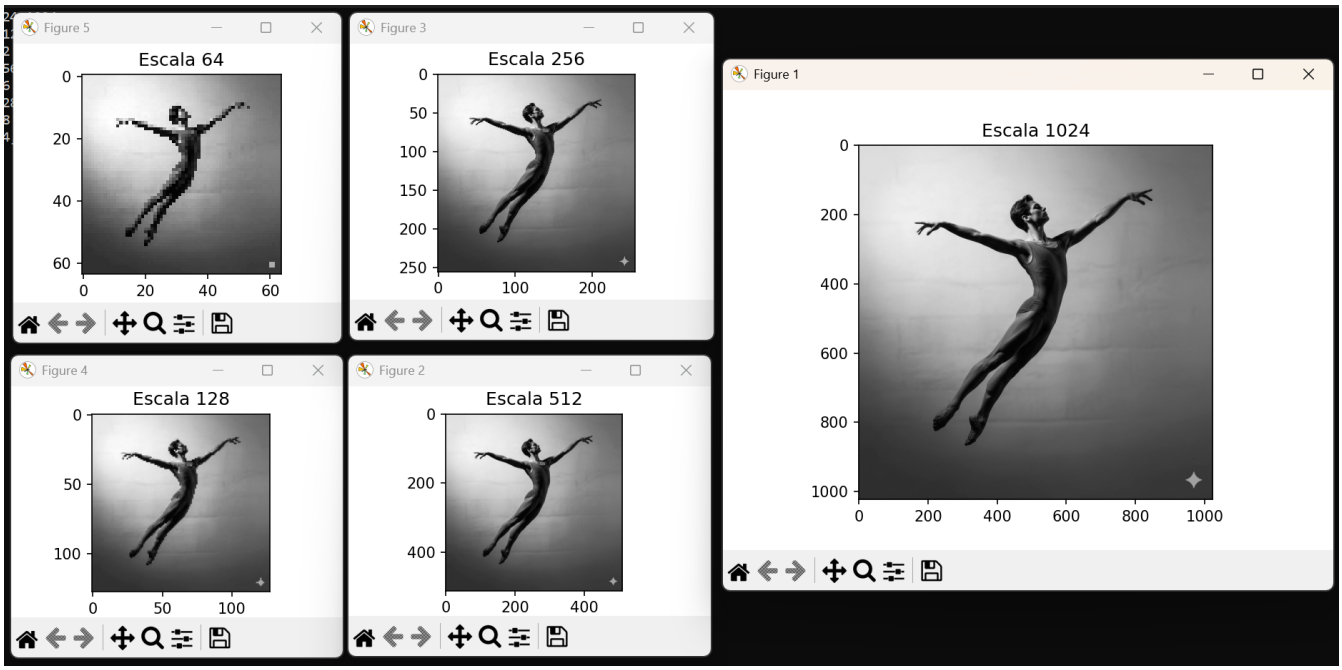
- `menu(), limpiarPantalla(), pausar(), ejecutarOpcion(opcion)` coordinan todo el flujo, el **dispatcher** llama a cada módulo según la opción $1 - 4$.

```
==================================================
        PRÁCTICA 2 — MENÚ DE EJERCICIOS
==================================================
1. Escalar imagen a la mitad (Ejercicio 1 y 2)
2. Escalar imagen al doble (Ejercicio 3)
3. Reducir niveles de gris (Ejercicio 4 y 5)
4. Adyacencias y distancias (Ejercicio 6, 7 y 8)
0. Salir
--------------------------------------------------
Elige una opción:
```

Menú del programa

EscalarMitad

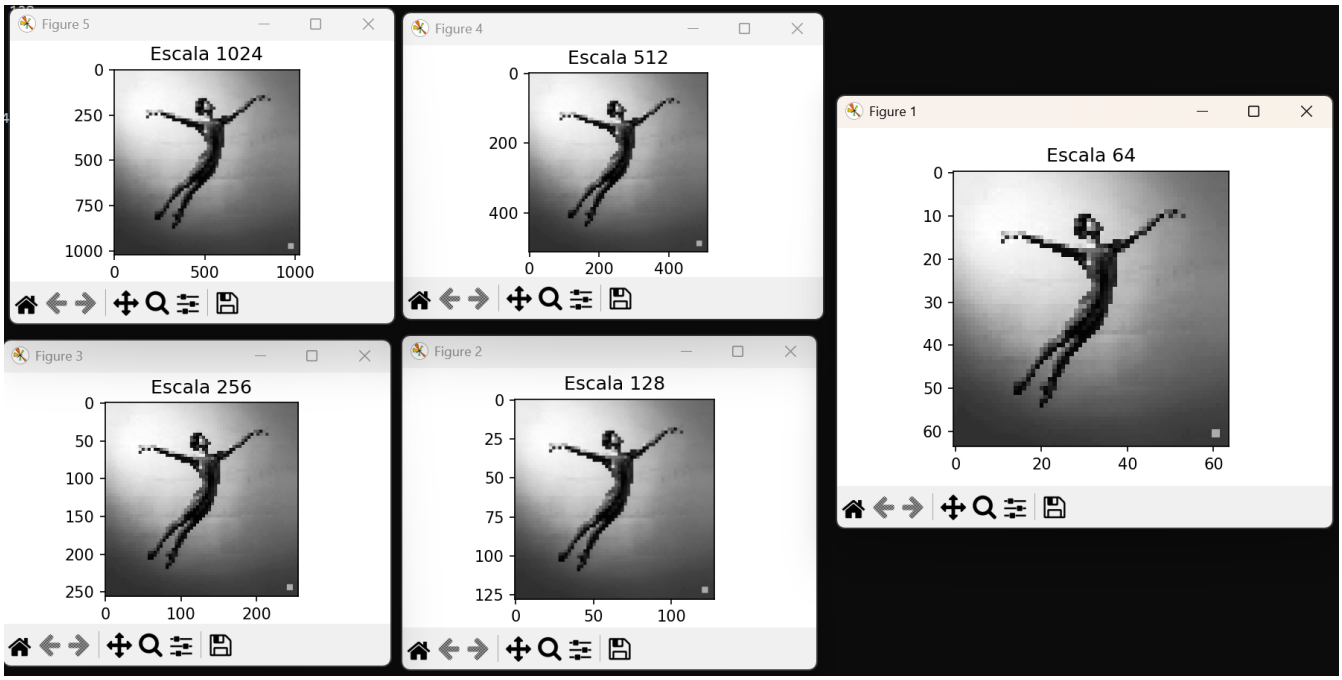
- `imagenEscalaMitad(imagen)`: submuestreo por 2 tomando un píxel por bloque 2×2.
- `iniciaProblema(ruta_imagen)`: carga 1024×1024 (canal 0 si es RGB), aplica 4 reducciones sucesivas y muestra cada escala.



Ejemplo

EscalarDoble

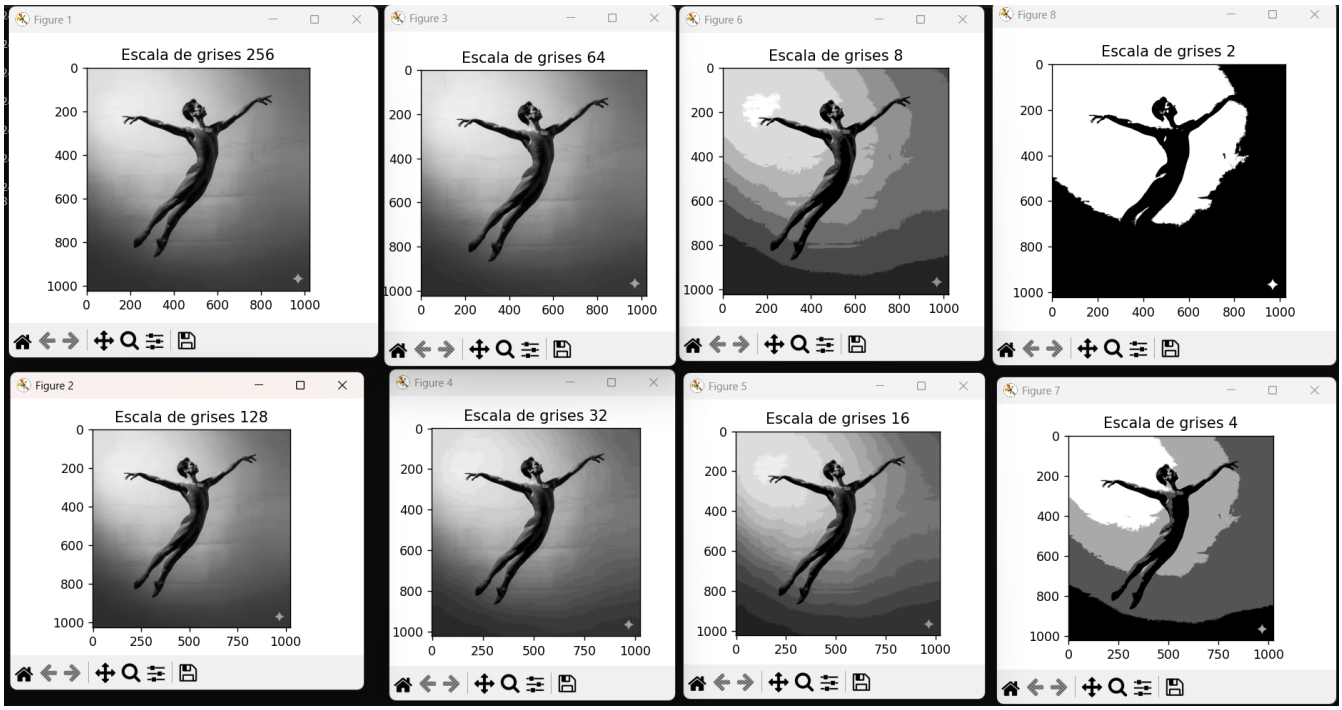
- `imagenEscalaDoble(imagen)`: replicación 2×2 de cada píxel.
- `iniciaProble(ruta_imagen)`: parte de 64×64 y duplica iterativamente hasta 1024×1024; muestra cada etapa.



Ejemplo

EscalarGrises

- `imagenIntensidad(imagen,nivel)`: cuantiza con $px - (px \,\%nivel)$.
- `iniciaProblema(ruta_imagen)`: carga a grises, genera versiones con niveles crecientes $(2, 4, 8, 16, \cdot)$ y las muestra.



Ejemplo

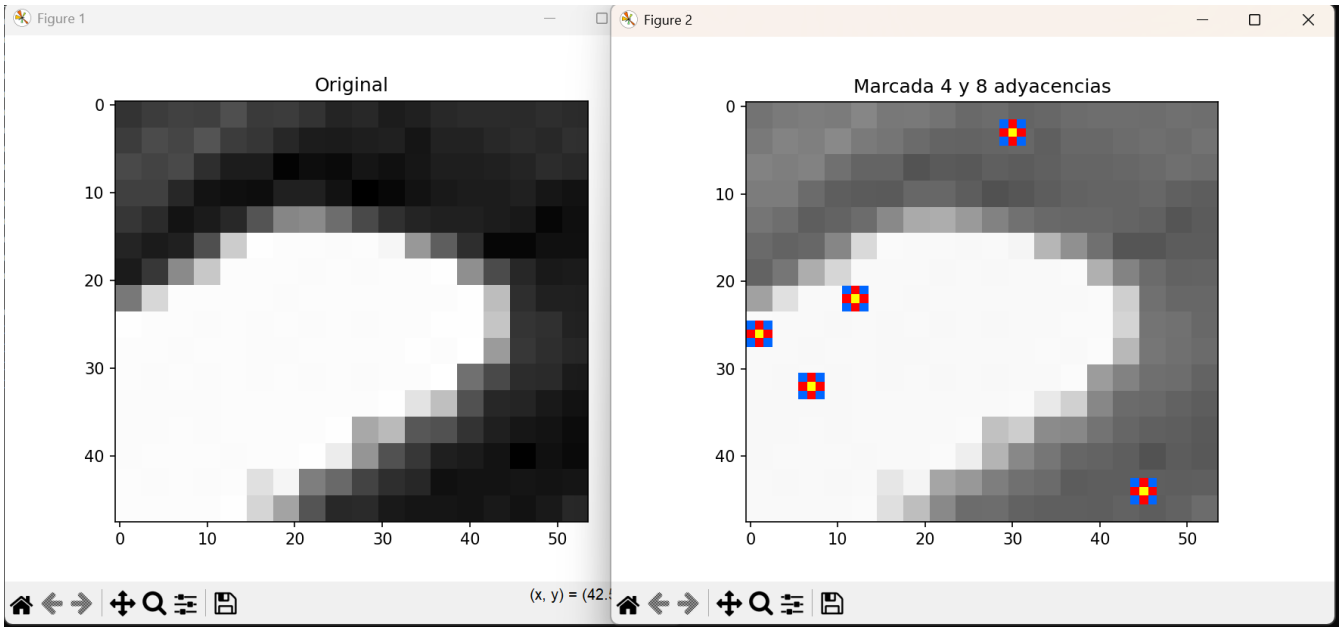
Adyacencias

- `vecinos4(y,x,H,W)` y `vecinos8(y,x,H,W)`: calculan vecinos válidos de un píxel en 4 y 8 direcciones.
- `imagenAdyacencia(imagen, n_pixeles)`: selecciona 5 píxeles al azar y lista sus vecindades.
- `marcarVecinosColores(ruta, coordenadas, r, salida)`: pinta los vecinos en colores (rojo, azul y amarillo) y guarda la imagen.
- `distanciaCityBlock(p,q)` y `distanciaChessboard(p,q)`: calculan distancias entre dos píxeles con métricas distintas.
- `distanciaEntrePuntos(coordenadas)`: toma el primer píxel como referencia y calcula distancias a los demás.
- `iniciaEjercicio(ruta_imagen, ruta_imgPintada)`: orquesta selección de píxeles, cálculo de distancias y visualización.

```
Pixel seleccionado: (32,7)
4-adyacencia: [(33, 7), (31, 7), (32, 8), (32, 6)]
8-adyacencia: [(33, 8), (33, 6), (31, 8), (31, 6), (33, 7), (31, 7), (32, 8), (32, 6)]
Pixel seleccionado: (3,30)
4-adyacencia: [(4, 30), (2, 30), (3, 31), (3, 29)]
8-adyacencia: [(4, 31), (4, 29), (2, 31), (2, 29), (4, 30), (2, 30), (3, 31), (3, 29)]
Pixel seleccionado: (44,45)
4-adyacencia: [(45, 45), (43, 45), (44, 46), (44, 44)]
8-adyacencia: [(45, 46), (45, 44), (43, 46), (43, 44), (45, 45), (43, 45), (44, 46), (44, 44)]
Pixel seleccionado: (22,12)
4-adyacencia: [(23, 12), (21, 12), (22, 13), (22, 11)]
8-adyacencia: [(23, 13), (23, 11), (21, 13), (21, 11), (23, 12), (21, 12), (22, 13), (22, 11)]
Pixel seleccionado: (26,1)
4-adyacencia: [(27, 1), (25, 1), (26, 2), (26, 0)]
8-adyacencia: [(27, 2), (27, 0), (25, 2), (25, 0), (27, 1), (25, 1), (26, 2), (26, 0)]

Píxel de elegido como referencia: (32, 7)
  -> hacia (3, 30): CityBlock=52, Chessboard=29
  -> hacia (44, 45): CityBlock=50, Chessboard=38
  -> hacia (22, 12): CityBlock=15, Chessboard=10
  -> hacia (26, 1): CityBlock=12, Chessboard=6
```

Ejemplo de como se ve en terminal



Ejemplo de visualización

Comando de ejecución

Se requiere no cambiar nombres de carpetas ni de archivos para que funciones correctamente, además cambia dependiendo el sistema operativo. Dentro de la carpeta src:

Windows

```
src/> python Main.py
```

Linux

```
src/> python3 Main.py
```

macOS

```
src/> python3 Main.py
```

Conclusión

La práctica me permitió comprender cómo se pueden manipular imágenes a nivel básico aplicando operaciones de reducción y aumento de resolución, así como cambios en la intensidad de los píxeles. También aprendí a calcular vecindades 4 y 8 y a medir distancias entre píxeles con diferentes métricas. El uso de módulos separados facilitó organizar el código y, con el menú principal, fue posible ejecutar cada ejercicio de manera clara y ordenada. En general, la práctica me permitió relacionar conceptos vistos en clase con ejemplos prácticos y visuales utilizando Python.

Referencias

- Pillow Documentation: https://pillow.readthedocs.io/en/stable/
- NumPy – Absolute Beginners Guide: https://numpy.org/doc/stable/user/absolute_beginners.html
- Matplotlib Quick Start: https://matplotlib.org/stable/users/explain/quick_start.html#axis-scales-and-ticks
- Recursos Python – Cómo limpiar la consola: https://micro.recursospython.com/recursos/como-limpiar-la-consola.html