Práctica 1: Filtrado y Detección de regiones

Visión por Computador

Alejandro Palencia Blanco

28/10/2020

Ejercicio 1

- **Apartado A**

  Para crear las tres máscaras pedidas, he decidido implementar una única función `mascaraGaussiana1D`, que permite calcular dichas máscaras a partir de un valor (que será la desviación típica o el tamaño de la máscara dependiendo del valor de un flag asociado) y el orden de la derivada. Si se pasa como parámetro la desviación típica, entonces el tamaño de la máscara será $2\,ceil(3\sigma)+1$, siendo $ceil$ la función techo. Por otro lado, si se pasa el tamaño de máscara, entonces se usa el máximo valor de sigma tal que $ceil(3\sigma) = \frac{tam-1}{2}$. Por tanto, la desviación típica será $\frac{tam-1}{6}$.

  A su vez, esta función llama a otras tres funciones que simplemente devuelven los valores de la gaussiana, su primera derivada o su segunda derivada, respectivamente:

  - `gaussiana`: $e^{\frac{-x^2}{2\sigma^2}}$
  - `gaussiana1Deriv`: $\frac{-x}{\sigma^2}e^{\frac{-x^2}{2\sigma^2}}$
  - `gaussiana2Deriv`: $\left(\frac{x^2}{\sigma^4}-\frac{1}{\sigma^2}\right)e^{\frac{-x^2}{2\sigma^2}}$

  Con el objetivo de optimizar la implementación, he aprovechado la paridad de la gaussiana y su segunda derivada ($f(-x)=f(x)$) y la imparidad de la primera derivada ($f(-x)=-f(x)$) para calcular los valores negativos a partir de los positivos y así reducir a la mitad las iteraciones en los respectivos bucles for.

  Para probar estas funciones, he creado máscaras para los tres órdenes de derivación pasando primero un sigma cuyo valor es 3 y luego un tamaño de máscara con valor 15.

- **Apartado B**

  La máscara 1D de la gaussiana necesaria para la convolución puede ser calculada mediante la función del apartado anterior. Para optimizar los cálculos, tendremos en cuenta lo siguiente:

  - Por la simetría de la máscara gaussiana, podemos calcular la comvolución como una correlación.
  - Por la separabilidad de la máscara gaussiana 2D, podemos calcular convolución pasando la máscara gaussiana 1D primero por filas y luego por columnas. Esto reducirá el número de operaciones de cuadrático a lineal.

  La función que implementará el cálculo de la convolución es `correlacionMascaras1D`, que aplica dos máscaras 1D primero por filas y luego por columnas. Además, también tiene un flag que le permite tratar los bordes de tres formas diferentes. Primero, esta función comprueba que las longitudes de ambas máscaras son la misma y que estas no superan a las dimensiones de la imagen. En caso afirmativo, se llama a una función auxiliar `aplicarMascaraReplicada` que permite pasar una máscara 1D por filas sobre una imagen. Esto lo hace en tres pasos:

1. Obtener una matriz con la máscara replicada por filas.
2. Orlar la imagen a izquierda y derecha según el borde especificado.
3. Aplicar la máscara replicada a la imagen: Barremos de izquierda a derecha la imagen orlada con la matriz de la máscara replicada, calculando primero la matriz producto elemento a elemento y luego sumando por filas. El resultado de cada iteración es una columna de píxeles de la imagen final.

Para orlar las imágenes, utilizamos estas tres funciones que implementan distintos tipos de bordes:

- `orlaCeros`: Bordes constante cero.
- `orlaReflejo`: Bordes reflejo 101.
- `orlaReplicados`: Bordes replicados.

Para realizar la correlación, primero se llama a `aplicarMascaraReplicada` con la imagen original y la máscara por filas. Para no tener que implementar otra función que análoga que realize el cálculo orlando por columnas, trasponemos la matriz obtenida, llamamos otra vez a `aplicarMascaraReplicada` con la máscara por columnas y volvemos a trasponer.

He probado esta función con un tamaño de máscara 15. Primero he mostrado la imagen original, luego la convolución calculada por `correlacionMascaras1D` y por último la calculada con `cv2.GaussianBlur`. En esta última, he especificado tanto el tamaño de máscara (15) como el valor de sigma, el cual se da en función del tamaño de la máscara y es el mismo que se usa en la función del apartado anterior ($\frac{tam-1}{6}$). Esto lo hago con el objetivo de que las imágenes calculadas por ambas funciones sean lo más parecidas posible. A simple vista, no se aprecian diferencias significativas entre las imágenes convolucionadas.

- **Apartado C**

  En función del eje en el apliquemos la máscara de la primera derivada, podemos obtener dos imágenes distintas: Aplicando la máscara de la derivada de la gaussiana por filas y luego la de la gaussiana por columnas, o viceversa. En cualquier caso, siempre se obtiene el mismo par de máscaras, por tanto solo tenemos que comparar las máscaras de la gaussiana y de su primera derivada.

  En todas las llamadas a la función `cv2.getDerivKernels`, he especificando que los órdenes de derivación respecto de x e y sean 1 y 0, respectivamente. Por tanto, en la primera posición se devuelve la máscara de la primera derivada de la gaussiana, y en la segunda, la máscara de la gaussiana. Además, esta función no permite calcular las máscaras en función de sigma, luego cuando haga comparaciones usando distintos valores sigma, tomaré como tamaño de máscara para `cv2.getDerivKernels` el mismo que usa mi función `mascaraGaussiana1D` para obtener el tamaño de la máscara a partir de un sigma ($2\,ceil(3\sigma)+1$, siendo *ceil* la función techo).

  Si analizamos las diferentes parejas de máscaras gaussianas obtenidas para distintos tamaños de máscara y desviaciones típicas, vemos que la forma de éstas usando la función del apartado A y `cv2.getDerivKernels` es similar. Sin embargo, también apreciamos que la escala cambia. Concretamente, `cv2.getDerivKernels` siempre devuelve una máscara de enteros positivos en la que el primer y último valor es 1. En cambio, mi función devuelve una máscara con valores dentro del intervalo $]0,1[$, lo cual era esperable ya que dichos valores suman 1.

  Por otro lado, al analizar las parejas de máscaras de derivadas gaussianas, también vemos que hay cierta similitud en la forma pero con una sutil diferencia: En mi máscara, los valores positivos se encuentran a la izquierda del cero y los negativos a la derecha, pero en la `cv2.getDerivKernels` es al contrario. Supongo que `cv2.getDerivKernels` devuelve los valores así porque la máscara ya está preparada para aplicarse en una convolución. De nuevo,

`cv2.getDerivKernels` proporciona valores enteros, pero ahora también pueden ser negativos. Además, siempre ocurre que el primer valor es -1 y el último es 1. En este caso, mi función devuelve los valores dentro del intervalo $]-1,1[$.

- **Apartado D**

  La laplaciana de una gaussiana tiene la siguiente forma, siendo $G(x,y,\sigma)$ la gaussiana:

  $$L(x,y,\sigma) = \sigma^2 \left(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)\right)$$

  Primero calculamos las derivadas parciales segundas respecto de x e y. Si tenemos que $G(x,y,\sigma) = e^{\frac{-x^2}{2\sigma^2}}\, e^{\frac{-y^2}{2\sigma^2}}$, entonces:

  $$G_{xx}(x,y,\sigma) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) e^{\frac{-x^2}{2\sigma^2}}\, e^{\frac{-y^2}{2\sigma^2}}$$

  $$G_{yy}(x,y,\sigma) = e^{\frac{-x^2}{2\sigma^2}} \left(\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right) e^{\frac{-y^2}{2\sigma^2}}$$

  Puesto que ambas derivadas segundas son el producto de una gaussiana 1D por la derivada segunda de otra gaussiana 1D, ambas con el mismo sigma, podemos implementar el cálculo a partir de dos máscaras 1D correspondientes a estas funciones. Concretamente, si calculamos la derivada parcial segunda respecto de x, realizaremos una correlación en la que primero pasaremos la máscara 1D de la segunda derivada de la gaussiana por filas, y luego la máscara 1D de la gaussiana por columnas. Para la segunda derivada parcial respecto de y, el procedimiento será análogo pero cambiando los papeles que juegan ambas máscaras. Podemos calcular estas dos correlaciones con la función `correlacionMascaras1D` ya implementada en el apartado B.

  A continuación, solo quedaría sumar las dos imágenes obtenidas y multiplicar el resultado por σ^2 para normalizar la escala. Todo este procedimiento se lleva a cabo en la función `laplaciana`.

  En este apartado, primero hago uso dela función `mascaraGaussiana1D` para calcular la máscaras de la gaussiana 1D y de su segunda derivada. Luego, aplico una normalización de escala a la máscara de la segunda derivada multiplicando por σ^2 y visualizo ambas máscaras. Por último, calculo la laplaciana de una imagen usando los tipos de bordes y los sigmas pedidos.

Ejercicio 2

- **Apartado A**

  Para representar la pirámide, he decidido hacerlo en una imagen que tiene la misma altura y el doble de anchura que la original. De esta forma, puedo colocar la imagen original a la izquierda de la imagen pirámide y, el resto de imágenes, a la derecha ésta, cada una justo abajo de la anterior.

  La función que he implementado para crear la pirámide, `piramideGaussiana`, recibe como parámetros la imagen original y, al igual que en apartados anteriores, un valor acompañado de un flag que puede contener tanto una desviación típica como un tamaño de máscara.

Este valor es necesario porque, para crear las sucesivas submuestras de la imagen original, primero tenemos que aplicar un filtro gaussiano y luego submuestrearla. Dicho valor especifica el sigma o el tamaño de la máscara 1D de gaussiana que se usará para realizar la convolución. Posteriormente, mediante la función `reducirSubmuestra`, obtengo una submuestra reducida con la mitad de filas y la mitad de columnas (si el número de filas o de columnas es impar, la imagen es tratada como si no existiese la última fila o columna). Los píxeles usados para la submuestra son aquellos que en la imagen original se encuentran en una fila y una columna impares. Esto es equivalente a eliminar de la imagen original las filas y las columnas pares.

- **Apartado B**

  La representación de esta pirámide será igual a la del apartado anterior. Para crear cada una de las imágenes laplacianas de la pirámide, primero aplico un filtro gaussiano a la submuestra reducida de la iteración anterior y la reduzco con `reducirSubmuestra`. A continuación, la amplio con la función `ampliarSubmuestra` y resto esta imagen a la submuestra anterior. Esta función duplica el tamaño de la imagen pasada como parámetro de la siguiente forma:

  - Si el píxel se encuentra en una fila y una columna impares, entonces toma el valor de uno de los píxeles de la imagen original: $img_{ampliada}[i][j] = img_{original}[(i-1)/2][(j-1)/2]$
  - Si el píxel se encuentra en una fila par y una columna impar, entonces toma el valor medio de los píxeles que se encuentran justo por encima y debajo suya en la imagen ampliada.
  - Si el píxel se encuentra en una fila impar y una columna par, entonces toma el valor medio de los píxeles que se encuentran justo por asu izquierda y derecha en la imagen ampliada.
  - Si el píxel se encuentra en una fila y una columna pares, entonces toma el valor medio de los cuatro píxeles que se encuentran justo arriba a la izquierda, arriba a la derecha, abajo a la izquierda y abajo a la derecha en la imagen ampliada.
  - Si el píxel se encuentra en una fila y una columna pares, entonces toma el valor medio de los cuatro píxeles que se encuentran justo arriba a la izquierda, arriba a la derecha, abajo a la izquierda y abajo a la derecha en la imagen ampliada.

  Por último, la función tiene dos flags que indican si es necesario añadir una fila o columna adicional (esto se hace para que se pueda hacer la resta entre imágenes sin que haya problemas de dimensión cuando la imágen original no tiene un número de filas y columnas pares). Cuando estos flags están activos, se añade una fila o columna adicional que es una copia de la fila o la columna inmediatamente anterior, respectivamente.

  Primero, la función `piramideLaplaciana` calcula las tres primeras submuestras de imágenes laplacianas y las almacena en una lista. A continuación, se llama a `normalizarListaImagenes01`, que simplemente normaliza todas las imágenes de la lista al intervalo $[0, 1]$ restando el mínimo global de los valores de todos los píxeles de todas las imágenes de la lista, y luego dividiendo entre la diferencia del máximo y mínimo globales. Después de esto, se añade a la lista la imagen obtenida en la última iteración mediante una reducción, previamente normalizada al intervalo $[0, 1]$. Por último, se añaden todas las imágenes de la lista a la imagen que guarda la pirámide.

Ejercicio 3

Para la resolución de este apartado, he creado una función `imagenHibrida` que necesita siete parámetros: dos imágenes (a una se le aplicará un filtro gaussiano y, a la segunda, un filtro laplaciano), dos sigmas para asociados a cada uno de los filtros, dos pesos y un flag.

Primero, la función aplica un filtro gaussiano a la primera imagen haciendo uso de `correlacionMascaras1D` y un filtro laplaciano a la segunda haciendo uso de `laplaciana`. Luego, normaliza las dos imágenes obtenidas al intervalo $[0, 1]$, las suma y vuelve a normalizar la imágen obtenida. A la hora de realizar la suma, cada imagen es multiplicada por uno de los dos pesos especificados como parámetros para dar mayor o menor protagonismo a cada imagen en la imagen híbrida. Finalmente, se muestran las tres imágenes en una misma ventana y se devuelve la imagen híbrida.

A la hora de elegir las parejas de imágenes para obtener imágenes híbridas, he intentado que las formas de aquello que aparece en ellas sean lo más similares posible y que, al combinarlas, los elementos que las componen queden ünos encima de otros.º superpuestos. Además, es importante también que las dimensiones de las imágenes sean las mismas. Siguiendo estos criterios, las parejas que he visto más adecuadas son: moto con bicicleta, gato con perro, Marilyn con Einstein y submarino con pez.

Una vez tenemos las parejas, hay que decidir qué imagen será sometida al filtro gaussiano y al filtro laplaciano en cada pareja. He llevado a cabo esta elección basándome en los cambios en las escalas de grises: Si una imagen presenta cambios más rápidos en la intensidad de los grises que su pareja, entonces esa será la mejor candidata para el filtro laplaciano, mientras que con la otra usaremos el filtro gaussiano. Teniendo esto en cuenta, he decidido que las imágenes que mejor aprovecharán el filtro laplaciano son bicicleta, perro, Einstein y pez.

Por último, hay que elegir tanto los sigmas que actuarán en cada uno de los filtro como los pesos que multiplicarán a cada una de las imágenes filtradas. Primero, he probado distintos valores de sigma para cada imagen, teniendo en cuenta que, cuanto mayor sea sigma, más difuminará el filtro gaussiano y mayor será la escala a la que el filtro laplaciano detecte cambios en la intensidad. En mi caso, las elecciones que me han resultado más complicada han sido las de aquellos sigmas asociados a filtros laplacianos. Luego, he intentado seleccionar los pesos de forma que ambas imágenes sean reconocibles en la híbrida. En algunos casos, esta tarea también me ha resultado un poco complicada.

En todas estas elecciones, la subjetividad de cada uno juega un papel muy importante. He intentado obtener los mejores resultados posibles siguiendo estos criterios. Aun así, creo que algunas de las imágenes híbridas que he obtenido tienen un gran margen de mejora.

Al construir las pirámides gaussianas de cada imagen híbrida, deberíamos observar que a medida que avanzamos a submuestras más reducidas, la imagen con filtro laplaciano es cada vez menos apreciable, y que por el contrario, la imagen con filtro gaussiano gana más protagonismo. Esto ocurre porque, antes de submuestrear en cada iteración, estamos aplicado un filtro gaussiano que favorece a las frecuencias bajas y perjudica a las altas. Este mismo efecto podemos apreciarlo al mirar una imagen desde cerca y luego alejarnos.

Bonus

- **Apartado 1**

  He decidido resolver este apartado creando una función `imagenHibridaColor`, que tiene los mismos parámetros que `imagenHibrida` (exceptuando el flag). Esta función calcula la imagen híbrida a color llamando a `imagenHibrida` tres veces, pasando como primer parámetro la imagen a color restringida a cada uno de los tres canales (es decir, tratamos cada uno de los canales como una imagen a escala de grises).

  He decidido calcular las imágenes híbridas de todas las parejas del ejercicio 3 exceptuando la de Einstein y Marilyn, ya que éstas no están a color.

- **Apartado 2**

  Empiezo buscando dos imágenes que, como ya se dijo en el ejercicio 3, tengan una forma similar. Además, intento que una de las imágenes presente cambios más rápidos en la intensidad y la otra cambios más suaves. Se me ocurrió que podría mezclar algún monumento con una montaña. Finalmente, me decanté por escoger imágenes de la torre Eiffel y del monte Cervino.

  Como las imágenes tienen dimensiones distintas, lo primero que hago es extraer imágenes más pequeñas de las originales de forma que tanto la montaña como el monumento queden en la misma zona de sus respectivas imágenes. Por otro lado, el monte Cervino se ve bastante más ancho que la torre Eiffel. Así que, después de reducir la torre Eiffel a 400x563 píxeles, decido reducir el monte Cervino a 800x563 píxeles, es decir, dejo la imagen el doble de ancha que la primera. Esto lo hago porque después aplico una técnica muy parecida a la usada en la pirámide gaussiana: primero, le aplico un filtro gaussiano a la imagen y, a continuación, submuestreo la imagen resultante eliminando solo la mitad de las columnas. Esto me da una imagen más estrecha sin alterar la altura, de forma que ahora ambas imágenes (la de la torre Eiffel y la del monte Cervino estrechado) tienen las mismas dimensiones.

  Por último, calculo la imagen híbrida pasándole el filtro gaussiano a la torre Eiffel y el filtro laplaciano al monte Cervino. Aprovechando que ya tengo la función `imagenHibridaColor` de la primera parte del bonus, también he hecho este mismo procedimiento para las imágenes a color.