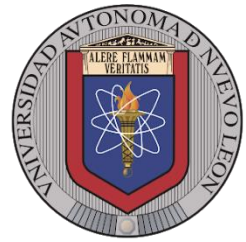


UANL

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

FIME

Nombre: Roberto Erick Aguilar Morales

Matricula: 1871004

Carrera: Ingeniero en Tecnologías del Software

5.– Vecindarios y Bordes

Materia: VISION COMPUTACIONAL LABORATORIO

Docente: RAYMUNDO SAID ZAMORA PEQUEÑO

Hora: N1-N2 **Días: Miércoles**

Fecha: 24/11/24

Objetivo

Encontrar los vecindarios en una imagen e identificar los pixeles borde.

Marco teórico

Procesamiento de Imágenes

El procesamiento de imágenes es una disciplina que se ocupa de la manipulación y análisis de imágenes digitales a través de computadoras. Se emplea en diversas áreas, desde la medicina y la astronomía, hasta la inteligencia artificial y el control de calidad industrial. Dentro de esta área, el propósito principal es extraer información útil de las imágenes o transformarlas en representaciones más adecuadas para su análisis.

El primer paso en el procesamiento de imágenes digitales suele ser la conversión de la imagen a escala de grises, lo que simplifica el análisis y permite un tratamiento más eficiente de los píxeles. Las imágenes en escala de grises son representaciones bidimensionales donde cada píxel tiene un valor de intensidad que oscila entre 0 (negro) y 255 (blanco).

Segmentación de Imágenes

La segmentación es un proceso fundamental en el análisis de imágenes, que permite dividir la imagen en regiones que tienen características similares. En este contexto, el algoritmo implementado en el código utiliza un enfoque basado en el valor de los píxeles para formar grupos llamados vecindarios.

Un vecindario de píxeles es un conjunto de píxeles que tienen características similares y están conectados entre sí. El código utiliza un rango de intensidad para definir qué píxeles pertenecen a un mismo vecindario. Los píxeles cuyo valor de intensidad está dentro de un rango específico respecto a un píxel inicial (llamado *pivote*) son agrupados en el mismo vecindario. Este tipo de segmentación es útil cuando se busca identificar regiones homogéneas en una imagen, como áreas de color similar, estructuras o patrones dentro de una imagen.

Búsqueda de Vecindarios

El código implementa un algoritmo para detectar vecindarios que se basa en la *búsqueda en anchura (BFS, por sus siglas en inglés)*. Este algoritmo es una estrategia de recorrido de grafos que expande de manera sistemática todos los píxeles conectados dentro de un vecindario. La búsqueda en anchura permite explorar cada píxel de manera iterativa, asegurando que se marquen como visitados todos los píxeles que forman parte del vecindario.

El algoritmo de búsqueda en anchura funciona con una cola (estructura de datos) que guarda los píxeles aún no procesados. Comienza con un píxel inicial y explora todos los píxeles vecinos, agregándolos a la cola si cumplen con la condición del rango de intensidad. El proceso continua hasta que no haya más píxeles para explorar.

Este tipo de algoritmo es muy adecuado cuando se trabaja con imágenes, ya que permite detectar áreas conectadas de manera eficiente, sin necesidad de recorrer la imagen de forma completamente secuencial.

La coloración de los vecindarios permite resaltar las áreas de la imagen que han sido identificadas como homogéneas en cuanto a sus valores de intensidad. Este paso es útil para aplicaciones donde es importante analizar visualmente la segmentación, por ejemplo, en el procesamiento de imágenes biomédicas o de satélite, donde se desea estudiar regiones específicas de la imagen.

Introducción

Este programa está diseñado para procesar imágenes en escala de grises y detectar agrupaciones de píxeles que comparten valores similares de intensidad, dentro de un rango especificado. Estos vecindarios son luego destacados visualmente mediante diferentes colores, facilitando su análisis y visualización. Además, el programa guarda tanto la imagen con los vecindarios resaltados como las matrices de píxeles originales y modificadas en archivos CSV, lo que permite un análisis detallado y reutilización de los datos procesados.

Este código utiliza una función que busca "vecindarios" de píxeles dentro de una imagen que tengan valores de intensidad similares (en escala de grises). Utiliza un algoritmo de búsqueda en amplitud (BFS) para explorar los píxeles vecinos que cumplen con el criterio de estar dentro de un rango de ±50 unidades de intensidad (por defecto). Se marcan los píxeles ya visitados para evitar recorrerlos múltiples veces. Para cada vecindario encontrado, se almacena la lista de píxeles que pertenecen a él y los píxeles que forman el borde del vecindario (es decir, aquellos que están fuera del rango de valores).

```
# Función para detectar vecindarios en un rango de +50 y -50
def detectar_vecindarios(image, rango=50):
    # Obtener las dimensiones de la imagen
    rows, cols = image.shape
    visitado = np.zeros((rows, cols), dtype=bool)  # Para marcar píxeles ya
revisados
    vecindarios = []  # Lista para almacenar los vecindarios encontrados
    bordes_vecindarios = []  # Lista para almacenar los bordes de
vecindarios
```

```
    def obtener_vecindario(r, c, pivote_valor, rango):
        vecindario = [(r, c)]
        borde = set()
        cola = [(r, c)]
        visitado[r, c] = True

        # Convertir el pivote a int para evitar desbordamiento
        pivote_valor = int(pivote_valor)

        while cola:
            x, y = cola.pop(0)
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < rows and 0 <= ny < cols and not
visitado[nx, ny]:
                        # Convertir el valor actual del píxel a int antes de
la comparación
                        if pivote_valor - rango <= int(image[nx, ny]) <=
pivote_valor + rango:
                            visitado[nx, ny] = True
                            vecindario.append((nx, ny))
                            cola.append((nx, ny))
                        else:
                            borde.add((nx, ny))
        return vecindario, borde

    # Recorrer la imagen
    for r in range(rows):
        for c in range(cols):
            if not visitado[r, c]:
                pivote_valor = image[r, c]
                vecindario, borde = obtener_vecindario(r, c, pivote_valor,
rango)
                vecindarios.append(vecindario)
                bordes_vecindarios.append(borde)

    return vecindarios, bordes_vecindarios
```

El código carga la imagen desde un archivo, si la imagen se carga correctamente, se convierte a escala de grises. Se llama a la función para analizar la imagen en escala de grises y se muestran los resultados: el número de vecindarios encontrados. Crea una nueva imagen en blanco (output_image) donde cada vecindario se colorea con un valor distinto para visualizarlo fácilmente. El color se

asigna de forma que cada vecindario tenga un tono diferente, calculado en función del índice del vecindario.

```
# Cargar imagen en escala de grises
imagen_a_color = cv2.imread('imagen recortada.png')
if imagen_a_color is None:
    print("Error: No se pudo cargar la imagen. Asegúrate de que el archivo 'Imagen a Color.png' está en el directorio correcto.")
else:
    imagen = cv2.cvtColor(imagen_a_color, cv2.COLOR_BGR2GRAY)

    # Detectar vecindarios y bordes
    vecindarios, bordes_vecindarios = detectar_vecindarios(imagen)

    # Mostrar cuántos vecindarios se encontraron
    print(f"Número de vecindarios encontrados: {len(vecindarios)}")

    # Para visualizar los vecindarios en la imagen
    output_image = np.zeros_like(imagen)
    for i, vecindario in enumerate(vecindarios):
        color_value = (255 - (i * 25)) % 256  # Asegurar que los valores estén entre 0 y 255
        for (r, c) in vecindario:
            output_image[r, c] = color_value  # Colorear cada vecindario de manera diferente

    cv2.imshow('Vecindarios', output_image)
    cv2.imwrite('Vecindarios.png', output_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Finalmente guarda la imagen original en escala de grises y la imagen con los vecindarios coloreados en un archivo CSV. Además guarda la imagen con los vecindarios en un archivo de imagen (Vecindarios.png).

```
    # Guardar la matriz de la imagen en escala de grises en un archivo CSV
    np.savetxt('imagen_gris.csv', imagen, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en 'imagen.csv'.")

    # Guardar la matriz de la imagen resultante en un archivo CSV
    np.savetxt('imagen_resultante.csv', output_image, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en 'imagen_resultante.csv'.")
```

Cálculos y Resultados

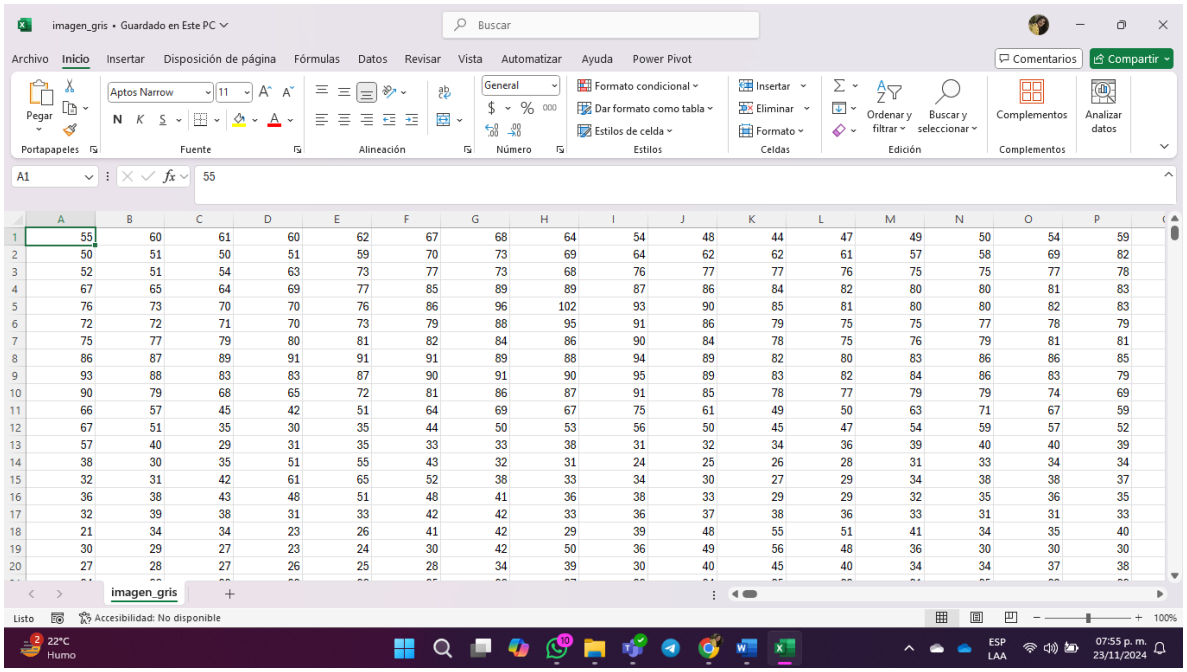
P05 Vecindarios y Bordes.py

```
import cv2
import numpy as np

# Función para detectar vecindarios en un rango de +50 y -50
def detectar_vecindarios(image, rango=50):
    # Obtener las dimensiones de la imagen
    rows, cols = image.shape
    visitado = np.zeros((rows, cols), dtype=bool)  # Para marcar píxeles ya
revisados
    vecindarios = []  # Lista para almacenar los vecindarios encontrados
    bordes_vecindarios = []  # Lista para almacenar los bordes de
vecindarios

    def obtener_vecindario(r, c, pivote_valor, rango):
        vecindario = [(r, c)]
        borde = set()
        cola = [(r, c)]
        visitado[r, c] = True

        # Convertir el pivote a int para evitar desbordamiento
        pivote_valor = int(pivote_valor)

        while cola:
            x, y = cola.pop(0)
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < rows and 0 <= ny < cols and not
visitado[nx, ny]:
                        # Convertir el valor actual del píxel a int antes de
la comparación
                        if pivote_valor - rango <= int(image[nx, ny]) <=
pivote_valor + rango:
                            visitado[nx, ny] = True
                            vecindario.append((nx, ny))
                            cola.append((nx, ny))
                        else:
                            borde.add((nx, ny))
        return vecindario, borde

    # Recorrer la imagen
    for r in range(rows):
```

```
        for c in range(cols):
            if not visitado[r, c]:
                pivote_valor = image[r, c]
                vecindario, borde = obtener_vecindario(r, c, pivote_valor,
rango)
                vecindarios.append(vecindario)
                bordes_vecindarios.append(borde)

    return vecindarios, bordes_vecindarios

# Cargar imagen en escala de grises
imagen_a_color = cv2.imread('imagen recortada.png')
if imagen_a_color is None:
    print("Error: No se pudo cargar la imagen. Asegúrate de que el archivo
'Imagen a Color.png' está en el directorio correcto.")
else:
    imagen = cv2.cvtColor(imagen_a_color, cv2.COLOR_BGR2GRAY)

    # Detectar vecindarios y bordes
    vecindarios, bordes_vecindarios = detectar_vecindarios(imagen)

    # Mostrar cuántos vecindarios se encontraron
    print(f"Número de vecindarios encontrados: {len(vecindarios)}")

    # Para visualizar los vecindarios en la imagen
    output_image = np.zeros_like(imagen)
    for i, vecindario in enumerate(vecindarios):
        color_value = (255 - (i * 25)) % 256  # Asegurar que los valores
estén entre 0 y 255
        for (r, c) in vecindario:
            output_image[r, c] = color_value  # Colorear cada vecindario de
manera diferente

    cv2.imshow('Vecindarios', output_image)
    cv2.imwrite('Vecindarios.png', output_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Guardar la matriz de la imagen en escala de grises en un archivo CSV
    np.savetxt('imagen_gris.csv', imagen, delimiter=',', fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen.csv'.")

    # Guardar la matriz de la imagen resultante en un archivo CSV
```

```
    np.savetxt('imagen_resultante.csv', output_image, delimiter=',',
fmt='%d')
    print("La matriz de la imagen en escala de grises se ha guardado en
'imagen_resultante.csv'.")
```

Imagen_gris.csv



Imagen_resultante.csv

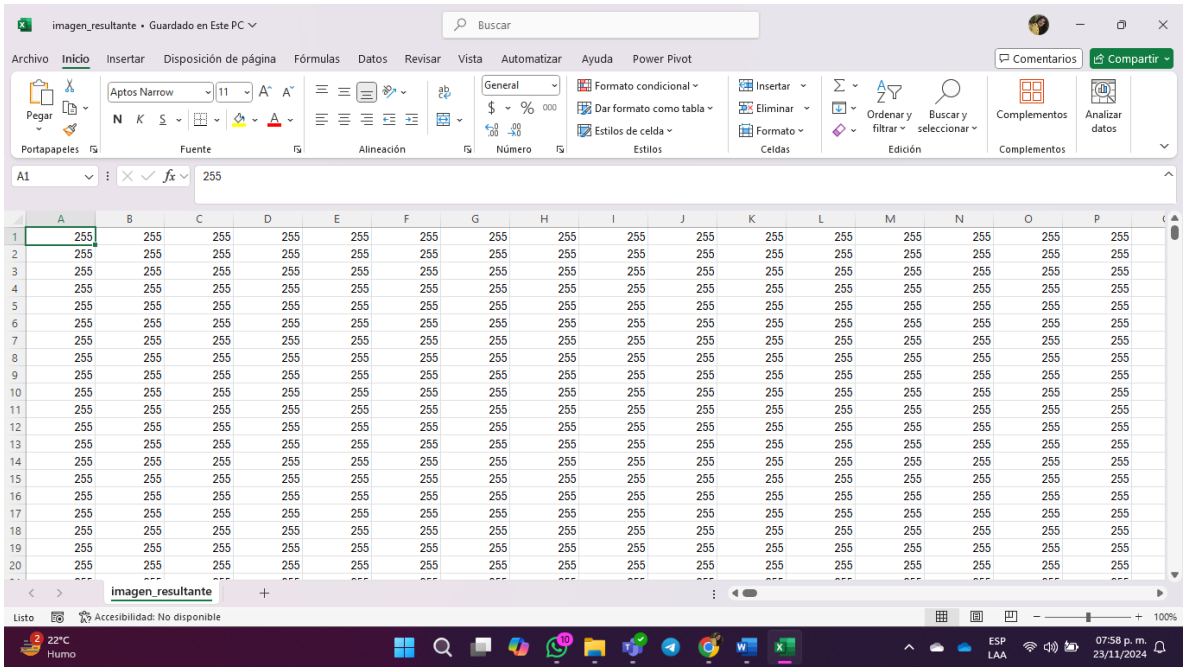


Imagen recortada.png

Vecindarios.png

Conclusiones

El programa desarrollado permite detectar y analizar vecindarios de píxeles en imágenes en escala de grises, utilizando un rango específico de intensidad como criterio. Emplea un algoritmo de búsqueda en anchura para identificar áreas conectadas de píxeles con valores similares, generando una representación visual donde cada vecindario se distingue mediante un color único. Además, guarda tanto la imagen original como la versión con vecindarios coloreados en formato CSV, facilitando su análisis posterior.

Este método resulta valioso para tareas de segmentación de imágenes al identificar regiones homogéneas, con aplicaciones potenciales en procesamiento de imágenes, análisis de texturas y visión computacional. No obstante, dado el enfoque básico del algoritmo, su eficiencia podría optimizarse en el caso de imágenes de gran tamaño, ya que el proceso puede ser computacionalmente demandante.

Bibliografía

OpenCV modules. OpenCV. (n.d.). https://docs.opencv.org/4.x/index.html

NumPy Org. (2024, June 17). NumPy. https://numpy.org/

Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing, Global Edition*. Pearson Higher Ed.