Python's eyes

Vision, imaging & visualization…

ɦ Menú

Filtros para la deteccion de bordes de una imagen con Python 3 (Parte 1)

Escrito por *Rvziel*

Deteccion de bordes utilizando tecnicas basadas en el gradiente

Entendiendo que la derivada es una razon de cambio, en el terreno de las imagenes se produce una elevacion positiva en todo lugar donde la intensidad de los pixeles aumenta y una negativa donde la intensidad disminuya. Sin embargo, la derivada en su forma «tradicional» no esta definida para funciones discretas por lo que necesitamos un metodo para calcularla. Ademas es necesario recalcar que para la deteccion de bordes por gradiente la derivada es considerada como unidimensional (derivada parcial) por lo que se calcula dos veces, una para el sentido horizontal y otra para el sentido vertical.

Es conocido que la derivada de una funcion continua en un punto *x* puede ser interpretada por la pendiente de la tangente en ese punto en particular. Sin embargo, para una funcion discreta (como es el caso de las imagenes) la derivada en un punto *u* (la pendiente de la tangente a ese punto) puede ser calculada a partir de la diferencia existente entre los puntos vecinos a *u* dividido por el valor de muestreo entre ambos puntos. Por lo que la derivada puede ser aproximada por:

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} \approx 0.5(f(u+1) - f(u-1))$$

Ecuacion 1

**El mismo proceso se lleva a cabo para el sentido horizontal y el sentido vertical*

La derivada parcial y gradiente

La derivada parcial puede ser considerada como la derivada de una funcion multidimensional a lo largo de un eje coordenado pero calculada solo con respecto de una de las variables de la funcion (en este caso un sentido, ya sea horizontal o vertical) y el valor del gradiente combina los valores parciales para crear un solo valor, queda entonces definido como la raiz cuadrada de la suma de los cuadrados de las derivadas parciales:

$$|\bigtriangledown I| = \sqrt{a^2 + b^2}$$

Ecuacion 2

$$donde \rightarrow a = \frac{\partial I}{\partial x}; b = \frac{\partial I}{\partial y}$$

Componentes o derivada
parciales de la ecuacion 2

El gradiente es invariante a rotaciones de la imagen y con ello tambien independiente de la orientacion de las estructuras contenidas en la misma. Esta propiedad es importante para la localizacion de los puntos bordes en la imagen, con ello **es el valor practico utilizado en la mayoria de los algoritmos utilizados para la deteccion de bordes.**

La forma de calcular el gradiente local correspondiente a cada pixel de la imagen es lo que fundamentalmente diferencia a cada uno de los diferentes operadores para la deteccion de bordes.

El filtro de la derivada

Las componentes a y b de la ecuacion 2 no son otra cosa que la primera derivada tanto en el sentido de los renglones como en el de las columnas de los pixeles de la imagen.

La forma de calcular la derivada en el sentido horizontal es posible a partir de la ecuacion 1 expresarlo en un filtro con la siguiente matriz de coeficientes:

$$\frac{\partial I}{\partial x} = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

donde el coeficiente -0.5 afecta al pixel I(x-1, y) y 0.5 al pixel I(x+1, y). El valor del pixel de en medio es multiplicado por cero lo que equivale a ser ignorado y es el pixel de referencia o bien es el pixel bajo el cual se calcula la propiedad en cuestion. De igual manera se puede establecer el mismo efecto del filtro pero ahora en sentido vertical, siendo su matriz de coeficientes:

$$\frac{\partial I}{\partial y} = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Para la aplicacion de este filtro simplificaremos el calculo convirtiendo previamente la imagen a escala de grises para el caso de las imagenes a color ya que los resultados no presentan variacion alguna entre el calculo con los tres planos del RGB y el calculo con el unico plano de la escala de grises reduciendo significativamente el tiempo y el costo computacional en este ultimo caso.

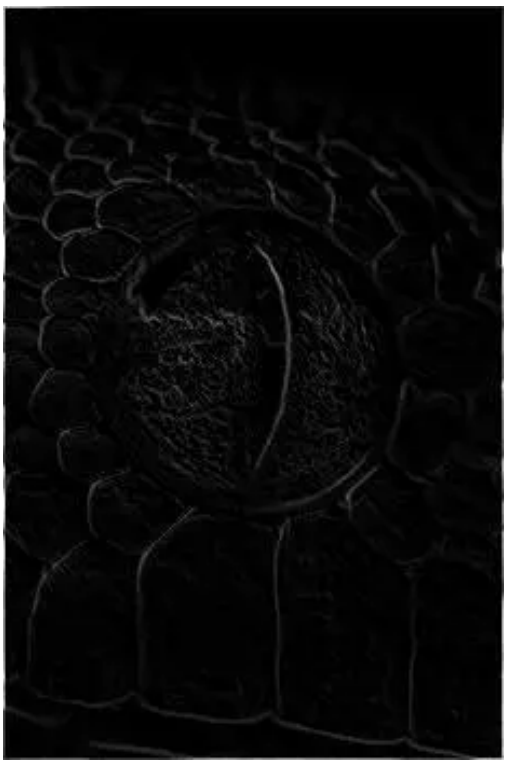
El algoritmo:

Debido a que el resultante de la convolucion contiene numeros negativos para el sentido de derecha a izquierda y de abajo hacia arriba (sentido inverso de la convolucion por los coeficientes con signo negativo) si se desea utilizar la funcion de filtro por definicion de kernel que contiene el modulo ImageFilter de PIL, es necesario aplicar dos veces el proceso, una con el set de coeficientes como se definieron anteriormente y otra con el mismo set de coeficientes pero con signo contrario y al final sumar los datos de ambas imagenes resultantes para conformar una imagen con los bordes para los 4 sentidos:

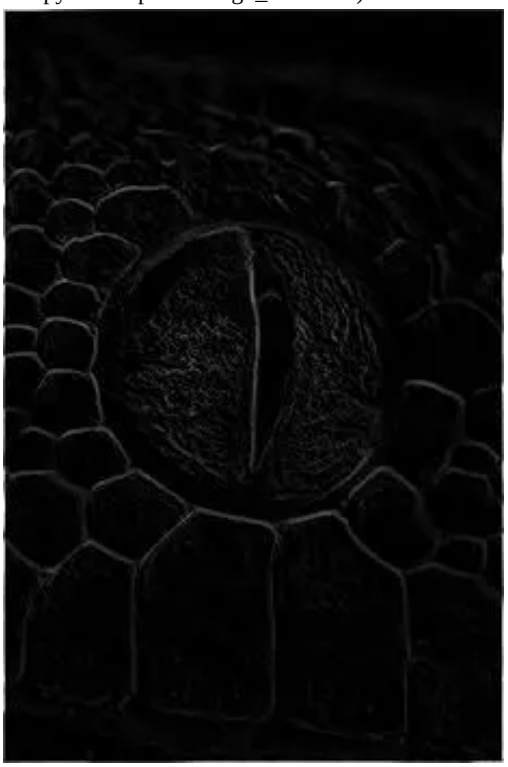
```
from PIL import Image, ImageFilter

#apertura de la imagen original
foto = Image.open('nombre_y_ruta_del_archivo)

#si la imagen no esta en escala de grises, se convierte
if foto.mode != 'L':
    foto = foto.convert('L')

#set de coeficientes originales definidos anteriormente
#coeficientes para el sentido horizontal en formato 3x3
coeficientes_h=[0, 0, 0, -1, 0, 1, 0, 0, 0]

#coeficientes para el sentido vertical en formato 3x3
coeficientes_v=[0, -1, 0, 0, 0, 0, 0, 1, 0]

#factor de division
factor = 2

tamaño = (3,3)
#derivadas parciales, horizontal y vertical
datos_h = imagen.filter(ImageFilter.Kernel(tamaño, coeficientes_h, factor)).getdata()
datos_v = imagen.filter(ImageFilter.Kernel(tamaño, coeficientes_v, factor)).getdata()

datos_derivada = []

for x in range(len(datos_h)):

    #raiz cuadrada de la suma de los cuadrados
    datos_derivada.append(round(((datos_h[x] ** 2) + (datos_v[x] ** 2)) ** 0.5))


#set de coeficientes con signo contrario a los originales
#y resto del proceso identico al anterior
coeficientes_h=[0, 0, 0, 1, 0, -1, 0, 0, 0]
coeficientes_v=[0, 1, 0, 0, 0, 0, 0, -1, 0]

datos_h = imagen.filter(ImageFilter.Kernel(tamaño, coeficientes_h, factor)).getdata()
datos_v = imagen.filter(ImageFilter.Kernel(tamaño, coeficientes_v, factor)).getdata()

datos_derivada1 = []


for x in range(len(datos_h)):

    datos_derivada1.append(round(((datos_h[x]**2)+(datos_v[x]**2))**0.5))


datos_bordes = []


for x in range(len(datos_derivada)):

    #suma de los datos de las dos imagenes para conformar la imagen final
    datos_bordes.append(datos_derivada[x] + datos_derivada1[x])


nueva_imagen = Image.new('L', foto.size)
nueva_imagen.putdata(datos_bordes)

#guardar el resultado
nueva_imagen.save('ruta_y_nombre_del_archivo)
```

```
#cerrar los objetos de la clase Image
foto.close()
nueva_imagen.close()
```

Tomando como referencia la imagen del logo del blog:



(https://pythoneyes.wordpress.com/2017/07/28/filtros-para-la-deteccion-de-bordes-de-una-imagen-con-python-3-parte-1/logo_derivada/)



(https://pythoneyes.wordpress.com/2017/07/28/filtros-para-la-deteccion-de-bordes-de-una-imagen-con-python-3-parte-1/logo_derivada1/)

En las imagenes anteriores se puede apreciar que los bordes son detectados en diferentes direcciones debido al signo de los coeficientes como se habia mencionado ya.

Imagen resultado de la combinacion
de las dos imagenes anteriores

Al combinar las dos imagenes podemos apreciar que los bordes se han detectado satisfactoriamente, sin embargo el resultado tiende a ser tenue por lo que es comun realizar un pre-procesamiento con mejora de Nitidez (hare un post de ese tema) o Linearizacion del Histograma (https://pythoneyes.wordpress.com/2017/06/01/ecualizacion-lineal-del-histograma/); aunque tambien podriamos servirnos de un post-procesamiento con Laplace (https://pythoneyes.wordpress.com/2017/06/20/filtro-de-laplace-realce-de-bordes-en-imagenes-con-python-3/) o Binarizacion (https://pythoneyes.wordpress.com/2017/05/24/segmentacion-por-umbral-binarizacion-de-una-imagen/) de la imagen.

Imagen pre-procesada con mejora
de Nitidez + Linearizacion del
Histograma y Deteccion de Bordes
por primera Derivada.

Peace

-Raziel

28 julio, 201729 agosto, 2017 · Publicado en Deteccion, Filtros, Sin categoría · Etiquetado como border detection, derivada, derivative, filter python 3, gradiente, imagen, pil, pillow, python ·

Un comentario en “Filtros para la deteccion de bordes de una imagen con Python 3 (Parte 1)”

1. Pingback: Filtros para la deteccion de bordes de una imagen con Python 3 (Parte 2) – Python's eyes