Detección de rostros con filtros de Haar.

Jorge Luis Vargas Barrera.
Centro de Investigación en Matemáticas. Unidad Monterrey
Email: jorge.vargas@cimat.mx

***Resumen*—En este documento estudiaremos la detección de rostros usando filtros de Haar. En particular estudiaremos el algoritmo de Viola-Jones [2], el primer sistema de detección de rostros en tiempo real. Los tres ingredientes necesarios para obtener una detección precisa y rápida son: la imagen integral para el calculo de las características de Haar, Adaboost para la selección de características y un clasificador por cascada para un rápido descarte de imágenes que no contienen rostros.**

I. Introducción

Un detector de rostros tiene que identificar si una imagen de tamaño arbitrario contiene un rostro humano o no, y donde se encuentra. Un marco natural para considerar este problema es el de la clasificación binaria, en donde se construye un clasificador para reducir el error de una clasificación incorrecta. Debido a que ninguna distribución puede describir la probabilidad a priori de que una imagen tenga rostro, el algoritmo debe minimizar los errores de falsos negativos y falsos positivos para lograr un desempeño aceptable. Esta tarea requiere un descripción numérica adecuada de que características separan a los rostros humanos de otros objetos. Resulta ser que estas características pueden ser extraídas con el algoritmo Adaboost, el cual se basa en un conjunto de clasificadores débiles los cuales forman un clasificador fuerte mediante un mecanismo de votos. Decimos que un clasificador es débil si, en general, no puede cumplir un objetivo determinado en términos del error.

II. Metodología

II-A. Características de Haar e Imagen integral.

El algoritmo de Viola-Jones usa las características de Haar,esto es, un producto escalar entre la imagen y un filtro de Haar. Más precisamente, denotemos a I y P como una imagen y un patrón, ambos de tamaño $N \times N$ (ver Figura 1). La característica asociada con el patrón P de la imagen I se define como

$$\sum_{1\leq i\leq N}\sum_{1\leq j\leq N} I(i,j)1_{P(i,j) \text{ es blanco}} - \sum_{1\leq i\leq N}\sum_{1\leq j\leq N} I(i,j)1_{P(i,j) \text{ es negro}}$$

Para compensar el efecto de diferentes condiciones de luz, todas las figuras se normalizan.

((a)) ((b)) ((c))

Figura 1: Características de Haar. Solamente los píxeles marcados en blanco y negro son usados cuando la correspondiente característica es calculada.

((a)) ((b)) ((c)) ((d)) ((e))

Figura 2: Cinco patrones de Haar. El tamaño y la posición del soporte de un patrón puede variar siempre y cuando los rectángulos negros y blancos tengan la misma dimensión, frontera entre ellos y mantienen sus posiciones relativas. Gracias a estas restricciones, el número de características que se pueden obtener de una imagen de 24×24 son 43200, 27600, 43200, 27600 y 20736 características de los cinco patrones.Es decir, 162336 características en total.

En la práctica, 5 patrones son considerados (ver Figura 2). Se asume que las características obtenidas contienen toda la información que caracteriza al rostro. Si embargo, hay otro crucial elemento que lleva al uso de estas características: la imagen integral permite calcularlas con un costo computacional muy bajo. En lugar de sumar todos los píxeles dentro de las ventanas rectangulares, esta técnica se asemeja al uso de funciones de distribución. La imagen integral II de I es

$$II(i,j) := \begin{cases} \sum_{1 \le s \le i} \sum_{1 \le t \le j} I(s,t) & , \quad 1 \le i \le N \; y \; 1 \le j \le N \\ 0 & , \quad \text{de otra forma} \end{cases}$$

Como resultado, calcular la suma en un rectángulo de la imagen requiere a lo mas cuatro operaciones elementales dada la imagen integral (ver Figura 3). Además, obtener la imagen integral puede hacerse en tiempo lineal usando el Algoritmo 1

Algorithm 1: Integral Image

```
Set II(1,1)=I(1,1);
for i=1 to N do
    for j=1 to M do
        II(i,j)=I(i,j)+II(i,j-1)+II(i-1,j)-II(i-1,j-
          1);

        II es definido como cero cuando (i,j)
          están fuera de I;
    end
end
```

Claramente la diferencia entre dos sumas rectangulares puede ser calculada con ocho referencias. Dado que las características con dos rectángulos definidas arriba involucran sumas de rectángulos adyacentes estas pueden ser calculadas en seis referencias, ocho en el caso de las características con tres rectángulos, y nueve para las características con cuatro rectángulos.

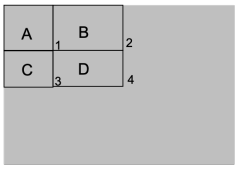


Figura 3: La suma de los píxeles dentro del rectángulo D pueden calcularse con 4 referencias a la imagen integral. El valor de la imagen integral en la ubicación 1 es la suma de de los píxeles en el rectángulo A. El valor en la ubicación 2 es A+B, en la ubicación 3 es A+C, y en la ubicación 4 es A+B+C+D. La suma solo de D puede calcularse como 4+1-(2+3).

II-B. Selección de características con AdaBoost.

Dado un conjunto de entrenamiento de imágenes positivas y negativas, podríamos usar cualquiera de los métodos de clasificación que se vieron en el curso para aprender una función de clasificación. Recordemos que se tienen más de 160000 características de Haar asociadas con cada sub-ventana en una imagen. Aún cuando se vio en la sección anterior que el calculo de estas características se puede realizar de forma eficiente, calcularlas todas resulta computacionalmente caro. En el clasificador de Viola-Jones se utiliza una variante de AdaBoost para seleccionar un conjunto pequeño de características y entrenar el clasificador. Recordemos que AdaBoost se compone de un conjunto de clasificadores débiles, en este algoritmo el clasificador débil esta diseñado para seleccionar una sola característica que mejor separe los ejemplos positivos y negativos. Para cada característica, el clasificador débil determina la función del umbral de clasificación, tal que el mínimo número de ejemplos sean mal clasificados. Por lo que un clasificador débil $h_j(x)$ consiste de una característica f_j, un umbral θ_j y una polaridad p_j indicando la dirección de la desigualdad:

$$h_j(x) = \begin{cases} 1 & si \; p_j f_j(x) < p_j \theta_j \\ 0 & \text{en otro caso} \end{cases}$$

Donde x es una sub-ventana de 24×24 de una imagen. El resumen del proceso de selección de características se encuentra en el Algoritmo 2

En [2] los autores muestran que para la tarea de detección las características seleccionadas inicial-

mente por AdaBoost son significativas y fáciles de interpretar. La primera característica parece basarse en que la zona de los ojos es usualmente mas oscura que la región de las mejillas y la nariz (ver Figura 4). La segunda característica depende en la propiedad que los ojos son más oscuros que el puente de la nariz.

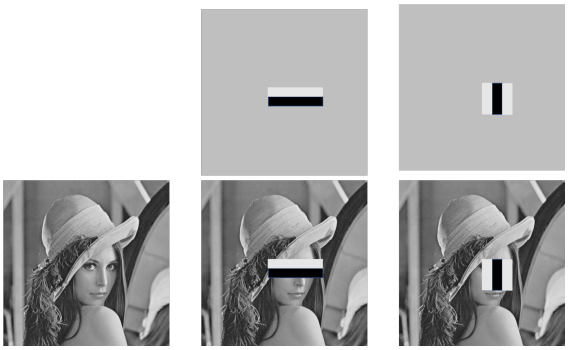


Figura 4: Las primera y segunda característica seleccionada por AdaBoost. La primera característica mide la diferencia en intensidad entre las regiones de los ojos y las mejillas. La segunda característica las intensidades en la región de los ojos y la intensidad alrededor del puente de la nariz.

Algorithm 2: AdaBoost.

Dadas ejemplos de imágenes $(x_1, y_1), \ldots, (x_n, y_n)$ donde $y_i = 0, 1$ para ejemplos negativos y positivos respectivamente;

Inicializa los pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ para $y_i = 0, 1$ respectivamente, donde m y l son el número de negativos y positivos respectivamente;

for $t = 1, \ldots, T$ **do**

Normaliza los pesos,

$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

;

Para cada característica, j, entrena un clasificador h_j que se restringe a una sola característica. El error con respecto a $w_t, \epsilon_t = \sum_i w_i |h_j(x_i) - y_i|$;

Escoge el clasificador $_t$ con el menor error ϵ_t;

Actualiza los pesos:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

donde $e_i = 0$ si el ejemplo x_i se clasifica correctamente, $e_i = 1$ en otro caso, y $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$;

El clasificador final es:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{en otro caso} \end{cases}$$

donde $\alpha_t = \log \frac{1}{\beta_r}$

end

II-C. Clasificador por cascada.

Ahora veremos un algoritmo para construir una cascada de clasificadores los cuales logran un desempeño mayor de detección y al mismo tiempo reducen el tiempo computacional. La clave es que clasificadores boost mas pequeños, y por lo tanto mas eficiente pueden se construidos y a la vez rechazar muchas de las sub-ventanas negativas mientras se detectan casi todas las instancias positivas (i.e. el umbral del clasificador boost puede ser ajustado de tal forma que el porcentaje de falsos negativos sea cercano a cero). Clasificadores más simples son usados para rechazar la mayoría de las sub-ventanas antes de que clasificadores más complejos sean llamados para para lograr bajos porcentajes de falsos positivos. EL proceso de detección es el de un árbol de decisión degenerado, al cuál se le llama *cascada* (ver Figura 5). Un resultado positivo en el primer clasificador activa la evaluación en el segundo clasificador el cual también fue ajustado para obtener altos porcentajes de detección. Un resultado positivo del segundo clasificador activa un tercer clasificador y así sucesivamente. Un resultado negativo en cualquiera de los clasificadores lleva a un rechazo de la sub-ventana.

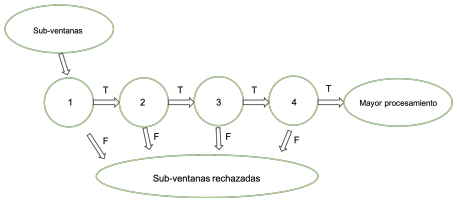


Figura 5: Esquema que muestra la cascada de clasificadores. Una serie de clasificadores es aplicado a cada sub-ventana. El clasificador inicial elimina un gran número de ejemplos negativos con muy poco procesamiento. Las siguientes camas eliminan más negativos, pero requieren más cálculos. Después de varias etapas de procesamiento el número de sub-ventanas se reduce radicalmente. La ultima etapa pueden ser más clasificadores o algún sistema de detección diferente.

Cada etapa de la cascada es construida al entrenar los clasificadores usando AdaBoost y después se ajusta el umbral para disminuir los falsos negativos. Recordemos que AdaBoost tradicional esta diseñado para obtener errores bajos en el conjunto de entrenamiento. En general, un umbral más bajo conlleva mayores niveles de detección y mayor número de falsos positivos.

II-C1. Entrenamiento de los clasificadores.: Dada una cascada de clasificadores, la proporción de falsos positivos de la cascada es

$$F = \prod_{i=1}^{K} f_i$$

donde F es la proporción de falsos positivos en el clasificador por cascada, K es el número de clasificadores, y f_i es la proporción de falsos positivos en el clasificador i-ésimo. Para la proporción de detección.

$$D = \prod_{i=1}^{K} d_i$$

donde D es la proporción de detección en el clasificador por cascada, K es el número de clasificadores, y d_i es la proporción de detección en el i-ésimo clasificador.

Dada metas concretas para las proporciones de falsos positivos y de detección para el clasificador. Estos objetivos se pueden ir determinando en cada etapa del proceso de cascada. El proceso por el cual se entrena a a cada elemento de la cascada debe de realizarse con cuidado. El algoritmo 2 solamente minimiza los errores, y no esta diseñado para lograr altos valores de detección y a la vez obtener altas proporciones de falsos positivos. El esquema que se propone en [2] es un esquema de compromiso entre estos errores al ajustar el umbral del perceptrón producido por AdaBoost. Altos umbrales conllevan a clasificadores con menores falsos positivos y menor valores de detección. Umbrales bajos resultan en clasificadores con más falsos positivos y mayor nivel de detección. Sin embargo no es claro si AdaBoost mantienen las características que lo distinguen como es la garantía de generalización y entrenamiento. El proceso de entrenamiento involucra dos tipos de compromiso. En la mayoría de los casos clasificadores con mas características obtendrán mayores niveles de detección y menor número de falsos positivos. Pero al mismo tiempo estos clasificadores requieren mayor tiempo de computo. Podemos definir un marco de optimización en donde

- el número de etapas de clasificación,
- el número de características, n_i, de cada etapa,
- el umbral de cada etapa

son comprometidos para buscar minimizar el valor esperado de características a evaluar N dado los objetivos F y D, donde

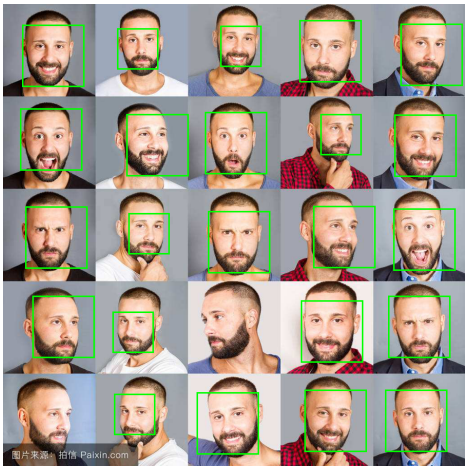
$$N = n_0 + \sum_{i=1}^{K} \left(n_i \prod_{j<i} p_j \right)$$

K es el número de clasificadores, p_i es la proporción de positivos del i-ésimo clasificador, y n_i es el número de características en el i-ésimo clasificador.

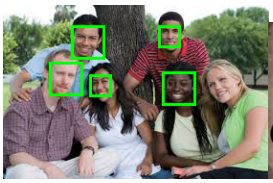
En la práctica se usa un método muy simple es usado para producir un clasificador efectivo con alta eficiencia. En cada etapa de la cascada se reduce la proporción de falsos positivos y decrece la proporción de detección. Un objetivo es seleccionado para la menor reducción de falsos positivos y un máximo decrecimiento de detección. Cada etapa es entrenada al agregar características hasta que los objetivos de detección y el nivel de falsos positivos se satisfacen (estos niveles se determinan al probar el detector en un conjunto de validación).

III. Implementación y Resultados.

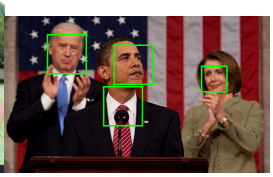
Como un primer análisis exploratorio del algoritmo de Viola-Jones se programo en Python un script donde se utiliza la función de *cv.CascadeClassifier()* con un clasificador preentrenado de openCV para la detección de rostros, *haarcascade_frontalface_alt.xml*, obteniendo muy buenos resultados e tiempo real al usar la webcam de mi computadora. También se probó con otras imágenes obtenidas de la web. Cabe destacar que este método falla al ponerse de perfil hacia la cámara así que es sensible a a la posición del rostro (ver Figura 6).



((a))



((b))



((c))

Figura 6: Resultado de la detección de rostros usando el algoritmo de Viola-Jones en la implementación de OpenCV.

Además utilizando una implementación en Python[1] se probó la precisión del algoritmo de Viola-Jones en un conjunto de datos[2]. Del cual se tomaron 2429 imágenes de caras, 2689 imágenes que no contienen caras. Por la limitante del tiempo se entrenaron dos modelo uno con 10 características de Haar y otro con 50 características de Haar. Obteniendo un accuracy de 85.5 % en el conjunto de entrenamiento y un 78 % en el conjunto de prueba para T=10 características. Para el modelo con T=50 características el modelo logró 91 % de accuracy en ambos conjuntos de entrenamiento y de prueba. El código se pueden encontrar en el repositorio del proyecto[3]

[1]https://github.com/ZihengZZH/viola-jones

[2]http://cbcl.mit.edu/software-datasets/FaceData2.html

[3]https://github.com/jlvb0/viola-jones_CD.git

IV. Conclusiones

El algoritmo desarrollado por Paul Viola y Michael Jones, fue el primero en su tipo, y revoluciono el campo de procesamiento de imágenes. A la fecha sigue siendo uno de los algoritmos más poderosos en detección de rostros. Introduce nuevos algoritmos, representaciones, los cuales aportan herramientas poderosas para al campo de visión computacional y procesamiento de imágenes. Cabe destacar el uso de AdaBoost como un discriminador para obtener características que ayuden a decidir de forma eficiente que sub-ventanas requieren mayor tiempo de preprocesamiento. Además del clasificador en cascada que es fundamental para lograr resultados en tiempo real.

Referencias

[1] Alan Julian Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. 1.ª ed. Springer Publishing Company, Incorporated, 2008. ISBN: 0387781889.

[2] Paul Viola y Michael Jones. *Rapid object detection using a boosted cascade of simple features*. 2001.

[3] Yi-Qing Wang. "An Analysis of the Viola-Jones Face Detection Algorithm". En: *Image Processing On Line* 4 (2014). https://doi.org/10.5201/ipol.2014.104, págs. 128-148.