Tutorial Viola-Jones

Modesto Fernando Castrillón Santana

14 de octubre de 2009

1. Introducción

OpenCV [2] incorpora[1], gracias a la implementación de R. Lienhart [3], las funciones necesarias tanto para entrenar un clasificador basado en el marco general de detección de objetos de Viola-Jones [5, 6] como para su uso. Este documento pretende describir las experiencias adquiridas por su autor en la elaboración de distintos clasificadores integrados en la distribución de OpenCV desde su versión 2.0. Una fuente indispensable para la consecución de las experiencias alcanzadas en este tutorial ha sido [4].

2. Creación de un clasificador

2.1. Creación del conjunto de datos de entrenamiento

Para crear una cascada basada en el método de Viola-Jones, para cualquier tipo de objeto, es necesario construir dos conjuntos de imágenes independientes [2]:

1. Conjunto de muestras positivas:
   - Conjunto de imágenes conteniendo muestras del objeto que deseamos detectar, es decir, aquel para el que diseñamos el clasificador. Este conjunto de imágenes es necesario para que el comando **createsamples** genere el conjunto de imágenes de entrenamiento en el formato

[1] Este tutorial ha sido desarrolllado trabajando sobre la versión 1.1

[2] Nota: Para usar los comandos indicados a continuación es necesario tener la ruta de instalación de OpenCV en la variable de entorno PATH, p.e. en un entorno Windows C:
Archivos de programa
OpenCV
bin)

esperado por el comando **haartraining**, que como indicaremos más adelante será quien realice efectivamente el cálculo del clasificador. Entre sus parámetors podemos destacar:

- *info*: permite especifica el fichero que lista las imágenes conteniendo el patrón de interés.
- *vec*: especifica el nombre del fichero de muestras resultante.
- *w* y *h*: ancho y alto respectivamente de las muestras positivas a generar.
- *num*: número de muestras a generar. Existen posibilidades de generación de muestras realizando pequeñas alteraciones, deformaciones y transformaciones sobre las aportadas originalmente, como se muestra en los ejemplos más abajo.

- Las imágenes positivas proporcionadas, listadas en el fichero *pos.txt* en el ejemplo aquí mostrado, no necesariamente deben ser todas del mismo tamaño, pero se sugiere que estén bastante alineadas, será **createsamples** quien las escale en base a los parámetros *w* y *h* proporcionados.
- El siguiente ejemplo toma la lista de muestras positivas contenidas en el fichero de texto *pos.txt*, extrae las ventanas conteniendo al objeto, las transforma al tamaño 20×20 generando 10000 imágenes que almacena en *samples.vec*

```
createsamples.exe -info pos.txt  -vec data\clasif\samples.vec   -
    num 10000 -w 20 -h 20
```

- Cada fila del fichero de muestras positivas, *pos.txt* en el ejemplo, localiza una imagen que contiene una o varias muestras del objeto.
- Tras la ruta de la imagen se indica el número de muestras positivas del patrón de interés contenidas en la imagen y el contenedor de cada una en el formato: Coordenadas x e y de la esquina superior izquierda, seguidas del ancho y alto del contenedor, ver por ejemplo la Figura 1 para un ejemplo concreto. De este forma el fichero de texto estaría conformado por líneas como:

```
samples\subject01_centerlight.png 1 126 88 121 137
samples\subject01_glasses.png 1 116 80 121 137
```

- Tras crear el fichero de muestras, en el ejemplo *samples.vec*, y ejecutar **createsamples** es posible visualizar las muestras positivas creadas una vez realizada la transformación con la opción *vec* del comando **createsamples**

Figura 1: Ejemplo de imagen tomada de la base de imágenes faciales de Yale [1] cuya línea en el fichero de muestras positivas sería algo como *samples\subject*01_*centerlight.png* 1 126 88 121 137. El contenedor del objeto, mostrado en la imagen inferior, ha sido estimado a partir de una anotación de los ojos, también mostrada en la imagen inferior.

```
createsamples -vec data\clasif\samples.vec -w 20 -h 20    %
    Visualiza el fichero .vec de muestras positivas
```

- **createsamples** permite además distorsionar las que tenemos para obtener un número mayor de muestras positivas a partir de un grupo inicial de imágenes más reducido.
- El ejemplo a continuación crea 10 muestras positivas a partir del patrón positivo contenido en *face.png*. Las diversas muestras se transforman de forma aleatoria en base a los parámetros indicados, usándose las imágenes contenidas en *negatives.txt* como fondo y generando un fichero de muestras positivas con sus contenedores denominado *test.dat*

```
\% Creación de muestras de entrenamiento distorsionadas a partir de
    una imagen
createsamples -img face.png -num 10 -bg negatives.txt -vec samples.
    vec -maxxangle 0.6 -maxyangle 0 -maxzangle 0.3 -maxidev 100 -
    bgcolor 0 -bgthresh 0

\% Creación de muestras de test distorsionadas a partir de una
    imagen
createsamples -img face.png -num 10 -bg negatives.txt -info test.
    dat -maxxangle 0.6 -maxyangle 0 -maxzangle 0.3 -maxidev 100 -
    bgcolor 0 -bgthresh 0
```

2. Conjunto de muestras negativas:

- De forma análoga a las muestras positivas, usamos un fichero texto para indicar la localización de cada muestra negativa.
- Cada fila del fichero de texto localiza una imagen sin presencia del patrón a buscar, es decir, contiene la ruta de la imagen. Se sugiere utilizar imágenes grandes estilo fondo de escritorio para poder obtener un dominio mayor de falsos positivos, ver por ejemplo la Figura 2.

```
.\samples\KIF_0201.jpg
.\samples\ocaso2.JPG
```

2.2. Entrenamiento

Una vez obtenidos y anotados los conjuntos de muestras de imágenes positivas y negativas, podemos proceder al entrenamiento. Para ello necesitamos hacer uso del comando **haartraining**, que dispone de las siguientes opciones:

Figura 2: Ejemplso de imágenes del conjunto de muestras negativas empleadas para detectores de rostros humanos, las imágenes originales tienen 2272 × 1704 píxeles

```
-data: Localización del clasificador resultante
-vec: Muestras positivas
-bg: Fichero indicando las imágenes negativas
-npos y -nneg: Muestras positivas y negativas a usar
-nstages: Etapas a calcular
-mode: ALL usa las features rotadas añadidas por Lienhart
-w y -h: Indican el ancho y alto de las muestras
-sym: Si se usa asume que el patrón es simétrico
-nsplit: número de árboles por etapa
-minhitrate: Mínima tasa de acierto en positivas
-maxfalsealarm: Máxima tasa de error en negativas



haartraining.exe -data data\clasif -vec \%Posvec\% -bg \%Neg\% -npos \%NUMPOS
    \% -nneg \%NUMNEG\% -nstages \%STAGES\%  -maxtreesplits 0 -mem 500 -mode
    ALL -w \%WIDTH\% -h \%HEIGHT\% -sym
```

2.3. Test

Para probar la bondad del clasificador hacemos uso del comando **performance**:

```
performance -data <localización del clasificador>    -info  <muestras de test>
```

3. Errores

Errores registrados en el foro, que deberían estar corregidos en las versiones más recientes:

- Con muestras no cuadradas
- Usando *performance*.

4. Ejercicio propuesto

- Desarrollar un detector basado en Viola-Jones tras escoger un objeto determinado a detectar
- Componer la base de imágenes positivas, quizás requiriendo anotación
- Componer la base de imágenes negativas
- Entrenar
- Usar el proyecto ViolaDetectorDemo para ver su funcionamiento en vivo.

Repositorio clasificadores:

- Alejandro Reimondo
- SIANI

4.1. Un ejemplo

Un ejemplo con la base de imágenes de Yale citeBelhumeur97 seguiría lso siguientes pasos para desarrollar un clasificador ejemplo que detecte rostros:

- Obtener muestras positivas
- Obtener muestras de imágenes negativas, interesan imágenes como fondos de pantalla sin el patrón del objeto de interés
- Anotar las positivas si no estuvieran ya anotadas, o traducir los ficheros de anotación al formato esperado por **createsamples**
- Configurar estructura de ficheros de datos

El siguiente código permite anotar el contenedor o recuadro de una muestra positiva empleado Matlab:

```
\%Leemos la imagen
img = imread('CaraPromedio.jpg');
imshow(uint8(img));

disp('Haga clic en las esquinas del contenedor de la cara');
[x y]=ginput(2);

hold on;
plot([round(x(1)) round(x(1)) round(x(2)) round(x(2)) round(x(1))],[round(y
    (1)) round(y(2))    round(y(2)) round(y(1))  round(y(1))],'-','Color','g
    ');
hold off;
```

El lanzador del entrenamiento para este pequeño ejemplo atiende a la estructura de directorios mostrada en la figura 3. Su cálculo no requiere más de unos segundos en un ordenador actual, y puede ejecutarse empleando el siguiente código:

```
set WIDTH=20
set HEIGHT=23
set NUMPOS=94
set NUMNEG=400
set STAGES=10

echo Estructura de ficheros y carpetas
set Clasif=data\classifier\
```

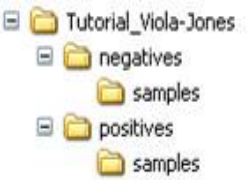

Figura 3: Estructura de directorios

```
9  set Posfile=positives\trainpos.txt
10 set Posvec=data\positives.vec
11 set Neg=negatives\negs.txt
12 set Createlog=logcreate.txt
13 set Trainlog=logtraining.txt
14
15 echo Creamos la carpeta
16 md %Clasif%
17
18 echo Creamos las muestras positivas a partir de un listado de imágenes con el
       contenedor correspondiente
19 createsamples.exe -info %Posfile% -vec %Posvec% -num %NUMPOS% -w %WIDTH% -h %
       HEIGHT% > %Createlog%
20
21 echo Lanzamos el entrenamiento
22 echo haartraining.exe -data %Clasif% -vec %Posvec% -bg %Neg% -npos %NUMPOS% -
       nneg %NUMNEG% -nstages %STAGES% -maxtreesplits 0 -mem 500 -mode ALL -w %
       WIDTH% -h %HEIGHT% -sym > %Trainlog%
```

4.2. En ejemplo más elaborado

Un detector creado con pocas muestras positivas y negativas adolecerá de dos problemas: 1) detectará poco al objeto de interés, y 2) detectará numerosos falsos positivos. Sin embargo, cuando estemos desarrollando un clasificador, es muy interesante obtener una primera impresión de lo que el clasificador puede ofrecer. Si se ha conseguido una cierta tasa de detección y el entrenamiento no se *colgado*, podemos ir más allá. En caso contrario, recomiendo jugar con los parámetros hasta mejorar las sensaciones con el primer clasificador simple. Una vez que las sensaciones sean positivas podemos abordar hacer un cómputo más elaborado, aumentando el número de muestras positivas y negativas. El siguiente código muestra el fichero tanda empleado para obtener los detectores de ojo derechi, izquierdo, nariz y boca incluidos en la última versión de OpenCV:

```
1 echo OJO DERECHO, destacar que tanto para el derecho como para el izquierdo no
       se indica como patrón simétrico
2 set WIDTH=18
3 set HEIGHT=12
4 set NUMPOS=14172
```

```
5   set NUMNEG=18000
6
7   echo Especificación de configuración de directorios y ficheros
8   set Clasif=data\classifier_OjoD\
9   set Posfile=SIANI\samplesOjoD.txt
10  set Posvec=data\positives_classifier_OjoD.vec
11  set Neg=negatives\train\train.txt.txt
12  set Createlog=logcreate_OjoD.txt
13  set Trainlog=logtraining_OjoD.txt
14
15  md %Clasif%
16  createsamples.exe -info %Posfile% -vec %Posvec% -num %NUMPOS% -w %WIDTH% -h %
        HEIGHT% > %Createlog%
17  haartraining.exe -data %Clasif% -vec %Posvec% -bg %Neg% -npos %NUMPOS% -nneg
        %NUMNEG% -nstages %STAGES% -maxtreesplits 0 -mem 500 -mode ALL -w %WIDTH
        % -h %HEIGHT% > %Trainlog%
18
19  echo OJO IZQUIERDO
20
21  set Clasif=data\classifier_OjoI\
22  set Posfile=SIANI\samplesOjoI.txt
23  set Posvec=data\positives_classifier_OjoI.vec
24  set Neg=negatives\train\train.txt.txt
25  set Createlog=logcreate_OjoI.txt
26  set Trainlog=logtraining_OjoI.txt
27
28  md %Clasif%
29  createsamples.exe -info %Posfile% -vec %Posvec% -num %NUMPOS% -w %WIDTH% -h %
        HEIGHT% > %Createlog%
30  haartraining.exe -data %Clasif% -vec %Posvec% -bg %Neg% -npos %NUMPOS% -nneg
        %NUMNEG% -nstages %STAGES% -maxtreesplits 0 -mem 500 -mode ALL -w %WIDTH
        % -h %HEIGHT% > %Trainlog%
31
32  echo BOCA
33  set WIDTH=25
34  set HEIGHT=15
35  set NUMPOS=7072
36  set NUMNEG=15000
37
38  set Clasif=data\classifier_Boca\
39  set Posfile=SIANI\samplesBoca.txt
40  set Posvec=data\positives_classifier_Boca.vec
41  set Neg=negatives\train\train.txt
42  set Createlog=logcreate_Boca.txt
43  set Trainlog=logtraining_Boca.txt
44
45  echo NARIZ
46  set WIDTH=18
47  set HEIGHT=15
48  set NUMPOS=7085
49  set NUMNEG=15000
50  set STAGES=20
51
52  set Clasif=data\classifier_Nariz\
53  set Posfile=SIANI\samplesNariz.txt
54  set Posvec=data\positives_classifier_Nariz.vec
55  set Neg=negatives\train\train.txt
56  set Createlog=logcreate_Nariz.txt
57  set Trainlog=logtraining_Nariz.txt
```

Referencias

[1] P.N. Belhumeur, J.P. Hespanha, and D.J. Kriegman. Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Trans. on PAMI*, 19(7):711–720, 1997. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.3247`.

[2] Intel. Intel Open Source Computer Vision Library, v1.1ore. http://sourceforge.net/projects/opencvlibrary/, October 2008.

[3] Rainer Lienhart and Jochen Maydt. An extended set of Haar-like features for rapid object detection. In *IEEE ICIP 2002*, volume 1, pages 900–903, September 2002.

[4] Naotoshi Seo. Tutorial: Opencv haartraining (rapid object detection with a cascade of boosted classifiers based on haar-like features). `http://note.sonots.com/SciSoftware/haartraining.html`.

[5] Paul Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, pages 511–518, 2001.

[6] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):151–173, May 2004.