Departamento de Ingeniería Eléctrica
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
EL7008-1 Procesamiento Avanzado de Imágenes

Tarea 1

Piramides de *Gauss* y *Laplace*

Diego Irarrázaval I.

Profesor:
Javier Ruiz del Solar.

Auxiliar:
Patricio Loncomilla Z.

Fecha:
26 de septiembre de 2020

Índice

1. Introducción **1**

2. Marco Teórico **2**
2.1. Cnvolución 2

3. Desarrollo **3**
3.1. Pirámide de *Gauss*: 3
3.2. Pirámide de *Laplace*: 4
3.3. Reconstrucción imagen: 4

4. Conclusión **4**

Bibliografía **5**

5. Anexos **5**

Índice de figuras

1. Convolucion en dos dimensiones con padding.[2] 2

Índice de tablas

Índice de Códigos

1. Implementación de convolución en Cython. 3

1. Introducción

En esta tarea, se implementará el cálculo de pirámides de *Gauss* y *Laplace* de una imagen y, luego, se reconstruirán a partir de dichas pirámides. Para lograr esto, se deberá implementar también la convolución en dos dimensiones.

Los principales objetivos corresponden en primer lugar a introducir algunas formas de representaciones multi-resolución calculadas a partir de una images e implementar operaciones desde cero (por ejemplo la convolución) que usualmente se cargan con librerías.

El informe comienza con el marco teórico donde se expone brevemente sobre la convolución, la pirámide de Gauss, la pirámide de Laplace y la reconstrucción de la imagen.

A continuación, la sección desarollo se divide en tres sub-secciones:

1. Pirámide de Gauss.
2. Pirámide de Laplace.
3. Reconstrucción imagen.

En las secciones enumeradas anteriormente, se incluye tanto el código implementado como análisis teórico de lo desarrollado.

Finalmente, se presentan las conclusiones del desarrollo de la tarea 1.

2. Marco Teórico

2.1. Cnvolución

En matemáticas, la convolución es una operación que recibe dos funciones (f y g) y entrega una tercera función ($f * g$) que describe como la forma de una es influida por la otra [1].

En procesamiento de señales, se puede entender como afecta se ve afectada señal pasar por un filtro. En este caso, la señal corresponde a una imagen y el filtro será otra imagen (le llamaremos indistintamente *kernel*, *mask* o máscara). A continuación, se muestra una ilustración que se utilizará para explicar la operación convolución en dos dimensiones:

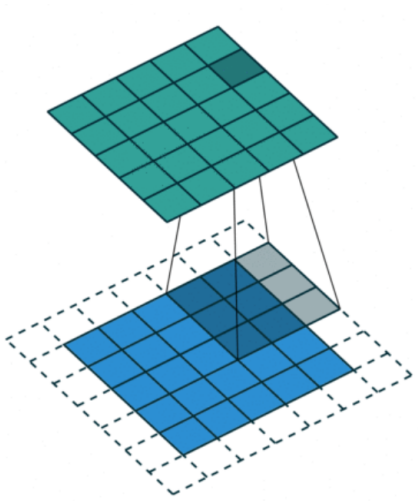

Figura 1: Convolucion en dos dimensiones con padding.[2]

En la figura se observa una imagen de entrada (representada por la matriz de cuadrados azules) con 5 pixeles de ancho por 5 de largo. Adicionalmente, se agrega un pixel de ancho y uno de largo de color blanco. Éstos representan el *padding*.El kernel por otro lado, se representa por los pixeles sombreados (kernel de 3x3).

Para el cálculo de la imagen de salida, se utiliza la siguiente ecuación:

$$y[i,j] = \sum_{m}^{M}\sum_{n}^{N} x[i-m, j-n] \cdot h[m,n] \quad \forall i \in [0, Height],\ j \in [0, Width] \tag{1}$$

En la ecuación 1, la imagen de salida (y), se construye con la operación del kernel (h) sobre todos los pixeles de la imagen de entrada (x) y los vecinos correspondientes. El sistema descrito corresponde a una imagen

Describir operación de convolución
Describir brevemente cálculo de la pirámide de Gauss
Describir brevemente cálculo de la pirámide de Laplace
Describir brevemente reconstrucción de la imagen original

3. Desarrollo

3.1. Pirámide de *Gauss*:

Convolución:

A continuación, se presenta el código de la implementación de la convolución en *Cython*:

```
cpdef float[:, :] convolution_cython(float [:, :] input, float [:, :] mask ):
cdef int a, b, r, c, rows, cols, row_init, col_init, i, j,
cdef float sum
# Imagen de salida
cdef np.ndarray output=np.zeros([input.shape[0], input.shape[1]], dtype = np.float32)

# Posicion a partir de la cual se puede realizar convolucion:
# Ejemplo 1: Para un kernel de 3x3, es (1,1).
# Ejemplo 2: Para un kernel de "a" x "b" es ("r"//2, "c"//2)
a = mask.shape[0]
b = mask.shape[1]

row_init = a // 2
col_init = b // 2

# tamano de la imagen
rows = input.shape[0]
cols = input.shape[1]

sum = 0

# Recorremos la imagen input:
for r in range(row_init, rows - row_init):
  for c in range(col_init, cols - col_init):
    # Se recorre la mascara o kernel:
    for i in range(a):
      for j in range(b):
        sum += mask[i,j] * input[r-i,c-j]
    # Guardamos el resultado de la suma correspondiente en el arreglo output:
    output[r, c] = sum
    sum = 0
return output
```

Código 1: Implementación de convolución en Cython.

La implementación en 1, corresponde a la convolución en dos dimensiones con padding. En el código, la sección más importante corresponde a los 4 ciclos de iteraciones anidados. Estos son los que recorren en primer lugar la imagen (`for r in rows` y luego `for c in columns`) y luego el *kernel* guardando en la variable `sum` el resultado de operar para cada pixel `(r,c)` de la imagen, el resultado de la operación convolución.

- Describir implementación de convolución, incluyendo código - Describir implementación de cálculo de máscaras, incluyendo código - Describir implementación de suavizado de imágenes, incluyendo código - Describir implementación de submuestreo, incluyendo código - Describir implementación de pirámide de Gauss, incluyendo código - Describir implementación: graficar pirámide de Gauss, incluyendo código - Prueba del sistema de cálculo de pirámide de Gauss sobre 4 imágenes entregadas, incluir las imágenes de las pirámides resultantes en el informe - Análisis del desempeño del cálculo de la pirámide de Gauss, analizando las imágenes resultantes

3.2. Pirámide de *Laplace*:

- Describir implementación de resta de imágenes, incluyendo código - Describir implementación de pirámide de Laplace, incluyendo código - Describir implementación de valor absoluto y escalamiento, incluyendo código - Describir implementación: graficar pirámide de Laplace, incluyendo código - Prueba del sistema de cálculo de pirámide de Laplace sobre 4 imágenes entregadas, incluir las imágenes de las pirámides resultantes en el informe - Análisis del desempeño del cálculo de la pirámide de Laplace, analizando las imágenes resultantes

3.3. Reconstrucción imagen:

- Describir implementación de suma de imágenes, incluyendo código - Describir implementación de duplicación de tamaño de imágenes con interpolación, incluyendo código - Describir implementación de reconstrucción de imagen original, incluyendo código - Prueba del sistema de reconstrucción de la imagen original usando las pirámides de las cuatro imágenes entregadas, incluir las imágenes reconstruidas en el informe - Análisis del resultado de la reconstrucción respecto a las imágenes originales

4. Conclusión

Bibliografía

[1] Wikipedia: Convolution.
https://en.wikipedia.org/wiki/Convolution#Visual_explanation

[2] Towards Data Science: Intuitively Understanding Convolutions for Deep Learning. By Irhum Shafkat.
https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning

5. Anexos