# Documentación del Código - Prueba 2

# Descripción General

Este documento proporciona la documentación detallada para el código de la Prueba 2, que es la solución propuesta para la entrevista en el cargo de analista de ingeniería en la empresa Intecol.

El problema abordado en esta prueba implica la diferenciación de galletas a las que se les aplicó la crema correctamente de aquellas a las que no. En ciertas circunstancias, algunas galletas reciben una cantidad adecuada de crema, mientras que otras no, debido a obstrucciones en la máquina encargada del proceso.

El algoritmo desarrollado tiene como objetivo identificar estas galletas y clasificarlas en dos categorías: aquellas que tienen la cantidad adecuada de crema y aquellas que no la tienen.

El código implementa un proceso de análisis de imágenes que permite detectar visualmente la presencia o ausencia de crema en las galletas. Utiliza algoritmos de procesamiento de imágenes para realizar esta clasificación de manera eficiente y precisa.

# Explicación Detallada del Código

A continuación se explicara de manera detallada cada una de las secciones del código implementado para la solución a  la prueba 2

**Clase Form1**

La clase Form1 representa el formulario principal de la aplicación.

**Método Form1 (Constructor):**

Este método inicializa los componentes del formulario.

**Método Cargar_Btn_Click:**

Este método maneja el evento de carga de imagen. Abre un diálogo para seleccionar una imagen BMP y la carga en una PictureBox.

**Método Procesar_Btn_Click:**

Este método maneja el evento de procesamiento de imagen. Aplica varios filtros y algoritmos de procesamiento de imagen a la imagen cargada y muestra el resultado en una PictureBox.

**Método AplicarMascaraCircular:**

Este método aplica una máscara circular a una imagen, recordándola según un radio específico.

**Método ContarPixelesNegrosSegmentacion:**

Este método cuenta la cantidad de píxeles negros en una imagen utilizando segmentación.

**Método AplicarFiltroSuavizado:**

Este método aplica un filtro de suavizado Gaussiano a una imagen.

**Método ConvertirAGrises:**

Este método convierte una imagen a escala de grises.

**Método DetectarBordesSobel:**

Este método detecta los bordes en una imagen utilizando el algoritmo de Sobel.

# Algoritmos Utilizados

## Máscara Circular:

La máscara circular se aplica recortando la imagen original según un radio específico. Esto se logra mediante la creación de una región circular y luego se recorta la imagen utilizando esta región como máscara.

## Filtro de Suavizado Gaussiano:

El filtro de suavizado Gaussiano se utiliza para reducir el ruido y las discontinuidades en la imagen. Se aplica convolucionando la imagen con un kernel Gaussiano. Esto borra los detalles finos y produce una versión más suavizada de la imagen.

## Algoritmo de Sobel:

El algoritmo de Sobel se utiliza para detectar bordes en una imagen. Calcula el gradiente de la intensidad de la imagen y destaca las regiones donde hay un cambio significativo en la intensidad, lo que indica la presencia de un borde.

## Pruebas de Rendimiento

Se realizarán pruebas de rendimiento para evaluar la eficiencia de la aplicación en diferentes escenarios. Esto incluirá medir el tiempo de ejecución para cargar, procesar y mostrar imágenes de diferentes tamaños y niveles de complejidad. Además, se medirá el consumo de memoria y la utilización de CPU durante estas operaciones.


**Figura 1: Consumo de memoria cuando el programa es ejecutado.**

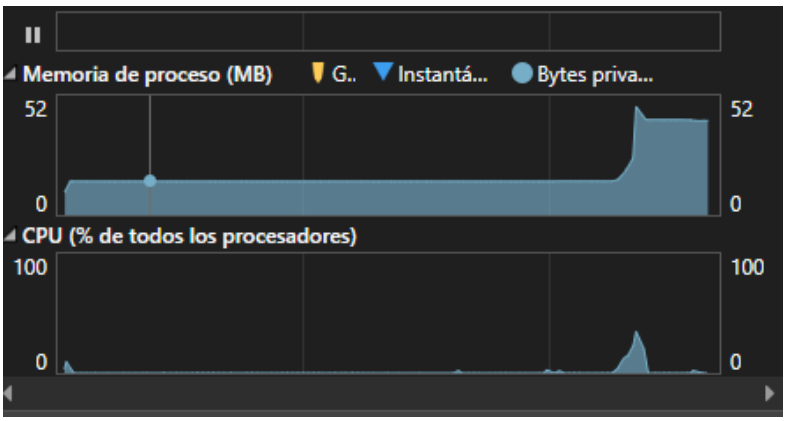



**Figura 2: Consumo de memoria cuando el programa abre el explorador de archivos.**

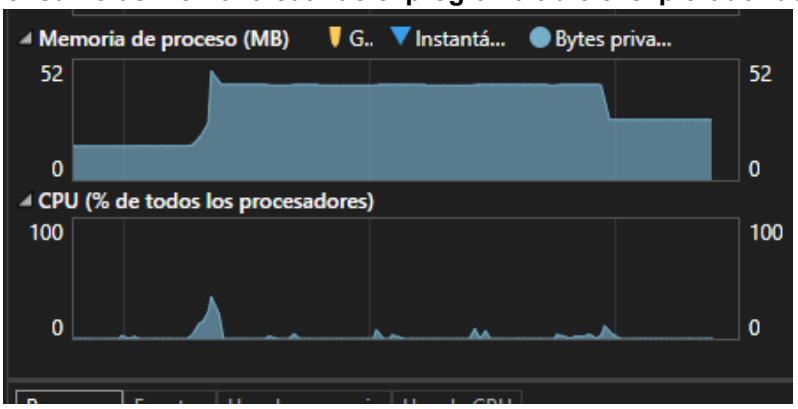



**Figura 3: Consumo de memoria cuando el programa carga la imagen.**

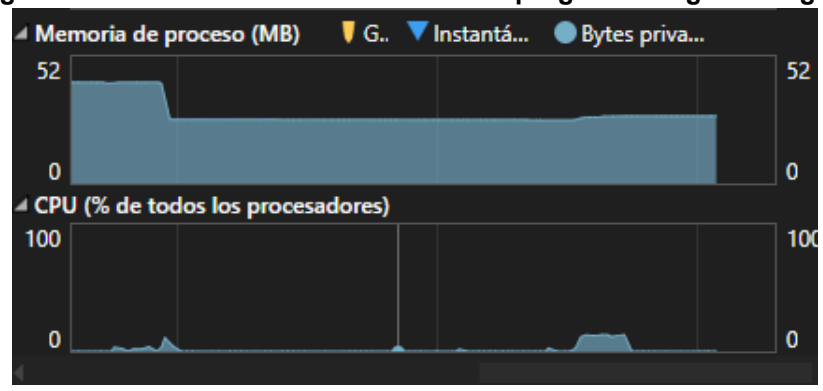
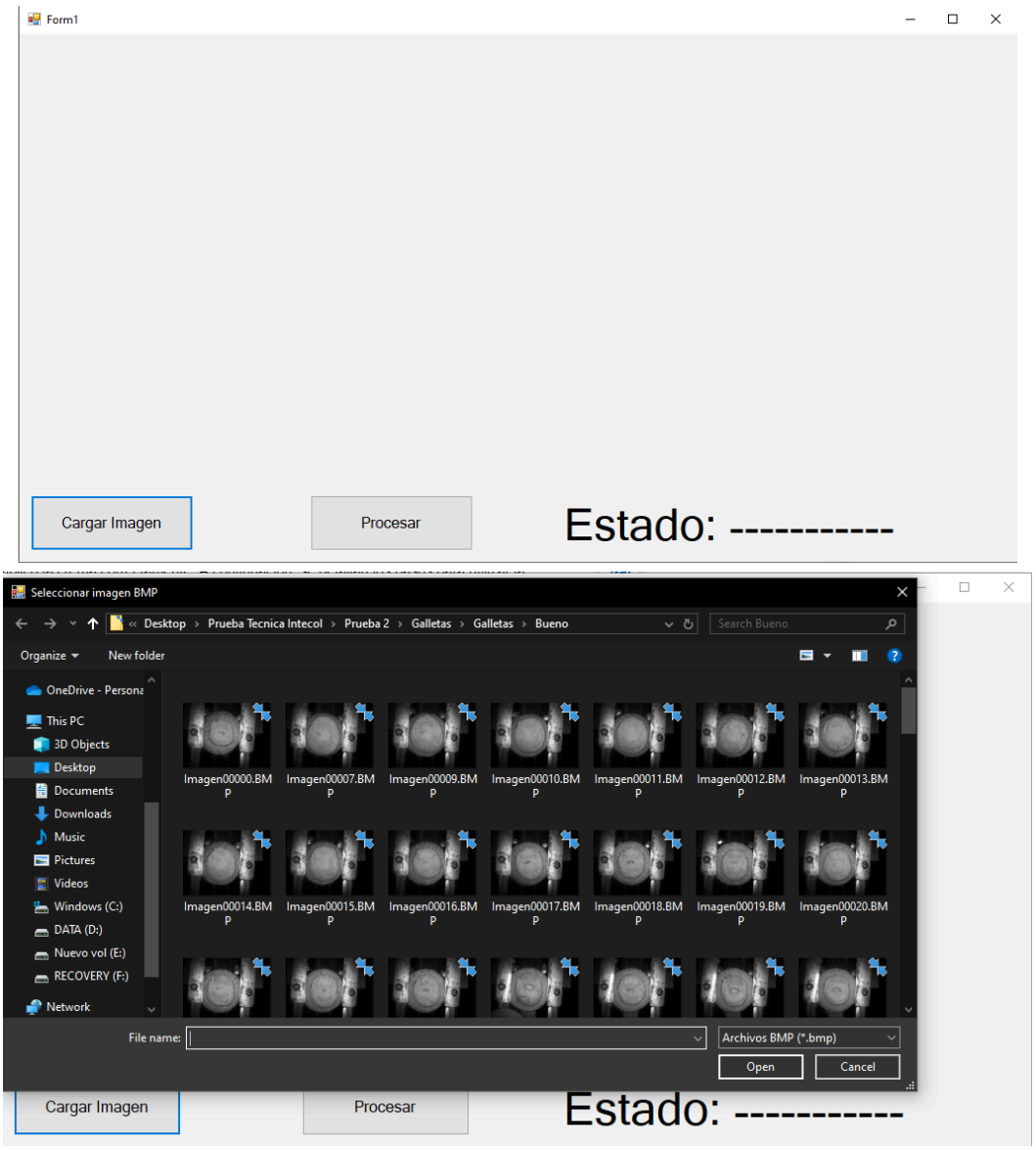



**Figura 4: Consumo de memoria cuando la imagen es procesada.**

## Uso de la Aplicación

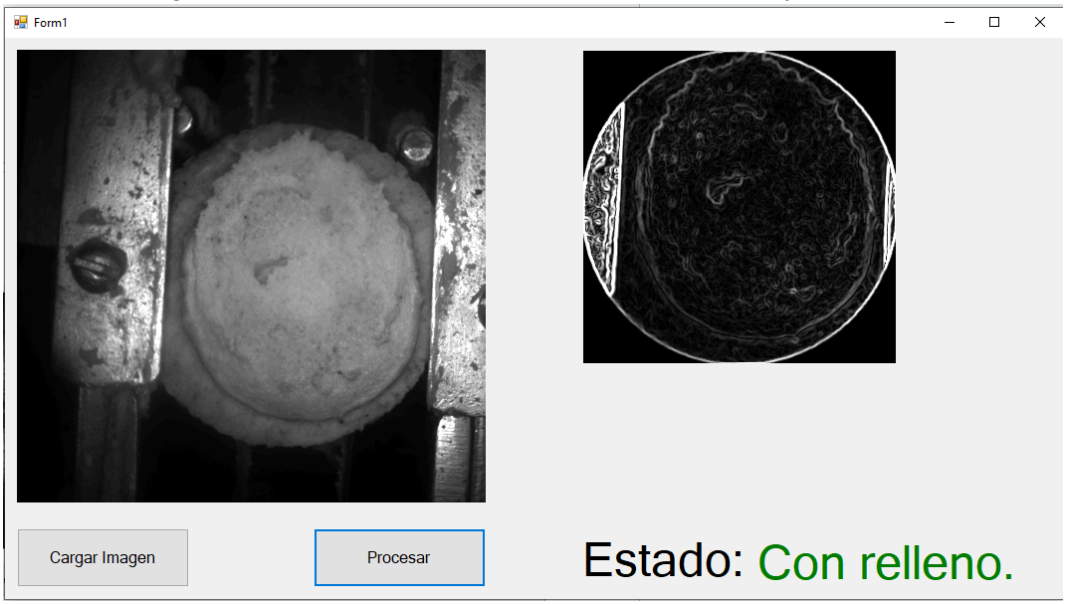
La aplicación proporciona una interfaz intuitiva para cargar imágenes de galletas y analizar si se les aplicó la crema correctamente. A continuación, se detallan los pasos para utilizar la aplicación de manera efectiva:

**Cargar una Imagen:** Haz clic en el botón "Cargar Imagen" para seleccionar una imagen de galletas desde tu computadora.




**Visualizar la Imagen:** Una vez cargada la imagen, se mostrará en la interfaz de la aplicación para su visualización.


**Análisis de la Imagen:** La aplicación realizará automáticamente un análisis de la imagen para detectar qué galletas tienen la cantidad adecuada de crema y cuáles no.

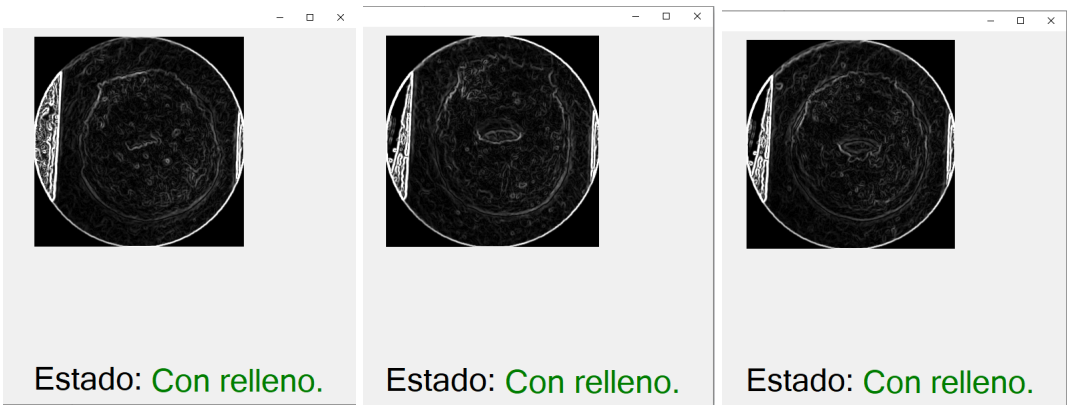



**Resultados del Análisis:** Los resultados del análisis se mostrarán en la interfaz de la aplicación. Las galletas se clasificaron como "Con Relleno" o "Sin relleno", lo que te permitirá identificar claramente las diferencias entre ellas.



**Interpretación de los resultados:** Utiliza los resultados del análisis para tomar decisiones sobre el proceso de producción de galletas. Puedes utilizar esta información para mejorar el rendimiento de la máquina encargada del proceso de aplicación de crema.

**Repetir el Proceso (Opcional):** Si lo deseas, puedes cargar otra imagen y repetir el proceso de análisis para realizar comparaciones adicionales.




C/C++

```
/*
 * Nombre del archivo: Form1.cs
 * Descripción: Este archivo contiene la implementación de la clase Form1,
que representa la interfaz gráfica principal de la aplicación de
procesamiento de imágenes.
 * Fecha de creación: 18/03/2024
 * Autor: Ing. Santiago Alejandro Zuñiga Melo
 * Descripción: Esta es mi solución a la prueba 2 para la entrevista en el
cargo de analista de ingeniería para la empresa Intecol.
 *              La aplicación permite cargar una imagen BMP, aplicar varios
filtros y algoritmos de procesamiento de imagen, como máscara circular,
suavizado y detección de bordes,
 *              y contar píxeles negros mediante segmentación. El resultado
se muestra en la interfaz gráfica, con el fin de detectar las galletas que
tienen relleno crrectamente y cuales no.
 */


using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

namespace prueba2
{
    /*
 * Clase: Form1
 * Descripción: Esta clase representa el formulario principal de la
aplicación. Contiene métodos para cargar una imagen BMP,
 *              procesarla y realizar varias operaciones de procesamiento de
imágenes, como aplicar filtros, detectar bordes
 *              y contar píxeles negros mediante segmentación.
 */
    public partial class Form1 : Form
    {
        private Image selectedImage; // Variable para almacenar la imagen
seleccionada
```

```
        /*
         * Método: Form1 (Constructor)
         * Descripción: Constructor de la clase Form1. Inicializa los
componentes del formulario.
         */
        public Form1()
        {
            InitializeComponent();
        }

        // Evento para cargar una imagen BMP
        /* Método: Cargar_Btn_Click
         * Descripción: Manejador de eventos para el botón de carga de
imagen.Abre un diálogo para seleccionar una imagen BMP y la carga en una
PictureBox.
         * Parámetros:
         *     - sender: El objeto que generó el evento.
         *     - e: Argumentos del evento.
         */
        private void Cargar_Btn_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog(); // Diálogo
para seleccionar un archivo
            openFileDialog.Filter = "Archivos BMP|*.bmp"; // Filtro para
mostrar solo archivos BMP
            openFileDialog.Title = "Seleccionar imagen BMP"; // Título del
diálogo

            if (openFileDialog.ShowDialog() == DialogResult.OK) // Si se
selecciona un archivo y se confirma
            {
                try
                {
                    // Cargar la imagen seleccionada en la PictureBox 1
                    selectedImage = Image.FromFile(openFileDialog.FileName);
                    pictureBox1.Image = selectedImage;
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error al cargar la imagen: " +
ex.Message); // Mostrar un mensaje de error si ocurre alguna excepción
                }
            }
        }

        // Evento para procesar la imagen
        /*
         * Método: Procesar_Btn_Click
         * Descripción: Manejador de eventos para el botón de procesamiento
de imagen. Aplica varios filtros y algoritmos de procesamiento de imagen
         *              a la imagen cargada y muestra el resultado en una
PictureBox.
         * Parámetros:
         *     - sender: El objeto que generó el evento.
         *     - e: Argumentos del evento.
```

```
         */
        private void Procesar_Btn_Click(object sender, EventArgs e)
        {
            // Verificar si se ha cargado una imagen
            if (selectedImage == null)
            {
                MessageBox.Show("Primero debe cargar una imagen.");
                return;
            }

            // Aplicar la máscara circular a la imagen y recortarla
            Bitmap imagenRecortada =
AplicarMascaraCircular((Bitmap)selectedImage, 4.2, 50); // Radio de 3 cm

            // Convertir la imagen recortada a escala de grises para
facilitar el procesamiento
            Bitmap grayBitmap = ConvertirAGrises(imagenRecortada);

            // Aplicar un filtro de suavizado antes de aplicar el algoritmo
de Sobel
            Bitmap smoothedBitmap = AplicarFiltroSuavizado(grayBitmap);

            // Aplicar el algoritmo de Sobel para detectar bordes
            Bitmap bordesBitmap = DetectarBordesSobel(smoothedBitmap);

            // Actualizar la imagen en la PictureBox 2 para mostrar los
bordes detectados
            pictureBox2.Image = bordesBitmap;

            // Mejora en el conteo de píxeles negros utilizando segmentación
            int pixelesNegros =
ContarPixelesNegrosSegmentacion(bordesBitmap);

            // Determinar si la cantidad de píxeles negros supera el umbral
            int umbral = 70000; // Se ajustó el umbral

            // Mostrar el resultado según el umbral
            if (pixelesNegros < umbral)
            {
                TextLabel.Text = ("Con relleno."); // Mostrar "Con relleno."
en el TextLabel
                TextLabel.ForeColor = Color.Green; // Color verde para el
TextLabel
            }
            else
            {
                TextLabel.Text = ("Sin relleno."); // Mostrar "Sin relleno."
en el TextLabel
                TextLabel.ForeColor = Color.Red; // Color rojo para el
TextLabel
            }
        }

        // Método para aplicar una máscara circular a una imagen
        /*
```

```
         * Método: AplicarMascaraCircular
         * Descripción: Aplica una máscara circular a una imagen,
recortándola según un radio específico.
         * Parámetros:
         *     - imagen: La imagen original a la que se aplicará la máscara.
         *     - radioCm: El radio en centímetros de la máscara circular.
         *     - desplazamiento: El desplazamiento de la máscara circular.
         * Retorno:
         *     - Bitmap: La imagen resultante después de aplicar la máscara.
         */
        private Bitmap AplicarMascaraCircular(Bitmap imagen, double radioCm,
int desplazamiento)
        {
            // Convertir el radio de cm a píxeles
            double dpiX = imagen.HorizontalResolution;
            double dpiY = imagen.VerticalResolution;
            int radioPixels = (int)(radioCm * dpiX / 2.54); // 1 cm = 2.54
pulgadas

            // Crear un Bitmap para almacenar la imagen recortada en forma
de círculo
            Bitmap imagenRecortada = new Bitmap(2 * radioPixels, 2 *
radioPixels);

            using (Graphics g = Graphics.FromImage(imagenRecortada))
            {
                // Crear un GraphicsPath para definir la forma del círculo
                GraphicsPath path = new GraphicsPath();
                path.AddEllipse(0, 0, 2 * radioPixels, 2 * radioPixels);

                // Crear una región utilizando el GraphicsPath
                Region region = new Region(path);

                // Establecer la región como el área de recorte
                g.SetClip(region, CombineMode.Replace);

                // Dibujar la porción recortada de la imagen original en el
nuevo Bitmap
                g.DrawImage(imagen, new Rectangle(0, 0, 2 * radioPixels, 2 *
radioPixels), (imagen.Width - 2 * radioPixels) / 2 - desplazamiento,
(imagen.Height - 2 * radioPixels) / 2, 2 * radioPixels, 2 * radioPixels,
GraphicsUnit.Pixel);
            }

            return imagenRecortada;
        }

        // Método para contar los píxeles negros utilizando segmentación
        /*
         * Método: ContarPixelesNegrosSegmentacion
         * Descripción: Cuenta la cantidad de píxeles negros en una imagen
utilizando segmentación.
         * Parámetros:
         *     - bitmap: La imagen en la que se contarán los píxeles negros.
         * Retorno:
```

```
         *     - int: La cantidad de píxeles negros en la imagen.
         */
        private int ContarPixelesNegrosSegmentacion(Bitmap bitmap)
        {
            int contador = 0;

            // Crear una matriz para mantener el estado de cada píxel
visitado
            bool[,] visitado = new bool[bitmap.Width, bitmap.Height];

            for (int x = 0; x < bitmap.Width; x++)
            {
                for (int y = 0; y < bitmap.Height; y++)
                {
                    // Si el píxel no ha sido visitado y es negro
                    if (!visitado[x, y] && bitmap.GetPixel(x,
y).GetBrightness() < 0.1f)
                    {
                        // Realizar un recorrido BFS desde el píxel negro
                        Queue<Point> queue = new Queue<Point>();
                        queue.Enqueue(new Point(x, y));
                        visitado[x, y] = true;

                        while (queue.Count > 0)
                        {
                            Point punto = queue.Dequeue();
                            contador++;

                            // Marcar los vecinos no visitados que sean
negros
                            for (int dx = -1; dx <= 1; dx++)
                            {
                                for (int dy = -1; dy <= 1; dy++)
                                {
                                    int nx = punto.X + dx;
                                    int ny = punto.Y + dy;

                                    if (nx >= 0 && nx < bitmap.Width && ny
>= 0 && ny < bitmap.Height &&
                                        !visitado[nx, ny] &&
bitmap.GetPixel(nx, ny).GetBrightness() < 0.1f)
                                    {
                                        queue.Enqueue(new Point(nx, ny));
                                        visitado[nx, ny] = true;
                                    }
                                }
                            }
                        }
                    }
                }
            }

            return contador;
        }
```

```
        // Método para aplicar un filtro de suavizado Gaussiano
        /*
         * Método: AplicarFiltroSuavizado
         * Descripción: Aplica un filtro de suavizado Gaussiano a una
imagen.
         * Parámetros:
         *     - bitmap: La imagen a la que se aplicará el filtro de
suavizado.
         * Retorno:
         *     - Bitmap: La imagen resultante después de aplicar el filtro
de suavizado.
         */
        private Bitmap AplicarFiltroSuavizado(Bitmap bitmap)
        {
            // Crear un filtro de suavizado Gaussiano
            double[,] filtroSuavizado = {
                { 1.0 / 16, 1.0 / 8, 1.0 / 16 },
                { 1.0 / 8, 1.0 / 4, 1.0 / 8 },
                { 1.0 / 16, 1.0 / 8, 1.0 / 16 }
            };

            Bitmap suavizadoBitmap = new Bitmap(bitmap.Width,
bitmap.Height);

            for (int x = 1; x < bitmap.Width - 1; x++)
            {
                for (int y = 1; y < bitmap.Height - 1; y++)
                {
                    double acumulador = 0.0;

                    // Aplicar el filtro a la vecindad de cada píxel
                    for (int i = -1; i <= 1; i++)
                    {
                        for (int j = -1; j <= 1; j++)
                        {
                            Color pixel = bitmap.GetPixel(x + i, y + j);
                            double intensidad = pixel.R;

                            acumulador += intensidad * filtroSuavizado[i +
1, j + 1];
                        }
                    }

                    // Establecer el nuevo color suavizado
                    int nuevoValor = (int)acumulador;
                    Color nuevoColor = Color.FromArgb(nuevoValor,
nuevoValor, nuevoValor);
                    suavizadoBitmap.SetPixel(x, y, nuevoColor);
                }
            }

            return suavizadoBitmap;
        }

        // Método para convertir una imagen a escala de grises
```

```
        /*
         * Método: ConvertirAGrises
         * Descripción: Convierte una imagen a escala de grises.
         * Parámetros:
         *     - bitmap: La imagen a convertir a escala de grises.
         * Retorno:
         *     - Bitmap: La imagen resultante en escala de grises.
         */
        private Bitmap ConvertirAGrises(Bitmap bitmap)
        {
            Bitmap grayBitmap = new Bitmap(bitmap.Width, bitmap.Height);

            for (int x = 0; x < bitmap.Width; x++)
            {
                for (int y = 0; y < bitmap.Height; y++)
                {
                    Color color = bitmap.GetPixel(x, y);
                    int promedio = (color.R + color.G + color.B) / 3; //
Calcular el promedio de los componentes RGB
                    Color nuevoColor = Color.FromArgb(promedio, promedio,
promedio); // Crear un nuevo color en escala de grises
                    grayBitmap.SetPixel(x, y, nuevoColor); // Establecer el
nuevo color en el bitmap en escala de grises
                }
            }

            return grayBitmap;
        }

        // Método para detectar bordes utilizando el algoritmo de Sobel
        /*
         * Método: DetectarBordesSobel
         * Descripción: Detecta los bordes en una imagen utilizando el
algoritmo de Sobel.
         * Parámetros:
         *     - bitmap: La imagen en la que se detectarán los bordes.
         * Retorno:
         *     - Bitmap: La imagen resultante con los bordes detectados.
         */
        private Bitmap DetectarBordesSobel(Bitmap bitmap)
        {
            Bitmap bordesBitmap = new Bitmap(bitmap.Width, bitmap.Height);

            int[,] filtroSobelX = { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 }
};
            int[,] filtroSobelY = { { -1, -2, -1 }, { 0, 0, 0 }, { 1, 2, 1 }
};

            for (int x = 1; x < bitmap.Width - 1; x++)
            {
                for (int y = 1; y < bitmap.Height - 1; y++)
                {
                    int gx = 0, gy = 0;

                    // Aplicar el filtro Sobel en la vecindad de cada píxel
```

```
                    for (int i = -1; i <= 1; i++)
                    {
                        for (int j = -1; j <= 1; j++)
                        {
                            Color pixel = bitmap.GetPixel(x + i, y + j);
                            int intensidad = pixel.R;

                            gx += intensidad * filtroSobelX[i + 1, j + 1];
                            gy += intensidad * filtroSobelY[i + 1, j + 1];
                        }
                    }

                    // Calcular el gradiente de magnitud
                    int magnitudGradiente = (int)Math.Sqrt(gx * gx + gy *
gy);

                    // Asegurarse de que la magnitud del gradiente esté en
el rango de 0 a 255
                    magnitudGradiente = Math.Max(0,
Math.Min(magnitudGradiente, 255));

                    // Establecer el nuevo color basado en la magnitud del
gradiente
                    Color nuevoColor = Color.FromArgb(magnitudGradiente,
magnitudGradiente, magnitudGradiente);
                    bordesBitmap.SetPixel(x, y, nuevoColor);
                }
            }

            return bordesBitmap;
        }
    }
}
```