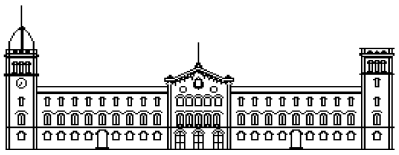


Treball fi de carrera

ENGINYERIA TÈCNICA
EN INFORMÀTICA DE SISTEMES

Facultat de Matemàtiques
Universitat de Barcelona

Multi-clasificación Discriminativa de Partes Corporales basada en Códigos Correctores de Errores

---

José Tomás Pérez Yarza

Directors: Sergio Escalera Guerrero
Miguel Ángel Bautista Martín
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi, UB.

Barcelona, 20 de Setembre de 2013

Resumen

Este proyecto tiene como finalidad la aplicación de distintas técnicas del campo de la Visión Artificial para la detección y segmentación de las extremidades de una persona sobre una base de datos de nueva creación.

Dicha base de datos contiene un gran número de imágenes donde aparecen varios sujetos realizando distintas poses. El objetivo es detectar las extremidades de dichos sujetos, como pueden ser los brazos, las piernas, el torso o la cabeza, entre otros, para posteriormente obtener una segmentación multi-extremidad.

Para realizar la detección se han entrenado diferentes cascadas de clasificadores sobre características Haar y HOG de regiones previamente orientadas donde aparecían extremidades. Tras el entrenamiento se han ejecutado varios experimentos sobre la base de datos para testear la detección de las extremidades mencionadas.

Se han utilizado también métodos para comprobar dichas detecciones. Finalmente se han aplicado técnicas de segmentación con dos finalidades: Por un lado segmentar al sujeto del fondo de la imagen y por otro cada una de las extremidades de dicho sujeto. En este caso se ha optado por una segmentación mediante la formulación Graph-cuts.

Resum

Aquest projecte té com a finalitat l'aplicació de diferents tècniques del camp de la Visió Artificial per a la detecció i segmentació de les extremitats d'una persona sobre una base de dades de nova creació.

Aquesta base de dades conté un gran número d'imatges on apareixen diversos subjectes tot realitzant diferents postures. L'objectiu és detectar les extremitats d'aquests subjectes com poden ser els braços, les cames, el tors o el cap, entre d'altres, per a que posteriorment obtinguem una segmentació multi-extremitat.

Per tal de realitzar la detecció s'han entrenat una sèrie de cascades de classificadors sobre característiques Haar i HOG de regions prèviament orientades on hi apareixien extremitats. En acabar l'entrenament s'han llançat diversos experiments sobre la base de dades per tal de testejar la detecció de les extremitats esmentades.

S'han utilitzat, també, mètodes per a comprovar aquestes deteccions. Finalment s'han aplicat tècniques de segmentació amb dues finalitats: Per una banda segmentar el subjecte del fons de la imatge i per l'altra cada una de les extremitats d'aquest subjecte. En aquest cas s'ha optat per una segmentació mitjançant la formulació Graph-cuts.

Abstract

This Project aims at the application of different techniques from the field of Artificial Vision for the detection and segmentation of human limbs on a newly created database.

The database contains a large number of images where multiple subjects appear performing various poses. The objective is to detect the limbs of such subjects, including the arms, legs, body or head, among others, to subsequently obtain a multi-limb segmentation map.

In order to perform this detection we trained different classifiers cascades on Haar and HOG features on targeted regions where limbs appeared. Once trained, several experiments have been released over the database for detecting the limbs mentioned.

Some methods have been used to verify the detections. Finally, segmentation techniques have been applied for two purposes: On one hand, segment the subject from the background of the image, and on the other hand, each limb of the subject. In this case we have chosen segmentation using Graph-cuts formulation.

ÍNDICE

1. Introducción

En este apartado se introduce este Proyecto de Final de Carrera, describiendo el contexto y motivación que han llevado al desarrollo del mismo. Además, se incluye un pequeño estudio del estado del arte sobre métodos que abordan la problemática de este mismo proyecto.

1.1 Contexto

La clave para realizar la detección de personas, poses y extremidades es disponer de un amplio conjunto de entrenamiento que contenga toda clase de situaciones y con toda clase de condiciones con tal de asegurar la máxima generalidad posible en los métodos que se aplicarán.

Para este proyecto se ha creado y utilizado una base de datos de imágenes en las que aparecen una serie de sujetos realizando una serie de poses. Con ella se procederá al análisis de varias técnicas de visión por computador con el objetivo de detectar las extremidades de dichos sujetos según la pose realizada.

La metodología de detección de extremidades de este proyecto está basada en el aprendizaje mediante cascadas a través de un entrenamiento realizado sobre un gran conjunto de ejemplos. Posteriormente, se optimizarán los resultados obtenidos mediante métodos de códigos correctores de errores.

Finalmente se segmentará a nivel de píxel tanto la persona como sus extremidades, separándolos totalmente del fondo.

1.2 Motivación

La motivación de este Proyecto de Fin de Carrera (PFC) consiste por un lado en la creación de una gran base de datos que permita trabajar con todo tipo de metodologías del ámbito de la visión por computador y contribuir en esta área de investigación, y por otro diseñar, implementar e integrar de forma modular un sistema capaz de detectar y segmentar extremidades de personas sobre la base de datos que se acaba de nombrar. Se utilizarán para ello varios algoritmos representativos del estado del arte que hemos considerado que abordan esta cuestión de un modo eficiente y que obtienen buenos resultados.

La detección de gestos es un área de investigación desafiante que aborda el problema de detección de personas en las imágenes, la detección y descripción de las partes del cuerpo, dando a entender su configuración espacial, y la realización de la acción/detección de gestos en imágenes fijas o secuencias de imágenes.

Debido a la enorme variabilidad de poses que pueden realizar los humanos, detectar la posición del cuerpo es un problema que tiene una gran dificultad e implica lidiar con varias distorsiones: los cambios de iluminación, las oclusiones parciales, cambios en el punto de vista de referencia, las deformaciones rígidas y elásticas, sólo por mencionar algunas. Incluso con la alta dificultad del problema, las técnicas modernas de visión por computador y las nuevas tendencias merecen más atención, ya que en los próximos años se esperan resultados prometedores.

Por otra parte, varias sub-áreas han sido recientemente creadas, tales como análisis de la conducta humana, la robótica social, etc. El esfuerzo que implica esta área de investigación se verá compensado por sus posibles aplicaciones: área de educación, investigación en el área de la sociología, vigilancia y seguridad, mejora de la calidad de vida a través del monitoreo o la asistencia artificial automática, etc. [1].

1.3 Estado del arte

En este proyecto se han utilizado una serie de métodos con una gran relación calidad-eficiencia. A continuación se procede a una breve explicación de los métodos utilizados en cada fase del proyecto. Dichos métodos serán tratados con mayor profundidad en secciones posteriores.

1.3.1 Descriptores

Los descriptores de características son el primer paso para poder encontrar la relación entre los píxeles que almacena una imagen digital y aquello que los seres humanos somos capaces de reconocer tras observarla. Dicha descripción cubre características visuales básicas como pueden ser el color, la textura o la forma entre otras.

La idea es mapear estas características en valores numéricos para ser tratados por otros métodos posteriormente.

Algunos de los descriptores que podemos encontrar en el estado del arte son SIFT (Scale Invariant Feature Transform) y SURF (Speeded Up Robust Features). El primer caso se trata de un algoritmo capaz de detectar puntos característicos estables de una imagen. Estos puntos son invariantes frente a distintas transformaciones como rotación, traslación, iluminación o escala. El segundo guarda cierta similitud con la filosofía de SIFT, sin embargo y según sus autores, presenta principalmente dos mejoras: una velocidad de cálculo considerablemente superior sin que ocasione pérdida de rendimiento y una mayor robustez antes posibles transformaciones de la imagen.

Los dos descriptores utilizados en este PFC son Histogramas de Orientación de Gradientes (HOG) y las características Haar-like. El primero trabaja sobre la idea de que cualquier imagen se puede describir como una distribución de las direcciones de cambio de intensidad mientras que el segundo se basa en las diferencias de luminancia encontradas en distintas regiones de la imagen. Ambos métodos serán comentados más adelante.

1.3.2 Clasificadores

Los clasificadores son algoritmos capaces de, aprender una cierta distribución de datos a partir de una serie de ejemplos de entrenamiento, para posteriormente poder predecir la clase a la que pertenecen nuevos ejemplos no utilizados en el entrenamiento.

Podemos encontrar varias ramas de clasificadores según su aprendizaje:

Aprendizaje supervisado

> Es una técnica de aprendizaje artificial que elabora una función matemática a partir de datos de entrenamiento previamente etiquetados como por ejemplo *SVM o Adaboost*.

Aprendizaje no supervisado

En este caso, el conjunto de entrenamiento no dispone de etiquetas conocidas, así que requiere de técnicas de agrupamiento que intenten construir estas etiquetas. Un ejemplo sería *Hidden Markov Models* (HMM).

Aprendizaje semi-supervisado

Es una combinación del aprendizaje supervisado y del no supervisado. Surge de la dificultad que conlleva obtener los datos etiquetados requeridos en el aprendizaje supervisado. Por esa razón este método recurre al uso de una parte de datos etiquetados y un conjunto más extenso sin etiquetar mejorando, de este modo, la construcción de los modelos.

Se asume que los datos no etiquetados siguen la misma distribución que los etiquetados puesto que de no ser así estos datos serían de poca utilidad. Por ejemplo, *Co-training* o *re-weighting.*

Aprendizaje por refuerzo

Este método no trata de aprender a partir de un conjunto de ejemplos si no a través de la experiencia. Por ejemplo *Q-Learning*.

El abanico de aplicaciones de los clasificadores es muy amplio, por ejemplo análisis de pruebas de drogas o de datos de resonancia magnética en medicina, decodificación de señal o corrección de errores en teléfonos móviles o reconocimiento facial, de personas o de toda clase de objetos en el ámbito de la visión por computador.

1.3.3 Algoritmos de Multiclasificación

En este proyecto se ha optado por utilizar los Códigos de corrección de Errores (CCE o ECOC). Se trata de un marco de trabajo para combinar clasificadores binarios con la intención de abordar problemas multi-clase, es decir, con un número de clases $N>2$. Esta metodología ha sido aplicada con éxito sobre una amplia gama de aplicaciones de reconocimiento de patrones con unos resultados satisfactorios. Posteriormente se revisará con mayor profundidad la metodología ECOC.

Otros de los métodos que podemos encontrar para afrontar el problema de la multi-clasificación son el C4.5, la estrategia *uno contra todos*, la estrategia *uno contra uno* o SVM-Multiclass. El primero construye árboles decisión a partir de un conjunto de entrenamiento de ejemplos ya clasificados, cada uno de los nodos de dicho árbol elige el atributo que mejor divide el conjunto de muestras en subconjuntos enriquecidos de una clase u otra.

El segundo se basa en la combinación de clasificadores binarios para los que existen varias estrategias como *uno contra todos*, en que se construyen k clasificadores que definirán k hiperplanos que separará una clase del resto. Se escogerá como clase final aquella que maximice el margen alrededor del hiperplano. Por otro lado existe la estrategia *de uno contra uno*, para la que se construye un clasificador para cada par de clases posible, enfrentándolas a todas una a una. De este modo, la decisión final se reduce al recuento de votos que realice cada clasificador, ganando el que más votos consiga. Finalmente, SVM-Multiclass intenta aprender una única frontera que sea capaz de separar las N clases que constituyen el problema.

En secciones posteriores se detallará SVM, dejando más claro el funcionamiento de SVM multiclass puesto que se trata de una aproximación del primero.

1.3.4 Segmentación

Uno de los algoritmos de segmentación más utilizados es Mean-Shift, también utilizado en el reconocimiento de palabras y muchas otras áreas del Procesamiento de Señales. Se trata básicamente en la aplicación recursiva del método con tal de encontrar el punto estacionario más cercano de la función de densidad, ayudando de esta forma a encontrar las modas de la misma.

Otro conocido algoritmo es K-means, se trata de un método de agrupamiento que busca formar grupos que serán representados por una serie de objetos. Cada uno de estos objetos "representantes" será el valor medio de los objetos que pertenecen a dicha agrupación.

Los métodos que se utilizarán en este proyecto para para abordar el problema de la segmentación son: Grab-cut, basado en la teoría de saturación (algoritmo de Ford-Fulkerson) de Graph-cut alpha-expansion, que extiende la teoría de graph cuts a un contexto multi-extremidad.

El primero se encargará de la segmentación binaria, en nuestro caso, segmentará a la persona del fondo de la imagen, y el segundo método realizará la segmentación multi-clase, es decir, de las distintas extremidades del cuerpo: cabeza, piernas, brazos, etc.
Se hablará con más detalle sobre estos dos métodos en secciones posteriores.

1.4 Propuesta

En este trabajo de final de carrera proponemos la creación de una base de datos de imágenes en las que aparecen varias personas realizando una serie de poses y la posterior aplicación de técnicas de visión por computador con el fin de detectar sus extremidades.

Primero se diseñará y realizará una gran base de datos con centenares de miles de imágenes en las que se etiquetaran extremidades de varias personas. Se procederá a la extracción de características de dichas imágenes las cuales pasarán a formar parte de la entrada de un sistema de entrenamiento de clasificadores.

Con los clasificadores entrenados, tras la fase de testeo, pasaremos los resultados por los ECOC que corregirán posibles errores en la clasificación.

Finalmente se aplicarán los métodos de segmentación para obtener los resultados finales.

1.5 Objetivos

Los objetivos establecidos en la planificación del proyecto son los siguientes:

- Diseño y creación de la base de datos Human Pose and Behaviour Analysis (HuPBA).
- Extracción de características de las imágenes utilizando descriptores.
- Entrenar distintos clasificadores en cascada capaces de clasificar las distintas extremidades de las personas.
- Aplicación de los códigos de corrección de errores para mejorar los resultados obtenidos durante la clasificación.
- Aplicar métodos de segmentación binaria y multi-clase para obtener una segmentación multi-extremidad a nivel de pixel.

1.6 Estructura de la memoria

La memoria ha sido dividida en las siguientes partes:

- Descripción de la base de datos
- Explicación de los distintos clasificadores utilizados junto a sus descriptores de características
- Uso de los árboles de decisión utilizados en combinación con SVM
- Conjunto de Experimentos realizados
- Resultados obtenidos
- Conclusiones

2. Base de Datos HuPBA

2.1 Introducción

En el día a día, nos encontramos con una, poco menos que infinita, variedad de objetos con los que convivimos diariamente, ya sea interactuando o simplemente cruzándonos con ellos en un momento dado. Distinguir personas y reconocer los gestos que realizan dentro de este contexto multi-clase es uno de los grandes problemas que nos encontramos en el campo de la Visión por Computador. La escasez de grandes bases de datos, tanto a nivel de tamaño como de protocolos de validación, ha propiciado el nacimiento de la base de datos HuPBA. A continuación se detalla el proceso de generación de la base datos.

2.2 Creación de la base de datos

La base de datos HuPBA se compone de un conjunto de vídeos, nueve en concreto, grabados en la Universidad de Barcelona. Todos ellos se desarrollan en un entorno controlado, es decir, nos encontramos con un fondo uniforme que permite centrar la atención en las personas que aparecen (desde ahora actores). Cada uno de los vídeos consta de un actor principal que realiza una serie de acciones en solitario (saludar, saltar, correr, …) y otras en las que interactúa con actores secundarios que van entrando en escena durante el transcurso de la grabación y realizando actividades conjuntamente con el actor principal (pelear, abrazarse, …). Los actores visten prendas de distintas formas y colores, algunas complican la distinción de ciertas partes del cuerpo y otras las hacen más evidentes.

Puesto que un vídeo no es más que una secuencia de imágenes (fotogramas o frames), procederemos a descomponer cada uno de ellos para obtener el conjunto de imágenes resultante. El objetivo es que esta base de datos pueda ser útil para cualquier campo de investigación y para ello es de vital importancia diseñar y aplicar una correcta nomenclatura a los recursos para permitir su fácil acceso en cualquier post-proceso que se realice sobre ellos.
En este caso para hacer referencia a las imágenes RGB originales resultantes tras fragmentar los vídeos utilizaremos:

idActor_numFrame.bmp

Donde:

idActor es el identificador del vídeo o actor principal.
numFrame el número de fotograma del vídeo.

Por ejemplo:


01_0873.bmp


05_1090.bmp


09_1305.bmp

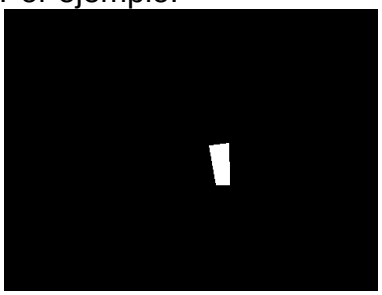
Con todo esto logramos un gran conjunto de imágenes de 480 x 360 píxeles por cada vídeo. Uno de los pilares de la base de datos HuPBA es el etiquetaje de las extremidades que forman el cuerpo humano, distinguiremos hasta 14 distintas que almacenaremos en forma de máscaras binarias y que formarán lo que denominamos Ground Truth, que como en el caso de las imágenes originales tendrán su propia nomenclatura para su correcta identificación.

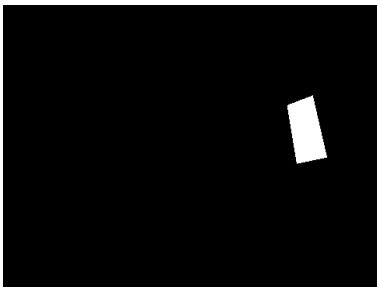
idActor_numFrame_idUser_idLimb.bmp

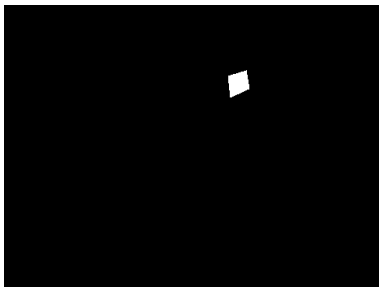
Donde:

idActor es el identificador del vídeo o actor principal.
numFrame el número de fotograma del vídeo.
idUser es el identificador de la persona que aparece en la imagen.
idLimb es el identificador de la extremidad en cuestión.

Por ejemplo:


01_0873_1_13.bmp


05_1090_2_2.bmp


09_1305_2_1.bmp

Para cada una de las 14 extremidades queda asignada una id, se muestran a continuación:

ID	Extremidad
1	Cabeza
2	Torso
3	Mano izquierda
4	Mano derecha
5	Antebrazo izquierdo
6	Antebrazo derecho
7	Brazo izquierdo
8	Brazo derecho
9	Pie izquierdo
10	Pie derecho
11	Pierna izquierda
12	Pierna derecha
13	Muslo izquierdo
14	Muslo derecho

Antes de empezar con la fase de etiquetaje, se deben de considerar varios factores, por un lado no es difícil darse cuenta de que la tarea de etiquetar 14 extremidades para cada una de las imágenes que forman los 9 vídeos es bastante tediosa. Por otro lado, si se analizan los fotogramas consecutivos el cambio de uno a otro es ínfimo, estos dos factores nos han llevado a considerar un etiquetado en el cual se obvia un fotograma entre fotogramas etiquetados. De esta manera se puede reducir, por tanto, el total de imágenes a etiquetar y en consecuencia el tiempo a invertir.

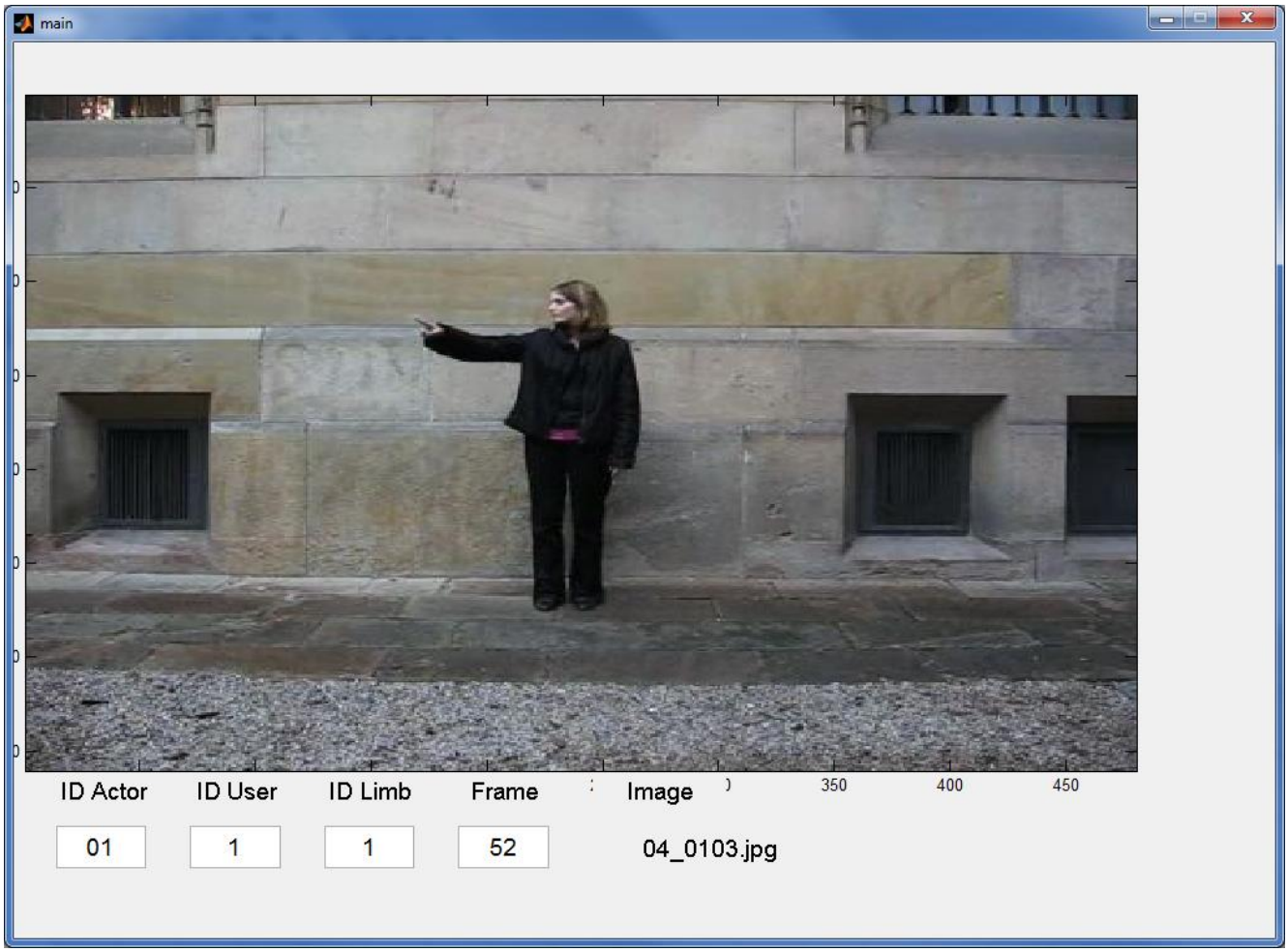
Hay otro aspecto de vital importancia que hay que tener muy en cuenta y este es el formato de las máscaras generadas en el proceso de etiquetaje. Uno de los formatos para archivos de imágenes más extendido es el *jpeg*, debido a la compresión que realiza supone una significativa reducción de peso de los ficheros. Por contra, esta compresión causa una pérdida de información.

En nuestro caso en concreto, guardar las máscaras en *jpeg* implica distorsionar las fronteras de nuestra región de interés y acarrea una serie de problemas a la hora de realizar operaciones sobre ellas. Por este motivo, se ha utilizado el formato *bmp*, que aunque aumenta considerablemente el peso total de nuestra base de datos, mantiene toda la información de la imagen.

2.2.1 Proceso de Etiquetaje

Para llevar a cabo esta labor se creó una pequeña aplicación en Matlab. Lo primero que se debe de hacer es generar en el directorio raíz de la aplicación una carpeta llamada */mask,* que será donde se almacenen las máscaras resultantes, hay que generar también una carpeta llamada */originales*, en ella se añadirán todos los fotogramas que forman el vídeo que se va a etiquetar.

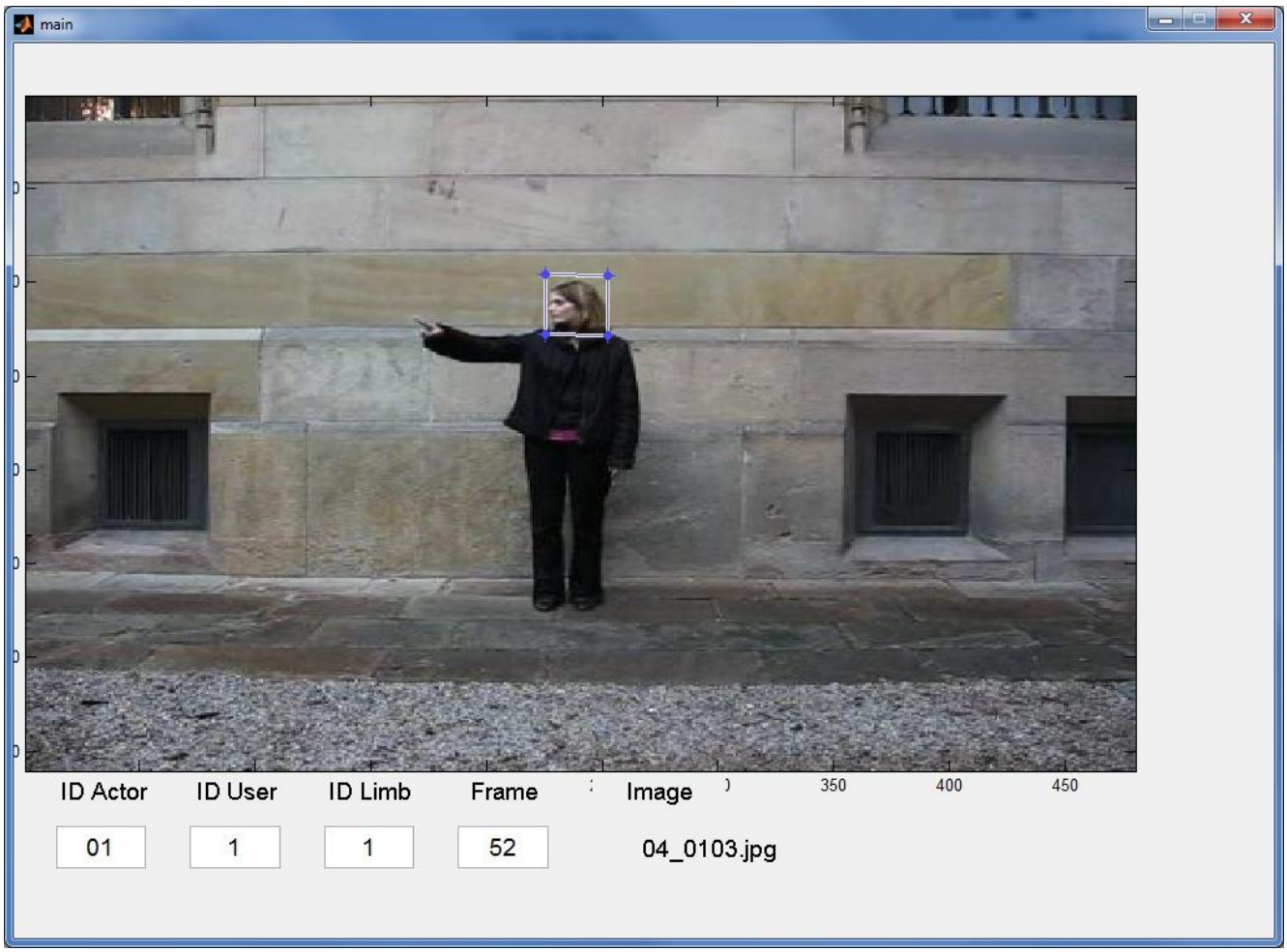
A continuación ejecutamos el programa y nos encontramos con una pantalla tal como esta:



En el panel superior vemos la imagen que se va a etiquetar, pulsando las teclas A y D de nuestro teclado pasaremos al fotograma anterior o siguiente respectivamente.

En la parte inferior podemos observar una serie de campos editables, en ellos debemos poner la información que aparecerá en el nombre de la máscara anteriormente comentado: la id del actor, el número de usuario, la id de extremidad, y el número de frame, este último se carga de forma automática a medida que avanzamos o retrocedemos con el teclado. El último campo es el nombre de la imagen original que tenemos en pantalla actualmente.

Pulsando la tecla W, entraremos en el modo dibujo, y podremos realizar nuestro polígono sobre la extremidad que estemos etiquetando. Una vez cerrado el polígono podemos desplazarlo libremente por la imagen con nuestro ratón, para una mayor flexibilidad a la hora de hacerlo coincidir con nuestra extremidad. El aspecto sería el siguiente:



Finalmente, pulsando la tecla S, se genera la máscara deseada y salta al siguiente fotograma, permitiendo continuar con el etiquetaje. Existen otras combinaciones de teclas que permiten mejorar la experiencia de etiquetaje como puede ser copiar el polígono anteriormente generado para saltos de fotograma en que la extremidad no ha variado demasiado, pueden moverse los vértices de nuestro polígono de forma independiente o todos a la vez pero con una mayor precisión que con el ratón.

El modo de proceder que ha resultado más eficaz ha sido el de etiquetar toda una sola extremidad para todo el vídeo, de esta forma se gana algo de inercia en el etiquetaje puesto que permite concentrarse en la forma de la extremidad en cuestión.

Primero junto a Jordi Suñé Fontanals y luego con Daniel Sánchez Abril, nos hemos encargado de llevar a cabo está fase de la creación de la base de datos. Tras ella, nos encontramos para cada uno de los vídeos con un directorio que incluye las imágenes RGB originales y otro con todas las máscaras generadas para las catorce extremidades de cada uno de las personas que aparecen.

El siguiente paso es el etiquetaje de los gestos. En este caso, la tarea es mucho menos tediosa que la anterior. El modo de proceder es el de analizar cada una de las imágenes RGB originales y almacenar el periodo que dura cada uno de los gestos. Utilizamos un fichero CSV (columnas separadas por ';') para guardar esta información.

El nombre del fichero:

idActor_gestures.csv

Donde:

idActor es el identificador del vídeo o actor principal.

En el interior del fichero:

idUser es el número de actor que está realizando el gesto.
Label_gesture es la etiqueta relacionada con el gesto.
start_frame es el número de fotograma en que comienza el gesto.
end_frame es el número de fotograma en que finaliza el gesto.

Las etiquetas para cada gesto:

ID	Gesto
1	Saludar
2	Señalar
3	Aplaudir
4	Agacharse
5	Saltar
6	Caminar
7	Correr
8	Dar la mano
9	Abrazar
10	Dar dos besos
11	Pelar

Una vez más, se ha colaborado con Jordi Suñé Fontanals y Daniel Sánchez Abril en esta parte. No porque se tratara de algo complicado sino por cambios que se han ido aplicando en varias fases del proyecto.

Otra parte importante del etiquetaje es la generación de la bounding box para cada persona. En este caso han sido mis dos compañeros ya nombrados los que se encargaron de generar el código y guardar dicha información en un nuevo fichero CSV.

El procedimiento consiste en unir en una sola imagen todas las máscaras pertenecientes a un mismo individuo en un mismo fotograma y encontrar el polígono mínimo que lo delimita, buscando las alturas y anchuras, máximas y mínimas de la máscara resultante.

El fichero CSV tiene el siguiente formato:

idActor_boundingbox.csv

Donde:

idActor es el identificador del vídeo o actor principal.

En su interior:

idUser es el número de actor de la escena.
Number_frame es el número de fotograma para el que generamos la boundig box.
x es la mínima coordenada X correspondiente a la parte superior izquierda de la persona.
y es la mínima coordenada Y correspondiente a la parte superior izquierda de la persona.
width ancho de la bounding box.
height alto de la bounding box.

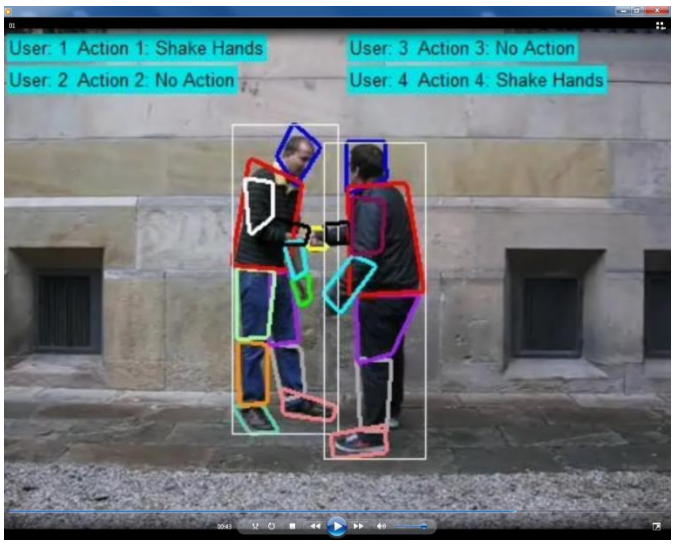
Y finalmente, entramos de lleno en la última fase, la validación de los datos.

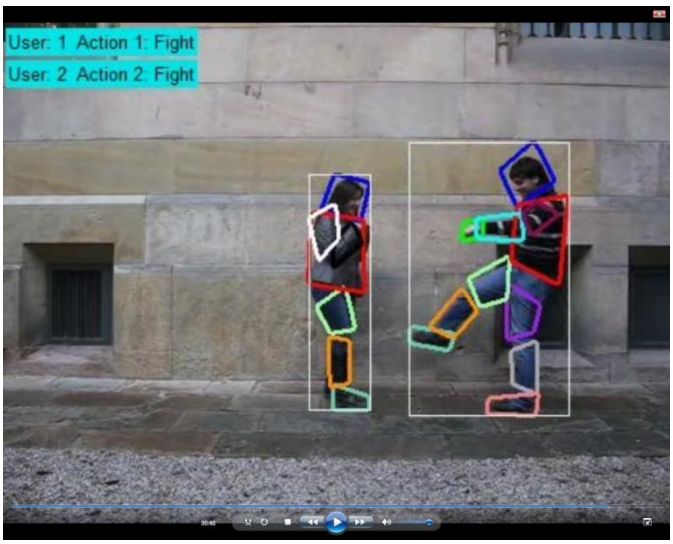
2.2.2 Validación de datos

Todas las fases anteriores, aunque no se trataban de tareas complicadas si eran tremendamente laboriosas y han requerido de una gran cantidad de tiempo y dedicación. Y precisamente por ese motivo, es de vital importancia validar los resultados obtenidos ya que son muy sensibles a pequeños errores de etiquetaje.

Por esta razón, Jordi Suñé y Daniel Sánchez se han encargado de generar una serie de vídeos en los que se pueden apreciar las bounding box, las extremidades y los gestos etiquetados sobre las imágenes RGB originales. Es un modo muy visual de analizar los resultados de cada una de las fases anteriores y poder realizar ciertas correcciones.

Un par de capturas de los vídeos en cuestión:





2.2.3 La base de datos en cifras

A continuación se muestran una serie de cifras que muestran un poco el volumen de nuestra base de datos.

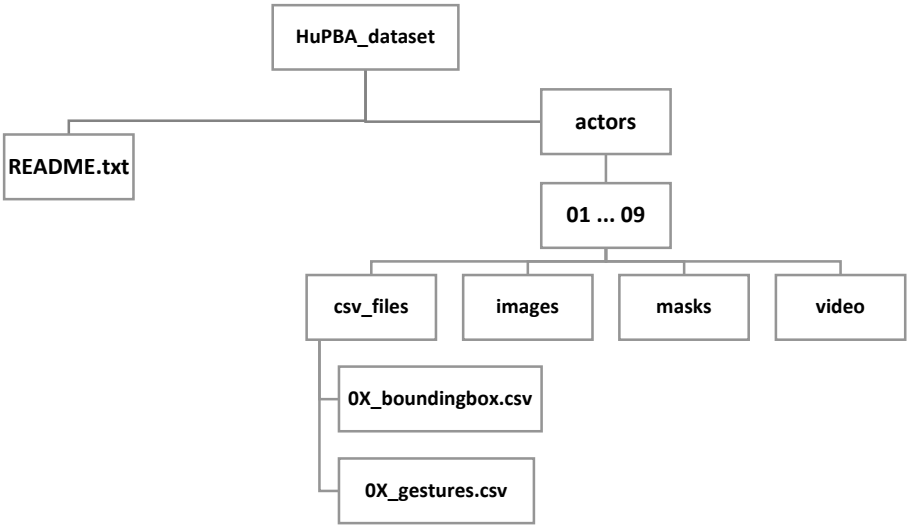
Imágenes RGB originales:

Vídeo	Fotogramas
01	906
02	866
03	1177
04	885
05	868
06	889
07	814
08	981
09	848
Total	**8234**

Máscaras generadas:

Vídeo	Máscaras
01	13771
02	12237
03	16824
04	15009
05	13325
06	11753
07	12969
08	14520
09	14311
Total	**124719**

Estructura de directorios:



2.3 Procesamiento de los datos de entrada

El paso previo a la extracción de características y el entrenamiento, es recortar cada una de las extremidades y rotarlas en función del ángulo dominante. Concretamente, es importante realizar primero la rotación y luego el recorte puesto que de otro modo surgen problemas de fiabilidad.

Hay que tener en cuenta que una vez recortada una extremidad, obtenemos una región que puede no ser cuadrada, como puede ser un brazo o una pierna. Por este motivo, dichas regiones son recortadas con el máximo valor de lado dado por la extremidad, de forma que consigamos regiones cuadradas y mantener así una homogeneidad en todas las extremidades.

Tras este proceso tenemos las regiones cuadradas y rotadas listas para proceder a la extracción de características por parte de los descriptores.

Algunos ejemplos de extremidades listas:

	Vídeo 1	Vídeo 2	Vídeo 3	Vídeo 4	Vídeo 5	Vídeo 6	Vídeo 7	Vídeo 8	Vídeo 9
Torso									
Cabeza									
Brazo									
Antebrazo									
Muslo									
Pierna									

De las 14 extremidades etiquetadas en la base de datos, se ha optado por reducir el número hasta 6, agrupando las que son similares por simetría (derecha-izquierda) como brazos, antebrazos, muslos, piernas y descartando otras como pies y manos, estás últimas debido su pequeño tamaño y el problema que supone su redimensión en cuanto a pérdida de calidad de la imagen se refiere.

El siguiente cuadro muestra la reagrupación de las extremidades:

1	Torso
2	Cabeza
3	Antebrazos
4	Brazos
5	Muslos
6	Piernas

3. Metodología

A continuación se procederá a la descripción detallada de los métodos utilizados. Primero se explicará el funcionamiento de los clasificadores en cascada *Adaboost* junto a la extracción de características *Haar*, a continuación el funcionamiento de los clasificadores SVM, pasando por la extracción de características HOG y finalmente se detallarán los ECOC.

3.1 Aprendizaje por Cascadas Adaboost

En este punto se describe el uso de las características Haar y el funcionamiento del algoritmo clasificador basado en una cascada de clasificadores débiles.

3.1.1 Introducción

Adaptative Boosting (Adaboost), es un algoritmo de clasificación formulado por Yoav Freund y Schapire Robert que se puede utilizar junto a muchos otros algoritmos de aprendizaje para mejorar su rendimiento.

Adaboost es un sistema constituido por varios clasificadores menores denominados “débiles”, que por sí solos no podrían constituir un sistema de clasificación debido a su alta inexactitud, pero que combinándolos permite la construcción de un clasificador mucho más preciso. Se trata, por tanto, de un clasificador adaptativo, que va ajustando su clasificación a través de los clasificadores “débiles”, adaptándose al conjunto de datos de entrenamiento.

Para la detección de extremidades se ha optado por la utilización de los clasificadores *Adaboost* en combinación con la conocida cascada de clasificadores de Viola & Jones para obtener una predicción muy robusta [2]. Cabe destacar que se trata de un algoritmo de clasificación binario y que por tanto, como se describirá posteriormente, se necesitarán otros métodos para solventar el problema multi-clase.

3.1.2 Características Haar

Previo al aprendizaje, es necesaria una extracción de características que describa la imagen de entrada. Las características *Haar* consisten en varias regiones blancas y negras en forma de rectángulo que se aplican a un conjunto de píxeles y en las cuales se definirá si se suma o se resta la luminancia (valor de los píxeles en escala de grises) de los píxeles afectados en cada una de las partes. La diferencia entre la suma de luminancia de la parte negra y la suma de la luminancia en la parte blanca da lugar al valor de la característica.

Se pueden obtener, por tanto, a partir de los rectángulos comentados las características de la imagen como evaluaciones de la intensidad del conjunto de píxeles. Todas estas características pueden ser rápidamente calculadas utilizando una representación intermedia de la imagen llamada imagen integral. En la Fig. 1 se pueden observar ejemplos de características *Haar*.

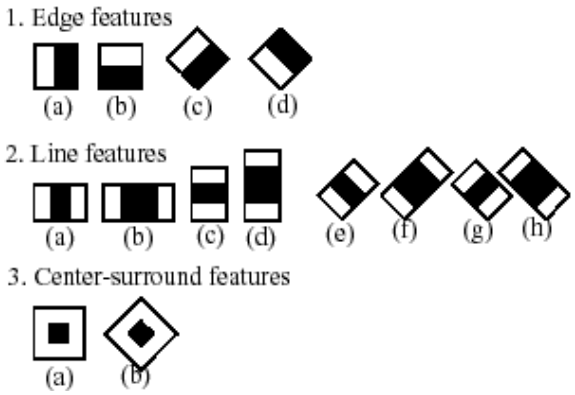


Fig. 1 – Representación de características Haar

3.1.3 Cálculo de la imagen integral

La ya comentada imagen integral fue una de las contribuciones de Viola-Jones y pueden definirse como tablas de búsqueda de dos dimensiones en forma de matriz con el mismo tamaño que la imagen original.

Cada elemento de la imagen integral contiene la suma de todos los píxeles situados en la región superior izquierda de la imagen original. Esto permite, con cuatro búsquedas, calcular la suma de las áreas rectangulares en la imagen en cualquier posición o escala.

La Fig. 2 muestra un ejemplo de cálculo de la imagen integral.

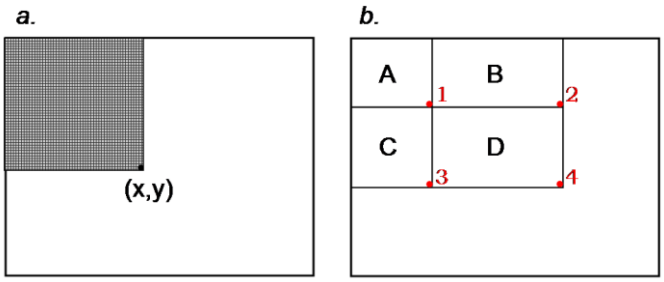


Fig. 2: Imagen integral

También representado por la fórmula:

$$i_{integral(x,y)} = \sum_{a \leq x, b \leq y} i_{source}(a,b)$$

El resultado de este cálculo es una imagen en escala de grises de muy rápida generación ya que se puede realizar en tiempo lineal. Como se tratan de operaciones de sumas y restas de valores enteros en regiones, la ejecución es muy eficiente, lo que permite descripciones en tiempo real.

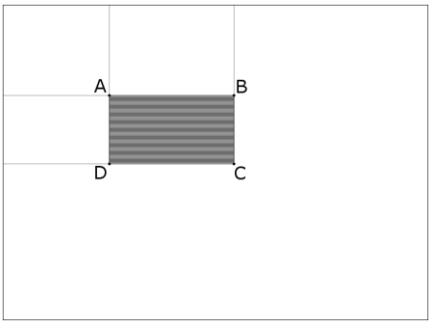


Fig. 3 Área que se debe calcular

El cálculo del área representada en la Fig. 3 se realizaría con la siguiente expresión:

$$\sum_{\substack{A(x)<x'\leq C(x) \\ A(y)<y'\leq C(y)}} i(x'y') = sum(A) + sum(C) - sum(B) - sum(D)$$

3.1.4 Aprendizaje de las características

Para entender mejor el aprendizaje de características *Haar* por parte de *Adaboost*, se detallará la implementación del detector de caras realizada por Viola-jones basado en dicho clasificador.

Como ya se ha comentado en la introducción de este punto, la detección de rostros propuesta por Viola-Jones se caracteriza por su robustez y alta velocidad. Está basada en una cascada de clasificadores que explora la imagen a múltiples escalas y localizaciones. Cada etapa de la cascada utiliza características *Haar* seleccionadas y combinadas mediante *Adaboost* durante la fase de entrenamiento. El funcionamiento de una cascada de estas características consiste en un conjunto de clasificadores que se aplican en distintas etapas.

La entrada de ejemplos positivos y negativos depende de los resultados obtenidos de la clasificación de la etapa anterior consiguiendo, de esta forma, que cada clasificador se centre en los ejemplos que el anterior no ha conseguido aprender. El objeto que consiga superar todos los niveles será clasificado. La eficiencia de este sistema radica en que la gran mayoría de ejemplos negativos que genera la exploración de la ventana son eliminados rápidamente en las primeras etapas del proceso. La Fig. 4 muestra un esquema de funcionamiento del sistema anterior.

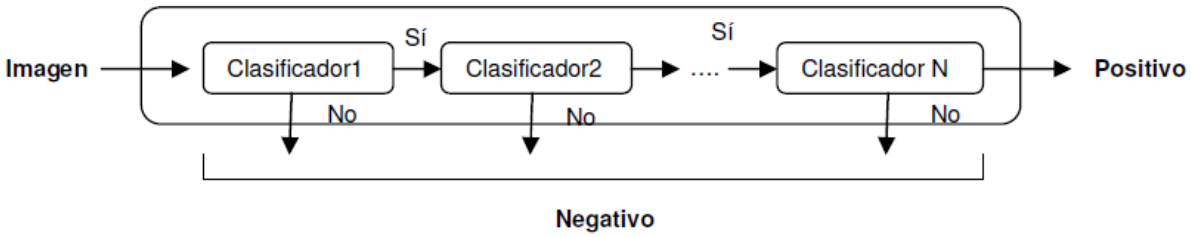


Fig. 4 – Cascada de detectores diseñada por Viola-Jones

3.1.5 Algoritmo Adaboost

En esta sección se profundiza en el algoritmo *Adaboost* detallando todas las etapas que lo componen. Inicialmente se define el conjunto de entrenamiento como:

$$(x_1, y_1), \dots, (x_m, y_m);$$

Donde cada x_i pertenece al espacio de características X y cada etiqueta y_i pertenece a un set de etiquetas Y. Simplificando la expresión los identificadores de las clases pueden ser +1 o -1, por tanto:

$$Y = \{+1, -1\}$$

Adaboost se genera a partir de un algoritmo de aprendizaje estadístico ***en t = 1, … T*** ciclos. Sobre el set de entrenamiento se establece un conjunto de pesos que se actualizan con cada iteración del algoritmo. Así pues, el peso sobre el ejemplo *i* en el ciclo *t* sería $D_t(i)$.

Inicialmente, el peso establecido será de $1/m$, donde *m* es el número total de elementos del set de entrenamiento. Con tal de obligar al siguiente clasificador a enfocarse en los ejemplos mal clasificados y por tanto, más difíciles, los pesos de estos aumentarán.

La tarea principal del clasificador "débil" es encontrar una hipótesis que seleccione para cada conjunto D_t la solución que tenga un mínimo error. Dicho error se obtiene con respecto al conjunto para el que el clasificador fue entrenado. Además se trata de la suma de los ejemplos mal clasificados. La respuesta de este clasificador débil viene dada por la expresión:

$$H(x) = sign\left(\sum_{t=1}^{T} \propto_t h_t(x)\right)$$

Con la hipótesis h_t completa, adaboost elige el parámetro $\propto$ midiendo la importancia que se asigna a este clasificador débil. A continuación, como ya se ha comentado anteriormente, se actualizan los pesos mediante la expresión:

$$D_{t+1}(i) = \frac{D_t(i)^{-\propto_t y_i h_t(x_i))}}{Z_t}$$

De este modo se incrementa el peso de los ejemplos mal clasificados y se disminuye el de los mal clasificados, concentrando así los pesos en los ejemplos más difíciles de aprender.

La hipótesis final o clasificador fuerte es la suma de la mayoría de pesos de las hipótesis *T* débiles, donde $\propto$ es la importancia asignada al clasificador h_t.

El clasificador retornará una salida binaria en la que se indicará la clase a la que pertenece el ejemplo de entrada.

3.1.6 Factores que afectan al clasificador

La Fig.11 representa una cascada de clasificadores *Adaboost*.

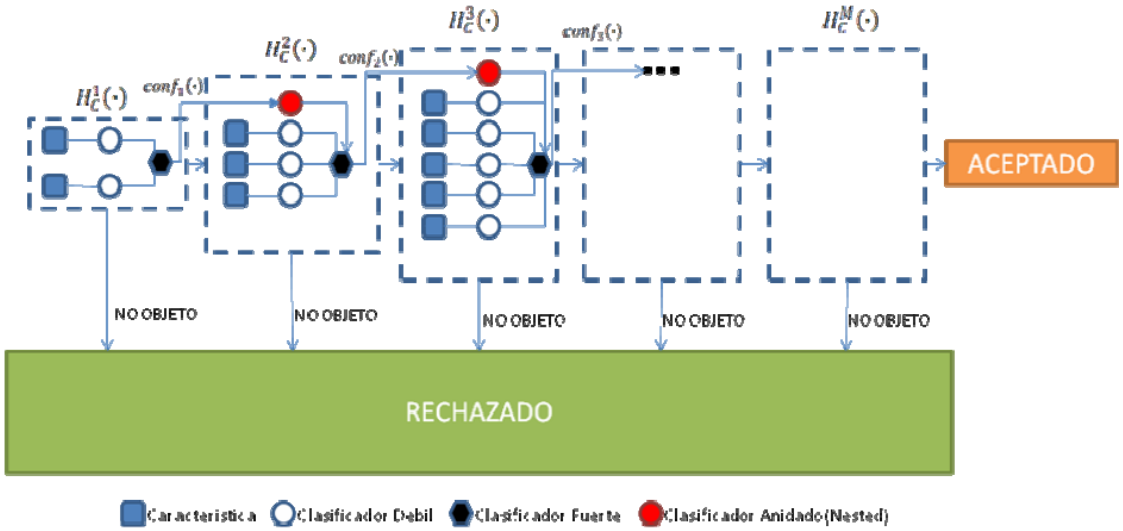


Fig. 5 – Diagrama de un clasificador Adaboost en cascada

Para mejorar los resultados, es importante la forma en que se monta la cascada, *Bootstraping* es una de estas formas.

Boostraping entrena cada etapa iterativamente añadiendo todos los ejemplos clasificados de forma errónea a la base de entrenamiento, de este modo el clasificador se entrena con ejemplos cada vez más difíciles, mejorando la calidad del conjunto de entrenamiento.

Por ejemplo, suponiendo que se está detectando un objeto en concreto, posibles candidatos a un ejemplo negativo serán todas aquellas partes de la imagen en las que no se encuentra dicho objeto, no obstante, lo que se desea en este caso es localizar aquellas partes que, aun siendo negativas, sean las que más se parezcan a la clase positiva.

Otro parámetro importante en este método es el número de etapas o clasificadores débiles que forman la cascada.

Si utilizamos un número reducido de estos, corremos el riesgo de pasar ejemplos negativos como positivos ocasionando serios problemas de detección. El proceso de filtraje que se realiza en todas etapas sería insuficiente para distinguir claramente cualquier ejemplo de entrada y por tanto tendríamos una cascada que ha aprendido poco.

En caso contrario, si utilizáramos demasiadas etapas de clasificadores implicaría que se está sobre-aprendiendo todo el conjunto de datos y por tanto el sistema sólo será capaz de detectar aquellos ejemplos que sean exactamente igual a los datos de entrenamiento.

3.2 Aprendizaje por Clasificadores SVM

A continuación se describe el funcionamiento del algoritmo clasificador *Support Vector Machines* o Máquinas de Soporte Vectorial (SVM) utilizado en este proyecto para implementar el sistema de aprendizaje de extremidades.

3.2.1 Introducción

Las SVM son un conjunto de algoritmos enfocados a problemas de aprendizaje, específicamente en clasificación y regresión. Estos métodos fueron desarrollados por Vladimir Vapnik y su equipo en la compañía AT&T, se presentaron en 1992 obteniendo unos resultados muy superiores al resto de opciones en aquella época, hecho que les hizo ganar una gran popularidad.

Las aplicaciones de estos algoritmos son innumerables, no obstante estas son algunas de ellas:

- Reconocimiento de caracteres, rostros, matrículas, …
- Clasificación de exámenes radiológicos, TAC, …
- Predicción de genes en el campo de la genética
- Clasificación de documentos
- Realizar predicciones en la economía

Como se ha comentado en la introducción, las SVM pertenecen a la rama de aprendizaje supervisado, es decir, requieren de datos de entrenamiento etiquetados para obtener un modelo capaz de predecir la etiqueta de una nueva muestra de datos no entrenada. Como en el caso de *Adaboost*, SVM es un clasificador binario y tal y como se explicará posteriormente, se necesitarán herramientas adicionales para abordar el problema de la multi-clasificación.

3.2.2 Funcionamiento

Como se comentó en secciones anteriores las SVM obtienen sus predicciones a partir de un modelo aprendido en base a los datos de entrenamiento. Dicho modelo se genera en la fase de entrenamiento y se trata de un hiperplano que separa las dos categorías a discriminar.

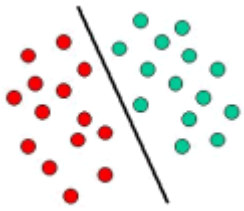


Fig. 11 – Ejemplo de espacio de características y modelo aprendido por SVM

Observando la Fig. 11, supongamos que los puntos son datos obtenidos de nuestra base de datos, los puntos rojos simbolizan ejemplos de cabeza y los verdes ejemplos de brazo mientras que la línea negra representaría el hiperplano que separa de forma óptima los datos de cada clase. Considerando como forma de separación óptima, aquella que obtenga para el hiperplano, un mayor margen entre los puntos más próximos de cada clase.

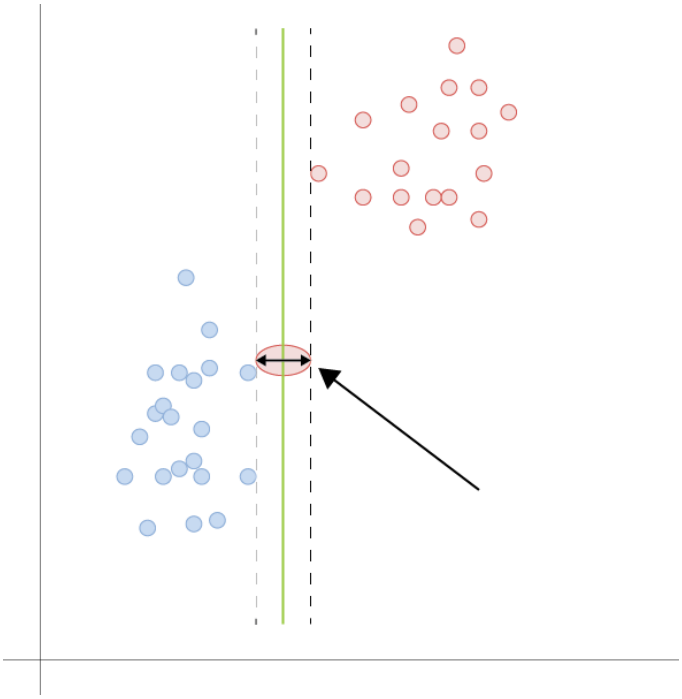


Fig. 12 – Margen pequeño

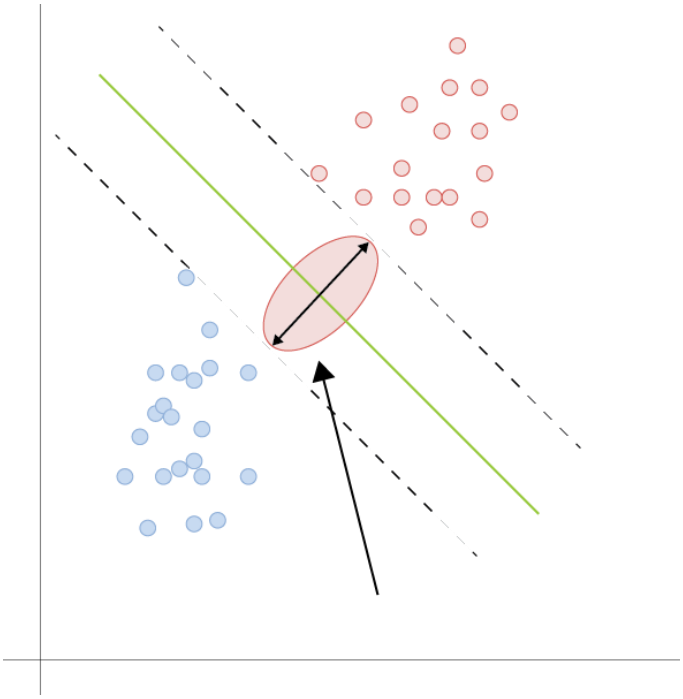


Fig. 13 – Margen grande

Aunque se pueden trazar infinitos hiperplanos para separar las dos clases de datos, tal y como se puede observar en las Fig. 12 y Fig. 13, se trata de encontrar el que consiga una mayor distancia entre clases.

Este problema de optimización puede ser expresado de la siguiente forma [9]:

$$min\,\rho(w,b) = \frac{1}{2}w^2$$

$$sujeto\ a\ \forall_i\ \ y_i w^T \varphi(x_i) + b)\ \geq 1$$

Resolver este problema directamente es difícil debido a las complejas restricciones. Una herramienta matemática que simplifica el problema es la Teoría de Dualidad de Lagrange, la cual nos conduce a:

$$max\ D(\propto) = \sum_{i=1}^{n} \propto_i - \frac{1}{2}\sum_{ij=1}^{n} y_i \propto_i y_j \propto_j \varphi(x_i)^T \varphi(x_j)$$

$$sujeto\ a \begin{cases} \forall_i\ \propto_i \geq 0, \\ \sum_i y_i \propto_i = 0. \end{cases}$$

El problema es ahora, computacionalmente hablando, más simple de resolver. La dirección w* del hiperplano óptimo es obtenida por una solución α* del problema de optimización dual:

$$w^* = \sum_i \propto_i^* y_i \propto_i \varphi(x_i)$$

Determinar la mejor b* se convierte, entonces, en un simple problema unidimensional. La función discriminante lineal se puede expresar como:

$$y(x) = w^{*T}x + b^* = \sum_{i=1}^{n} y_i \propto_i^* \varphi(x_i)^T \varphi(x) + b^*$$

3.2.3 Separación entre los datos

En las Figuras 12 y 13 se ilustra perfectamente el funcionamiento de SVM en el momento de realizar la separación de las clases, pero se trata de un caso ideal el cual no es para nada común en problemas reales. De esta manera, en prácticamente la mayoría de ocasiones los datos de entrenamiento no van a ser linealmente separables en el espacio de características.

Para abordar dicho problema, SVM se sirve de una función de Kernel. Esta técnica consiste en proyectar los datos en un espacio de mayor dimensionalidad que el espacio de características original, lo cual aumenta la separabilidad de los datos y hace posible el uso de clasificadores lineales.

3.2.4 Ejemplo visual de funcionamiento

Veamos un ejemplo visual de cómo realizar una separación entre clases que no es posible realizar mediante un clasificador lineal como podemos observar en la Fig.14.

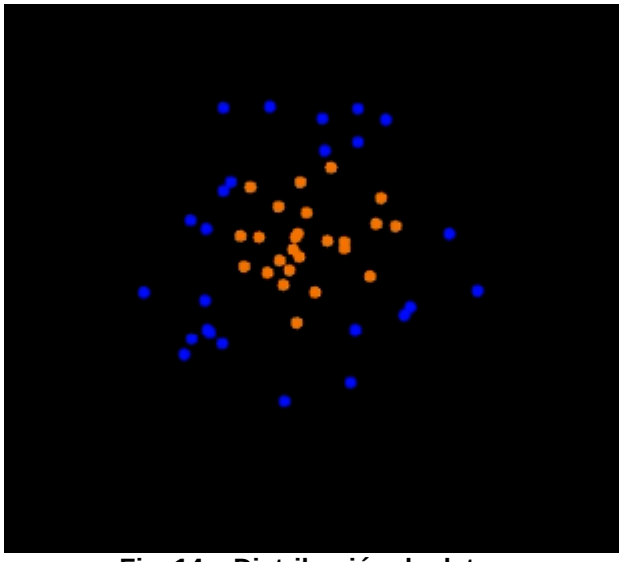


Fig. 14 – Distribución de datos

Como se ha comentado, la solución pasa por representar la información en un espacio de mayor “dimensionalidad”. De ello se encarga un kernel que proyectaría los datos como podemos observar en la Fig. 15.

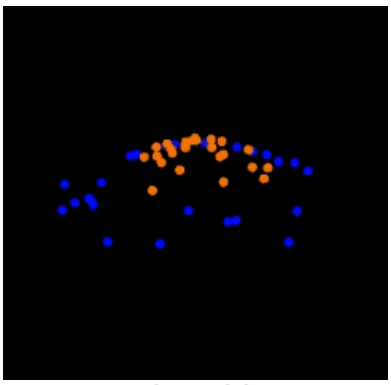


Fig. 15(a)

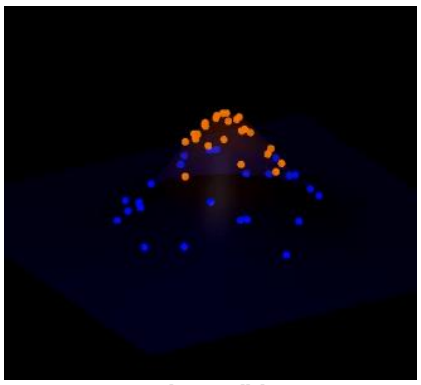


Fig. 15(b)

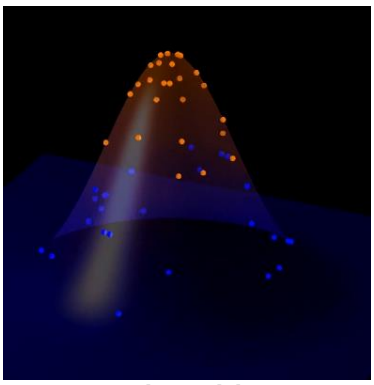


Fig. 15(c)

Proyección de los datos en un espacio cuadrático

Una vez se ha realizado la transformación ya podemos separar las clases con un hiperplano. Volviendo a representar los datos en un espacio con la dimensionalidad inicial podemos observar la clasificación final de los datos.

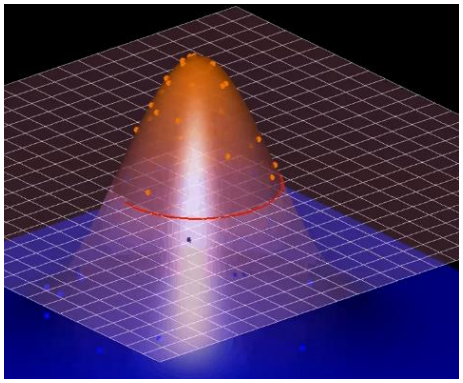


Fig. 16(a)

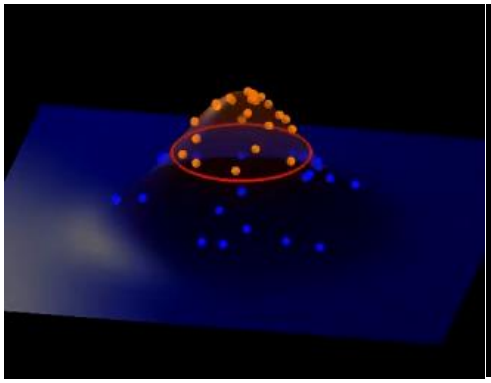


Fig. 16(b)

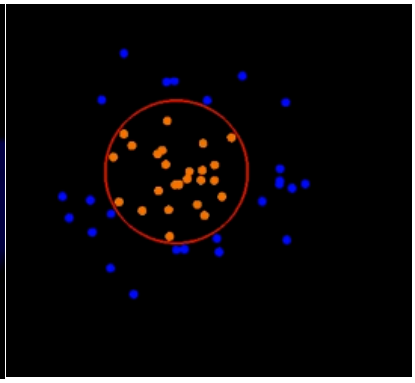


Fig. 16(c)

Volviendo al espacio de dimensiones originales

3.3 Uso del descriptor HOG

A continuación se explicará el funcionamiento del descriptor HOG que es el encargado de describir las imágenes, convirtiéndolas en vectores numéricos que son posteriormente utilizados en la clasificación mediante SVM.

3.3.1 Introducción

En anteriores secciones hemos visto como SVM genera un modelo para realizar la clasificación, pero para aprender dicho modelo son necesarios un conjunto de datos de aprendizaje. Estos datos son imágenes pero es necesario hacer una descripción y representarla de un modo numérico para trabajar con ellas. Uno de los métodos para realizar dicha descripción es HOG (Histogram Oriented Gradients), muy utilizado junto a SVM en la detección de objetos debido a sus buenos resultados.

HOG se basa en que cualquier imagen puede ser descrita como una distribución de las direcciones de cambio de intensidad. Lo que HOG realiza es dividir la imagen de entrada en varias regiones conectadas de un mismo tamaño que llamaremos celdas. Para cada una de estas celdas se calcula un histograma de todas las orientaciones detectadas en función del cambio de intensidad.

Combinando todos estos histogramas obtenemos lo que llamamos descriptor de la imagen, que permite almacenar esta información en forma de vector numérico. Para minimizar los errores y en consecuencia mejorar los resultados, HOG realiza un pre procesamiento que calcula la media de la intensidad de la imagen para cada celda, de modo que haya menos variancia en la intensidad en toda la región de la imagen.

La ventaja de HOG frente a otros descriptores radica en el hecho de que se utilicen varias regiones en vez de analizar la imagen entera, esto hace que afecte menos la forma geométrica del objeto a detectar. Además el cálculo de las distintas regiones se puede paralelizar ya que los cálculos de cada celda son independientes del resto, aumentando el rendimiento considerablemente.

3.3.2 Implementación del algoritmo HOG

Lo primero es calcular los gradientes de la imagen, suavizando la imagen con un filtro Gausiano y aplicando una máscara que realice un la derivada de la imagen. Por ejemplo la máscara Sobel, que devolverá impulsos en aquellas zonas en que haya grandes cambios de intensidad.

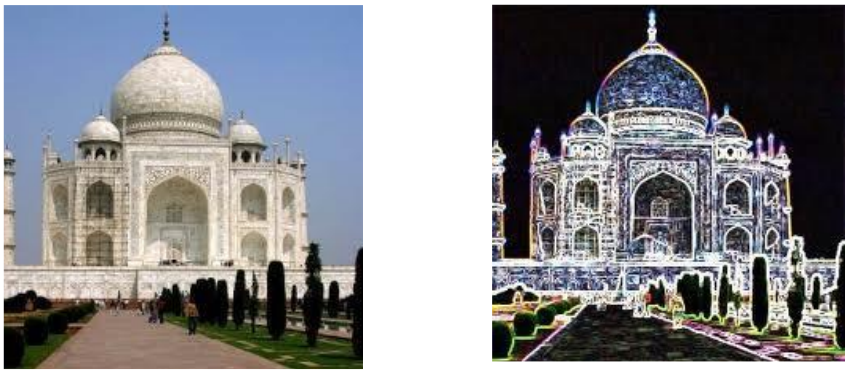


Fig. 17 – Resultado de la operación Sobel

Una vez ha sido aplicada y tenemos la imagen resultante con los valores de gradiente, se deben calcular las orientaciones. Como ya se ha comentado, la imagen de gradientes se divide en varias celdas, para cada una se calculará un histograma con la distribución de orientaciones, que van de 0º a 180º o de 0º a 360º, en función de si son ángulos con o sin signo.

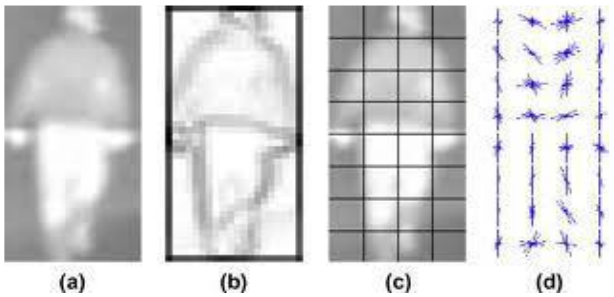


Fig. 18 – Ejemplo HOG

Las celdas se unen en bloques de mayor tamaño que darán como resultado descriptores mucho más completos y a su vez todos serán unidos en un único descriptor resultante mejorando así los resultados.

3.3.3 Observaciones sobre el tamaño del descriptor

Se ha utilizado la implementación de HOG que podemos encontrar dentro de las librerías OpenCV [3], para la que hay que predefinir parámetros como el tamaño de la imagen o el número de bloques y celdas. Además, es imprescindible que el tamaño del descriptor resultante sea siempre el mismo, por lo que estos parámetros no pueden alterarse en tiempo de ejecución lo cual supone un problema teniendo en cuenta que las extremidades etiquetadas son de multitud de tamaños. Por este motivo se ha decidido re-escalar todas las imágenes a un mismo tamaño antes de describirla. Esto se realiza tanto en la fase de entrenamiento para generar el modelo como en la de test efectuando la clasificación.

3.4 ECOC: Error – Correcting Output Codes (Códigos de Corrección de Errores)

A continuación se describirá el funcionamiento de los Códigos de Corrección de Errores con los que se aborda el problema de la multi-clase.

3.4.1 Introducción

ECOC ha demostrado ser un marco de trabajo sencillo a la par que potente haciendo frente a los problemas de multi-clasificación. Los buenos resultados obtenidos por el método han hecho que incluso se aplique en casos en que los clasificadores utilizados soporten la multi-clase por si solos.

Dado un problema de multi-clasificación de *N* clases, ECOC lo atacaría dividiéndolo en *n* subproblemas de menor complejidad, se tata de bi-particiones de las *N* clases. Como resultado a estas bi-particiones cada clase obtiene lo que se denomina como *palabra clave*, formada por tantos bits como clasificadores base y donde cada bit $\in \{+1, -1, 0\}$.

Si utilizamos cada una de las palabras clave anteriormente comentadas como filas de una matriz, se forma $M \in \{-1,1,0\}^{N\,x\,n}$ siendo *N* el número de clases y *n* el número de clasificadores. La Fig. 19 muestra un ejemplo de la estructura de una matriz ECOC.

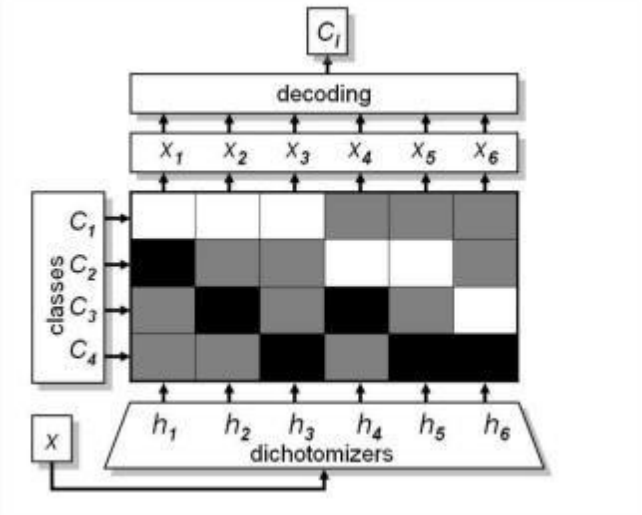


Fig. 19 – Ejemplo de estructura de matriz ECOC

De este modo el marco de trabajo de los ECOC se divide en dos etapas

- *Codificación*: Proceso con el que se obtiene la matriz *M* de codificación.
- *Decodificación*: Proceso mediante el cual se obtienen las nuevas predicciones corregidas.

3.4.2 Codificación ECOC

A continuación se explicarán las metodologías de codificación más importantes [11]:

- *Uno contra Todos:* Se trata de una de las estrategias más utilizadas. Consiste en entrenar cada clasificador base con el objetivo de distinguir una categoría del resto. Dado un conjunto de *N* clases, se entrenarían *N* clasificadores o lo que es la mismo, utilizar una longitud de palabra clave de *N* bits lo que generaría una matriz ECOC de *N* columnas.
- *Uno contra uno:* En este caso se introduce un tercer símbolo en el diseño de las matrices con el objetivo de excluir de las agrupaciones a determinadas clases. Este tercer símbolo requiere incrementar la longitud de los códigos si se quieren obtener resultados robustos.

3.4.3 Decodificación ECOC

El proceso de decodificación consiste en obtener una nueva *palabra clave* a partir de las predicciones realizadas por los clasificadores base. Esta nueva *palabra clave* será comparada con cada una de las filas de la matriz *M. La clase vinculada a la palabra clave* que sea más "parecida" a la recién obtenida será la que se asignará a la predicción final.

Dos posibles estrategias a seguir en la etapa de decodificación serían:

- *Decodificación Hamming (HD):* este tipo de decodificación trata de calcular la distancia Hamming entre la palabra predicha por los clasificadores base que forman el sistema ECOC y cada una de las filas que forman la matriz *M*. La clase asignada a la palabra que obtenga la menor de las distancias será la predicción final del sistema.

$$HD(x, y_i) = \sum_{j=1}^{n} \left(1 - sign\left(x^j \cdot y_i^j\right)\right)/2$$

- *Decodificación Euclídea:* Consiste en calcular distancia Euclídea de la palabra predicha por los clasificadores y cada una de las filas de la Matriz M. Al igual que en la estrategia anterior, la predicción final será la clase que obtenga la menor de las distancias.

$$ED(x, y_i) = \sqrt{\sum_{j=1}^{n} \left(x^j - y_i^j\right)^2}$$

3.4.4 La configuración de ECOC utilizada

A continuación se va a ejemplificar toda la teoría sobre ECOC comentada anteriormente mediante la configuración utilizada en este proyecto.

A partir de la agrupación de extremidades que se ha utilizado en este proyecto y que ha sido comentada en el apartado 2.3 se ha optado por la combinación de clases que muestran las figuras 20(a) y 20(b)

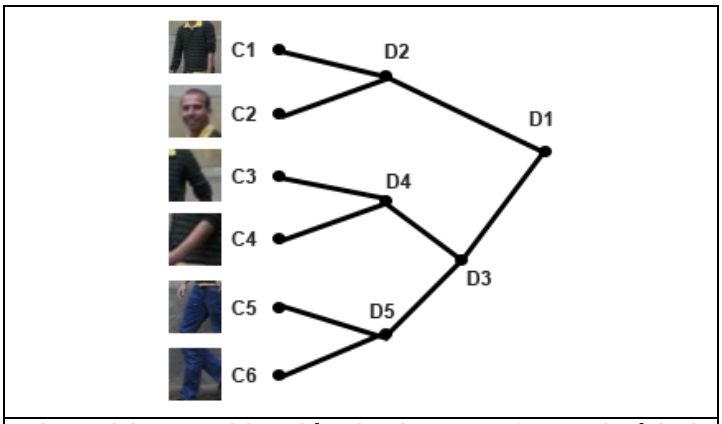


Fig. 20(a) – Combinación de clases en forma de árbol

Clasificador	Clase	Extremidades
D1	C1,C2 vs C3, C4, C5, C6	Torso, Cabeza VS Antebrazo, Brazo, Muslo, Pierna
D2	C1 vs C2	Torso VS Cabeza
D3	C3,C4 vs C5, C6	Antebrazo, Brazo VS Muslo, Pierna
D4	C3 vs C4	Antebrazo VS Brazo
D5	C5 vs C6	Muslo VS Pierna

Fig. 20(b) –Combinación de clases en forma de tabla

Estas son las combinaciones que forman la codificación de la Matriz ECOC utilizada en este proyecto. D1, D2, D3, D4 y D5 son los 5 clasificadores base que forman nuestro sistema. Cada una de las clases recibe una *palabra clave* que forman las filas de la matriz. La figura 21 muestra el resultado de la codificación de nuestra matriz ECOC.

	D1	D2	D3	D4	D5
C1	1	1	0	0	0
C2	1	-1	0	0	0
C3	-1	0	1	1	0
C4	-1	0	1	-1	0
C5	-1	0	-1	0	1
C6	-1	0	-1	0	-1

Fig. 21 – Codificación de la matriz ECOC utilizada en este proyecto

Finalmente, para proceder con la decodificación de la matriz, se ha calculado la distancia entre la *palabra clave* predicha y cada una de las filas que forman la matriz mediante la siguiente expresión:

$$d(x,y) = \frac{\sum_{i=1}^{5} -x^i \cdot y^i \cdot W^i}{\sum |Y|}$$

Cabe destacar que la variable *W* de la expresión anterior, corresponde al peso de cada uno de los clasificadores base que forman el sistema ECOC. Dichos pesos representan cuan bien han aprendido sus datos de entrada dichos clasificadores. Por tanto, es importante guardar estos parámetros durante la fase de entrenamiento.

3.5 GrabCut

A continuación se detallará grabCut y el método en que está basado: graph cut.

3.5.1 Graph cuts

Graph cut (o corte de grafos) es el método en que se basa GrabCut, y fue implementado por primera vez en el campo de la visión por computador por Greig, Porteous y Seheult de la universidad de Durham, con la intención de reducir el ruido de imágenes de mala calidad. Puesto que se trataba de imágenes binarias, el algoritmo produjo una solución exacta [4].

Graph cut es un método de segmentación de imágenes basado en regiones que resuelve de modo eficiente, dentro del campo de la visión por computador, una gran variedad de problemas como pueden ser el suavizado de imágenes o la asignación de etiquetas a píxeles [5].
Los pasos del método graph cut son:

1. *Segmentación de imágenes:* Se divide una imagen en varias regiones de modo que sean conexas, disjuntas y que cumplan ciertos criterios de homogeneidad.

2. *Clasificación de regiones:* Identificar cada una de las regiones como parte de una categoría o clase.

3. *Etiquetado de regiones:* Asignación de una etiqueta distinta a cada componente conexa.

4. *Conectividad entre píxeles:* Se considera que dos píxeles de un conjunto están conectados si existe un camino que une los dos píxeles y el todos los píxeles del camino están en el mismo conjunto

    a. *Conectividad 4 (utilizada en segmentación):* cada píxel tiene 4 vecinos en horizontal y en vertical.

    b. *Conectividad 8:* Cada píxel tiene 8 vecinos, en vertical, horizontal i diagonal.

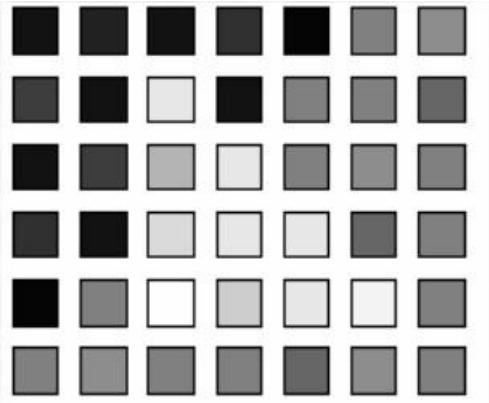


Fig. 22(a) Etiquetado de una imagen: escala de grises

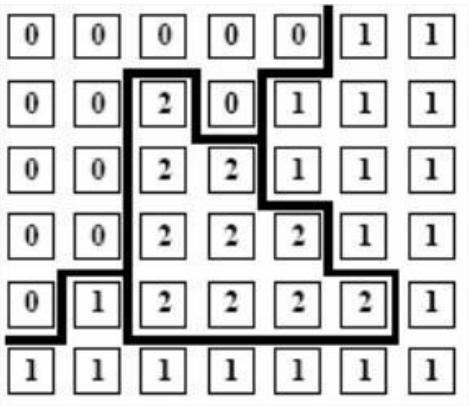


Fig. 22(b) Etiquetado de una imagen: asignación de etiqueta

En las Fig. 22(a) y 22(b) ilustran un ejemplo de asignación de etiquetas a una imagen en escala de grises en función de su intensidad. Podemos observar como a las tres intensidades predominantes (negro, gris, gris claro) se les asigna una etiqueta para diferenciarlas de las demás. Cuando todos los píxeles tienen asignados una etiqueta, ya disponemos de tres clases distintas.

3.5.2 Descripción

GrabCut fue desarrollado en el año 2004 por Carsten Rother, Vladimir Kolmogorov y Andrew Blake en Microsoft Research Cambridge, UK. Es un método de segmentación binaria y por tanto utilizado para resolver problemas en los que hay que separar únicamente dos clases.

Todo lo que debe de hacer el usuario para realizar la segmentación es dibujar un rectángulo alrededor del objeto de interés. GrabCut devolverá esa separación con dos tipos de etiquetas distintos, una para el objeto en cuestión y otra para el fondo.

La Fig. 23 compara varios métodos de segmentación destacando la simple interacción con el usuario por parte de GrabCut con respecto al resto de métodos [6].

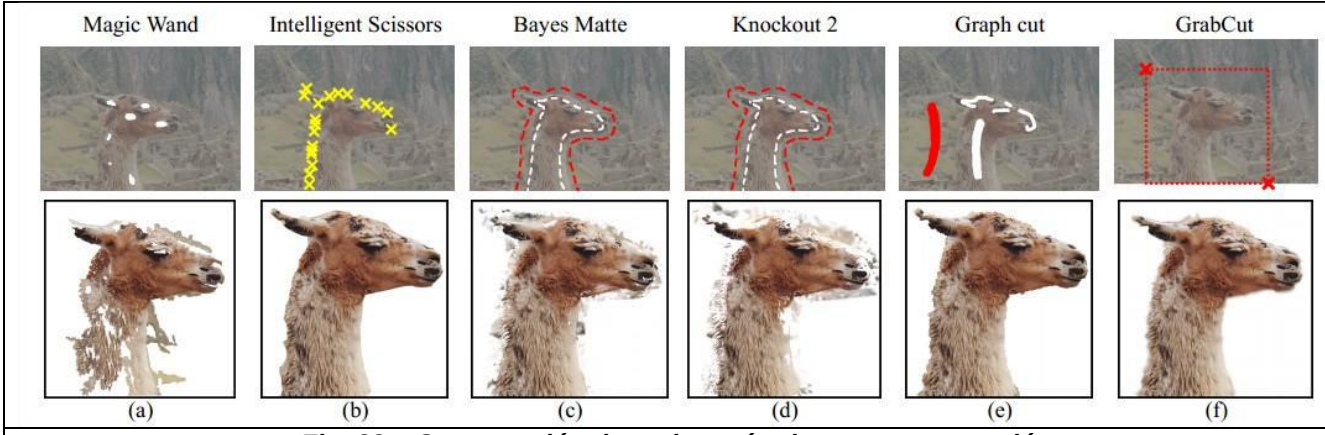


Fig. 23 – Comparación de varios métodos se segmentación

A continuación se expone de una forma más formal la optimización Grabcut para la extracción del objeto en cuestión en el proceso de segmentación [12]:

Dada una Imagen en color *I*, consideramos una matriz $z = (Z_1, \dots, Z_q, \dots, > Z_Q)$ de *Q* píxeles donde $Z_i = (R_i G_i B_i)$, $i \in [1, \dots , Q]$ en el espacio RGB.

Definimos la segmentación como la matriz $\propto = (\propto_1, \dots, \propto_Q), \propto_i \in \{0,1\}$, asignando una etiqueta a cada píxel de la imagen para distinguir si pertenece al fondo o al objeto en cuestión.

Se definen 3 regiones $T_B T_F T_U$, que contienen las etiquetas para el fondo, el objeto y los píxeles inciertos respectivamente. Grabcut no podrá modificar las etiquetas de los dos primeros conjuntos pero si del último. La información de color se introduce mediante GMM (Gaussian Mixture Model). Una covarianza completa de los componentes de U es definida para los píxeles del fondo $(\propto_i = 0)$ y $(\propto_i = 1)$ parametrizado de la siguiente manera:

$$\theta = \left\{\pi(\alpha, u), \mu(\alpha, u), \sum(\alpha, u), \alpha \in \{0,1\}, u = 1..U\right\}$$

Donde π son los pesos, μ las medias y $\sum$ las matrices de covarianza del modelo.

Consideramos también la matriz $u = \{u_1, \dots, u_i, \dots, u_Q\}, u_i \in \{1, \dots U\}, i \in [1, \dots, Q]$ indicando el componente del fondo o del objeto al que pertenece el píxel Z_i. La función de energía para la segmentación E es:

$$E(\propto, u, \theta, z) = U(\alpha, u, \theta, z) + V(\alpha, z)$$

Donde U es el potencial de probabilidad sobre la base de las probabilidades p(·) del GMM,

$$U(\propto, u, \theta, z) = \sum_i -\log p(z_i|\alpha_i, u_i, \theta) - \log \pi(\alpha_i, u_i)$$

I V es una regularización previa suponiendo que las regiones segmentadas deben de ser coherentes en términos de color teniendo en cuenta la vecindad que rodea a cada píxel,

$$V(\propto, z) = \gamma \sum_{\{m,q\} \in C} [\propto_q \neq \propto_m] exp\left(-\beta \|z_m z_q\|^2\right)$$

3.5.3 Implementación

En nuestro caso, en anteriores fases del proyecto se realizó la segmentación binaria mediante GrabCut, con la implementación que viene con las librerías OpenCV [3] y obteniendo unos resultados aceptables [6].

En las Fig. 24 y 24 podemos observar un pequeño ejemplo de los resultados de segmentación binaria obtenidos.

Fig. 24 – Imagen original

Fig. 25 – Imagen segmentada

La Fig. 26 muestra un diagrama de flujo que refleja el bucle de funcionamiento de la implementación utilizada.

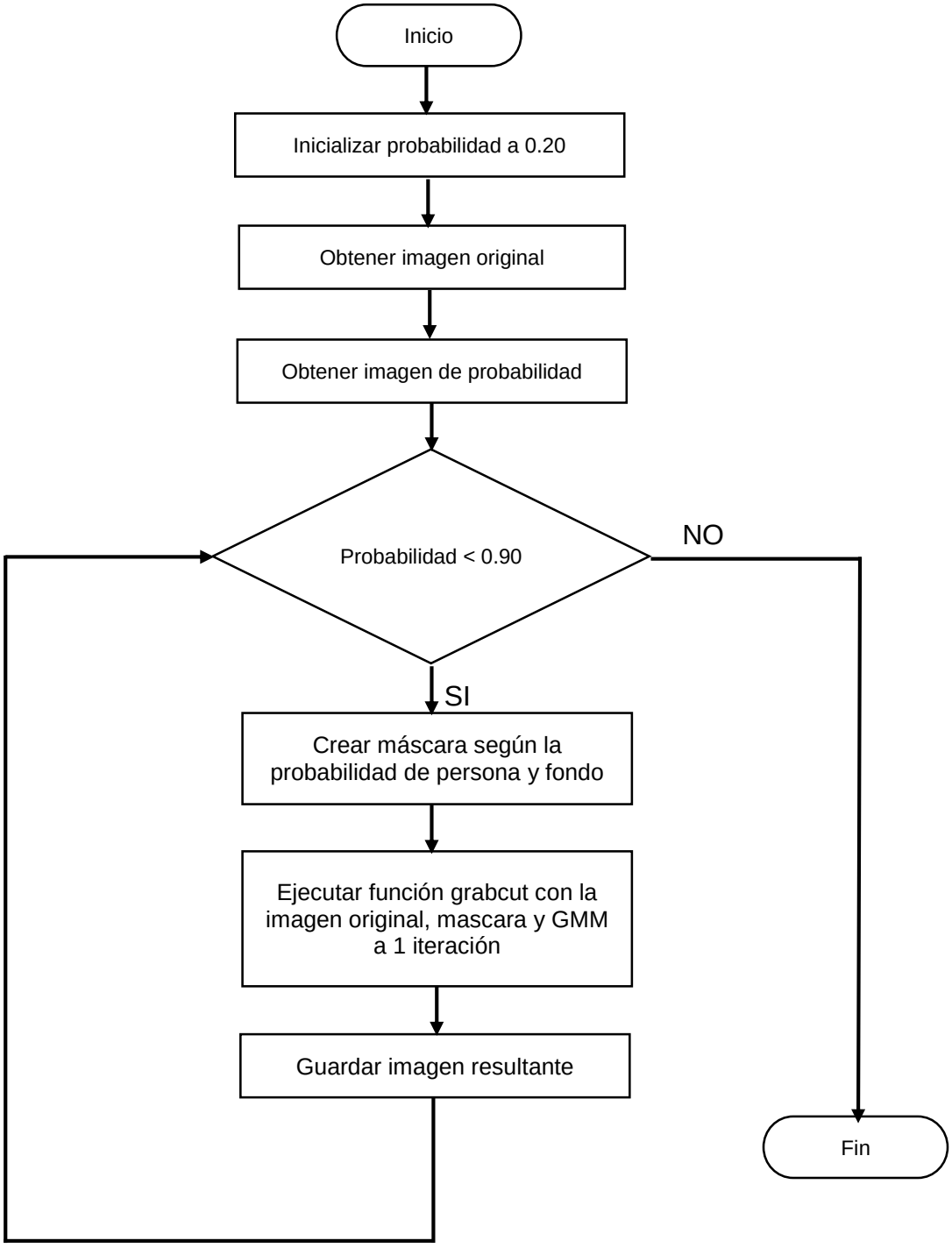


Fig.26 – Diagrama de flujo de la aplicación de GrabCut sobre las imágenes

3.6 Multi-label alpha-beta-swap graph cuts

A continuación se explicará el funcionamiento del algoritmo alpha-beta-swap, utilizado para realizar la segmentación multi-clase. Dicho algoritmo realiza la segmentación etiquetando todos los píxeles de la forma, pero, de tal manera, que permite muchos de estos píxeles cambien sus etiquetas de forma simultánea.

3.6.1 Alpha-beta-swap

Este algoritmo se caracteriza por permitir el intercambio de píxeles de una etiqueta α a otra β y viceversa y se resume de la siguiente manera:

1. Comenzar con una etiqueta arbitraria f.
2. Establecer éxito = 0.
3. Para cada par de etiquetas α y β.
   a. Encontrar $f'' = argminE(f')$ siendo f' el valor de la etiqueta al producir un único intercambio entre *α* y *β*.
   b. Si $E(f'') \leq E(f)$ entonces establecer $f = f''$ y éxito = *1*.
4. Si éxito = *1* volver al paso 2
5. Devolver el resultado final de la etiqueta f.

La Fig. 27 ilustra este cambio de etiquetas, en la primera imagen se observa el etiquetado inicial, en la segunda se puede observar el cambio de una única etiqueta en un paso y finalmente en la tercera imagen se aprecia el cambio de un gran número de etiquetas simultáneamente.

Fig. 27 – Proceso de intercambio de etiquetas de alpha-beta-swap.

3.6.2 Implementación

La implementación del algoritmo se ha realizado mediante el uso de la librería gco-3.0 realizada por Olga Veksler y Andrew Delong [8].

La Fig. 28 muestra el diagrama de flujo de la aplicación que encapsula el algoritmo alpha-beta-swap.

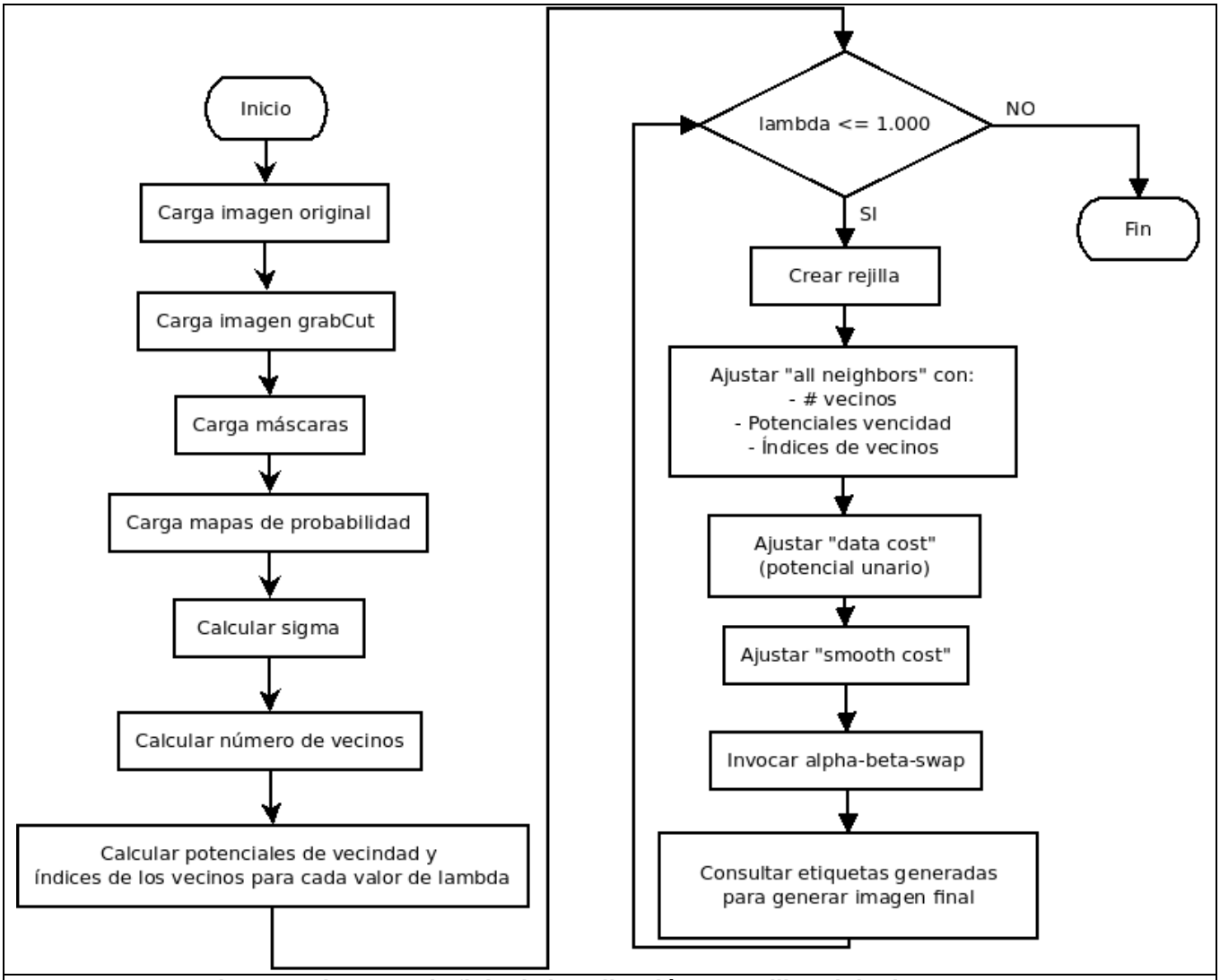


Fig. 28 – Diagrama de flujo de la aplicación que utiliza alpha-beta-swap.

Funciona de la siguiente manera:

1. Se carga la imagen original en escala de grises.

2. Cargar la imagen que proviene del GrabCut con tal de generar su respectiva máscara del Ground truth y realizar una operación AND entre ésta y la imagen resultante del multi-label obteniendo así el Ground truth de la persona segmentada con todas sus extremidades.

3. Cargar todas las máscaras de la base de datos que pertenecen a la imagen original. Esto es para calcular el acierto del algoritmo sobre la imagen original.

4. Cargar los mapas de probabilidad obtenidos de la multi-clasificación.

5. Calcular sigma, valor utilizado en el cálculo de los potenciales de vecindad. Consiste en sumar las distancias entre los vecinos del píxel en cuestión con dicho píxel teniendo en cuenta que no se repita la relación vecino-píxel. Una vez realizada la suma de todos los píxeles, se dividirá entre el par de vecinos no repetidos. Por ejemplo, si se ha contabilizado el par de vecinos (p1, p2) se omitirá (p2, p1), puesto que en temas de distancias sería lo mismo.

6. Calcular el número de vecinos para cada píxel.

7. Se calculan los potenciales de vecindad y los índices de los vecinos. Véase la expresión:

$$\lambda * exp(-\frac{(Wpixel - Wneighbor)^2}{\sigma})$$
$$donde\ \lambda: [1, 10, 100, 1000]$$

   Para cada valor de λ:

8. A partir del número de etiquetas y píxeles, crear la rejilla con estructura de 8 vecinos como máximo por píxel.

9. Ajustar todos los vecinos a partir de: número de vecinos, potenciales de vecindad e índices de vecinos.

10. Calcula el data cost cumpliendo las siguientes condiciones:

    - Para todas las etiquetas que no sean fondo, si el píxel detectado es fondo se asigna un coste de 1000, en cambio, si el píxel detectado pertenece a la persona, se asigna un coste mediante la siguiente fórmula:

    - Decir que los valores de los píxeles de los mapas de probabilidad son potenciales en vez de coste y están en un rango [0, 255]. Uno de los requisitos para que se puedan utilizar satisfactoriamente en la fórmula es que estén en un rango [0, 1], por este motivo se normaliza mediante la división entre el valor máximo del mapa de probabilidad de la extremidad en cuestión. El paso de potencial a coste se realiza mediante la función *log().*

    - En caso de que la etiqueta corresponda al fondo y el píxel detectado sea también fondo, se asigna un coste 0. Si por contra el píxel detectado es persona, el coste asignado será de 1000.

11. Ajustar smooth cost.

12. Llamar a alpha-beta-swap.

13. Se consultan las diferentes etiquetas de la rejilla para dibujar una imagen de salida una representación de dicha etiqueta con un color que la distinga del resto.

14. El resultado obtenido serán 4 imágenes (1 por valor de λ).

Una vez más, podemos consultar los resultados obtenidos en el trabajo realizado por Sr. Daniel Sánchez Abril [7].

4. Resultados

Como ya se ha comentado anteriormente, se ha construido un sistema en diferentes fases en las que se han aplicado distintas metodologías. A continuación se procederá a dar una visión general de los procedimientos seguidos, así como los resultados obtenidos.

Se verá con más detalle la parte en la que está enfocada en este PFC, como es la obtención de los mapas de probabilidad multi-extremidad partiendo de las características obtenidas por HOG y mediante la combinación de clasificadores binarios SVM y los códigos ECOC.

4.1 Visión General

El primer objetivo es el de conseguir segmentar a la persona del fondo de la imagen y para ello se ha realizado un entrenamiento de cascadas *Adaboost* mediante las características *Haar-like*. Con la cascada de clasificadores ya entrenada se pasa a la fase de testeo en la que se recorre la imagen mediante una ventana o *Sliding window* (este método se detallará posteriormente) que enviará a la cascada de clasificadores la regiones por las que pase con el fin de predecir a la clase a la que pertenece. A dicha predicción le será aplicada una matriz ECOC que ajustará el resultado para obtener la extremidad definitiva.

Mediante el cálculo de estas pseudo-probabilidades, este proceso dará como resultado una imagen de intensidad en la que aparecerán con varias intensidades las distintas probabilidades de que los píxeles de la misma formen parte de la persona o no. A partir de esta imagen, se realizará la segmentación binaria mediante Grabcut, y obtendremos las máscaras necesarias para continuar con la siguiente fase.

Las figuras 29(a) y 29(b) muestran ejemplos de los resultados obtenidos por el proceso.

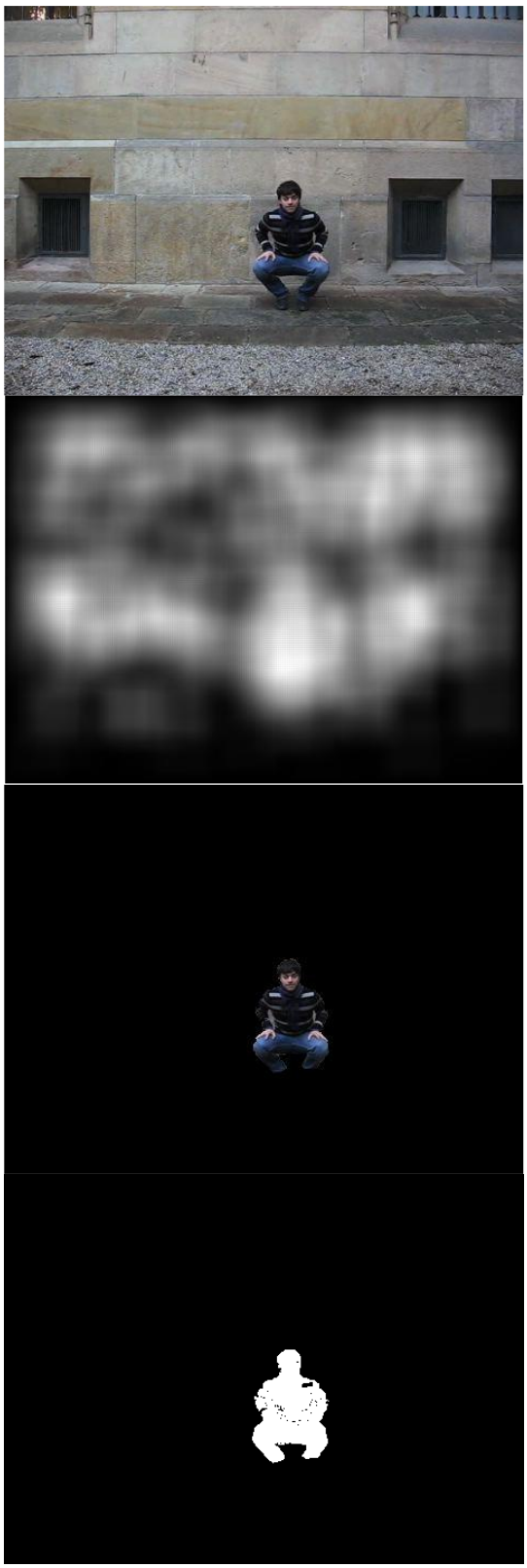


Fig. 29 (a) – Ejemplo resultado Grabcut

Fig. 29 (b) – Ejemplo resultado Grabcut

El siguiente paso es la multi-clasificación de las extremidades del cuerpo de las personas que aparecen en las imágenes que forman nuestra base de datos. Como ya se ha comentado, este proceso se verá con más detalle en el siguiente punto. Se trata de detectar las extremidades mediante la combinación del algoritmo clasificador binario SVM y los códigos ECOC que harán posible la multi-clasificación. Todo ello a partir de los modelos generados en el entrenamiento de SVM mediante las características obtenidas por HOG.

Este proceso dará como resultados 6 imágenes de probabilidad, una por extremidad configurada, que se utilizarán posteriormente en la multi-segmentación, punto en el que serán segmentadas cada una de las extremidades mediante *alpha-beta-swap graphcuts*.

Suponiendo que se ha realizado el previo entrenamiento tanto de las cascadas *Adaboost* como de SVM, la figura 30 muestra un diagrama con el proceso completo.

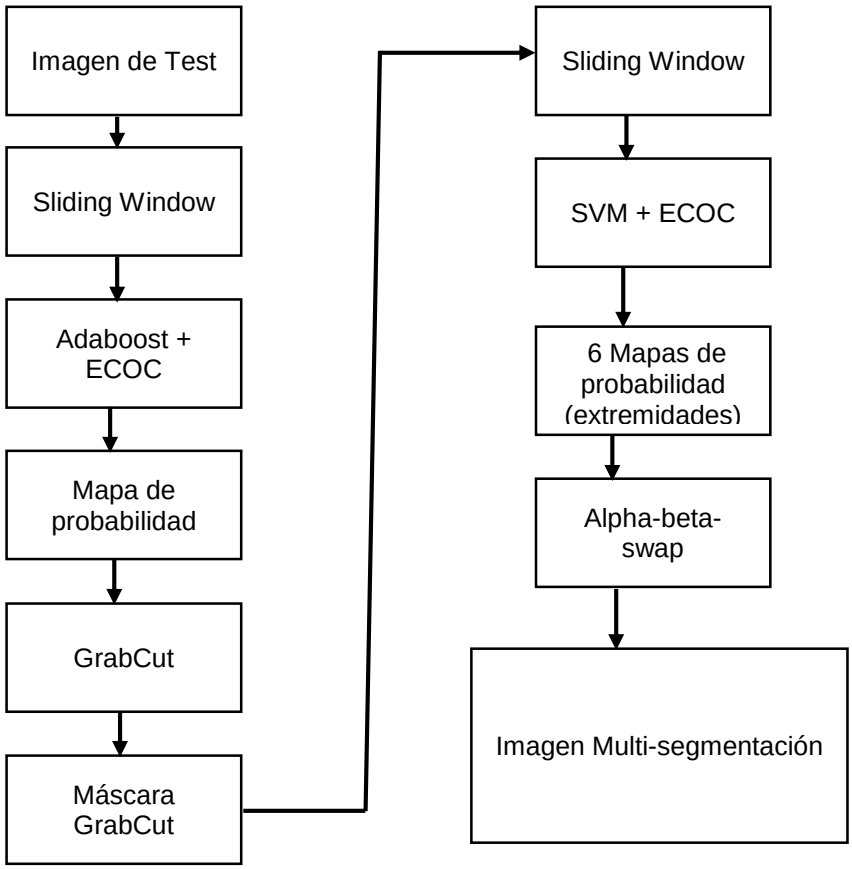


Fig. 30 Diagrama que muestra el proceso completo

4.2 Multi-clasificación con SVM + ECOC

En este punto se detalla la fase del sistema sobre la que está enfocada este PFC, la multi-clasificación mediante SVM, ECOC y las características HOG. Se mostrarán, también, los resultados obtenidos.

4.2.1 Introducción

Esta fase se ha llevado a cabo mediante el lanzamiento de 9 experimentos que contienen los 9 actores que forman la base de datos, con la particularidad de que en cada experimento se utilizan 8 actores para realizar el entrenamiento y el restante para el test. Finalizando con un total de 9 conjuntos de resultados, uno por cada actor.

4.2.2 Características HOG

La obtención de las características HOG, es un paso esencial para obtener tanto los ejemplos de entrenamiento como de test de los clasificadores. Para ello se ha utilizado la implementación de HOG que nos ofrece OpenCV. En la sección 3.3 se ha comentado que se requería de una serie de parámetros que debían estar prefijados y también de la necesidad tanto de redimensionar como de rotar según su orientación predominante las imágenes que iban a ser descritas para mantener un tamaño único de descriptor y obtener buena descripción independientemente de la orientación. Los parámetros utilizados para el cálculo de las características HOG son:

- **Win Size**: 32x32 Especifica el tamaño de la ventana a analizar.
- **Block Size**: 16x16 Especifica el tamaño del bloque de celdas.
- **Block Stride**: 8x8 Especifica las divisiones dentro de un bloque.
- **Cell Size**: 8x8 Especifica las dimensiones de cada celda.
- **Num bins**: 8 El número de bins del histograma por celda.

El procedimiento para obtener las imágenes de entrenamiento consiste en recorrer el directorio con las máscaras binarias que forman el Ground truth. Para cada máscara encontrada se identifica la imagen RGB original que le corresponde mediante el nombre.

Puesto que dichas máscaras están formadas por unos y ceros y tienen el mismo tamaño que la imagen original, se trata básicamente de localizar los puntos máximos y mínimos que forman la zona etiquetada de la máscara y utilizar esa misma ubicación en la imagen original.

A continuación se realiza la rotación y posterior recorte de la región de interés (la extremidad en cuestión) y se le pasa a HOG para obtener el vector de características correspondiente que será almacenado en un fichero. El resultado final de todo el proceso serán 6 ficheros (uno por cada extremidad) que contendrán los vectores de características en el siguiente formato que viene dado por la implementación de SVM de la librería LibSVM [14]:

idClase index[0] : value … index[n] #idActor

Donde:

idClase es la identificación de la clase a la que pertenece la extremidad descrita

index[] es el índice del elemento HOG que forma el vector

value es el valor numérico de la característica dada por HOG.

#idActor es el identificador del Actor al que pertenece la extremidad descrita.

Gracias a idClase e idActor, se puede acceder a características de una extremidad en concreto de cualquiera de los actores.

Las figuras 31(a) y 31(b) muestran un diagrama que ilustra el proceso comentado anteriormente y los ficheros resultantes obtenidos.

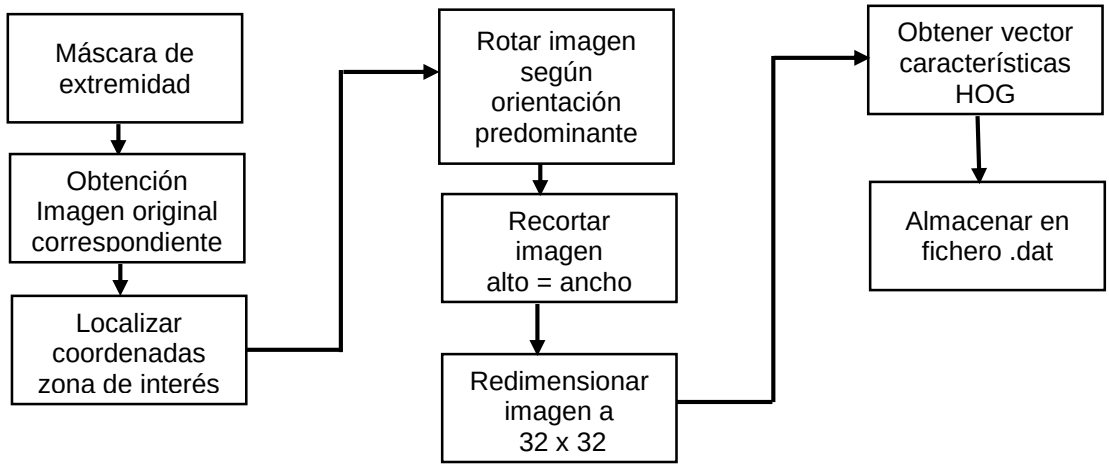


Fig. 31(a) – Diagrama obtención características HOG

Fichero	Extremidad
HOG_1.dat	Torso
HOG_2.dat	Cabeza
HOG_3.dat	Brazo
HOG_4.dat	Antebrazo
HOG_5.dat	Muslo
HOG_6.dat	Pierna

Fig. 31(b) – Ficheros HOG finales

4.2.3 Entrenamiento SVM

Una vez generados los ficheros, se procede al entrenamiento de los clasificadores SVM. Para ello montaremos una serie de ficheros, derivados de los 6 anteriores, en los que constaran además de las combinaciones de clases necesarias para montar el sistema ECOC explicado en el punto 3.4.4, las agrupaciones de actores comentadas anteriormente con tal de configurar la entrada para cada uno de los 9 experimentos que serán lanzados.

Este proceso ha sido llevado a cabo una pequeña utilidad de java que se ha desarrollado para este propósito y que se ha beneficiado de las id de los ficheros HOG, especificadas en el punto anterior. Obtendremos como resultado un total de 45 ficheros (5 grupos de extremidades, una por clasificador, multiplicados por los 9 experimentos).

Hay que destacar, que por motivos de tiempo, se ha decidido lanzar dichos experimentos con un 10% del total de la muestra, es decir, para generar los ficheros de entrenamiento de cada clasificador se ha utilizado tan solo un 10% de cada una de las clases descritas en los ficheros HOG iniciales. Posteriormente, se realizará una breve comparativa de los tiempos de ejecución de los experimentos. . Las características *Haar* consisten en varias regiones blancas y negras en forma de rectángulo que se aplican a un conjunto de píxeles y en las cuales se definirá si se suma o se resta la luminancia (valor de los píxeles en escala de grises) de los píxeles afectados en cada una de las partes. La diferencia entre la suma de

luminancia de la parte negra y la suma de la luminancia en la parte blanca da lugar al valor de la característica.

La figura 32 muestra una representación de los ficheros generados en este paso.

EXP 1	EXP 2	■ ■ ■	EXP 8	EXP 9
EXP1_H1.dat	EXP2_H1.dat		EXP8_H1.dat	EXP9_H1.dat
EXP1_H2.dat	EXP2_H2.dat		EXP8_H2.dat	EXP9_H2.dat
EXP1_H3.dat	EXP2_H3.dat		EXP8_H3.dat	EXP9_H3.dat
EXP1_H4.dat	EXP2_H4.dat		EXP8_H4.dat	EXP9_H4.dat
EXP1_H5.dat	EXP2_H5.dat		EXP8_H5.dat	EXP9_H5.dat

Fig.32 – Tabla con los ficheros de entrenamiento generados para cada expermineto

Para realizar el entrenamiento, se ha utilizado la utilidad que viene con LibSVM llamada svm_train, que se encarga de generar el modelo de entrenamiento partir de los ficheros de entrada:

svm_train [options] training_set_file [model file]

Donde:

Options se utilizan las opciones que vienen por defecto tipo de SVM multi-class y el tipo de kernel RBF (Radial Basis Function).

Training_set_file la ubicación y nombre del fichero que contiene el set de entrenamiento generado en el paso anterior

Model_file ubicación de destino del fichero de modelo generado

Tras ejecutar 45 veces dicha utilidad, obtenemos los 45 ficheros de modelos listos para ser utilizados en la fase de clasificación. La figura 33 muestra una tabla con los ficheros generados.

EXP 1	EXP 2	■ ■ ■	EXP 8	EXP 9
EXP1_H1.model	EXP2_H1.model		EXP2_H1.model	EXP2_H1.model
EXP1_H2.model	EXP2_H2.model		EXP2_H2.model	EXP2_H2.model
EXP1_H3.model	EXP2_H3.model		EXP2_H3.model	EXP2_H3.model
EXP1_H4.model	EXP2_H4.model		EXP2_H4.model	EXP2_H4.model
EXP1_H5.model	EXP2_H5.model		EXP2_H5.model	EXP2_H5.model

Fig.33 – Tabla con los ficheros de modelos generados para cada experimento

Antes de comenzar con la clasificación propiamente dicha, y como se había comentado con anterioridad, en el apartado dedicado a ECOC, es importante calcular los porcentajes que indican lo bien que ha entrenado sus ejemplos cada uno de los clasificadores, con tal de poder utilizar estas cifras posteriormente.

Para ello, haremos uso de la utilidad de predicción que viene también con LibSVM:

svm_predict [options] test_file model_file output_file

Donde:

options son opciones adicionales que no utilizaremos.

test_file es el fichero de test, en este caso serán los mismos utilizados para el entrenamiento de los modelos.

model_file es el fichero de modelo generado con svm_train anteriormente.

output_file es el fichero de salida que muestra las predicciones realizadas para cada uno de los ejemplos de entrada.

La misma utilidad, muestra por pantalla el valor que indica cuan bien ha aprendido sus ejemplos el clasificador, no obstante, se puede obtener este dato a partir del fichero de salida, ya que muestra las etiquetas predichas por SVM, y contamos también con las etiquetas originales de los datos de entrada, de modo que la simple división del total de datos correctos entre el total de datos de entrada nos daría el mismo porcentaje.

La figura 34 muestra los valores de los pesos obtenidos para cada uno de los clasificadores entrenados:

EXP 1	EXP 2	EXP 3	EXP 4	EXP 5	EXP 6	EXP 7	EXP 8	EXP 9
0.880612	0.880932	0.884001	0.878521	0.880802	0.881845	0.877632	0.887489	0.878578
0.964114	0.956074	0.975558	0.964706	0.966827	0.969648	0.968368	0.968951	0.969844
0.786686	0.766855	0.780051	0.778113	0.777456	0.771656	0.775973	0.777144	0.781783
0.749912	0.742105	0.751042	0.768515	0.735757	0.750885	0.757184	0.752659	0.749476
0.817991	0.791004	0.796505	0.801356	0.802329	0.82657	0.803753	0.807307	0.796539

Fig. 34 – Valores de los pesos para cada uno de los clasificadores entrenados

4.2.4 Clasificación SVM + ECOC

El siguiente paso consiste en generar los mapas de probabilidad para cada una de las 6 extremidades escogidas para la multi-segmentación. Se ha integrado todo el procedimiento de generación de los mapas de probabilidad en un único proceso que realiza todos los pasos necesarios para llegar al objetivo. Para realizar la predicción de los clasificadores entrenados se ha optado por añadir el código fuente de predict de la librería libSVM comentada anteriormente.

Todo ello ha sido realizado en C/C++ debido a las grandes ventajas que aporta en cuestiones de eficiencia. Se han preparado 9 variantes del programa, una por experimento, listos para ser ejecutados de forma independiente.

Para cada experimento se requiere un directorio con las máscaras Grabcut generadas en la fase anterior, un directorio con los modelos entrenados y un directorio más con las imágenes RGB originales que serán procesadas para obtener sus mapas de probabilidad. Para cada experimento, se trata de las imágenes del actor que no ha sido incluido en el set de entrenamiento.

Este procedimiento contiene también la técnica del Sliding Window, que ha sido nombrada anteriormente y que se detalla a continuación:

Sliding Window, consiste básicamente en recorrer una imagen con una ventana de tamaño variable y con un cierto desplazamiento entre posiciones de dicha ventana.
De este modo, conseguimos realizar la detección de objetos independientemente del tamaño que tengan puesto que se realiza un barrido completo de la imagen a distintos tamaños. La figura 35 muestra el funcionamiento de dicho sistema.

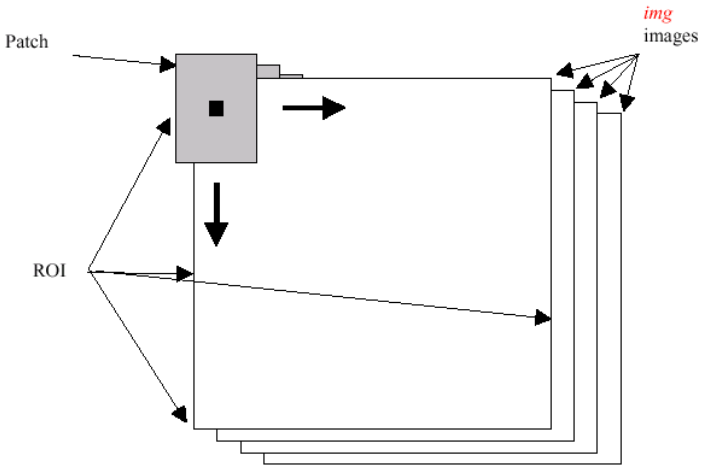


Figura 35: Funcionamiento de sliding window (ventana deslizante)

Cada una de las subregiones dadas por dicha ventana, son rotadas, recortadas y enviadas a HOG para que realice la descripción necesaria para enviar a los clasificadores que obtendrán la predicción de dicha subregión.

La configuración ECOC tal y como se ha descrito en la sección 3.4.4 es aplicada a estas predicciones obteniendo la etiqueta final para cada uno de los píxeles que forma la subregión en cuestión. De este modo, se realizará una votación sobre los píxeles de la zona en la imagen de probabilidad correspondiente a la extremidad predicha, obteniendo al final del proceso los 6 mapas de probabilidad con las votaciones realizadas.

Como puede observarse, para cada región obtenida por la Sliding window se realiza una gran cantidad de procesos, lo que obliga a tomar una serie de medidas con tal de reducir los tiempos de ejecución.

La primera medida tomada, como se ha comentado anteriormente, es la reducción de la muestra inicial de ejemplos de entrenamiento a un 10%. Otra de las medidas ha sido la de utilizar una distancia de desplazamiento entre ventanas de 2 píxeles.

El hecho de contar con la segmentación binaria realizada por grabcut en la fase anterior, nos permite descartar de nuestra detección una gran parte de la imagen en la que no se encontrarán extremidades. Esta reducción de la superficie a ser barrida por parte de la Sliding window reduce considerablemente el tiempo de ejecución.

Finalmente, se ha decido acotar el tamaño de ventana a los tamaños de las extremidades que van a ser detectadas, con la intención de obtener los resultados más significativos posibles, descartando aquellos tamaños de ventana que no tienen un número elevado de extremidades para ser detectadas.

La figura 36 muestra una gráfica con los tamaños de las extremidades que van a ser testeadas para identificar los que podemos encontrar con más frecuencia. La figura 37

muestra una pequeña comparativa con los tiempos de ejecución para cada una de las distintas configuraciones de parámetros que han sido utilizadas.

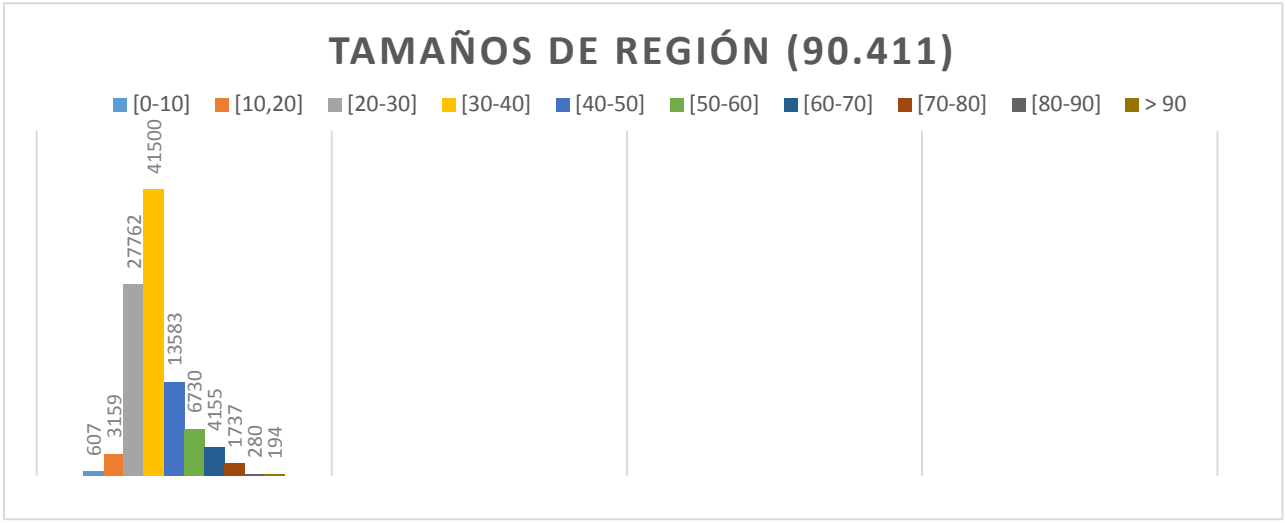


Figura 36: Gráfica con los tamaños de extremidad más habituales en las imágenes de test utilizadas

Teniendo en cuenta estos números, el rango de ventana escogido finalmente ha sido el de 20 - 70 píxeles.

	Toda la muestra	10% de la muestra	10% de la muestra
Tamaño ventana	30-60	30-60	20-70
Desplazamiento	1 píxel	1 píxel	2 píxeles
Factor crecimiento	1.1	1.1	1.1
Tiempo proceso	>6 horas	Aprox. 1 hora	Aprox. 16 min.

Fig. 37 – Tabla comparativa de los tiempos de ejecución de los procesos

Como se puede observar, las medidas obtenidas para la reducción de tiempos lo hacen de forma considerable. Cabe destacar que los tiempos varían mucho en función de la imagen en cuestión, es decir, será muy diferente una imagen en la que aparezcan dos personas que una. Los tiempos obtenidos en la tabla anterior hacen referencia a imágenes “simples”, en las que aparece una sola persona. Los tiempos aumentan considerablemente en imágenes “complejas”.

Los pasos realizados por todo el proceso comentado anteriormente son:

1. Carga en memoria de los modelos del experimento en cuestión

2. Carga de la imagen que va a ser procesada y su máscara grabcut asociada

3. Localización y recorte de la zona de la imagen en la que está ubicada la persona objeto de detección de la imagen de entrada.

4. Para cada región obtenida por la Sliding window

    - Rotación y recorte de la subregión.

    - Redimensión de del tamaño de la subregión a 32 x 32.

- Obtener la descripción HOG de la subregión

- Obtener el vector de predicciones por parte de SVM a partir de la descripción HOG anterior

- Calcular la predicción final por parte de ECOC.

- Realizar la votación de los píxeles de la subregión sobre la imagen de probabilidad correspondiente a la predicción realizada

5. Generación de las 6 imágenes de probabilidad calculadas.

4.2.5 Resultados cualitativos

La figura 38 muestra algunos ejemplos de las imágenes resultantes del proceso anterior, han sido normalizadas y se les ha aplicado la máscara de grabcut para facilitar la comprobación de las probabilidades calculadas.

Como se puede observar, se han obtenido unos resultados aceptables, teniendo en cuenta el sacrificio realizado con la reducción de los datos de entrenamiento en hasta un 10% y la configuración de parámetros dada.

Las extremidades con mejores resultados han sido la cabeza, los muslos y las piernas mientras que los resultados de brazos y torso no son tan claros.

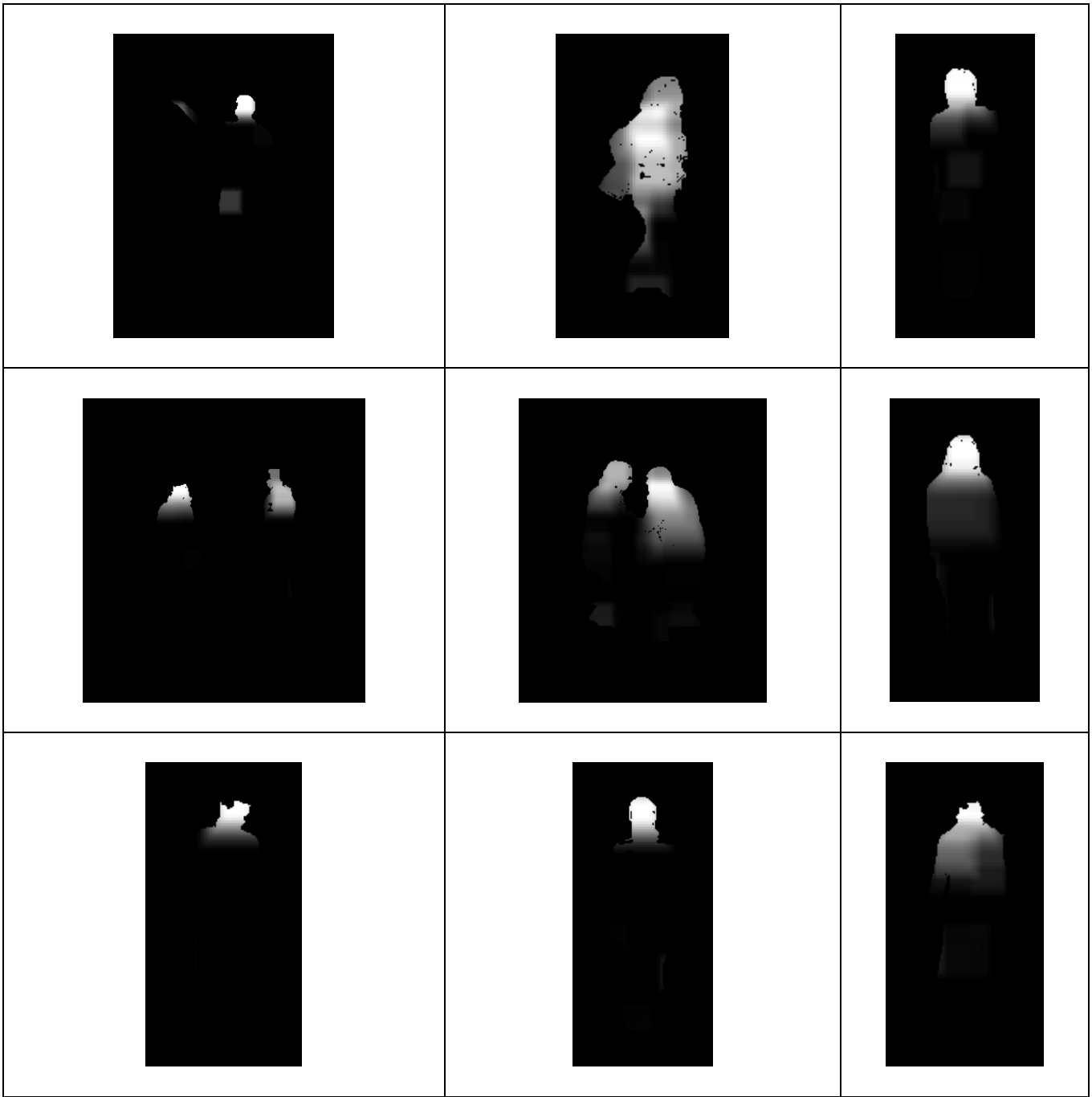


Fig. 38(a) – Resultados obtenidos tras el proceso de multi-clasificación (Cabeza)

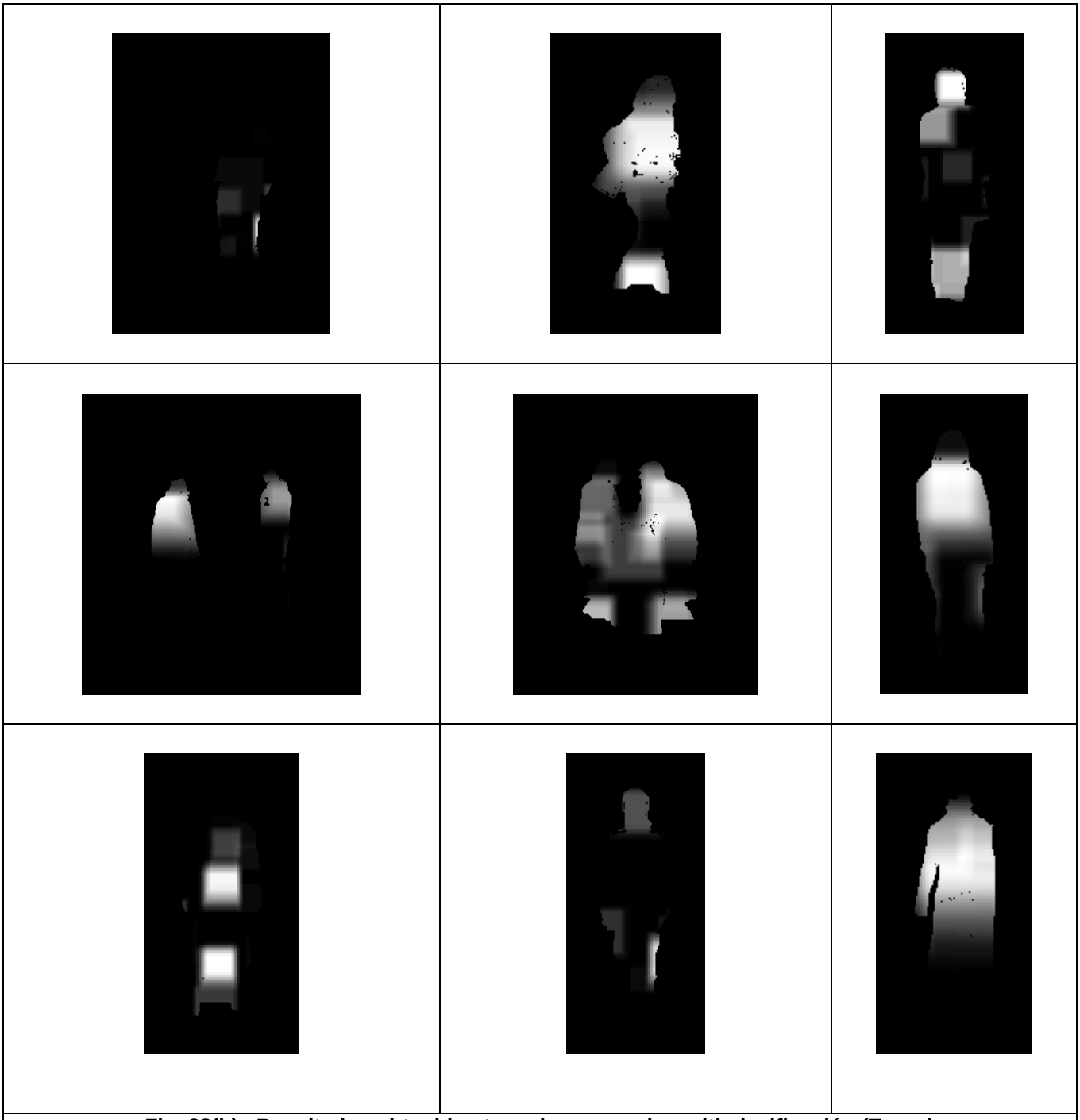


Fig. 38(b) –Resultados obtenidos tras el proceso de multi-clasificación (Torso)

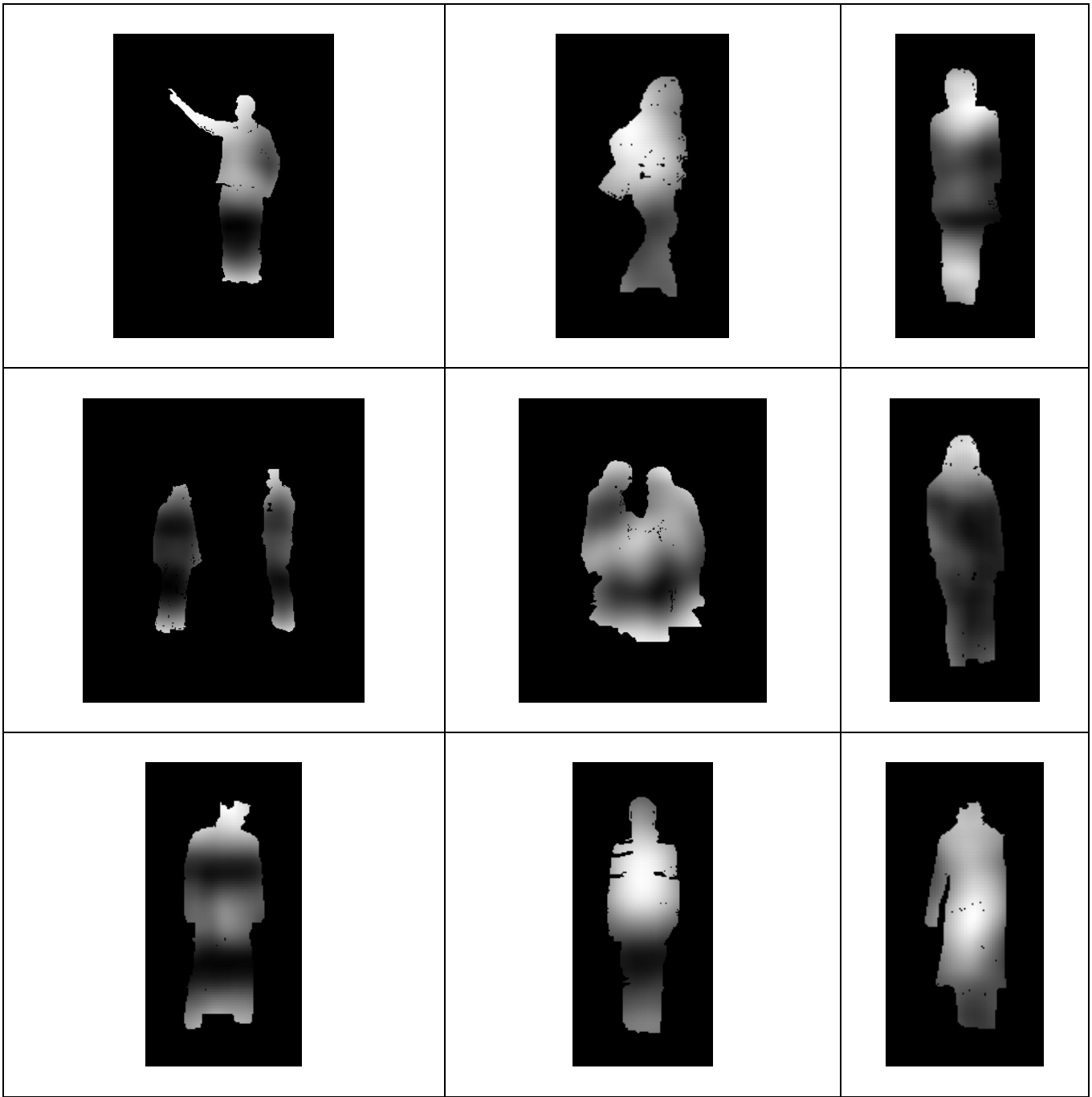


Fig. 38(c) – Resultados obtenidos tras el proceso de multi-clasificación (Antebrazo)

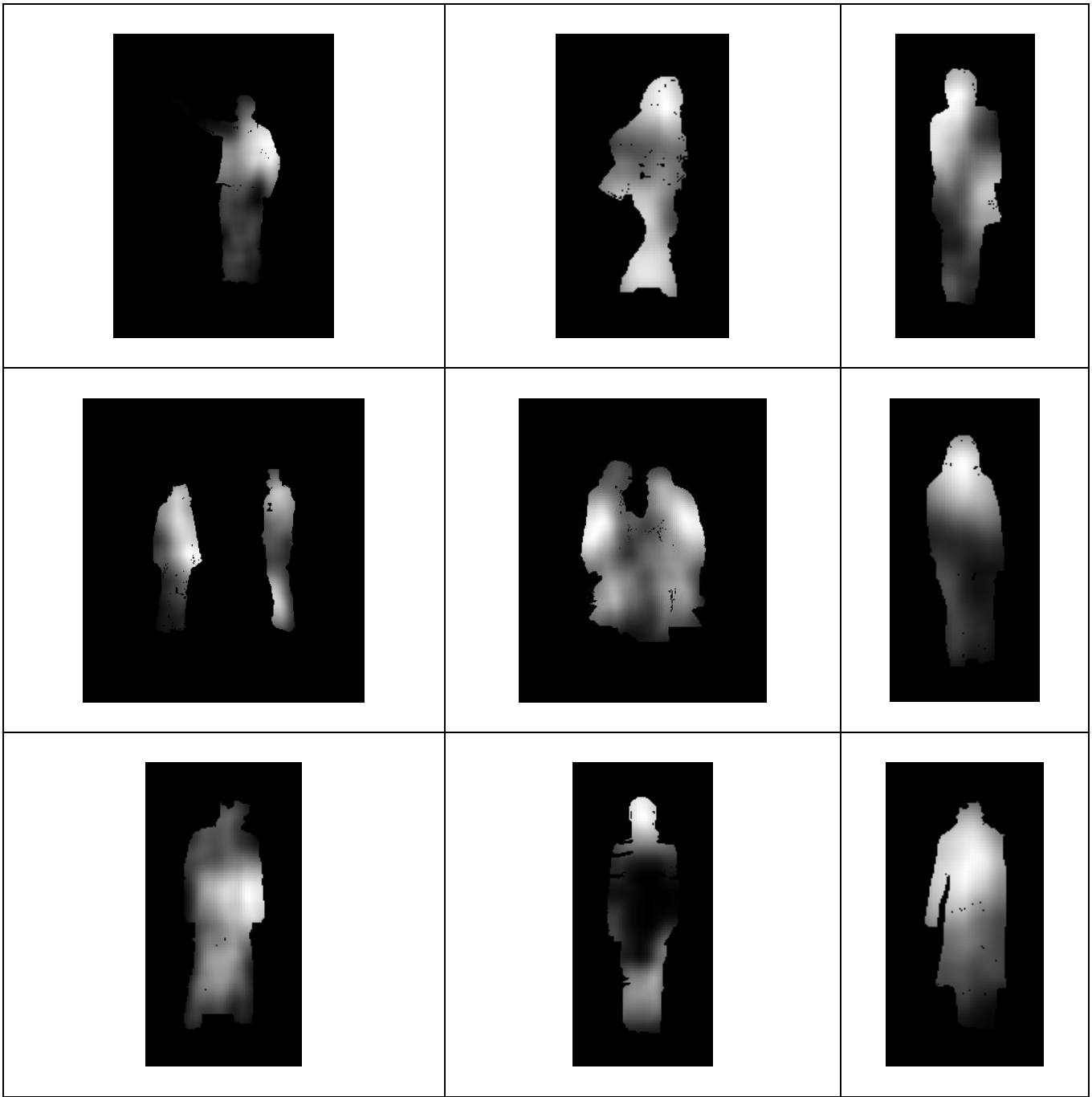


Fig. 38(d) – Resultados obtenidos tras el proceso de multi-clasificación (Brazo)

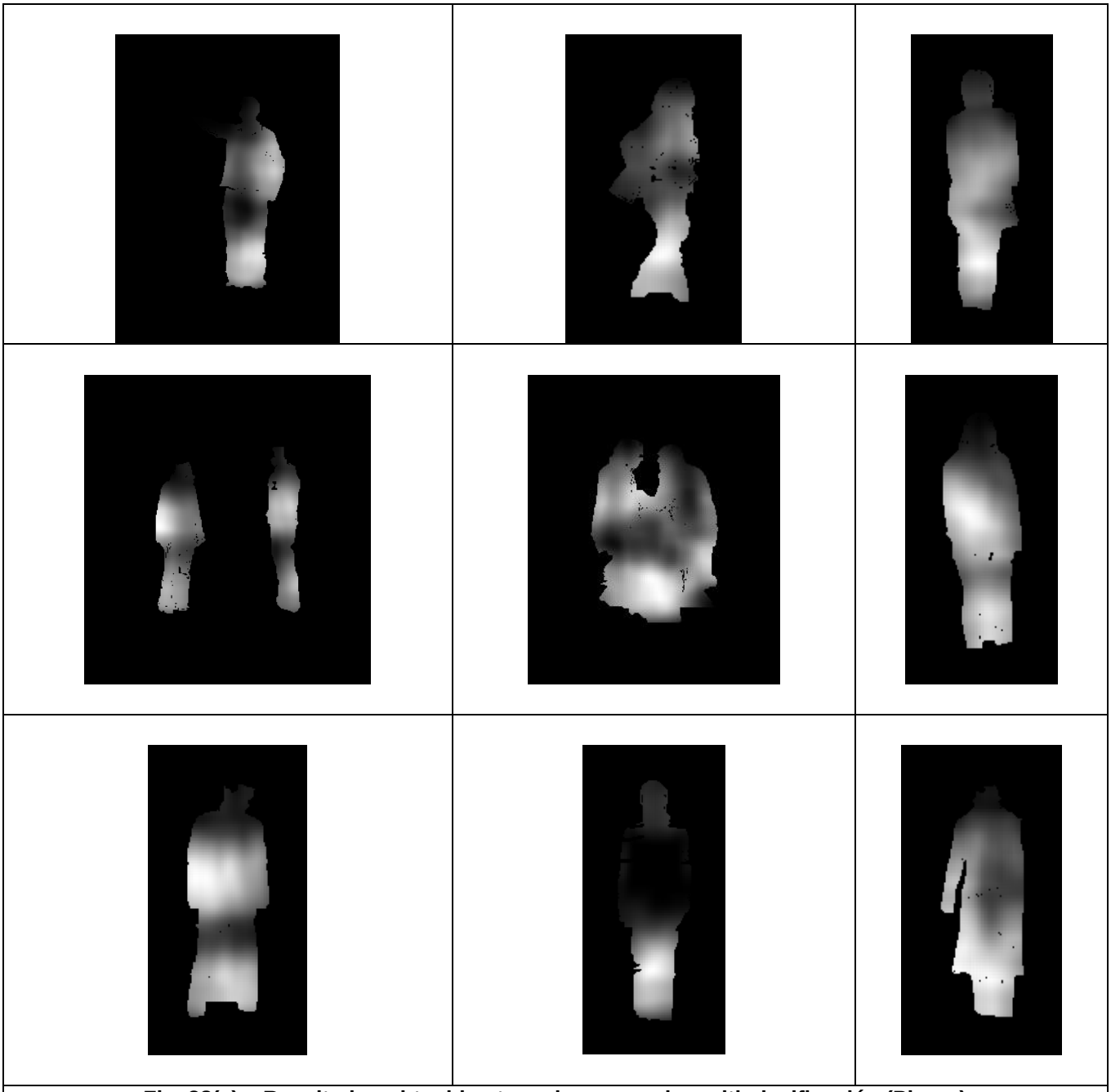


Fig. 38(e) – Resultados obtenidos tras el proceso de multi-clasificación (Pierna)

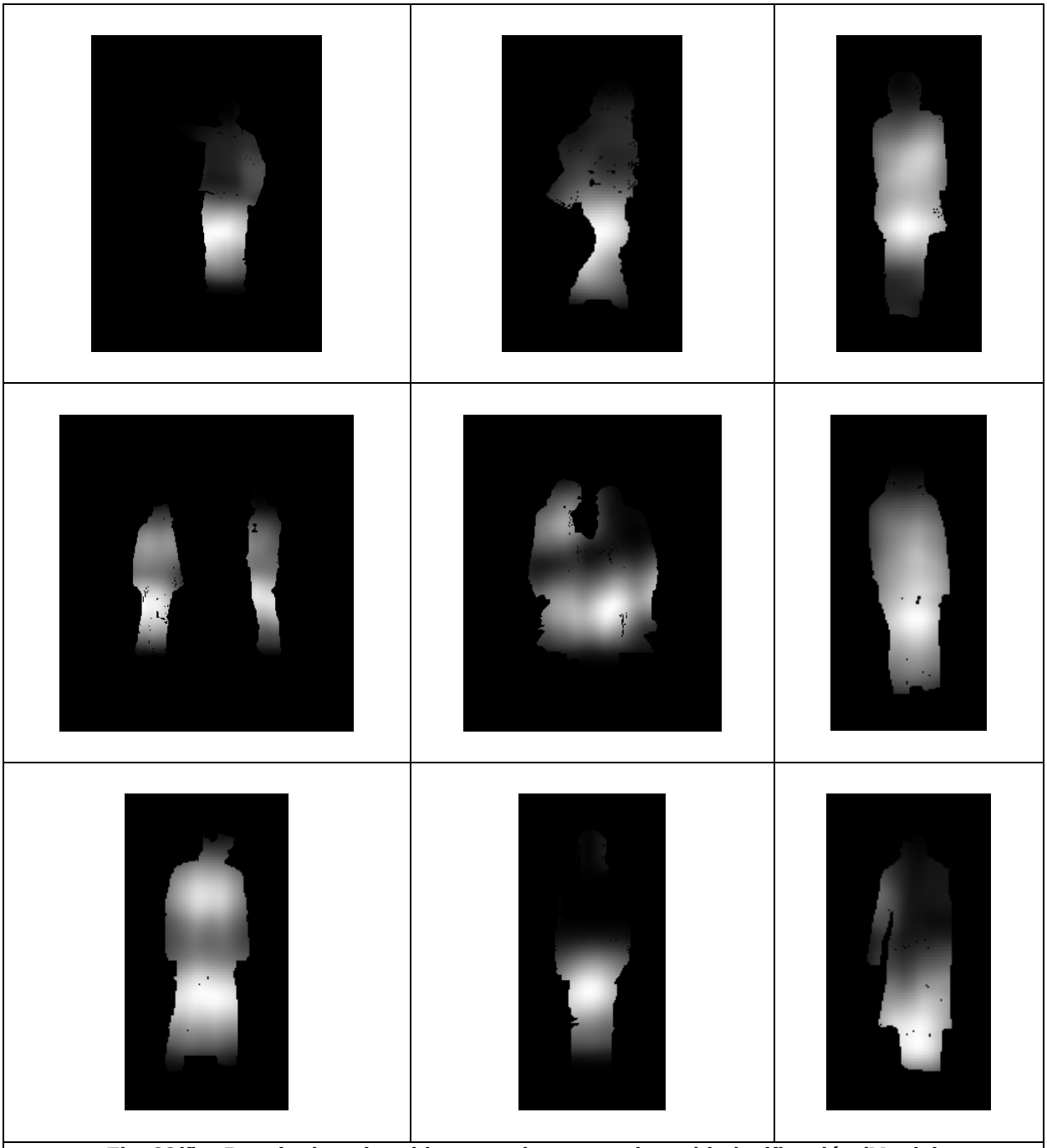


Fig. 38(f) – Resultados obtenidos tras el proceso de multi-clasificación (Muslo)

4.2.6 Resultados cuantitativos

Como se ha comentado y se ha podido observar en las figuras anteriores, cabeza, piernas y muslos son las extremidades que han dado unos resultados más claros en los mapas de probabilidad obtenidos. Con el propósito de corroborar estos datos se han calculado una serie de matrices de confusión. Dichas matrices muestran el porcentaje de acierto de cada extremidad independientemente. Se trata de un modo fácil de analizar que partes se han aprendido de un modo más satisfactorio.

La primera de las matrices muestra el porcentaje de acierto contra los datos que han sido utilizados para realizar el entrenamiento del experimento en cuestión. La segunda son los porcentajes contra los datos de test del mismo experimento. Para el experimento 1, se muestran un total de 4 matrices. Esto es debido a que se ha querido demostrar, de un modo más visual, la merma de los buenos resultados con la reducción de hasta el 90% de los ejemplos utilizados para el entrenamiento. Para el resto de experimentos, se muestran, únicamente, las matrices que pertenecen a los resultados obtenidos tras la reducción de la muestra.

Experimento 1

Total de la muestra

Clase (training)	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
C1 (torso)	62%					
C2 (cabeza)		91%				
C3 (brazos)			61%			
C4 (antebrazos)				58%		
C5 (muslos)					79%	
C6 (piernas)						66%

Clase (test)	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
C1 (torso)	60%					
C2 (cabeza)		95%				
C3 (brazos)			48%			
C4 (antebrazos)				65%		
C5 (muslos)					35%	
C6 (piernas)						41%

10% de la muestra

Clase (training)	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
C1 (torso)	15%					
C2 (cabeza)		89%				
C3 (brazos)			44%			
C4 (antebrazos)				53%		
C5 (muslos)					81%	
C6 (piernas)						64%

Clase (testing)	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
C1 (torso)	25%					
C2 (cabeza)		92%				
C3 (brazos)			55%			
C4 (antebrazos)				44%		
C5 (muslos)					50%	
C6 (piernas)						36%

Experimento 2

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	12%					
C2 (cabeza)		89%				
C3 (brazos)			49%			
C4 (antebrazos)				45%		
C5 (muslos)					76%	
C6 (piernas)						56%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	0%					
C2 (cabeza)		88%				
C3 (brazos)			27%			
C4 (antebrazos)				77%		
C5 (muslos)					84%	
C6 (piernas)						64%

Experimento 3

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	17%					
C2 (cabeza)		90%				
C3 (brazos)			48%			
C4 (antebrazos)				49%		
C5 (muslos)					79%	
C6 (piernas)						59%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	5%					
C2 (cabeza)		80%				
C3 (brazos)			22%			
C4 (antebrazos)				44%		
C5 (muslos)					90%	
C6 (piernas)						58%

Experimento 4

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	12%					
C2 (cabeza)		87%				
C3 (brazos)			51%			
C4 (antebrazos)				50%		
C5 (muslos)					79%	
C6 (piernas)						61%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	4%					
C2 (cabeza)		86%				
C3 (brazos)			17%			
C4 (antebrazos)				54%		
C5 (muslos)					84%	
C6 (piernas)						61%

Experimento 5

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	15%					
C2 (cabeza)		89%				
C3 (brazos)			44%			
C4 (antebrazos)				53%		
C5 (muslos)					78%	
C6 (piernas)						61%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	5%					
C2 (cabeza)		94%				
C3 (brazos)			45%			
C4 (antebrazos)				43%		
C5 (muslos)					60%	
C6 (piernas)						67%

Experimento 6

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	15%					
C2 (cabeza)		89%				
C3 (brazos)			44%			
C4 (antebrazos)				51%		
C5 (muslos)					79%	
C6 (piernas)						65%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	4%					
C2 (cabeza)		82%				
C3 (brazos)			43%			
C4 (antebrazos)				34%		
C5 (muslos)					87%	
C6 (piernas)						18%

Experimento 7

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	12%					
C2 (cabeza)		88%				
C3 (brazos)			44%			
C4 (antebrazos)				53%		
C5 (muslos)					77%	
C6 (piernas)						63%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	18%					
C2 (cabeza)		95%				
C3 (brazos)			40%			
C4 (antebrazos)				44%		
C5 (muslos)					73%	
C6 (piernas)						57%

Experimento 8

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	18%					
C2 (cabeza)		89%				
C3 (brazos)			48%			
C4 (antebrazos)				51%		
C5 (muslos)					80%	
C6 (piernas)						60%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	9%					
C2 (cabeza)		77%				
C3 (brazos)			42%			
C4 (antebrazos)				54%		
C5 (muslos)					79%	
C6 (piernas)						52%

Experimento 9

Clase (training)	C1	C2	C3	C4	C5	C6
C1 (torso)	11%					
C2 (cabeza)		90%				
C3 (brazos)			46%			
C4 (antebrazos)				52%		
C5 (muslos)					79%	
C6 (piernas)						60%

Clase (testing)	C1	C2	C3	C4	C5	C6
C1 (torso)	25%					
C2 (cabeza)		80%				
C3 (brazos)			54%			
C4 (antebrazos)				40%		
C5 (muslos)					70%	
C6 (piernas)						73%

Como se ha podido observar, la claridad en los resultados reflejada en las imágenes del punto anterior coincide con los datos numéricos dados por las matrices de confusión, donde se ve una clara diferencia de porcentajes en las 3 extremidades comentadas anteriormente con respecto al resto. Cabe destacar, una vez más, que la reducción de los datos de entrenamiento ha rebajado de forma considerable la calidad de los resultados obtenidos, especialmente en el caso del torso, que según los datos, ha sido la parte del cuerpo más afectada por las medidas adoptadas.

4.3 Multi-segmentación

Finalmente, queda por ver la fase de multi-segmentación, donde a través del algoritmo alpha-beta-swap graphcuts, sobre el que se hablado con anterioridad, se quieren unir las imágenes de probabilidad obtenidas en el paso anterior a partir de SVM y ECOC en una única imagen que muestre cada una de las extremidades en un color distinto. Para una explicación mucho más detallada del proceso realizado durante la fase de multi-segmentación, consultar [7].

La figura 39 muestra algunos resultados de la multi-segmentación sobre un conjunto de imágenes de test de la base de datos.

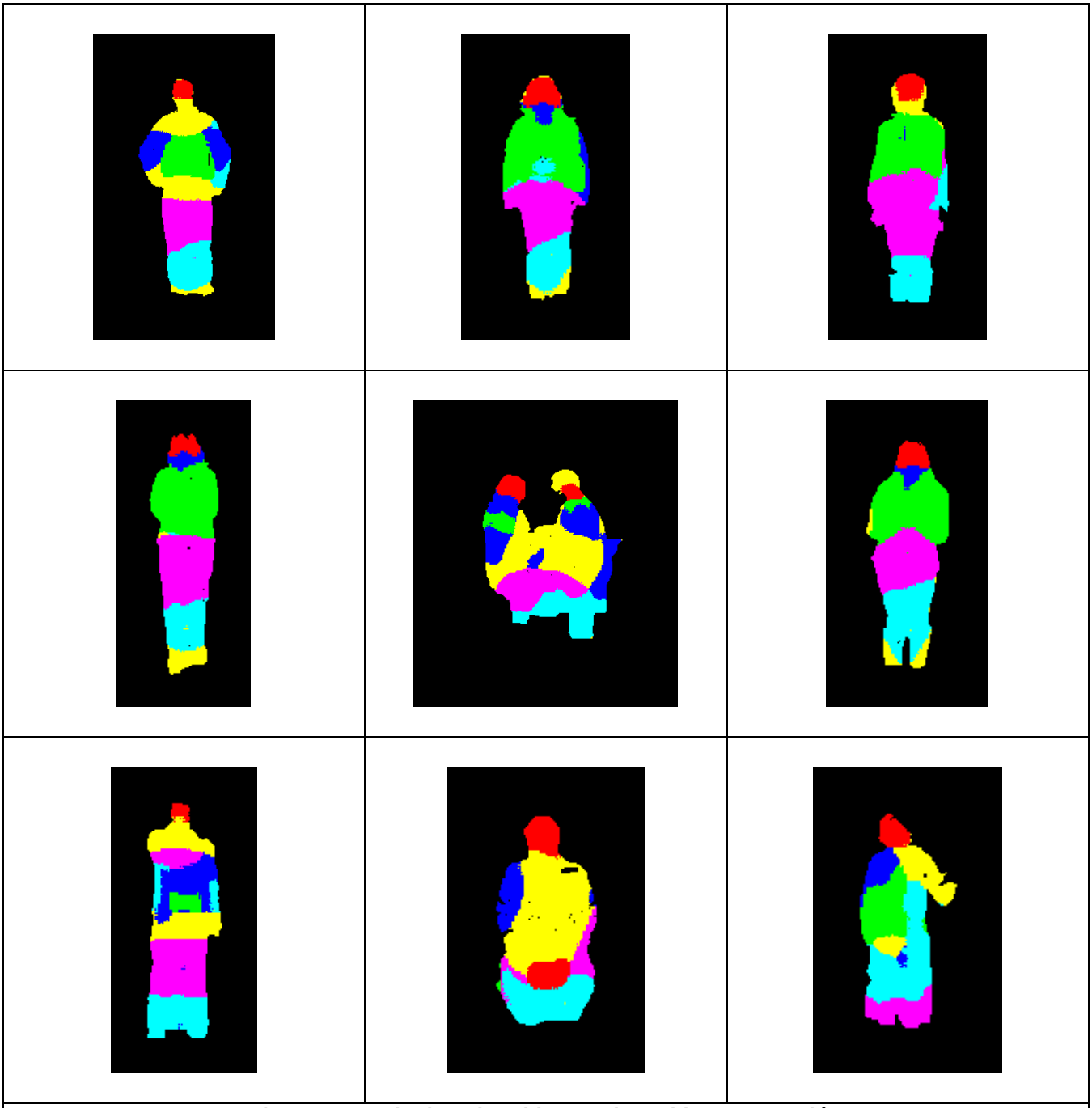


Fig. 39 – Resultados obtenidos por la multi-segmentación

Se puede observar que, tal y como se ha ido comentando, cabeza, muslos y piernas han dado unos resultados mucho más superiores que el resto de partes del cuerpo, que aunque en muchos casos sí tienen una buena situación en la imagen, en general no son tan satisfactorios como las 3 extremidades nombradas.

5. Conclusiones

La Visión por Computador es un campo que se encuentra en constante progreso, lo que conlleva la aparición de nuevas técnicas que mejoran previos resultados publicados e incluso resuelven problemas más complejos.

En nuestro caso en concreto, nos hemos centrado en el problema de la multi-detección y multi-segmentación de partes corporales de personas en secuencias de datos RGB, donde aparecen sujetos realizando diferentes comportamientos en un entorno abierto. Para proponer una solución a dicho problema nos hemos centrado en métodos actuales del estado del arte que han dado resultados satisfactorios en otros problemas de reconocimiento de objetos y multi-clasificación, y que en este proyecto los hemos adaptado al problema de análisis de personas en datos visuales.

A modo de resumen, se concluyen los siguientes logros desarrollados en este proyecto:

-Se ha diseñado una interfaz de etiquetado y se han etiquetado múltiples regiones y gestos en secuencias de datos RGB de múltiples usuarios capturado en un entorno abierto.

-Se ha implementado un aprendizaje de partes corporales mediante descripciones Haar-like y HOG, normalizando regiones para hacerlas invariantes a rotación, y aprendiendo cascadas de clasificadores.

-Mediante los detectores anteriores se han realizado recorridos sobre imágenes de test con el fin de obtener mapas de densidad de probabilidad de las partes detectadas. Estas detecciones se han efectuado mediante la multi-clasificación de los clasificadores en el marco de ensamblado de los Códigos Correctores de Errores.

-Finalmente los mapas de probabilidad también se han usado dentro de un marco de optimización de grafos basado en Graphcuts multi-etiqueta.

-Los resultados obtenidos y validados con el ground truth con medida de overlap muestran el potencial de la propuesta implementada y representan el baseline actual sobre la base de datos desarrollada.

A nivel de análisis crítico y posible trabajo futuro, el gran hándicap con el que nos hemos encontrado ha sido el tiempo. Los tiempos de ejecución para algunos de los procesos eran realmente grandes y por este motivo se ha optado por tomar una serie de medidas que permitieran hacer este proyecto viable en términos de tiempo. Dichas medidas han afectado directamente a los resultados obtenidos puesto que toda reducción de tiempo se ha llevado a cabo a costa de sacrificar ciertos aspectos importantes para la calidad de los resultados:

Reducir el total de ejemplos de entrenamiento: esta medida ha sido especialmente sensible, puesto que esta reducción ha implicado que ciertos clasificadores no hayan aprendido lo suficientemente bien sus modelos.

Acotar el tamaño de ventana de la Sliding Window: este aspecto ha implicado el descarte directo de ciertas extremidades que no han sido analizadas en el proceso, complicando de este modo su detección.

Aumentar el desplazamiento entre ventanas de la Sliding Window: lo cual no ha permitido recorrer exhaustivamente todos los píxeles de la imagen.

Al margen de dichas medidas, un mejor aprendizaje de las extremidades de los actores de la base de datos, teniendo en cuenta las distintas vestimentas, requeriría, también, de una mayor capacidad computacional.

No obstante, los resultados obtenidos son aceptables y la capacidad de mejora existente. Hay mucho margen de maniobra a la hora combinar los distintos métodos que podemos encontrar en la actualidad. Mientras nuevas técnicas y métodos se aproximan por el horizonte, el trabajo por realizar y la mejoría a alcanzar son prácticamente incalculables.

6. Referencias

[1] http://www.maia.ub.es/~sergio/sergio%20escalera%20homepage_006.htm

[2] P. Viola and M. Jones, “Robust real-time face detection”. In Proc. Of IEEE Workshop on Statistical and Computational Thories of Vision, 2001.

[3] OpenCV. http://opencv.willowgarage.com/wiki/

[4] M. Saíz, “Reconstrucción tridimensional mediante visión estéreo y técnicas de optimización”, *UPC, Junio 2010*

[5] T. Collins, “Graph Cut Matching In Computer Vision”, *February 2004*

[6] C. Rother, V. Kolmogorov, A.Blake, “”GrabCut” – Interactive Foreground Extraction using Iterated Graph Cuts”, *Microsoft Reseach Cambridge, UK, 2004*

[7] Daniel Sánchez Abril, “*Códigos correctores de cascadas de clasificadores y graph cuts para segmentación multi-extremidad*”, Universidad de Barcelona, 2012.

[8] O. Veksler, A. Delong, “Multi-label optimization”, *12 Abril 2011,* http://vision.csd.uwo.ca/code/

[9] León Bottou, Chih-Jen Lin *“Support Vector Machine Solvers”* Department of Computer Science Nationak Taiwan University, Taipei, Taiwan

[10] Thomas G. Dietterich, Ghulum Bakiri, “*Solving Multiclass Learning Problems via Error-Correcting Output Codes*”, 1994.

[11] Miguel Ángel Bautista Martín, “Diseño sublineal evolutivo de Códigos Correctores de Errores”, Universidad de Barcelona, 2010

[12] Daniel Sánchez, Juan Carlos Ortega, Miguel Ángel Bautista, Sergio Escalera, “*Human Body Segmentation with Multi-limb Error-Correcting Output Codes Detection and Graph Cuts Optimization*”, Dept. Matemàtica Aplicada i Anàlisi UB, Centre de Visió pe Computador Campus UAB.

[13] Juan Carlos Ortega Pérez “*Multi-clasificación discriminativa por partes mediante Códigos Correctores de Errores*” Universidad de barcelona, 2012.

[14] LibSVM http://www.csie.ntu.edu.tw/~cjlin/libsvm/