Nivel 4 – Laboratorio 1
Visor de imágenes con matrices

Objetivos

1. Practicar el manejo de matrices representadas como listas de listas.
2. Implementar algoritmos de recorrido y transformación de matrices.
3. Familiarizarse con estructuras complejas de datos, como listas de listas de tuplas.
4. Aplicar operaciones matemáticas sobre matrices para transformar imágenes digitales.

Preparación del ambiente de trabajo

Para este laboratorio se le proporcionará un módulo de consola (`consola_visor_imagenes.py`) que contiene un menú interactivo para probar sus funciones.

Usted debe completar el módulo de funciones llamado `visor_imagenes.py` en el que escribirá las funciones correspondientes a las actividades de este laboratorio. La consola importará automáticamente sus funciones desde este módulo.

También se le proporcionarán tres archivos de imagen en formato PNG (`imagen1.png`, `imagen2.png`, `imagen3.png`) que podrá utilizar para probar las transformaciones implementadas.

Representación de imágenes

En este laboratorio trabajaremos con imágenes digitales en formato PNG (Portable Network Graphics). Para poder realizar transformaciones sobre estas imágenes, la información de cada pixel se guarda en una matriz.

La matriz que utilizaremos tiene el formato de lista de listas de tuplas, donde:

- La lista externa representa las filas de la imagen (altura).
- Cada lista interna representa una fila de pixeles (ancho).
- Cada tupla representa un pixel con sus tres componentes de color en formato RGB.

El color de un pixel está representado en el sistema RGB (Red-Green-Blue), es decir, un componente rojo, uno verde y uno azul. Estos componentes son números de punto flotante que van de $0{,}0$ a $1{,}0$.

Ejemplo de representación:

Una imagen de 3×3 pixeles se representa de la siguiente manera:

```
[
  [(0.0, 1.0, 0.0), (0.0, 0.0, 1.0), (1.0, 0.0, 0.0)],
  [(1.0, 0.39, 0.0), (0.59, 0.0, 1.0), (0.0, 1.0, 0.78)],
  [(0.0, 0.0, 0.0), (0.39, 0.39, 0.39), (1.0, 1.0, 1.0)]
]
```

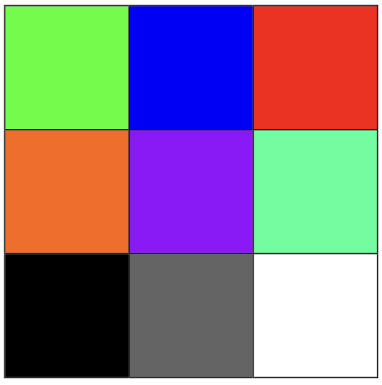

Figura 1: Ejemplo de una imagen de 3x3 pixeles.

En este ejemplo:

- En la posición `[0][0]` hay un pixel verde con componentes (R: $0{,}0$, G: $1{,}0$, B: $0{,}0$)
- En la posición `[0][1]` hay un pixel azul con componentes (R: $0{,}0$, G: $0{,}0$, B: $1{,}0$)
- En la posición `[2][2]` hay un pixel blanco con componentes (R: $1{,}0$, G: $1{,}0$, B: $1{,}0$)

Funciones proporcionadas

El módulo `visor_imagenes.py` ya incluye dos funciones implementadas:

- `cargar_imagen(ruta_imagen: str) ->list`: Carga una imagen PNG desde la ruta especificada y la retorna como una lista de listas de tuplas.
- `visualizar_imagen(imagen: list) ->None`: Muestra la imagen en una ventana utilizando `matplotlib`.

Estas funciones ya están completamente implementadas y **no deben ser modificadas**. Úselas para cargar y visualizar las imágenes mientras implementa las transformaciones.

Especificaciones técnicas

Todas las funciones de transformación deben:

- Recibir una matriz de imagen como lista de listas de tuplas.
- Retornar una nueva matriz de imagen (no modificar la original).

Recuerde que las tuplas son inmutables. Para modificar un pixel, debe crear una nueva tupla. Los elementos de las tuplas deben ser de tipo `float`.

Para los ejemplos a continuación, se utilizará la siguiente imagen de la pintura Las señoritas de Avignon de Pablo Picasso.

Figura 2: Ejemplo de imagen original antes de aplicar transformaciones.

Actividad 1: Binarizar imagen

Complete la función `binarizar_imagen` que recibe como parámetros una matriz de imagen (lista de listas de tuplas) y un umbral (`float` entre $0{,}0$ y $1{,}0$), y retorna una nueva matriz con la imagen binarizada.

La binarización consiste en convertir todos los colores de la imagen a solo dos colores: negro $(0{,}0, 0{,}0, 0{,}0)$ o blanco $(1{,}0, 1{,}0, 1{,}0)$.

Proceso:

1. Para cada pixel, calcular el promedio de sus tres componentes RGB.
2. Si el promedio es mayor o igual al umbral, reemplazar el pixel por blanco $(1{,}0, 1{,}0, 1{,}0)$.
3. Si el promedio es menor al umbral, reemplazar el pixel por negro $(0{,}0, 0{,}0, 0{,}0)$.

Fórmula del promedio:

$$\text{promedio} = \frac{R + G + B}{3}$$

Ejemplo:

Con un umbral de $0{,}5$:

- Un pixel $(0{,}8, 0{,}9, 0{,}7)$ tiene promedio $0{,}8 \geq 0{,}5$, se convierte en $(1{,}0, 1{,}0, 1{,}0)$ (blanco).
- Un pixel $(0{,}2, 0{,}3, 0{,}1)$ tiene promedio $0{,}2 < 0{,}5$, se convierte en $(0{,}0, 0{,}0, 0{,}0)$ (negro).

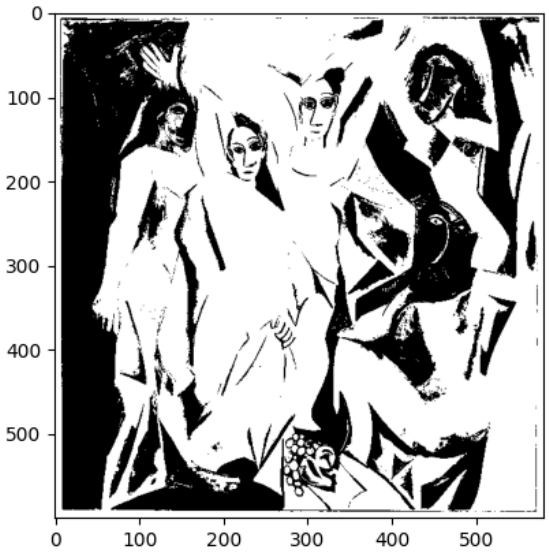

Figura 3: Ejemplo de binarización con umbral $0{,}5$ aplicada a una imagen.

Actividad 2: Convertir imagen a negativo

Complete la función `convertir_negativo` que recibe como parámetro una matriz de imagen (lista de listas de tuplas) y retorna una nueva matriz con la imagen en negativo.

Para calcular el negativo de una imagen se debe calcular el color negativo de cada pixel. Para ello, se resta cada componente RGB de $1{,}0$ (el valor máximo). Con los nuevos valores de los componentes se forma cada nuevo color.

Fórmula:

Para cada componente RGB del pixel:

$$\text{nuevo\_componente} = 1{,}0 - \text{componente\_original}$$

Ejemplo:

Si un pixel tiene el color $(0{,}2, 0{,}8, 0{,}5)$, su negativo sería $(0{,}8, 0{,}2, 0{,}5)$.

Nota importante: Como las tuplas son inmutables, debe crear una nueva tupla para cada pixel y construir una nueva matriz completa con estos nuevos pixeles.

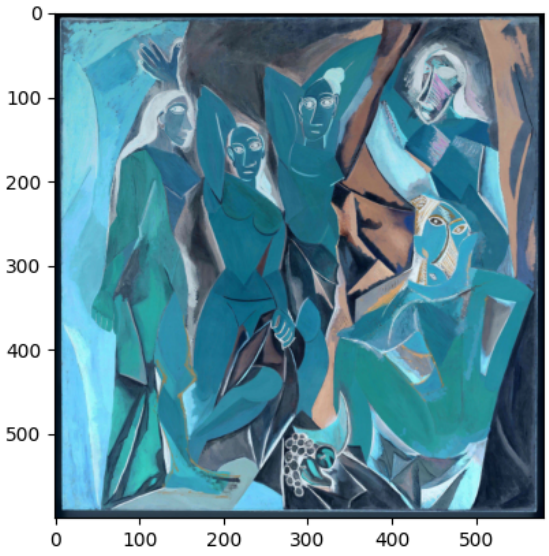

Figura 4: Ejemplo de transformación a negativo aplicada a una imagen.

Actividad 3: Convertir a escala de grises

Complete la función `convertir_a_grises` que recibe como parámetro una matriz de imagen (lista de listas de tuplas) y retorna una nueva matriz con la imagen en escala de grises.

Para convertir la imagen a escala de grises, se promedian los tres componentes RGB de cada pixel y se crea un nuevo color donde cada componente tiene el valor de dicho promedio.

Proceso:

1. Para cada pixel, calcular el promedio de sus tres componentes RGB.
2. Crear un nuevo pixel donde $R = G = B =$ promedio.

Fórmula:

$$\text{gris} = \frac{R + G + B}{3}$$

$$\text{nuevo\_pixel} = (\text{gris}, \text{gris}, \text{gris})$$

Ejemplo:

Un pixel con color $(0{,}8, 0{,}5, 0{,}2)$ tiene un promedio de $0{,}5$, por lo que se convierte en $(0{,}5, 0{,}5, 0{,}5)$.

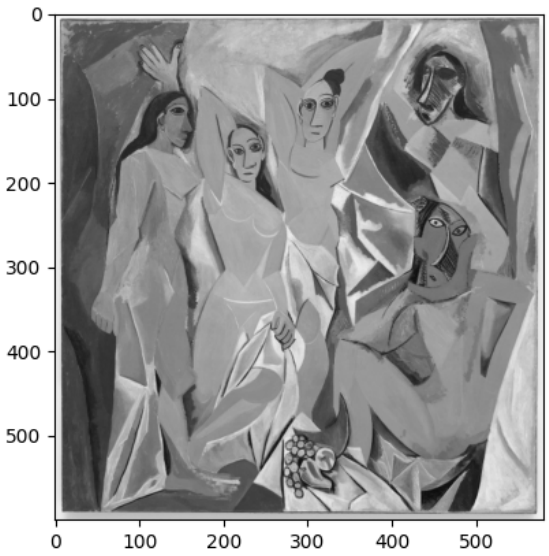

Figura 5: Ejemplo de conversión a escala de grises aplicada a una imagen.

Actividad 4: Reflejar imagen verticalmente

Complete la función `reflejar_imagen` que recibe como parámetro una matriz de imagen (lista de listas de tuplas) y retorna una nueva matriz con la imagen reflejada verticalmente.

Reflejar la imagen verticalmente consiste en crear una imagen espejo sobre una línea imaginaria vertical en el centro de la figura. Para hacer esta transformación, se intercambian las columnas de pixeles de la imagen: la primera columna con la última, la segunda con la penúltima, etc.

Proceso:

- Para cada fila de la imagen, invertir el orden de los pixeles (tuplas).
- La primera columna se convierte en la última, la segunda en la penúltima, y así sucesivamente.

Ejemplo:

Si una fila contiene los pixeles `[A, B, C, D]`, después de reflejar quedaría `[D, C, B, A]`.

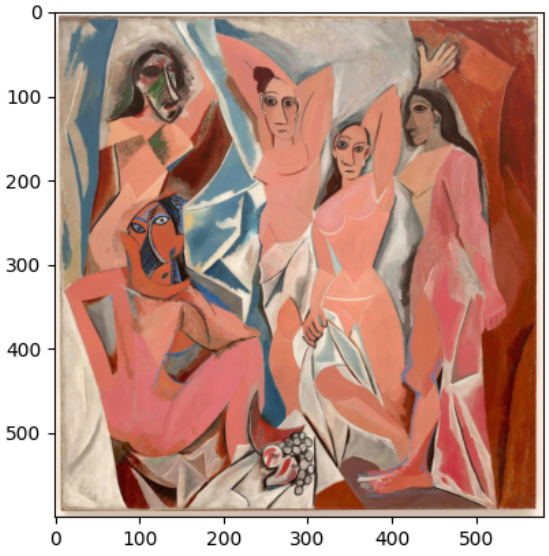

Figura 6: Ejemplo de reflexión vertical aplicada a una imagen.

Actividad 5: Convolución de imagen

Complete la función `convolucion_imagen` que recibe como parámetro una matriz de imagen (lista de listas de tuplas) y retorna una nueva matriz con la convolución aplicada.

La convolución es una operación que transforma cada pixel utilizando sus pixeles vecinos y una matriz de factores. Dependiendo de la matriz de convolución elegida, se pueden producir diferentes efectos visuales en la imagen.

Matrices de convolución y sus efectos:

1. Afilado (Sharpen) - Resalta detalles y bordes, hace la imagen más nítida:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

2. Desenfoque (Blur) - Suaviza la imagen, reduce detalles:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

3. Detección de bordes - Resalta solo los contornos y transiciones:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

4. Relieve (Emboss) - Crea un efecto de relieve o grabado:

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Para este laboratorio, implemente la función utilizando la matriz de **afilado (opción 1)** como matriz por defecto. Sin embargo, una vez implementada la función, puede experimentar cambiando la matriz en el código para observar los diferentes efectos que produce cada una.

Proceso para cada pixel:

1. Alinear el centro de la matriz de convolución con el pixel a procesar.
2. Para cada componente de color (R, G, B) por separado:
   - Multiplicar cada pixel vecino (incluyendo el central) por el factor correspondiente de la matriz.
   - Sumar todos estos productos.
   - Dividir la suma por la suma total de los factores que se utilizaron.

3. Crear un nuevo pixel con los valores resultantes para R, G y B.

Manejo de bordes:

Para pixeles en los bordes de la imagen, solo se utilizan los vecinos que existen. Por ejemplo:

- Un pixel en la esquina superior izquierda solo tiene 4 vecinos (él mismo, derecha, abajo, diagonal abajo-derecha).
- Un pixel en el borde superior tiene 6 vecinos.
- Un pixel en el centro tiene los 9 vecinos completos.

La suma de factores se ajusta según cuántos vecinos fueron efectivamente utilizados.

Ejemplo simplificado con la matriz de afilado:

Para un pixel central con vecinos completos y componente rojo:

- Se multiplica cada R vecino por su factor correspondiente (algunos negativos, algunos positivos).
- Se suman los 9 productos (incluyendo los negativos).
- Se divide por la suma de factores utilizados (para afilado: $0 + (-1) + 0 + (-1) + 5 + (-1) + 0 + (-1) + 0 = 1$).

Nota sobre factores negativos: Algunas matrices (como afilado, detección de bordes y relieve) utilizan valores negativos que restan información de los vecinos. Esto permite crear efectos especiales como resaltar bordes o crear texturas. Para matrices de desenfoque, todos los valores son positivos, lo que promedia los pixeles vecinos.

Experimentación: Una vez que implemente la función correctamente con la matriz de afilado, puede cambiar fácilmente la variable `convolucion` en el código para probar las otras matrices y observar cómo cambia el efecto. Por ejemplo:

```
# Para desenfoque:
convolucion = [[1, 2, 1], [2, 4, 2], [1, 2, 1]]

# Para detección de bordes:
convolucion = [[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]
```

Nota importante: La operación debe hacerse sobre la imagen original. No acumule transformaciones de pixeles ya procesados. Trabaje siempre con los valores originales de todos los vecinos.

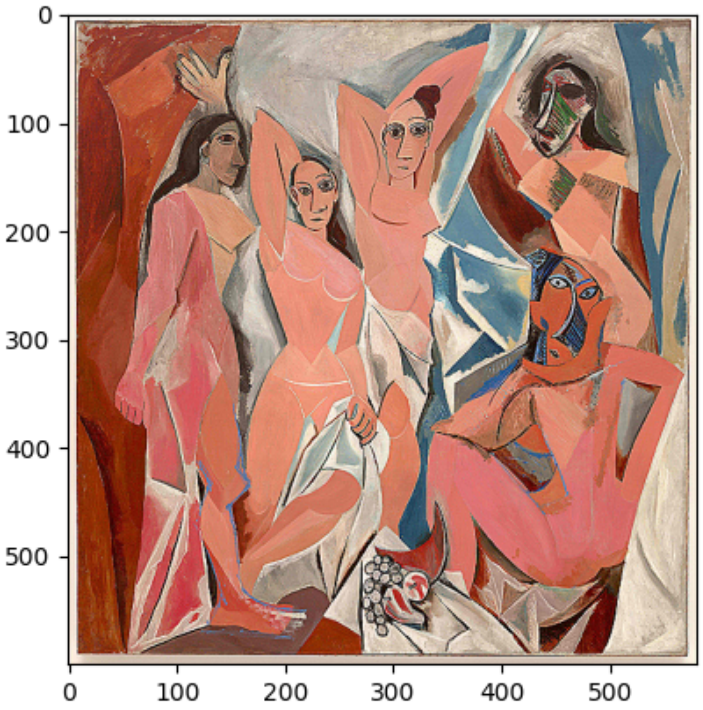

Figura 7: Ilustración del proceso de convolución y resultado aplicado a una imagen.

Entrega

Entregue el archivo `visor_imagenes.py` a través de Brightspace en el laboratorio del Nivel 4 designado como “N4-L1: Visor de imágenes con matrices”.

Puede usar el módulo `consola_visor_imagenes.py` proporcionado para probar sus funciones antes de la entrega. Su archivo debe funcionar correctamente desde la consola. Asegúrese de tener los archivos de imagen PNG en el mismo directorio para probar su implementación.

Pruebas sugeridas:

1. Cargue una imagen y visualízela (imagen original).
2. Aplique cada transformación individualmente y compare con la imagen original.
3. Aplique transformaciones en secuencia (por ejemplo, primero binarizar, luego negativo).
4. Pruebe la binarización con diferentes umbrales ($0{,}3$, $0{,}5$, $0{,}7$).
5. Experimente con las diferentes matrices de convolución cambiando el código: pruebe afilado, desenfoque, detección de bordes y relieve. Observe cómo cada una produce un efecto visual completamente diferente.
6. Para verificar que sus transformaciones funcionan correctamente, cargue nuevamente la imagen original después de cada prueba.