Ingeniería Eléctrica
FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Tarea 1: Filtrado de Imágenes y Ecualización de Histograma

EL7008 – Procesamiento Avanzado de Imágenes

Nombre:	Sebastián Parra
Profesor:	Javier Ruiz del Solar
Auxiliar:	Patricio Loncomilla
Ayudante:	Francisco Leiva
	Nicolás Cruz
Fecha:	5 de octubre de 2018

Introducción

El presente informe trata sobre las actividades realizadas en el marco de la primera tarea del curso EL7008 – Procesamiento Avanzado de Imágenes, cuyo objetivo es lograr una familiarización con la librería OpenCV en el lenguaje C++ realizando operaciones básicas de procesamiento de imágenes como filtrados y ecualizaciones de histograma. Además, a través de esta tarea se espera poder comprender tanto las diferencias entre estas técnicas, reconociendo las aplicaciones de cada una, como los distintos efectos de filtrado que se pueden obtener dependiendo de tipo de máscara que se utilice.

Para lograr estos objetivos, la tarea consistió en implementar funciones de convolución y ecualización de histogramas en C++ usando OpenCV, basándose en archivos de código incompletos proporcionados por el equipo docente. Posteriormente, se procedió a utilizar la función de convolución programada para probar distintos filtros sobre un conjunto de imágenes entregadas por el equipo docente, obteniendo además las transformadas de Fourier de cada resultado para realizar un posterior análisis y comparaciones. Finalmente, se pone a prueba la función de ecualización de histogramas sobre 6 imágenes previamente entregadas, para luego comparar los distintos resultados obtenidos.

Marco Teórico

Para lograr comprender las actividades realizadas en esta tarea, es necesario definir algunos conceptos afines, los cuales serán explicados en esta sección:

Convolución

En el área de procesamiento de imágenes, se utiliza el operador de convolución para realizar filtrados. Esta operación consiste en calcular la convolución 2D entre una imagen $I(x,y)$ y una determinada máscara o *kernel* $K(x,y)$, cuya fórmula se puede apreciar en la ecuación 1. Sin embargo, este procedimiento suele simplificarse a calcular la correlación entre ambas imágenes, cuya fórmula se encuentra en la ecuación 2. Esto se debe en parte a que ambas ecuaciones producen el mismo resultado si el filtro es simétrico.

$$H(x,y) = \sum_{i=-h}^{h} \sum_{j=-h}^{h} I(x-i, y-j) K(i,j)$$

Ecuación 1: Fórmula de la convolución 2D entre una imagen $I(x,y)$ y un filtro $K(x,y)$.

$$\mathrm{H(x,y)} = \sum_{\mathrm{i=0}}^{\mathrm{M_i-1}} \sum_{\mathrm{j=0}}^{\mathrm{M_j-1}} \mathrm{I(x+i-a_i, y+j-a_j) K(i,j)}$$

Ecuación 2: Fórmula de la correlación 2D entre una imagen $I(x,y)$ y un filtro $K(x,y)$. Las variables M_i y M_J representan las dimensiones del *kernel*

Una importante característica del proceso de correlación en imágenes es que puede visualizarse sencillamente. En efecto, para programar computacionalmente esta función basta con seguir los siguientes pasos, los cuales aparecen visualizados en la figura 1:

1. Ubicar el pixel central del filtro sobre un pixel determinado de la imagen, dejando el resto de los pixeles del filtro sobre los pixeles correspondientes de la imagen.
2. Multiplicar cada coeficiente del *kernel* por los valores correspondientes en la imagen y sumar los resultados.
3. El resultado de la suma será el pixel resultante en la ubicación equivalente a donde se situó el centro del filtro.
4. Repetir este procedimiento para todos los pixeles de la imagen

Cabe destacar que como este procedimiento se realiza sobre todos los pixeles de la imagen y además para cada pixel se recorre completamente la matriz del filtro, este algoritmo toma un tiempo $O(n_i*n_j*m_i*m_j)$, donde n_i y n_j son las dimensiones de la imagen, mientras que m_i y m_j son las dimensiones del *kernel*, por lo que, si el filtro es demasiado grande, la convolución puede demorar un tiempo demasiado grande. Debido a esto, los *kernels* suelen tener dimensiones pequeñas, que la mayoría de las veces no sobrepasan los 5x5 pixeles.

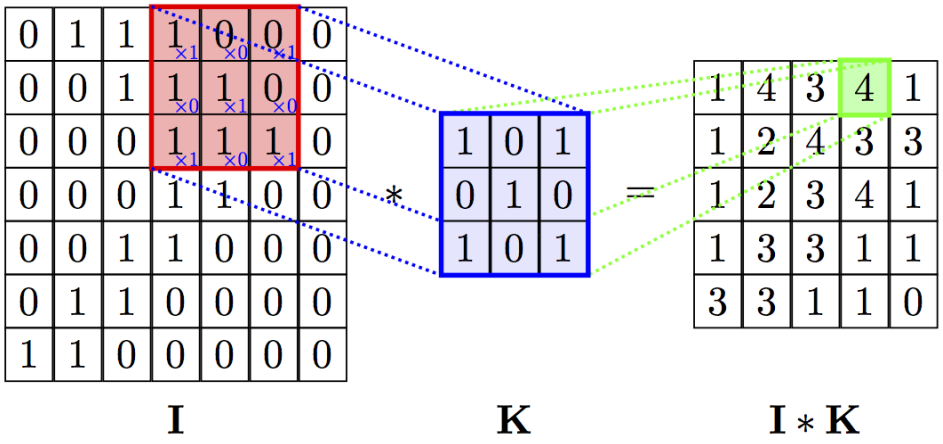

Figura 1: Correlación 2D en imágenes.

Otro aspecto importante a notar sobre la convolución es el caso de los puntos en los bordes de la imagen, puesto que en estas situaciones el *kernel* completo no cabrá completamente dentro de la imagen. Si bien no existe una solución definitiva para este problema, existen distintas formas de tratar estos casos, los más básicos consistiendo en no procesar los bordes (y por lo tanto terminar con una imagen de menor tamaño) o rellenar las celdas faltantes con ceros.

Transformada Rápida de Fourier (FFT)

Para poder comprender este concepto, primero es necesario ahondar sobre qué es la transformada discreta de Fourier (DFT por sus siglas en inglés). Esta transformada es un objeto matemático que permite transportar una señal discreta y finita en el dominio del tiempo, a una función compleja en el espacio de Fourier que indicará el comportamiento de la señal en el dominio de la frecuencia, como se puede apreciar en la figura 2. Cabe destacar que la DFT asume que la señal finita se repite infinitamente en el tiempo para obtener sus resultados.

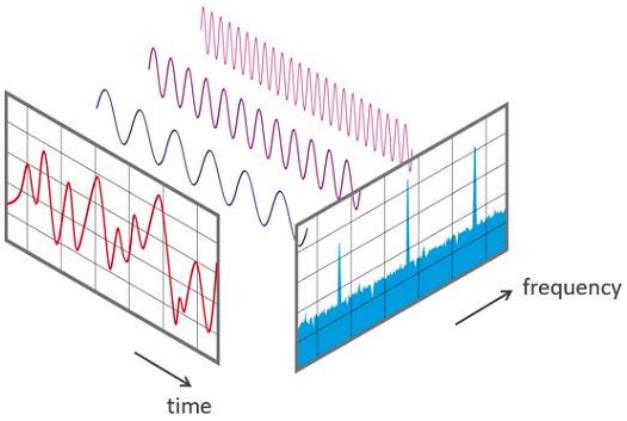

Figura 2: Visualización de una señal en el dominio del tiempo y la frecuencia.

Al igual que con la convolución, existen variantes de la DFT tanto unidimensional, cuya fórmula se muestra en la ecuación 3, como bidimensional, cuya fórmula aparece en la ecuación 4. Si bien la transformada tiene una interpretación intuitiva para señales unidimensionales, como se puede apreciar en la figura 2, los conceptos de dominio del tiempo y frecuencia pierden un poco su significado cuando se pasa al mundo de las imágenes. En estos casos, el dominio del tiempo pasa a ser representado por el dominio espacial de los pixeles en la imagen, mientras que el dominio de la frecuencia, que solía ser representado por las amplitudes y frecuencias de las sinusoides que forman la señal original, pasa a ser representado por frecuencias y amplitudes de imágenes sinusoidales, como se puede observar en la figura 3.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

Ecuación 3: Fórmula de la transformada discreta de Fourier unidimensional para una señal discreta de N muestras.

$$X_{k,l} = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{m,n} e^{2\pi i (\frac{k}{M} m + \frac{l}{N} n)}$$

Ecuación 4: Fórmula de la transformada discreta de Fourier bidimensional para una señal discreta y finita.

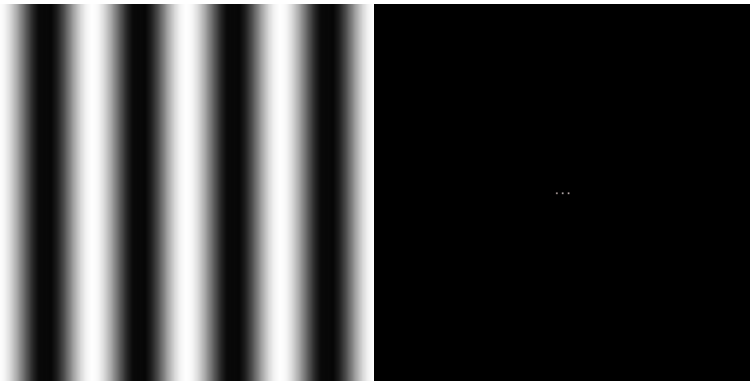

Figura 3: Visualización de una imagen sinusoidal en el dominio del espacio (izquierda) y la frecuencia (derecha).

De esta manera, la representación en el dominio de Fourier de una imagen pasa a ser la colección de frecuencias y amplitudes de imágenes sinusoidales que forman la imagen original, como se visualiza en la figura 4. En cuanto a su interpretación, se tiene que la DFT de una imagen es simétrica con respecto al origen, y que puntos que se encuentran cercanos al borde de la imagen transformada corresponden a sinusoides de alta frecuencia, mientras que puntos que se encuentren más cercanos al centro corresponden a bajas frecuencias, donde la intensidad promedio de la imagen original se ubicará justamente en el centro, como se puede apreciar en la figura 5.

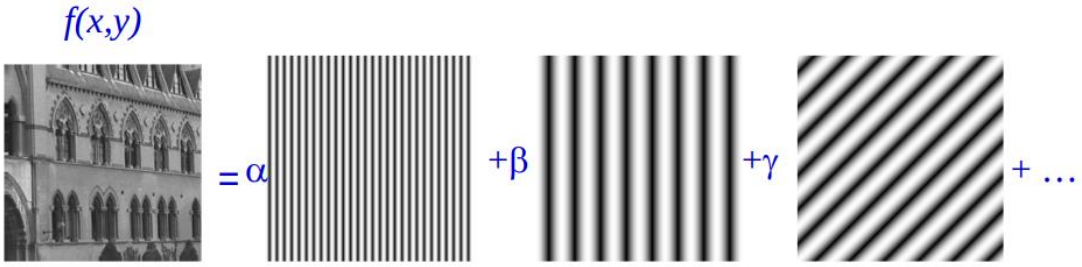

Figura 4: Descomposición de una imagen en imágenes sinusoidales de distinta frecuencia, amplitud y orientación.

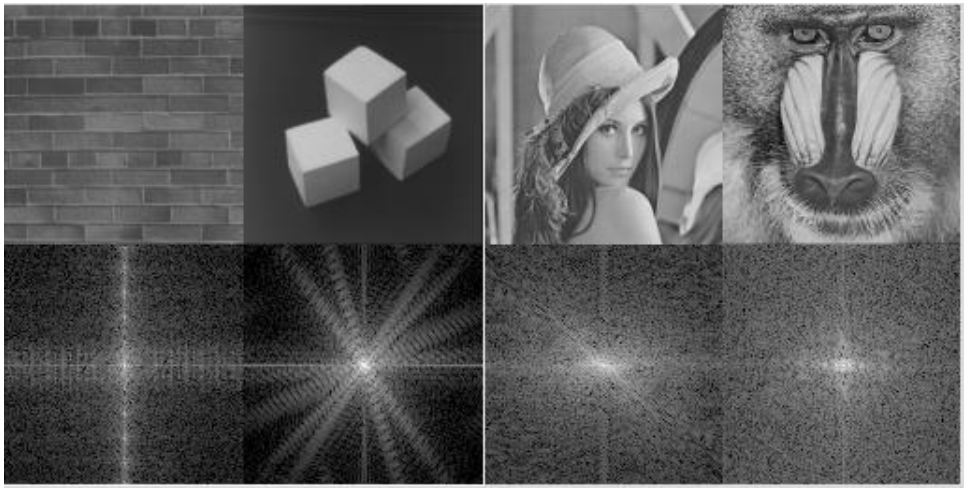

Figura 5: Cuatro ejemplos de imágenes (arriba) con sus respectivas transformadas de Fourier (abajo)

Finalmente, resta definir la transformada rápida de Fourier (FFT por sus siglas en inglés), que consiste en un algoritmo que implementa de manera más eficiente la DFT al realizar factorizaciones de algunas matrices para volverlas *sparse* (varios términos nulos, lo que permite optimizar el álgebra matricial), y así poder disminuir el tiempo de ejecución del algoritmo de $O(n^2)$ a $O(n \log n)$.

Histogramas de Imágenes

Los histogramas son una herramienta utilizada en procesamiento de imágenes que ordena los pixeles de una imagen según su intensidad, generando un gráfico que muestra la cantidad de pixeles presentes en la imagen para cada intensidad, como se muestra en la figura 6. En el caso de figuras a color, se pueden generar 3 histogramas (uno para cada color RGB), o bien transformar la imagen a escala de grises, y luego obtener el histograma. En general, a partir de los histogramas se pueden realizar varias operaciones para modificar una figura, la más importante siendo la ecualización de histogramas, que se explicará con más detalle a continuación.

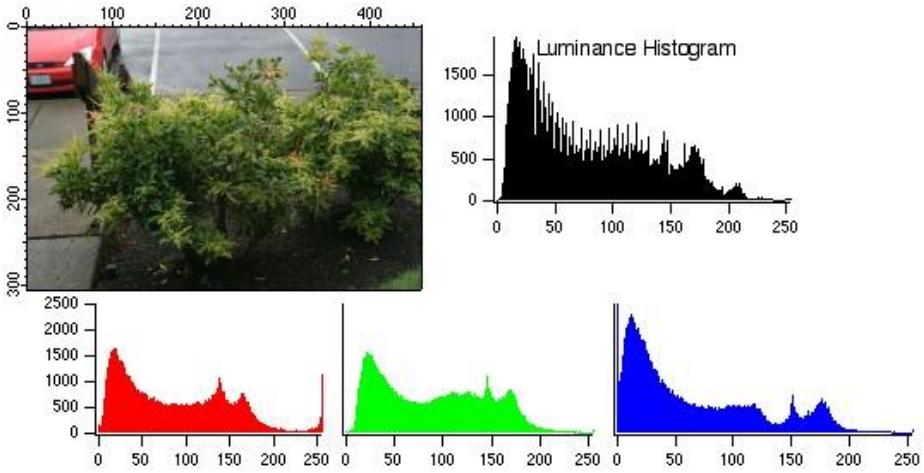

Figura 6: Histograma de una imagen a color.

Ecualización de Histogramas

La ecualización de histogramas es una técnica aplicada sobre el histograma de una imagen y tiene como objetivo ajustar las intensidades de los pixeles de la figura para mejorar su contraste. Para esto, la técnica busca obtener un histograma donde las intensidades de los pixeles estén repartidas de la manera más homogénea posible, como se puede apreciar en la figura 7.

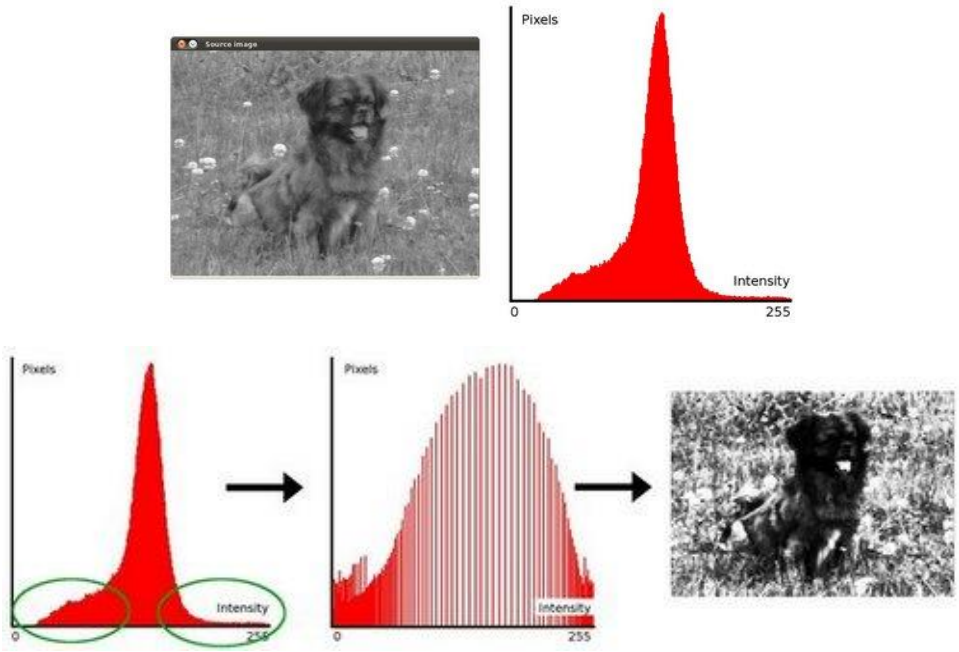

Figura 7: Imagen original y su histograma (arriba) seguida por su histograma ecualizado y la imagen resultante (abajo).

Para lograr este efecto, se plantea como objetivo lograr que la curva del histograma acumulado de la imagen, que grafica para cada intensidad de pixel la cantidad de puntos con intensidad menor o igual que la fijada, se ajuste de mejor manera a una función lineal, lo cual es posible si se aplica sobre cada intensidad de pixel presente en la imagen, la transformación de la ecuación 5, donde $h(v)$ es la nueva intensidad que tendrán los pixeles que originalmente tenían intensidad v, $cdf(v)$ corresponde al valor del histograma acumulado para la intensidad v, $cdf_{\min}$ es el mínimo valor no-nulo de la curva del histograma acumulado, M x N representan el número de pixeles en la imagen original, y L es el número de niveles de gris utilizados.

$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Ecuación 5: Transformación que permite obtener las nuevas intensidades de los pixeles ecualizados.

Desarrollo y Resultados

Programación de la Convolución

La primera actividad de esta tarea consistió en completar la función *convolucion()* en el archivo *main_conv.cpp* proporcionado por el equipo docente, de manera que el código pudiese computar la convolución entre una imagen y una máscara, y guardase el resultado en la matriz *output*, cuyas dimensiones se asumen serán iguales que las de la imagen de entrada. El código implementado funciona de la siguiente manera:

- Se recorren todas las filas y columnas de la imagen de entrada.
- Para cada combinación (fila, columna):

- Asumiendo que el punto ancla de la máscara se encuentra en su centro, se calcula la posición en la imagen original donde encajaría la esquina superior izquierda de la máscara mediante la fórmula mostrada en la figura 8, donde r y c corresponden a la combinación (fila, columna) obtenida en el paso anterior, respectivamente, y *mask.rows* representa el número de filas de la máscara, mientras que *mask.cols* representa el número de columnas. Notar que como estos valores son enteros, la división por 2 se trunca al entero más cercano.

```
//Calcular posicion (fila,columna) donde se debe comenzar a calcular la convolución (i.e. la vecindad del
//punto ancla)
int convR = r - mask.rows/2;
int convC = c - mask.cols/2;
```

Figura 8: Cálculo de la posición (fila, columna) donde encajaría la esquina superior izquierda de la máscara.

- Para cada combinación (fila, columna) de la imagen que encaja con algún valor del *kernel*:
  - Se calcula la correlación entre el pixel de la imagen y el del filtro mediante una multiplicación, y el resultado se suma a un acumulador.
  - Si la combinación (fila, columna) se encuentra fuera de la imagen, no se hace nada y se pasa a la siguiente coordenada. Este procedimiento se muestra en la figura 9.

```
for (int maskR=0; maskR < mask.rows; maskR++)
{
    for (int maskC=0; maskC < mask.cols; maskC++)
    {
        //Caso de borde: Si el pixel de la vecindad no existe, rellenar con ceros
        if(convR >= 0 && convR < input.rows && convC >= 0 && convC < input.cols)
        {
            sum += input.at<float>(convR,convC) * mask.at<float>(maskR,maskC);
        }
        convC++;
    }
    convC = c - mask.cols/2;
    convR++;
}
```

Figura 9: Cálculo de la convolución y tratamiento de casos de borde.

- Finalmente, el valor de la salida en la posición (fila, columna) obtenida en el primer paso será el valor final del acumulador.
- Se reinicia el acumulador.

Una vez programada esta función, se procedió a utilizarla para probar distintos filtros sobre dos imágenes de prueba proporcionadas por el equipo docente, cuyas versiones originales se muestran en la figura 10.

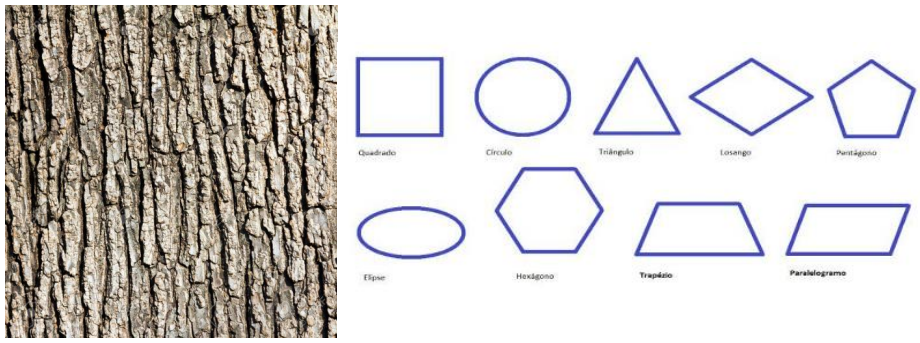

Figura 10: Figuras *corteza.png* (izquierda) y *figuras.png* (derecha) proporcionadas por el equipo docente.

Filtro Pasa Bajos Recto de 3x3

Para esta sección se trató con dos variantes de este filtro. La primera consistió en aplicar la versión 2D de la máscara con una convolución estándar, mientras que para la segunda se utilizó una versión 1D del *kernel*, aplicando convolución por filas y luego por columnas. En la figura 11 se muestran los filtros utilizados, mientras que en las figuras 12 y 13 se pueden encontrar los resultados obtenidos sobre las imágenes de prueba junto a sus respectivas FFT. Cabe destacar que las figuras fueron pasadas a escala de grises en su preprocesamiento.

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix}$$

Figura 11: Filtro pasa bajos recto 2D de 3x3 (izquierda) y filtro pasa bajos recto 1D de 1x3 (derecha).

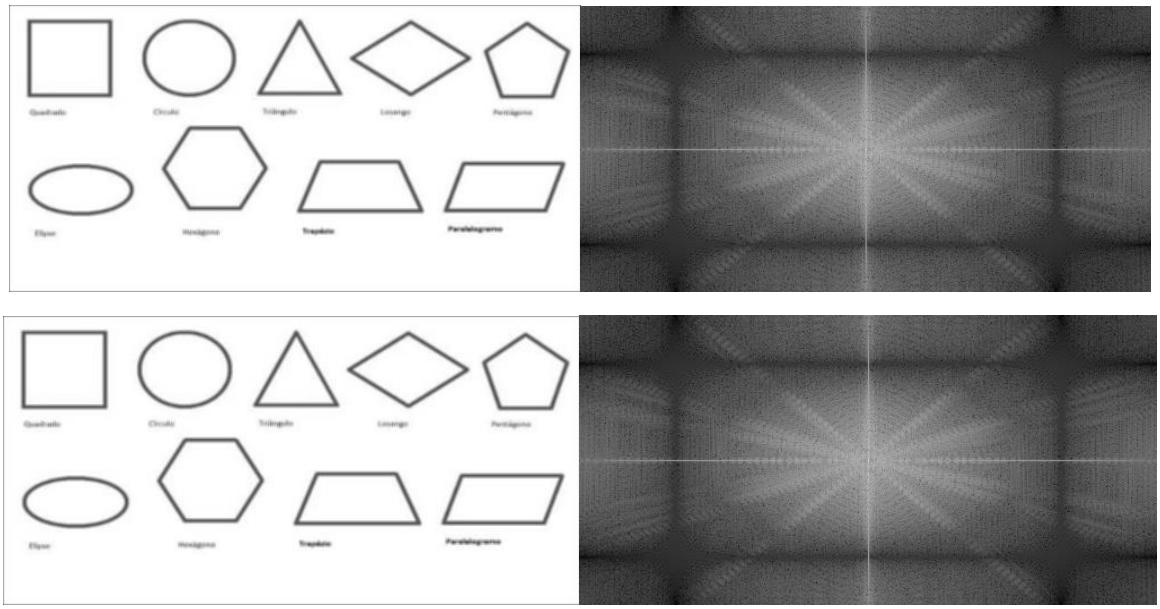

Figura 12: Arriba: Imagen resultante luego de aplicar un filtro recto 2D de 3x3 sobre la imagen *figuras.png* (izquierda) y su FFT (derecha). Abajo: Resultado de aplicar el *kernel* recto 1D por filas y columnas sobre la misma figura (izquierda) y su FFT (derecha)

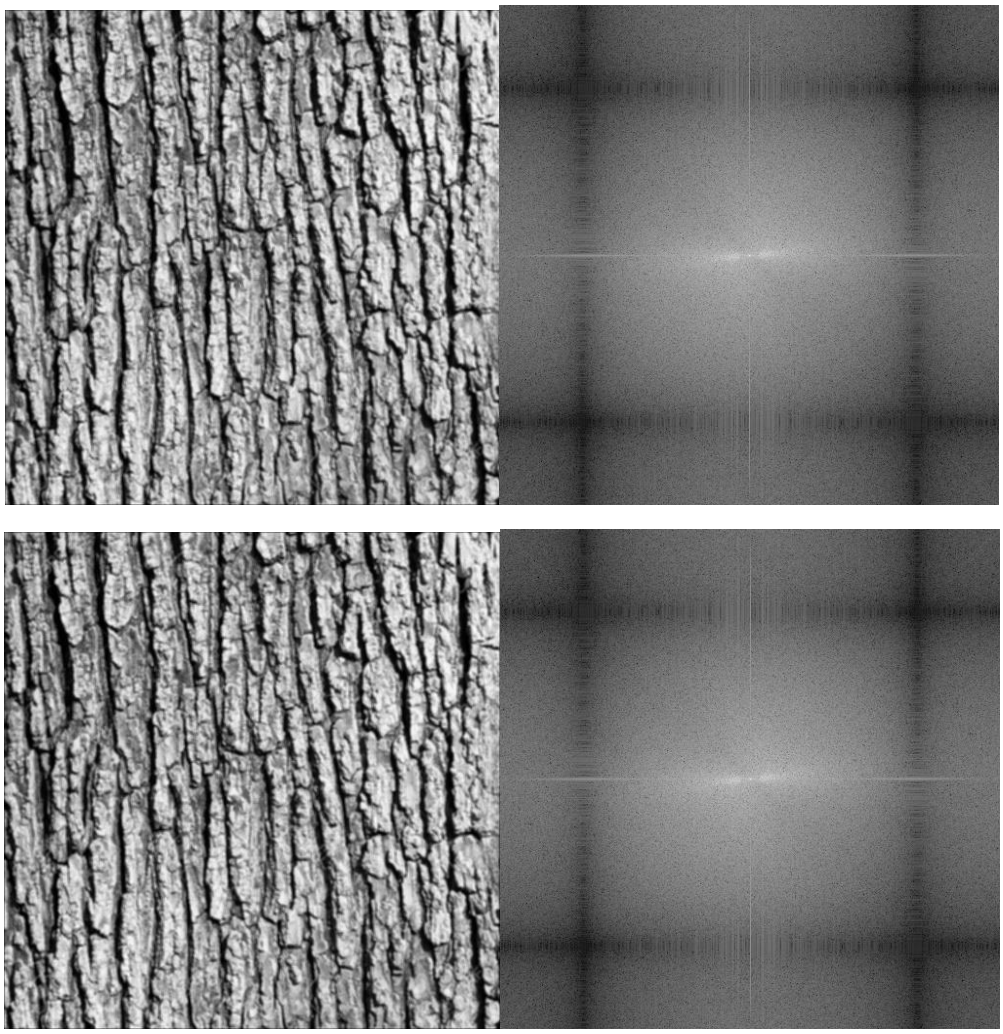

Figura 13: Arriba: Imagen resultante luego de aplicar un filtro recto 2D de 3x3 sobre la imagen *corteza.png* (izquierda) y su FFT (derecha). Abajo: Resultado de aplicar el *kernel* recto 1D por filas y columnas sobre la misma figura (izquierda) y su FFT (derecha)

Filtro Gaussiano Pasa Bajos de 5x5 Con $\sigma = 1$

Al igual que en la sección anterior, en esta parte se probaron dos filtros: Un gaussiano 2D de 5x5 estándar, y uno 1D de 1x5, pero aplicado por filas y columnas. En la figura 14 se muestran las máscaras utilizadas, mientras que en las figuras 15 y 16 se muestran los resultados obtenidos.

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figura 14: Filtro pasa bajos gaussiano 2D de 3x3 (izquierda) y filtro pasa bajos gaussiano 1D de 1x3 (derecha).

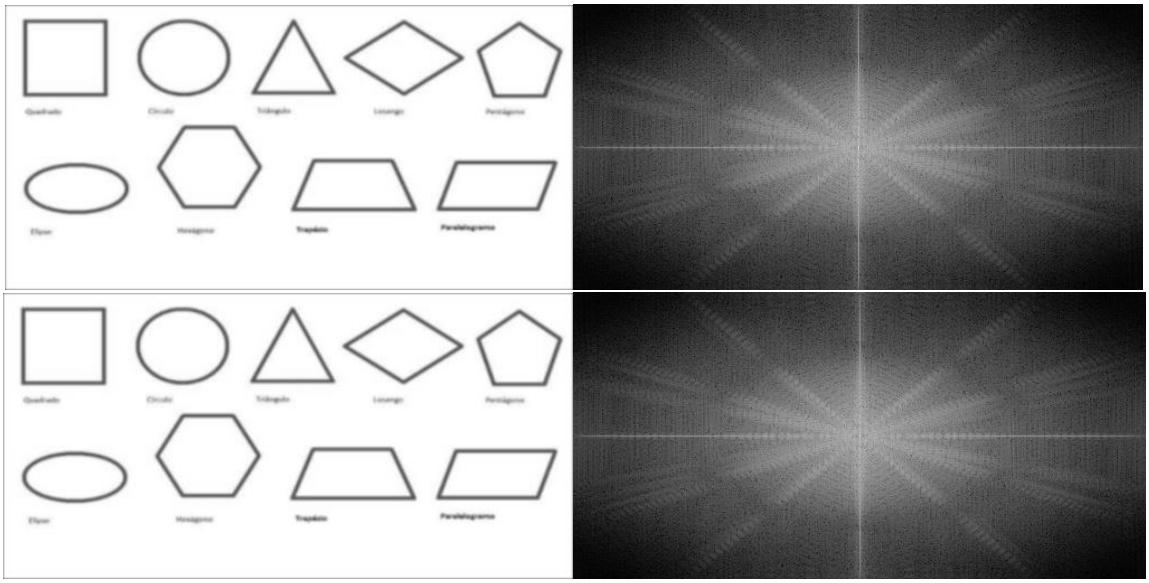

Figura 15: Arriba: Imagen resultante luego de aplicar un filtro gaussiano 2D de 5x5 sobre la imagen *figuras.png* (izquierda) y su FFT (derecha). Abajo: Resultado de aplicar el *kernel* gaussiano 1D por filas y columnas sobre la misma figura (izquierda) y su FFT (derecha).

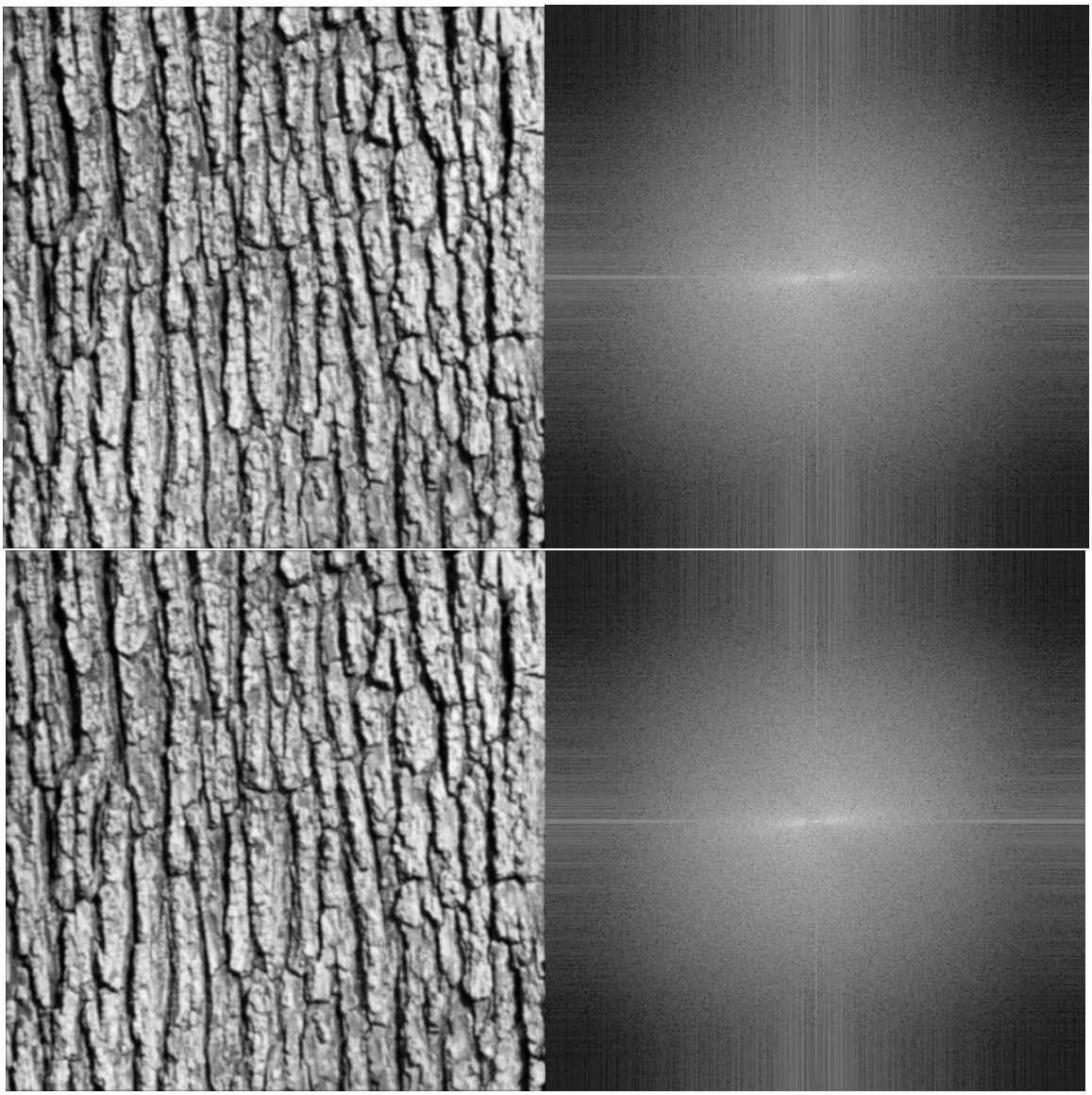

Figura 16: Arriba: Imagen resultante luego de aplicar un filtro gaussiano 2D de 5x5 sobre la imagen *figuras.png* (izquierda) y su FFT (derecha). Abajo: Resultado de aplicar el *kernel* gaussiano 1D por filas y columnas sobre la misma figura (izquierda) y su FFT (derecha).

Filtro Prewitt Vertical

En esta sección se prueba utilizando un filtro pasa altos Prewitt vertical, cuya máscara se puede apreciar en la figura 17, mientras que los resultados se pueden observar en las figuras 18 y 19.

$$\frac{1}{9}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Figura 17: Filtro pasa altos Prewitt vertical.

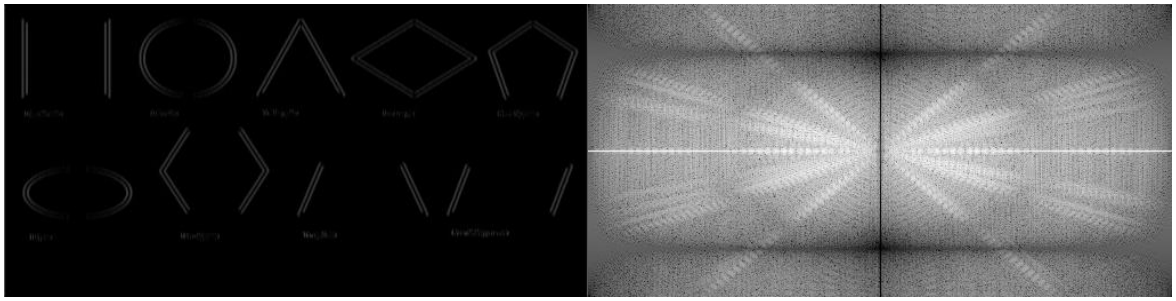

Figura 18: Imagen resultante luego de aplicar un filtro Prewitt vertical sobre la imagen *figuras.png* (izquierda) y su FFT (derecha).

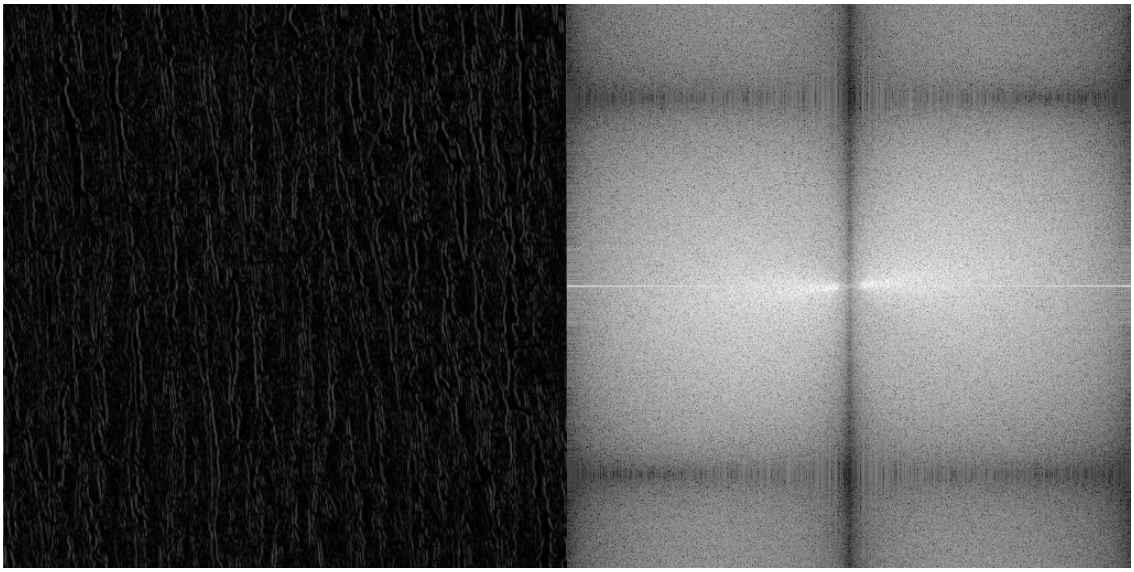

Figura 19: Imagen resultante luego de aplicar un filtro Prewitt vertical sobre la imagen *corteza.png* (izquierda) y su FFT (derecha).

Filtro Prewitt Horizontal

En esta sección se aplica la variante horizontal del filtro anterior, cuyo *kernel* puede apreciarse en la figura 20, mientras que los resultados obtenidos se exponen en las imágenes 21 y 22.

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Figura 20: Filtro pasa altos Prewitt horizontal.

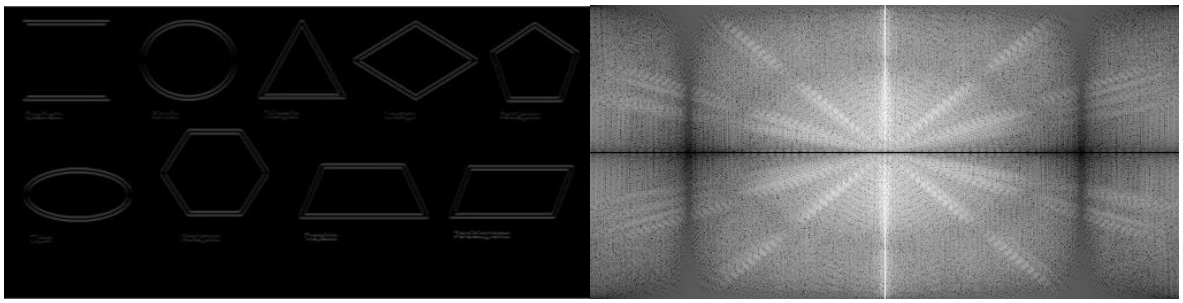

Figura 21: Imagen resultante luego de aplicar un filtro Prewitt horizontal sobre la imagen *figuras.png* (izquierda) y su FFT (derecha).

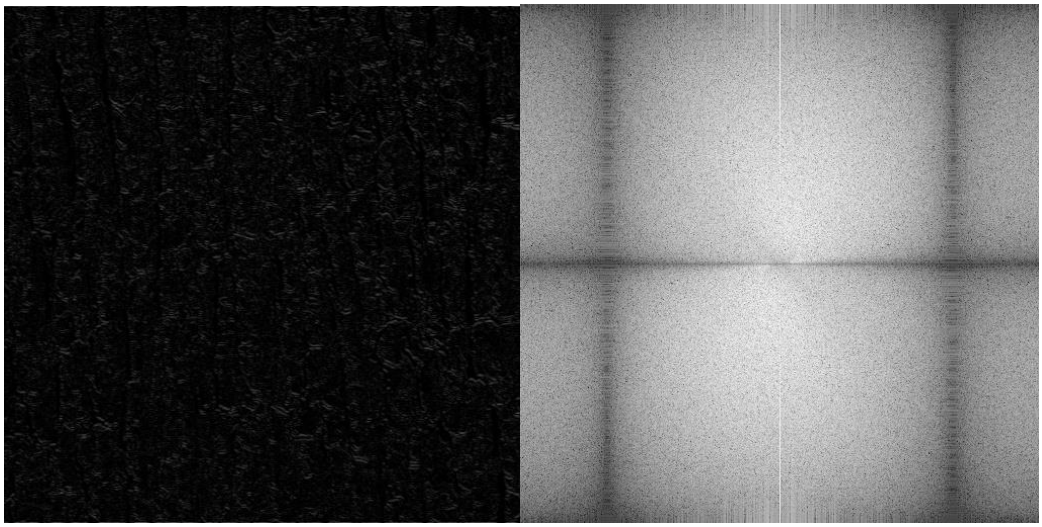

Figura 22: Imagen resultante luego de aplicar un filtro Prewitt horizontal sobre la imagen *corteza.png* (izquierda) y su FFT (derecha).

Filtro Laplaciano de 3x3

Este es otro tipo de filtro pasa altos, cuyo *kernel* puede apreciarse en la figura 23, mientras que las imágenes resultantes de aplicar esta máscara se exponen en las figuras 24 y 25.

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figura 23: Filtro Laplaciano de 3x3.

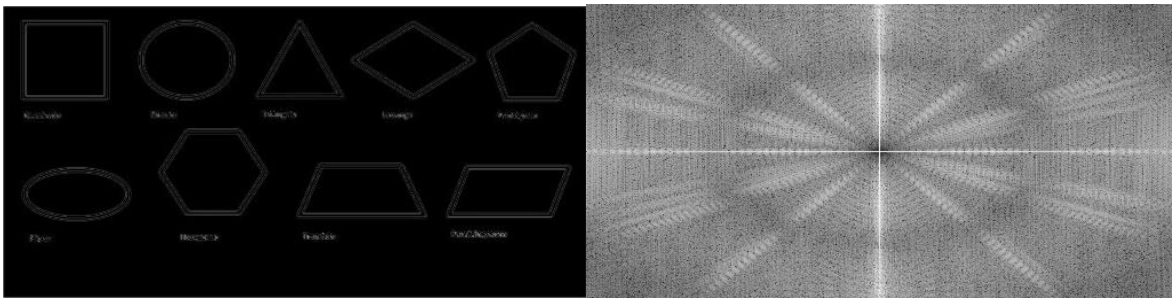

Figura 24: Imagen resultante luego de aplicar un filtro Laplaciano de 3x3 sobre la imagen *figuras.png* (izquierda) y su FFT (derecha).

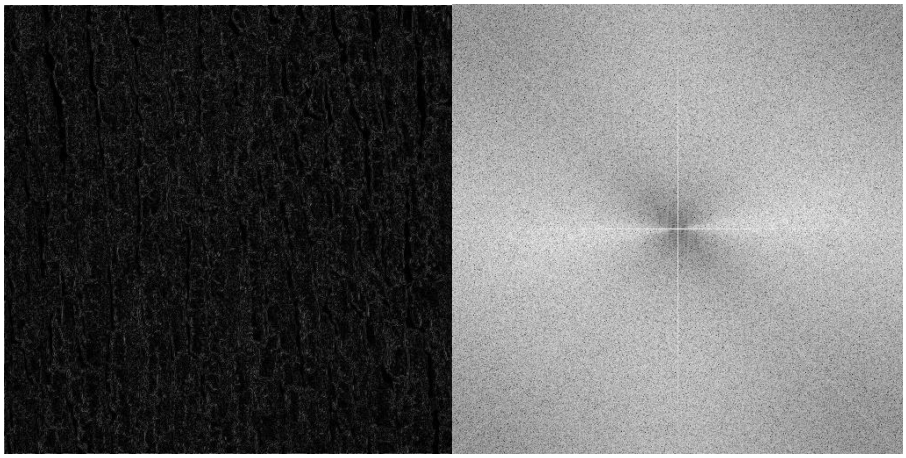

Figura 25: Imagen resultante luego de aplicar un filtro Laplaciano de 3x3 sobre la imagen *corteza.png* (izquierda) y su FFT (derecha).

Filtro Laplaciano de Gaussiana 5x5 con $\sigma = 1$

Este filtro es equivalente a pasar un filtro gaussiano de 5x5 con $\sigma = 1$sobre la imagen, y luego convolucionarla con un *kernel* Laplaciano de 5x5. Su máscara se puede apreciar en la figura 26, mientras que los resultados obtenidos luego de aplicar este filtro se exponen en las figuras 27 y 28.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figura 26: Filtro Laplaciano de Gaussiana de 5x5 con $\sigma = 1$.

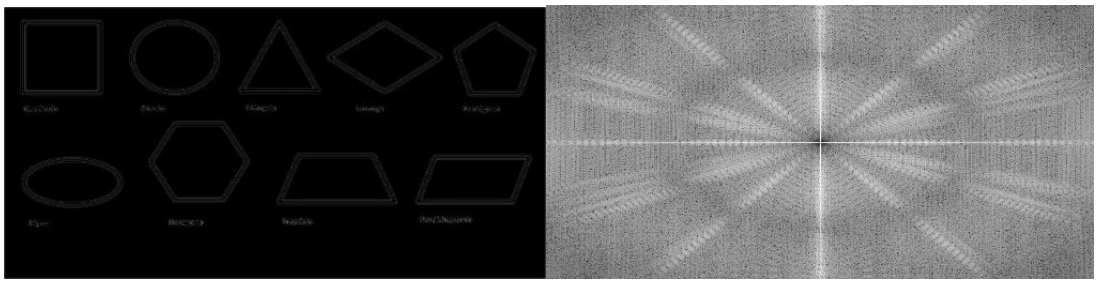

Figura 27: Imagen resultante luego de aplicar un filtro Laplaciano de Gaussiana de 5x5 con $\sigma = 1$ sobre la imagen *figuras.png* (izquierda) y su FFT (derecha).

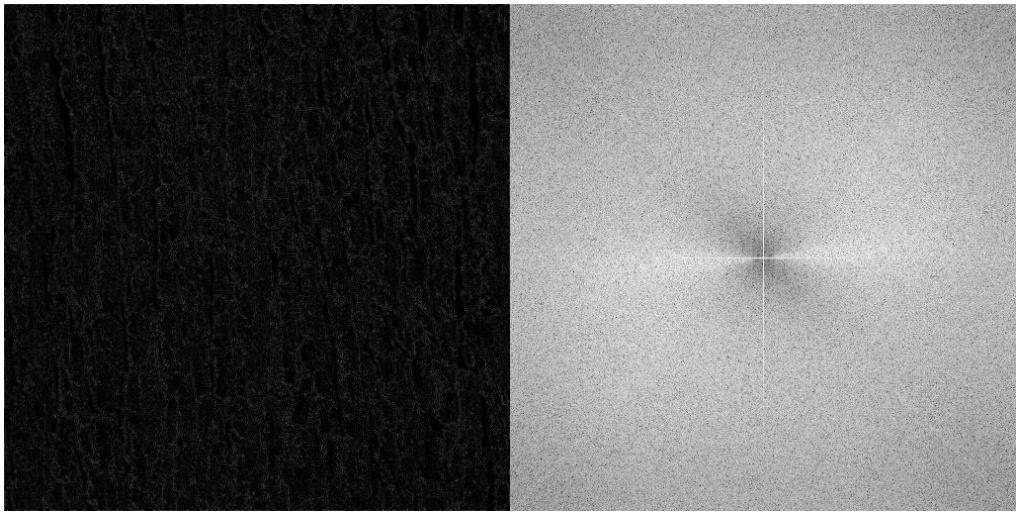

Figura 28: Imagen resultante luego de aplicar un filtro Laplaciano de Gaussiana de 5x5 con $\sigma = 1$ sobre la imagen *corteza.png* (izquierda) y su FFT (derecha).

Programación de la Ecualización de Histogramas

Para la última actividad de esta tarea se debió completar la función *ecualizar()* en el archivo *main_histeq.cpp* proporcionado por el equipo docente, de manera que el código pudiese computar la ecualización del histograma de una imagen de entrada *input*, y guardase el resultado en la matriz *output*, cuyas dimensiones se asumen serán iguales que las de la entrada. El código implementado funciona de la siguiente manera:

- Se crea un arreglo de 256 valores que será utilizado para calcular el histograma, donde el valor i del arreglo indicará la cantidad de pixeles que existen con intensidad i.
- Para cada pixel de la imagen, se determina su intensidad y se suma uno al valor correspondiente del arreglo creado en el paso anterior.
- Se crea un nuevo arreglo de 256 valores y se recorre el arreglo con el histograma para calcular el histograma acumulado con la fórmula recursiva de la ecuación 6, donde $h_{acc}(i)$ representa el histograma acumulado para la intensidad i, y $h(i)$ representa el histograma para la intensidad i. En este paso también se guarda el valor mínimo del histograma acumulado distinto de cero.

$$h_{acc}(0) = h(0)$$
$$h_{acc}(i) = h(i) + h_{acc}(i-1) \forall i > 0$$

Ecuación 6: Fórmula recursiva para obtener el histograma acumulado de una imagen

- Se vuelven a recorrer los pixeles de la imagen de entrada, esta vez aplicando a cada pixel la ecuación 5 y guardando el resultado en la posición correspondiente de la imagen de salida.

Luego de programar esta función, se procedió a probarla sobre 6 imágenes proporcionadas por el equipo docente, las cuales se muestran en la figura 29.

Figura 29: Fotos proporcionadas para la actividad de ecualización de histogramas. De izquierda a derecha y de arriba abajo: *agua.png*, *casa.png*, *constr.png*, *corteza.png*, *mala_ilum.jpg*, y *termal.png*.

A continuación, en la figura 30 se presentan cara a cara las imágenes originales versus su versión ecualizada, a modo de comparación. Para el caso de las imágenes a color, también se muestra su versión en escala de grises sin ecualizar

A)

B)

C)

D)

E)

F)

Figura 30: Resultados obtenidos luego de aplicar el ecualizador de histogramas sobre las imágenes A) *agua.png*, B) *casa.png* (con la entrada en escala de grises mostrada en el centro), C) *constr.png*, D) *corteza.png* (con la entrada en escala de grises mostrada en el centro), E) *mala_ilum.jpg*, y F) *termal.png*.

Análisis de los Resultados

Convolución 2D versus convolución 1D por filas y columnas

Si se observan los resultados obtenidos en las figuras 12, 13, 15, y 16, se puede notar que no pareciera existir ninguna diferencia evidente entre las imágenes obtenidas aplicando la versión 2D de un filtro, o su versión 1D por filas y columnas. Estos resultados concuerdan con lo esperado, puesto que la teoría indica que, si un *kernel* K de dimensiones M x N puede ser separado en dos vectores, uno de M x 1 y otro de 1 x N, entonces la convolución de una imagen I por el filtro K puede ser escrita como se muestra en la ecuación 7, y como se puede apreciar al ver las figuras 11 y 14, las máscaras utilizadas cumplían esta propiedad. Esta propiedad tiene como ventaja que permite reducir el tiempo de cálculo de una convolución de $O(n^2)$ a $O(2n) = O(n)$.

$$I[x,y] * K[m,n] = I[x,y] * (k_1[m] \cdot k_2[n]) = k_1[m] * (k_2[n] * I[x,y])$$

Ecuación 7: Propiedad de separabilidad de la convolución.

Convolución con filtros pasa bajos

Volviendo a estudiar los resultados obtenidos en las figuras 12, 13, 15, y 16, se puede apreciar en las FFT que el filtro recto no logra realizar un filtrado ideal, puesto que no atenúa correctamente las componentes de alta frecuencia de la imagen original, apareciendo sólo un par de rectas que atenúan un grupo específico de frecuencias, pero que no tienen mucho efecto sobre frecuencias que estén sobre o por debajo de este rango. En cambio, si se observan las FFT obtenidas en las figuras 15 y 16, se puede notar que el filtro Gaussiano presenta un comportamiento más cercano al ideal, que sería cortar completamente las frecuencias sobre un cierto valor umbral, ya que entrega una FFT cuya intensidad disminuye mientras más cercano esté el pixel del borde de la imagen, lo que equivale a las frecuencias más altas. Estos experimentos corroboran los contenidos vistos en clases, donde se afirmó que un filtro Gaussiano es la mejor aproximación que se puede tener de un filtro ideal.

Convolución con filtros pasa altos

Si se observan las imágenes de las figuras 18, 19, 21, y 22, se puede notar cómo es que los filtros de Prewitt logran calcular borden horizontales o verticales según la variante que se ocupe, especialmente en el caso de la imagen *figuras.png*, cuyos bordes están fuertemente marcados en la imagen original, lo que se traduce en imágenes claras en los resultados. Estos resultados concuerdan con lo esperado teóricamente, puesto que la forma de los *kernels* mostrados en las figuras 17 y 20 muestran que este filtro calcula una aproximación del gradiente de la imagen en los ejes (x, y), dependiendo de la variante que se ocupe.

En cuanto al impacto de los filtros Laplacianos, en las figuras 24 y 25 se puede apreciar que esta máscara permite hacer una detección completa de bordes en todas las direcciones, a diferencia de los filtros de Prewitt, lo cual se hace aún más evidente en el caso del archivo *figuras.png*. Sin embargo, también se puede notar que los resultados son mucho más ruidosos para los bordes que no son rectos, lo cual se intensifica aún más en la imagen *corteza.png*, que presenta tantos bordes irregulares que no se puede reconocer con claridad la imagen filtrada. Esto corrobora lo visto en clases, donde se mencionó que los bordes de las imágenes tienen como característica presentar segunda derivada nula, por lo que un filtro laplaciano sería adecuado para realizar detección de bordes, aunque tienen como desventaja ser muy sensibles al ruido.

Finalmente, si se observan las figuras 27 y 28 se puede notar que, al igual que con el filtro Laplaciano, el *kernel* Laplaciano de Gaussiana también logra hacer detección de bordes con una sola operación de convolución, aunque en este caso se obtienen imágenes más claras y menos ruidosas, lo cual se puede apreciar de mayor manera en el caso de *corteza.png*, donde ahora si se pueden reconocer las características principales de la imagen, aunque de todas maneras los bordes se encuentran menos definidos que con los filtros de Prewitt. Esto se explica debido a que este filtro es equivalente a aplicar un filtro de gaussiana seguido de un filtro laplaciano, lo cual se traduce en pasar la imagen por un proceso de suavizado previo a la detección de bordes, así logrando reducir sus componentes ruidosas.

Ecualización de histogramas

Observando las imágenes de la figura 30, se puede notar que el proceso de ecualización de histogramas puede presentar resultados muy variados, que oscilan entre mejoras considerables a la visibilidad de la imagen, como es el caso del archivo *mala_ilum.jpg*, o imágenes poco claras e incluso más ruidosas que la original, como sucedió con el archivo *casa.png*. A partir de esto, se puede concluir que la decisión de si aplicar o no esta herramienta debe ser cuidadosamente estudiada con anterioridad, puesto que el resultado dependerá fuertemente de las características específicas de la imagen a procesar. Por ejemplo,

si el histograma original se encuentra compactado en un rango de intensidades relativamente centrado, la ecualización será de gran ayuda para mejorar su visualización. En cambio, si los problemas de contraste de la imagen no son generalizados, podría ocurrir un fenómeno similar al que se obtuvo con el archivo *agua.png*, donde debido a la ecualización, el área de la figura donde se encontraba el sol aparece más brillante que en la original, provocando una mayor pérdida de información en los pixeles cercanos al sol.

Conclusiones

A partir de las actividades realizadas en esta tarea, se puede concluir que las operaciones de ecualización de histogramas y filtrado por convolución constituyen un importante conjunto de herramientas básicas en el área de procesamiento de imágenes que pueden ser utilizadas para realizar una gran cantidad de operaciones más complejas, especialmente para el caso de la convolución. Además, se pudo comprobar experimentalmente la funcionalidad de un filtro pasa bajos como eliminador de ruido, y la de un filtro pasa altos como detector de bordes, reconociendo la existencia de distintos *kernels* que se pueden utilizar, cada uno con sus ventajas y desventajas.

En cuanto a las principales dificultades encontradas, el mayor obstáculo correspondió al manejo de la librería C++, puesto que no se tenía experiencia previa con el lenguaje y éste presenta algunas complejidades sobre otros lenguajes de programación al no entregar retroalimentación detallada en el caso de encontrarse con un error, por lo que la detección de errores en el código en un principio fue una tarea compleja.

Finalmente, y como se mencionó en la sección de análisis, se concluye para todas las actividades que lograron exitosamente corroborar empíricamente los conocimientos teóricos obtenidos en clases.