UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio 3: Procesamiento de imágenes

Nombre: Juan Arredondo
Alberto Rodríguez
Curso: Redes de Computadores
Sección 0-L-1
Profesor: Carlos González
Ayudante: Nicole Reyes

17 de Mayo de 2019

Tabla de contenidos

Índice de figuras

1. Introducción

Las redes computacionales han ido evolucionando a lo largo del tiempo, creando aplicaciones en diferentes áreas de la tecnología. Sin embargo, todo esto nace de la base teórica que se tiene de las redes, más en específico, enfocado en el desarrollo de la capa física, donde aparecen procedimientos matemáticos que nos ayudan a aplicar los conocimientos adquiridos en cátedra. Uno de esas herramientas matemáticas es la convolución.

La convolución de señales se define como un operador matemático que transforma dos funciones f y g en una tercera función que en cierto sentido representa la magnitud en la que se superponen f y una versión trasladada e invertida de g. Analizándolo desde una mirada en 2D, este tiene aplicaciones en las imágenes, donde la función f sería la entrada y la función g seria un kernel (filtro), generando la imágen de salida filtrada.

De ésta manera, el siguiente laboratorio nos propone el reto de aplicar un filtro de manera manual a una imágen y analizar los tipos de salidas generadas, tales como su variación de frecuencias, la transformación de las imágenes y los resultados que se dan si se les aplica la transformada de fourier. Todos los pasos para la creación del programa serán explicados a cabalidad tales como su razonamiento lógico y las herramientas que se utilizaron en el software.

Finalmente se concluirán los resultados comparándolos con los fenómenos teóricos de la convolución.

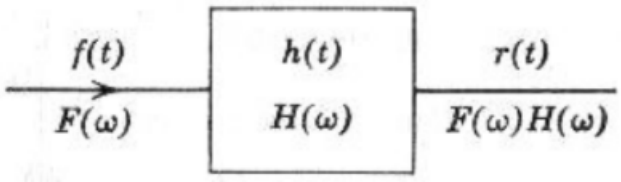

Figura 1: Ilustración de como es la convolución en un sistema.

2. Marco teórico

En la elaboración del laboratorio número tres se utilizó una variedad de herramientas tanto teóricas como tecnológicas para el desarrollo de este, algunas de estas son:

2.1. Convolución en la teoría

Intuitivamente podemos mirar a la convolución de dos funciones **f(x)** y **g(z)** como la función resultante que aparece después de efectuar los siguientes pasos:

1. Girar respecto del origen los valores de una de ellas, es decir **g(z)= g(-z)** para todo z desde -∞ a ∞.

2. Ir trasladando la función girada sobre la otra f(z)g(x-z)

3. En cada punto **x** calculamos el valor que resulta de sumar los productos obtenidos de multiplicar para todos los **z** los correspondiente valores de las funciones **f(z) y g(x-z)**. En esencia estamos calculando para cada valor de x una especie de valor ponderado de una de las funciones **f(x)** con los valores de la otra **g(x)**.

En el caso de que el área encerrada por la curva de **g(x)** fuese igual **1** entonces estaríamos calculando para **x** una media ponderada. Matemáticamente la expresión para esta operación es.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(n)g(t-n)dn \tag{1}$$

2.2. Operadores de convolución

Aplicando distintos operadores de convolución es posible obtener alguno de estos

2.2.1. Operadores de Suavizado

Al realizar una convolución con este operador se obtiene una imagen más difuminada, reducir los contrastes abruptos de la imagen, algunas veces se usan en la restauración y mejora de las imágenes.

Por lo general el largo y ancho de la matriz son impares y existe un píxel central. Existe una media ponderada, donde los pesos toman la forma de la campana de gauss. Las filas del triángulo de Pascal forman discretizaciones de la campana de Gauss.

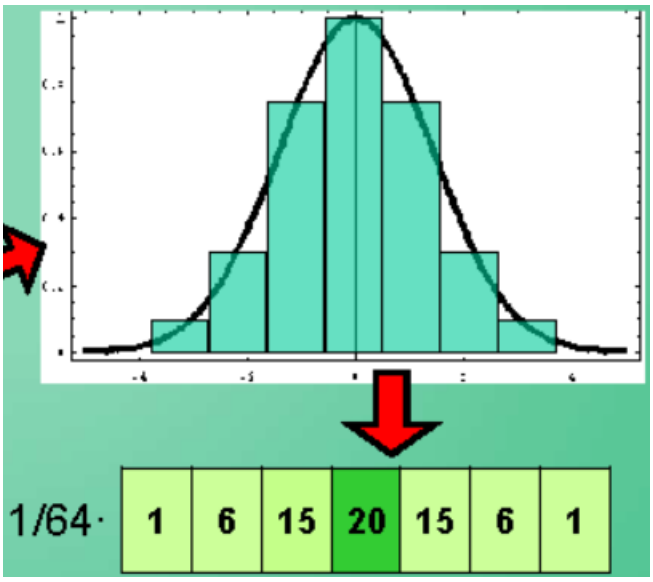

Figura 2: Ilustración de la campana de Gauss con los valores.

Figura 3: Máscara o Kernel gaussiana de 3x3.

2.2.2. Operadores de Bordes

Así como la operación suavizado reduce las variaciones de la imagen, las operaciones de bordes, tal cual como su nombre encuentra las zonas de variación (bordes). Matemáticamente, la variación de una función f(x) viene dada por la derivada de esa función (si f(x) ¿0 es creciente, f(x) = 0 es uniforme, f(x) ¡0 es decreciente). Para este caso tenemos funciones discretas. La "derivada discreta" se obtiene calculando diferencias:

$$f'(x) = \Delta f/\Delta x f'(x) = f(x) - f(x-1) \tag{2}$$

$$\Delta f = f(x) - f(x-1)\Delta x = 1 \tag{3}$$

$$f'(x) = f(x) - f(x-1) \tag{4}$$

Por lo tanto la derivada se calcula con matrices tipo -1 y 1. Además los operadores de bordes son muy sensibles al ruido, por lo que es posible combinar los operadores de bordes con los de suavizado, obteniendo una matriz de este tipo:

Por lo tanto la derivada se calcula con matrices tipo -1 y 1. Además los operadores de bordes son muy sensibles al ruido, por lo que es posible combinar los operadores de bordes con los de suavizado, obteniendo una matriz de este tipo:

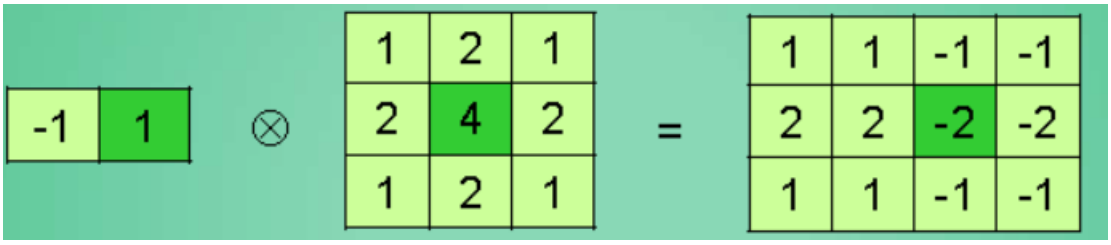

Figura 4: Combinación entre operadores de borde y suavizado.

2.3. Aplicación de la transformada de Fourier en imágenes

Para una señal sinusoidal, si la amplitud varía muy rápido en poco tiempo, se puede decir que es una señal de alta frecuencia. Si varía lentamente, es una señal de baja frecuencia. Se puede extender la misma idea a las imágenes. ¿Dónde varía drásticamente la amplitud en las imágenes? En los puntos extremos, o ruidos. Así aplicando la transformada de fourier en imágenes podemos decir que los bordes y los ruidos son contenidos de alta frecuencia en una imagen. Si no hay muchos cambios en la amplitud, se trata de una componente de baja frecuencia.

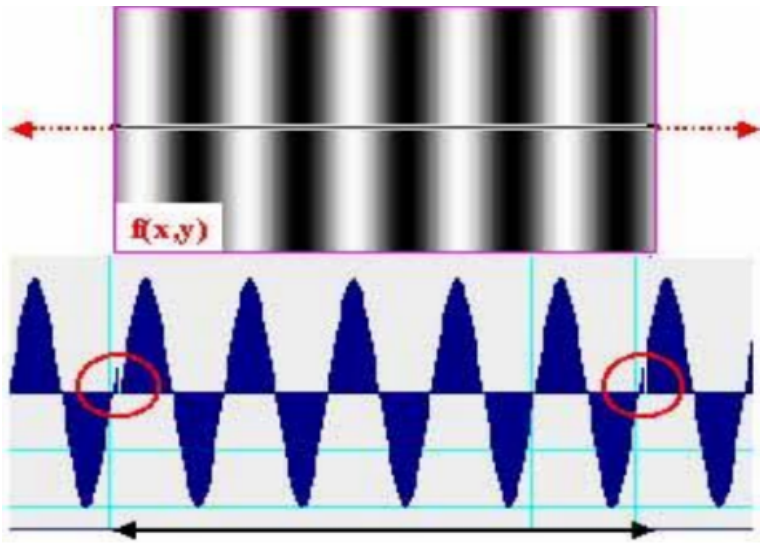

Figura 5: Proceso de conversión de la transformada a escala de grises.

3. Desarollo de la experiencia

3.1. Herramientas Utilizadas

Primeramente, antes de comenzar la experiencia debemos tener instaladas las herramientas básicas para un correcto funcionamiento del programa, estas herramientas son:

- **Python:** Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

- **Numpy:** Es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.

- **Scipy:** Es una biblioteca open source de herramientas y algoritmos matemáticos para Python. Además contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen y otras tareas para la ciencia e ingeniería.

- **Matplotlib:** Es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays.

3.2. Lectura de la imagen

Se lee la imagen a través de la función imageio.imread, con la cual transforma los pixeles en una tupla de tuplas en la que en cada tupla contiene 3 valores, que equivalen a los colores RGB, es decir, rojo,verde y azul. La función anteriormente mencionada puede leer cualquier formato de imagen sea JPG, PNG, etc. En este caso probamos con una imagen de formato "bmp". Además se puede obtener el largo y ancho de la imagen actual.

```
#########Lectura de una imagen#############
path="leena512.bmp"
img= Image.open(path)
f=  imageio.imread('leena512.bmp')
type(f)
```

Figura 6: Código que muestra cómo leer la imagen seleccionada.

Figura 7: Imagen original.

3.3. Implementación de la convolución entre la imagen y un kernel

Primero antes de aplicar el kernel en la imagen, se debe saber dónde se van a guardar los nuevos píxeles de la convolución. Se crean 3 listas, las cuales son rojo, verde y azul en las cuales se irán guardando los nuevos valores de cada color, para después obtener el píxel resultante de la convolución. Ya creando estas listas de puede trabajar con el kernel.

3.3.1. Kernel para un filtro de suavizado Gaussiano

El primer kernel a utilizar es un filtro de suavizado gaussiano, el cual se expresa como una matriz de 5x5. En cada posición de la matriz anterior se multiplica por 1/256 ya que corresponde a la fila 8 del triángulo de pascal.

De esa forma el Kernel utilizado será:

$$M = \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 246 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 246 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix} \cdot 1/256$$

Ya realizada la convolución se obtiene una imagen nueva aplicando el filtro de Suavizado.

Figura 8: Imagen nueva usando filtro de suavizado gaussiano.

3.3.2. Kernel para un filtro detector de bordes

El procedimiento para aplicar el filtro detector de bordes es similar al anterior. Este filtro se particulariza porque los bordes de una imagen digital se pueden definir como transiciones entre dos regiones de niveles de gris significativamente distintos. Suministran una valiosa información sobre las fronteras de los objetos y puede ser utilizada para segmentar la imagen, reconocer objetos, etc.

De esa forma el Kernel utilizado será:

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

Tal como visualizamos en la imágen, el tipo de kernel utilizado será el operador Sobel. Este se encarga de calcular el gradiente de la intensidad de una imagen en cada punto (píxel). Así, para cada punto, este operador da la magnitud del mayor cambio posible, la dirección de éste y el sentido desde oscuro a claro. El resultado muestra cómo de abruptamente o suavemente cambia una imagen en cada punto analizado y, en consecuencia, cuán probable es que éste represente un borde en la imagen y, también, la orientación a la que tiende ese borde.

Por lo tanto, la imágen obtenida al aplicar este kernel será:

Figura 9: Imagen nueva usando filtro de borde.

3.4. Transformada de Fourier para imágenes

Se utiliza la transformada de Fourier para analizar las características de frecuencia de varios filtros. Para las imágenes, la transformada discreta de Fourier 2D se utiliza para encontrar el dominio de frecuencia. Para el cálculo de la DFT se utiliza un algoritmo rápido llamado Transformada Rápida de Fourier (o Fast Fourier Transform en inglés, abreviado como FFT).

3.4.1. Transformada de Fourier en imágen original

Primero veremos cómo encontrar Transformada de Fourier usando Numpy. Numpy tiene un paquete FFT para hacer esto. La función **np. fft. fft2()** nos proporciona la transformación de frecuencia, la cual será una matriz compleja. Su primer argumento es la imagen de entrada, que deberá estar en escala de grises (8 bits). El segundo argumento es opcional y decide el tamaño de la matriz de salida.

En los casos de las imágenes filtradas además se debió utilizar la función .convert() ya que estas eran entregadas en formato 24 bits, lo cual para la transformada era complicado de procesar y no realizaba el procedimiento correctamente, en cambio, con una imágen tipo 8 bits la calidad de la imágen se ve compatible y realiza las salidas exitosamente.

Utilizando este procedimiento, logramos obtener los siguientes gráficos de las imágenes obtenidas, donde se puede apreciar una región más blanca en el centro mostrando que el contenido de baja frecuencia es mayor.

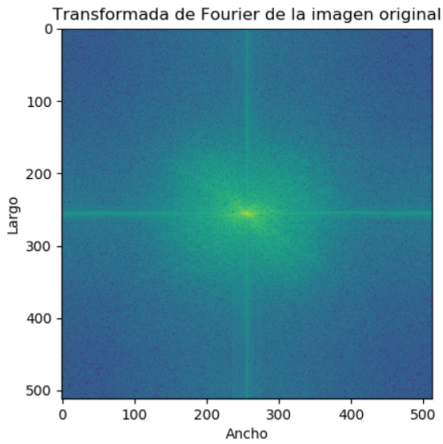

Figura 10: Gráfico de la transformada de Fourier de la imagen original.

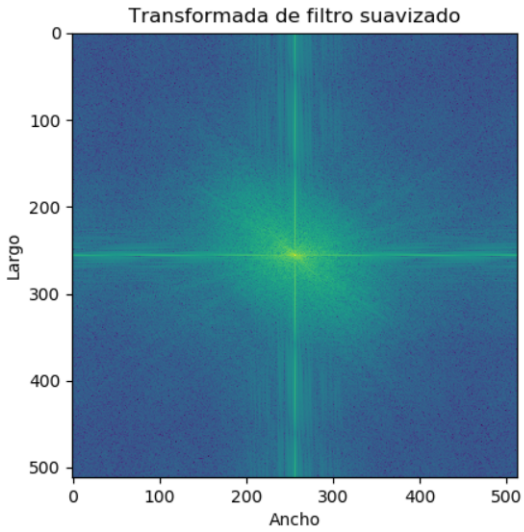

Figura 11: Gráfico de la transformada de Fourier de la imagen con filtro de suavizado.

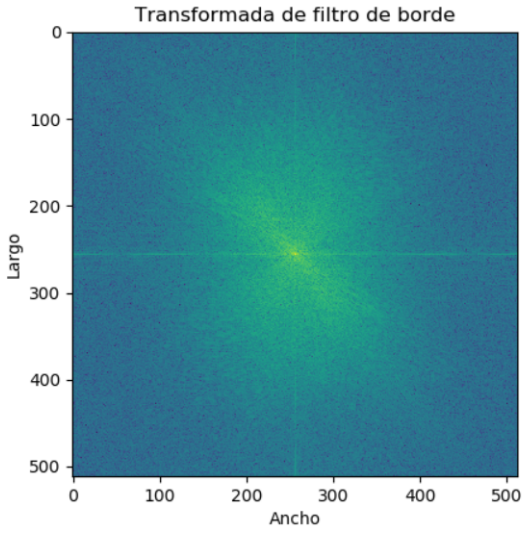

Figura 12: Gráfico de la transformada de Fourier de la imagen con filtro de borde.

4. Análisis de los resultados

La experiencia arrojó varios resultados que se logra visualizar cómo cambió la imagen utilizando cada kernel y cuáles fueron sus respectivas transformadas de Fourier

4.1. Aplicación del filtro de suavizado gaussiano

En la imagen filtrada se puede apreciar que está más difuminada que la original, es decir, a disminuido la claridad y la exactitud del contorno de la imagen, reduce contrastes abruptos en la imagen. Al analizar el gráfico que muestra la transformada de Fourier de este podemos notar que hay uno "rayos claros" en el gráfico, los cuales indican que en esos puntos de la imagen hay concentrado mayores niveles de esa frecuencia, es decir, existe más "energía", que muestra el nivel de repetición de esa frecuencia. Al ir alejándose del centro son las frecuencias más altas donde claramente se reduce la intensidad.

4.2. Aplicación del filtro de borde

En la imagen filtrada se puede apreciar que toda la imagen se oscureció casi en su totalidad, pero tal como su nombre se dice remarca se aprecia un color más claro en los bordes, esto se debe porque en estos puntos existe la máxima pendiente de variación de la imagen, en donde el cambio de los valores de los píxeles es más grande que el resto. Al analizar el gráfico de la transformada de Fourier se puede notar igual que en el filtro anterior hay un "rayo claro" en forma de cruz es donde están concentrados los mayores niveles de una frecuencia.

4.3. Comparación de filtros

Si comparamos ambas imágenes de los filtros, notamos que son muy distintas entre ellas, ya que, cada una tiene un objetivo distinto en la imagen los cuales fueron explicados en los puntos anteriores, aunque, el kernel de borde combina los bordes con suavizados, ya que los operadores de bordes son muy sensibles al ruido (ver capítulo 2. marco teórico, subsección 2.2.2).

Al comparar ambos gráficos de la transformada de Fourier de cada filtro, podemos notar

similitudes en algunas cosas. Una de ellas es que las frecuencias altas son las que están más lejos del centro del gráfico, así que podemos notar que en ellas reduce su intensidad. Una diferencia que se puede ver al comparar ambos filtros es, que en el filtro de suavizado, se ve que las frecuencias más bajas se remarcan mayormente, los cuales son los blancos de la imagen. Por el contrario, el filtro de bordes es más disperso, esto se debe a que las frecuencias varían de blanco a negro muy "drásticamente" entonces la variación es mayor en varios puntos de la imagen.

5. Conclusiones

Luego de la experiencia realizada pudimos analizar y entender de mejor manera los conceptos relacionados con la transformada de Fourier en señales reales y la convolución., Para aquello esta vez se utilizó imágenes .bmp, donde pudimos aplicar las funciones que nos permite realizar la convolución, como lo es el filtro de imágenes y la transformada de fourier en 2D.

Para aquello se seleccionó la clásica imágen "leena.bmp" en el cual manualmente realizamos la aplicación del kernel a la imágen, donde por cada pixel realizamos la multiplicación de matrices, lo sumamos y normalizamos dividiendo por la cantidad de sumas, dándonos un valor entre 0 - 255 en formato rgb. Luego al combinar esos números nos entregaba el nuevo color del píxel, donde al juntarlos todos en una matriz nos entregó la nueva imagen filtrada.

Posteriormente, para analizar su variación en frecuencias respecto a la imagen original calculamos la transformada de fourier de cada una, donde visualizamos explícitamente como este se comporta al aplicar los filtros, en que las frecuencias se modifican dependiendo la función que se ingrese.

De ésta forma al analizar las salidas obtenidas, pudimos comprobar la materia vista en clases, en que claramente se ve la aplicación de la convolución en señales, donde un función de entrada que sería la imágen de se ve modificada por una función intermedia (filtro), para obtener la función final que sería la imágen filtrada. Por lo que podemos decir que al aplicar los conocimientos aprendidos en cátedra y laboratorio en la elaboración del programa, el proceso fue realizado exitosamente obteniendo las salidas deseadas y comprobando la teoría vista en clases.

Finalmente, podemos decir que gracias al laboratorio número tres hemos reforzado nuestros conocimientos acerca de las señales eléctricas y de igual manera nuestros conocimientos en el área de la programación, donde se espera en un próximo trabajo de laboratorio seguir complementando los conocimientos vistos en este laboratorio con otras nuevas enseñanzas de las clases.

6. Referencias

- Turmero, P. (2015). Filtros y convoluciones (Procesamiento de imágenes). Disponible en: https://www.monografias.com/trabajos108/filtros-y-convoluciones/filtros-y-convoluciones2.shtml
- Unipython SA. (2018). Transformada de Fourier. Disponible en:https://unipython.com/transformada-de-fourier/
- Roncaglolio, P. (2007). Tratamiento de imágenes en el dominio de la frecuencia. Disponible en: http://www2.elo.utfsm.cl/ elo328/pdf1dpp/PDI09$_F recuencia_1 dpp.pdf$