Ciencia y Tecnología

Secretaría de Ciencia, Humanidades, Tecnología e Innovación

CIMAT
UNIDAD MÉRIDA

DETECCIÓN DE OBJETOS USANDO MÉTODOS TRADICIONALES

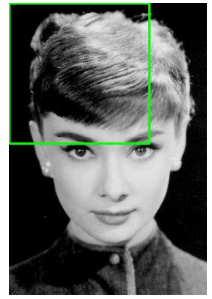
Dr. Francisco J. Hernández López
SECIHTI – CIMAT-Mérida
fcoj23@cimat.mx, www.cimat.mx/~fcoj23

DESARROLLO DE MODELOS DE DET. DE OBJ. TRADICIONALES

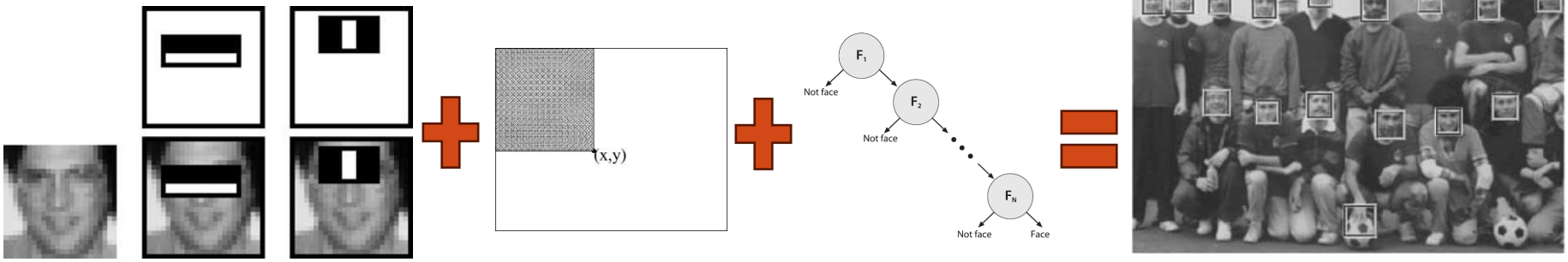
- **Selección de regiones informativas.** Se utilizan ventanas deslizantes a diferentes escalas para buscar a los diferentes objetos en toda la imagen. Debido a la gran cantidad de ventanas candidatas, esto puede resultar comp. pesado y producir muchas ventanas redundantes.
- **Extracción de características.** Necesitamos caract. visuales que puedan proveer una representación robusta, por ej.: SIFT, HOG, Haar-like. Debido a la diversidad de apariencia, condiciones de iluminación, fondos de los escenarios, etc., es difícil diseñar de forma manual caract. que describan a todos los tipos de obj.
- **Clasificación.** Necesitamos distinguir a un objeto a partir de diversas categorías y hacer las representaciones más jerárquicas, semánticas e informativas para el reconocimiento visual. Normalmente se utilizan: SVM, AdaBoost y DPM.

Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, *30*(11), 3212-3232.

DETECTOR DE VIOLA & JONES, 2001, 2004



- Detector de rostros humanos.
- Velocidad: 700 MHz en una CPU Pentium III.
- Forma de detección: Con ventanas deslizantes se recorren todas las posibles posiciones y escalas en la imagen para ver si alguna ventana contiene un rostro.



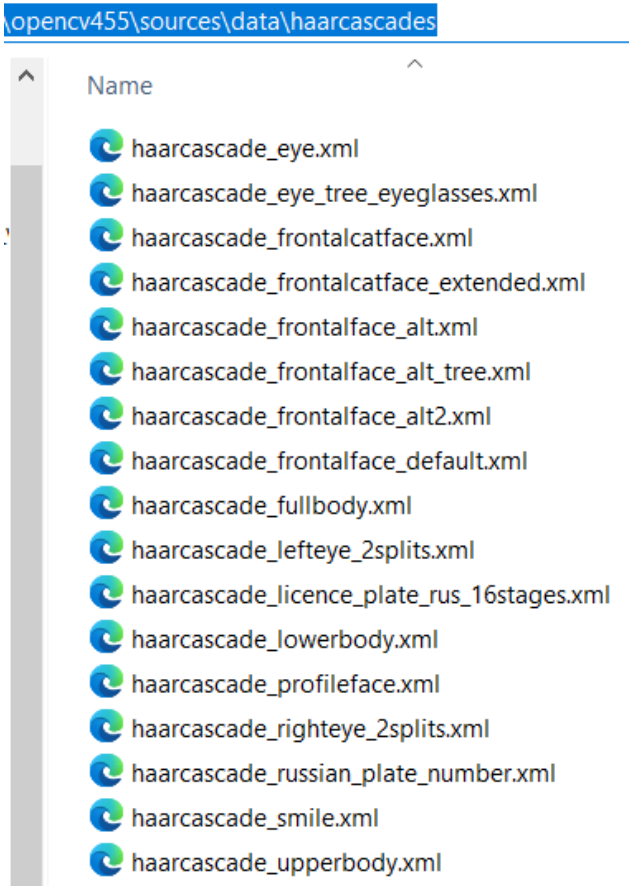
Características Haar | Imagen integral | Clasificador | Detección de rostros

Viola, P., & Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001* (Vol. 1, pp. I-I). Ieee.

Viola, P., & Jones, M. J. (2004). Robust real-time face detection. *International journal of computer vision*, *57*, 137-154.

DETECCIÓN DE ROSTROS EN OPENCV

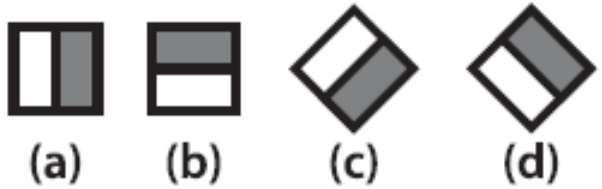
- En OpenCV existe una implementación del algoritmo de Viola & Jones, el cual fue luego extendido por Lienhart & Maydt.
- OpenCV llama a esta implementación el "*Haar classifier*".
- Además de detectar rostros, también es posible usar el método para detectar otros tipos de objetos rígidos como: autos, bicis, cuerpo humano, etc.
- Los objetos ya preentrenados que están disponibles en OpenCV están en la carpeta: /opencv/data/haarcascades



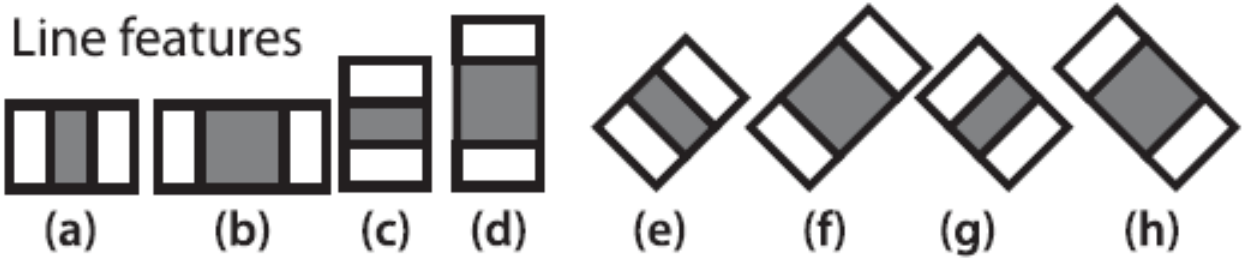
Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.".
Lienhart, R., & Maydt, J. (2002, September). An extended set of haar-like features for rapid object detection. In *Proceedings. international conference on image processing* (Vol. 1, pp. I-I). IEEE.

CARACTERÍSTICAS TIPO HAAR

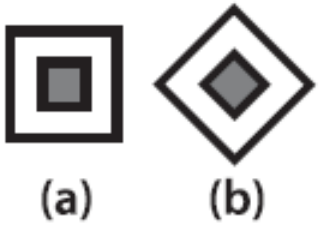
1. Edge features



2. Line features



3. Center-surround features

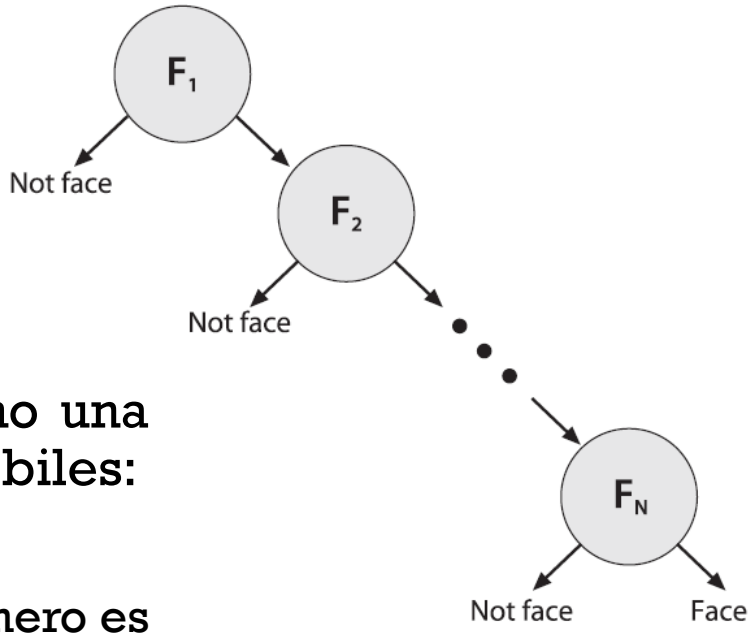


Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc."

CLASIFICADOR

- Utiliza un clasificador de impulso (*boosting*), en el cual los clasificadores débiles en cada nodo son árboles de decisión que frecuentemente tienen solo un nivel de profundidad, por ej.:
  - "¿Es el valor v de una característica f por arriba o debajo de un umbral T?"
    - Si → Rostro
    - No → No rostro

$$f_i = \begin{cases} +1 & v_i \geq T_i \\ -1 & v_i < T_i \end{cases}$$



- Se construye un clasificador como una suma pesada de clasificadores débiles:

$$F = sign(w_1 f_1 + w_2 f_2 + \cdots + w_n f_n)$$

La función $sign$ devuelve -1 si el número es < 0, 0 si es igual a 0 y $+1$ si el número es > 0

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc."

CÓDIGO FACEDETECT.CPP DE OPENCV

```
38  int main( int argc, const char** argv )
39  {
40      VideoCapture capture;
41      Mat frame, image;
42      string inputName;
43      bool tryflip;
44      CascadeClassifier cascade, nestedCascade;
45      double scale;
46
47      cv::CommandLineParser parser(argc, argv,
48          "{help h||}"
49          "{cascade|data/haarcascades/haarcascade_frontalface_alt.xml|}"
50          "{nested-cascade|data/haarcascades/haarcascade_eye_tree_eyeglasses.xml|}"
51          "{scale|1|}{try-flip||}{@filename|1|}"//En filename podemos cambiar el ID del puerto
52                                               //para elegir la WebCam
53      );
```

Podemos modificar el modelo pre-entrenado

CARGANDO LOS MODELOS

```
59      cascadeName = parser.get<string>("cascade");
60      nestedCascadeName = parser.get<string>("nested-cascade");
61      scale = parser.get<double>("scale");
62      if (scale < 1)
63          scale = 1;
64      tryflip = parser.has("try-flip");
65      inputName = parser.get<string>("@filename");
66      if (!parser.check())
67      {
68          parser.printErrors();
69          return 0;
70      }
71      if (!nestedCascade.load(samples::findFileOrKeep(nestedCascadeName)))
72          cerr << "WARNING: Could not load classifier cascade for nested objects" << endl;
73      if (!cascade.load(samples::findFile(cascadeName)))
74      {
75          cerr << "ERROR: Could not load classifier cascade" << endl;
76          help(argv);
77          return -1;
78      }
```

PROCESANDO IMÁGENES (WEBCAM)

```
110     if( capture.isOpened() )
111     {
112         cout << "Video capturing has been started ..." << endl;
113
114         for(;;)
115         {
116             capture >> frame;
117             if( frame.empty() )
118                 break;
119
120             Mat frame1 = frame.clone();
121             detectAndDraw( frame1, cascade, nestedCascade, scale, tryflip );
122
123             char c = (char)waitKey(10);
124             if( c == 27 || c == 'q' || c == 'Q' )
125                 break;
126         }
127     }
```

PROCEDIMIENTO PRINCIPAL

```
172  void detectAndDraw( Mat& img, CascadeClassifier& cascade,
173                      CascadeClassifier& nestedCascade,
174                      double scale, bool tryflip )
175  {
176      double t = 0;
177      vector<Rect> faces, faces2;
178      const static Scalar colors[] =
179      {
180          Scalar(255,0,0),
181          Scalar(255,128,0),
182          Scalar(255,255,0),
183          Scalar(0,255,0),
184          Scalar(0,128,255),
185          Scalar(0,255,255),
186          Scalar(0,0,255),
187          Scalar(255,0,255)
188      };
189      Mat gray, smallImg;
```

```
191     cvtColor( img, gray, COLOR_BGR2GRAY );
192     double fx = 1 / scale;
193     resize( gray, smallImg, Size(), fx, fx, INTER_LINEAR_EXACT );
194     equalizeHist( smallImg, smallImg );
195
196     t = (double)getTickCount();
197     cascade.detectMultiScale( smallImg, faces,
198         1.1, 2, 0
199         //|CASCADE_FIND_BIGGEST_OBJECT
200         //|CASCADE_DO_ROUGH_SEARCH
201         |CASCADE_SCALE_IMAGE,
202         Size(30, 30) );
203     if( tryflip )
204     {
205         //flip(smallImg, smallImg, 1);//Por si en la cámara o la foto el rostro está voltedo
206         flip(smallImg, smallImg, 0);//Ojo, el flip debería aplicarse c.r. al eje x
207                                     //es decir, voltear la imagen de forma vertical
208         cascade.detectMultiScale( smallImg, faces2,
209                                   1.1, 2, 0
210                                   //|CASCADE_FIND_BIGGEST_OBJECT
211                                   //|CASCADE_DO_ROUGH_SEARCH
212                                   |CASCADE_SCALE_IMAGE,
213                                   Size(30, 30) );
214         for( vector<Rect>::const_iterator r = faces2.begin(); r != faces2.end(); ++r )
215         {
216             //faces.push_back(Rect(smallImg.cols - r->x - r->width, r->y, r->width, r->height));
217             faces.push_back(Rect(r->x, smallImg.rows-r->y-r->height, r->width, r->height));
218             //Ojo, la transformación debería aplicarse c.r. al eje y
219         }
220     }
```

Detección de rostros

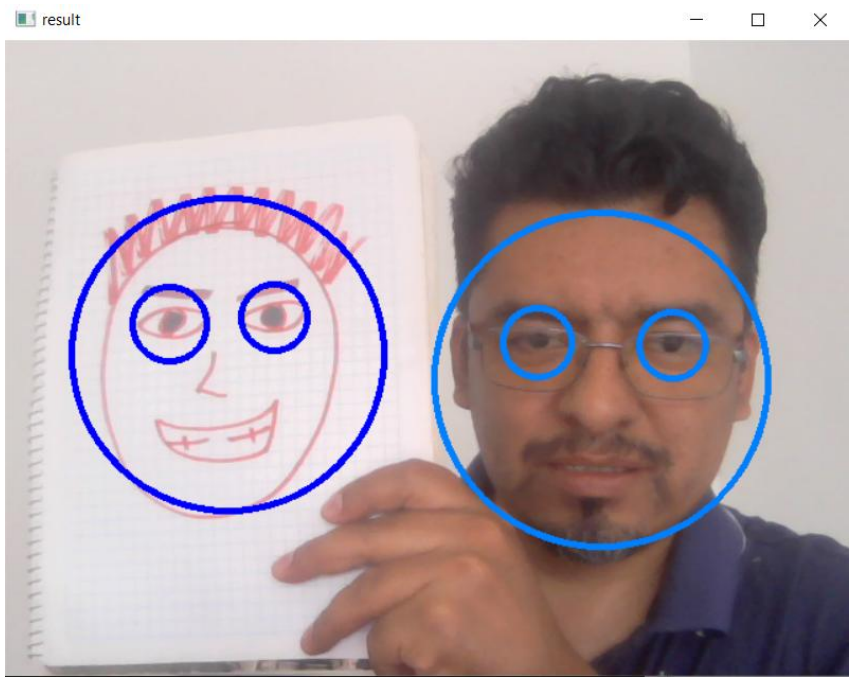
Detección de rostros volteados

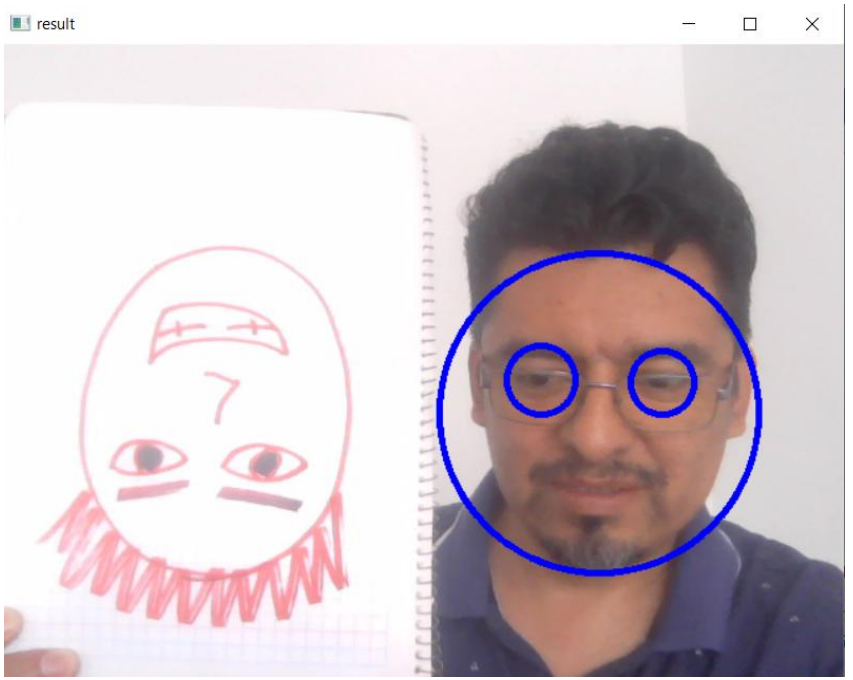
```
217     t = (double)getTickCount() - t;
218     printf( "detection time = %g ms\n", t*1000/getTickFrequency());
219     for ( size_t i = 0; i < faces.size(); i++ )
220     {
221         Rect r = faces[i];
222         Mat smallImgROI;
223         vector<Rect> nestedObjects;
224         Point center;
225         Scalar color = colors[i%8];
226         int radius;
227 
228         double aspect_ratio = (double)r.width/r.height;
229         if( 0.75 < aspect_ratio && aspect_ratio < 1.3 )
230         {
231             center.x = cvRound((r.x + r.width*0.5)*scale);
232             center.y = cvRound((r.y + r.height*0.5)*scale);
233             radius = cvRound((r.width + r.height)*0.25*scale);
234             circle( img, center, radius, color, 3, 8, 0 );
235         }
236         else
237             rectangle( img, Point(cvRound(r.x*scale), cvRound(r.y*scale)),
238                        Point(cvRound((r.x + r.width-1)*scale), cvRound((r.y + r.height-1)*scale)),
239                        color, 3, 8, 0);
240         if( nestedCascade.empty() )
```

```
240         if( nestedCascade.empty() )
241             continue;
242         smallImgROI = smallImg( r );
243         nestedCascade.detectMultiScale( smallImgROI, nestedObjects,
244             1.1, 2, 0
245             //|CASCADE_FIND_BIGGEST_OBJECT
246             //|CASCADE_DO_ROUGH_SEARCH
247             //|CASCADE_DO_CANNY_PRUNING
248             |CASCADE_SCALE_IMAGE,
249             Size(30, 30) );
250         for ( size_t j = 0; j < nestedObjects.size(); j++ )
251         {
252             Rect nr = nestedObjects[j];
253             center.x = cvRound((r.x + nr.x + nr.width*0.5)*scale);
254             center.y = cvRound((r.y + nr.y + nr.height*0.5)*scale);
255             radius = cvRound((nr.width + nr.height)*0.25*scale);
256             circle( img, center, radius, color, 3, 8, 0 );
257         }
258     }
259     imshow( "result", img );
260 }
```

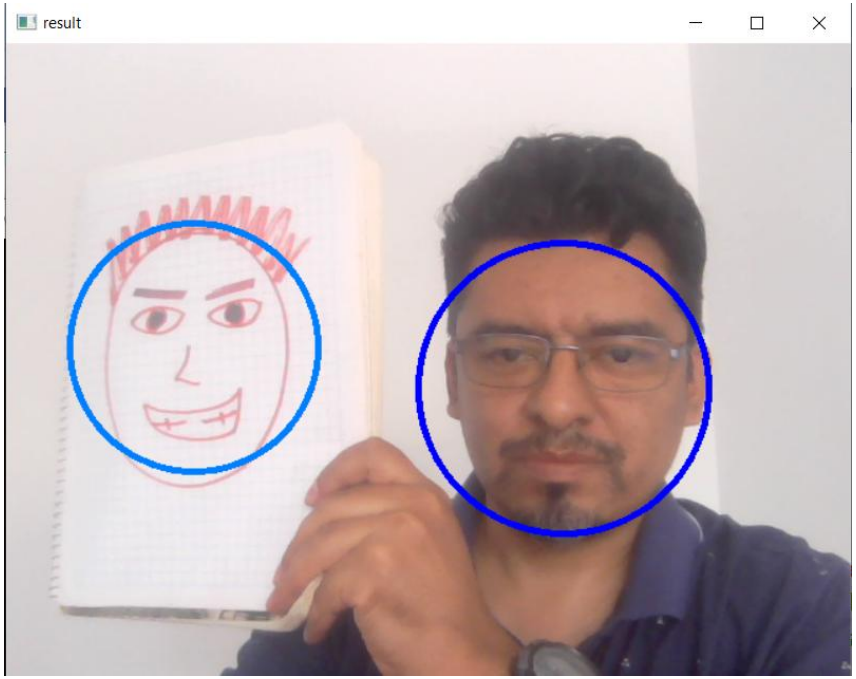
Detección de ojos

PRUEBA 1 {try-flip||}



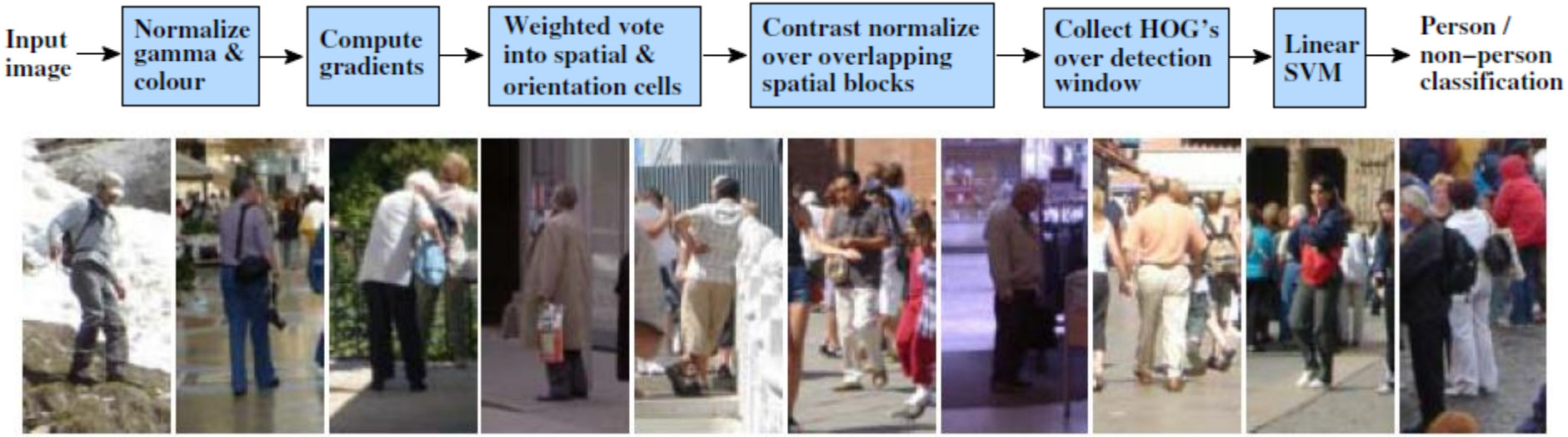


PRUEBA 2 {try-flip|1|}



DETECTOR HOG, DALAL & TRIGGS, 2005

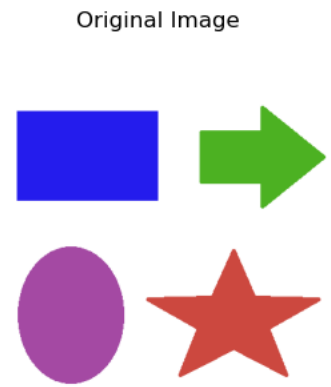
- HOG: Descriptor de características con base en el histograma de gradientes orientados.
- Aunque se puede usar para detectar una variedad de clases de objetos, fue motivado por el problema de detección de peatones.

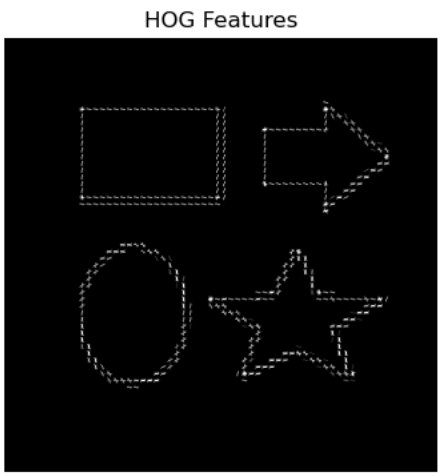


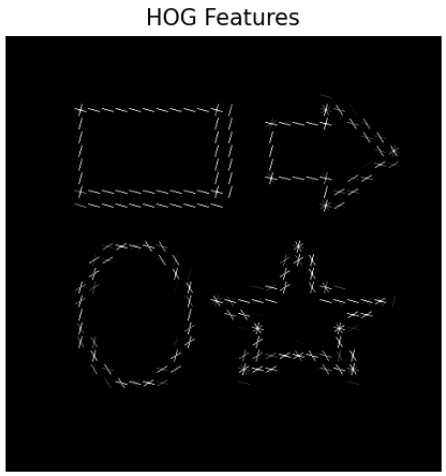
Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)* (Vol. 1, pp. 886-893). Ieee.

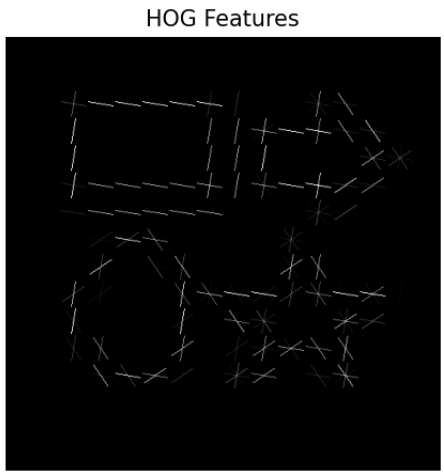
```python
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import data, exposure
from skimage.io import imread
#import cv2
image=imread("Figs1.png")
#image=cv2.imread("Figs1.png", cv2.IMREAD_COLOR)

# Compute HOG features
fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16,16),
                    cells_per_block=(1, 1), visualize=True, multichannel=True)
```









512×512

Celda=[8,8]

fd.shape= 32768

Celda=[16,16]

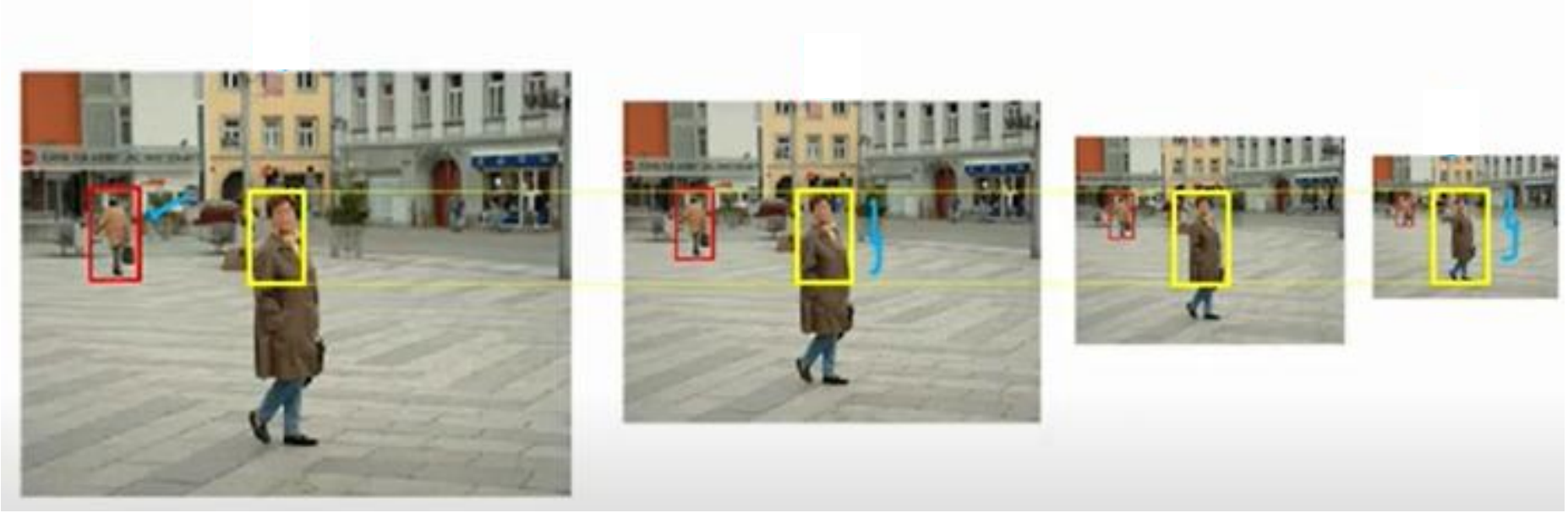
8192

Celda=[32,32]

2048

DETECTOR HOG MULTIESCALA

- Para detectar objetos de diferentes tamaños se re-escala la imagen de entrada múltiples veces mientras se mantiene el tamaño de la ventana de detección.



C37 | Dalal & Triggs Object Detection | HOG + SVM | Computer Vision | Machine Learning | EvODN

Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)* (Vol. 1, pp. 886-893). Ieee.

CÓDIGO PEOPLEDETECT.CPP DE OPENCV

```
5   #include <opencv2/objdetect.hpp>
6   #include <opencv2/highgui.hpp>
7   #include <opencv2/imgproc.hpp>
8   #include <opencv2/videoio.hpp>
9   #include <iostream>
10  #include <iomanip>
11
12  using namespace cv;
13  using namespace std;
15  class Detector
16  {
17      enum Mode { Default, Daimler } m;
18      HOGDescriptor hog, hog_d;
19  public:
20      Detector() : m(Default), hog(), hog_d(Size(48, 96), Size(16, 16),
21                                            Size(8, 8), Size(8, 8), 9)
22      {
23          hog.setSVMDetector(HOGDescriptor::getDefaultPeopleDetector());
24          hog_d.setSVMDetector(HOGDescriptor::getDaimlerPeopleDetector());
25          //Entrenado con el dataset Daimler
26      }
```

```
27      void toggleMode() { m = (m == Default ? Daimler : Default); }
28      string modeName() const { return (m == Default ? "Default" : "Daimler"); }
29      vector<Rect> detect(InputArray img)
30      {
31          // Run the detector with default parameters. to get a higher hit-rate
32          // (and more false alarms, respectively), decrease the hitThreshold and
33          // groupThreshold (set groupThreshold to 0 to turn off the grouping completely).
34          vector<Rect> found;
35          if (m == Default)
36              hog.detectMultiScale(img, found, 0, Size(8,8), Size(), 1.05, 2, false);
37          else if (m == Daimler)
38              hog_d.detectMultiScale(img, found, 0, Size(8,8), Size(), 1.05, 2, true);
39          return found;
40      }
41      void adjustRect(Rect & r) const
42      {
43          // The HOG detector returns slightly larger rectangles than the real objects,
44          // so we slightly shrink the rectangles to get a nicer output.
45          r.x += cvRound(r.width*0.1);
46          r.width = cvRound(r.width*0.8);
47          r.y += cvRound(r.height*0.07);
48          r.height = cvRound(r.height*0.8);
49      }
50  };
51
52  static const string keys = "{ help h    |    | print help message }"
53                             "{ camera c | 1 | capture video from camera (device index
54                             "{ video v  |    | use video as input }";
```

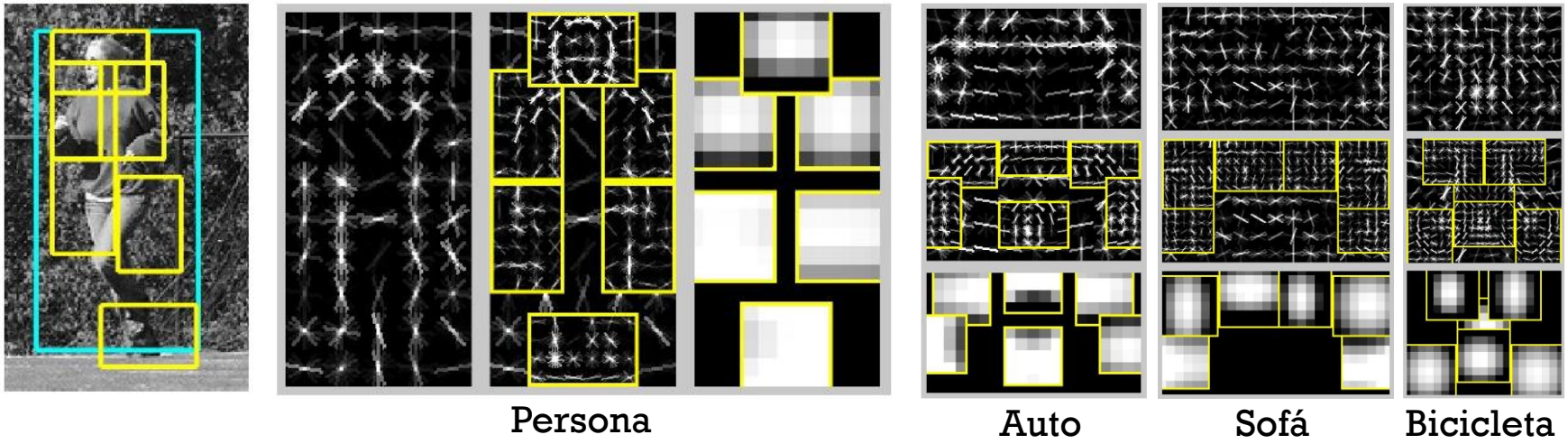
```
54  int main(int argc, char** argv)
55  {
56      CommandLineParser parser(argc, argv, keys);
57      parser.about("This sample demonstrates the use of the HoG descriptor.");
58      if (parser.has("help"))
59      {
60          parser.printMessage();
61          return 0;
62      }
63      int camera = parser.get<int>("camera");
64      string file = parser.get<string>("video");

85      cout << "Press 'q' or <ESC> to quit." << endl;
86      cout << "Press <space> to toggle between Default and Daimler detector" << endl;
87      Detector detector;
88      Mat frame;
89      for (;;)
90      {
91          cap >> frame;
92          if (frame.empty())
93          {
94              cout << "Finished reading: empty frame" << endl;
95              break;
96          }
97          int64 t = getTickCount();
98          vector<Rect> found = detector.detect(frame);
99          t = getTickCount() - t;
```

```
101         // show the window
102         {
103             ostringstream buf;
104             buf << "Mode: " << detector.modeName() << " ||| "
105                 << "FPS: " << fixed << setprecision(1) <<
106                 (getTickFrequency() / (double)t);
107             putText(frame, buf.str(), Point(10, 30),
108                 FONT_HERSHEY_PLAIN, 2.0, Scalar(0, 0, 255), 2, LINE_AA);
109         }
110         for (vector<Rect>::iterator i = found.begin(); i != found.end(); ++i)
111         {
112             Rect &r = *i;
113             detector.adjustRect(r);
114             rectangle(frame, r.tl(), r.br(), cv::Scalar(0, 255, 0), 2);
115         }
116         imshow("People detector", frame);
118         // interact with user
119         const char key = (char)waitKey(1);
120         if (key == 27 || key == 'q') // ESC
121         {
122             cout << "Exit requested" << endl;
123             break;
124         }
125         else if (key == ' ')
126         {
127             detector.toggleMode();
128         }
129     }
130     return 0;
131 }
```

MODELO DPM, FELZENSZWALB ET AL., 2008

- DPM: Modelo de partes deformables.
- Extensión del detector HOG.
- Sigue la filosofía de divide y vencerás. Por ej., para detectar una persona, el problema se puede dividir en detectar la cabeza, los brazos, el dorso y las piernas.



Felzenszwalb, P., McAllester, D., & Ramanan, D. (2008, June). A discriminatively trained, multiscale, deformable part model. In 2008 IEEE conference on computer vision and pattern recognition (pp. 1-8). Ieee.

GRACIAS POR SU ATENCIÓN

Francisco J. Hernández López

fcoj23@cimat.mx

WebPage:

www.cimat.mx/~fcoj23