***UNIVERSIDAD NACIONAL DE INGENIERÍA***

***FACULTAD DE CIENCIAS***

***SISTEMAS CONCURRENTES Y DISTRIBUIDOS***


**EXAMEN PARCIAL**

**ALUMNOS:**

Palacios Asenjo, Jorge
Ruegg Yupa. Josias
León Rios, Naro
Ponce Pinedo, Victor
Espinoza Mansilla, Carlos

**PROFESOR:**

Yuri Nuñez Medrano

1. Introducción

En este documento se describe como se crean un servidor y sus esclavos para que puedan realizar el proceso de tratamiento de imágenes concurrentes y distribuidos, con operaciones de matrices que son imágenes, con otras matrices que serían los kernels y estos dan resultados que serían nuevas imágenes con los siguientes procedimientos:

- Un servidor que recepciones varios archivos con extensión jpg.
- Cada archivo se transforme a escala de grises.
- Cada archivo en escala de grises multiplicarlo por un kernel y generar un nuevo. archivo.
- Posteriormente pasar cada archivo que se procesó con el kernel a un archivo jpg.

La conversión de imágenes a escalas se grises se hace mediante convolución de imágenes.

2. Objetivos
   -

3. Marco Teórico
   3.1. Escala de Grises

En fotografía digital, las imágenes generadas por computadora, una imagen en escala de grises o en escala de grises es aquella en la que el valor de cada píxel es una muestra única que representa solo una cantidad de luz, es decir, solo transporta información de intensidad. Las imágenes en escala de grises, una especie de monocromo en blanco y negro o gris, se componen exclusivamente de tonos de gris. El contraste varía de negro a la intensidad más débil a blanco en el más fuerte.

Las imágenes en escala de grises son distintas de las imágenes en blanco y negro bitonales de un bit que, en el contexto de las imágenes por computadora, son imágenes con solo dos colores: blanco y negro (también llamadas imágenes de dos niveles o binarias). Las imágenes en escala de grises tienen muchos tonos de gris en el medio.

La intensidad de un píxel se expresa dentro de un rango dado entre un mínimo y un máximo, inclusive. Este rango se representa de manera abstracta como un rango de 0 (o 0%) (ausencia total, negro) y 1 (o 100%) (presencia total, blanco), con cualquier valor fraccionario en el medio. Esta notación se usa en trabajos académicos, pero no define qué es "negro" o "blanco" en términos de colorimetría. A veces, la escala se invierte, como en la impresión, donde la intensidad numérica indica la cantidad de tinta que se emplea en los medios tonos, donde el 0% representa el blanco del papel (sin tinta) y el 100% es un negro sólido (tinta completa).

Los usos técnicos (por ejemplo, en imágenes médicas o aplicaciones de detección remota) a menudo requieren más niveles, para aprovechar al máximo la precisión del sensor (típicamente 10 o 12 bits por muestra) y para reducir los errores de redondeo en los cálculos. Dieciséis bits por muestra (65.536 niveles) a menudo es una opción conveniente para tales usos, ya que las computadoras administran palabras de 16 bits de manera eficiente. Los formatos de archivo de imagen TIFF y PNG (entre otros) admite escala de grises de 16 bits de forma nativa, aunque los navegadores y muchos programas de imágenes tienden a ignorar los 8 bits de orden inferior de cada píxel. Internamente para el cálculo y el almacenamiento de trabajo, el software de procesamiento de imágenes generalmente usa números enteros o de coma flotante de tamaño 16 o 32 bits.

Para la conversión de una imagen de color a escala de grises, no hay un solo método.

**Conversión colorimétrica a escala de grises**

Una estrategia común es utilizar los principios de fotometría o, más ampliamente, colorimetría para calcular los valores de escala de grises (en el espacio de color de escala de grises objetivo) para tener la misma luminancia (luminancia técnicamente relativa) que la imagen de color original (de acuerdo con su espacio de color ). Además de la misma luminancia (relativa), este método también asegura que ambas imágenes tendrán la misma luminancia absoluta cuando se muestren, como puede medirse por instrumentos en sus unidades SI de candelas por metro cuadrado , en cualquier área dada de la imagen, puntos blancos iguales. La luminancia en sí misma se define utilizando un modelo estándar de visión humana, por lo que preservar la luminancia en la imagen en escala de grises también conserva otras medidas de claridad perceptiva , como $L^*$ (como en el espacio de color CIE *L* ab de 1976 ) que está determinada por la luminancia lineal *Y* en sí (como en el espacio de color CIE 1931 *XYZ* ) al que nos referiremos aquí como $Y_{\text{lineal}}$ para evitar cualquier ambigüedad.

**Luma coding en sistemas de video**

Para imágenes en espacios de color como Y'UV y sus parientes, que se utilizan en sistemas de televisión y video en color estándar como PAL , SECAM y NTSC , un componente de luma no lineal ( *Y* ′ ) se calcula directamente a partir de intensidades primarias comprimidas con rayos gamma como una suma ponderada que, aunque no es una representación perfecta de la luminancia colorimétrica, puede calcularse más rápidamente sin la expansión y compresión gamma utilizada en los cálculos fotométricos/colorimétricos. En los modelos Y'UV e Y'IQ utilizados por PAL y NTSC, el rec601 luma ( *Y* ′ ) componente se calcula como

$$Y' = 0.299R' + 0.587G' + 0.114B'$$

donde usamos el primo para distinguir estos valores no lineales de los valores no lineales sRGB (discutidos anteriormente) que usan una fórmula de compresión gamma algo diferente, y de los componentes RGB lineales. El estándar ITU-R BT.709 utilizado para HDTV desarrollado por ATSC utiliza diferentes coeficientes de color, calculando el componente luma como

$$Y' = 0.2126R' + 0.7152G' + 0.0722B' .$$

Aunque estos son numéricamente los mismos coeficientes utilizados en sRGB arriba, el efecto es diferente porque aquí se están aplicando directamente a valores comprimidos con rayos gamma en lugar de a los valores linealizados. El estándar ITU-R BT.2100 para televisión HDR utiliza coeficientes aún diferentes, calculando el componente luma como

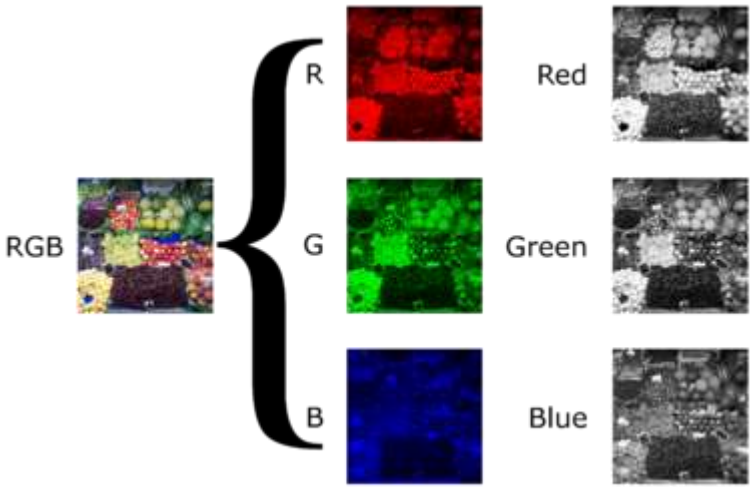
$$Y' = 0.2627R' + 0.6780G' + 0.0593B' .$$

Normalmente, estos espacios de color se transforman nuevamente en R'G'B 'no lineales antes de renderizarlos para su visualización. En la medida en que se mantenga la precisión suficiente, se pueden representar con precisión.

**Escala de grises como canales individuales de imágenes en color multicanal**

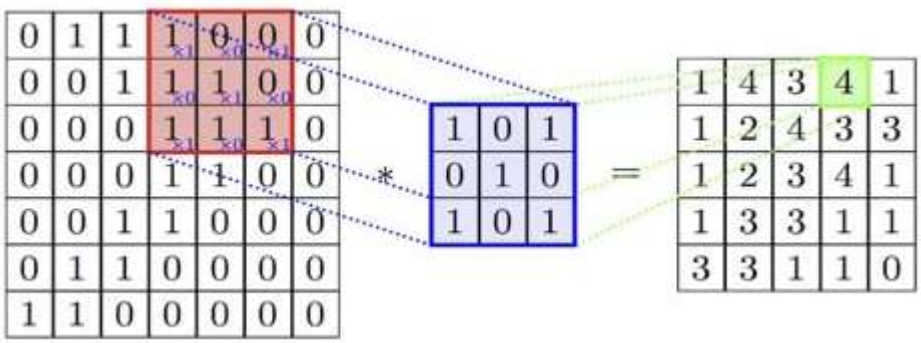
Las imágenes en color a menudo se componen de varios canales de color apilados, cada uno de los cuales representa niveles de valor del canal dado. Por ejemplo, las imágenes RGB se componen de tres canales independientes para los componentes de color primario rojo, verde y azul; Las imágenes CMYK tienen cuatro canales para placas de tinta cian, magenta, amarilla y negra, etc.

Aquí hay un ejemplo de división de canales de color de una imagen de color RGB completa. La columna de la izquierda muestra los canales de color aislados en colores naturales, mientras que a la derecha están sus equivalencias en escala de grises:

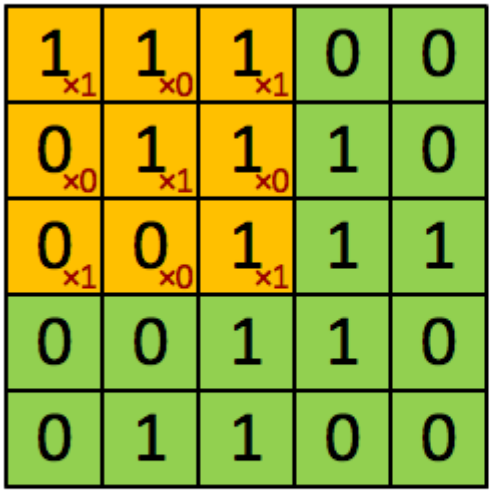


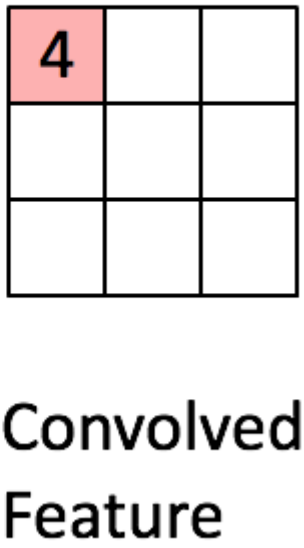

### 3.2. Convolución Imágenes

Una convolución es una operación que convierte una función en otra. Hacemos convoluciones para que podamos transformar la función original en un formulario para obtener más información. Las convoluciones se han utilizado durante mucho tiempo en el procesamiento de imágenes para desenfocar y enfocar las imágenes, y realizar otras operaciones, como mejorar bordes y relieve.




Aquí, la imagen original es la de la izquierda y la matriz de números en el medio es la matriz o filtro convolucional. Una operación de convolución es una operación de multiplicación matricial por elementos. Donde una de las matrices es la imagen, y la otra es el filtro o núcleo que convierte la imagen en otra cosa. El resultado de esto es la imagen enrevesada final.




Si la imagen es más grande que el tamaño del filtro, deslizamos el filtro a las diversas partes de la imagen y realizamos la operación de convolución. Cada vez que hacemos eso, generamos un nuevo píxel en la imagen de salida.

El número de píxeles por los que deslizamos el núcleo se conoce como el paso. El paso generalmente se mantiene como 1, pero podemos aumentarlo. Cuando se aumenta, es posible que tengamos que aumentar el tamaño de la imagen unos pocos píxeles para que quepa en el núcleo en los bordes de la imagen. Este aumento se llama relleno.

Hablaré más sobre cómo esto puede ayudarnos a obtener más información de una imagen en una sección posterior.

### 3.3. Image Kernel

Un núcleo de imagen es una matriz pequeña que se usa para aplicar efectos como los que puede encontrar en Photoshop o Gimp, como desenfoque, nitidez, contorno o estampado. También se utilizan en el aprendizaje automático para la 'extracción de características', una técnica para determinar las partes más importantes de una imagen. En este contexto, el proceso se conoce más generalmente como "convolución"

Para ver cómo funcionan, comencemos inspeccionando una imagen en blanco y negro. La matriz de la izquierda contiene números, entre 0 y 255, que corresponden al brillo de un píxel en una imagen de una cara. La imagen grande y granulada ha sido ampliada para que sea más fácil de ver; la última imagen es el tamaño "real".

Veamos cómo aplicar el siguiente núcleo de enfoque 3x3 a la imagen de una cara desde arriba.

afilar

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

A continuación, por cada bloque de píxeles de 3x3 en la imagen de la izquierda, multiplicamos cada píxel por la entrada correspondiente del núcleo y luego tomamos la suma. Esa suma se convierte en un nuevo píxel en la imagen de la derecha. Desplácese sobre un píxel en cualquiera de las imágenes para ver cómo se calcula su valor.


Una sutileza de este proceso es qué hacer a lo largo de los bordes de la imagen. Por ejemplo, la esquina superior izquierda de la imagen de entrada solo tiene tres vecinos. Una forma de solucionar esto es extender los valores de borde en uno en la imagen original mientras se mantiene nuestra nueva imagen del mismo tamaño. En esta demostración, hemos ignorado esos valores al hacerlos negros.

Aquí hay un patio de juegos donde puedes seleccionar diferentes matrices de kernel y ver cómo afectan la imagen original o construir tu propio kernel. También puede cargar su propia imagen o usar video en vivo si su navegador lo admite.

3.4. Tipos de Kernel

3.4.1.Simple box blur

Es el más simple. Este núcleo de convolución tiene un efecto promedio. Entonces terminas con un ligero desenfoque. El núcleo de convolución de imagen es:

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

Tenga en cuenta que la suma de todos los elementos de esta matriz es 1.0. Esto es importante. Si la suma no es exactamente una, la imagen resultante será más brillante u oscura.

Aquí hay un desenfoque que obtuve en una imagen:

3.4.2.Gaussian blur

El desenfoque gaussiano tiene ciertas propiedades matemáticas que lo hacen importante para la visión por computadora. Y puede aproximarlo con una convolución de imagen. El núcleo de convolución de imagen para un desenfoque gaussiano es:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| 0 | 5 | 18 | 32 | 18 | 5 | 0 |
| 0 | 18 | 64 | 100 | 64 | 18 | 0 |
| 5 | 32 | 100 | 100 | 100 | 32 | 5 |
| 0 | 18 | 64 | 100 | 64 | 18 | 0 |
| 0 | 5 | 18 | 32 | 18 | 5 | 0 |
| 0 | 0 | 0 | 5 | 0 | 0 | 0 |

Aquí hay un resultado que obtuve:

### 3.4.3.Line detection with image convolutions

Con las convoluciones de imagen, puede detectar fácilmente líneas. Aquí hay cuatro circunvoluciones para detectar horizontal, vertical y líneas a 45 grados:

| | | |
|---|---|---|
| -1 | -1 | -1 |
| 2 | 2 | 2 |
| -1 | -1 | -1 |

Horizontal lines

| | | |
|---|---|---|
| -1 | 2 | -1 |
| -1 | 2 | -1 |
| -1 | 2 | -1 |

Vertical lines

| | | |
|---|---|---|
| -1 | -1 | 2 |
| -1 | 2 | -1 |
| 2 | -1 | -1 |

45 degree lines

| | | |
|---|---|---|
| 2 | -1 | -1 |
| -1 | 2 | -1 |
| -1 | -1 | 2 |

135 degree lines

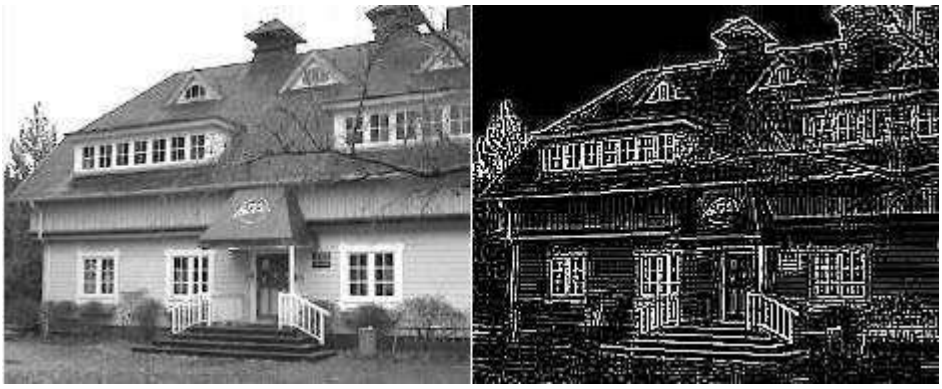
Busqué líneas horizontales en la imagen de la casa. El resultado que obtuve para esta convolución de imagen fue:

### 3.4.4.Edge detection

Los núcleos anteriores son en cierto modo detectores de borde. Lo único es que tienen componentes separados para líneas horizontales y verticales. Una forma de "combinar" los resultados es fusionar los núcleos de convolución. El nuevo núcleo de convolución de imagen se ve así:

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

Debajo del resultado que obtuve con la detección de bordes:



3.4.5.The Sobel Edge Operator

Los operadores anteriores son muy propensos al ruido. Los operadores de borde Sobel tienen un efecto de suavizado, por lo que se ven menos afectados por el ruido. De nuevo, hay un componente horizontal y un componente vertical.

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Horizontal

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Vertical

Al aplicar esta convolución de imagen, el resultado fue:

3.4.6.The laplacian operator

El laplaciano es la segunda derivada de la imagen. Es extremadamente sensible al ruido, por lo que no se usa tanto como otros operadores. A menos que, por supuesto, tenga requisitos específicos.

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

The laplacian operator

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

The laplacian operator (include diagonals)

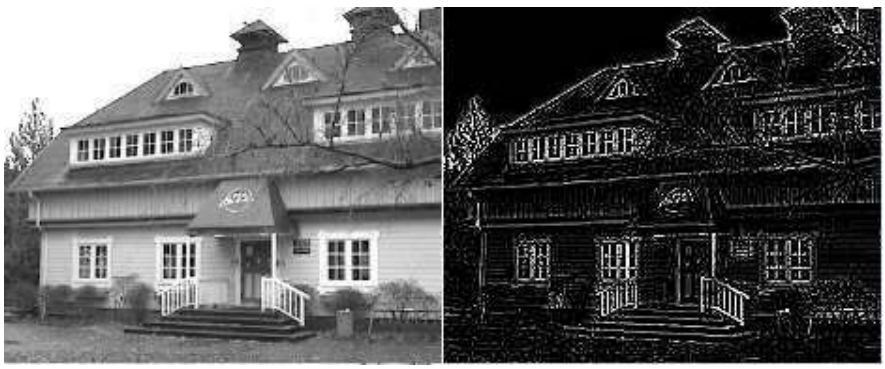
Aquí está el resultado con el núcleo de convolución sin diagonales:

3.4.7.The Laplacian of Gaussian

El laplaciano solo tiene la desventaja de ser extremadamente sensible al ruido. Entonces, alisar la imagen antes de que un laplaciano mejore los resultados que obtenemos. Esto se hace con un núcleo de convolución de imagen de 5x5.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | -1 | 0 | 0 |
| 0 | -1 | -2 | -1 | 0 |
| -1 | -2 | 16 | -2 | -1 |
| 0 | -1 | -2 | -1 | 0 |
| 0 | 0 | -1 | 0 | 0 |

El resultado al aplicar esta convolución de imagen fue:



4. Estructura del programa

4.1. LoadImage

Usamos las siguientes librerias

```java
package parcial;
import java.awt.*;
import java.awt.image.BufferedImage;

import java.io.*;

import javax.imageio.ImageIO;
import javax.swing.JFrame;

```

Definimos para la convolución

```java
public class LoadImage {

    volatile int[][][] lista_de_imagenes = null;
```

Creamos un método para obtener una matriz de la imagen

```java

    public static int[][] getMatrixOfImage(String filename) {
        BufferedImage bufferedImage = null;
        try {
            bufferedImage = ImageIO.read(new File(filename));
        } catch (IOException e) {
            System.err.println("Error al cargar el archivo");
            System.err.println(e);
            return null;
        }

        int width = bufferedImage.getWidth(null);
        int height = bufferedImage.getHeight(null);
        int[][] pixels = new int[width][height];
        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                pixels[i][j] = bufferedImage.getRGB(i, j);
            }
        }

        return pixels;
    }

```

Para convertir la imagen a una escala de grises

```java
public static int paintItBlack(int argb, boolean _alpha_) {
    int annn, alpha, red, green, bleu;
    bleu = argb & 0xff;
    green = (argb & 0xff00)>>8;
    red = (argb & 0xff0000)>>16;
    if (_alpha_) alpha = argb & 0xff;
    else alpha = 0;
    int n = (bleu + green + red) / 3;

    annn = alpha + n + (n<<8) + (n<<16);
    //System.out.println("ANNN: "+annn+" ALPHA: "+alpha+" N: "+n+" N<<8: "+(n<<8)+" N<<16: "+(n<<16));
    return annn;
}
```

```java
    public static int[][] toGrayScale(int[][] matrix, boolean _alpha_){
        int maxX = matrix.length;
        int maxY = matrix[0].length;
        int [][] pixels = new int[maxX][maxY];

        for (int i = 0; i < maxX; i++) {
            for (int j = 0; j < maxY; j++) {
                pixels[i][j] = paintItBlack(matrix[i][j], _alpha_);
            }
        }

        return pixels;
    }
}
```

Para obtener una imagen a partir de la matrix

```java
public static void saveImageFromMatrix(int[][] matrix, String filename) {
    int width = matrix.length;
    int height =  matrix[0].length;

    BufferedImage bufferedImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            bufferedImage.setRGB(i, j, matrix[i][j]);
        }
    }
    File outputfile = new File(filename);
    try {
        ImageIO.write(bufferedImage, "jpg", outputfile);
    }
    catch(Exception e) {
        System.err.println("ERROR AL GRABAR");
        System.err.println(e);
    }
}
```

4.2. TCPClient

Es la clase que contiene al cliente. Cada instancia TCPClient recibe la imagen como paquete de bytes, la procesa, y envía 4 convoluciones de la imagen misma con 4 diferentes kernel.

4.3. TCPServer

Es la clase que contiene al servidor, invoca los hilos para manejar los sockets. El servidor espera conexiones al puerto 12345, por defecto, y crea un hilo por cada conexión.

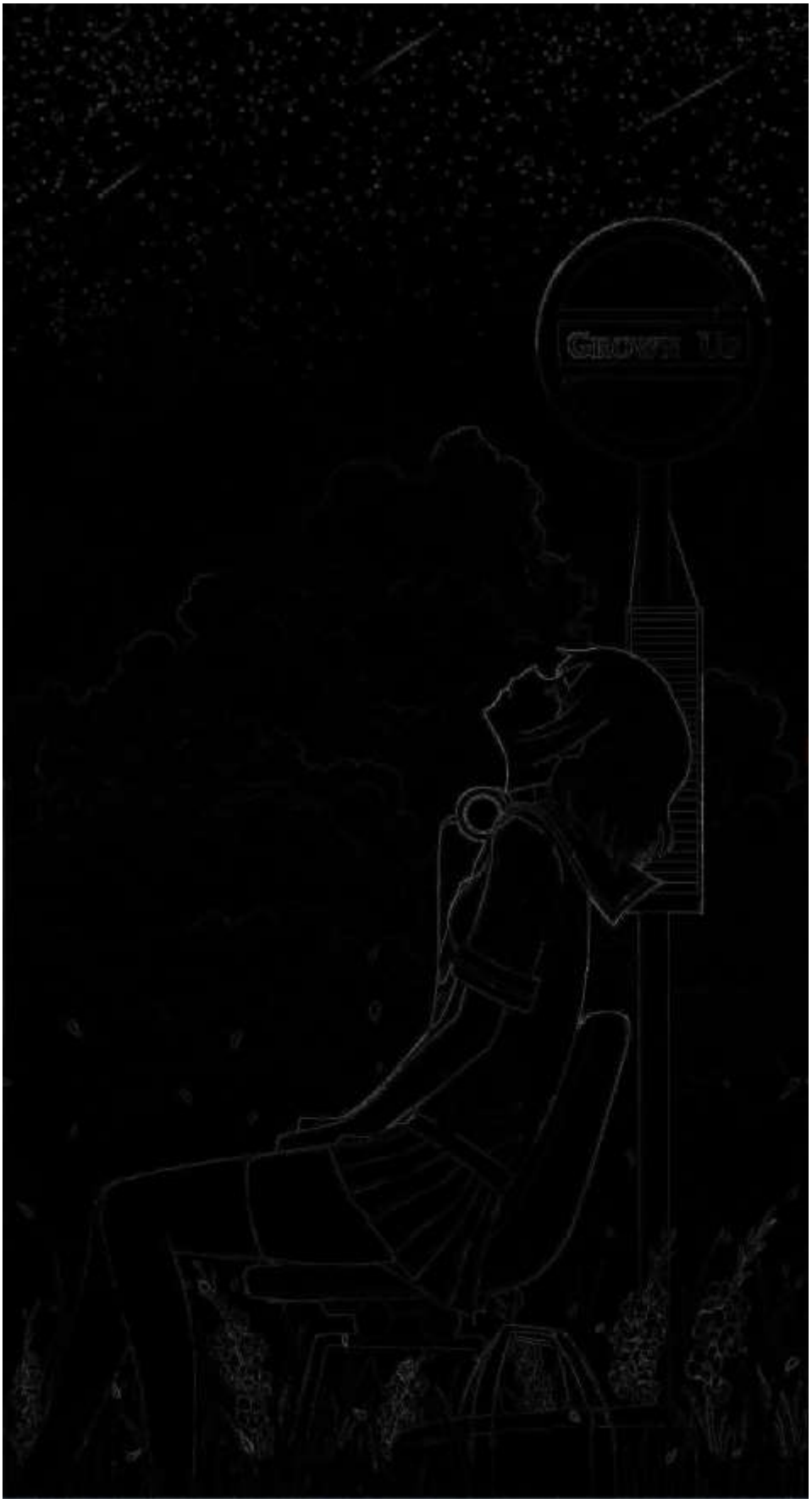
4.4. TCPServerThread

Es la clase que maneja los sockets de un servidor. Envía las imágenes desde el servidor a los clientes para que estos las procesen. También se encarga de recepcionar las imágenes procesadas y las guarda en el servidor.

El código en su totalidad puede encontrarse en la siguiente liga:
https://github.com/marcoleonrios/CC462/

5. Resultados


GROWN UP


GROWN UP


GROWN UP

6. Conclusiones

   Nuestro algoritmo resuelve el problema en buen tiempo con un trabajo equitativo por parte de los nodos del cluster con excepción de los tiempos de espera para la transmisión y recepción de las imágenes.

7. Referencias


- https://brilliant.org/wiki/recursive-backtracking/
- https://www.saama.com/blog/different-kinds-convolutional-filters/
- https://www.geeksforgeeks.org/the-c-standard-template-library-stl/
- https://docs.gimp.org/2.8/en/plug-in-convmatrix.html
- https://wallhaven.cc/