

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

# Desempeño en tiempo de ejecución de algoritmos de filtrado de imágenes

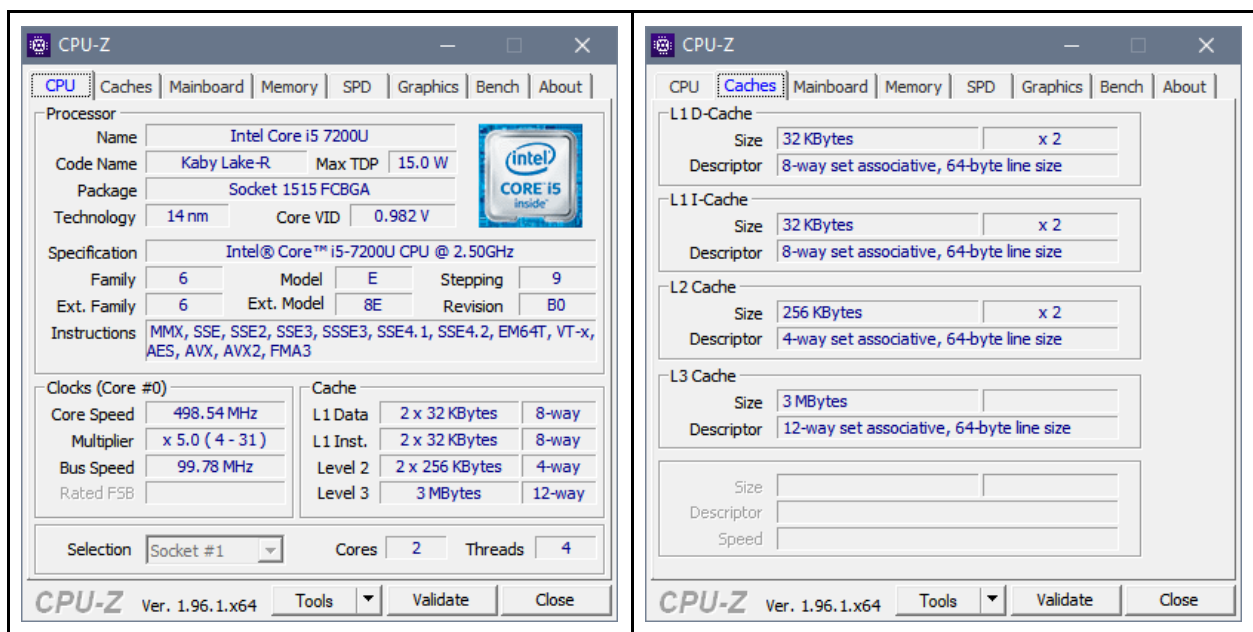
## Descripción del problema

El experimento se realiza con el fin de medir el impacto de diferentes versiones de algoritmo para filtración de imágenes en el tiempo de ejecución, cada versión con distintos grados de aprovechamiento del principio de localidad espacial. La técnica de filtrado que se usará es la de convolución de matrices. El filtro o kernel que se utilizará es el de detección de bordes, el formato de las imágenes de entrada es el de mapa de bits (bmp) en escala de grises con profundidad de 8 bits, el sistema operativo en que se desarrolla y prueba es Windows 10 Home de 64 bits, el lenguaje de programación es C# y se tienen tres unidades experimentales, una por cada computador personal de los integrantes. Los factores que se tendrán en cuenta son de dos tipos: software y hardware. A continuación se listan los factores experimentales de interés.

## Factores experimentales

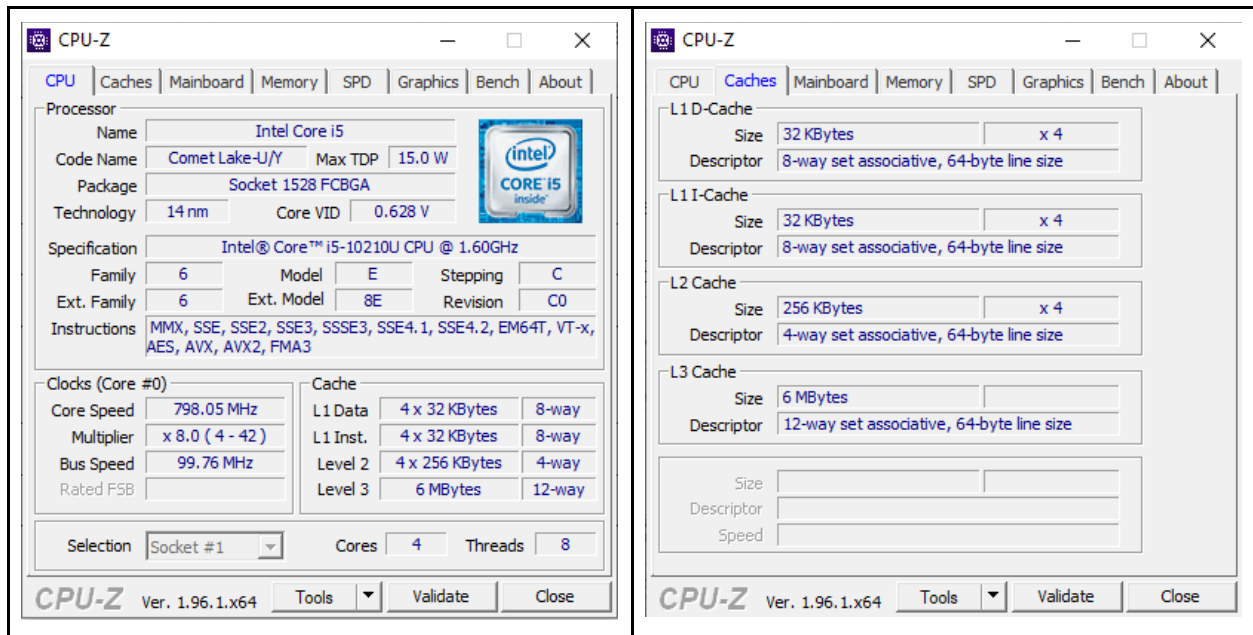
### Recursos computacionales

Daniel:

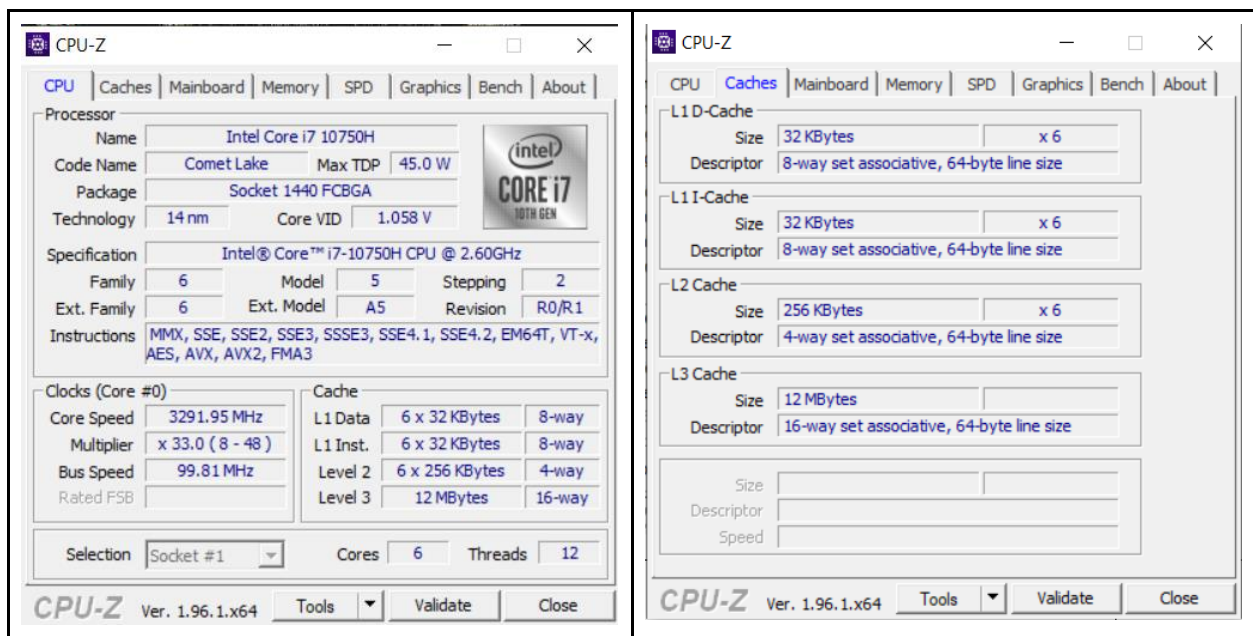


Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

Cristian:



Jose:



## Algoritmo

A continuación se presentan las 6 versiones de algoritmo utilizadas, 4 de ellas simplemente se basaron en el cambio de orden de los ciclos, las otras dos combinan esta idea con el concepto

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

de desenroscado de bucles<sup>3</sup>. Este concepto consiste en la eliminación de ciclos con el fin de prescindir de variables de conteo, instrucciones de comparación y de cambio. Dichos ciclos se remplazan por las instrucciones individuales que habrían ejecutado. La aplicación de esta estrategia a menudo resulta en un impacto significativamente positivo en el tiempo de ejecución de algoritmos, a pesar de que la complejidad temporal sea equivalente. El tipo de archivo BMP se compone de 1078 bits de offset<sup>1</sup> (sección reservada del archivo para metadatos) y el resto que contiene los datos de la imagen. Por esta razón se accede al arreglo de datos empezando en la posición 1078, valor que se encuentra almacenado en la variable estática *offset*.

1 - x-y-i-j

```
static byte[] filteringAlgorithmXYIJ(byte[] data)
{
    byte[] C = (byte[])data.Clone();
    int n = (int)Math.Sqrt(data.Length - offset);
    int xlim = offset + n * (n - 1) + 1 - n; // subtract n because image lower limit is ignored
    long sum = 0;
    long count = 0;
    long start = DateTime.Now.Ticks;
    int k = kernel.Length;
    for (int x = offset + n; x < xlim; x += n) // starts at offset + n because image upper limit of the image is ignored
    {
        for (int y = 1; y < n - 1; y++) // starts at 1 because
        {
            C[x + y] = 0; // resets target matrix on the go
            for (int i = 0; i < k; i++)
            {
                for (int j = 0; j < k; j++)
                {
                    int row = x + (i * n) - n - 1;
                    int col = y + j - 1;

                    C[x + y] = (byte)(C[x + y] + data[row + col] * kernel[i][j]);
                }
            }
            count++;
            sum += C[x + y];
        }
    }
    long end = DateTime.Now.Ticks;
    // according to https://docs.microsoft.com/en-us/dotnet/api/system.datetime.ticks?view=net-5.0#remarks, Ticks are 100
    Console.WriteLine("Time executing(ns) is " + ((end - start) * 100)); // print time in nanoseconds
    Console.WriteLine("The sum of the processed pixels is " + sum);
    Console.WriteLine("The number of pixels processed is " + count);
    return C;
}
```

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

2 - x-y-j-i

```
static byte[] filteringAlgorithmXYJI(byte[] data)
{
    byte[] C = (byte[])data.Clone();
    int offset = 1078;
    int n = (int)Math.Sqrt(data.Length - offset);
    int xlim = offset + n * (n - 1) + 1 - n; // subtract n because image lower limit is ignored
    long sum = 0;
    long count = 0;
    int k = 3;
    long start = DateTime.Now.Ticks;
    for (int x = offset + n; x < xlim; x += n) // starts at offset
                                                // + n because image upper limit of the image is ignored
    {
        for (int y = 1; y < n - 1; y++) // starts at 1 because
        {
            C[x + y] = 0; // resets target matrix on the go
            for (int j = 0; j < k; j++)
            {
                for (int i = 0; i < k; i++)
                {
                    int row = x + (i * n) - n - 1;
                    int col = y + j - 1;

                    C[x + y] = (byte)(C[x + y] + data[row + col] * kernel[i][j]);
                }
            }
            count++;
            sum += C[x + y];
        }
    }
    long end = DateTime.Now.Ticks;
    // according to https://docs.microsoft.com/en-us/dotnet/api/system.datetime.ticks?view=net-5.0#remarks
    Console.WriteLine("Time executing(ns) is " + ((end - start) * 100)); // print time in nanoseconds
    Console.WriteLine("The sum of the processed pixels is " + sum);
    Console.WriteLine("The number of pixels processed is " + count);
    return C;
}
```

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

### 3 - x-y-unrolling

```
static byte[] filteringAlgorithmXYUnrolling(byte[] data)
{
    byte[] C = (byte[])data.Clone();
    int n = (int)Math.Sqrt(data.Length - offset);
    int xlim = offset + n * (n - 1) + 1 - n; // subtract n because image lower limit is ignored
    long sum = 0;
    long count = 0;
    long start = DateTime.Now.Ticks;
    for (int x = offset + n; x < xlim; x += n) // starts at offset + n because image upper limit of the image is ignored
    {
        for (int y = 1; y < n - 1; y++) // starts at 1 because
        {
            // f(i,j) = X + (i-1)*n + y + j - 2
            C[x + y] = (byte)
                (data[x - n + y - 2] * kernel[0][0] // 0,0
                + data[x - n - 1 + y] * kernel[0][1] // 0,1
                + data[x - n + y] * kernel[0][2] // 0,2
                + data[x + y - 2] * kernel[1][0] // 1,0
                + data[x + y - 1] * kernel[1][1] // 1,1
                + data[x + y] * kernel[1][2] // 1,2
                + data[x + n + y - 2] * kernel[2][0] // 2,0
                + data[x + n + y - 1] * kernel[2][1] // 2,1
                + data[x + n + y] * kernel[2][2] // 2,2
                );

            count++;
            sum += C[x + y];
        }
    }
    long end = DateTime.Now.Ticks;
    // according to https://docs.microsoft.com/en-us/dotnet/api/system.datetime.Ticks?view=net-5.0#remarks, Ticks are 10
    Console.WriteLine("Time executing(ns) is " + ((end - start) * 100)); // print time in nanoseconds
    Console.WriteLine("The sum of the processed pixels is " + sum);
    Console.WriteLine("The number of pixels processed is " + count);
    return C;
}
```

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

4 - y-x-i-j

```
static byte[] filteringAlgorithmVXIJ(byte[] data)
{
    byte[] C = (byte[])data.Clone();
    int n = (int)Math.Sqrt(data.Length - offset);
    int xlim = offset + n * (n - 1) + 1 - n;
    long sum = 0;
    long count = 0;
    long start = DateTime.Now.Ticks;
    int k = kernel.Length;
    for (int y = 1; y < n - 1; y++)
    {
        for (int x = offset + n; x < xlim; x += n)
        {
            C[y + x] = 0;
            for (int i = 0; i < k; i++)
            {
                for (int j = 0; j < k; j++)
                {
                    int col = y + j - 1;
                    int row = x + (i * n) - n - 1;

                    C[y + x] = (byte)(C[y + x] + data[col + row] * kernel[i][j]);
                }
            }
            count++;
            sum += C[y + x];
        }
    }
    long end = DateTime.Now.Ticks;
    // according to https://docs.microsoft.com/en-us/dotnet/api/system.datetime.ticks
    Console.WriteLine("Time executing(ns) is " + ((end - start) * 100)); // print
    Console.WriteLine("The sum of the processed pixels is " + sum);
    Console.WriteLine("The number of pixels processed is " + count);
    return C;
}
```

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

5 - y-x-j-i

```
static byte[] filteringAlgorithmVXJI(byte[] data)
{
    byte[] C = (byte[])data.Clone();
    int offset = 1078;
    int n = (int)Math.Sqrt(data.Length - offset);
    int xlim = offset + n * (n - 1) + 1 - n;
    long sum = 0;
    long count = 0;
    long start = DateTime.Now.Ticks;
    int k = kernel.Length;
    for (int y = 1; y < n - 1; y++)
    {
        for (int x = offset + n; x < xlim; x += n)
        {
            C[y + x] = 0;
            for (int j = 0; j < k; j++)
            {
                for (int i = 0; i < k; i++)
                {
                    int col = y + j - 1;
                    int row = x + (i * n) - n - 1;

                    C[y + x] = (byte)(C[y + x] + data[col + row] * kernel[i][j]);
                }
            }
            count++;
            sum += C[y + x];
        }
    }
    long end = DateTime.Now.Ticks;
    // according to https://docs.microsoft.com/en-us/dotnet/api/system.datetime.ticks
    Console.WriteLine("Time executing(ns) is " + ((end - start) * 100)); // print time
    Console.WriteLine("The sum of the processed pixels is " + sum);
    Console.WriteLine("The number of pixels processed is " + count);
    return C;
}
```

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

## 6 - y-x-unrolling

```
static byte[] filteringAlgorithmYXUnrolling(byte[] data)
{
    byte[] C = (byte[])data.Clone();
    int n = (int)Math.Sqrt(data.Length - offset);
    int xlim = offset + n * (n - 1) + 1 - n; // subtract n because image lower limit is ignored
    long sum = 0;
    long count = 0;
    long start = DateTime.Now.Ticks;
    for (int y = 1; y < n - 1; y++) // starts at offset + n because image upper limit of the image is ignored
    {
        for (int x = offset + n; x < xlim; x += n) // starts at 1 because
        {
            // f(i,j) = x + (i-1)*n + y + j - 2
            C[x + y] = (byte)
            (
                data[x - n + y - 2] * kernel[0][0] // 0,0
                + data[x - n - 1 + y] * kernel[0][1] // 0,1
                + data[x - n + y] * kernel[0][2] // 0,2
                + data[x + y - 2] * kernel[1][0] // 1,0
                + data[x + y - 1] * kernel[1][1] // 1,1
                + data[x + y] * kernel[1][2] // 1,2
                + data[x + n + y - 2] * kernel[2][0] // 2,0
                + data[x + n + y - 1] * kernel[2][1] // 2,1
                + data[x + n + y] * kernel[2][2] // 2,2
            );

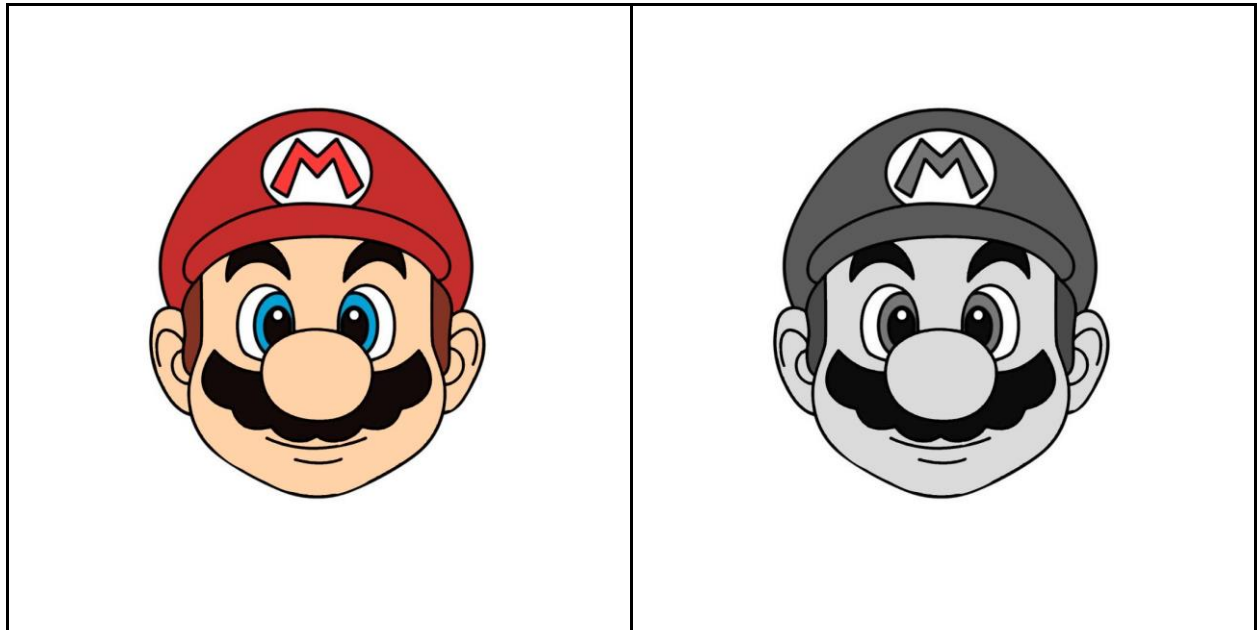
            count++;
            sum += C[x + y];
        }
    }
    long end = DateTime.Now.Ticks;
    // according to https://docs.microsoft.com/en-us/dotnet/api/system.datetime.Ticks?view=net-5.0#remarks, Ticks are in nanoseconds
    Console.WriteLine("Time executing(ns) is " + ((end - start) * 100)); // print time in nanoseconds
    Console.WriteLine("The sum of the processed pixels is " + sum);
    Console.WriteLine("The number of pixels processed is " + count);
    return C;
}
```

## Imágenes de entrada

Después de varias pruebas preliminares del algoritmo pudimos observar que la imagen resultante del filtrado tenía los bordes mejor identificados entre menos texturas, colores, sombras y profundidades tuviera la imagen de entrada. Es por esto que decidimos utilizar una imagen con colores sólidos y bien definidos, pues además de realizar el experimento y hacer un análisis de los datos, nos interesaba ver resultados satisfactorios de la aplicación de la convolución de matrices para el filtrado de imágenes.

Imagen original 1024x1024 px	Referencia en escala de grises a 8 bits de profundidad
------------------------------	--





Para este experimento se tomó en cuenta la capacidad de almacenamiento de las cachés de nivel 1 al 3 de los tres computadores, basado en ello y tratando de que la imagen se almacene en todos estos niveles se proponen las siguientes dimensiones para la imagen de entrada, donde se trató de seleccionar dichas dimensiones de manera que se ocupara proporciones similares del nivel 3 de caché. El proceso para elegir estos tamaños fue estimar la cantidad de bytes de la imagen resultante del cambio de dimensiones mediante la expresión  $1078 + n^2$ , siendo  $n$  el lado de la imagen en píxeles y compararlos con las proporciones de la caché acordadas.

#### L1 caché

- 100x100 (11078 bytes. 10,8 KB)
- 150x150 (23878 bytes. 23,3 KB)
- 178x178 (33118 bytes. 32,3 KB)

#### L2 caché

- 250x250 (64078 bytes. 62,5 KB)
- 327x327 (108334 bytes. 105 KB)
- 429x429 (186406 bytes. 182 KB)
- 511x511 (262710 bytes. 256 KB)

#### 3 MB L3 caché (Daniel)

- 724x724 (525254 bytes. 512 KB)
- 1024x1024 (1049654 bytes. 1 MB)
- 1440x1440 (2074678 bytes. 1,97 MB)

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

- 1580x1580 (2497478 bytes. 2,38 MB)
- 1776x1776 (3155254 bytes. 3,0 MB)

#### 6 MB L3 caché (Cristian)

- 1440x1440 (2074678 bytes. 1,97 MB)
- 1776x1776 (3155254 bytes. 3,0 MB)
- 2023x2023 (4095630 bytes. 3,9 MB)
- 2255x2255 (5088358 bytes. 4,85 MB)
- 2508x2508 (6291142 bytes. 5,99 MB)

#### 12 MB L3 caché (Jose)

- 2023x2023 (4095630 bytes. 3,9 MB)
- 2508x2508 (6291142 bytes. 5,99 MB)
- 2900x2900 (8411078 bytes. 8,02 MB)
- 3150x3150 (9929878 bytes. 9,46 MB)
- 3550x3550 (12610678 bytes. 12 MB)

## Tipo de experimento e hipótesis

Se escogieron 2 tipos de experimento. Para el análisis individual (por unidad experimental) se desarrolló el bifactorial, ya que solo se analiza el impacto en el tiempo de los factores algoritmo y tamaño de imagen. Para el análisis conjunto de las tres unidades experimentales se desarrolló el factorial de tres factores, donde el factor adicional era el computador. Para cada factor se debe sacar una hipótesis nula y una alternativa, pero teniendo en cuenta los tipos de experimento y los factores experimentales, y el hecho de que buscamos probar una igualdad o diferencia en todos los casos, se puede sacar una hipótesis común que sirva para todas las pruebas de hipótesis en el experimento, por lo cual las hipótesis que se plantearon fueron las siguientes:

**H<sub>0</sub>:** Los tiempos normalizados de ejecución son iguales.

**H<sub>a</sub>:** Al menos un par de tiempos normalizados de ejecución difiere.

## Factores de ruido

Algunos factores de influencia sobre la variable de respuesta no pudieron ser controlados estrictamente al punto de anular su efecto. Entre ellos se encuentran, por mencionar algunos, el uso del navegador para registrar los datos en un documento compartido, el uso de Discord para estar en videollamada y coordinar el procedimiento, y los procesos en segundo plano del sistema y otros programas. El impacto de estos factores se trató de minimizar cerrando todos los programas no necesarios y teniendo la mínima cantidad de pestañas abiertas en el navegador. Las interrupciones del sistema también son un factor de influencia no controlable,

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

sin embargo, se trató de no usar periféricos como el ratón y el teclado durante la ejecución del programa, así mismo se evitó por completo el uso de instrucciones de impresión por consola dentro de los ciclos principales del algoritmo, siendo la cantidad de interrupciones minimizada en el experimento.

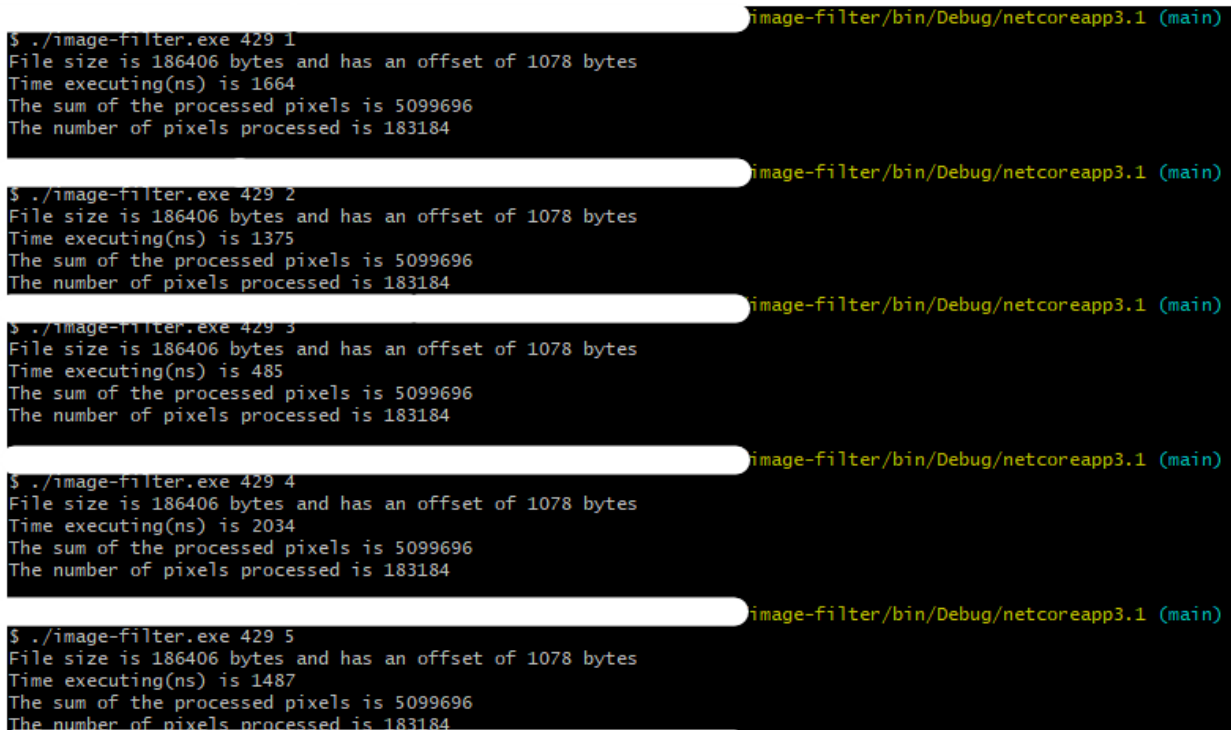
## Variable de respuesta

Cómo se explica en la descripción del problema, la variable de respuesta que se medirá en el experimento es el tiempo de ejecución, la unidad de medida son nanosegundos y se toma desde que inicia hasta que termina el ciclo más externo de la aplicación del filtro.

## Desarrollo del experimento

### Validaciones preliminares

Lo primero que se verificó fue que las seis versiones del algoritmo produjeran los mismos resultados. Para esto, además de aplicar el filtro, en el ciclo se aumenta un contador que indica la cantidad de píxeles que pasan por el procesamiento y se imprime esto junto con la suma de todos los resultados obtenidos para cada píxel.



```
$ ./image-filter.exe 429 1
File size is 186406 bytes and has an offset of 1078 bytes
Time executing(ns) is 1664
The sum of the processed pixels is 5099696
The number of pixels processed is 183184

$ ./image-filter.exe 429 2
File size is 186406 bytes and has an offset of 1078 bytes
Time executing(ns) is 1375
The sum of the processed pixels is 5099696
The number of pixels processed is 183184

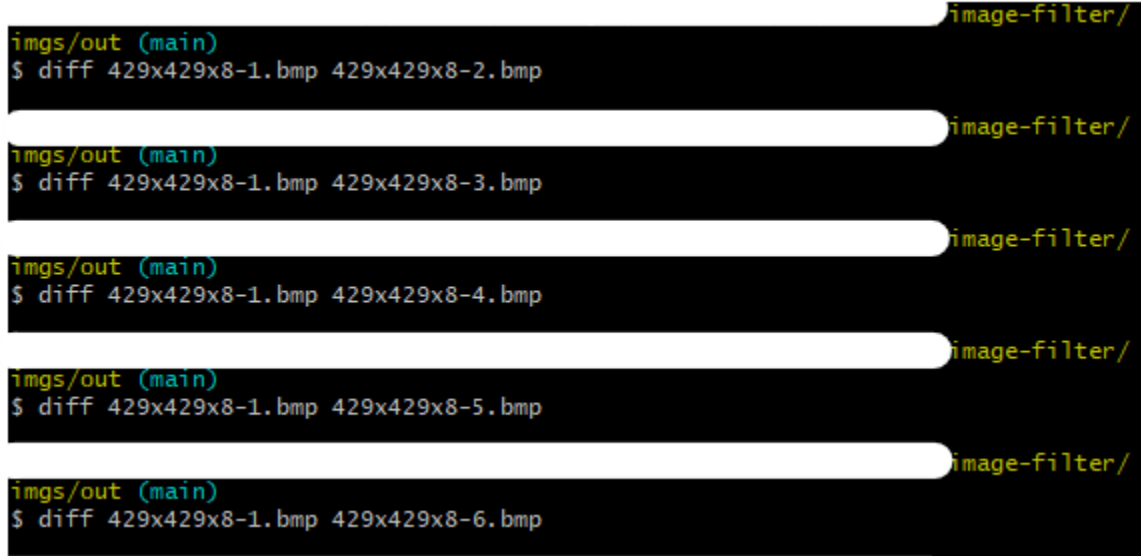
$ ./image-filter.exe 429 3
File size is 186406 bytes and has an offset of 1078 bytes
Time executing(ns) is 485
The sum of the processed pixels is 5099696
The number of pixels processed is 183184

$ ./image-filter.exe 429 4
File size is 186406 bytes and has an offset of 1078 bytes
Time executing(ns) is 2034
The sum of the processed pixels is 5099696
The number of pixels processed is 183184

$ ./image-filter.exe 429 5
File size is 186406 bytes and has an offset of 1078 bytes
Time executing(ns) is 1487
The sum of the processed pixels is 5099696
The number of pixels processed is 183184
```

Adicional a ello se verificó las imágenes de salida con el programa de consola *diff*, el cual corrobora que todos los algoritmos produjeron la misma imagen.

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles



```
image-filter/  
imgs/out (main)  
$ diff 429x429x8-1.bmp 429x429x8-2.bmp  
  
image-filter/  
imgs/out (main)  
$ diff 429x429x8-1.bmp 429x429x8-3.bmp  
  
image-filter/  
imgs/out (main)  
$ diff 429x429x8-1.bmp 429x429x8-4.bmp  
  
image-filter/  
imgs/out (main)  
$ diff 429x429x8-1.bmp 429x429x8-5.bmp  
  
image-filter/  
imgs/out (main)  
$ diff 429x429x8-1.bmp 429x429x8-6.bmp
```

## Recolección de datos

La metodología definida para recolectar datos fue la de repeticiones consecutivas de las combinaciones de factores de interés. Se tomaron en cuenta 3 de 4 repeticiones, ignorando la primera con el fin de reducir posibles variabilidades. Este mismo proceso se llevó a cabo en cada máquina de prueba, dando un resultado de 216 mediciones por computador y 648 en todo el experimento. Se decidió ejecutar el programa por consola para evitar cargar cualquier herramienta de debug o procesos extra de la IDE.

Los pasos que se siguieron en todas las unidades experimentales pueden ser resumidos de la siguiente manera:

1. Reiniciar el equipo de cómputo.
2. Cerrar todos los programas innecesarios corriendo en segundo plano.
3. Abrir una consola o terminal de comandos en el directorio donde se encuentra el ejecutable de nuestro programa.
4. Ejecutar las repeticiones de las combinaciones de factores definidas anteriormente y registrar los datos en el archivo compartido en Google Drive.

Los datos recolectados se encuentran en el archivo *ANOVA-jimenez-lasso-fernandez.xlsx* dentro del directorio raíz del proyecto.

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

## Resultados

### Análisis de varianza por computadora

Daniel

#### Información del factor

Factor	Tipo	Niveles	Valores
Algorithm Version	Fijo	6	1-XYIJ; 2-XYJI; 3-XYUnrolling; 4-YXIJ; 5-YXJI; 6-YXUnrolling
n (img size nxn)	Fijo	12	100; 150; 178; 250; 327; 429; 511; 724; 1024; 1440; 1580; 1776

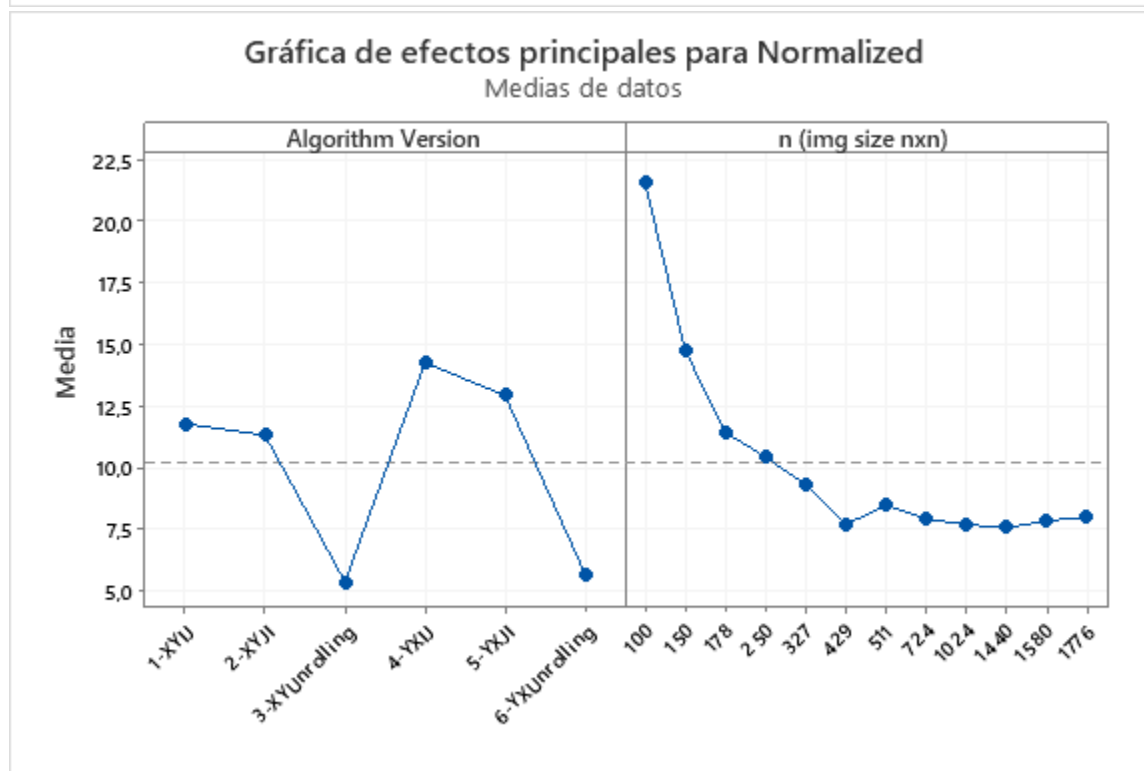
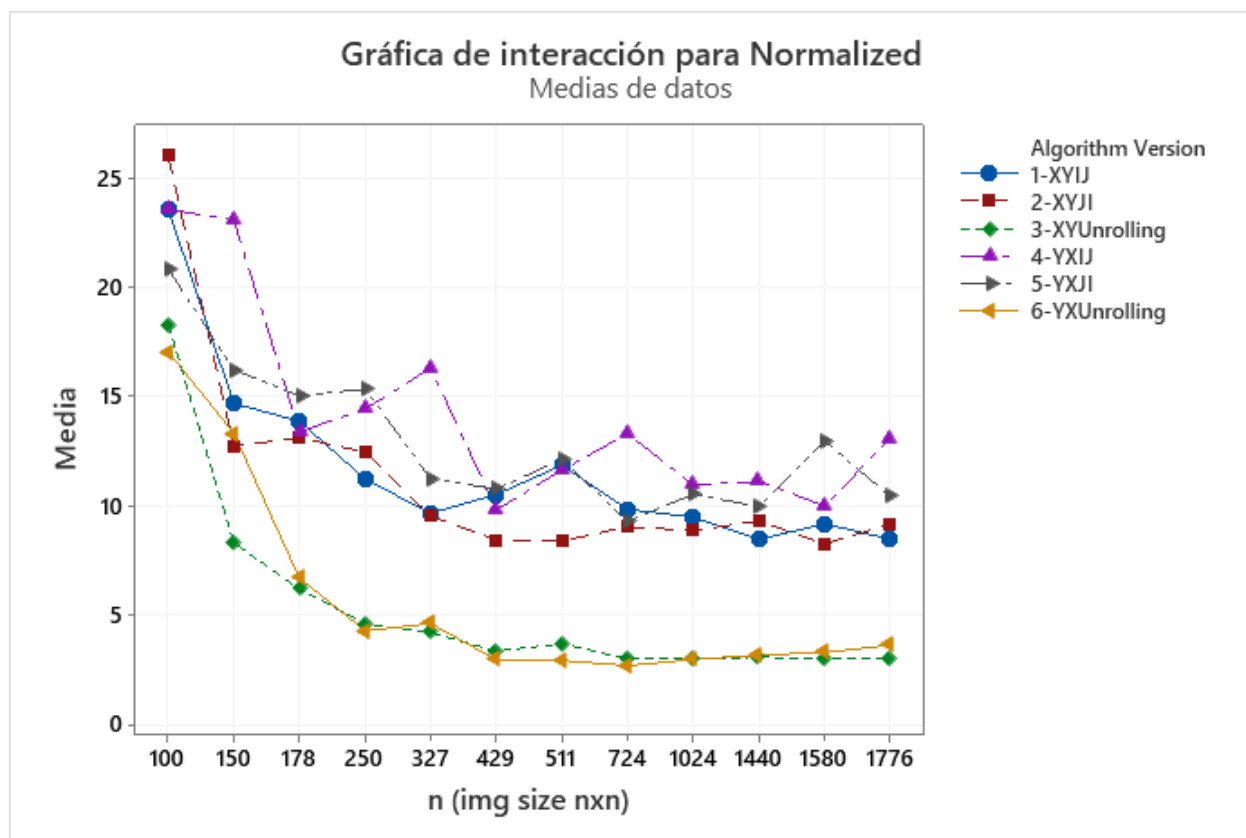
#### Análisis de Varianza

Fuente	GL	SC	Ajust. MC	Ajust. Valor F	Valor p
Algorithm Version	5	2578,0	515,603	54,00	0,000
n (img size nxn)	11	3444,5	313,139	32,80	0,000
Algorithm Version*n (img size nxn)	55	401,6	7,302	0,76	0,872
Error	144	1374,8	9,548		
Total	215	7799,0			

#### Resumen del modelo

	R-cuad.		R-cuad.	
S	R-cuad. (ajustado)		(pred)	
3,08991	82,37%	73,68%	60,34%	

Jose David Jimenez Flores  
 Cristian Camilo Lasso Hernandez  
 Daniel Alejandro Fernández Robles



Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

Cristian

### Información del factor

Factor	Tipo	Niveles	Valores
Algorithm Version	Fijo	6	1-XYIJ; 2-XYJI; 3-XYUnrolling; 4-YXIJ; 5-YXJI; 6-YXUnrolling
n (img size nxn)	Fijo	12	100; 150; 178; 250; 327; 429; 511; 1440; 1776; 2023; 2255; 2508

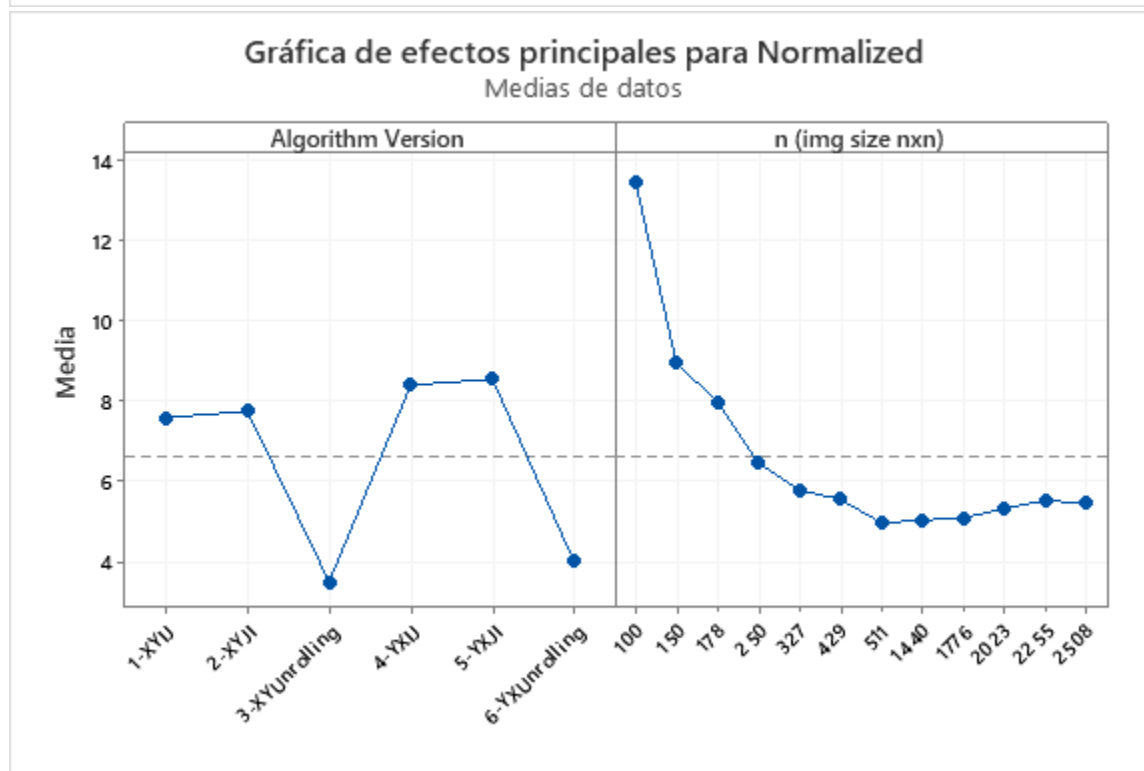
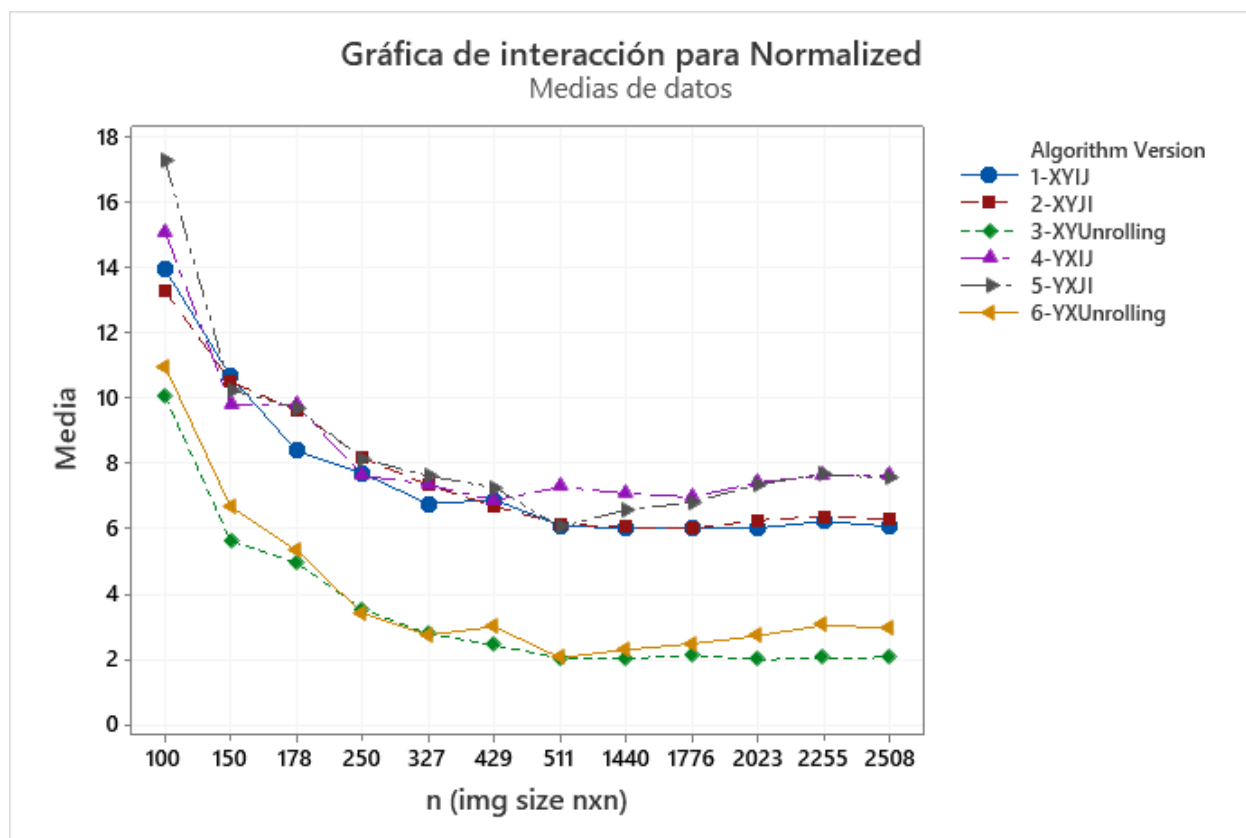
### Análisis de Varianza

Fuente	GL	SC	Ajust.	MC	Ajust.	Valor F	Valor p
Algorithm Version	5	924,67	184,934	264,17	0,000		
n (img size nxn)	11	1221,63	111,057	158,64	0,000		
Algorithm Version*n (img size nxn)	55	38,69	0,703	1,00	0,478		
Error	144	100,81	0,700				
Total	215	2285,80					

### Resumen del modelo

	S	R-cuad.	R-cuad.	R-cuad.
		(ajustado)	(pred)	
0,836696	95,59%	93,42%	90,08%	

Jose David Jimenez Flores  
 Cristian Camilo Lasso Hernandez  
 Daniel Alejandro Fernández Robles





Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

Jose

### Factor Information

Factor	Type	Levels	Values
n (img size nxn)	Fixed	12	100, 150, 178, 250, 327, 429, 511, 2023, 2508, 2900, 3150, 3550
Algorithm Version	Fixed	6	1-XYIJ, 2-XYJI, 3-XYUnrolling, 4-YXIJ, 5-YXJI, 6-YXUnrolling

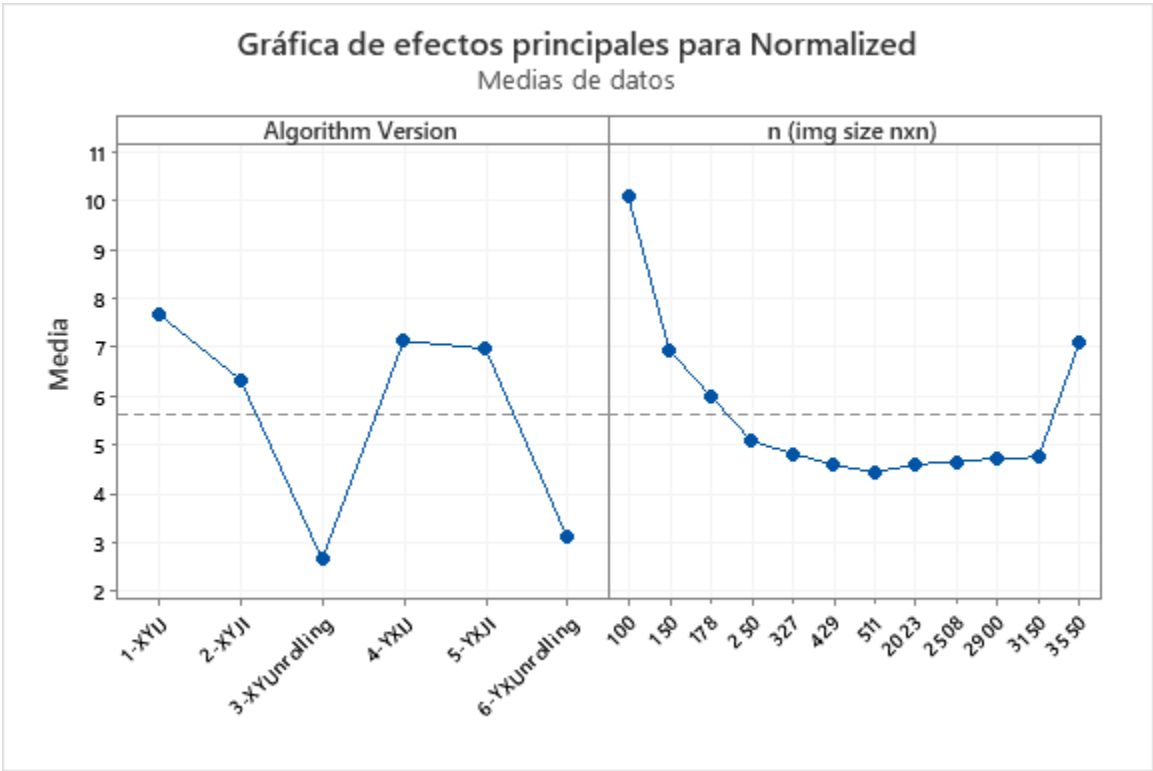
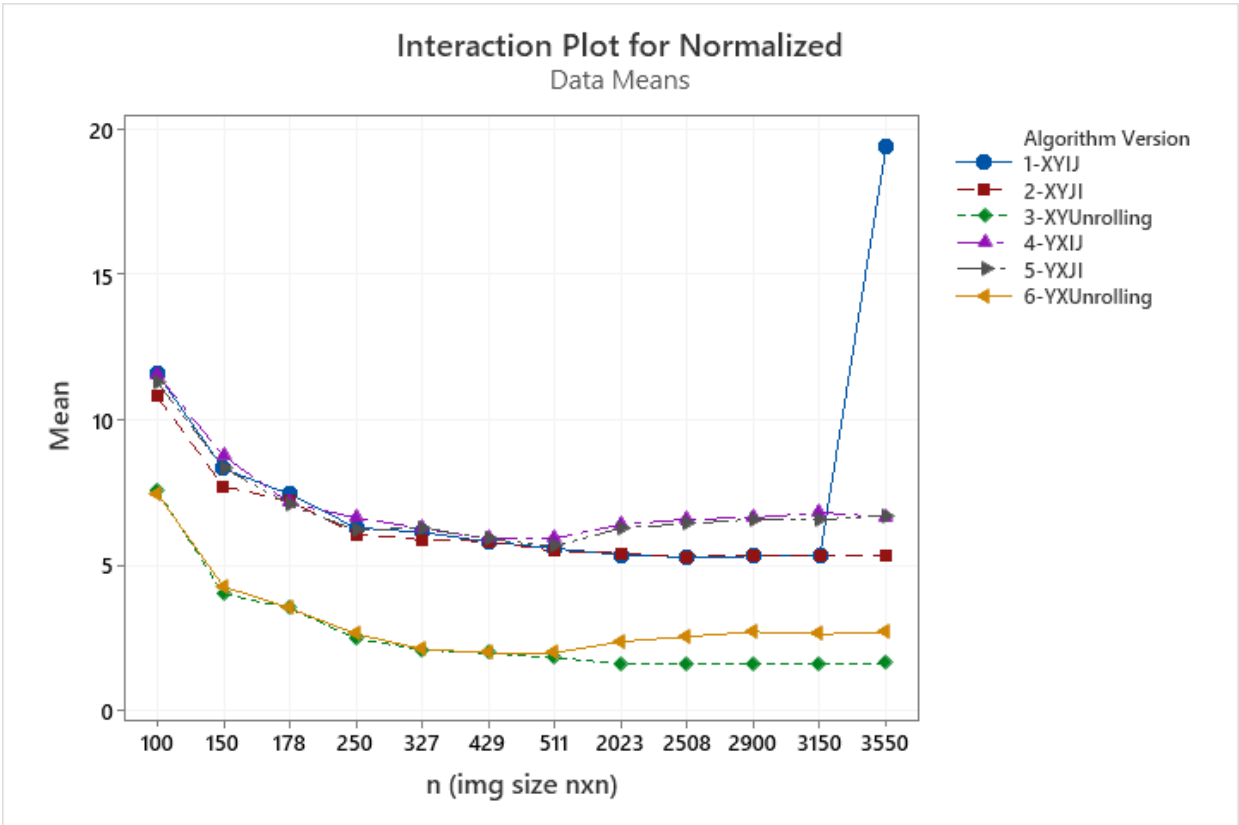
### Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
n (img size nxn)	11	554.4	50.405	6.04	0.000
Algorithm Version	5	861.8	172.368	20.66	0.000
n (img size nxn)*Algorithm Version	55	433.3	7.878	0.94	0.587
Error	144	1201.4	8.343		
Total	215	3051.0			

### Resumen del modelo

	S	R-cuad.	R-cuad.
		(ajustado)	(pred)
	2,88847	60,62%	41,21%
			11,40%

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles



Jose David Jimenez Flores  
 Cristian Camilo Lasso Hernandez  
 Daniel Alejandro Fernández Robles

## Análisis de la varianza general

### Información del factor

Factor	Tipo	Niveles	Valores
Computador	Fijo	3	1-JD; 2-CL; 3-DF
Algorithm Version	Fijo	6	1-XYIJ; 2-XYJI; 3-XYUnrolling; 4-YXIJ; 5-YXJI; 6-YXUnrolling
n (img size nxn)	Fijo	12	1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12

### Análisis de Varianza

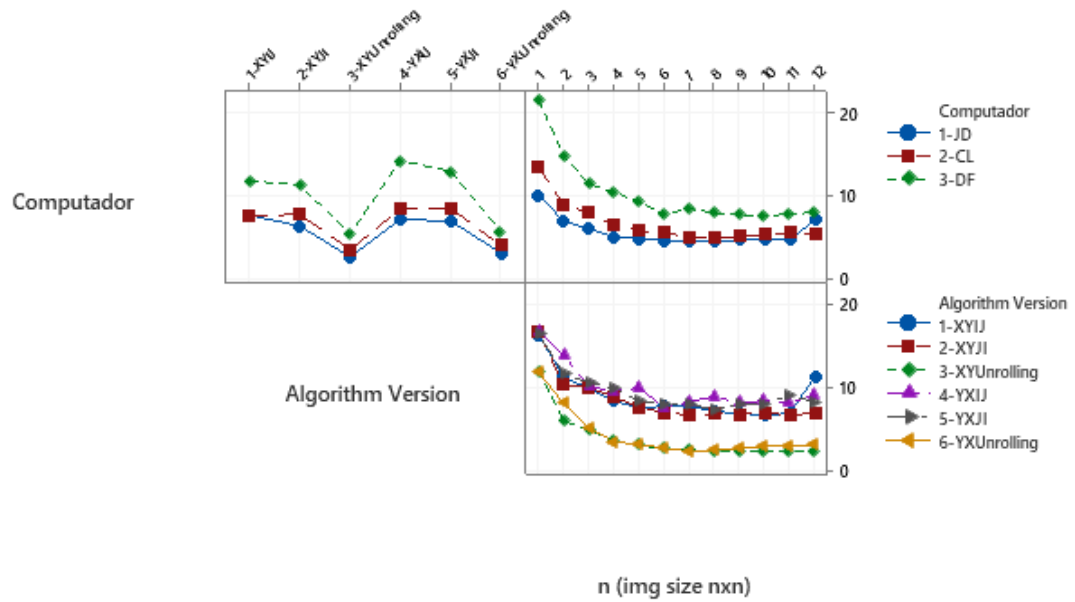
Fuente	GL	SC	Ajust.	MC	Ajust.	Valor F	Valor p
Computador	2	2502,9		1251,44		204,31	0,000
Algorithm Version	5	3992,5		798,49		130,36	0,000
n (img size nxn)	11	4409,1		400,83		65,44	0,000
Computador*Algorithm Version	10	372,1		37,21		6,07	0,000
Computador*n (img size nxn)	22	811,5		36,89		6,02	0,000
Algorithm Version*n (img size nxn)	55	230,8		4,20		0,69	0,959
Error	542	3319,9		6,13			
Falta de ajuste	110	642,8		5,84		0,94	0,639
Error puro	432	2677,1		6,20			
Total	647	15638,7					

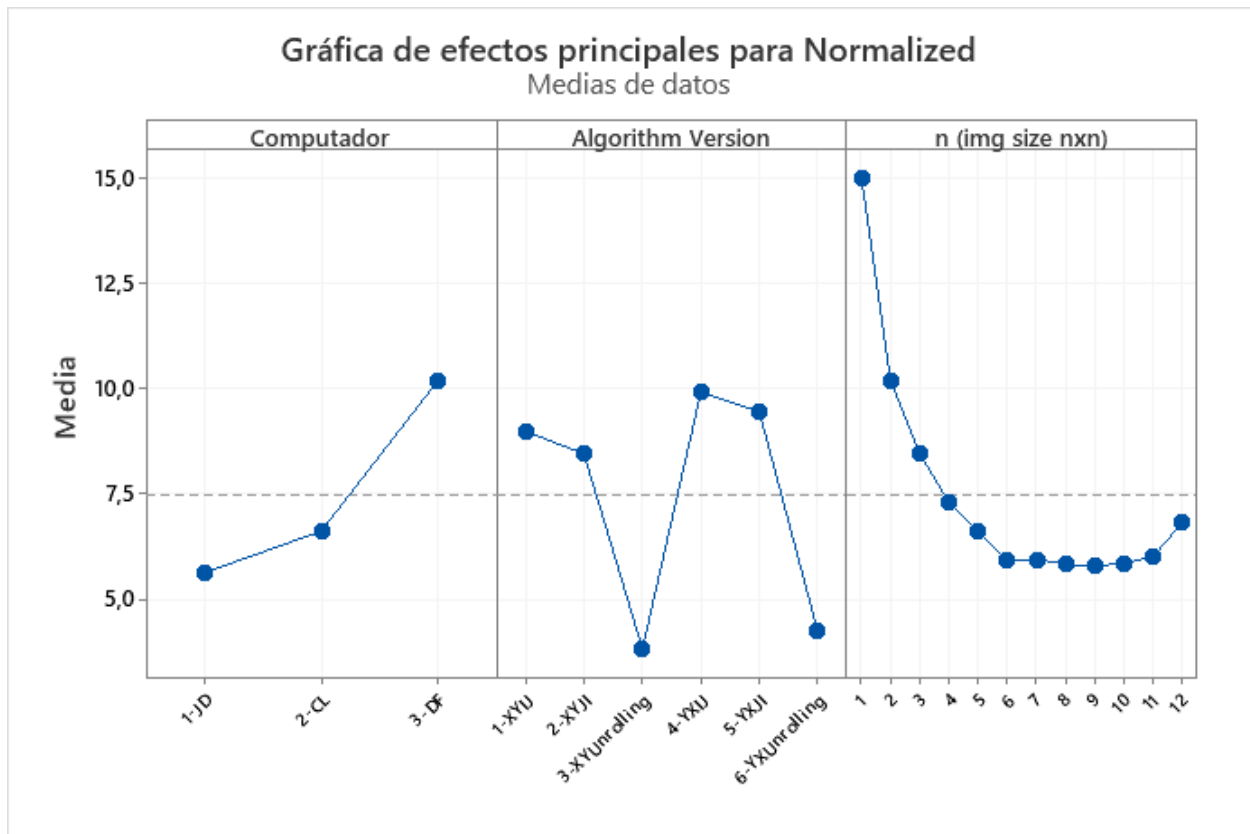
### Resumen del modelo

	R-cuad.		R-cuad.	
	S R-cuad. (ajustado)		(pred)	
	2,47493	78,77%	74,66%	69,66%

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

### Gráfica de interacción para Normalized Medias de datos





## Análisis de resultados

Analizando los resultados estadísticamente, al comparar los tres ANOVA obtenidos en cada máquina, nos damos cuenta de que son consecuentes, ya que los factores que influyen en los tiempos de ejecución normalizados son los mismos a pesar de que existen diferencias significativas entre los recursos de hardware disponibles en los computadores (Incluso las gráficas de interacción y de efectos principales son idénticas). Esta similitud en los resultados nos sirve como punto de referencia para afirmar que el experimento se llevó a cabo de manera estricta, siguiendo los pasos definidos de ejecución en los tres computadores. También se coincide en la combinación de factores que no implica una diferencia en la variable de respuesta (La versión del algoritmo con el tamaño de la imagen). Esto tiene sentido desde el punto de vista estadístico porque es de esperar que el tiempo normalizado sea igual para el mismo algoritmo con el mismo tamaño de imagen, independientemente de la repetición que se elija observar. Lo que se aprecia en las gráficas de interacción es que el tiempo normalizado inicia en un pico, pero al aumentar el tamaño de la imagen este va oscilando alrededor de un valor, por lo que podemos inferir que este es el valor de tendencia del tiempo normalizado para ese algoritmo. Tiene sentido hablar de este valor de tendencia porque al tratarse de la misma instrucción más interna, es razonable pensar que el tiempo que tarde sea idéntico independientemente del tamaño de la entrada. Analizando el modelo general que incluye como nuevo factor los computadores, nos damos cuenta de que todos los factores analizados a

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

excepción de la fijación entre el tamaño de la imagen y la versión del algoritmo influyen en que los tiempos de ejecución sean diferentes. Cada factor afecta de mayor o menor manera y esto lo podemos observar en el valor  $F$  de cada uno, entre más grande sea el valor más afecta. Podemos afirmar que tenemos un buen modelo estadístico, puesto que el indicador de correlación  $R^2$  es de 79% y para tener tantos factores de ruido es muy bueno.

Anteriormente se mencionó el pico con el que inician las gráficas de interacción para tamaños de  $n$  pequeños. Desde el punto de vista del curso, esto se puede justificar porque aunque la cantidad de misses va a ser menor que con imágenes más grandes, porcentualmente representa uno de los mayores miss rates. Esto lo podemos apreciar si llevamos el análisis al extremo; Si tenemos una matriz de  $1 \times 1$  tendremos un miss rate de 1.0. Si la matriz es de  $2 \times 2$  tendremos un miss rate de 0.25. De ahí en adelante es fácil notar cómo este miss rate se va disminuyendo hasta que en algún momento va a oscilar alrededor de un valor constante. También podemos observar que, si quitamos las versiones de unrolling de la gráfica de efectos principales, las otras cuatro versiones están alrededor de un valor cercano de tiempo normalizado, lo que da una idea de lo buenos que son los algoritmos que sacan partido de los beneficios del desenroscado de bucles. Es notable cómo esto implica una “equivalencia” entre las cuatro versiones que no usan desenroscado de bucles pues en efecto sus promedios son similares. Esto significa que el orden de los ciclos en este algoritmo no es de afectación sobre el tiempo normalizado, ya que los accesos a 9 direcciones de memoria en cada paso (literalmente a 3 filas de la matriz en el mismo paso más interno de los ciclos) es el verdadero determinante del miss rate y contribuye a que sea más o menos igual.

Los resultados obtenidos en cada uno de los tres computadores de manera individual indican unánimemente que las mejores versiones de algoritmo son la 3 (x-y-unrolling) y la 6 (y-x-unrolling). Así mismo, las peores versiones de algoritmo en los tres computadores son la 4 (y-x-i-j) y la 5 (y-x-j-i). Esto lo sabemos por las gráficas de interacción con el tiempo normalizado. Los resultados en las tres máquinas son similares, a pesar de que los modelos lineales tienen grados de correlación ( $R^2$ ) más fuertes que otros.

## Conclusiones

Individualmente podemos concluir que en los modelos de las pruebas realizadas obtuvimos porcentajes confiables de  $R^2$ , por lo cual podemos obtener información de confianza en caso de utilizar el modelo de regresión. Sin embargo, el mayor punto de referencia es la consistencia evidente en los resultados de las tres máquinas, la cual se debe a la decisión inicial de hacer que las imágenes ocupen proporciones iguales de las cachés de nivel 3, lo que al momento del experimento nos permitió asegurar que cada procesador por mejor o peor que fuese con respecto a los otros dos, tuviese que trabajar en iguales condiciones en cuanto a porcentaje de caché. Es por ello por lo que concluimos que fue una excelente decisión, pues nuestro modelo resultó ser muy confiable y fácilmente podemos comparar los resultados entre máquinas a pesar de sus diferencias de capacidad.

Jose David Jimenez Flores  
Cristian Camilo Lasso Hernandez  
Daniel Alejandro Fernández Robles

Tanto la versión del algoritmo utilizado, como el tamaño de la imagen procesada influyen en el tiempo de ejecución del programa. Por esto rechazamos la hipótesis nula y podemos decir que al menos un par de tiempos normalizados de ejecución difiere, y adicionalmente que, independientemente del procesador esta variación de tiempo será similar, proporcionalmente hablando, pues depende del algoritmo y el tamaño de imagen más no del hardware utilizado.

Nos gustaría en el futuro cercano hacer un experimento similar a este, pero donde además de las dimensiones actuales de las imágenes tengamos algunas que superen la capacidad del nivel 3 de caché. Consideramos que esto puede ser de importancia si queremos hacer un análisis en el que debamos tener en cuenta muchos más fallos en acceso a caché, además de que, por la naturaleza del filtrado de imágenes, no sería sorprendente que alguien quisiera usar un algoritmo así con imágenes de mucha mayor resolución y peso.

## Referencias

- [1] Autor desconocido. (2018). Programmer Sought. [Online]. Disponible en: <https://www.programmersought.com/article/9604683029/>
- [2] Microsoft Corporation. (2021). .NET Documentation. [Online] Disponible en: <https://docs.microsoft.com/en-us/dotnet/api/system.datetime.ticks?view=net-5.0#remarks>
- [3] Wikipedia. (2021) Desenroscado de bucles. [Online] Disponible en: [https://es.wikipedia.org/wiki/Desenroscado\\_de\\_bucles](https://es.wikipedia.org/wiki/Desenroscado_de_bucles)
- [4] Toni Justamante Jacobs. (2021). Improveyourdrawings. [Online] Disponible en: <https://improveyourdrawings.com/wp-content/uploads/2019/02/Super-Mario-Tutorial-step-10-1024x1024.jpg>