Instituto Politécnico Nacional
Escuela Superior de Cómputo

Práctica 10

Operaciones morfológicas binarias

Procesamiento Digital de Imágenes
Grupo 4BV1

Alumno: Padilla García Andrea Miranda

Objetivo: Realizar las operaciones morfológicas sobre imágenes binarias vistas en clase e implentar el algoritmo de esqueletizado

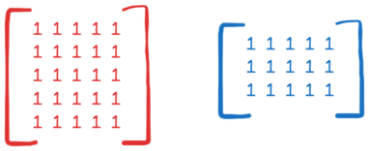
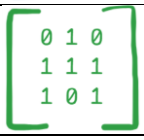
Introducción

Las operaciones morfológicas son un conjunto de técnicas utilizadas en el procesamiento de imágenes para analizar y modificar la forma y estructura de los objetos dentro de una imagen. Estas operaciones se basan en la teoría matemática de la morfología, que estudia las propiedades de las formas y los patrones.

Las operaciones morfológicas se aplican generalmente a imágenes binarias, aunque algunas también pueden aplicarse a imágenes en escala de grises. Estas operaciones se utilizan para eliminar ruido, separar o extraer formas específicas y realizar diversas tareas en el análisis de imágenes. En esta práctica, se implementarán las siguientes operaciones:

- Erosión:
- Dilatación
- Apertura
- Cierre

Para implementar cualquiera de las operaciones, requerimos de un elemento estructurante. Un elemento estructurante es una matriz que se utiliza para definir la *vecindad* o el patrón de píxeles que se procesan conjuntamente. La forma y el tamaño del elemento estructurante determinan cómo interactúa la operación morfológica con los píxeles de la imagen. Hay diferentes tipos de elementos estructurantes, entre ellos:

Cuadrado/Rectángulo	Afecta uniformemente a todos los píxeles vecinos.	
Disco/Círculo	Proporciona efectos isotrópicos, es decir, cambios uniformes en todas las direcciones.	
Cruz	Resalta los cambios en las direcciones horizontal y vertical.	
Elipse	Presenta efectos anisotrópicos, es decir, mayor influencia en una dirección.	
Personalizadas	Proporciona flexibilidad para tareas de procesamiento de imágenes únicas.	

Erosión

La erosión reduce/disminuye los límites de los objetos en primer plano en una cierta cantidad en función de un elemento estructurante (núcleo).

La erosión tiene dos componentes:

- Elemento estructurante
- Imagen binaria: La erosión se aplica sobre imágenes binarias donde los píxeles son 1 (blanco) o 0 (negro).

Este proceso da como resultado lo siguiente:

- Reducción de objetos en primer plano: Se eliminan los pequeños detalles del primer plano.
- Límites de los objetos: Los bordes de los objetos se erosionan (retraen) hacia adentro según el tamaño del elemento estructurante.

Dilatación

La dilatación es otra operación morfológica fundamental que complementa la erosión. Mientras que la erosión reduce el tamaño de los objetos en una imagen binaria, la dilatación los expande. La idea básica de la dilatación es aumentar o ampliar los límites de los objetos en primer plano a partir de un elemento estructurante.

La dilatación tiene dos componentes:

- Elemento estructurante
- Imagen binaria

Este proceso da como resultado lo siguiente:

- Los objetos en primer plano se expanden: Los objetos en primer plano (regiones blancas) crecen hacia afuera.
- Rellenar huecos: Se pueden rellenar pequeños huecos o agujeros en los objetos del primer plano.

Apertura

La apertura es una operación morfológica que combina la erosión seguida de la dilatación. Se utiliza principalmente para eliminar objetos pequeños o ruido de una imagen, preservando la forma y el tamaño de los objetos más grandes.

Los componentes de la apertura son:

1. Erosión: El primer paso en la apertura es la erosión, que elimina objetos pequeños o detalles del primer plano (regiones blancas) reduciendo los límites de los objetos.
2. Dilatación: Tras la erosión, se aplica la dilatación, que restaura el tamaño de los objetos restantes. Sin embargo, dado que los objetos más pequeños fueron completamente erosionados en el primer paso, no se restauran, eliminándolos efectivamente de la imagen.

El principal efecto de la apertura es que suaviza el contorno de los objetos, elimina istmos estrechos y disimula pequeñas protuberancias. Resulta especialmente útil para eliminar ruido u objetos pequeños que no están relacionados con los objetos principales de la imagen.

Cierre

El cierre combina dilatación seguida de erosión. Esencialmente, es la operación inversa a la de apertura y se utiliza para cerrar pequeños agujeros, huecos o espacios dentro de los objetos en primer plano, preservando al mismo tiempo su forma general.

Los componentes del cierre son:

1. Dilatación: El primer paso para cerrar la imagen es la dilatación, que expande los límites de los objetos en primer plano (regiones blancas) en la imagen binaria. Este paso ayuda a rellenar pequeños agujeros o huecos dentro de los objetos.
2. Erosión: Tras la dilatación, se aplica la erosión. Este paso reduce los límites, restaurando el tamaño original de los objetos. Sin embargo, cualquier pequeño agujero o hueco que se haya rellenado durante la dilatación permanecerá relleno, *cerrándolos* efectivamente.

El efecto clave del cierre es que suaviza los contornos de los objetos, fusiona fisuras estrechas y abismos largos y delgados, elimina pequeños agujeros y rellena huecos en los contornos de los objetos.

Esqueletizado

El objetivo de los algoritmos de adelgazamiento es tomar una imagen binaria y dibujar un *esqueleto* de 1 píxel de ancho de esa imagen, conservando la forma y la estructura de la imagen completa.

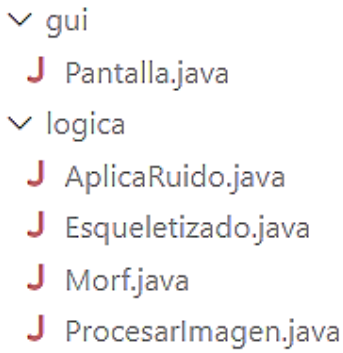
El algoritmo de adelgazamiento de Zhang-Suen es probablemente el más utilizado. Desarrollado en 1984, este algoritmo es de "*dos pasadas"*, lo que significa que en cada iteración realiza dos conjuntos de comprobaciones para eliminar píxeles de la imagen. El primer conjunto de comprobaciones elimina píxeles de la esquina sureste (inferior derecha) y el segundo, de la esquina noroeste (superior izquierda).

En pocas palabras, el algoritmo consiste en:

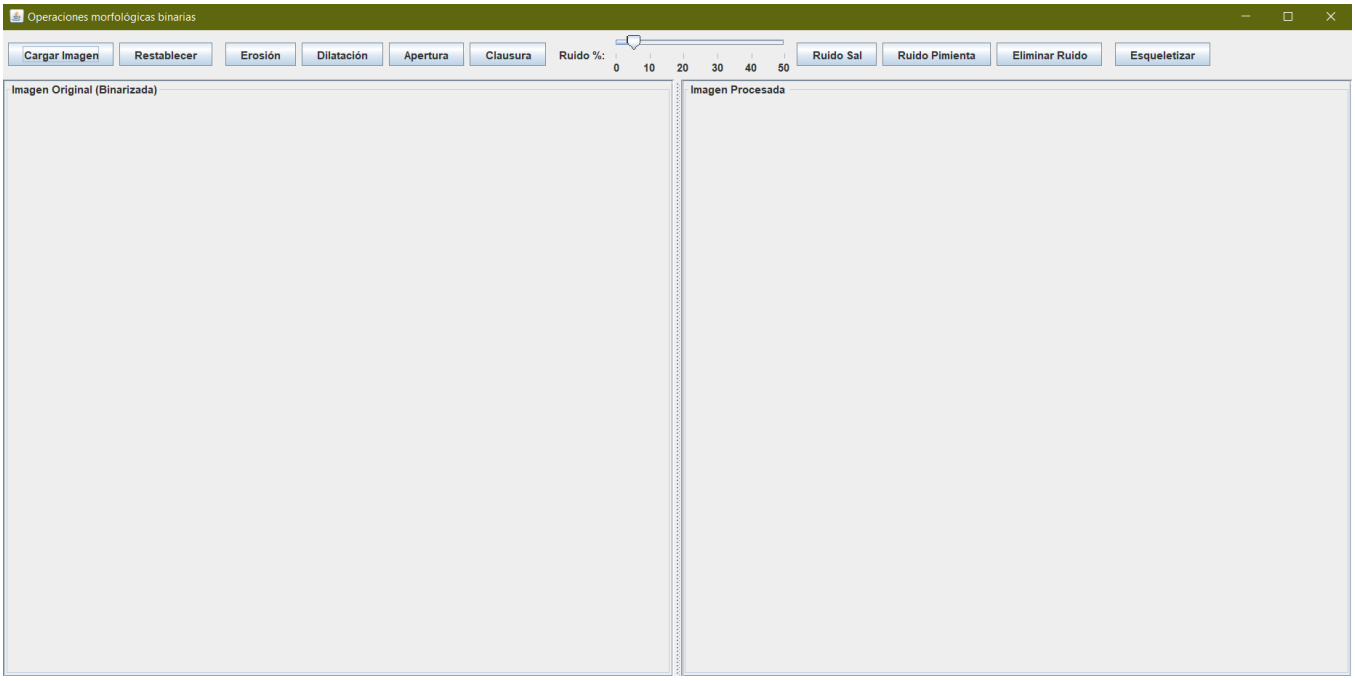
- Tomar la silueta binaria de la figura
- Quitar un *"capa"* externa de pixeles
- Revisar que no se rompa la forma
- Quitar otra capa en otra dirección
- Repetir y repetir hasta lograr un píxel de ancho
- La figura se afina hasta quedar en el esqueleto

Desarrollo

Para estructurar y organizar mi código, propongo la siguiente organización:



El diseño/organización de la pantalla (gui) fue estructurado de la siguiente manera:



Dentro de la clase ProcesarImagen, se encuentra lo siguiente:

```java
/**
 * La clase contiene los métodos necesarios para que la imagen se muestre en Pantalla
 * se reescala o redimensiona para que, independientemente de las dimensiones de la imagen,
 * se muestre perfectamente en pantalla.
 * también se obtiene el canal gris de la imagen para, posteriormente, umbralizar o binarizar la imagen
 */

public class ProcesarImagen {

    // Reescalar imagen
    public static BufferedImage reescalar(BufferedImage original, int width, int height) {
        Image tmp = original.getScaledInstance(width, height, Image.SCALE_SMOOTH);
        BufferedImage reescalar = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = reescalar.createGraphics();
        g2d.drawImage(tmp, x: 0, y: 0, observer: null);
        g2d.dispose();
        return reescalar;
    }

    // Convertir RGB a YIQ para obtener gris (Y)
    public static BufferedImage grises(BufferedImage img) {
        int w = img.getWidth();
        int h = img.getHeight();
        BufferedImage gris = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_GRAY);

        for (int y = 0; y < h; y++) {
            for (int x = 0; x < w; x++) {
                int rgb = img.getRGB(x, y);
                int r = (rgb >> 16) & 0xff;
                int g = (rgb >> 8) & 0xff;
                int b = rgb & 0xff;

                // Componente Y (de YIQ)
                int Y = (int)(0.299 * r + 0.587 * g + 0.114 * b);

                int grisRGB = (Y << 16) | (Y << 8) | Y;
                gris.setRGB(x, y, grisRGB);
            }
        }
        return gris;
    }

    // Binarización con umbral
    public static BufferedImage binariza(BufferedImage img, int umbral) {
        int w = img.getWidth();
        int h = img.getHeight();
        BufferedImage bin = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_BINARY);

        for (int y = 0; y < h; y++) {
            for (int x = 0; x < w; x++) {
                int rgb = img.getRGB(x, y) & 0xff;

                int val = (rgb < umbral) ? 0 : 255;
                int binRGB = (val << 16) | (val << 8) | val;
                bin.setRGB(x, y, binRGB);
            }
        }

        return bin;
    }
}
```

Dentro de la clase Morf, se encuentran las operaciones morfológicas:

```java
/**
 * Clase para implementar las operaciones morfológicas
 * se emplea un elemento estructurante fijo: un disco de 5x5
 * Métodos realizados: Eroción, Dilatación, Apertura y Cierre
 * en todos los casos, recibe la imagen binaria y regresa el resultado según la opeeración
 */

public class Morf {

    private static final int[][] eEstruct = {   //disco
        {0,0,1,0,0},
        {0,1,1,1,0},
        {1,1,1,1,1},
        {0,1,1,1,0},
        {0,0,1,0,0}
    };

    public static BufferedImage erosion(BufferedImage img) {
        int w = img.getWidth();
        int h = img.getHeight();

        BufferedImage out = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_BINARY);

        for (int y = 1; y < h-1; y++) {
            for (int x = 1; x < w-1; x++) {
                boolean quedar = true;
                for (int ky = -1; ky <= 1; ky++) {
                    for (int kx = -1; kx <= 1; kx++) {
                        if (eEstruct[ky+1][kx+1] == 1) {
                            int val = (img.getRGB(x+kx, y+ky) & 0xff);

                            if (val == 0) { // si algún pixel es negro, se erosiona
                                quedar = false;
                                break;
                            }
                        }
                    }
                }
                out.setRGB(x, y, quedar ? 0xffffff : 0x000000);
            }
        }

        return out;
    }
```

```java
    public static BufferedImage dilatacion(BufferedImage img) {
        int w = img.getWidth();
        int h = img.getHeight();

        BufferedImage out = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_BINARY);

        for (int y = 1; y < h-1; y++) {
            for (int x = 1; x < w-1; x++) {

                boolean blanco = false;
                for (int ky = -1; ky <= 1; ky++) {
                    for (int kx = -1; kx <= 1; kx++) {
                        if (eEstruct[ky+1][kx+1] == 1) {
                            int val = (img.getRGB(x+kx, y+ky) & 0xff);
                            if (val == 255) {
                                blanco = true;
                                break;
                            }
                        }
                    }
                }
                out.setRGB(x, y, blanco ? 0xffffff : 0x000000);
            }
        }

        return out;
    }

    public static BufferedImage apertura(BufferedImage img) {
        return dilatacion(erosion(img));
    }

    public static BufferedImage cierre(BufferedImage img) {
        return erosion(dilatacion(img));
    }
}
```

En la clase de AplicaRuido:

```java
/**
 * Clase encargada de aplicar ruido de sal o pimienta a la imagen
 * recibe ña imagen binarizada y el porcentaje tomado desde el slider de la pantalla
 */


public class AplicaRuido {

    private static Random rnd = new Random();

    // ruido sal: píxeles blancos
    public static BufferedImage sal(BufferedImage img, int porcentaje) {
        int w = img.getWidth();
        int h = img.getHeight();
        int n = w * h * porcentaje / 100;

        for (int i = 0; i < n; i++) {
            int x = rnd.nextInt(w);
            int y = rnd.nextInt(h);
            img.setRGB(x, y, rgb: 0xffffff);
        }
        return img;
    }

    // ruido pimienta: píxeles negros
    public static BufferedImage pimienta(BufferedImage img, int porcentaje) {
        int w = img.getWidth();
        int h = img.getHeight();
        int n = w * h * porcentaje / 100;

        for (int i = 0; i < n; i++) {
            int x = rnd.nextInt(w);
            int y = rnd.nextInt(h);
            img.setRGB(x, y, rgb: 0x000000);
        }

        return img;
    }
}
```

Y, por último, en la clase para aplicar el esqueletizado:

```
/**
 * Clase que implementa el algoritno de esqueletizado de Zhang-Suen
 * resulta en un dibujo de un píxel de ancho de la silueta blanca de la imagen
 *
*/

public class Esqueletizado {

    public static BufferedImage esqueleto(BufferedImage img) {

        int w = img.getWidth();
        int h = img.getHeight();
        int[][] p = new int[h][w];

        // leer imagen
        for (int y = 0; y < h; y++)
            for (int x = 0; x < w; x++)
                p[y][x] = (img.getRGB(x,y) & 0xff) == 255 ? 1 : 0;

        boolean cambio;
        do {
            cambio = false;
            List<int[]> aMover = new ArrayList<>();

            // Paso 1
            for (int y = 1; y < h-1; y++) {
                for (int x = 1; x < w-1; x++) {
                    int P = p[y][x];
                    if (P != 1) continue;

                    int[] nb = vecinos(p, x, y);
                    int bp = transicion(nb);
                    int count = sum(nb);

                    if (count >= 2 && count <= 6 &&
                        bp == 1 &&
                        nb[0]*nb[2]*nb[4] == 0 &&
                        nb[2]*nb[4]*nb[6] == 0) {

                        aMover.add(new int[]{x,y});
                    }
                }
            }
```

```
        if (!aMover.isEmpty()) cambio = true;
        for (int[] pix : aMover) p[pix[1]][pix[0]] = 0;
        aMover.clear();

        // Paso 2
        for (int y = 1; y < h-1; y++) {
            for (int x = 1; x < w-1; x++) {
                int P = p[y][x];
                if (P != 1) continue;

                int[] nb = vecinos(p, x, y);
                int bp = transicion(nb);
                int count = sum(nb);

                if (count >= 2 && count <= 6 &&
                    bp == 1 &&
                    nb[0]*nb[2]*nb[6] == 0 &&
                    nb[0]*nb[4]*nb[6] == 0) {

                    aMover.add(new int[]{x,y});
                }
            }
        }

        for (int[] pix : aMover) p[pix[1]][pix[0]] = 0;

    } while (cambio);

    // convertir de vuelta
    BufferedImage out = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_BINARY);
    for (int y = 0; y < h; y++)
        for (int x = 0; x < w; x++)
            out.setRGB(x, y, p[y][x] == 1 ? 0xffffff : 0);

    return out;
}
```

```java
    private static int[] vecinos(int[][] p, int x, int y) {
        return new int[]{
            p[y-1][x],   // P2
            p[y-1][x+1], // P3
            p[y][x+1],   // P4
            p[y+1][x+1], // P5
            p[y+1][x],   // P6
            p[y+1][x-1], // P7
            p[y][x-1],   // P8
            p[y-1][x-1]  // P9
        };
    }

    private static int transicion(int[] nb) {
        int count = 0;
        for (int i = 0; i < 7; i++)
            if (nb[i] == 0 && nb[i+1] == 1) count++;
        if (nb[7] == 0 && nb[0] == 1) count++;
        return count;
    }

    private static int sum(int[] nb) {
        int s = 0;
        for (int v : nb) s += v;
        return s;
    }
}
```

Conclusión

Para concluir, realizar esta práctica logré implementar las operaciones morfológicas básicas en imágenes binarias: erosión, dilatación, apertura y clausura. Al implementar, probar y buscar imágenes de ejemplo, dí con un texto al que había que dilatar para lograr visualizarlo mejor, lo cual me resultó útil para solucionar problemas que probablemente lleguemos a experimentar y que se soluciona de manera sencilla con un método basado en imágenes binarias. También, experimentando con la eliminación del ruido de sal y pimienta, dependiendo de que la imagen sea mayormente negra o blanca se debe usar una u otra operación (apertura o cierre, respectivamente).

Además, la integración del algoritmo de esqueletizado (de Zhang–Suen) me permitió explorar un procesamiento más avanzado. Al reducir las figuras a su esqueleto, entendí mejor cómo se puede obtener la estructura esencial de un objeto manteniendo su forma y conectividad.

Referencias

M, T. (2025, 11 noviembre). *Morphological Operations in Image Processing*. Roboflow Blog. https://blog.roboflow.com/morphological-operations/

Morfologia. (s. f.). https://grupo.us.es/gtocoma/pid/tema5-1.pdf

Zhang-Suen Thinning Algorithm. (s. f.). https://rstudio-pubs-static.s3.amazonaws.com/302782_e337cfbc5ad24922bae96ca5977f4da8.html#:~:text=The%20Zhang-Suen%20Thinning%20algorithm,remove%20pixels%20from%20the%20image.