Visión por Computadora 2024

Lab 02

14.marzo.2024

En esta práctica vamos a implementar el algoritmos *Histogram of Gradients* (HOG) de Dalal y Triggs, desde *scratch*. Para ello, se utilizarán las librerías de scikit-image y OpenCV.

1. Implementar una función en Python que acepte una imagen a colores, y haga las siguientes transformaciones:

   - conversión de RGB a escala de grises,
   - reescalado (*resize*) a un tamaño $(128k, 64k)$, (ratio 2:1).

   En este caso $k \geq 1$ es un parámetro de escala que da el tamaño de la imagen de salida, y debe ser indicado por el usuario. La salida debe ser una imagen en formato 8 bits o float de tamaño $(128k, 64k)$, esto es un numpy arrar de tamaño $(128k, 64k)$.

2. Implementar una función en Python que calcule la magnitud del y ángulo del gradiente de una imagen en escala de grises. Internamente su algoritmo debe calcular las correlaciones G_x y G_y de la imagen con los filtros de Prewitt

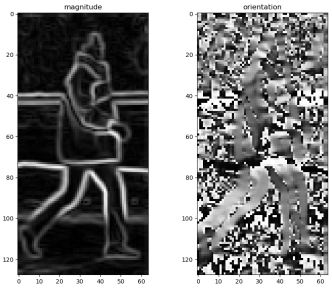
$$\nabla_x = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \qquad \nabla_y = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}.$$

   (no olvidar hacer un *padding* adecuado en los bordes de la imagen a filtrar, y dar una salida del mismo tamaño de la imagen de entrada).

   En este caso la entrada es una imagen en escala de grises I. La salida de la función deben ser dos imágenes μ y θ dadas por

$$\mu = \sqrt{G_x^2 + G_y^2} \qquad \text{y} \qquad \theta = \arctan 2(-G_y, G_x) \cdot \frac{180}{\pi},$$

   ambas del mismo tamaño de la imagen de entrada I.
   (Considerar que se debe hacer módulo 180 en la imagen θ para que el rango de los ángulos obtenidos sea siempre $0 \leq \theta < 180$).



3. Implementar una función en Python que construya el stack de bloques de tamaño `block_size` $\times$ `block_size`, usando un `stride` de `block_size`$//2$. En este caso, `block_size` debe ser un parámetro indicado por el usuario (se sugiere que sea un múltiplo de 4, *e.g.* 4×4, 8×8, 16×16).

   La entrada de la función debe ser la imagen μ y θ obtenidas del gradiente, y su función debe devolver dos stack o numpy arrays de tamaño $MN \times$`block_size` $\times$ `block_size`, uno para la magnitud y otros para el ángulo del gradiente. Aquí

$$M = \frac{128k}{\texttt{stride}} - 1 \qquad \text{y} \qquad N = \frac{64k}{\texttt{stride}} - 1.$$

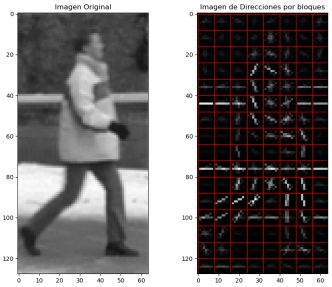
4. Implementar una función en Python que construya el histograma de gradientes, a partir de los stacks de bloques μ y θ. La construcción del histograma debe recibir un número de bines n indicado por el usuario (se sugiere usar $n = 9$). Internamente la función deberá calcular los centros de cada bin, y hacer un barrido de cada uno de los bloques en los stacks. Para cada bloque, debe calcularse un vector (numpy array de tamaño n) donde se guardará la información de su respectivo histograma.

   La salida de la función debe ser un numpy array H de tamaño $MN \times n$ con la información de los cada histograma. Los histogramas deberán normalizarse utilizando la norma euclideana:

$$H[i] = \frac{H[i]}{\sqrt{|| \, H[i] \, ||^2 + \varepsilon}}.$$

5. Escribir una función en python que a cada bloque en el stack, convierta el histograma de ese bloque a un representación visual. Esta representación visual será una matriz de tamaño `block_size` $\times$ `block_size`. Esta imagen se trabaja como una imagen con fondo negro. Por cada índice en el histograma de dicho bloque, $i = 1, 2, \ldots, n$, se dibujará una línea con ángulo en el centroide C_i del bin i, y de magnitud o intensidad igual al valor del histograma en el bin i.

   Construir una imagen de direcciones que dibuje los histogramas visuales de cada bloque. (Aquí se sugiere usar un `block_size` $=$ 8 y un `stride` $=$ `block_size` $= 8$). Generar una visualización similar a



6. Comparar sus resultados del algoritmo anterior (Ejercicios 1 a 5), contra la función de scikit-image `hog`.

```
from skimage.feature import hog
fd, hogI = hog(I, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True).


plt.figure()
plt.imshow(hogI)
plt.show()
```

7. Implementar un método de detección de personas, utilizando la librería de OpenCV. Mostrar resultados de su detección con diferentes imágenes de su elección.

```
import cv2
hog2 = cv2.HOGDescriptor()
hog2.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

locations, confidence = hog2.detectMultiScale(I)

for (x, y, w, h) in locations:
    cv2.rectangle(I, (x, y), (x + w, y + h), (0, 0, 255), 5)
I_rgb = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(20,10))
plt.imshow(I_rgb)
plt.axis('off')
plt.show()
```
