Proyecto Final

Detección de personas mediante Histogramas de Gradientes Orientados

Farid Arbai Chentouf

03/05/18

Contenido

1. Introducción

En el presente proyecto se ha implementado la técnica de detección de personas en imágenes propuesta por Navneet Dalal y Bill Triggs en [1]. La finalidad del mismo es la de evaluar y analizar la propia técnica, por lo que se presentarán las bases de la misma, la implementación realizada y los resultados derivados.

En primer lugar, se expone una explicación de los distintos pasos seguidos en la presente técnica para lograr el reconocimiento de individuos en imágenes. Posteriormente, se procede a explicar la implementación realizada en Python de esta misma técnica. A continuación, se evalúa el modelo obtenido tanto numéricamente como empíricamente. Finalmente, se derivan las conclusiones obtenidas del desarrollo del presente proyecto y se referencias las fuentes bibliográficas en las que se basa el mismo.

2. Descripción de la técnica

En el presente algoritmo se parte de la suposición de que el contorno del cuerpo humano es un factor determinista para la detección del mismo. En este sentido, si se es capaz de extraer un vector de características que representen el contorno del mismo entonces será posible realizar una clasificación binaria de calidad aceptable.

Partiendo de la idea anterior, los autores se decantaron por el uso de *histogramas de gradientes orientados* para la caracterización del cuerpo humano. Esta técnica se basa en calcular un vector representativo de una imagen que de algún modo está relacionada con la distribución de los gradientes en la misma. Para ello, esta técnica se divide en dos fases: cálculo de histogramas y normalización.

2.1. Cálculo de histogramas

En esta primera fase se procesa la información relativa a los gradientes de la imagen en cuestión. Dado que el vector final de salida dependerá del tamaño de la imagen, esta misma habrá de estar previamente acotada. Para ello, se realiza un primer paso de recorte a un tamaño cuyo ratio de aspecto sea 1:2 para posteriormente reescalar al tamaño de análisis de 64x128. Este procedimiento se esquematiza en la figura Fig. 3, extraída del libro [3], representando la misma el ejemplo sobre el que realizaremos el análisis que sigue.

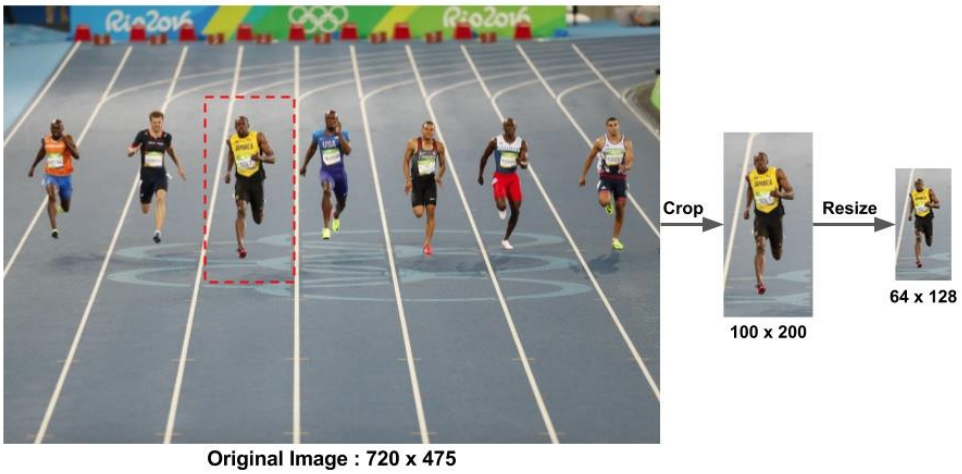


Fig. 1. Ejemplo de preprocesamiento de la imagen a analizar.

Una vez se posee la imagen en el tamaño de procesamiento, se procede al cálculo de las dos imágenes gradiente, correspondientes a sendas direcciones vertical y horizontal, tal y como se muestra en la figura Fig. 2. Nótese que el gradiente en este caso se representa a color, pues el mismo es

computado sobre los tres canales de la imagen con un filtro de Sobel sobre cada una de las dos direcciones.

Fig. 2. Cómputo del gradiente para la imagen analizada.

Una vez que se dispone de la información relativa al gradiente, se procede al cálculo de las imágenes magnitud y orientación, computándose estas por cada pixel para el canal RGB que mayor magnitud proporcione, pues se demuestra que para el presente problema esta técnica mejora notablemente los resultados [1]. Definidas las matrices finales de orientación y magnitud, se procede al cálculo del histograma orientado por regiones de 8x8 pixeles. De este modo, el análisis por cada región o celda queda restringuido a la misma, consiguiendo así describir la información de orientación del entorno. Una muestra de este procedimiento se muestra en la figura Fig. 3 donde el mismo ha sido aplicado al ejemplo aquí analizado.

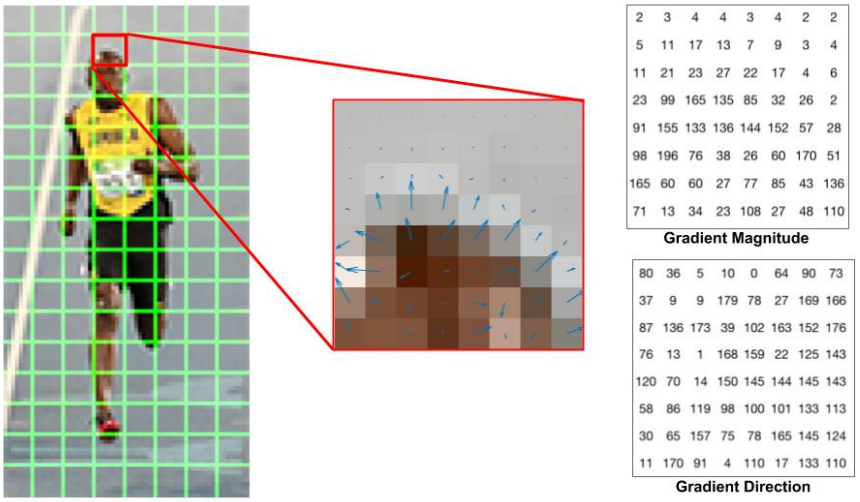


Fig. 3. Cálculo de la orientación y la magnitud del gradiente por cada celda.

El siguiente y último paso en esta primera fase es el cálculo del histograma de cada región como descriptor de la misma. Para ello, se recurre a un cálculo de histograma un poco especial. En primer lugar, se

define un total de 9 bins de 20º de anchura cada uno, lo que implica que el descriptor de esta matriz de 8x8 será un vector de 9 componentes. Como segundo paso, se realiza el cálculo del histograma con una serie de pasos adicionales. En primer lugar, la presencia de una orientación correspondiente a un bin no vota en el mismo bin una unidad de frecuencia si no que vota una unidad de magnitud, en particular, una magnitud proporcional a su propia magnitud. Esta magnitud que se añade al bin en cuestión resulta de una interpolación lineal en la que se reparte la magnitud de un pixel entre el bin al que pertece su orientación y el bin más próximo, consiguiendo así realizar un cálculo más exacto. Esto aparece expuesto en la figura Fig. 4, donde se observa que la magnitud del pixel cuya orientación es 80º se reparte completamente al bin al que pertence por estar dicha orientación centrada en el bin en cuestión, mientras que para el caso del pixel cuya orientación es de 10º su magnitud se reparte de forma proporcional mediante interpolación lineal a los dos bins más cercanos, el centrado en 0º y el centrado en 20º.

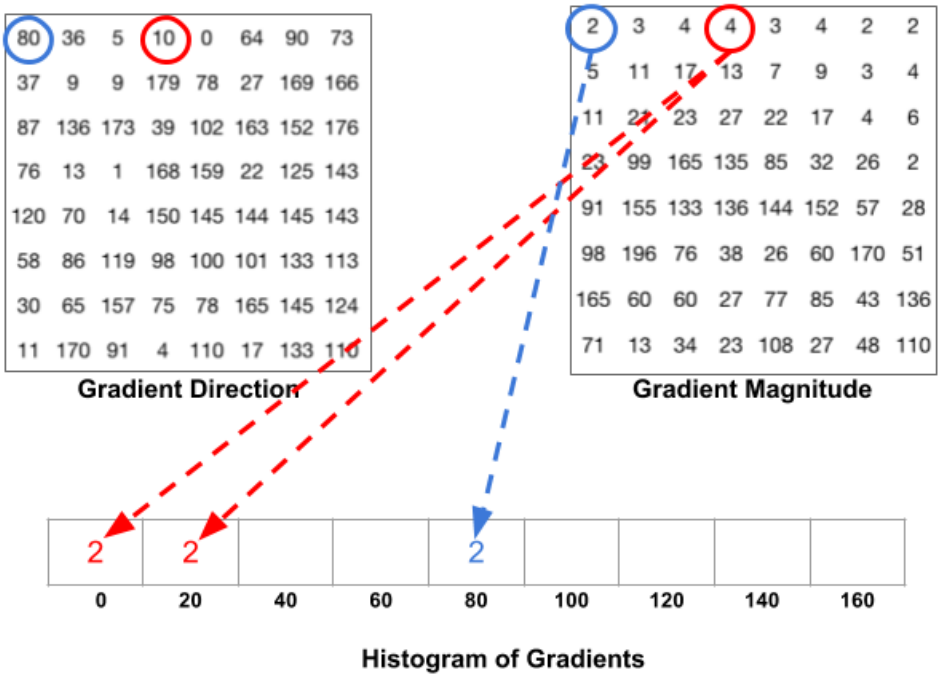


Fig. 4. Cómputo del histograma de gradientes orientados.

2.2. Normalización de histogramas

Una vez ejecutado el paso anterior se dispone de un descriptor de 9 componentes por cada bloque de 8x8 pixeles, el cual representa la información de los contornos dentro del mismo. No obstante, estos bloques pueden estar afectados por sesgos en su magnitud de unas imágenes a otras en función del entorno de captación de la imagen, por lo que resulta necesaria la normalización de este descriptor.

Para conseguir esto, los autores de [1] proponen el uso de bloques en los que se agrupen celdas sobre las que realizar la normalización de los histogramas. Para este fin, la norma L2-Hys es la que muestra mejores resultados, definida como la normalización de los histogramas pertenecientes al bloque por su norma L2 seguida de la saturación a 0.2 de los histogramas y el recómputo de la normalización por la nueva norma L2. Adicionalmente, los autores recomiendan el solapamiento de estos bloques en pasos de un 50% de su tamaño a fin de añadir información adicional al proceso. De este modo, el proceso de normalización se realiza con la secuencia esquematizada en la figura Fig. 5 para el tamaño de bloque de 2x2 celdas aconsejado.

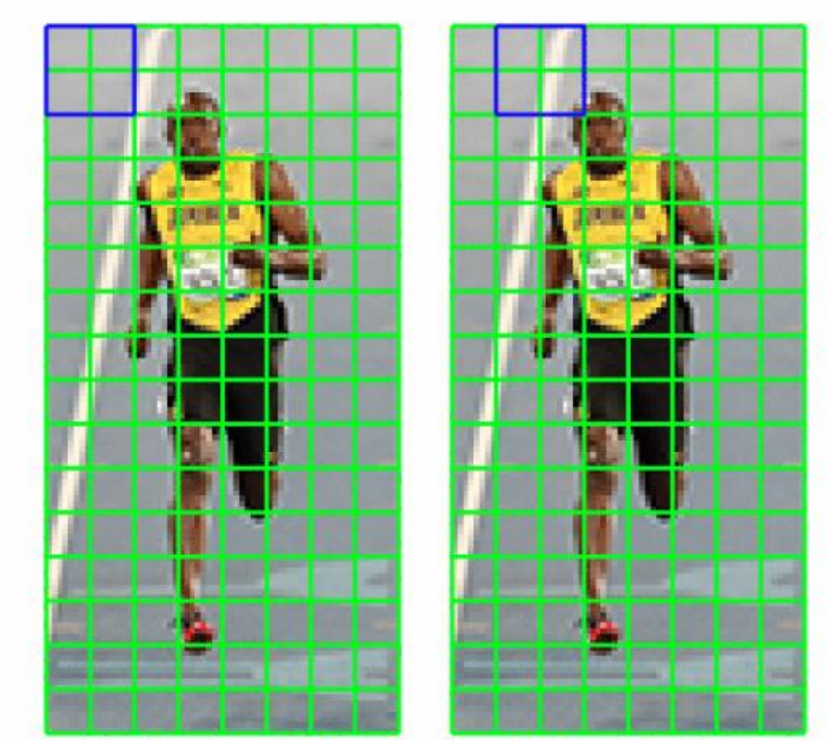


Fig. 5. Cómputo de la normalización de histogramas con solapamiento del 50%.

Ejecutados los pasos anteriores, obtendremos por cada zona de 8x8 su histograma correctamente normalizado en norma. De este modo, poseemos descriptores de los contornos a nivel regional por lo que el descriptor completo de la imagen se obtiene de la concatenación de todos estos descriptores regionales. Dado que los bloques de normalización iteran celda a celda y la imagen está compuesta por 8x16 celdas, en la misma se podrán realizar 7 iteraciones de normalización en horizontal y 15 en vertical, formando ello un total de 7x15 = 105 iteraciones de normalización. Por cada iteración se genera un bloque de 4 histogramas siendo cada uno de ellos un vector de 9 componentes, por lo que el resultado final se deriva en que el descriptor HOG de la imagen posee 105x4x9 = 3780 componentes.

La representación espacial del histograma puede visualizarse en la figura Fig. 6, donde por cada celda se añaden las direcciones con una longitud proporcional a su magnitud ya normalizada. Como se puede apreciar, estas direcciones tienden a seguir los contornos de su alrededor, por lo que la concatenación de todos estos descriptores puede suponerse un buen descriptor de la figura del cuerpo humano si esta está aproximadamente centrada en la imagen.

Fig. 6. Visualización del histograma de gradientes sobre la imagen analizada.

2.3. Procesado multiescala de imagenes

El análisis anterior resuelve el problema de la parametrización de cada región de imágenes, pero no la detección de una persona en una imagen genérica. Para ello, en [1] se ha propuesto el uso de análisis multiescala con pasos de 1.2, esto es, reducir en cada escala el tamaño de la imagen por un factor de aproximadamente 83%. Por cada una de estas escalas se propone aplicar un método de ventana deslizante con una máscara de 64x128, obteniendo por cada iteración el descriptor de la región contenida en dicha zona, que servirá para clasificar de forma binaria la presencia o ausencia de una persona en dicha zona.

Para la clasificación se propone el uso de SVM, en particular, de separación lineal sin aplicación de kernels, pues la mejora obtenida mediante el uso de kernels gaussianos es prácticamente insignificante [1].

En consecuencia, tras deslizar la ROI por cada escala y clasificar en cada deslizamiento, se obtendrán regiones rectangulares en las que se ha tomado la decisión de si subyace o no un individuo en ellas. Dado que el deslizamiento propuesto es de 8 píxeles, se obtendrán bastantes zonas potencialmente válidas alrededor de cada individuo por lo que Navneet Dalal sugiere el uso de *mean shifting* o *non-maxima supresión*, técnicas de las cuales se ha escogido la segunda por haber mostrado una mayor robustez frente a errores.

3. Implementación

El procedimiento expuesto con anterioridad ha sido implementado en Python a fin de poder evaluar la presente técnica. Para ello, cabe exponer de forma no muy exhaustiva las principales funciones desarrolladas para lograr el objetivo propuesto, estando todo el código que aquí no se expone correctamente documentado mediante comentarios en el proyecto software adjunto a este informe.

El desarrollo de este código tiene como primer objetivo el de poder entrenar un clasificador SVM que posteriormente será almacenado en un fichero para poder realizar detecciones sin necesidad de reentrenar en cada ejecución del programa. Por ello, primero hemos de empezar analizando este proceso. Dado que se deseaba estudiar las desventajas de la técnica inherentes a su estructura y no a la cantidad de datos disponibles, se ha realizado un incremento de la base de datos usada entrenamiento mediante la recopilación y preparación de diversas fuentes disponibles en [4]. De este modo, se ha conseguido generar una base de datos de entrenamiento con 39.000 imágenes de ejemplos positivos y aproximadamente 60.000 de ejemplos negativos, generando ello en un total de aproximadaente 98.000 imágenes de entrenamiento. No obstante, los ejemplos negativos han sido generados recortando imágenes de forma aleatoria en una serie de imágenes base proporcionadas en la base de datos INRIA [5], por lo que el siguiente código ha sido desarrollado para dicho fin.

```
1.  def generateNegativeDatabase(orig_path, dst_path):
2.      '''''
3.      Esta función genera la base de datos de muestras
4.      negativas recortando N_NEG_CROPS imagenes de
5.      64x128 px. obtenidas de orig_path y almacenando
6.      cada crop en dst_path
7.
8.      :param orig_path:    Ruta de la carpeta con las
9.      imagenes de las que extraer negativos.
10.
11.     :param dst_path: Ruta de la carpeta en la
12.     que guardar cada imagen de 64x128 px.
13.     '''
14.
15.     filenames = os.listdir(orig_path);
16.
17.     # Para cada fichero se generan N_NEG_CROPS imagenes
18.     # en una posicion (x0,y0) totalmente aleatoria
19.     for name in filenames:
20.         orig_image_path = "%s/%s"%(orig_path, name);
21.         image = cv2.imread(orig_image_path);
22.         x0_max = image.shape[1]-64;
23.         y0_max = image.shape[0]-128;
24.
25.         for i in range(N_NEG_CROPS):
26.                 dst_image_path = "%d_%s/%s"%(i,dst_path, name);
27.
28.                 x0 = random.randint(0,x0_max);
29.                 y0 = random.randint(0,y0_max);
```

```
                xf = x0 + 64;
                yf = y0 + 128;

                random_crop = image[x0:xf, y0:yf];

                cv2.imwrite(dst_image_path, random_crop);
```

Una vez que se dispone de las dos carpetas con ejemplos positivos y negativos, queda extraer las características y etiquetas de estos para almacenarlos en dos archivos. Para la extracción de las características se usa el módulo *cv2.HOGDescriptor* que para una ventana de 64x128 pixeles nos devuelve su descriptor HOG de 3780 componentes, realizando el cálculo de forma paralela y con optimizaciones a nivel hardware, razón por la que se ha usado este módulo. La configuración del extractor de dichas características se ha realizado como sigue, adapandolo así al problema aquí resuelto.

```
winSize                 = (64,128);
'''
    Tamaño de las imagenes de las que
    vamos a extraer caracteristicas
'''

nbins               = 9;
'''
    Numero de bins que contiene cada
    histograma de gradientes, desde
    0 grados hasta 180 grados en este
    caso de 20 grados de separación.
'''

cellSize            = (8,8);
'''
    Tamaño de la celula para la cual
    sacar el histograma de 9 bins
'''

blockSize           = (16,16);
'''
    Tamaño del bloque de celulas
    que se va a usar para normalizar
    Como es 2x2 celulas, queda en
    16x16 px.
'''
blockStride         = (8,8);
'''
    Desplazamiento de los bloques, al
    solaparse a 50% es de 1 bloque y
    por lo tanto 8px
'''

histogramNormType   = 0;
'''
    Código para la norma L2-Hys
'''

L2HysThreshold  = 0.2;
'''
    Valor de clipping para la saturación
    o histéresis de L2-Hys.
'''
```

```
45.
46. EXTRACTOR = cv2.HOGDescriptor(winSize, blockSize,
47.                                 blockStride, cellSize,
48.                                 nbins, histogramNormType,
49.                                 L2HysThreshold);
```

A continuación, se generarán los vectores de características y de etiquetas iterando sobre las base de datos de imágenes. Para ello, en la lectura de cada imagen hay que hacer un recorte a 64x128 pixeles dado que algunas de ellas poseen un padding adicional, lo cual se soluciona mediante la siguiente función.

```
1.  def readDatabaseImage(path):
2.      WINDOW_HEIGHT = 128;
3.      WINDOW_WIDTH = 64;
4.      image = cv2.imread(path);
5.      height, width, channels = np.shape(image);
6.
7.      #Las imagenes pueden tener un pequeño padding lateral o
8.      #vertical, por lo que se elimina durante su lectura
9.
10.     x_pad = (width - WINDOW_WIDTH)//2;
11.     y_pad = (height - WINDOW_HEIGHT)//2;
12.
13.     image = image[y_pad:y_pad+WINDOW_HEIGHT, x_pad:x_pad+WINDOW_WID
    TH];
14.
15.     return image;
```

Dado que tenemos 98000 observaciones, el tamaño de la variable que almacena las características será (*98000,3780*) mientras que el de las etiquetas será (*98000,1*). Para su almacenamiento, se usa el sistema de ficheros *h5py* para las características dado su elevado tamaño (*+2GB*) mientras que se usa *npy* para las etiquetas.

```
1.  def generateFeatures(pos_path, neg_path, features_dst, labels_dst):
2.      '''''
3.      Esta funcion genera las 3780 caracteristicas de cada imagen
4.      que se encuentra en los directorios pos_path y neg_path
5.      y almacena las caracteristicas en un fichero .h5 en
6.      features_dst y las etiquetas en un fichero .npy dentro
7.      de labels_dst
8.
9.      :param pos_path:
10.     :param neg_path:
11.     :param features_dst:
12.     :param labels_dst:
13.     :return:
14.     '''
15.
16.     pos_filenames = os.listdir(pos_path)
17.     neg_filenames = os.listdir(neg_path)
18.
19.     n_pos = len(pos_filenames);
20.     n_neg = len(neg_filenames);
21.     n_data = n_pos + n_neg;
22.
23.     features = np.zeros((n_pos+n_neg, 3780));
24.
```

```
    classes_code = np.array([0,1]);

    labels = np.concatenate((np.repeat(1,n_pos),
                                        np.repeat(0,n_neg)));

    count = 0;
    filenames = pos_filenames+neg_filenames;

    '''''
        Como sabemos cuantas imagenes positivas hay, en el
        momento en que count >= n_pos filenames solo contedra
        imagenes negativas.
    '''

    for filename in filenames:
        if (count<n_pos):
            folder = pos_path;
        else:
            folder = neg_path;

        image_path = ("%s/%s")%(folder, filename);
        image = readDatabaseImage(image_path);

        feature = EXTRACTOR.compute(image);
        features[count, :] = np.reshape(feature, (3780,));

        count += 1;

    #Guardo las etiquetas en un fichero .npy
    np.save(labels_dst, labels);

    #Guardo las características en un fichero .h5
    h5_file = h5py.File(features_dst, 'w');
    h5_file.create_dataset("features", data=features);
    h5_file.close();
```

Con esto ya tenemos ambos vectores generados y solo queda entrenar el clasificador SVM. Para ello, se realiza un proceso iterativo de entrenamiento variando el valor del factor de penalización y analizando los resultados de precisión en test, escogiendo finalmente el clasificador de mayor precisión. De esta forma, el siguiente código genera y almacena el clasificador final a usar.

```
def generateOptimalClassifier(training_features_path,
                                        training_labels_path,
                                        testing_features_path,
                                        testing_labels_path,
                                        classifier_dst_path):
    '''''
    Esta función entrena N clasificadores SVM lineales con
    distintos valores de factor de penalizacion, en el rango
    2**(-15) hasta 2**4 con pasos multiplicativos de 2. Una
    vez se encuentra el clasificador optimo, se guarda en
    el archivo classifier_dst_path en formato .pkl para
    cargarlo de cara a futuras clasificaciones y no tener
    que entrenar en cada ejecución.

    :param training_features_path: Ruta donde se encuentra
    un fichero .h5 con las características extraidas de la
    base de datos de entrenamiento.

    :param training_labels_path: Ruta donde se encuentra
```

```
    un fichero .npy con las etiquetas asociadas a cada
    vector de características de la base de datos de
    entrenamiento

    :param testing_features_path: Ruta donde se encuentra
    un fichero .npy con las características extraídas de la
    base de datos de testing.

    :param testing_labels_path: Ruta donde se encuentra
    un fichero .npy con las etiquetas asociadas a los
    vectores de características de la base de datos
    de testing.
    '''
    features_training = np.load(training_features_path);
    labels_training = np.load(training_labels_path);
    features_test = np.load(testing_features_path);
    labels_test = np.load(testing_labels_path);

    v_c = []; # vector con los valores de C

    for i in range(-12, 5):
        v_c.append(2**i); # valores de C

    optimal_classifier = [];
    miss_opt = 1;

    for c in v_c:
        classifier = svm.LinearSVC(C=c);
        classifier.fit(features_training, labels_training);

        predictions = classifier.predict(features_test);
        n_positives = np.sum(predictions==labels_test);

        score = (n_positives/len(labels_test));
        miss = 1-score;

        if(miss < miss_opt):
            miss_opt = miss;
            optimal_classifier = classifier;

    joblib.dump(classifier_dst_path, optimal_classifier);
```

Teniendo presente el clasificador almacenado en memoria, ya sólo queda recurrir al análisis multiescala y ejecutar la supresión de no-máximos, lo cual he implementado en la siguiente función.

```
def showWindows(orig_image, classifier):
    '''''
    Esta función detecta a las personas de la imagen en varias
    escalas usando el clasificador SVM proporcionado y usando
    NMS para suprimir los solapamientos. Una vez detectados
    todos los rectángulos, muestra estos junto a la imagen a
    la par que los recorta a parte por si se quieren analizar.

    :param orig_image: Imagen original en la que hacer la
        detección
    :param classifier: Clasificador SVM ya entrenado
    '''

    WINDOW_HEIGHT = 128;
    WINDOW_WIDTH = 64;
    SCALE_STEP = 1.2;
    WINDOW_STEP = 8;

```

```
    GAUSS_KERNEL_SIZE = (5,5);
    GAUSS_SIGMA = 0.6;

    height, width, channels = np.shape(orig_image);

    pyramid = [orig_image]

    new_width = width//SCALE_STEP;
    new_height = height//SCALE_STEP;

    while (new_height>WINDOW_HEIGHT and new_width>WINDOW_WIDTH):
        blurred = cv2.GaussianBlur(pyramid[-1], ksize=GAUSS_KERNEL_SIZE,
                                                sigmaX=GAUSS_SIGMA, sigmaY=GAUSS_SIGMA);
        pyramid.append(cv2.resize(src=blurred, dsize=(int(new_width),int(new_height))))

        new_height //= SCALE_STEP;
        new_width //= SCALE_STEP;


    scaling_factor = 1;
    windows = [];

    for image in pyramid:
        height, width, channels = np.shape(image);

        y_pad = (height%WINDOW_STEP)//2;
        x_pad = (width%WINDOW_STEP)//2;

        x_end = width-WINDOW_WIDTH-WINDOW_STEP;
        y_end = height-WINDOW_HEIGHT-WINDOW_STEP;

        for y in range(y_pad, y_end, WINDOW_STEP):
            for x in range(x_pad, x_end, WINDOW_STEP):
                window = image[y:y+WINDOW_HEIGHT, x:x+WINDOW_WIDTH];
                features = np.transpose(EXTRACTOR.compute(window));


                prediction = classifier.predict(features);
                if (prediction==1):
                    windows.append(np.array([
                                            x*scaling_factor,
                                            y*scaling_factor,
                                            (x+WINDOW_WIDTH)*scaling_factor,
                                            (y+WINDOW_HEIGHT)*scaling_factor,
                                            classifier.decision_function(features)
                                        ]));

        scaling_factor *= SCALE_STEP;


    '''''
        Supresión de no-máximos usando la función non_max_suppression del
        paquete imutils, el estándar para hacer NMS.
    '''
    windows = imutils.object_detection.non_max_suppression(np.array(windows),

            probs=None,
```

```
74.
                overlapThresh=0.65)
75.     n_windows = len(windows);
76.
77.     cp_image = orig_image.copy();
78.
79.     for i in range(n_windows):
80.         x1 = int(windows[i][0]);
81.         y1 = int(windows[i][1]);
82.         x2 = int(windows[i][2]);
83.         y2 = int(windows[i][3]);
84.         cv2.rectangle(orig_image, (x1, y1), (x2, y2), (0, 0, 255),
    2);
85.         cv2.imshow("Cropped window number %d"%(i+1), cp_image[y1:y2
    ,x1:x2]);
86.
87.     cv2.imshow("Detected windows",orig_image);
```

Con lo anterior, ya queda totalmente implementada la técnica propuesta en [1] para detección de personas en imágenes, por lo que cabe proceder a la evaluación de la técnica.

4. Evaluación

4.1. Resultados analíticos

A fin de realizar una primera evaluación numérica, se ha realizado un estudio de la bondad del clasificador obtenido sobre los datos de test. Este estudio ha sido realizado durante el entrenamiento realizado en *generateOptimalClassifier*, durante el cual se ha extraído la tasa de fallo, la tasa de falsos positivos y la tasa de verdaderos negativos para cada valor del factor de penalización. Esto, en consecuencia, ha permitido evaluar el modelo bajo las distintas condiciones de clasificación.

En primer lugar, en relación a la tasa de fallos, su evolución observada se muestra en la figura Fig. 7. Como se observa, existe un valor del factor de penalización de entorno a 0.3 que permite alcanzar un mínimo en esta tasa de error. Es decir, este valor representa el equilibrio perfecto entre la importancia del margen de la región de decisión y la correcta clasificación durante entrenamiento para intentar generalizar el modelo lo mejor posible y obtener buenos resultados en test, resultando ello en un error logarítmico inferior a -45dB.

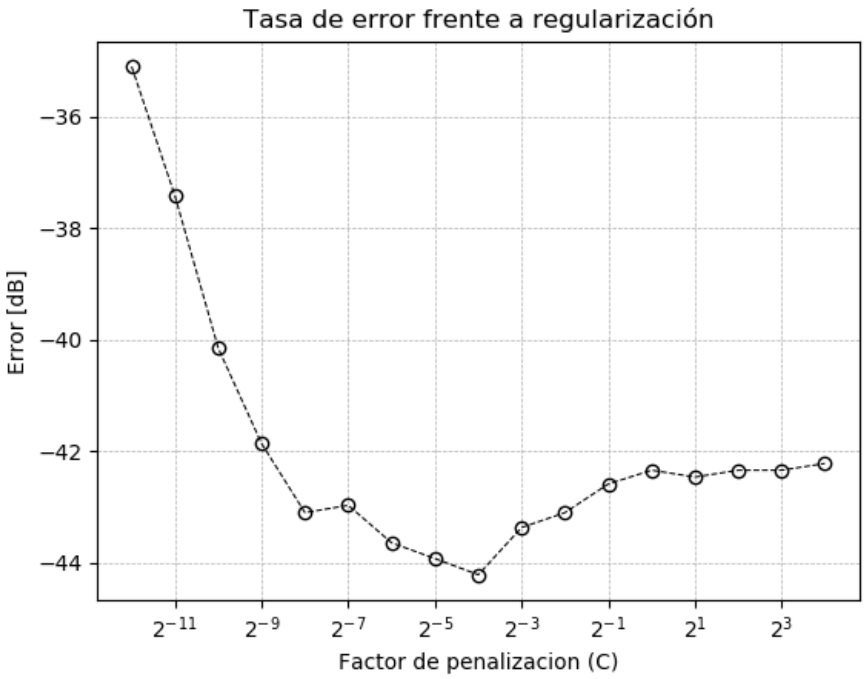


Fig. 7. Tasa de error frente a penalización

En segundo lugar, cabe estudiar cómo evolucionan los falsos positivos y los verdaderos negativos en función del coeficiente anterior durante le proceso de entrenamiento. Para ello, se han extraído las gráficas de estas tasas durante el mismo proceso de entrenamiento y testing anterior haciendo

uso de la matriz de confusión, pues ambos quedan representados por la diagonal no-principal de esta matriz. Los resultados obtenidos se muestran en la figura Fig. 8, donde se puede apreciar que mientras los verdaderos negativos mejoran los falsos positivos empeoran y viceversa, lo que apunta a que no existe una separación perfecta entre ambas clases mediante el presente kernel lineal. Además, se observa que durante el intervalo analizado, obtenido de la recomendación de *sklearn* en [6], el comportamiento de ambas gráficas no posee la misma monotonía que la tasa de error, por lo que habrá que guiarse por un criterio de selección del factor de penalización a usar. En este caso, en base a la experimentación empírica realizada, se han preferido los resultados obtenidos con el clasificador que minimiza la tasa de fallo, pero en aplicaciones reales tales como videovigilancia o tracking de individuos el criterio más apto podría basarse más en sendas tasas de falsos positivos y verdaderos negativos.

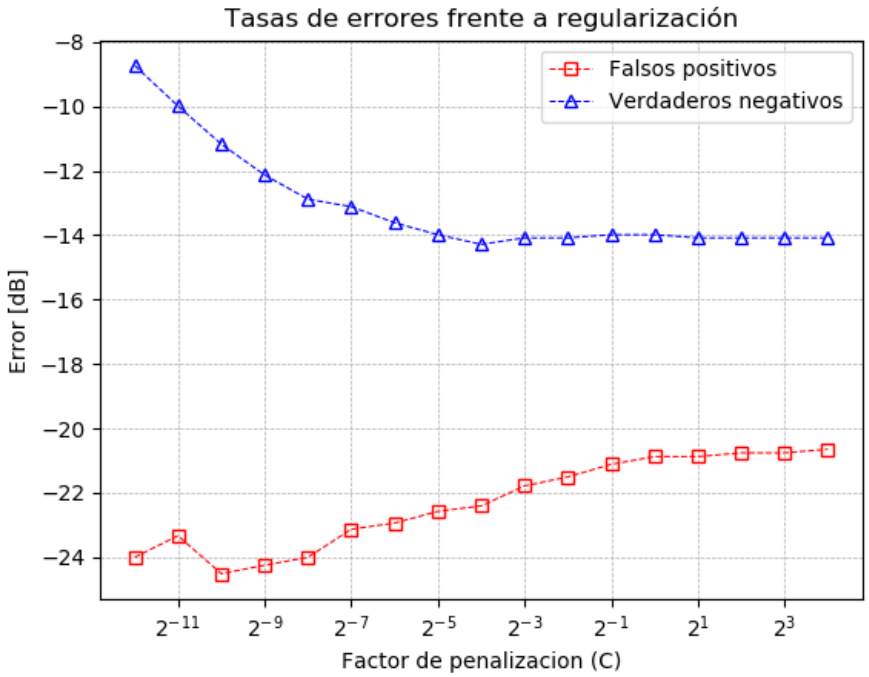


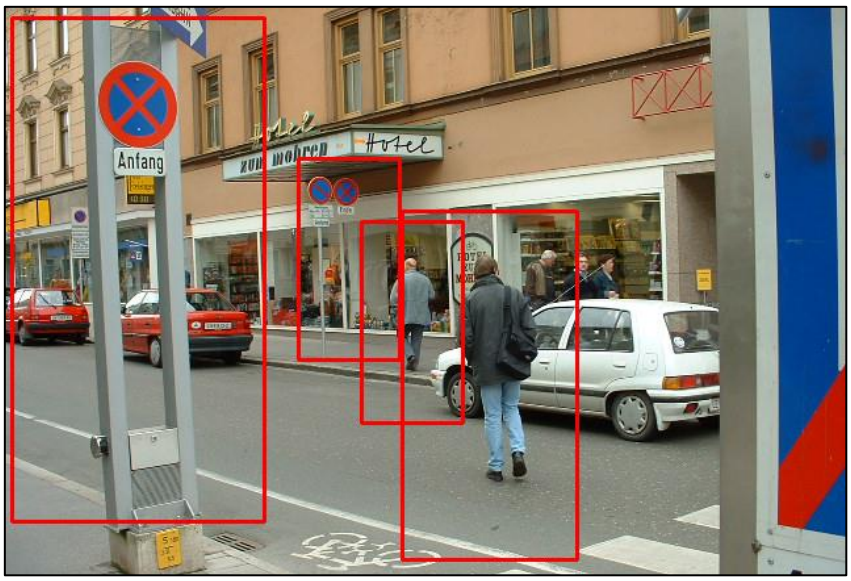
Fig. 8. Tasa de falsos positivos y verdaderos negativos

4.2. Evaluación empírica

Analizado el comportamiento analítico del modelo en base a las bases de datos de test, cabe realizar ahora un estudio práctico y visual de los errores y limitaciones del mismo en base a los resultados obtenidos en detección.

En primer lugar, tal y como se muestra en la figura Fig. 9, el clasificador comete dos falsos positivos clasificando a las señales de tráfico como personas. A priori, resulta complicado derivar la razón de ello por lo que habremos de indagar en el propio procedimiento llevado a cabo en dicha

clasificación. Como se indicó al inicio, la parametrización de cada ventana se obtiene mediante el histograma de gradientes orientados. Esto hace que la clasificación se realice en base al parecido de la orientación de los gradientes en una imagen con la orientación que poseen estos en el cuerpo de un humano. Como consecuencia, aparecerán diversos falsos negativos cuando un objeto posea una forma cuyo contorno quede distribuido de forma semejante al de un humano. Este fenómeno queda visible a través de la figura Fig. 10, donde se observa que la distribución de gradientes circular de la señal y la distribución vertical de su soporte hacen que esta sea confundible con la forma de una persona. Como consecuencia de ello, derivamos en primer lugar que los falsos positivos se deben a la similitud de algunos objetos con la forma humana.



Fig,. 9. Ejemplo con dos falsos positivos

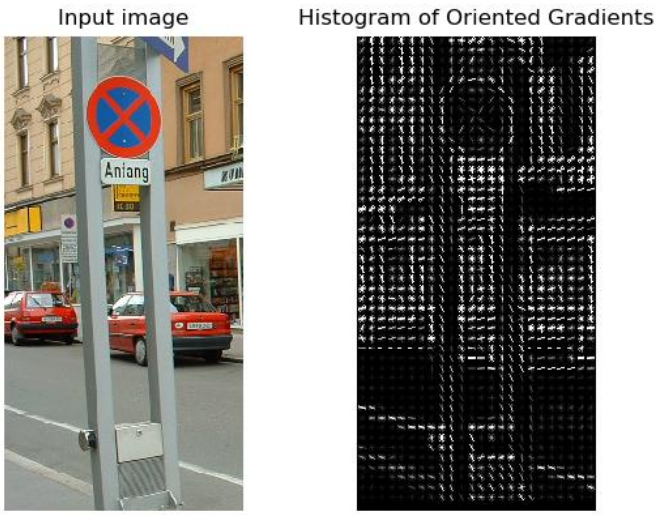


Fig. 10. Estudio del histograma del falso positivo más grande.

Tras haber derivado la causa de los falsos positivos, procedemos ahora a justificar la ocurrencia de verdaderos negativos, esto es, personas que no están siendo detectadas. Como imagen de ejemplo partimos de la mostrada en la figura Fig. 11, donde se observa que el padre no ha sido detectado por el algoritmo. Nuevamente, derivar la causa a simple vista no resulta trivial por lo que es necesario el análisis de la información contenida en los gradientes.

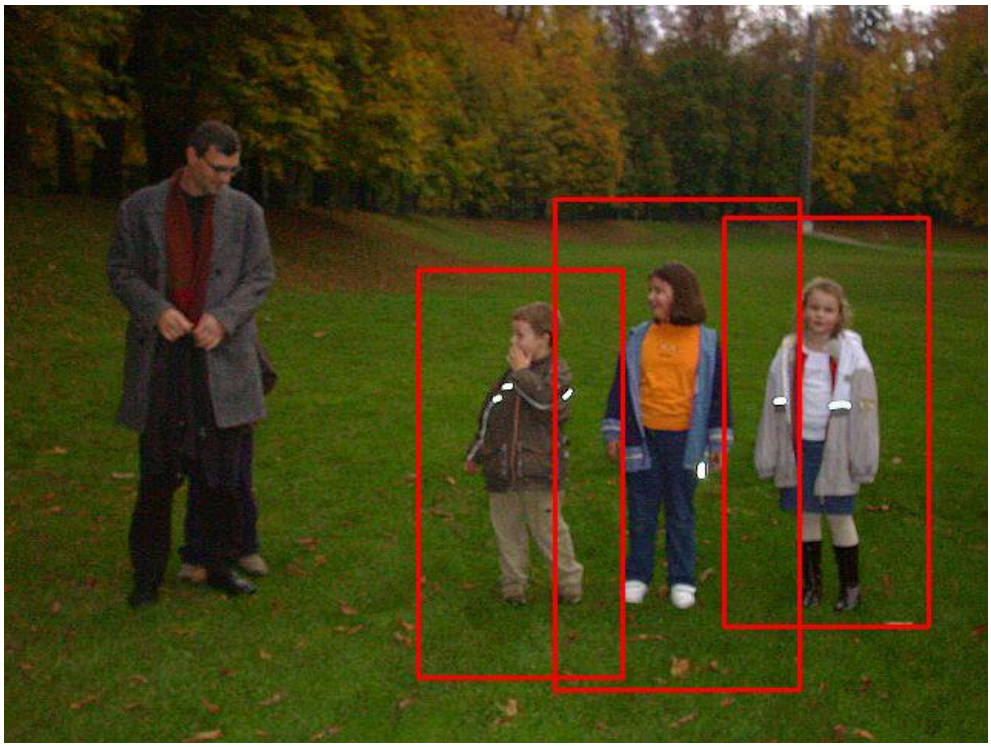


Fig. 11. Ejemplo de verdadero negativo, de momento de causa desconocida.

Procediendo de forma análoga al caso anterior, se ha computado para la imagen de la figura anterior su distribución de histogramas de gradientes, obteniendo como consecuencia la imagen de la figura Fig. 12. En ella, podemos observar claramente una fuerte distribución de gradientes verticales en el cuerpo de los niños que han sido detectados, a la par que una tendencia circular en los gradientes asociados a la zona donde se encuentran sus cabezas. Esto no ocurre de la misma forma para el caso del padre, donde por problemas de iluminación los gradientes relativos al torso son muy débiles y su postura no-erguida hace que se pierda bastante información en los gradientes relativos a su cabeza, por lo que no es clasificado correctamente. Podemos derivar pues, que el presente método falla cuando las condiciones luminosas hacen que los gradientes no estén lo saturados que habrían de estar a la par que existe una alta sensibilidad con la postura adoptada por la persona, pues esta influye a la silueta corporal que la presente técnica extrae.

Fig. 12. Distribución espacial HoG del caso verdadero negativo.

Finalmente, cabe analizar el caso en que se presentan verdaderos negativos a pesar de que los gradientes son perfectamente computables y distintivos. Como ejemplo se muestra imagen de la figura Fig. 13, la cual posee una nitidez y brillo espacial suficientes como para poder computar los gradientes sin gran error. No obstante, se observan dos verdaderos positivos por lo que habremos de recurrir nuevamente a nuestro análisis HoG.

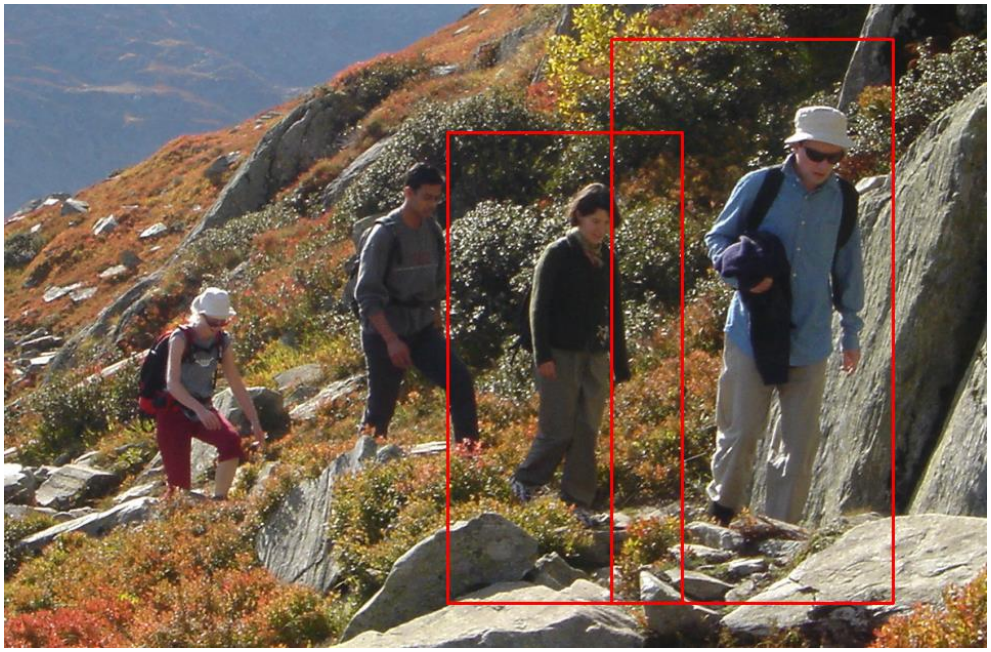


Fig. 12. Distribución espacial HoG del caso verdadero negativo.

De la figura anterior, el cómputo de la distribución espacial de los histogramas queda mostrado en la figura Fig. 13. En ella, obseravmos que las personas clasificadas correctamente se encuentran en una postura

distintivamente erguida mientras que los dos verdaderos negativos se sostienen bajo posturas más inclinadas. Como consecuencia, el algoritmo no es capaz de clasificarlos como personas dado que la distribución de gradientes en sus cuerpos no corresponde al patrón vertical que el modelo conoce al ser este el predominante en la base de datos de entrenamiento. Como consecuencia de lo presente, podemos concluir que la técnica sufrirá de casos con verdaderos negativos cuando las personas en cuestión se encuentren en posturas no erguidas.

Fig. 13. Distribución espacial HoG para el caso con personas no erguidas.

5. Conclusiones

Un análisis detallado acerca de la técnica HoG para detección de personas ha sido desarrollado en el presente proyecto, derivando en consecuencia las principales limitaciones de la misma en base a datos analíticos y empíricos obtenidos de la implementación desarrollada.

En el estudio aquí presentado, se ha observado cómo la técnica se limita a la comparación de distribuciones de gradientes con una generalización de la silueta humana. Este hecho, aunque robusto en una gran variedad de casos, conduce a una serie de errores por diversos motivos. En primer lugar, objetos con una distribución vertical en su soporte y oval en su extremo, tales como árboles, papeleras o farolas, serán plausibles de ser clasificados erróneamente dada la semejanza entre su contorno y la silueta humana. En segundo lugar, personas que no se encuentren en una posición relativamente erguida perderán la información relevante asociada a la distribución de sus gradientes, siendo susceptibles en consecuencia a una clasificación errónea.

Las limitaciones expuestas con anterioridad se pueden reducir a una única causa diferenciable: la parametrización. Esta forma de proceder parte de la heurística asociada al hecho de que la parametrización de la silueta humana será suficiente para conseguir separar las clases en un espacio N-dimensional. No obstante, tal y como hemos observado con anterioridad, existen ciertos objetos que por la forma de su contorno son proyectados en la región del espacio N-dimensional perteneciente a la clase contraria, siendo muy difícil para el algoritmo de aprendizaje automático la separación de estos casos.

Motivados por la deficiencia anterior, numerosos trabajos evolucionaron en la línea de incluir cierta flexibilidad en la parametrización a fin de que el modelo quedase lo menos afectado posible por las heurísticas pre-supuestas [7]. Tanto fue así que las redes neuronales convolucionales, siendo el estado del arte, no presuponen ningún tipo de parametrización en particular. Estos modelos se basan en la extracción de características adaptable, es decir, los propios filtros de extracción evolucionan durante la fase de entrenamiento para aprender a extraer las características más relevantes de la imagen. De este modo, el modelo resultante requiere de la mínima intervención humana posible, reduciendo en consecuencia los errores por pre-suposiciones que se cometen en técnicas como la presente.

Podemos concluir en definitiva que la presente técnica, aunque deficiente en varios aspectos, resultó no sólo en un claro avance en la fecha de su publicación si no también en un impulso de las técnicas sucesoras hacia la mejora de la separabilidad basada en parametrización, siendo en consecuencia una técnica revolucionaria en la visión por computador.

6. Referencias

[1] Navneet Dalal, Bill Triggs. Histograms of Oriented Gradients for Human Detection. Cordelia Schmid and Stefano Soatto and Carlo Tomasi. International Conference on Computer Vision & Pattern Recognition (CVPR '05), Jun 2005, San Diego, United States. IEEE Computer Society, 1, pp.886–893, 2005,

[2] R. Szeliski, "Computer Vision: Algorithms and Applications", *Springer-Verlag*, Heidelberg, Berlin, June 2010. ISBN: 1848829345

[3] S. Mallick, V. Gupta, V. S. Chandel, "Computer Vision Resources: OpenCV for beginners", *LearnOpenCV*, April 2016,

[4] Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H. (2012). *Learning from data: A Short Course*, AMLBook, May 2012, ISBN: 1600490069

[5] N. Dalal, "INRIA Person Dataset", URL: pascal.inrialpes.fr/data/human

[6] J. Smola, B.Schölkopf, "A Tutorial on Support Vector Regression", Statistics and Computing archive, Volume 14 Issue 3, August 2004, p.199-222.

[7] M. Szarvas, A. Yoshizawa, M. Yamamoto and J. Ogata, "Pedestrian detection with convolutional neural networks," *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 224-229.

ANEXO A. Guía de Uso

El código entregado ha sido diseñado con el objetivo de poder ser ejecutado por cualquiera que disponga de la versión 3 de python a fin de que se puedan visualizar de primera mano los resultados extraídos en este trabajo con otras imágenes distintas a las aquí presentadas.

Para visualizar una imagen, simplemente hay que ejecutar el archivo *main.py* proporcionado, el cual lanza una interfaz gráfica como la mostrada en la figura Fig. 14. Como se puede observar hay dos funcionalidades principales. La primera de ellas, permite seleccionar una imagen del directorio y visualizar los resultados la presente técnica, para lo que se recomienda hacer uso de la carpeta *Test* adjunta de cara a visualizar las clasificaciones. La segunda de ellas permite mostrar las distribuciones HOG de una imagen seleccionada a fin de poder obtener más información acerca de la clasificación en la misma.

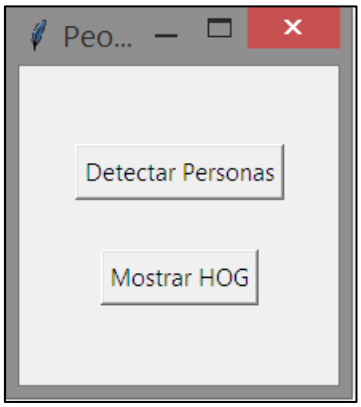


Fig. 14. Interfaz gráfica para la obtención de resultados.

Finalmente, todo el código ha sido constantemente actualizado desde su inicio en *http://www.github.com/FaridArbai/PeopleDetector* a fin de que se pueda evaluar la integridad del mismo y el desarrollo autónomo y original que he realizado en este proyecto a lo largo de las últimas semanas.