**Universidad Nacional de La Matanza**

**Departamento de Ingeniería e Investigaciones Tecnológicas**

**Programación Avanzada**

**Complejidad Computacional**

**Filtro con kernel variable**

**Grupo: Eta-1**

**Días de Cursada:** Jueves  **Turno:** Noche **Aula:** Lab 522 - Ingeniería

**Docentes:** Videla, Lucas

Aubin, Veronica Inés

**Integrantes:**

Schereik, Brenda DNI: 45128557

Coronel, Thiago DNI: 44834604

Perez, Thiago DNI: 43097103

Ruggieri, Franco DNI: 43630882

Arab, Santiago DNI: 46500280

Franco, Francisco DNI: 42116434

**Contenido**

[Introducción 3](#_30j0zll)

[Desarrollo 3](#_il9847clpvjp)

[Filtros con kernel variable 3](#_1fob9te)

[Tipos de filtros 3](#_jv7uihr6kd0e)

[Formato PGM ASCII 5](#_45cwd9fjlnab)

[Algoritmo 5](#_xv4hmdi48xmg)

[Complejidad computacional 6](#_d83xyvuf2zuk)

[Resultados 7](#_cb94k8dlnmy1)

[Conclusión 8](#_2s8eyo1)

[Bibliografía 10](#_17dp8vu)

# Introducción

En el procesamiento digital de imágenes, una imagen en escala de grises puede representarse como una matriz bidimensional de enteros, donde cada valor indica la intensidad de un píxel: valores bajos representan tonos oscuros y valores altos, tonos claros. Este enfoque permite aplicar algoritmos directamente sobre la estructura matricial para mejorar, transformar o analizar imágenes.

Una técnica fundamental en este contexto es el uso de filtros con kernel, también conocidos como máscaras de convolución. Estos filtros modifican la imagen aplicando una operación matemática local sobre cada píxel y sus vecinos, lo que permite obtener diversos efectos.

En este trabajo, el objetivo principal es analizar la complejidad computacional de un algoritmo que implementa el filtrado de imágenes con kernel variable, aplicado sobre imágenes en escala de grises. Se estudiará cómo influye el tamaño de la imagen y del kernel en el rendimiento de este algoritmo, utilizando notación Big O como herramienta de análisis. Para validar el estudio, se desarrollará una implementación práctica en Java y se aplicará el mismo algoritmo a distintos filtros y tamaños de entrada, con el fin de observar y comparar los resultados obtenidos.

# Desarrollo

## Filtros con kernel variable

En el procesamiento de imágenes, los filtros con kernel variable son técnicas que permiten modificar una imagen aplicando una operación matemática sobre los píxeles y sus vecinos. Un kernel (o máscara) es una pequeña matriz de valores que se superpone sobre cada píxel de la imagen para realizar una operación llamada convolución.

Durante la convolución, el kernel se desplaza sobre toda la imagen, píxel por píxel. En cada posición, se multiplica cada valor del kernel por el valor del píxel correspondiente de la imagen (en la vecindad del píxel actual), y luego se suman todos esos productos. El resultado reemplaza al valor del píxel central en una nueva imagen.

Este mecanismo permite aplicar una gran variedad de efectos a la imagen original, donde el comportamiento del filtro depende completamente de los valores del kernel y de su tamaño. Los más comunes son de tamaño 3x3, aunque pueden usarse kernels más grandes para efectos más suaves o más intensos.

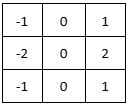
## 

## Tipos de filtros

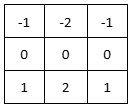
Los distintos tipos de filtros con kernel variable se diferencian por los valores que contienen sus máscaras y el efecto que producen sobre la imagen. A continuación se describen los filtros seleccionados para este trabajo.

1. Blur (Suavizado / Desenfoque)

Este filtro se utiliza para reducir el ruido o suavizar una imagen. Promedia los valores de píxeles cercanos para suavizar las transiciones de color. Es útil en aplicaciones donde se desea reducir detalles finos o eliminar imperfecciones en la imagen.

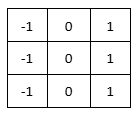
1. Sobel Horizontal (Detección de bordes horizontales)

Se utiliza para detectar bordes horizontales en una imagen. Resalta las áreas donde los píxeles cambian bruscamente de valor en la dirección horizontal, lo que suele indicar un borde.



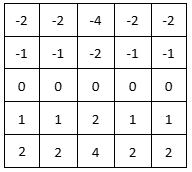
1. Sobel Vertical (Detección de bordes verticales)

Similar al horizontal, pero está diseñado para detectar bordes verticales. Resalta los bordes donde los píxeles cambian de intensidad de manera brusca en la dirección vertical



1. Prewitt Horizontal (Bordes horizontales)

También detecta bordes horizontales, pero usa una matriz diferente a la de Sobel. Es similar en concepto, pero su enfoque es más simple y puede ser menos preciso que el filtro Sobel en algunas situaciones.

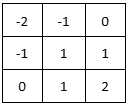


1. Prewitt Vertical (Bordes verticales)

Al igual que el filtro Prewitt horizontal, el Prewitt vertical se usa para detectar bordes verticales en una imagen, con un enfoque diferente al de Sobel.

1. Laplaciano (Detección de bordes general)

Se utiliza para detectar bordes generales, resaltando los contornos y los detalles finos al detectar cambios rápidos de intensidad. Existe el operador Laplaciano Positivo ( se utiliza para eliminar los bordes exteriores de una imagen) y el Negativo ( se utiliza para eliminar los bordes internos de una imagen )

1. Emboss (Repujado)

Crea un efecto de relieve (o repujado) en la imagen, dándole una apariencia tridimensional. Al hacerlo, crea sombras en las áreas donde hay transiciones de intensidad, imitando la apariencia de una superficie elevada.

## 

## Formato PGM ASCII

Para este trabajo, se utilizará el formato PGM (Portable GrayMap), un tipo de archivo simple y ampliamente utilizado para representar imágenes en escala de grises. Este formato forma parte de la familia Netpbm y permite almacenar la información de la imagen como una matriz de valores enteros, lo que lo convierte en una opción ideal para aplicar algoritmos de procesamiento a nivel de píxel.

El formato PGM tiene dos variantes:

* P2: codificación en texto (ASCII).
* P5: codificación en binario (más eficiente en tamaño y velocidad, pero menos legible).

En esta ocasión, se utilizará la variante P2 por su simplicidad de lectura y escritura desde código.

Estructura

P2

# Comentarios opcionales

<ancho> <alto>

<valor máximo>

<píxeles...>

## 

## Algoritmo

public static PGMImage applyFilter(PGMImage img, int[][] kernel) {

int height = img.height;

int width = img.width;

int kSize = kernel.length;

int kOffset = kSize / 2;

int kernelSum = calcularSumaKernel(kernel);

int[][] newPixels = new int[height][width];

for (int i = kOffset; i < height - kOffset; i++) {

for (int j = kOffset; j < width - kOffset; j++) {

int sum = 0;

for (int ki = -kOffset; ki <= kOffset; ki++) {

for (int kj = -kOffset; kj <= kOffset; kj++) {

int pixel = img.pixels[i + ki][j + kj];

int kernelValue = kernel[ki + kOffset][kj + kOffset];

sum += kernelValue \* pixel;

}

}

if (kernelSum != 0) {

sum /= kernelSum;

}

sum = Math.*max*(0, Math.*min*(img.maxValue, sum));

newPixels[i][j] = sum;

}

}

return new PGMImage(img.width, img.height, img.maxValue, newPixels);

}

El algoritmo implementado, en primer lugar, obtiene las dimensiones de la imagen y del kernel, calcula el offset para centrarlo, y crea una nueva matriz para los resultados. Luego recorre los píxeles internos de la imagen, multiplica cada valor del kernel por el píxel correspondiente, acumula la suma y la limita al rango válido. Finalmente, guarda el resultado en una nueva imagen y la retorna.

## 

## Complejidad computacional

El algoritmo implementado aplica una operación de convolución con kernel variable de K x K sobre una imagen en escala de grises representada como una matriz de tamaño M x N, donde:

* M es la altura (número de filas).
* N es el ancho (número de columnas).
* K es el tamaño del kernel

Complejidad temporal

Calcular la complejidad temporal de un algoritmo implica analizar cuántas operaciones básicas (como sumas, multiplicaciones, comparaciones, etc) se ejecutan en función del tamaño de la entrada.

Como primer paso, habría que identificar bucles. En este caso, contamos con varios bucles anidados:

for (int i = kOffset; i < height - kOffset; i++) {

for (int j = kOffset; j < width - kOffset; j++) {

// Operaciones constantes

for (int ki = -kOffset; ki <= kOffset; ki++) {

for (int kj = -kOffset; kj <= kOffset; kj++) {

// Operaciones constantes

}

}

// Operaciones constantes

}

}

Los bucles externos recorren casi todos los píxeles de la imagen, aproximadamente height × width, es decir O(). Mientras que los bucles internos recorren todos los elementos del kernel, K x K, eso es .

Como los bucles están anidados, se multiplican sus complejidades:

=

Dentro de los bucles se realizan operaciones constantes, con O(1), es decir no alteran la complejidad general.

Por lo tanto, la complejidad temporal es:

Tipo de complejidad

El tipo de complejidad se refiere a cómo crece el tiempo de ejecución del algoritmo en relación con el tamaño de sus entradas.

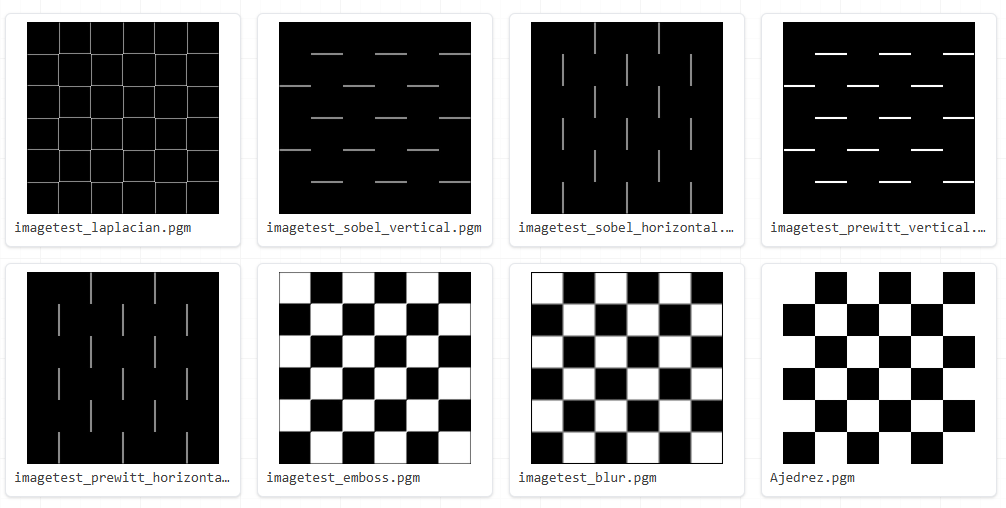
La complejidad del algoritmo que analizamos se considera polinomial, ya que el tiempo de ejecución del algoritmo crece de acuerdo a un polinomio en función de las variables de entrada. En este caso, se trata de un polinomio de grado 4, compuesto por dos variables lineales (H y W) y una cuadrática ().

## 

## Resultados

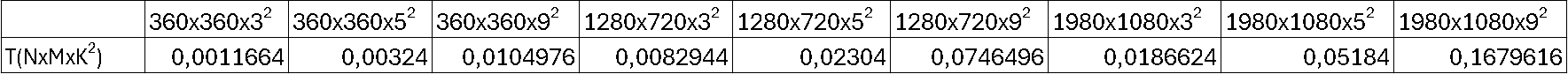
Para evaluar el comportamiento del algoritmo y observar cómo influyen las distintas entradas, se aplicaron los siete filtros seleccionados sobre una serie de imágenes en formato PGM, variando el contenido y tamaño de las mismas. ( Estos resultados no reflejan el uso de los mismos valores de Kernels que los ejemplificados en el apartado “Tipos de Filtros” )

### 



* El tiempo de ejecución depende tanto de la resolución de la imágen como del tamaño del kernel aplicado a la misma. Para el mismo tamaño de kernel, crece linealmente entre diferentes resoluciones de imágenes mientras que para la misma resolución y cambiando el kernel, crece cuadráticamente.
* El tipo de filtro no afecta la complejidad del algoritmo, ya que todos siguen la misma estructura de convolución. Lo que cambia es el resultado visual, no la cantidad de operaciones.

La siguiente tabla muestra el tiempo en segundos para una máquina que ejecuta mil millones de instrucciones por segundo (1/1000000000)s



# 

# Conclusión

En este trabajo se analizó la complejidad computacional de un algoritmo de filtrado de imágenes mediante convolución con kernel variable. A través del estudio teórico y la implementación práctica en Java, se comprobó que el algoritmo presenta una complejidad temporal polinomial respecto al número de píxeles de la imagen y del tamaño del kernel, es decir, O() para kernels de tamaño variable.

Los distintos filtros aplicados permitieron visualizar cómo el mismo algoritmo puede generar efectos visuales diferentes según el kernel utilizado. Sin embargo, es importante destacar que el tamaño del kernel impacta en el rendimiento computacional, ya que un kernel más grande implica una mayor cantidad de operaciones por píxel. Esto refuerza la idea de que los valores del kernel determinan la función visual del filtro, mientras que el costo computacional del algoritmo está ligado tanto a la resolución de la imagen como al tamaño del kernel utilizado.

Finalmente, nuestros resultados experimentales confirmaron el análisis teórico: el tiempo de ejecución crece de forma lineal con la cantidad de píxeles de la imagen y cuadrática con el tamaño del kernel aplicado. Esto valida el enfoque adoptado y demuestra que la evaluación de la complejidad computacional es fundamental para comprender y anticipar el rendimiento de algoritmos como estos.

# Bibliografía

[1] Wikipedia, “Kernel (image processing),” [Online]. Disponible en: <https://en.wikipedia.org/wiki/Kernel_(image_processing)>

[2] GIMP Documentation, “Convolution Matrix Filter,” Version 2.6, [Online]. Disponible en: <https://docs.gimp.org/2.6/en/plug-in-convmatrix.html>

[3] F. Querejeta and P. Simbeni, “Procesamiento de Imágenes: Matriz de Convolución,” Notas de Aplicación FVC, Departamento de Ingeniería Eléctrica y de Computadoras, Universidad Nacional del Sur, Argentina. [Online]. Disponible en:

[Filtros y convoluciones (Procesamiento de imágenes) (página 2)](https://www.monografias.com/trabajos108/filtros-y-convoluciones/filtros-y-convoluciones2)  
[4] P. Querejeta y P. Simbeni, Procesamiento Digital de Imágenes: Filtros y convoluciones, Universidad Nacional del Sur, Bahía Blanca, Argentina. [En línea]. Disponible en: <http://lcr.uns.edu.ar/fvc/NotasDeAplicacion/FVC-QuerejetaSimbeniPedro.pdf>

[5] Wikipedia, “Operador Sobe” [En línea]. Disponible en: <https://es.wikipedia.org/wiki/Operador_Sobel>

[6] Academia Lab, “Operador Prewitt” [En línea]. Disponible en: <https://academia-lab.com/enciclopedia/operador-prewitt/>

[7] “Filtros y convoluciones (Procesamiento de imágenes), página 2.” [En línea]. Disponible en: <https://www.monografias.com/trabajos108/filtros-y-convoluciones/filtros-y-convoluciones2>