



universidad
de león



Visión Artificial - Conteo y aforo

Prácticas externas

Alumno: Sergio Chimeno Alegre

Grado en ingeniería informática



Índice

Datos de la práctica	3
Análisis de estrategias	5
Estrategia #1: Haar Cascade Classifier	5
Estrategia #2: Background subtraction method	8
Estrategia #3: HOG + SVM	10
Bibliografía	12



Datos de la práctica

Título del proyecto: Visión Artificial - Conteo y aforo

Empresa: Proconsi

Tutores:

- David García Prieto - david.garcia@proconsi.com
- David Jáñez Gonzalez – david.janez@proconsi.com



Objetivos

Familiarizarse con el lenguaje Python y la librería OpenCV

Decidir la estrategia a seguir para realizar el conteo de personas, tras haber analizando pros y contras de cada una de ellas.

Implementar un sistema capaz de contar personas en una determinada zona mediante cámara de vídeo preferentemente cenital (por ejemplo desde un drone). Debe funcionar en diferentes escenarios (playas, calles, conciertos.....)

Análisis de estrategias

Estrategia #1: Haar Cascade Classifier

Para esta estrategia usaremos [Haar feature-based cascade classifier](#) para detectar objetos, en este caso personas.

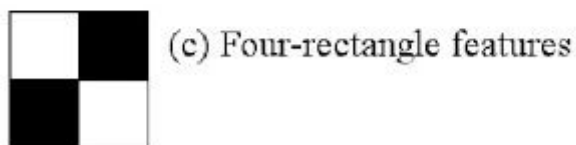
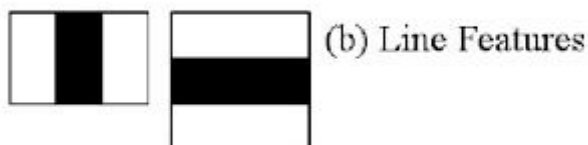
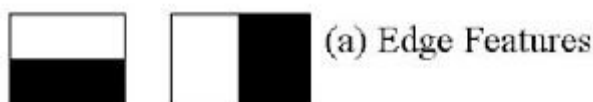
Este es un metodo basado en machine learning, donde una función cascada es entrenada a partir de un dataset de imagenes para clasificar imágenes como positivas(si contienen el tipo de objeto para el que han sido entrenadas) o negativas. Una vez entrenado el clasificador se puede utilizar para detectar objetos.

Opencv ya nos provee una serie de modelos preentrenados:

<https://github.com/opencv/opencv/tree/master/data/haarcascades> ; uno de estos modelos(haarcascade_fullbody.xml) puede ser útil para la detección de personas.

Este clasificador funciona con imagenes en escala de grises. Este algoritmo hace uso de: Haar Features, Integral images, adaboost y cascading.

Haar features: Son equivalentes a aplicar convolutional kernels, como los de la imagen de abajo, donde la región negra de la matriz son -1 y la región blanca son 1; es decir, es como restar la parte negra de la parte blanca. Estos kernels se aplicarán en múltiples tamaños a lo largo de la imagen.

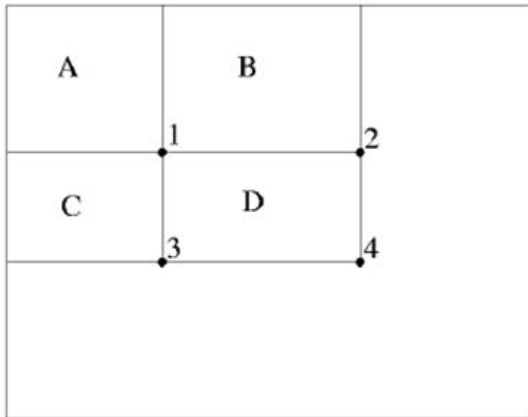


Puesto que aplicar estos kernels requieren mucho coste computacional(muchas operaciones) se introduce el uso de las **integrales de imágenes**, que son un método que permite encontrar la suma de todos los píxeles bajo un rectángulo, con solo saber los puntos de las esquinas del rectángulo.

En una integral image el valor del pixel(x,y) es la suma de todos los pixeles que están por encima y a la izquierda.

La integral de una imagen solo es necesario calcularla una vez.

Una vez se tiene la integral de una imagen la suma de pixeles se realiza así:



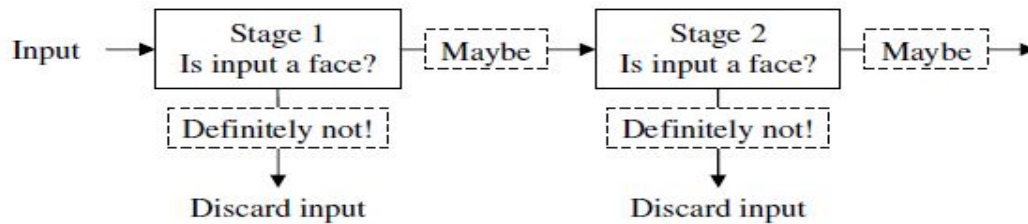
$$\begin{aligned} D &= 1 + 4 - (2 + 3) \\ &= A + (A + B + C + D) - (A + C + A + B) \\ &= D \end{aligned}$$

En una imagen de 24+24 habría 160000 Haar features. Pero la mayoría de estas características son irrelevantes. Por lo tanto se hace una subselección de las mejores Haar features con **Adaboost**, que hace lo siguiente:

1. Entrena una serie de clasificadores débiles(un clasificador débil es aplicar una Haar feature y decidir a partir de una threshold) y elige los mejores.
2. Crea un clasificador fuerte como una weighted sum de clasificadores débiles

Por último, se usa **cascading**, que lo que hace es aplicar los clasificadores débiles en grupos, de manera que si no se cumple con lo mínimo requerido por un grupo de clasificadores débiles se pueda descartar esta clasificación(marcar como negativo) y continuar con otra, ahorrando así coste computacional. Pero si la imagen cumple con lo requerido se pasa al siguiente grupo de clasificadores débiles, de manera que si se llega hasta el final se clasifica como positivo.

La cascada tiene una cantidad de stages(grupos de clasificadores débiles), que se deben de indicar para hacer el entrenamiento.



Detección multiescala:

Básicamente [arrastra una ventana de un cierto tamaño por toda la imagen](#).

El tamaño de la ventana es el tamaño con el que hemos entrenado el clasificador.

Por cada iteración o posición de la ventana, se toma esa región como la entrada al clasificador; el clasificador nos dirá si detecta o no el objeto.

Al acabar de arrastrar la ventana por toda la imagen, la imagen es reducida según el parámetro **scaleFactor** y se vuelve a arrastrar la ventana desde el comienzo de la imagen.

El parámetro **minNeighbors** es el número de veces que tiene haber sido detectado el objeto en alguna de las ventanas para devolvernos un positivo.

Entrenamiento:

Ventajas:

- Nos permite saber la posición exacta de las personas
- Puede adaptarse a múltiples escenarios si el clasificador está bien entrenado
- Es un método de detección de objetos rápido (ya era fluido con los dispositivos que había en 2004)
- Ya hay modelos entrenados

Desventajas:

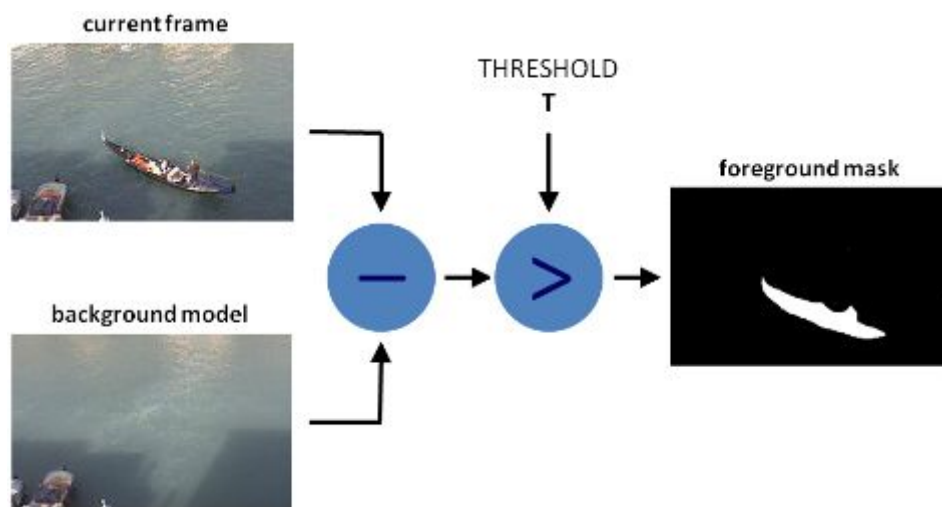
- Puede que las Haar features sólo se adapten a uno o varios tipos de escenario, no siendo generalizables a todas las posiciones o ángulos.

Estrategia #2: Background subtraction method

Background subtraction es un metodo que nos permite generar una máscara, de los objetos en primer plano (foreground mask)

Algoritmo de background subtraction:

1. Estimar el fondo (background)
2. Obtener el valor absoluto de la resta entre el fondo estimado y el frame actual
3. Aplicar la función threshold con un valor T , para obtener la máscara como una imagen binaria



Puede haber multiples formas de estimar el fondo:

- El frame anterior al actual
- La media de los n ultimos frames
- ...

Opencv nos ofrece MOG2 y KNN como background subtractors

Algunos parámetros que podemos modificar son:

- `varThreshold`: (T) el valor del treshold
- `detectShadow`: si queremos que detecte las sombras

Además de esto habría que limpiar algo de ruido con transformaciones morfológicas o blurs.

Y por último decidir cómo diferenciar una persona de cualquier otro objeto en movimiento. Es decir, como diferenciar la máscara de un coche de la de una persona.

Un enfoque simple sería considerar como persona cuando hay un cierto rango de pixeles blancos juntos.



Otro enfoque sería usar un clasificador cascada y entrenarlo con máscaras de personas

Ventajas:

- Funciona bien para detectar objetos que se mueven, por ejemplo para contar el número de personas que cruzan una raya puede ser útil
- Puede funcionar en cámaras en movimiento(dron) si se actualiza la estimación del background

Desventajas:

- Con objetos estáticos, puede que funcione peor, por ejemplo si una persona no se mueve de un sitio será más difícil detectarla, ya que esta persona será detectada como parte del fondo
- Si dos personas están juntas puede ser difícil diferenciarlas
- Se pueden confundir otros objetos que se mueven con personas

Estrategia #3: HOG + SVM

Este método consiste en obtener un descriptor de la imagen, que sería el histograma de gradientes, y luego pasarlo por una SVM para que lo clasifique. El HOG se puede ver como un preprocesamiento de la imagen para extraer características de los bordes de los objetos. La idea a la hora de entrenar es juntar imágenes positivas y negativas y preprocesarlas a través del algoritmo HOG. Los vectores de características resultantes serán con los que se entrene el SVM.

HOG(Histogram Of Gradients): HOG trata de extraer características de los bordes y esquinas de los objetos que hay una imagen a partir de obtener las diferencias de intensidad entre píxeles próximos, y la dirección de este cambio en intensidad. Es decir, nos da la magnitud del gradiente y la dirección del mismo.

Para ello divide la imagen en celdas de 8x8 píxeles, para cada una de estas celdas se calcula un feature vector:

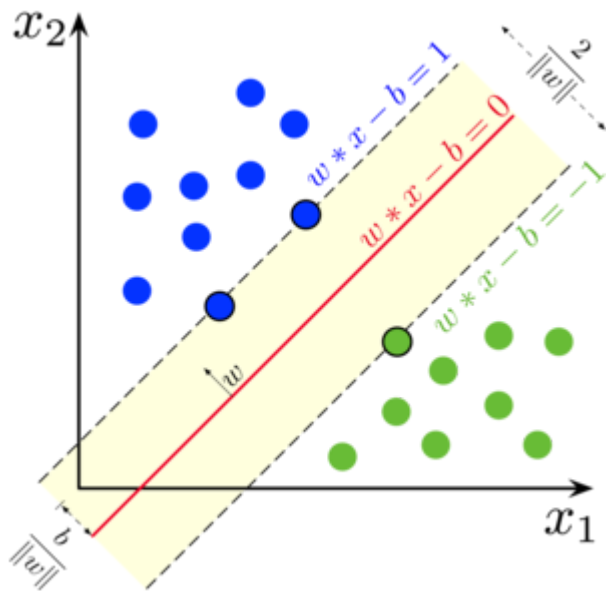
1. Se hallan los gradientes y sus direcciones para cada pixel
2. Se forma un vector, que es realmente un histograma dividido en compartimentos por grados, el valor de cada compartimento es la magnitud de los gradientes con esos grados.

Luego se forman matrices que contienen 4 celdas de 8x8 y se concatenan los feature vector de cada una de las celdas.

Por último cada feature vector se normaliza.

SVM(Support Vector Machine): Es un clasificador lineal, que trata de separar linealmente los vectores de características en el espacio.

Además de buscar separar linealmente los vectores de características, trata de maximizar el margen de espacio entre ambos conjuntos de datos



Como se ve en la imagen los puntos de cada color son un conjunto, y el SVM trata de maximizar el margen entre ellos (zona amarillenta).

Ventajas:

- Opencv ya viene con modelos preentrenados para detectar peatones en imágenes
- Con el hog de la imagen suele ser suficiente para identificar a una persona

Desventajas:

- La separación lineal que nos ofrece el SVM puede que no sea una hipótesis lo suficientemente compleja como para hacer una buena clasificación y se produzca underfitting
- Puede que para una vista centinela sea difícil distinguir el HOG de una persona del de otro objeto, como por ejemplo un árbol

Posibles mejoras:

- Cambiar el clasificador SVM por una red neuronal



Bibliografía

- https://docs.opencv.org/master/d9/df8/tutorial_root.html
- <https://www.youtube.com/watch?v=WfdYYNamHZ8>
- https://en.wikipedia.org/wiki/Support_vector_machine
-