UNIVERSIDAD
DE GRANADA

Visión por Computador
grado en ingeniería informática

Trabajo 1

Filtrado y Detección de Regiones

Autor
Vladislav Nikolov Vasilev

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación

Curso 2019-2020

Índice

1. Ejercicio sobre filtros básicos **2**
1.1. Apartado A 2
1.2. Apartado B 6

2. Ejercicio sobre pirámides y detección de regiones **7**
2.1. Apartado A 7
2.2. Apartado B 7
2.3. Apartado C 7

3. Imágenes híbridas **8**
3.1. Apartado 1 8
3.2. Apartado 2 8
3.3. Apartado 3 8

4. Bonus **9**
4.1. Convolución 2D propia 9
4.2. Imágenes híbridas a color 9
4.3. Imágen híbrida propia 9

Referencias **10**

1. Ejercicio sobre filtros básicos

USANDO LAS FUNCIONES DE OPENCV: escribir funciones que implementen los siguientes puntos:

A) El cálculo de la convolución de una imagen con una máscara 2D. Usar una Gaussiana 2D (GaussianBlur) y máscaras 1D dadas por getDerivKernels). Mostrar ejemplos con distintos tamaños de máscara, valores de sigma y condiciones de contorno. Valorar los resultados.

B) Usar la función Laplacian para el cálculo de la convolución 2D con una máscara normalizada de Laplaciana-de-Gaussiana de tamaño variable. Mostrar ejemplos de funcionamiento usando dos tipos de bordes y dos valores de sigma: 1 y 3.

1.1. Apartado A

Para realizar todas las pruebas, vamos a utilizar una imágen en blanco y negro. De esta forma, los resultados se podrán ver de forma más clara. Se puede utilizar cualquiera de las imágenes que se han proporcionado, pero vamos a utilizar la foto del gato. A continuación se puede ver la imágen en cuestión:

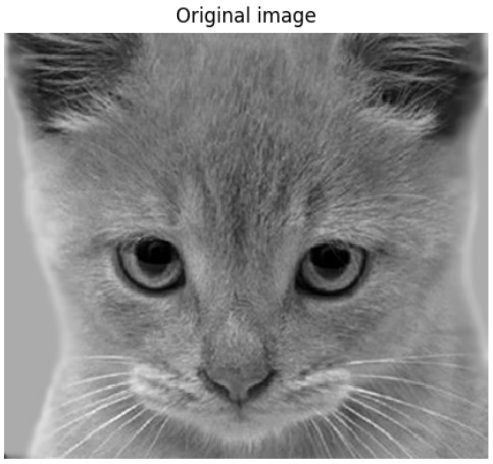


Figura 1: Imágen original del gato en blanco y negro.

Antes de ver cuáles son los resultados y compararlos. Hace falta aclarar algunos puntos:

- Cuando se cargan las imágenes, se convierten a *float64*, para así tener más precisión a la hora de hacer las posteriores operaciones (aplicar filtros, sumar o restar imágenes, etc.), además de que así no nos limitamos a usar solo valores en el rango $[0, 255]$.
- Las operaciones para aplicar filtros de OpenCV aplican una correlación. Por tanto, para aplicar una convolución, se debe realizar un *flip* del *kernel*; esto es, darle la vuelta en el eje X y en el eje Y. Como en todos los casos trabajaremos con *kernels* separables, solo será necesario darles la vuelta en un sentido.
- Respecto al punto anterior, al trabajar con *kernels* separables, con tal de mejorar la eficiencia, se puede aplicar primero el *kernel* del eje X y, sobre el resultado, aplicar el *kernel* en el eje Y. Esto es mucho más rápido que aplicar directamente un *kernel* 2D sobre la imágen, debido a que el número de operaciones es mucho menor.

Como en este ejercicio se piden aplicar *kernels* de Gaussiana y de derivada, se han hecho funciones específicas las cuáles pueden ser consultadas en el código. Estas funciones son *gaussian_kernel()* y *derivative_kernel()*, respectivamente. No se va a especificar exactamente lo que hacen debido a que solo obtienen los *kernels* con los parámetros adecuados (valores de σ y tamaño de cada *kernel* para el caso de la Gaussiana, y tamaño del *kernel* y número de veces que se deriva en cada eje en el caso de la derivada).

Lo que sí que es importante destacar es que estas dos funciones utilizan una común para realizar la convolución. Como se dijo anteriormente, OpenCV realiza como tal la correlación, pero se puede aplicar una convolución realizando unos pocos cambios. La función que se puede ver a continuación recibe un *kernel* para cada eje y aplica el filtro mediante la convolución:

```
def apply_kernel(img, kx, ky, border):
    """
    Funcion que aplica un kernel separable sobre una imagen,
    realizando una convolucion

    Args:
        img: Imagen sobre la que aplicar el filtro
        kx: Kernel en el eje X
        ky: Kernel en el eje Y
        border: Tipo de borde
    Return:
        Devuelve una imagen filtrada
    """
```

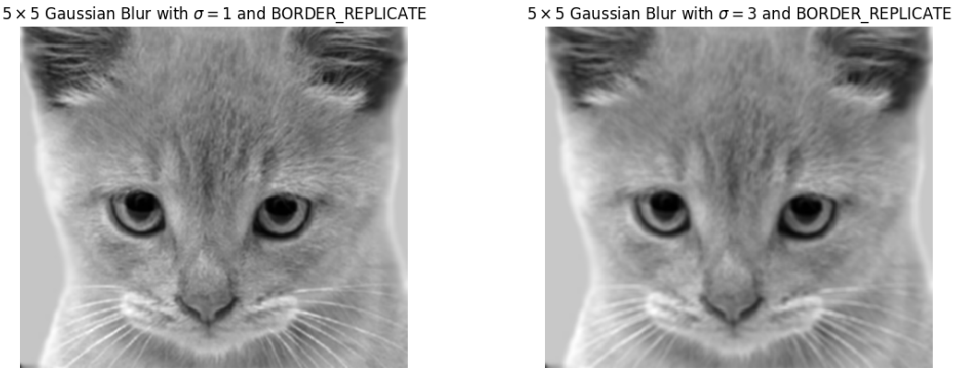
```
    # Hacer el flip a los kernels para aplicarlos como una
    convolucion
    kx_flip = np.flip(kx)
    ky_flip = np.flip(ky)

    # Realizar la convolucion
    conv_x = cv.filter2D(img, cv.CV_64F, kx_flip.T,
                         borderType=border)
    conv = cv.filter2D(conv_x, cv.CV_64F, ky_flip,
                       borderType=border)

    return conv
```

Como se puede ver, se tiene que hacer un *flip* de los *kernels* para poder aplicar una convolución. Al aplicar cada *kernel* de forma separada mediante $filter2D$, se consigue una mayor eficiencia (como se ha mencionado anteriormente). Es importante destacar que primero se aplica el *kernel* sobre el eje de las X y, sobre el resultado obtenido, se aplica el *kernel* en el eje Y. También es importante destacar que, cuando se aplica el *kernel* sobre el eje X, se le pasa la traspuesta del *kernel*. Esto se debe a que OpenCV proporciona los *kernels* como vectores columnas, y para pasarlo por las filas, necesitamos que sea un vector fila. Al traponer el vector columna obtenemos, como parece lógico, un vector columna.

Otra cosa muy importante a destacar son los *kernels* que se van a utilizar. Uno de ellos es el *kernel* Gaussiano, el cuál es simétrico tanto en el eje X como en el Y. Al aplicarlo, por tanto, se podría ahorrar la operación del *flip*, ya que lo va a dejar igual. Sin embargo, debido a que se ha implementado la función para que sea genérica, se realiza esta operación. El otro *kernel* que se utiliza es el de las derivadas, que no es más que un *kernel* de Sobel, el cuál combina alisamiento Gaussiano con la derivada. A diferencia del anterior *kernel*, este no es simétrico, y por tanto, la operación de *flip* no lo va a dejar igual; por tanto, no puede ser ahorrada en este caso.

Con esto comentado, ya podemos empezar a hablar de los resultados que se han obtenido. Para ello, vamos a comenzar comentando los resultados que se obtienen para el filtro Gaussiano.

Lo primero que se ha probado es un *kernel* de 5×5, variando los valores de σ para ver cómo cambiaba. A continuación se pueden ver los resultados:

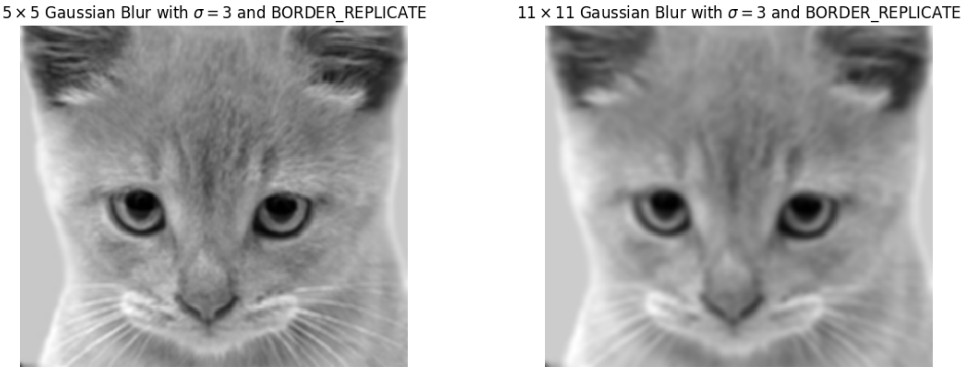


(a) Filtro Gaussiano con $\sigma = 1$. (b) Filtro Gaussiano con $\sigma = 3$.

Figura 2: Filtro Gaussiano con diferentes tamaños de σ.

Como se puede ver, comparando cualquiera de las imágenes de la figura 2 con 1, existen ciertas diferencias. Podemos ver claramente como al aplicar el filtro Gaussiano se ha perdido cierto detalle, ya que se están eliminando las frecuencias altas. Además, si comparamos las figuras 2a y 2b, podemos ver que a medida que aumentamos el tamaño de σ, se van perdiendo más detalles, ya que se emborrona más.

Se ha probado también a variar el tamaño de la máscara, conservando el mismo valor de σ, para ver qué es lo que sucede. A continuación, se puede ver lo que se ha obtenido:



(a) Filtro Gaussiano de tamaño 5×5. (b) Filtro Gaussiano de tamaño 11×11.

Figura 3: Filtro Gaussiano con diferentes tamaños de *kernel*.

Tal y como se puede ver en la figura 3, al modificar el tamaño del *kernel* se han obtenido diferentes resultados. Esto se debe a que, si se tiene un valor de σ alto con una máscara pequeña, esta tiende a comportarse como un filtro de caja, ponderando más o menos de forma equitativa a los vecinos del píxel

1.2. Apartado B

2. Ejercicio sobre pirámides y detección de regiones

IMPLEMENTAR funciones para las siguiente tareas:

A) Una función que genere una representación en pirámide Gaussiana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes y justificar la elección de los parámetros.

B) Una función que genere una representación en pirámide Laplaciana de 4 niveles de una imagen. Mostrar ejemplos de funcionamiento usando bordes.

C) Construir un espacio de escalas Laplaciano para implementar la búsqueda de regiones usando el siguiente algoritmo:

- **a.** Fijar sigma
- **b.** Repetir para N escalas
- **c.** Realizar supresión de no-máximos en cada escala
  - i. Filtrar la imagen con la Laplaciana-Gaussiana normalizada en escala
  - ii. Guardar el cuadrado de la respuesta para el actual nivel del espacio de escalas
  - iii. Incrementar el valor de sigma por un coeficiente k.(1.2-1.4)
- **d.** Mostrar las regiones encontradas en sus correspondientes escalas. Dibujar círculos con radio proporcional a la escala.

2.1. Apartado A

2.2. Apartado B

2.3. Apartado C

3. Imágenes híbridas

Mezclando adecuadamente una parte de las frecuencias altas de una imagen con una parte de las frecuencias bajas de otra imagen, obtenemos una imagen híbrida que admite distintas interpretaciones a distintas distancias (ver hybrid images project page).

Para seleccionar la parte de frecuencias altas y bajas que nos quedamos de cada una de las imágenes usaremos el parámetro sigma del núcleo/-máscara de alisamiento gaussiano que usaremos. A mayor valor de sigma mayor eliminación de altas frecuencias en la imagen convolucionada. Para una buena implementación elegir dicho valor de forma separada para cada una de las dos imágenes (ver las recomendaciones dadas en el paper de Oliva et al.). Recordar que las máscaras 1D siempre deben tener de longitud un número impar.

Implementar una función que genere las imágenes de baja y alta frecuencia a partir de las parejas de imágenes (solo en la versión de imágenes de gris). El valor de sigma más adecuado para cada pareja habrá que encontrarlo por experimentación.

1. Escribir una función que muestre las tres imágenes (alta, baja e híbrida) en una misma ventana. (Recordar que las imágenes después de una convolución contienen número flotantes que pueden ser positivos y negativos)
2. Realizar la composición con al menos 3 de las parejas de imágenes
3. Construir pirámides gaussianas de al menos 4 níveles con las imágenes resultado. Explicar el efecto que se observa.

3.1. Apartado 1

3.2. Apartado 2

3.3. Apartado 3

4. Bonus

4.1. Convolución 2D propia

Implementar con código propio la convolución 2D con cualquier máscara 2D de números reales usando máscaras separables.

4.2. Imágenes híbridas a color

Realizar todas las parejas de imágenes híbridas en su formato a color (solo se tendrá en cuenta si la versión de gris es correcta).

4.3. Imágen híbrida propia

Realizar una imagen híbrida con al menos una pareja de imágenes de su elección que hayan sido extraídas de imágenes más grandes. Justifique la elección y todos los pasos que realiza.

Referencias

[1] Texto referencia
`https://url.referencia.com`